



Security Audit Report

Neutron 2024 Q1: Neutron Chain Manager

Authors: Ivan Gavran, Aleksandar Ignjatijevic

Last revised 4 April, 2024

Table of Contents

Audit Overview 1

Scope 1

Conclusion 1

Audit Dashboard 2

Target Summary 2

Engagement Summary 2

Severity Summary 2

Findings 3

An address with permissions for one kind of proposal message, can execute any proposal message (for which it has no permissions) 4

An admin proposal could circumvent the timelock mechanism 7

There is no guarantee that the mainDAO remains among managers 8

Handling of conflicting permissions in AllowOnly strategies 9

Appendix: Vulnerability Classification 11

Impact Score 11

Exploitability Score 11

Severity Score 12

Disclaimer 14

Audit Overview

Scope

In March 2024, [Informal Systems](#) conducted a security audit for Neutron. The audit focused on the new feature for expediting chain administration - `neutron_chain_manager` contract.

The audit was performed from March 21, 2024 to March 28, 2024 by the following personnel:

- Ivan Gavran
- Aleksandar Ignjatijevic

Relevant Code Commits

The scope of the audit was the `neutron-dao` repository at commit [c6411a8](#).

Conclusion

We reviewed the contract's code and found it to be well implemented. We found one implementation issue allowing for unprivileged admin actions, and three more findings of lower severity.

The dev team promptly addressed the unprivileged action issue and we reviewed the fix. The rest of the issues will be addressed at a later point, because careful governance process can avoid them completely.

Audit Dashboard

Target Summary

- **Type:** Protocol and Implementation
- **Platform:** CosmWasm

Engagement Summary

- **Dates:** 21.3.2024. - 28.3.2024.
- **Method:** Manual code review

Severity Summary

Finding Severity	#
Critical	0
High	1
Medium	0
Low	2
Informational	1
Total	4

Findings

Title	Type	Severity	Status
An address with permissions for one kind of proposal message, can execute any proposal message (for which it has no permissions)	Implementation	3 High	Resolved
An admin proposal could circumvent the timelock mechanism	Implementation	1 Low	Acknowledged
There is no guarantee that the mainDAO remains among managers	Protocol	0 Informational	Acknowledged
Handling of conflicting permissions in AllowOnly strategies	Implementation	1 Low	Acknowledged

An address with permissions for one kind of proposal message, can execute any proposal message (for which it has no permissions)

Type	Implementation
Severity	3 High
Impact	3 High
Exploitability	2 Medium
Status	Resolved

Involved artifacts

- [neutron-dao/dao/neutron-chain-manager/src/contract.rs](#)

Description

In `neutron-chain-manager`, when handling `ExecuteMessages`, there is a sequence of checks to see if the messages could be executed.

1. The sender **must have** a strategy associated with them.
2. For `AllowOnly`, `check_allow_only_permissions` function iterates over all messages and makes sure that `check_neutron_msg` function does not return an error.
3. The following steps **branch** depending on the type of the `NeutronMsg` received inside `check_neutron_msg`. For `NeutronMsg::SubmitAdminProposal`, the `check_submit_admin_proposal_message` function **is called**.
4. In that function, there is another branch on the type of `AdminProposal` wrapped in `NeutronMsg::SubmitAdminProposal`. For `ProposalExecuteMessage`, the `check_proposal_execute_message` **is called**.
5. This function checks if the proposal (as deserialized from the JSON string) is of the type `MSG_TYPE_UPDATE_PARAMS_CRON`. If it is, it **runs an additional check**. But if it is not, it simply returns `Ok`.

Problem Scenarios

Assume an address that is authorized to update cron parameters. That address can then freely send proposals to update other parameters as well, eg `feeburner` parameters.

The problem is illustrated with the following test:

```
pub fn test_execute_execute_message_update_params_feeburner() {
    let msg = CosmosMsg::Custom(NeutronMsg::SubmitAdminProposal {
```

```

        admin_proposal: AdminProposal::ProposalExecuteMessage(ProposalExecuteMessage
{
    message: r#"{"@type":"/neutron.feeburner.MsgUpdateParams",
    "authority":"neutron1hxsksfdxpp5hqgtjj6am6nkjefhfzj359x0ar3z",
    "params":{"security_address": "addr1", "limit": 16}}"#
        .to_string(),
    },
});

let mut deps = mock_dependencies();
let env = mock_env();
let info = mock_info("neutron_dao_address", &[]);

instantiate(
    deps.as_mut(),
    env.clone(),
    info.clone(),
    InstantiateMsg {
        initial_strategy_address: Addr::unchecked("neutron_dao_address".to_string
()),
        initial_strategy: Strategy::AllowAll,
    },
)
.unwrap();

let info = mock_info("neutron_dao_address", &[]);
execute_add_strategy(
    deps.as_mut(),
    info.clone(),
    Addr::unchecked("addr1".to_string()),
    Strategy::AllowOnly(vec![UpdateParamsPermission(
        CronUpdateParamsPermissionEnumField(CronUpdateParamsPermission {
            security_address: true,
            limit: true,
        })
    )]),
)
.unwrap();

let info = mock_info("addr1", &[]);
// This should fail because the message is not for the cron module (which is the
only authorization "addr1" has)
let res = execute_execute_messages(deps.as_mut(), info.clone(), vec![msg]);

assert!(res.is_err());
}

```

Note: We deem this problem to be of high impact, but only of moderate exploitability. This is due to existing checks: first, only a small number of addresses will every be added with a strategy. Second, for every subDAOs proposal, there will be a timelock period, in which its decision can be reverted. (This property is somewhat jeopardized by [An admin proposal could circumvent the timelock mechanism.](#))

Recommendation

Make sure to return an error in case of an unsupported proposal type.

Status

Successfully resolved in [PR#101](#).

An admin proposal could circumvent the timelock mechanism

Type	Implementation
Severity	1 Low
Impact	3 Medium
Exploitability	1 Low
Status	Acknowledged

Involved artifacts

- [neutron-dao/dao/neutron-chain-manager/src/contract.rs](https://github.com/neutron-dao/dao/neutron-chain-manager/src/contract.rs)

Description

The chain manager allows for adding a strategy for an arbitrary address. This includes addresses of non-DAOs.

Problem Scenarios

If a non-DAO address were added with a strategy, it can send the `ExecuteMessages` directly to the chain manager contract, thus circumventing the timelocking mechanism.

The problem scenario is strongly limited by the fact that an actor with `AllowAll` strategy needs to add each new actor with an admin strategy. Thus, there is a strict control by (likely) the mainDAO.

Recommendation

Use the `CodeInfo` query to check whether the newly added strategy has an appropriate codeID.

Status

Acknowledged, but will likely be resolved at a later point, since now the issue is highly guarded by governance process.

There is no guarantee that the mainDAO remains among managers

Type	Protocol
Severity	0 Informational
Impact	1 High
Exploitability	0 None
Status	Acknowledged

Involved artifacts

- [neutron-dao/dao/neutron-chain-manager/src/contract.rs](#)

Description

The chain manager contract makes sure that there is always at least one admin with `AllowAll` strategy. However, this does not need to be the mainDAO: an another admin with `AllowAll` strategy may remove any other admin.

Problem Scenarios

We recognize that the current implementation is the intended design. The invariant that there is always at least one `AllowAll` strategy makes sure that the contract is not blocked. However, it would make sense to insist that the mainDAO retains control at all times. That invariant (mainDAO always has `AllowAll` strategy) then also covers the original one, of always having at least one admin with `AllowAll` strategy.

Recommendation

Replace the `no_admins_left` check in `execute_remove_strategy` and `execute_add_strategy` in case of demotion with check that mainDAO is never removed nor demoted.

Status

Acknowledged, but will likely be resolved at a later point, since now the issue is highly guarded by governance process.

Handling of conflicting permissions in AllowOnly strategies

Type	Implementation
Severity	1 Low
Impact	1 Low
Exploitability	1 Low
Status	Acknowledged

Involved artifacts

- [neutron-dao/contracts/dao/neutron-chain-manager/src/contract.rs](#)

Description

When adding a new strategy, the strategy is simply [saved to the storage](#) (provided that the sender is authorized to add a strategy). An `AllowOnly` strategy may contain conflicting permissions. (For instance, a strategy be the following vector: `[CronPermission{add_schedule: true, remove_schedule: false}, CronPermission{add_schedule: true, remove_schedule: false}]`)

Problem Scenarios

The ability to contain conflicting strategies creates an ambiguous situation. In the code of `neutron-chain-manager`, it is not handled consistently for different `Permission` variants.

Let us examine how the properties are checked for different `Permission` variants

- `CronPermission`: The check is performed in `has_cron_add_schedule_permission` (and, similarly, in `has_cron_remove_schedule_permission`). This check [iterates](#) over all permissions in the `AllowOnly` strategy, and each time it finds a `CronPermission`, it updates the variable that will be returned at the end of the loop, `has_permission`, to be equal to `cron_permission.add_schedule`. Thus, the final return value will be the return `add_schedule` value of the **last** `CronPermission` present.
- `AdminProposal::ParamChangeProposal`: Here, due to the structure of `ParamChangePermission`, there can be no conflicting permissions. Simply, a change is permitted if there is a `param_change_permission` that [has the same](#) `subspace` and `key` fields as the `param_change` request.
- `AdminProposal::ProposalExecuteMessage`: For this kind of message, only the check for `MSG_TYPE_UPDATE_PARAMS_CRON` is supported and checked in the `check_cron_update_msg_params` function. The function [first loads](#) the

`cron_update_param_permission` by iterating over all permissions and returning the first `CronUpdateParamsPermission`. In effect, the **first** value will be the relevant one.

This inconsistency may create confusion. In particular, it will reject as unauthorized a request to add a schedule, even though the strategy may have permission to do it. Similarly, it will allow a change of cron parameters even though there may be an explicit permission forbidding it, as well as vice versa: it may deny a change even if there is an explicit permission allowing it.

We note that this scenario is guarded by the governance process of a (sub)DAO with `AllowAll` strategy.

Recommendation

We recommend adding permissions validation before storing new strategies to eliminate the possibility of conflicting strategies.

Status

Acknowledged, but will likely be resolved at a later point, since now the issue is highly guarded by governance process.





Appendix: Vulnerability Classification

For classifying vulnerabilities identified in the findings of this report, we employ the simplified version of [Common Vulnerability Scoring System \(CVSS\) v3.1](#), which is an industry standard vulnerability metric. For each identified vulnerability we assess the scores from the *Base Metric Group*, the [Impact score](#), and the [Exploitability score](#). The *Exploitability score* reflects the ease and technical means by which the vulnerability can be exploited. That is, it represents characteristics of the *thing that is vulnerable*, which we refer to formally as the *vulnerable component*. The *Impact score* reflects the direct consequence of a successful exploit, and represents the consequence to the *thing that suffers the impact*, which we refer to formally as the *impacted component*. In order to ease score understanding, we employ [CVSS Qualitative Severity Rating Scale](#), and abstract numerical scores into the textual representation; we construct the final *Severity score* based on the combination of the Impact and Exploitability sub-scores.

As blockchains are a fast evolving field, we evaluate the scores not only for the present state of the system, but also for the state that deems achievable within 1 year of projected system evolution. E.g., if at present the system interacts with 1-2 other blockchains, but plans to expand interaction to 10-20 within the next year, we evaluate the impact, exploitability, and severity scores wrt. the latter state, in order to give the system designers better understanding of the vulnerabilities that need to be addressed in the near future.

Impact Score

The Impact score captures the effects of a successfully exploited vulnerability on the component that suffers the worst outcome that is most directly and predictably associated with the attack.

Impact Score	Examples
 High	Halting of the chain; loss, locking, or unauthorized withdrawal of funds of many users; arbitrary transaction execution; forging of user messages / circumvention of authorization logic
 Medium	Temporary denial of service / substantial unexpected delays in processing user requests (e.g. many hours/days); loss, locking, or unauthorized withdrawal of funds of a single user / few users; failures during transaction execution (e.g. out of gas errors); substantial increase in node computational requirements (e.g. 10x)
 Low	Transient unexpected delays in processing user requests (e.g. minutes/a few hours); Medium increase in node computational requirements (e.g. 2x); any kind of problem that affects end users, but can be repaired by manual intervention (e.g. a special transaction)
 None	Small increase in node computational requirements (e.g. 20%); code inefficiencies; bad code practices; lack/incompleteness of tests; lack/incompleteness of documentation

Exploitability Score

The Exploitability score reflects the ease and technical means by which the vulnerability can be exploited; it represents the characteristics of the vulnerable component. In the below table we list, for each category, examples of actions by actors that are enough to trigger the exploit. In the examples below:

- *Actors* can be any entity that interacts with the system: other blockchains, system users, validators, relayers, but also uncontrollable phenomena (e.g. network delays or partitions).
- *Actions* can be

- *legitimate*, e.g. submission of a transaction that follows protocol rules by a user; delegation/redelegation/bonding/unbonding; validator downtime; validator voting on a single, but alternative block; delays in relaying certain messages, or speeding up relaying other messages;
- *illegitimate*, e.g. submission of a specially crafted transaction (not following the protocol, or e.g. with large/incorrect values); voting on two different alternative blocks; alteration of relayed messages.
- We employ also a *qualitative measure* representing the amount of certain class of power (e.g. possessed tokens, validator power, relayed messages): *small* for < 3%; *medium* for 3-10%; *large* for 10-33%, *all* for >33%. We further quantify this qualitative measure as relative to the largest of the system components. (e.g. when two blockchains are interacting, one with a large capitalization, and another with a small capitalization, we employ *small* wrt. the number of tokens held, if it is small wrt. the large blockchain, even if it is large wrt. the small blockchain)

Exploitability Score	Examples
● High	illegitimate actions taken by a small group of actors; possibly coordinated with legitimate actions taken by a medium group of actors
● Medium	illegitimate actions taken by a medium group of actors; possibly coordinated with legitimate actions taken by a large group of actors
● Low	illegitimate actions taken by a large group of actors; possibly coordinated with legitimate actions taken by all actors
● None	illegitimate actions taken in a coordinated fashion by all actors



Severity Score

The severity score combines the above two sub-scores into a single value, and roughly represents the probability of the system suffering a severe impact with time; thus it also represents the measure of the urgency or order in which vulnerabilities need to be addressed. We assess the severity according to the combination scheme represented graphically below.



As can be seen from the image above, only a combination of high impact with high exploitability results in a Critical severity score; such vulnerabilities need to be addressed ASAP. Accordingly, High severity score receive vulnerabilities with the combination of high impact and medium exploitability, or medium impact, but high exploitability.

Severity Score	Examples
● Critical	Halting of chain via a submission of a specially crafted transaction
● High	Permanent loss of user funds via a combination of submitting a specially crafted transaction with delaying of certain messages by a large portion of relayers
● Medium	Substantial unexpected delays in processing user requests via a combination of delaying of certain messages by a large group of relayers with coordinated withdrawal of funds by a large group of users

Severity Score	Examples
 Low	2x increase in node computational requirements via coordinated withdrawal of all user tokens
 Informational	Code inefficiencies; bad code practices; lack/incompleteness of tests; lack/incompleteness of documentation; any exploit for which a coordinated illegitimate action of all actors is necessary

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability, etc.) set forth in the associated Services Agreement. This report provided in connection with the Services set forth in the Services Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement.

This audit report is provided on an “as is” basis, with no guarantee of the completeness, accuracy, timeliness or of the results obtained by use of the information provided. Informal has relied upon information and data provided by the client, and is not responsible for any errors or omissions in such information and data or results obtained from the use of that information or conclusions in this report. Informal makes no warranty of any kind, express or implied, regarding the accuracy, adequacy, validity, reliability, availability or completeness of this report. This report should not be considered or utilized as a complete assessment of the overall utility, security or bugfree status of the code.

This audit report contains confidential information and is only intended for use by the client. Reuse or republication of the audit report other than as authorized by the client is prohibited.

This report is not, nor should it be considered, an “endorsement”, “approval” or “disapproval” of any particular project or team. This report is not, nor should it be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts with Informal to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor does it provide any indication of the client’s business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should it be leveraged as investment advice of any sort.

Blockchain technology and cryptographic assets in general and by definition present a high level of ongoing risk. Client is responsible for its own due diligence and continuing security in this regard.