Pauser, WalletConnectConfig, LockedTokenStaker & Stakeweight

Reown (fka WalletConnect)

HALBORN

Pauser, WalletConnectConfig, LockedTokenStaker & Stakeweight - Reown (fka WalletConnect)

Prepared by: HALBORN

Last Updated Unknown date

Date of Engagement: November 12th, 2024 - November 15th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
6	0	0	0	1	5

TABLE OF CONTENTS

1. Introduction

The `Wallet Connect Foundation` team engaged `Halborn` to conduct a security assessment on their smart contracts beginning on November 12th, 2024 and ending on November 15th, 2024. The security assessment was scoped to the smart contracts provided to Halborn. Commit hashes and further details can be found in the Scope section of this report.

The `Wallet Connect Foundation` codebase in scope mainly consists of smart contracts that allow for managing pause states of various system functions, configuring the Wallet Connect system, handling staking and updates made to a vote-escrowed token model.

2. Assessment Summary

`Halborn` was provided 4 days for the engagement and assigned 1 full-time security engineer to review the security of the smart contracts in scope. The engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, `Halborn` identified some improvements to reduce the likelihood and impact of risks, which were mostly addressed by the `Wallet Connect Foundation team`. The main identified issues were:

- `Consider implementing a more comprehensive mechanism that includes all system components when pausing and unpausing.`
- `Ensure the documentation clearly outlines the potential risks associated with the forceWithdrawAll() function to maintain transparency with users.`
- `Ensure that the val input address are contract addresses.`

3. Test Approach And Methodology

`Halborn` performed a combination of manual review of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices.

The following phases and associated tools were used throughout the term of the assessment:

- Research into architecture, purpose and use of the platform.
- Smart contract manual code review and walkthrough to identify any logic issue.
- Thorough assessment of safety and usage of critical Solidity variables and functions in scope that could led to arithmetic related vulnerabilities.
- Local testing with custom scripts (`Foundry`).
- Fork testing against main networks (`Foundry`).
- Static analysis of security for scoped contract, and imported functions (`Slither`).

4. Caveats

The current security assessment was limited to the following:

- The `Pauser`, `WalletConnectConfig`, and `LockedTokenStaker` contracts were reviewed in their entirety.
- The `StakeWeight` contract assessment was scoped to changes made between two commits (**https://github.com/WalletConnectFoundation/contracts/compare/74de69f6c77ddf4e291a3f22089dff4043696e1c...f6eb8c0f802b9e6f9952048f7bd44aca855ee935**).

It's important to note that despite these caveats, the security assessment aimed to provide a thorough evaluation of the protocol's security posture. However, the limitations mentioned above should be considered when interpreting the findings and recommendations.

5. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

5.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

5.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (C:N) Low (C:L) Medium (C:M) High (C:H) Critical (C:C)	0 0.25 0.5 0.75 1

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

5.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

6. SCOPE

REPOSITORY

(a) Repository: contracts

(b) Assessed Commit ID: 26afde9

(c) Items in scope:

- src/Pauser.sol
- src/WalletConnectConfig.sol
- src/LockedTokenStaker.sol
- src/StakeWeight.sol
- **WalletConnectFoundation/contracts/compare/74de69f6c77ddf4e291a3f22089dff4043696e1c...f6eb8c0f802b9e6f9952048f7bd44aca855ee935**

Out-of-Scope: Third party dependencies and economic attacks.

Out-of-Scope: New features/implementations after the remediation commit IDs.

7. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
INCOMPLETE PAUSING/UNPAUSING IMPLEMENTATION	LOW	SOLVED - 11/17/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
ADMIN CONTROL OVER USER FUND WITHDRAWAL	INFORMATIONAL	ACKNOWLEDGED - 11/18/2024
NON CONTRACT ADDRESSES CAN BE SET AS CONTRACTS	INFORMATIONAL	ACKNOWLEDGED - 11/18/2024
UNUSED COMPONENTS	INFORMATIONAL	SOLVED - 11/17/2024
REDUNDANT EXPRESSION	INFORMATIONAL	SOLVED - 11/18/2024
MISSING EVENT	INFORMATIONAL	ACKNOWLEDGED - 11/18/2024

8. FINDINGS & TECH DETAILS

8.1 INCOMPLETE PAUSING/UNPAUSING IMPLEMENTATION

// LOW

Description

In the **Pauser** contract, the `pauseAll()` and `unpauseAll()` functions change the status of the `isStakeWeightPaused` and `isSubmitOracleRecordsPaused` variables. This might be misleading, since the `isLockedTokenStakerPaused`, `isNodeRewardManagerPaused`, and `isWalletRewardManagerPaused` variables are not affected by these functions. In emergency situations, this partial pause could leave critical system components still running.

```
136 /// @notice Pauses all actions
137 function pauseAll() external onlyRole(PAUSER_ROLE) {
138     _setIsStakeWeightPaused(true);
139     _setIsSubmitOracleRecordsPaused(true);
140 }
141
142 /// @notice Unpauses all actions
143 function unpauseAll() external onlyRole(UNPAUSER_ROLE) {
144     _setIsStakeWeightPaused(false);
145     _setIsSubmitOracleRecordsPaused(false);
146 }
```

BVSS

AO:S/AC:L/AX:L/R:N/S:U/C:N/A:C/I:N/D:N/Y:N (2.0)

Recommendation

Consider implementing a more comprehensive mechanism that includes all system components when pausing and unpausing. Alternatively, update the name of the function and documentation to clarify the limited scope of the existing implementation.

Remediation Comment

SOLVED: The **Wallet Connect Foundation team** solved this finding in commit `fb9670c` by following the mentioned recommendation.

References

WalletConnectFoundation/contracts/src/Pauser.sol#L136-L146

8.2 ADMIN CONTROL OVER USER FUND WITHDRAWAL

// INFORMATIONAL

Description

The `forceWithdrawAll()` function in the **StakeWeight** contract allows the address with the `DEFAULT_ADMIN_ROLE` role to withdraw all tokens from a user's lock. While it is expected that the `DEFAULT_ADMIN_ROLE` role is held by trusted parties, it is important to consider the potential risks associated with centralized control.

If these privileged role is compromised or act maliciously, they could withdraw user funds unexpectedly.

```
805 function forceWithdrawAll(address to) external onlyRole(DEFAULT_ADMIN_ROLE) {
806     if (to == address(0)) {
807         revert InvalidAddress(to);
808     }
809     StakeWeightStorage storage s = _getStakeWeightStorage();
810
811     if (Pauser(s.config.getPauser()).isStakeWeightPaused()) revert Paused();
812
813     LockedBalance memory lock = s.locks[to];
814
815     uint256 amount = SafeCast.toUint256(lock.amount);
816
817     if (amount == 0) revert NonExistentLock();
818
819     uint256 end = lock.end;
820     uint256 transferredAmount = lock.transferredAmount;
821
822     _unlock(to, lock, amount);
823
824     // transfer remaining back to owner
825     if (transferredAmount > 0) {
826         IERC20(s.config.getL2wct()).safeTransfer(to, transferredAmount);
827     }
828
829     emit ForcedWithdraw(to, amount, transferredAmount, block.timestamp, end);
830 }
```

BVSS

AO:S/AC:L/AX:L/R:P/S:U/C:N/A:C/I:C/D:C/Y:N (1.5)

Recommendation

Ensure the documentation clearly outlines the potential risks associated with the `forceWithdrawAll()` function to maintain transparency with users. Additionally, consider implementing a multi-signature model, where multiple admin addresses must approve any forced withdrawals.

Remediation Comment

ACKNOWLEDGED: The **Wallet Connect Foundation team** made a business decision to acknowledge this finding and not alter the contracts, stating that:
The `forceWithdrawAll` function is a required operational control for managing token lockups in specific administrative scenarios. It's protected by `DEFAULT_ADMIN_ROLE`, can only execute when not paused, and emits clear events for transparency.

References

WalletConnectFoundation/contracts/src/StakeWeight.sol#L805-L830

8.3 NON CONTRACT ADDRESSES CAN BE SET AS CONTRACTS

// INFORMATIONAL

Description

The `_setContract()` function in the `WalletConnectConfig` contract does not verify that the `val` input address is a contract address prior to updating the `_contractsMap` mapping:

```
148 /// @dev Sets a contract address
149 /// @param key The key identifying the contract
150 /// @param val The new contract address
151 function _setContract(bytes32 key, address val) private {
152     if (val == address(0)) {
153         revert InvalidAddress();
154     }
155     if (_contractsMap[key] == val) {
156         revert IdenticalValue();
157     }
158     _contractsMap[key] = val;
159     emit ContractSet({ key: key, val: val });
160 }
```

This could potentially allow non-contract addresses to be set as contract addresses, which may lead to unexpected behavior or vulnerabilities in the system.

BVSS

AO:S/AC:L/AX:L/R:F/S:U/C:N/A:C/I:C/D:N/Y:N (0.6)

Recommendation

Consider adding a check to ensure that the `val` input address is a contract address prior to updating the `_contractsMap` mapping, e.g. by checking if the `val` address contains code.

Remediation Comment

ACKNOWLEDGED: The `Wallet Connect Fondation` team made a business decision to acknowledge this finding and not alter the contracts, stating that:
The lack of code verification is intentional to support Create2 contract deployments, where contract addresses are deterministic and known before deployment. Adding `extcodesize` *checks would break the ability to configure addresses prior to contract deployment. The admin controls on* `_setContract` *already provide sufficient security bounds.*

References

WalletConnectFoundation/contracts/src/WalletConnectConfig.sol#L148-L157

8.4 UNUSED COMPONENTS

// INFORMATIONAL

Description

In the `WalletConnectConfig` contract, the `AccountSet` event and `_accountsMap` mapping are defined but never used.

Similarly, in the `StakeWeight` contract, the following imports are never used:

```
import {Math} from "@openzeppelin/contracts/utils/math/Math.sol";
import {StorageSlot} from "@openzeppelin/contracts/utils/StorageSlot.sol";
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

Consider removing unused components or implementing intended functionality.

Remediation Comment

SOLVED: The `Wallet Connect Foundation` team solved this finding in commit `3c799e5` by following the mentioned recommendation.

References

WalletConnectFoundation/contracts/src/WalletConnectConfig.sol#L25
WalletConnectFoundation/contracts/src/WalletConnectConfig.sol#L34
WalletConnectFoundation/contracts/src/StakeWeight.sol#L6
WalletConnectFoundation/contracts/src/StakeWeight.sol#L12

8.5 REDUNDANT EXPRESSION

// INFORMATIONAL

Description

In the `_createLock()` function of the `StakeWeight` contract, the following expression is present:

```
517 | if (amount <= 0) revert InvalidAmount(amount);
```

However, since the `amount` is a `uint256` type value, it will always be greater than or equal to 0, making this check redundant.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

Update the expression to revert only if the `amount` is equal to 0.

Remediation Comment

SOLVED: The `Wallet Connect Foundation` team solved this finding in commit `2b9ce1d` by following the mentioned recommendation.

References

WalletConnectFoundation/contracts/src/StakeWeight.sol#L517

8.6 MISSING EVENT

// INFORMATIONAL

Description

The `checkpoint()` function in the **StakeWeight** contract does not emit an event. In Solidity development, emitting events is a recommended practice when state-changing functions are invoked. This absence may hamper effective state tracking in off-chain monitoring systems.

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

Consider adding an event to the `checkpoint()` function to provide transparency and allow external systems to track the state changes.

Remediation Comment

ACKNOWLEDGED: The `Wallet Connect Fondation` team made a business decision to acknowledge this finding and not alter the contracts, stating that:
This function is called frequently as an internal function. Adding events would increase gas costs while providing limited value. Adding events only to external calls would create inconsistent tracking since it wouldn't capture all checkpoint updates. State changes can already be monitored through existing events functions.

References

WalletConnectFoundation/contracts/src/StakeWeight.sol#L304

STATIC ANALYSIS REPORT

Description

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their abis and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.
The security team assessed all findings identified by the Slither software, however, findings with related to external dependencies are not included in the below results for the sake of report readability.

Output

The findings obtained as a result of the Slither scan were reviewed, and were not included in the report because they were determined as false positives.


Slither results (1)


Slither results (2)


Slither results (3)
