// Private Cosmos Security Assessment 06.03.2024 - 06.21.2024

Ondo - Tokenized Treasury Bill

Noble

HALBORN

Ondo - Tokenized Treasury Bill - Noble

Prepared by: HALBORN

Last Updated 07/16/2024

Date of Engagement by: June 3rd, 2024 - June 21st, 2024

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
10	0	0	3	0	7

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 forwarding module, beginning on *06/03/2024* and ending on *06/22/2024*. 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 3 weeks 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 Modules** operates as intended.

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

In summary, Halborn identified some security concerns that were accepted and 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 **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: aura

(b) Assessed Commit ID: f51c30a

(c) Items in scope:

- x/aura

Out-of-Scope:

FILES AND REPOSITORY

(a) Repository: aura

(b) Assessed Commit ID: f553945

(c) Items in scope:

Out-of-Scope:

REMEDIATION COMMIT ID:

- 44c811a44c811a

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	3	0	7

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
NO VALIDATION CHECK FOR NEW OWNER IN TRANSFEROWNERSHIP	MEDIUM	SOLVED - 06/06/2024
LACK OF BLACKLIST CHECK WHEN ADDING BURNERS, PAUSERS, AND OWNERS	MEDIUM	RISK ACCEPTED
PAUSE FUNCTIONALITY DOES NOT AFFECT MINT AND BURN OPERATIONS	MEDIUM	SOLVED - 06/16/2024
DUPLICATE TRANSACTIONS IN THE MEMPOOL'S DATA STRUCTURES	INFORMATIONAL	ACKNOWLEDGED
ASA-2023-002: DEFAULT BLOCKPARAMS.MAXBYTES CONFIGURATION MAY INCREASE BLOCK TIMES AND AFFECT CONSENSUS PARTICIPATION IN THE NOBLE APP CHAIN	INFORMATIONAL	ACKNOWLEDGED
ASA-2024-004: DEFAULT EVIDENCE CONFIGURATION PARAMETERS MAY LIMIT WINDOW OF VALIDITY FOR THE NOBLE APP CHAIN	INFORMATIONAL	ACKNOWLEDGED
POTENTIAL INCONSISTENCY IN BLOCKED ADDRESS HANDLING FOR USDY TOKEN TRANSFERS	INFORMATIONAL	ACKNOWLEDGED
LACK OF SIMULATION AND FUZZING OF THE MODULE INVARIANT	INFORMATIONAL	ACKNOWLEDGED
MISSING LONG DESCRIPTIONS IN CLI AFFECTS USABILITY AND USER EXPERIENCE	INFORMATIONAL	ACKNOWLEDGED
BLOCKLIST OWNER CAN ADD ITSELF TO THE BLOCKLIST	INFORMATIONAL	ACKNOWLEDGED

7. FINDINGS & TECH DETAILS

7.1 NO VALIDATION CHECK FOR NEW OWNER IN TRANSFEROWNERSHIP

// MEDIUM

Description

In the `TransferOwnership` function of the `blocklistMsgServer`, there is a missing validation check to ensure that the new owner is different from the previous owner. Currently, the function allows transferring ownership to the same address as the current owner.

The specific issue lies in the following code snippet:

```
func (k blocklistMsgServer) TransferOwnership(goCtx context.Context, msg
*blocklist.MsgTransferOwnership) (*blocklist.MsgTransferOwnershipResponse, error) {
	ctx := sdk.UnwrapSDKContext(goCtx)
	owner := k.GetBlocklistOwner(ctx)
	if owner == "" {
		return nil, blocklist.ErrNoOwner
	}
	if msg.Signer != owner {
		return nil, errors.Wrapf(blocklist.ErrInvalidOwner, "expected %s, got %s",
owner, msg.Signer)
	}
	k.SetBlocklistPendingOwner(ctx, msg.NewOwner)
	return &blocklist.MsgTransferOwnershipResponse{},
ctx.EventManager().EmitTypedEvent(&blocklist.OwnershipTransferStarted{
		PreviousOwner: owner,
		NewOwner:      msg.NewOwner,
	})
}
```

BVSS

AO:A/AC:L/AX:L/C:N/I:C/A:N/D:N/Y:N/R:P/S:C (6.3)

Recommendation

To address this issue, it is recommended to add a validation check in the `TransferOwnership` function to ensure that the new owner is different from the previous owner.

Remediation Plan

SOLVED: The **Noble team** solved the issue by new owner check.

Remediation Hash

https://github.com/noble-assets/aura/commit/44c811a1f60e7d5975cf27ae5d08aa1d72cf33b9

References

noble-assets/aura/x/aura/keeper/msg_server_blocklist.go#L32

7.2 LACK OF BLACKLIST CHECK WHEN ADDING BURNERS, PAUSERS, AND OWNERS

// MEDIUM

Description

The provided code for the `msgServer` in the `aura` module allows adding burners, pausers, and owners without checking if the addresses are in the blacklist. This can lead to a situation where a blacklisted address can be assigned roles with special privileges, such as burning tokens, pausing the module, or transferring ownership.
The specific issues are:
1. In the `AddBurner` function, the code checks if the signer is the current owner and if the burner address is not already a burner. However, it does not check if the burner address is in the blacklist before adding it as a burner.
2. Similarly, in the `AddPauser` function, the code checks if the signer is the current owner and if the pauser address is not already a pauser. However, it does not verify if the pauser address is in the blacklist before assigning the pauser role.
3. In the `TransferOwnership` function, the code allows the current owner to transfer ownership to a new address without checking if the new owner address is in the blacklist.

BVSS

AO:A/AC:L/AX:L/C:N/I:C/A:N/D:N/Y:N/R:P/S:C (6.3)

Recommendation

Consider adding a blocklist check on the privileged account additions.

Remediation Plan

RISK ACCEPTED: The **Noble team** accepted the risk of the issue.

7.3 PAUSE FUNCTIONALITY DOES NOT AFFECT MINT AND BURN OPERATIONS

// MEDIUM

Description

The current implementation of the pause functionality in the module does not prevent the execution of mint and burn operations. This poses a potential issue, as the purpose of pausing the module is to temporarily halt all critical operations.
When the module is paused using the MsgPause message, it is expected that all major operations, including minting and burning, should be suspended until the module is unpaused. However, upon reviewing the code, it is evident that the Mint and Burn functions do not check the paused state of the module before executing their respective operations.
This oversight allows users with the appropriate permissions (minters and burners) to continue minting and burning tokens even when the module is meant to be paused. This undermines the effectiveness and purpose of the pause functionality and may lead to unexpected behavior or potential vulnerabilities.

```
func (k msgServer) Mint(goCtx context.Context, msg *types.MsgMint)
(*types.MsgMintResponse, error) {
    ctx := sdk.UnwrapSDKContext(goCtx)

    if !k.HasMinter(ctx, msg.Signer) {
        return nil, types.ErrInvalidMinter
    }
    allowance := k.GetMinter(ctx, msg.Signer)
    if allowance.LT(msg.Amount) {
        return nil, sdkerrors.Wrapf(types.ErrInsufficientAllowance, "minter %s has an
allowance of %s", msg.Signer, allowance.String())
    }

    to, err := sdk.AccAddressFromBech32(msg.To)
    if err != nil {
        return nil, sdkerrors.Wrapf(err, "unable to decode account address %s",
msg.To)
    }

    if !msg.Amount.IsPositive() {
        return nil, errors.New("amount must be positive")
    }

    coins := sdk.NewCoins(sdk.NewCoin(k.Denom, msg.Amount))
    err = k.bankKeeper.MintCoins(ctx, types.ModuleName, coins)
    if err != nil {
        return nil, sdkerrors.Wrapf(err, "unable to mint to module")
    }
    err = k.bankKeeper.SendCoinsFromModuleToAccount(ctx, types.ModuleName, to, coins)
    if err != nil {
        return nil, sdkerrors.Wrapf(err, "unable to transfer from module to user")
    }

    k.SetMinter(ctx, msg.Signer, allowance.Sub(msg.Amount))

    // NOTE: The bank module emits an event for us.
    return &types.MsgMintResponse{}, nil
}
```

BVSS

AO:A/AC:L/AX:L/C:N/I:C/A:N/D:N/Y:N/R:P/S:C (6.3)

Recommendation

To address this issue and ensure the integrity of the pause functionality, the following recommendations are proposed:
1. Modify the `Mint` and `Burn` functions to check the paused state of the module before executing their operations. If the module is paused, these functions should return an appropriate error indicating that the operation is not allowed during the paused state.

Remediation Plan

SOLVED: The **Noble team** has already solved the issue with the adding restricting on the keeper.

7.4 DUPLICATE TRANSACTIONS IN THE MEMPOOL'S DATA STRUCTURES

// INFORMATIONAL

Description

A synchronization issue has been identified in the mempool component of the Noble App Chain, which utilizes CometBFT as its consensus engine. The mempool maintains a list and a map to keep track of outstanding transactions. These two data structures are expected to be in sync at all times, with the map tracking the index of each transaction in the list. However, it has been discovered that under certain circumstances, these data structures can become out of sync, leading to the presence of duplicate transactions in the list.
When this issue occurs, the list may contain multiple copies of the same transaction, while the map only tracks a single index. As a result, it becomes impossible to remove all copies of the duplicated transaction from the list, even if the transaction is later committed in a block. The only way to remove the stuck transaction is by restarting the node.

Affected Component:
CometBFT
Affected Versions:
The specific version of CometBFT used by the Noble App Chain, as mentioned in the `go.mod` file:
`
github.com/tendermint/tendermint => github.com/cometbft/cometbft v0.34.27
`

Score

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

Recommendation

To mitigate the impact of this issue, the following workarounds and fixes are recommended:
1. Upgrade CometBFT to the patched versions:

- For the v0.34.x series, upgrade to v0.34.29 or later.
- For the v0.37.x series, upgrade to v0.37.2 or later.

Remediation Plan

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

References

cometbft/cometbft

7.5 ASA-2023-002: DEFAULT BLOCKPARAMS.MAXBYTES CONFIGURATION MAY INCREASE BLOCK TIMES AND AFFECT CONSENSUS PARTICIPATION IN THE NOBLE APP CHAIN

// INFORMATIONAL

Description

The Noble App Chain, which utilizes CometBFT as its consensus engine, has been identified to have a default configuration for `BlockParams.MaxBytes` that may be too large for its specific use case. This default value can potentially increase block times and affect consensus participation when fully utilized by chain participants. It is recommended that the Noble App Chain team consider their specific needs and evaluate the impact of having proposed blocks with the maximum allowed block size, especially on bandwidth usage and block latency.

Score

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

Recommendation

To mitigate the potential impact of this default configuration, it is recommended that the Noble App Chain team take the following step:

1. Evaluate the specific needs and requirements of the Noble App Chain use case and determine an appropriate value for `BlockParams.MaxBytes`.

Remediation Plan

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

References

cometbft/cometbft

7.6 ASA-2024-004: DEFAULT EVIDENCE CONFIGURATION PARAMETERS MAY LIMIT WINDOW OF VALIDITY FOR THE NOBLE APP CHAIN

// INFORMATIONAL

Description

The Noble App Chain, which utilizes a vulnerable version of CometBFT as its consensus engine, has been identified to have a default configuration issue. The default values for the `EvidenceParams.MaxAgeNumBlocks` and `EvidenceParams.MaxAgeDuration` consensus parameters may not provide sufficient coverage for the entire unbonding period of the chain. This could potentially lead to the premature expiration of evidence and allow for unpunished Byzantine behavior if evidence is discovered outside the defined window.

Affected Component:

CometBFT

Affected Versions:

The specific version of CometBFT used by the Noble App Chain, as mentioned in the `go.mod` file:

`

github.com/tendermint/tendermint => github.com/cometbft/cometbft v0.34.27

`

Score

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

Recommendation

It is recommended that chain ecosystems and their maintainers set the consensus parameters `EvidenceParams.MaxAgeNumBlocks` and `EvidenceParams.MaxAgeDuration`.

Remediation Plan

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

References

cometbft/cometbft

7.7 POTENTIAL INCONSISTENCY IN BLOCKED ADDRESS HANDLING FOR USDY TOKEN TRANSFERS

// INFORMATIONAL

Description

The current implementation of the SendCoinsFromModuleToAccount function in the Cosmos SDK and the SendRestrictionFn function in the USDY token module may lead to inconsistent behavior when handling blocked addresses during USDY token transfers.

The SendCoinsFromModuleToAccount function includes a check using the BlockedAddr function to prevent sending coins to blocked addresses. If the recipient address is blocked, an error is returned, and the transfer is not allowed to proceed.

On the other hand, the SendRestrictionFn function performs additional checks specifically for USDY token transfers. It checks if the sender address is blocked using the HasBlockedAddress function only if the transfer is not a minting operation (i.e., not from the module address to a non-module address). Additionally, it checks if the recipient address is blocked using the same HasBlockedAddress function.

The inconsistency arises from the fact that the SendCoinsFromModuleToAccount function uses the BlockedAddr function to check for blocked addresses, while the SendRestrictionFn function uses the HasBlockedAddress function. If these two functions have different implementations or BlockedAddress is available on base coin transfers but not USDY, it could lead to inconsistent behavior when handling blocked addresses.Consequently, a blocked address that is allowed to receive funds according to the SendCoinsFromModuleToAccount function might still be blocked by the SendRestrictionFn function, leading to unexpected transfer failures or inconsistent application of blocked address rules.

```go
// SendCoinsFromModuleToAccount transfers coins from a ModuleAccount to an
AccAddress.
// An error is returned if the module account does not exist or if
// the recipient address is black-listed or if sending the tokens fails.
func (k BaseKeeper) SendCoinsFromModuleToAccount(
    ctx context.Context, senderModule string, recipientAddr sdk.AccAddress, amt
sdk.Coins,
) error {
    senderAddr := k.ak.GetModuleAddress(senderModule)
    if senderAddr == nil {
        return errorsmod.Wrapf(sdkerrors.ErrUnknownAddress, "module account %s does
not exist", senderModule)
    }

    if k.BlockedAddr(recipientAddr) {
        return errorsmod.Wrapf(sdkerrors.ErrUnauthorized, "%s is not allowed to
receive funds", recipientAddr)
    }

    return k.SendCoins(ctx, senderAddr, recipientAddr, amt)
}
```

Score

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

Recommendation

Review the implementation of the `BlockedAddr` function **used in the** `SendCoinsFromModuleToAccount` **function and the** `HasBlockedAddress` **function used in the** `SendRestrictionFn` **function. Ensure that both functions have consistent behavior and use the same underlying blocked address list or mechanism.**

Remediation Plan

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

References

cosmos/cosmos-sdk/x/bank/keeper/keeper.go#L272

7.8 LACK OF SIMULATION AND FUZZING OF THE MODULE INVARIANT

// INFORMATIONAL

Description

The *Noble AppChain* system lacks comprehensive CosmosSDK simulations and invariants for its *Aura* module. More complete use of the simulation feature would make it easier to fuzz test the entire blockchain and help ensure that invariants hold.

Score

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

Recommendation

Eventually, extend the simulation module to cover all operations that can occur in a real Noble Chain deployment, along with all possible error states, and run it many times before each release.

Make sure of the following:- All module operations are included in the simulation module.

- The simulation uses some accounts (e.g., between 5 and 20) to increase the likelihood of an interesting state change.
- The simulation uses the currencies/tokens that will be used in the production network.
- The simulation continues to run when a transaction fails.
- All paths of the transaction code are executed. (Enable code coverage to see how often individual lines are executed.)

Remediation Plan

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

7.9 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.

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 issue of missing long descriptions in the CLI, it is recommended to:

- 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

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

References

- https://github.com/noble-assets/aura/blob/main/x/aura/client/cli/tx_blocklist.go#L31
- https://github.com/noble-assets/aura/blob/main/x/aura/client/cli/tx_blocklist.go#L81
- https://github.com/noble-assets/aura/blob/main/x/aura/client/cli/tx_blocklist.go#L106

7.10 BLOCKLIST OWNER CAN ADD ITSELF TO THE BLOCKLIST

// INFORMATIONAL

Description

In the current implementation of the `AddToBlocklist` function in the `blocklistMsgServer`, there is no check to prevent the blocklist owner from adding itself to the blocklist. This means that the owner can inadvertently or intentionally block their own address, which could lead to unintended consequences and potentially lock them out of performing further blocklist operations.

```
func (k blocklistMsgServer) AddToBlocklist(goCtx context.Context, msg
*blocklist.MsgAddToBlocklist) (*blocklist.MsgAddToBlocklistResponse, error) {
    ctx := sdk.UnwrapSDKContext(goCtx)

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

    for _, account := range msg.Accounts {
        address, err := sdk.AccAddressFromBech32(account)
        if err != nil {
            return nil, errors.Wrapf(err, "unable to decode account address %s",
account)
        }

        k.SetBlockedAddress(ctx, address)
    }

    return &blocklist.MsgAddToBlocklistResponse{},
ctx.EventManager().EmitTypedEvent(&blocklist.BlockedAddressesAdded{
        Accounts: msg.Accounts,
    })
}
```

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 prevent the blocklist owner from accidentally or intentionally adding itself to the blocklist, it is recommended to add a validation check in the `AddToBlocklist` function.

Remediation Plan

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

References

noble-assets/aura/x/aura/keeper/msg_server_blocklist.go#L61

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.
