# Security Audit – Squads Multisig v4

| | |
|---|---|
| Lead Auditor: | Robert Reith |
| Second Auditor: | Sebastian Fritsch |
| Administrative Lead: | Thomas Lambertz |

September 22$^{nd}$ 2023

# Table of Contents

# Executive Summary

**Neodyme** audited **Squads'** on-chain Multisig v4 program during May, June and September 2023. Due to the complex threat model for multisig contracts, the scope of this audit included both implementation security and social engineering resilience. The auditors found that Squads' multisig program comprised a clean design and far-above-standard code quality. According to Neodymes Rating Classification, **no critical or high vulnerabilities** and only **two medium-severity issues** were found. The number of findings identified throughout the audit, grouped by severity, can be seen in Figure 1.

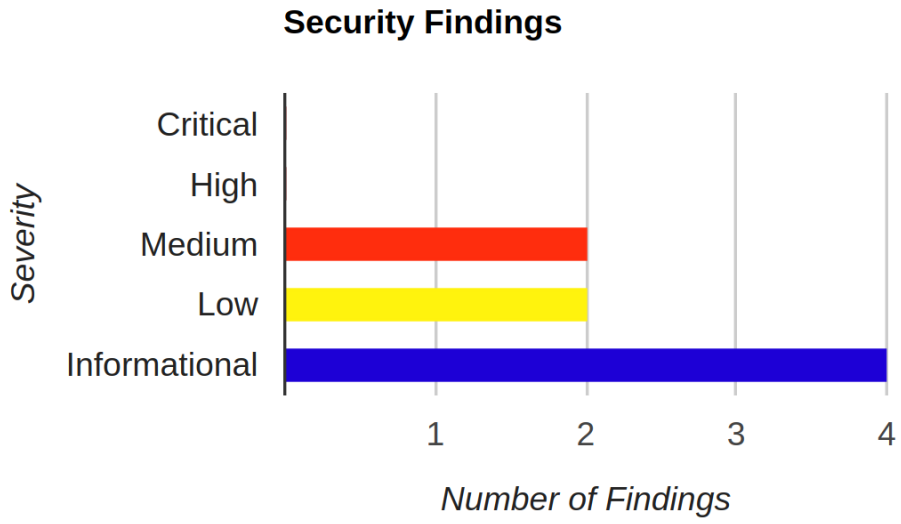


**Figure 1:** Overview of Findings

All findings were reported to the Squads developers and addressed promptly. The security fixes were verified for completeness by Neodyme. In addition to these findings, Neodyme delivered the Squads team a list of nit-picks and additional notes that are not part of this report.

# 1 Introduction

During spring 2023, Squads commissioned Neodyme to conduct a detailed security analysis of Squads' on-chain multisig v4 program. Two senior auditors performed the audit between May 15th and June 15th, 2023. To account for changes in the contract, additional auditing was carried out between September 19th and September 22nd, 2023. This report details all findings from both time spans.

The audit mainly focused on the contract's technical security, but also considered its design, especially concerning social engineering attack vectors. After the introduction, this report details the audit's Scope, gives a brief Overview of the Contract's Design, then goes on to document Findings, and finally discusses Social Engineering Attack Surface.

Neodyme would like to emphasize the high quality of Squads' work. Squads' team always responded quickly and **competent** to findings of any kind. Their **in-depth knowledge** about multisig programs was apparent during all stages of the cooperation. Evidently, Squads invested significant effort and resources into their product's security. Their **code quality is far above standard**, as the code is well documented, naming schemes are clear, and the overall architecture of the program is **clean and coherent**. The contract's source code has no unnecessary dependencies, relying mainly on the well-established anchor framework.

Yet, a large attack surface remains, as all multisig contracts are susceptible to non-technical social engineering attacks. Some of these risks are further discussed in the Social Attacks section.

## Findings Summary

During the audit, **four security-relevant** and **four informational** findings were identified. Squads remediated all of those findings before the protocol's launch.

In total, the audit revealed:

**0** critical • **0** high-severity • **2** medium-severity • **2** low-severity • **4** informational

issues.

The two medium-severity findings address a vulnerability where anyone could remove the spending-allowances of users, and a social engineering attack vector. All findings are detailed in section Findings.

# 2 | Scope

The contract audit's scope comprised of three major components:

- **Implementation** security of the source code
- Security of the **overall design**
- Resilience again **social engineering attacks**

All of the source code, located at https://github.com/Squads-Protocol/v4, is in scope of this audit. However, third-party dependencies are not. As Squads only relies on the well-established Anchor library, the security-txt standard, and the address-lookup program provided by Solana, this does not seem problematic.

On September 19th, 2023, additional changes to the Squads's code were included in the scope of this work. Those September changes included new instructions to add and remove spending limits in controlled multisigs and a design change to separate rent-paying signer accounts from contract-logic accounts.

Relevant source code revisions are:

- `ce140d682642d98666b5154656e2ec11d26e8933` • Start of the audit
- `c2e81e1748d9980c164b124880af7417a01b48ed` • Security fixes
- `72eef066f0269c2eada600e54bc5114675c9b416` • Last reviewed revision

# 3 | Project Overview

This section briefly outlines the Squads multisig v4 functionality, design and architecture followed by a discussion on multisig authorities and security features.

## Functionality

Squads Multisig v4 is the latest version of the Squads multisig on Solana. Using the program, anyone can create and administer multisig wallets. Such wallets are distinct from ordinary blockchain wallets in their property of being controlled by multiple parties instead of just one. This is achieved by defining a list of members of a wallet, a "Squad", that can create proposals about actions. Those actions can then be performed using the assets held by the multisig wallet. Every Squad member may vote on these proposals. Once consensus is achieved, a proposal can be executed. The exact definition of consensus can vary between different Squads. In particular, a variable called `threshold` defines how many members of a Squad need to agree on a proposal for it to become executable.

Squads implements three different types of proposals, **Config**, **Vault**, and **Batch**:

- **Config** proposals change the configuration of the multisig itself.
- **Vault** proposals perform a single blockchain transaction using one of the multisig's vaults. Simplified, this means each vault can sign as a regular Solana key, whereas the Squads program holds the private key. More technically, these vaults are program-derived-addresses (PDAs) that the multisig program controls.
- **Batch** proposals allow the chaining of multiple vault transactions into one proposal.

In addition, the contract implements a feature to allocate a spending limit to a set of members. These members are then allowed to spend (only) a certain amount from a predefined vault. Limits of this kind can be renewed periodically. For instance, once per week or month.

To allow large numbers of accounts to be passed to multisig transactions, which some third-party dapps might require, the multisig uses Address Lookup Tables (ALT). ALTs are a relatively new feature in Solana. They allow specifying transaction accounts in a special lookup account instead of the transaction directly.

Altogether, Squads Multisig v4 implements a basic but complete set of features to conduct blockchain operations in an organization. The Squads team is dedicated to delivering a set of features that contains no unnecessary functionality while maintaining high usability and flexibility. The code has an overall high quality with good documentation.

## On-Chain Data and Accounts

The on-chain Multisig v4 contract maintains multiple accounts to keep track of its state and data. In particular, the following data needs to be tracked:

- Configuration parameters of each multisig
- Member key list
- Proposal details
- Proposal instructions
- Votes
- Spending budgets
- Vaults

To represent this data on-chain, each instance of Multisig v4 has a `Multisig` account. This account stores the configuration and member list. To do so, the contract uses multiple PDAs. How these PDAs are seeded can be seen in Listing 1.

```
// ms : multisig
// es : ephemeral_signer
// idx : index
// tx : transaction
Multisig: "multisig" + "multisig" + create_key
Proposal: "multisig" + ms_key + "transaction" + ms_tx_idx + "proposal"
Batch: "multisig" + ms_key + "transaction" + ms_tx_idx
Batch Tx: "multisig" + ms_key + "transaction" + batch_idx + "
    batch_transaction" + batch_tx_idx
Config Transaction: "multisig" + ms_key + "transaction" + ms_tx_idx
Spending Limit: "multisig" + ms_key + "spending_limit" + create_key
Vault: "multisig" + ms_key + "vault" + vault_idx
Vault Transaction: "multisig" + ms_key + "transaction" + ms_tx_idx
Ephemeral Signer: "multisig" + tx_key + "ephemeral_signer" + es_idx
```

**Listing 1**: PDA Seeding Code

Any PDAs of the Multisig v4 instance are bound to their respective multisig state account's address via derivation seeds. Ony exception to this is the `Ephemeral Singer` PDA. It is derived from one specific transaction proposal account, which in turn is derived from the multisig key. Furthermore, `Batch`, `Config Transaction`, and `Vault Transaction` share their derivation path, as they each essentially represent one multisig action and the `multisig_tx_index` is incremented with each PDA. This design creates a clear separation between authorities of different multisigs configs. A curious technicality is that the `create_key` can be arbitrarily chosen but has to be a signer. It is essentially used as an arbitrary byte-string with a constant length to make multiple multisig instances possible.

The contract's funds are stored in the `Vault` PDAs. Each multisig can have multiple `Vault`s identified

by a `u8` index. To use them, a `Vault Transaction` has to be created. When executed, it will sign with the specified vault.

## Fees and Rent

The Squads multisig v4 program functions without any fees. Effectively, users can create a multisig for free, only paying Solana transaction fees and rent. Most accounts that the Squads Multisig v4 maintains are intended never to be deleted so that a multisig account history stays on-chain. One exception to this is the `Spending Limit` account, which can be removed to reclaim its rent into an arbitrary account.

## Instructions

The contract employs 21 instructions, which are briefly summarize here for completeness.

**Table 1:** Instructions with Descriptions

| Instruction | Category | Summary |
|---|---|---|
| multisigCreate | Permissionless | Create a multisig. |
| multisigAddMember | Admin-Only | Add a new member to the controlled multisig. |
| multisigRemoveMember | Admin-Only | Remove a member/key from the controlled multisig. |
| multisigSetTimeLock | Admin-Only | Set the `time_lock` config parameter for the controlled multisig. |
| multisigChangeThreshold | Admin-Only | Set the `threshold` config parameter for the controlled multisig. |
| multisigSetConfigAuthority | Admin-Only | Set the multisig `config_authority`. |
| multisigAddSpendingLimit | Admin-Only | Create a new spending limit for the controlled multisig. |
| multisigRemoveSpendingLimit | Admin-Only | Remove the spending limit from the controlled multisig. |
| configTransactionCreate | Member-Only | Create a new config transaction. |

| Instruction | Category | Summary |
|---|---|---|
| configTransactionExecute | Member-Only | Execute a config transaction. The transaction must be `Approved`. |
| vaultTransactionCreate | Member-Only | Create a new vault transaction. |
| vaultTransactionExecute | Member-Only | Execute a vault transaction. The transaction must be `Approved`. |
| batchCreate | Member-Only | Create a new batch. |
| batchAddTransaction | Member-Only | Add a transaction to the batch. |
| batchExecuteTransaction | Member-Only | Execute a transaction from the batch. |
| proposalCreate | Member-Only | Create a new multisig proposal. |
| proposalActivate | Member-Only | Update status of a multisig proposal from `Draft` to `Active`. |
| proposalApprove | Member-Only | Approve a multisig proposal on behalf of the `member`. The proposal must be `Active`. |
| proposalReject | Member-Only | Reject a multisig proposal on behalf of the `member`. The proposal must be `Active`. |
| proposalCancel | Member-Only | Cancel a multisig proposal on behalf of the `member`. The proposal must be `Approved`. |
| spendingLimitUse | Member-Only | Use a spending limit to transfer tokens from a multisig vault to a destination account. |

## Authority Structure and Off-Chain Components

A crucial part of the designs overview are authorities and components running off-chain. These authorities and components are described in the following.

### Upgrade Authority

A program's upgrade authority allows complete control over its behavior, signatures, and funds. Therefore it should be well protected. To our knowledge, the Squads team plans to burn the upgrade authority after the release. While this may make bugs unfixable, it also completely removes any risk associated with maintaining an upgrade authority.

#### Config Authority

Typically, multisigs are governed by their members. However, Squads Multisig v4 supports a special case where a multisig can be `controlled` by an admin. This feature is supposed to help with the initial setup of a new multisig. It allows the `config authority` to perform configuration actions such as adding or removing members, changing the threshold, changing spending limits, setting a timelock, or changing the config authority itself. Naturally, this may be abused when users are unaware of this functionality and join a multisig with set config authority. We recommend that any multisig with an active config authority be clearly marked in the UI.

#### Off-Chain Components

Users shall interact with squads primarily through its web-based user interface and an SDK. No other off-chain components are needed to use the program.

## Security Features

Squad relies on Solana's main framework, Anchor, which takes care of basic security checks such as owner checks. Additionally, Squads multisig v4 uses a `validate()` function to perform security checks on instruction calls. Each `validate()` function checks that all accounts are correctly linked to each other, permissions are in order, and account states are correct before anything is executed further. Furthermore, Squads implements three `invariant` functions, namely one for:

- multisig
- batches
- spending limits

These functions are executed after the a specific instructions to confirm that certain properties are always held up. For instance, these `invariant` functions check that there are no duplicate members and that the threshold is at most the number of permissioned voters of a multisig.

# 4 | Findings

This section outlines all of our findings. They are classified into one of five severity levels, detailed in Appendix C. In addition to these findings, Neodyme delivered the Squads team a list of nit-picks and additional notes which are not part of this report.

All findings are listed in Table 2 and further described in the following sections.

**Table 2:** Findings

| Name | Severity | State |
|---|---|---|
| [ND-SQD1-M1] Remove Any Spending Limit | Medium | Resolved |
| [ND-SQD1-M2] Approved Proposals Don't Become Stale | Medium | Resolved |
| [ND-SQD1-L1] Lookup Tables Used in Unfrozen State | Low | Resolved |
| [ND-SQD1-L2] Callee Programs Might be Updated Before Being Executed | Low | Resolved |
| [ND-SQD1-I1] Minority Multisigs Allow for Proposal Griefing | Low | Acknowl. |
| [ND-SQD1-I2] Reentrancy Voting Possible | Info | Resolved |
| [ND-SQD1-I3] TransactionMessage Not Validated | Info | Resolved |
| [ND-SQD1-I4] Controlled Multisigs Create Social Attack Vectors | Info | Acknowl. |

## ND-SQD1-M1 – Remove Any Spending Limit

| Severity | Impact | Affected Component | Status |
|---|---|---|---|
| **Medium** | Anyone can remove spending limits from all squads and obtain the rent. | Config Transaction Execute | Resolved |

In `config_transaction_execute`, the config action `ConfigAction::RemoveSpendingLimit` does not find the program address, but takes it from `remaining_accounts` without validating that it belongs to the current multisig. Therefore, an attacker can set up their own multisig and remove the spending limits of other multisigs, disabling the feature on the attacked multisig. The rent of the deleted account is transferred to an attacker-controlled account. An unpermissioned attacker could therefore steal all the rent from all spending limits, effectively turning off the feature for all squads.

#### Suggestion

Check that the supplied spending limit account is derived from the current multisig.

#### Remediation

Squads fixed this issue in commit: `cf349898befab373b8a14c4bb4bc56e73929e94d`.

The issue was fixed by requiring that the `SpendingLimit` account belongs to the multisig.

## ND-SQD1-M2 – Approved Proposals Don’t Become Stale

| Severity | Impact | Affected Component | Status |
|---|---|---|---|
| **Medium** | Abuse Potential | Proposal Execution | Resolved |

Once a proposal has been approved, it can be executed forever and doesn’t become stale, even after the multisig changes (e.g. as new members join etc.). This issue could be abused by an attacker in the following way:

1. Create a new multisig with one member
2. Create BatchTx or VaultTx to withdraw many tokens from vaults
3. Vote approve
4. Tx becomes approved
5. Don’t execute it
6. Add more members and pretend this never happened
7. Many epochs later, use approved Tx to perform a rug pull

#### Suggestion

Any execution validation function should check that the to-be-executed transaction isn’t stale.

#### Remediation

Squads fixed this issue in commit: `dcc07dcce96d0a9986a503fce8a7193d1dad76b8`.

The issue was fixed with a new policy regarding approved proposals.

The policy is defined as follows:

- All stale proposals can be neither approved nor rejected after they have become stale
- Config transactions with an approved but stale proposal cannot be executed
- Vault or batch transactions with an approved but stale proposal can still be executed. The logic here is that a change in the multisig config should not automatically invalidate some previously approved spending of funds
- All approved but stale proposals can be canceled. This allows members to cancel any lingering but approved vault/batch transactions

## ND-SQD1-L1 – Lookup Tables Used in Unfrozen State

| Severity | Impact | Affected Component | Status |
|---|---|---|---|
| **Low** | Abuse Potential | Transaction Validation | Resolved |

Lookup tables can be appended to after a transaction has been proposed and even approved. This may create a situation where members of a multisig approve an action, which is then changed before execution. As lookup tables are append-only, an attacker must reference indices in the `MessageAddressTableLookup` that are out of bounds at the time of creation, so this could be noticed beforehand. This issue could be avoided altogether by only using frozen lookup tables.

### Suggestion

After deserializing a new proposed vault- or batch- transaction, the address lookup tables should be verified to be frozen.

### Remediation

The Squads team brought to our attention that crucial third-party programs do not freeze their address lookup tables. Therefore, enforcing a freeze could severely diminish ecosystem interoperability. As a compromise, Neodyme recommended a fix by adding a warning in the Squads UI. Squads confirmed that such a fix would be implemented.

## ND-SQD1-L2 – Callee Programs Might be Updated Before Being Executed

| Severity | Impact | Affected Component | Status |
|---|---|---|---|
| **Low** | Abuse Potential | Proposal Execution | Resolved |

When a batch or vault transaction calls an external program, there is no assurance that the program being called will not be modified during or after voting. Many programs on Solana have an upgrade-authority that can fully replace the controlled program. In the ideal case, all multisig members always vote on the same program and can rest assured that the program they voted on is the same when the proposal is executed. This issue also exists for standard, non-multisig Solana transaction signatures, which could be front-runned with an upgrade. But due to being an on-chain program, Squads can add a check for this and thus be even more secure.

#### Suggestion

To mitigate this issue, the multisig could verify upon execution that the target program has not been upgraded since the proposal was started. To do so, the ProgramData account of Upgradeable Programs has to be checked for the last deployed slot. The last deployed slot should not be after the proposal creation slot.

#### Remediation

Squads proposed fixing this in the UI by adding a warning. In addition, they are considering a “high-security-mode” where such a check is enforced. However, a general check was declined due to UX considerations.

## ND-SQD1-I1 – Minority Multisigs Allow for Proposal Griefing

| Severity | Impact | Affected Component | Status |
|---|---|---|---|
| **Info** | Inconsistency | Voting | Acknowledged |

In a minority multisig, e.g. an $m/n$ multisig where the threshold $m < n/2$, a situation may arise where the multisig will behave like $m = n/2 + 1$. When $m$ members approve a transaction, and the remaining $(n - m)$ disapprove, the transaction is approved overall and ready for execution. However, because the cancellation threshold is the same as the approval threshold, now the $n - m$ disapproving members have enough voting power to cancel the transaction instantly. When there is a timelock, this may theoretically lead to all multisig transactions being canceled when the approving side is not at least $n/2 + 1$ members. This might be working as intended, but it is an interesting consideration we wanted to address.

#### Suggestion

A separate cancellation threshold could be considered.

#### Remediation

Acknowledged and accepted without changes.

## ND-SQD1-I2 – Reentrancy Voting Possible

| Severity | Impact | Affected Component | Status |
|---|---|---|---|
| **Info** | Inconsistency | Proposal Execution | Resolved |

To avoid issues arising from reentrancy, Squads forbids it for most instruction types. An interesting case occurs when reentrancy is used to call `proposal_vote::cancel`. This is because the state of a proposal is only set to `Executed` at the end of execution. During execution, the state of the proposal is `ProposalStatus::Approved`, so it is eligible for cancellation. Therefore, one could create a transaction that votes cancel on itself. If the threshold is met, the transaction will change its status to `Cancelled`, execution will finish, and the status will be set to `Executed`. So far, there is no impact from this peculiarity. However, it might be cleaner to avoid this possibility altogether.

### Suggestion

We suggest two different approaches to mitigate this issue. One way would be to disallow reentrancy to the vote instruction altogether. Another way could be to introduce a transitory `Proposal::Executing` state. This state would be activated when the execution of a proposal starts and be active while it is executing.

### Remediation

Squads fixed this issue in commit: `c2e81e1748d9980c164b124880af7417a01b48ed`.

The fix implements, as suggested, the transitory `Proposal::Executing` state. Squads later improved this fix further in Pull Request #24 (https://github.com/Squads-Protocol/v4/pull/24). This change prevents a vault transaction to pass protected accounts as writable. Protected accounts include all accounts used by Squads in this transaction.

## ND-SQD1-I3 – TransactionMessage Not Validated

| Severity | Impact | Affected Component | Status |
|---|---|---|---|
| **Info** | Unexecutable Proposals | Transaction Validation | Resolved |

Users are able to create `VaultTx` or `BatchTx` that contain completely bogus transaction messages. This includes transactions with indices that are out of bounds, invalid program ids, and so on.

These kind of proposal will be unexecutable, but could be vectors in combination with social attacks, or become executable with changes to the Solana runtime.

#### Suggestion

While this does not directly impact the security of the program, we recommend checking for invalid transactions beforehand.

#### Remediation

Squads fixed this issue in commit: `e115a4047e64c97f6cc5d22021ff390403bf59e1`.

## ND-SQD1-I4 – Controlled Multisigs Create Social Attack Vectors

| Severity | Impact | Affected Component | Status |
|---|---|---|---|
| **Info** | Abuse Potential | Controlled Multisigs | Acknowledged |

The “controlled multisig” feature effectively gives the controller power over the multisig. This undermines the purpose of a multisig. This can be useful during setup, so the feature has legitimate use cases. However, it may be abused for rug pulls by creating a controlled multisig. The multisig’s capabilities might not be obvious to users, creating social engineering attack surface.

#### Suggestion

We recommend enforcing that users cannot vote in controlled multisigs. Alternatively, a warning about controlled multisigs should be added to the UI.

#### Remediation

Squads acknowledged the issue but noted that some users need it for their specific use case.

# 5 | Social Engineering Attack Vectors

In this section, we discuss several social attack patterns arising from the program's functionality. None of them are currently of major concern to the program's security itself.

## Attacks Related to Social Engineering by Adversarial Members

One of the biggest threats that members of multisigs face is that of adversarial members. If another member wants to gain control over a multisig, they would most likely either try to:

- use a **controlled multisig** as described in ND-SQD1-I4
- create **manipulative proposals** where the impact of running said proposal is not easily understood by unsuspecting parties

#### Backdoored Proposals

When creating a new transaction and corresponding proposal containing a description, there is no guarantee that the description matches what the proposal does. A malicious party could claim that some Sol will be transferred to wallet X, but actually, the transfer is sent to wallet Y. If users don't carefully check the instructions of each proposal they vote on, they could easily be fooled. However, such attacks can't really be addressed on-chain. The best countermeasure is to educate users and give them good tools to inform themselves.

#### Key Replacement Attack

This attack makes it possible to kick a user from a $N : N$ multisig - even if the user wouldn't agree to being kicked. To do so, the other $N - 1$ parties conspire and create a proposal where one claims that their key needs to be replaced. In this proposal, they want to execute two instructions:

1. Remove a member
2. Add this member with their new key

This sounds completely innocent, but it changes the $N : N$ multisig into a $N - 1 : N$ multisig. When the member is removed, the multisig turns into a $N - 1 : N - 1$ multisig. When the new key is added, the number of members increases, but the threshold is not adjusted upwards. Now the conspiring members could kick that user.

# A | About Neodyme

Security is difficult.

To understand and break complex things, you need a certain type of people. People who thrive in complexity, who love to play around with code, and who don't stop exploring until they fully understand every aspect of it. That's us.

Our team never outsources audits. Having found over 80 High or Critical bugs in Solana's core code itself, we believe that Neodyme hosts the most qualified auditors for Solana programs. We've also found and disclosed critical vulnerabilities in many of Solana's top projects and have responsibly disclosed issues that could have resulted in the theft of over $10B in TVL on the Solana blockchain.

All of our team members have a background in competitive hacking. During such hacking competitions, called CTFs, we competed and collaborated while finding vulnerabilities, breaking encryption, reverse engineering complicated algorithms, and much more. Through the years, many of our team members have won national and international hacking competitions, and keep ranking highly among some of the hardest CTF events world-wide. In 2020, some of our members started experimenting with validators and became active members in the early Solana community. With the prospect of an interesting technical challenge and bug bounties, they quickly encouraged others from our CTF team to look for security issues in Solana. The result was so successful that after reporting several bugs, in 2021, the Solana Foundation contracted us for source code auditing. As a result, Neodyme was born.

# B | Methodology

Neodyme prides itself on not being a checklist auditor. We adapt our approach to each audit, investing considerable time into understanding the program upfront and exploring its expected behavior, edge cases, invariants, and ways in which the latter could be violated. We use our uniquely deep knowledge of Solana internals, and our years-long experience in auditing Solana programs to even find bugs that others miss. We often extend our audit to cover off-chain components, in order to see how users could be tricked or the contract affected by bugs in those components.

Nonetheless, we also have a list of common vulnerability classes, which we always exhaustively look for. We provide a sample of this list below.

- Rule out common classes of Solana contract vulnerabilities, such as:
  - Missing ownership checks
  - Missing signer checks
  - Signed invocation of unverified programs
  - Solana account confusions
  - Redeployment with cross-instance confusion
  - Missing freeze authority checks
  - Insufficient SPL account verification
  - Missing rent exemption assertion
  - Casting truncation
  - Arithmetic over- or underflows
  - Numerical precision errors
- Check for unsafe design which might lead to common vulnerabilities being introduced in the future
- Check for any other, as-of-yet unknown classes of vulnerabilities arising from the structure of the Solana blockchain
- Ensure that the contract logic correctly implements the project specifications
- Examine the code in detail for contract-specific low-level vulnerabilities
- Rule out denial of service attacks
- Rule out economic attacks
- Check for instructions that allow front-running or sandwiching attacks
- Check for rug pull mechanisms or hidden backdoors

# C | Vulnerability Severity Rating

**Critical** Vulnerabilities that will likely cause loss of funds. An attacker can trigger them with little or no preparation, or they are expected to happen accidentally. Effects are difficult to undo after they are detected.

**High** Bugs that can be used to set up loss of funds in a more limited capacity, or to render the contract unusable.

**Medium** Bugs that do not cause direct loss of funds but that may lead to other exploitable mechanisms, or that could be exploited to render the contract partially unusable.

**Low** Bugs that do not have a significant immediate impact and could be fixed easily after detection.

**Info** Bugs or inconsistencies that have little to no security impact.

**Neodyme AG**

Dirnismaning 55
Halle 13
85748 Garching
E-Mail: contact@neodyme.io

https://neodyme.io