Authority Module

Noble

HALBORN

Authority Module - Noble

Prepared by: HALBORN

Last Updated 05/17/2024

Date of Engagement by: May 7th, 2024 - May 17th, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
3	0	0	1	0	2

TABLE OF CONTENTS

1. Introduction
2. Assessment summary
3. Test approach and methodology
4. Risk methodology
5. Scope
6. Assessment summary & findings overview
7. Findings & Tech Details
    - 7.1 Single-step authority transfer
    - 7.2 Missing event for updateauthority function
    - 7.3 Missing long descriptions in cli affects usability and user experience
8. Automated Testing

1. Introduction

The `Noble team` engaged Halborn to conduct a security assessment on their authority module, beginning on Halborn to conduct a security assessment on the forwarding module, beginning on *05/07/2024* and ending on *05/07/2024*. The security assessment was scoped to the sections of code that pertain to the forwarding module updates. Commit hashes and further details can be found in the Scope section of this report.

2. Assessment Summary

Halborn was provided 10 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 Authority Module** operates as intended.

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

In summary, Halborn identified some security concerns that were mostly addressed 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 **Authority Module.**

Out-Of-Scope

- External libraries and financial-related attacks.
- New features/implementations after/with the **remediation 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: authority

(b) Assessed Commit ID: dfb231a

(c) Items in scope:

- x/authority

Out-of-Scope:

REMEDIATION COMMIT ID:

- bda8ae2bda8ae2
- ba89b76ba89b76

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	0	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
SINGLE-STEP AUTHORITY TRANSFER	Medium	SOLVED - 05/14/2024
MISSING EVENT FOR UPDATEAUTHORITY FUNCTION	Informational	SOLVED - 05/14/2024

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
MISSING LONG DESCRIPTIONS IN CLI AFFECTS USABILITY AND USER EXPERIENCE	Informational	ACKNOWLEDGED

7. FINDINGS & TECH DETAILS

7.1 SINGLE-STEP AUTHORITY TRANSFER

// MEDIUM

Description

The `UpdateAuthority` function in the provided code allows the current authority to transfer the authority to a new address in a single step. This poses a risk because it enables the current authority to unilaterally change the authority without any checks or safeguards in place.
In the current implementation, the `UpdateAuthority` function performs the following steps:
1. It ensures that the signer of the `MsgUpdateAuthority` message is the current authority.
2. It decodes the new authority address from the message.
3. It checks if the new authority address is the same as the current authority address and returns an error if they are the same.
4. It sets the new authority address using `k.Authority.Set(ctx, newAuthority)`.
The issue with this approach is that it allows the current authority to transfer the authority to any arbitrary address without any additional verification or approval process.

BVSS

AO:A/AC:L/AX:L/C:N/I:H/A:N/D:N/Y:N/R:P/S:C (4.7)

Recommendation

To mitigate the risks associated with single-step authority transfers, it is recommended to implement a multistep process with additional safeguards.

Remediation Plan

SOLVED: The **Noble team** solved the issue by implementing a two-step authority transfer.

Remediation Hash

https://github.com/noble-assets/authority/commit/bda8ae2a18b3da36290147b6a6671ee300f8c85c

References

noble-assets/authority/x/authority/keeper/msg_server.go

7.2 MISSING EVENT FOR UPDATEAUTHORITY FUNCTION

// INFORMATIONAL

Description

In the provided code, the `UpdateAuthority` function is responsible for updating the authority address. However, it lacks the necessary event emission to notify and record the authority update event.
The `UpdateAuthority` function performs the following steps:
1. It ensures that the signer of the `MsgUpdateAuthority` message is the current authority.
2. It decodes the new authority address from the message.
3. It checks if the new authority address is the same as the current authority address and returns an error if they are the same.
4. It sets the new authority address using `k.Authority.Set(ctx, newAuthority)`.
However, after successfully updating the authority, the function does not emit any events to indicate that the authority has been changed.

```
func (k msgServer) UpdateAuthority(ctx context.Context, msg
*types.MsgUpdateAuthority) (*types.MsgUpdateAuthorityResponse, error) {
    authority, err := k.EnsureAuthoritySigner(ctx, msg.Signer)
    if err != nil {
        return nil, err
    }

    newAuthority, err :=
k.accountKeeper.AddressCodec().StringToBytes(msg.NewAuthority)
    if err != nil {
        return nil, errors.Wrap(err, "failed to decode new authority address")
    }
    if bytes.Equal(newAuthority, authority) {
        return nil, types.ErrSameAuthority
    }

    err = k.Authority.Set(ctx, newAuthority)
    return &types.MsgUpdateAuthorityResponse{}, err
}
```

Score

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

Recommendation

To address the missing events issue for the `UpdateAuthority` function, it is recommended to add event emission logic.

Remediation Plan

SOLVED: The **Noble team** solved the issue by adding events to the function.

Remediation Hash

https://github.com/noble-assets/authority/commit/ba89b7610f020b6323e1487af8acdf49f0b6f38c

7.3 MISSING LONG DESCRIPTIONS IN CLI AFFECTS USABILITY AND USER EXPERIENCE

// INFORMATIONAL

Description

The Command Line Interface (CLI) for the module is currently lacking long descriptions, which are available in the Cosmos SDK. Long descriptions provide users with detailed information about the purpose and usage of CLI commands. The absence of these descriptions reduces the overall usability and user experience of the CLI, as users may face difficulty in understanding the functionality and proper usage of various commands.

```
func NewCmdExecute() *cobra.Command {
    cmd := &cobra.Command{
        Use:   "execute [path to underlying tx file]",
        Short: "Execute arbitrary messages as authority module",
        Args:  cobra.ExactArgs(1),
        RunE: func(cmd *cobra.Command, args []string) error {
            clientCtx, err := client.GetClientTxContext(cmd)
            if err != nil {
                return err
            }

            underlying, err := authclient.ReadTxFromFile(clientCtx, args[0])
            if err != nil {
                return err
            }

            msg := types.NewMsgExecute(clientCtx.FromAddress.String(),
underlying.GetMsgs())

            return tx.GenerateOrBroadcastTxCLI(clientCtx, cmd.Flags(), msg)
        },
    }

    flags.AddTxFlagsToCmd(cmd)

    return cmd
}
```

Score

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

Recommendation

Update the CLI to include long descriptions for all commands, following the guidelines and format provided by the Cosmos SDK. Ensure that these descriptions are comprehensive and accurately convey the purpose and usage of each command.

Remediation Plan

SOLVED: The **Noble team** acknowledged this finding.

References

noble-assets/authority/x/authority/client/cli/tx.go#L28

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

```
[authority-
dfb231af25a2d19b80f1ca00cc6b4be242aa8fe1/x/authority/keeper/msg_server.go:68] - G601
(CWE-118): Implicit memory aliasing in for loop. (Confidence: MEDIUM, Severity:
MEDIUM)
    67:     for _, event := range events {
  > 68:         _ = k.eventService.EventManager(ctx).Emit(ctx, &event)
    69:     }
```

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.