WalletConnect Foundation - L2WCT Token Upgrade

Reown (fka WalletConnect)

HALBORN

WalletConnect Foundation - L2WCT Token Upgrade - Reown (fka WalletConnect)

Prepared by: HALBORN

Last Updated 03/25/2025

Date of Engagement: March 20th, 2025 - March 24th, 2025

Summary

0% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

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

TABLE OF CONTENTS

1. Introduction
2. Assessment summary
3. Test approach and methodology
4. Caveats
5. Static analysis report
   - 5.1 Description
   - 5.2 Output
6. Risk methodology
7. Scope
8. Assessment summary & findings overview
9. Findings & Tech Details
   - 9.1 Room for clarity in deprecated variables naming

1. Introduction

The `Wallet Connect Foundation` team engaged `Halborn` to conduct a security assessment on their smart contracts beginning on March 20th, 2025 and ending on March 24th, 2025. 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 consists mainly of a token which implements ERC-7802 for interoperability within the OP Stack Superchain ecosystem and adds Wormhole's Non-Transferable Token (NTT) standard for bridging to other ecosystems.

2. Assessment Summary

`Halborn` was provided 3 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.

`Halborn` has conducted a comprehensive security assessment of the provided code, and verified that all components within scope meet industry security standards. The analysis confirms the proper implementation of storage compatibility, bridge isolation mechanisms, permission models, and ERC-7802 compliance. The design effectively prevents interactions with the deprecated Optimism Standard Bridge while maintaining the required functionality for both Superchain and Wormhole bridge operations. No critical security vulnerabilities were identified in the scope of this audit, though there is one area where an improvement could further reduce potential risks:

- `Rename deprecated variables with the _DEPRECATED suffix to make their status immediately visible and update all associated getter functions accordingly to maintain backward compatibility.`

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 focused primarily on the `L2WCT` contract and its dependencies for the upgrade, evaluating the changes introduced in the following pull request:
https://github.com/WalletConnectFoundation/contracts/pull/32/commits/4d34ec3b5b468b6b15a199675e89f0d6ce801166

While the assessment aimed to provide a comprehensive evaluation of the protocol's security posture, it is important to consider the limitations mentioned above when interpreting the findings and recommendations.

5. Static Analysis Report

5.1 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.

5.2 Output

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

```
INFO:Detectors:
L2WCT.REMOTE_TOKEN (src/L2WCT.sol#34) is never initialized. It is used in:
        - L2WCT.l1Token() (src/L2WCT.sol#115-117)
        - L2WCT.remoteToken() (src/L2WCT.sol#127-129)
L2WCT.BRIDGE (src/L2WCT.sol#38) is never initialized. It is used in:
        - L2WCT.l2Bridge() (src/L2WCT.sol#121-123)
        - L2WCT.bridge() (src/L2WCT.sol#133-135)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables
INFO:Detectors:
NttTokenUpgradeable.__NttToken_init(address,string,string).name (src/NttTokenUpgradeable.sol#38) shadows:
        - ERC20Upgradeable.name() (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/ERC20Upgradeable.sol#76-79) (function)
        - IERC20Metadata.name() (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol#15) (function)
NttTokenUpgradeable.__NttToken_init(address,string,string).symbol (src/NttTokenUpgradeable.sol#39) shadows:
        - ERC20Upgradeable.symbol() (lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/ERC20Upgradeable.sol#85-88) (function)
        - IERC20Metadata.symbol() (lib/openzeppelin-contracts/contracts/token/ERC20/extensions/IERC20Metadata.sol#20) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
L2WCT._update(address,address,uint256) (src/legacy/LegacyL2WCT.sol#200-209) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp ≤ transferRestrictionsDisabledAfter (src/legacy/LegacyL2WCT.sol#202)
NttManager.completeInboundQueuedTransfer(bytes32) (src/utils/wormhole/NttManagerFlat.sol#7581-7598) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp - queuedTransfer.txTimestamp < rateLimitDuration (src/utils/wormhole/NttManagerFlat.sol#7589)
NttManager.completeOutboundQueuedTransfer(uint64) (src/utils/wormhole/NttManagerFlat.sol#7601-7632) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp - queuedTransfer.txTimestamp < rateLimitDuration (src/utils/wormhole/NttManagerFlat.sol#7615)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
L2WCT (src/L2WCT.sol#17-261) does not implement functions:
        - INttToken.burn(uint256) (src/interfaces/INttToken.sol#35)
WCT (src/Updated_WCT.sol#15-87) does not implement functions:
        - INttToken.burn(uint256) (src/interfaces/INttToken.sol#35)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions
INFO:Detectors:
L2WCT.BRIDGE (src/L2WCT.sol#38) should be constant
L2WCT.REMOTE_TOKEN (src/L2WCT.sol#34) should be constant
WCT.BRIDGE (src/Updated_WCT.sol#30) should be constant
WCT.REMOTE_TOKEN (src/Updated_WCT.sol#27-28) should be constant
WCT.transferRestrictionsDisabledAfter (src/Updated_WCT.sol#18-19) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Slither:. analyzed (186 contracts with 100 detectors), 14 result(s) found
```

/ DRAFT /

6. 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.

6.1 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
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:

$$E = \prod m_e$$

6.2 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
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:

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

6.3 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

SEVERITY	SCORE VALUE RANGE
Informational	0 - 1.9

7. SCOPE

REPOSITORY

(a) Repository: contracts

(b) Assessed Commit ID: 4d34ec3

(c) Items in scope:

- src/interfaces/IERC7802.sol
- src/interfaces/INttToken.sol
- src/L2WCT.sol
- src/WCT.sol
- src/NttTokenUpgradeable.sol

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

8. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - ROOM FOR CLARITY IN DEPRECATED VARIABLES NAMING	INFORMATIONAL	PENDING

9. FINDINGS & TECH DETAILS

9.1 (HAL-01) ROOM FOR CLARITY IN DEPRECATED VARIABLES NAMING

// INFORMATIONAL

Description

The current implementation marks deprecated storage variables with comments, but their names remain unchanged, which could lead to accidental usage.

For example, in the `L2WCT` contract:

```
/// @notice Address of the corresponding version of this token on the remote chain
/// @custom:deprecated This storage variable is no longer used but preserved for storage layout compatibility
address public REMOTE_TOKEN;

/// @notice Address of the StandardBridge on this network
/// @custom:deprecated This storage variable is no longer used but preserved for storage layout compatibility
address public BRIDGE;
```

Without clear naming indicators, future developers might attempt to build new functionality on top of these deprecated variables.

BVSS

AO:S/AC:L/AX:L/R:F/S:U/C:N/A:N/I:L/D:N/Y:N (0.1)

Recommendation

Rename deprecated variables with a `_DEPRECATED` suffix to make their status immediately visible. Update all associated getter functions accordingly to maintain backward compatibility.

For more reference, see **here** and **here**.

References

WalletConnectFoundation/contracts/evm/src/L2WCT.sol#L42-L48
WalletConnectFoundation/contracts/evm/src/WCT.sol#L20-L34

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.