Security Audit - Token-2022

conducted by Neodyme AG

Lead Auditor:	Robert Reith
Second Auditor:	Mathias Scherer
Administrative Lead:	Thomas Lambertz

April 22, 2024

Table of Contents

1 Executive Summary **3**

2 Introduction **4**
Summary of Findings 4

3 Scope **5**

4 Project Overview **6**
Functionality 6
On-Chain Data and Accounts 6
Instructions 7
Authority Structure and Off-Chain Components 13

5 Findings **15**
[ND-ANZ-LO-01] Self delegation circumvents CPI guard 16
[ND-ANZ-IN-01] Mint Resurrection Attacks 18

Appendices

A About Neodyme **20**

B Methodology **21**
Select Common Vulnerabilities 21

C Vulnerability Severity Rating **23**

1 | Executive Summary

Neodyme audited **Anza's** on-chain token program from April 2024 until May 2024.

The auditors found that Anza's Token-2022 comprised a clean design and far-above-standard code quality. According to Neodymes Rating Classification, **1 security relevant** and **1 informational** were found. The number of findings identified throughout the audit, grouped by severity, can be seen in Figure 1.

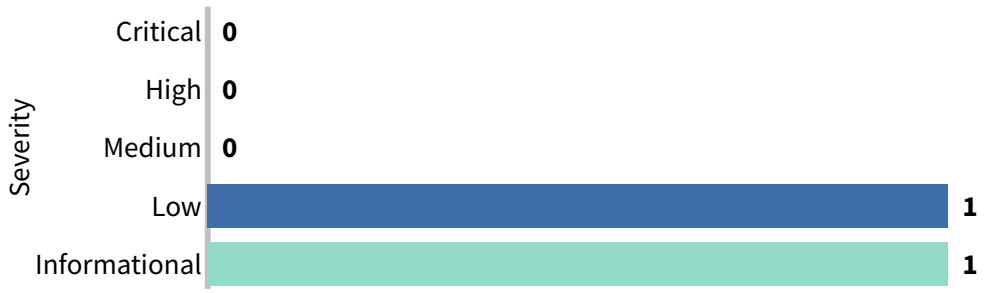


Figure 1: Overview of Findings

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

2 | Introduction

From April 2024 until May 2024, Anza engaged Neodyme to do a detailed security analysis of their Token-2022 program. Two senior security researchers from Neodyme conducted independent full audits of the contract between the 22th of April 2024 and the 13th of May 2024. Both auditors have a long track record of finding critical and other vulnerabilities in Solana programs, as well as in Solana's core code itself.

The audit focused on the contract's technical security. In the following sections, we present our findings and discuss worst-case scenarios for authority compromise, and provide some general notes for considerations that may be useful in the future.

Neodyme would like to emphasize the high quality of Anza's work. Anza's team always responded quickly and **competently** to findings of any kind.

Their **deep proficiency** in developing programs for the Solana ecosystem was evident at every stage of the audit, reflecting a thorough understanding of Solana's ecosystem and its intricacies. It was clear that Anza had invested significant resources into building a secure and robust token implementation. While some minor technical debt was observed within the contract—typical of rapidly evolving platforms like Solana—it was limited in scope and did not pose any immediate risks to the overall security posture.

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 upon the `solana-program` crate.

Summary of Findings

All found issues were **quickly remediated**. In total, the audit revealed:

0 critical • **0** high-severity • **0** medium-severity • **1** low-severity • **1** informational

issues. We further discuss all authorities in Section 4.4.

3 | Scope

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

- Primarily, the **Implementation** security of the contract's source code
- Additionally, security of the **overall design**

Neodyme considers the source code, located at https://github.com/solana-labs/solana-program-library/tree/master/token/program-2022, in scope for this audit. Third-party dependencies are not in scope. Anza only relies on the solana-program library, which is well-established. During the audit, minor changes and fixes were made by Anza, which the auditors also reviewed in-depth.

Relevant source code revisions are:

- `8c8e7de68b96f8853fdc555ce0af3cfdc717bf55` · Start of the audit
- `dc85c9810bb23f852dbdfbd170ec4fafe9420ebb` · Last reviewed revision

4 | Project Overview

This section briefly outlines Token-2022's functionality, design, and architecture, followed by a detailed discussion of all related authorities.

Functionality

Token-2022 (a.k.a. Token Extensions) enhances and expands the capabilities of the original Token program, adding more options and use cases for token creators. The extension support introduces new features that users can benefit from.

An extension adds extra features to standard tokens, allowing developers to customize token functionality, such as adding transfer hooks or minting restrictions, without changing the core token program. This flexibility enables more tailored token implementations on the Solana blockchain.

Extensions can be applied to both mints and tokens accounts.

The data required by the extensions is stored in either the token account or mint account within the `tlv_data` section which is located at the end of the account.

On-Chain Data and Accounts

The on-chain token program needs to keep track of token configurations, total supply, and the amount of tokens each user holds.

A mint account, which represents a token, stores the mint authority, total supply, decimal points used, an optional freeze authority, and all additional data for extensions such as the default account state, the transfer hook used, etc. This account isn’t derived from any particular seed and has to be rent-exempt before initializing it.

A token account is responsible for keeping track of how many tokens an address holds. It stores not only the amount of tokens but also the mint it is associated with, the owner of the account, the state (frozen or not), an optional close authority, and if a specific amount is delegated to a different address. This account isn’t derived from any particular seed and has to be rent-exempt before initializing it.

A multisig account is a simple implementation of a multisig that allows up to 11 addresses as members. While the multisig account doesn’t hold any token information and therefore is independent of any Mint, one multisig account can control an infinite amount of token and mint accounts. The program supports the use of multisigs on any instruction that requires an authority.

Instructions

The contract has a total of 56 instructions, which we briefly summarize here. Some extensions have instructions of their own, which are also part of the contract. For better readability, we split the instructions into sections.

Base Instructions

Instruction	Category	Summary
InitializeMint	Permissionless	Initializes a new mint account.
InitializeMint2	Permissionless	Like InitializeMint but requires the Rent sysvar to be provided.
InitializeAccount	Permissionless	Initializes a new token account for the given mint.
InitializeAccount2	Permissionless	Initializes a new token account for the given mint. The owner is passed via instruction data.
InitializeAccount3	Permissionless	Like InitializeAccount2 but requires the Rent sysvar to be provided.
InitializeMultisig	Permissionless	Initializes a new multisig.
InitializeMultisig2	Permissionless	Like InitializeMultisig but requires the Rent sysvar to be provided.
Transfer (deprecated)	TokenOnwer, Delegatee, PermanentDelegatee	Transfers token from one token account to another.
TransferChecked	TokenOnwer, Delegatee, PermanentDelegatee	Transfers token from one token account to another. It also checks if the passed mint account has the expected amount of decimal places.
Approve	TokenOwner	Approves a new delegate with a given amount.
ApproveChecked	TokenOwner	Approves a new delegate with a given amount. It also checks if the passed mint account has the expected amount of decimal places.
Revoke	TokenOwner, Delegatee	Revokes a delegation.

Instruction	Category	Summary
SetAuthority	TokenAuthority, MintAuthority	Sets a new authority for the given `AuthorityType`. Can be called by the authority that holds that power.
MintTo	MintAuthority	Mints new tokens to the given token account.
MintToChecked	MintAuthority	Mints new tokens to the given token account. It also checks if the passed mint account has the expected amount of decimal places.
Burn	TokenOwner, Delegatee, PermanentDelegatee	Burns the given amount of tokens from the token account
BurnChecked	TokenOwner, Delegatee, PermanentDelegatee	Burns the given amount of tokens from the token account. It also checks if the passed mint account has the expected amount of decimal places.
CloseAccount	TokenOwner, TokenCloseAuthority, MintCloseAuthority	Closes the given account.
FreezeAccount	FreezeAuthority	Freezes the given token account.
ThawAccount	FreezeAuthority	Thaws the given token account.
SyncNative	Permissionless	Updates a wrapped / native token account amount with the underlying lamports.
GetAccountDataSize	Permissionless	Gets the required size of an account for the given mint and extension types.
AmountToUiAmount	Permissionless	Returns the UI representation of an amount for the given mint. Adds interest if the mint is an interest-bearing token.
UiAmountToAmount	Permissionless	Returns the raw representation of an amount for the given mint. Removes interest if the mint is an interest-bearing token.
Reallocate	TokenOwner	Resizes the given token account if necessary to fit new extensions.
CreateNativeMint	Permissionless	Can only be invoked once to create a mint account to represent native SOL.
WithdrawExcessLamports	AccountOwner	Transfers excess lamports from the given account to the destination account.

Instruction	Category	Summary
InitializeImmutableOwner	Permissionless	Initializes the Immutable Owner extension for the given uninitialized token account.
InitializeMintCloseAuthority	Permissionless	Initializes the Mint Close Authority extension for the given uninitialized mint account.
InitializeNonTransferableMint	Permissionless	Initializes the Non Transferable extension for the given uninitialized mint account.
InitializePermanentDelegate	Permissionless	Initializes the Permanent Delegate extension for the given uninitialized mint account.

Transfer Fee Instructions

Instruction	Category	Summary
InitializeTransferFeeConfig	Permissionless	Initializes the Transfer Fee extension for the given uninitialized mint account.
TransferCheckedWithFee	TokenOnwer, Delegatee, PermanentDelegatee	Like as TransferChecked but checks the actual fee versus the passed expected fee.
WithdrawWithheldTokensFromMint	WithdrawAuthority	Transfers all withheld tokens in the mint to an account.
WithdrawWithheldTokensFromAccounts	WithdrawAuthority	Transfers all withheld tokens in the given token accounts to an account.
HarvestWithheldTokensToMint	Permissionless	Transfers all withheld tokens in the given token accounts to the mint account.
SetTransferFee	TransferFeeConfigAuthority	Sets a new transfer fee.

Confidential Transfer Instructions

Instruction	Category	Summary
InitializeMint	Permissionless	Initializes the Confidential Transfer extension for the given uninitialized mint account.

Instruction	Category	Summary
UpdateMint	ConfidentialTransferAuthority	Updates the extension settings.
ApproveAccount	ConfidentialTransferAuthority	Approves the given token account to use the extension.
EmptyAccount	TokenOwner	Sets all balances to 0 so the account can be closed in a second step.
Deposit	TokenOwner	Transfers tokens from the non-confidential to the confidential side in the same token account.
Withdraw	TokenOwner	Transfers tokens from the confidential to the non-confidential side in the same token account.
Transfer	TokenOwner	Confidentially transfers token from one token account to another.
ApplyPendingBalance	TokenOwner	Applies the pending balance to the available balance. This allows received tokens to be spent.
EnableConfidentialCredits	TokenOwner	Configure a token account to accept incoming confidential transfers.
DisableConfidentialCredits	TokenOwner	Configure a token account to reject any incoming confidential transfers.
EnableNonConfidentialCredits	TokenOwner	Configure a token account to accept incoming non-confidential transfers.
DisableNonConfidentialCredits	TokenOwner	Configure a token account to reject incoming non-confidential transfers.
TransferWithSplitProofs	TokenOwner	Like Transfer but with support to split large ZK proofs.

Default Account State Instructions

Instruction	Category	Summary
Initialize	Permissionless	Initializes the extension for the given uninitialized mint account.
Update	FreezeAuthority	Updates the configured default account state.

Memo Transfer Instructions

Instruction	Category	Summary
Enable	TokenOwner	Enables the requirement for a Memo IX within the same TX as a transfer.
Disable	TokenOwner	Disables the requirement for a Memo IX within the same TX as a transfer.

Interest Bearing Mint Instructions

Instruction	Category	Summary
Initialize	Permissionless	Initializes the Interest Bearing Mint extension for the given uninitialized mint account.
UpdateRate	RateAuthority	Updates interest rate of the given mint account.

CPI Guard Instructions

Instruction	Category	Summary
Enable	TokenOwner	Enables the CPI Guard extension on the given token account. It cannot be called via CPI.
Disable	TokenOwner	Disables the CPI Guard extension on the given token account. It cannot be called via CPI.

Transfer Hook Instructions

Instruction	Category	Summary
Initialize	Permissionless	Initializes the Transfer Hook extension for the given uninitialized mint account.
Update	TransferHookAuthority	Updates the transfer hook program used by the given mint account.

Confidential Transfer Fee Instructions

Instruction	Category	Summary
InitializeConfidentialTransferFeeConfig	Permissionless	Initializes the Confidential Transfer Fee extension on the given uninitialized mint account.
WithdrawWithheldTokensFromMint	WithdrawAuthority	Transfer all withheld confidential tokens in the mint to an account.
WithdrawWithheldTokensFromAccounts	WithdrawAuthority	Transfers all withheld confidential tokens in the given token accounts to an account.
HarvestWithheldTokensToMint	Permissionless	Transfers all withheld confidential tokens in the given token accounts to the mint account.
EnableHarvestToMint	TransferFeeAuthority	Enables the ability to harvest fees from token accounts into the mint account.
DisableHarvestToMint	TransferFeeAuthority	Disables the ability to harvest fees from token accounts into the mint account.

Metadata Pointer Instructions

Instruction	Category	Summary
Initialize	Permissionless	Initializes the Metadata Pointer extension on the given uninitialized mint account.
Update	MetadataPointerAuthority	Updates the metadata address in the given mint account.

Group Pointer Instructions

Instruction	Category	Summary
Initialize	Permissionless	Initializes the Group Pointer extension on the given uninitialized mint account.
Update	GroupPointerAuthority	Updates the group address in the given mint account.

Group Member Pointer Instructions

Instruction	Category	Summary
Initialize	Permissionless	Initializes the Group Member Pointer extension on the given uninitialized mint account.
Update	GroupMemberAuthority	Updates the group member address in the given mint account.

Authority Structure and Off-Chain Components

The Token-2022 program has many different authorities that control various aspects of mint and token accounts. In the following table, the Storage location describes if the authority is in the Base account data, inherited from the old Token program, or if it is defined by a certain extension.

Storage location	Account	Authority	Capabilities
Base	Mint	mint_authority	Set new mint authority, mint new tokens, withdraw excess lamports.
Base	Mint	freeze_authority	Update default account state, freeze or unfreeze an account.
Confidential Transfer Ext.	Mint	authority	Approve accounts, change auto-approval, change auditor's public key.
Confidential Transfer Ext.	Mint	auditor	Decrypt confidential transfer amounts.
Confidential Transfer Fee Ext.	Mint	authority	Withdraw fees from accounts and mint, sending them to fee recipient.
Mint Close Authority Ext.	Mint	close_authority	Close the mint.
Group Member Pointer Ext.	Mint	authority	Update member address.
Group Member Ext.	Mint	authority	Update group address.
Interest Bearing Config Ext.	Mint	rate_authority	Update the interest rate.

Storage location	Account	Authority	Capabilities
Metadata Pointer Ext.	Mint	authority	Update the metadata address.
TokenGroup Ext.	Mint	update_authority	Update the group authority, initialize new members, update the group maximum size.
Token Metadata Ext.	Mint	update_authority	Update metadata fields and values, remove metadata fields.
Transfer Fee Ext.	Mint	transfer_fee_config_authority	Update the transfer fee configuration.
Transfer Fee Ext.	Mint	withdraw_withheld_authority	Withdraw withheld amounts from accounts and mints.
Transfer Hook Ext.	Mint	authority	Update the transfer hook program.
Permanent Delegate Ext.	Mint	delegate	Transfer or burn any amount from any token account.
Base	Account	owner	Transfer tokens, initialize extensions, delegate a specific amount to another address, revoke a delegation, burn tokens, withdraw excess lamports.
Base	Account	close_authority	Close the account.
Base	Account	delegate	Transfer or burn the delegated amount.

Each authority listed here can update to a new public key. The only exception is the auditor set in the confidential transfer extension.

Upgrade Authority

As with any contract, the upgrade authority has complete control over the program and the funds it controls. It is hence one of the most critical components of the security of the protocol. By maliciously upgrading the contract, the upgrade authority can irreversibly transfer control of all tokens to itself or other parties.

Anza is aware of this and has put considerable effort into making the upgrade authority safe. They have established a robust 4-out-of-6 Squads V3 multisig as its upgrade authority.

5 | 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 Anza team a list of nit-picks and additional notes which are not part of this report.

All findings are listed in Table 13 and further described in the following sections.

Identifier	Name	Severity	Status
ND-ANZ-LO-01	Self delegation circumvents CPI guard	LOW	Resolved
ND-ANZ-IN-01	Mint Resurrection Attacks	INFORMATIONAL	Accepted

Table 13: Findings

[ND-ANZ-LO-01] Self delegation circumvents CPI guard

Severity	Impact	Affected Component	Status
LOW	Loss of funds	CPIGuard	Resolved

We found that the CPI lock is disabled when a transfer is called as a delegate. However, an account may be delegated to itself, which would effectively disable the CPI lock unintentionally. An example attack would look like this:

1. User creates an account and activates the CPI lock
2. Attacker convinces the user to delegate to themselves -> From a user's perspective their tokens aren't at risk because the CPI lock is active and they delegated to themselves and not some random party.
3. The attacker convinces the user to call their program.
4. The attacker's program forwards the signer to token-2022 and calls transfer. Here the CPI lock isn't enforced because the delegate has a "higher" priority compared to the owner and thus the code would treat it as a delegation transfer and not an owner transfer.

processor.rs, lines 397-418

```
if cmp_pubkeys(authority_info.key, &delegate) => {
    Self::validate_owner(
        program_id,
        &delegate,
        authority_info,
        authority_info_data_len,
        account_info_iter.as_slice(),
    )?;
    let delegated_amount = u64::from(source_account.base.delegated_amount);
    if delegated_amount < amount {
        return Err(TokenError::InsufficientFunds.into());
    }
    if !self_transfer {
        source_account.base.delegated_amount = delegated_amount
            .checked_sub(amount)
            .ok_or(TokenError::Overflow)?
            .into();
        if u64::from(source_account.base.delegated_amount) == 0 {
            source_account.base.delegate = PodCOption::none();
        }
    }
}
```

Resolution

This was fixed by adding a check for CPI guard, ensuring the delegatee is not to the owner while the transfer is called via a CPI.

```
if delegate == source_account.base.owner
    && cpi_guard.lock_cpi.into()
    && in_cpi()
{
    return Err(TokenError::CpiGuardTransferBlocked.into());
}
```

The commit with the fix is dc85c9810bb23f852dbdfbd170ec4fafe9420ebb.

[ND-ANZ-IN-01] Mint Resurrection Attacks

Severity	Impact	Affected Component	Status
INFORMATIONAL	Extension bypass	Mint Close extension	Accepted

We found that the new closable mint extension introduces some attack vectors that may create an inconsistent on-chain state by reopening closed mints with different extensions. When you want to use an extension on a mint, you need to initialize the mint with that extension. Most extensions live completely within the mint account, but some extensions store information in token accounts too. Using the closable mint extension, we can close a mint and reinitialize it. The only condition is that the supply is 0. However, a 0 supply does not ensure that no token accounts for this mint exist. We could thus perform the following attack:

1. create mint with closable and non-transferable extensions
2. create ATA account for some program, e.g. a DEX that uses ATA for token vaults (don't mint any tokens. Supply stays at 0)
3. this account now has the non-transferable extension
4. close the mint
5. reopen the mint, now without extensions
6. call the DEX to create a new pool, deposit tokens
7. DEX vault is now non-transferable even though mint is transferable
8. any deposits to the pool would go through but withdraws not

A variation of this attack could be to create a mint with no notable extensions and create some token accounts. Then close the mint and reinitialize it with a transfer hook, or transfer fee extension. The accounts that were created before now can do transfers without calling the transfer hook or paying transfer fees.

processor.rs, lines 1176-1193

```
} else if let Ok(mint) =
    PodStateWithExtensions::<PodMint>::unpack(&source_account_data) {
  let extension = mint.get_extension::<MintCloseAuthority>()?;
  let maybe_authority: Option<Pubkey> = extension.close_authority.into();
  let authority =
maybe_authority.ok_or(TokenError::AuthorityTypeNotSupported)?;
  Self::validate_owner(
      program_id,
      &authority,
      authority_info,
      authority_info_data_len,
      account_info_iter.as_slice(),
  )?;

  if u64::from(mint.base.supply) != 0 {
      return Err(TokenError::MintHasSupply.into());
```

```
    }
} else {
    return Err(ProgramError::UninitializedAccount);
}
```

Resolution

The Anza team is aware of this possibility, while it does pose some risk to users the benefits outweigh the risk introduced.

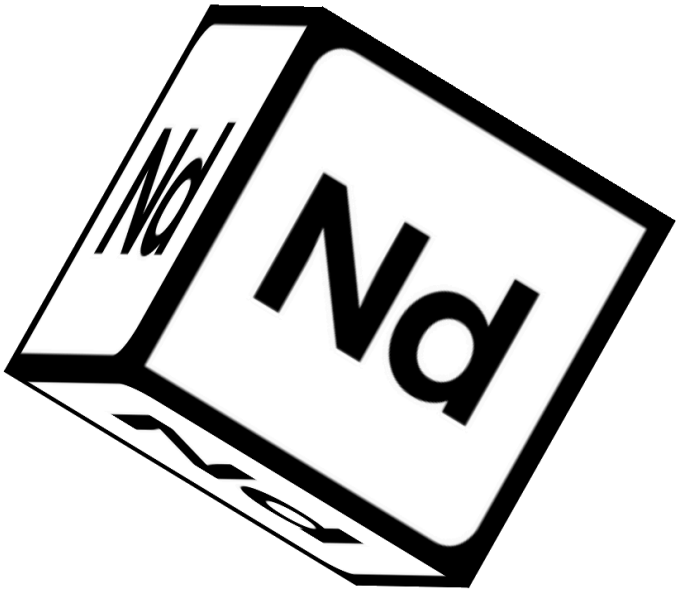
A | About Neodyme

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 worldwide. 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

We are not checklist auditors.

In fact, we pride ourselves on that. 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 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 here.

Select Common Vulnerabilities

Our most common findings are still specific to Solana itself. Among these are vulnerabilities such as the ones listed below:

- Insufficient validation, such as:
  - Missing ownership checks
  - Missing signer checks
  - Signed invocation of unverified programs
  - Account confusions
  - Missing freeze authority checks
  - Insufficient SPL account verification
  - Dangerous user-controlled bumps
  - Insufficient Anchor account linkage
- Account reinitialization vulnerabilities
- Account creation DoS
- Redeployment with cross-instance confusion
- Missing rent exemption assertion
- Casting truncation
- Arithmetic over- or underflows
- Numerical precision and rounding errors
- Anchor pitfalls, such as accounts not being reloaded
- Non-unique seeds
- Issues arising from CPI recursion
- Log truncation vulnerabilities
- Vulnerabilities specific to integration of Token Extensions, for example unexpected external token hook calls

Apart from such Solana-specific findings, some of the most common vulnerabilities relate to the general logical structure of the contract. Specifically, such findings may be:

- Errors in business logic
- Mismatches between contract logic and project specifications
- General denial-of-service attacks
- Sybil attacks
- Incorrect usage of on-chain randomness
- Contract-specific low-level vulnerabilities, such as incorrect account memory management
- Vulnerability to economic attacks
- Allowing front-running or sandwiching attacks

Miscellaneous other findings are also routinely checked for, among them:

- Unsafe design decisions that might lead to vulnerabilities being introduced in the future
    - Additionally, any findings related to code consistency and cleanliness
- Rug pull mechanisms or hidden backdoors

Often, we also examine the authority structure of a contract, investigating their security as well as the impact on contract operations should they be compromised.

Over the years, we have found hundreds of high and critical severity findings, many of which are highly nontrivial and do not fall into the strict categories above. This is why our approach has always gone way beyond simply checking for common vulnerabilities. We believe that the only way to truly secure a program is a deep and tailored exploration that covers all aspects of a program, from small low-level bugs to complex logical vulnerabilities.

C | Vulnerability Severity Rating

We use the following guideline to classify the severity of vulnerabilities. Note that we assess each vulnerability on an individual basis and may deviate from these guidelines in cases where it is well-founded. In such cases, we always provide an explanation.

Severity	Description
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.
INFORMATIONAL	Bugs or inconsistencies that have little to no security impact, but are still noteworthy.

Additionally, we often provide the client with a list of nit-picks, i.e. findings whose severity lies below Informational. In general, these findings are not part of the report.

Neodyme AG

Dirnismaning 55
Halle 13
85748 Garching
Germany

E-Mail: contact@neodyme.io

https://neodyme.io