Forwarding Module Eden Upgrade

Noble

HALBORN

Forwarding Module Eden Upgrade - Noble

Prepared by: HALBORN

Last Updated 05/02/2024

Date of Engagement by: April 29th, 2024 - May 1st, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

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

TABLE OF CONTENTS

1. Introduction

`The Noble team` engaged Halborn to conduct a security assessment on their module update, beginning on Halborn to conduct a security assessment on the forwarding module, beginning on *04/29/2024* and ending on *05/01/2024*. 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 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 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 Forwarding Module updates** operate as intended.

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

In summary, Halborn identified some security concerns that were considered as not applicable 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 **Forwarding 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: forwarding

(b) Assessed Commit ID: 0884f39

(c) Items in scope:

- x/forwarding

Out-of-Scope: Other modules., Changes out-of-PR.

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	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-02 - TOTAL FORWARDS NOT MIGRATED IN 0.50 UPGRADE	Informational	NOT APPLICABLE
HAL-01 - MISSING PACKAGE TYPES/MESSAGES IN THE COSMOS MODULE	Informational	NOT APPLICABLE

7. FINDINGS & TECH DETAILS

7.1 (HAL-02) TOTAL FORWARDS NOT MIGRATED IN 0.50 UPGRADE

// INFORMATIONAL

Description

During the 0.50 upgrade of the Cosmos module, it was discovered that the `TotalForwarded` data is not being migrated properly. The `Migrate1to2` function in the `keeper` package, responsible for migrating data from version 1 to version 2, does not include the migration of the `TotalForwarded` data.
he module's state will be inconsistent after the upgrade, as the `TotalForwarded` data will not reflect the accurate total forwarded amounts for each channel.

```
package keeper

import (
    "github.com/cosmos/cosmos-sdk/runtime"
    sdk "github.com/cosmos/cosmos-sdk/types"
    v1 "github.com/noble-assets/forwarding/v2/x/forwarding/migrations/v1"
)

// Migrator is a struct for handling in-place store migrations.
type Migrator struct {
    keeper *Keeper
}

// NewMigrator returns a new Migrator.
func NewMigrator(keeper *Keeper) Migrator {
    return Migrator{keeper: keeper}
}

// Migrate1to2 migrates from version 1 to 2.
func (m Migrator) Migrate1to2(ctx sdk.Context) error {
    adapter := runtime.KVStoreAdapter(m.keeper.storeService.OpenKVStore(ctx))

    for channel, count := range v1.GetAllNumOfAccounts(adapter) {
        err := m.keeper.NumOfAccounts.Set(ctx, channel, count)
        if err != nil {
            return err
        }
    }

    for channel, count := range v1.GetAllNumOfForwards(adapter) {
        err := m.keeper.NumOfForwards.Set(ctx, channel, count)
```

```
        if err != nil {
            return err
        }
    }

    return nil
}
```

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 this issue, it is recommended to update the `Migrate1to2` function in the `keeper` package to include the migration of the `TotalForwarded` data. The following steps should be taken:
1. Update the `Migrate1to2` function:

- Add a new loop that iterates over the `TotalForwarded` data from version `1`.
- Retrieve the total forwarded amount for each channel using a new function `v1.GetAllTotalForwarded`.
- Set the migrated `TotalForwarded` data in the new version using `m.keeper.TotalForwarded.Set`.

Remediation Plan

NOT APPLICABLE : The **Noble team** has confirmed that the missing migration of `TotalForwarded` is intentional. In version 0.50, `collections` are designed to be backwards compatible with legacy state by default. The migrations were added specifically to address the state incompatibility between the encoding of `uint64` in the legacy state and the encoding used by `collections`. As `TotalForwarded` does not involve `uint64` state, it does not require migration. The **Noble team** has thoroughly reviewed the implementation and determined that no further action is necessary regarding this matter.

7.2 (HAL-01) MISSING PACKAGE TYPES/MESSAGES IN THE COSMOS MODULE

// INFORMATIONAL

Description

The package "types/messages" has been deleted from the Cosmos module, which contains important type definitions and validation functions. This deletion affects the functionality of the module, particularly the `ValidateBasic` function.
The `ValidateBasic` function is responsible for validating the basic integrity of the `MsgRegisterAccount` and `MsgClearAccount` messages. It checks if the `Signer` address is valid and if the `Channel` ID is valid for the `MsgRegisterAccount` message. For the `MsgClearAccount` message, it validates both the `Signer` and `Address` fields.

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 resolve this issue, it is recommended to restore the deleted package "types/messages" in the Cosmos module. The package should include the following:
1. The necessary type definitions for `MsgRegisterAccount` and `MsgClearAccount` messages.
2. The `ValidateBasic` function implementations for both messages.
3. Any other required utility functions or constants used in the validation process.

Remediation Plan

NOT APPLICABLE : The **Noble team** has confirmed that the `ValidateBasic` method, which was used to validate signers and channel IDs, has been deprecated in version 0.47 and will be removed in version 0.51. In version 0.50, signers are automatically verified using Protobuf annotations, eliminating the need for manual validation in the `ValidateBasic` method. The provided code snippets demonstrate how the channel ID and address checks are replaced by the Protobuf annotations. The **Noble team** has thoroughly reviewed the implementation and determined that no further action is necessary regarding this matter, as the validation is now handled automatically through the Protobuf annotations.

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

```
types/query.pb.gw.go:16:2: "github.com/golang/protobuf/descriptor" is deprecated: See the "google.golang.org/protobuf/reflect/protoreflect" package for how to obtain an EnumDescriptor or MessageDescriptor in order to programatically interact with the protobuf type system.  (SA1019)
types/query.pb.gw.go:17:2: "github.com/golang/protobuf/proto" is deprecated: Use the "google.golang.org/protobuf/proto" package instead.  (SA1019)
types/query.pb.gw.go:33:9: descriptor.ForMessage is deprecated: Not all concrete message types satisfy the Message interface. Use MessageDescriptorProto instead. If possible, the calling code should be rewritten to use protobuf reflection instead. See package "google.golang.org/protobuf/reflect/protoreflect" for details.  (SA1019)
```

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.