Ondo - Tokenized Treasury Bill - Additional Aura Module

Noble

HALBORN

Ondo - Tokenized Treasury Bill - Additional Aura Module - Noble

Prepared by: HALBORN

Last Updated 07/16/2024

Date of Engagement by: July 1st, 2024 - July 5th, 2024

Summary

100% 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

The `Noble team` engaged Halborn to conduct a security assessment on their forwarding module updates, beginning on Halborn to conduct a security assessment on the aura module updates, beginning on *07/01/2024* and ending on *07/04/2024*. The security assessment was scoped to the sections of code that pertain to the aura module. Commit hashes and further details can be found in the Scope section of this report.

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 experts with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Ensure that the **Noble Aura Module Updates** operates as intended.

- Identify potential security issues with the **Noble Aura Module Updates** in the Noble Chain.

In summary, Halborn identified some security concerns that were accepted by the `Noble team`.

3. Test Approach And Methodology

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the custom modules. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of structures 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 and purpose.

- Static Analysis of security for scoped repository, and imported functions. (e.g., `staticcheck`, `gosec`, `unconvert`, `codeql`, `ineffassign` and `semgrep`)

- Manual Assessment for discovering security vulnerabilities in the codebase.

- Ensuring the correctness of the codebase.

- Dynamic Analysis of files and modules related to the **Aura Module.**

Out-Of-Scope

- External libraries and financial-related attacks.
- New features/implementations after/with the **remediation/given commit IDs**.

4. 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.

4.1 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:

EXPLOITABILIY 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

EXPLOITABILIY 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:

$$E = \prod m_e$$

4.2 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 (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I: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:

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

4.3 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

5. SCOPE

FILES AND REPOSITORY

(a) Repository: usdy-noble

(b) Assessed Commit ID: 58ecd2a

(c) Items in scope:

- x/aura

Out-of-Scope:

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	0	1

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
IMPLEMENT MULTIPLE CHANNEL SUPPORT FOR ADDBLOCKEDCHANNEL	INFORMATIONAL	ACKNOWLEDGED

7. FINDINGS & TECH DETAILS

7.1 IMPLEMENT MULTIPLE CHANNEL SUPPORT FOR ADDBLOCKEDCHANNEL

// INFORMATIONAL

Description

Currently, the `AddBlockedChannel` function only supports blocking a single channel at a time. To improve efficiency and user experience, we should modify this function to accept and process an array of channels.

Proposed Changes:

1. Update the `MsgAddBlockedChannel` type to include an array of channels instead of a single channel.
2. Modify the `AddBlockedChannel` function to iterate through the array of channels.
3. Implement batch processing of channels, adding each to the blocked list.
4. Update error handling to provide feedback on which channels were successfully blocked and which encountered errors.
5. Modify the emitted event to include all successfully blocked channels.

```
func (k msgServer) AddBlockedChannel(goCtx context.Context, msg
*types.MsgAddBlockedChannel) (*types.MsgAddBlockedChannelResponse, error) {
    ctx := sdk.UnwrapSDKContext(goCtx)

    owner := k.GetOwner(ctx)
    if owner == "" {
        return nil, types.ErrNoOwner
    }
    if msg.Signer != owner {
        return nil, sdkerrors.Wrapf(types.ErrInvalidOwner, "expected %s, got %s",
owner, msg.Signer)
    }

    if k.HasBlockedChannel(ctx, msg.Channel) {
        return nil, fmt.Errorf("%s is already blocked", msg.Channel)
    }

    k.SetBlockedChannel(ctx, msg.Channel)

    return &types.MsgAddBlockedChannelResponse{},
ctx.EventManager().EmitTypedEvent(&types.BlockedChannelAdded{
        Channel: msg.Channel,
    })
}
```

Score

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

Recommendation

Consider adding multi-channel messages on the keeper.

Remediation Plan

ACKNOWLEDGED: The **Noble team** acknowledged the issue.

8. AUTOMATED TESTING

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped component. Among the tools used were **staticcheck**, **gosec**, **semgrep**, **unconvert**, **codeql** and **nancy**. After Halborn verified all the code and scoped structures in the repository and was able to compile them correctly, these tools were leveraged on scoped structures. With these tools, Halborn can statically verify security related issues across the entire codebase.

Staticcheck

No result.

Gosec

No result.

Errcheck

No result.

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.