Accretion

SECURITY ASSESSMENT

Provided by Accretion Labs Pte Ltd. for Realms
May 29, 2025
A25REA1

Realms

AUDITORS

Role	Name
Lead Auditor	Robert Reith (robert@accretion.xyz)
Auditor	Mahdi Rostami (mahdi@accretion.xyz)

CLIENT

Realms (https://realms.today) is the primary DAO governance platorm in the Solana ecosystem. It is used by most Solana DAOs to manage their decision process and provides a complete solution for Solana DAOs including DAO token management, council tokens, different voting configurations, treasury management, and more. Realms engaged Accretion to conduct a security assessment of a PR which implements support for versioned transaction proposals and token extensions.

ENGAGEMENT SCOPE

Realms Program PR 153

Link: https://github.com/Mythic-Project/solana-program-library/pull/153
Commit: 889d5b20164faafa882dfb63d62841e94999969f
ProgramID: GovER5Lthms3bLBqWub97yVrMmEogzX7xNjdXpPPCVZw

ENGAGEMENT TIMELINE

07 May — Project Kickoff
Initial planning and scope definition

08 May — Assessment Begins
Security review and testing phase

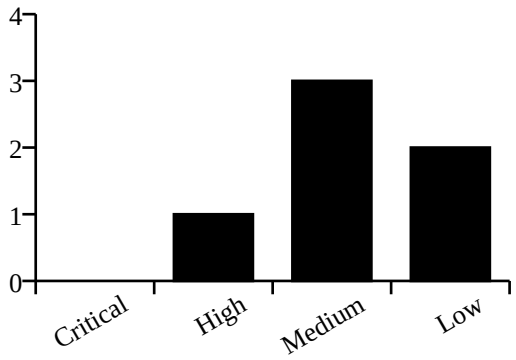
21 May — Review Fixes
Security recommendations are given and implemented

29 May — Project Completion
Report delivery

ASSESSMENT

Our security assessment of Realms' PR revealed a well-architected extension to the Realms DAO program with strong implementation quality. We identified one high-severity issue where the program misinterprets mint instructions as transfers, resulting in inappropriate fee deductions. Additionally, several medium-severity issues were discovered, appearing as isolated incidents without systematic patterns. The Realms development team demonstrated excellent responsiveness and technical expertise in addressing all identified vulnerabilities promptly and effectively. This assessment confirms the overall integrity of the implementation with all issues successfully resolved.

SEVERITY DISTRIBUTION



AUDITED CODE

Program 1

ProgramID: GovER5Lthms3bLBqWub97yVrMmEogzX7xNjdXpPPCVZw
Repository: https://github.com/Mythic-Project/solana-program-library/pull/153
Commit: 78f338c6ac3e54b4a84e216f7fba23cde3560683

ISSUES SUMMARY

ID	TITLE	SEVERITY	STATUS
ACC-H1	Fee Is Incorrectly Deducted in `deposit_governing_tokens` When Using `spl_token_2022` Without Token Transfer	high	fixed
ACC-M1	Potential Type Confusions	medium	fixed
ACC-M2	Invalid Account Index Range Check in `ProposalTransactionMessage::try_from` Allows Faulty Transactions	medium	fixed
ACC-M3	Missing Signer Check for `creator_info` in `process_extend_transaction_buffer` Allows Unauthorized Buffer Extend	medium	fixed
ACC-L1	Address Lookup Tables used in unfrozen state	low	fixed
ACC-L2	`process_create_mint_governance` Does Not Transfer SPL2022 Extension Authorities	low	fixed
ACC-I1	`CreateRealm` Enforces Same Token Standard for Both Community and Council Mints	info	fixed
ACC-I2	Missing Signer Check for Payer in `process_create_transaction_buffer` Allows Unauthorized Buffer Creation	info	fixed

DETAILED ISSUES

ID	ACC-H1
Title	Fee Is Incorrectly Deducted in `deposit_governing_tokens` When Using `spl_token_2022` Without Token Transfer
Severity	high
Status	fixed

Description

We found that in `deposit_governing_tokens`, the program checks `is_token_2022`, and if true, it uses `spl_token_2022::id()` and adds an extra account: `governing_token_mint`. Later, in `process_deposit_governing_tokens`, the deposited amount is calculated by checking for the optional account `expected_mint_info`. If this account exists, the function reduces the amount by a fee.

The issue is that a user might use `spl_token_2022` to **mint** tokens rather than transfer them. In such a case, the logic incorrectly treats the operation as a transfer and applies a fee, even though no actual token transfer occurred. This results in a wrongful fee deduction from the deposited amount.

Location

https://github.com/Mythic-Project/solana-program-library/blob/889d5b20164faafa882dfb63d62841e94999969f/governance/program/src/instruction.rs#L922-L930

https://github.com/Mythic-Project/solana-program-library/blob/889d5b20164faafa882dfb63d62841e94999969f/governance/program/src/instruction.rs#L934-L936

https://github.com/Mythic-Project/solana-program-library/blob/889d5b20164faafa882dfb63d62841e94999969f/governance/program/src/processor/process_deposit_governing_tokens.rs#L122-L130

Relevant Code

```
AccountMeta::new_readonly(
            if is_token_2022 {
                spl_token_2022::id()
            } else {
                inline_spl_token::id()
            },
            false,
        ),
        AccountMeta::new_readonly(realm_config_address, false),
```

```
    if is_token_2022 {
        accounts.push(AccountMeta::new(*governing_token_mint, false));
    };
```

```
    let deposit_amount = match expected_mint_info {
        Some(mint_info) => {
            amount.checked_sub(get_current_mint_fee(mint_info, amount)?)
            .ok_or(GovernanceError::MathematicalOverflow)?
        }
        _ => {
            amount
        }
```

```
    };
```

Mitigation Suggestion

In `process_deposit_governing_tokens`, when calculating the `deposit_amount`, ensure that both `expected_mint_info` is present and `governing_token_source_info` is a valid token account (using `is_spl_token_account`), not a mint account.

```
let fee = if is_spl_token_account(governing_token_source_info) {
    match expected_mint_info {
        Some(mint_info) => get_current_mint_fee(mint_info, amount)?,
        None => 0,
    }
} else {
    0
};

let deposit_amount = amount
    .checked_sub(fee)
    .ok_or(GovernanceError::MathematicalOverflow)?;
```

Remediation

Fixed in commit `18c5da552c397d1ad4df616b4b6b6f808f28cf03`.

ID	ACC-M1
Title	Potential Type Confusions
Severity	medium
Status	fixed

Description

We noticed that you employ the following pattern in multiple cases throughout the code:

https://github.com/Mythic-Project/solana-program-library/blob/889d5b20164faafa882dfb63d62841e94999969f/governance/program/src/state/proposal.rs#L1200-L1200

```
/// proposal.rs L1200-L1200
    get_account_data::<ProposalV2>(program_id, proposal_info)
```

In this pattern you check the discriminator for V1 of some account, here ProposalV1 in an if statement, and then assume in the negative case that the account has the V2 type without checking the discriminator. This may lead to type confusion issues, and in this case may be well exploitable due to multiple variable length account types.

ID	ACC-M2
Title	Invalid Account Index Range Check in `ProposalTransactionMessage::try_from` Allows Faulty Transactions
Severity	medium
Status	fixed

Description

We found that in `ProposalTransactionMessage::try_from`, the validation logic incorrectly checks account indexes against the total number of accounts. The valid range for account indexes is from `0` to `len(accounts) - 1`. However, the current implementation does not account for this, allowing `len(accounts)` to be treated as a valid index — which is out of bounds.

As a result, malformed transactions with invalid account indexes can be inserted. These transactions will pass validation but later fail during [execution](https://github.com/Mythic-Project/solana-program-library/blob/889d5b20164faafa882dfb63d62841e94999969f/governance/program/src/tools/executable_transaction_message.rs#L274).

Location

https://github.com/Mythic-Project/solana-program-library/blob/889d5b20164faafa882dfb63d62841e94999969f/governance/program/src/state/proposal_versioned_transaction.rs#L217-L228

Relevant Code

```
        // Validate that all program ID indices and account indices are within the
        // bounds of the account keys.
for instruction in &instructions {
            if usize::from(instruction.program_id_index) > num_all_account_keys {
                return Err(GovernanceError::InvalidTransactionMessage.into());
            }
            for account_index in &instruction.account_indexes {
                if usize::from(*account_index) > num_all_account_keys {
                    return Err(GovernanceError::InvalidTransactionMessage.into());
                }
            }
        }
```

Mitigation Suggestion

Update the index range check to ensure all account indexes are strictly less than `accounts.len()`.

```
        for instruction in &instructions {
-            if usize::from(instruction.program_id_index) > num_all_account_keys {
+            if usize::from(instruction.program_id_index) >= num_all_account_keys {
                return Err(GovernanceError::InvalidTransactionMessage.into());
            }
            for account_index in &instruction.account_indexes {
-                if usize::from(*account_index) > num_all_account_keys {
+                if usize::from(*account_index) >= num_all_account_keys {
                    return Err(GovernanceError::InvalidTransactionMessage.into());
                }
            }
        }
```

Remediation

Fixed in commit `48d579cf77808f41331488ca5a09f3ca610e35aa`.

ID	ACC-M3
Title	Missing Signer Check for `creator_info` in `process_extend_transaction_buffer` Allows Unauthorized Buffer Extend
Severity	medium
Status	fixed

Description

We found that in `process_extend_transaction_buffer` there is no check for `creator_info` to be a signer, so as a result anyone could extend another user's Transaction buffer.

Location

https://github.com/Mythic-Project/solana-program-library/blob/889d5b20164faafa882dfb63d62841e94999969f/governance/program/src/processor/proposal_versioned_transactions/process_extend_transaction_buffer.rs#L38

Relevant Code

```
let creator_info = next_account_info(account_info_iter)?; // 3
```

Mitigation Suggestion

Explicitly check that the `creator_info` is a signer before proceeding.

Remediation

Fixed in commit `03e8fbf0775375c257448b8aa944c7524f0b96bf`.

ID	ACC-L1
Title	Address Lookup Tables used in unfrozen state
Severity	low
Status	fixed

Description

We found that when a ProposalVersionedTransaction is serialized into a proposal, potential ALT accounts aren't verified to be in a frozen state. This means that one can create a proposal with an ALT that hasn't been finalized, reference accounts at indices that haven't been added yet, and then create the proposal and initiate voting. Then, just before execution, an arbitrary account may be pushed into the ALT. This gives the proposer the power to create proposals with changeable accounts after a proposal is accepted.

Location

https://github.com/Mythic-Project/solana-program-library/blob/889d5b20164faafa882dfb63d62841e94999969f/governance/program/src/processor/proposal_versioned_transactions/process_insert_proposal_versioned_transaction.rs#L135-L145

Relevant Code

```
/// process_insert_proposal_versioned_transaction.rs L135-L145
    let proposal_versioned_transaction_data = ProposalVersionedTransaction {
        account_type: GovernanceAccountType::ProposalVersionedTransaction,
        proposal: *proposal_info.key,
        option_index,
        transaction_index,
        executed_at: None,
        execution_index: 0,
        ephemeral_signer_bumps,
        message: transaction_message.try_into()?,
        execution_status: TransactionExecutionStatus::None,
    };
```

Mitigation Suggestion

When serializing a TransactionMessage, enforce that all ALT accounts are passed into the instruction and validate that they are frozen.

Remediation

Fixed in commit `6aac97ab45f3bad508da9a3dba1b751adcc72a4d` by checking if an ALT has been extended after the vote has started.

ID	ACC-L2
Title	`process_create_mint_governance` Does Not Transfer SPL2022 Extension Authorities
Severity	low
Status	fixed

Description

We found that in `process_create_mint_governance`, the implementation does not handle SPL2022 mint extensions when attempting to `transfer_mint_authorities`. Specifically, there are two important extensions that control mint behavior:

• **CloseMint**: Allows the mint to be closed if it is empty. • **PermanentDelegate**: Grants unlimited delegate rights, enabling burning or transferring any tokens under that mint.

Failing to transfer these authorities leaves critical control outside the governance scope, potentially allowing the original authority to bypass governance restrictions.

Location

https://github.com/Mythic-Project/solana-program-library/blob/889d5b20164faafa882dfb63d62841e94999969f/governance/program/src/processor/process_create_mint_governance.rs#L92

Mitigation Suggestion

Ensure that the `CloseMint` and `PermanentDelegate` authorities are also transferred when governance takes over the mint.

Remediation

The instruction has been deprecated in commit `6d621ff3eacf3ed57749c46502da5a0774cacad2`.

ID	ACC-I1
Title	`CreateRealm` Enforces Same Token Standard for Both Community and Council Mints
Severity	info
Status	fixed

Description

We found that the current implementation of `CreateRealm` enforces both the Community Token Mint and Council Token Mint to use the same token program — either SPL or SPL2022. This limitation prevents users from using, for example, an SPL token for the community and an SPL2022 token for the council.

Location

https://github.com/Mythic-Project/solana-program-library/blob/889d5b20164faafa882dfb63d62841e94999969f/governance/program/src/instruction.rs#L59

Mitigation Suggestion

Introduce two separate flags: • `is_token_2022_for_community` • `is_token_2022_for_council`

This allows each mint to independently specify its token program type.

Remediation

Fixed in commit `51416094a23ea15ec77cfabdb23f513bf343f722`.

ID	ACC-I2
Title	Missing Signer Check for Payer in `process_create_transaction_buffer` Allows Unauthorized Buffer Creation
Severity	info
Status	fixed

Description

We found that in `process_create_transaction_buffer`, the function assumes the `payer` is a signer, but there is no explicit check enforcing this. As a result, an attacker could create transaction buffers on behalf of others since the PDA is derived using:

```
['transaction_buffer', proposal, creator, buffer_index]
```

The contract assumes the `payer` will be validated within `create_and_serialize_account_signed`, but the signer check there only triggers if the `proposal_transaction_buffer_info` requires lamports. An attacker could bypass this by sending enough lamports themselves, making the account rent-exempt and avoiding the signer requirement altogether.

There is no impact from the fact that a token owner or their delegate can create a `proposal_transaction_buffer` for their own `proposal` and One of the seeds for the `proposal_transaction_buffer` is the `proposal` itself.

This means that even if an attacker wants to create a `transaction_buffer` for user A, they can only do so for proposals that do not belong to user A and only belong to attacker itself.

As a result, only `governing_token_owner` and `governance_delegate` could attack each other.

Location

https://github.com/Mythic-Project/solana-program-library/blob/889d5b20164faafa882dfb63d62841e94999969f/governance/program/src/instruction.rs#L683-L684

Relevant Code

```
///     * PDA seeds: ['transaction_buffer', proposal, creator, buffer_index]
    ///   5. `[signer]` Payer
```

Mitigation Suggestion

Explicitly check that the `payer` is a signer.

The same issue exists for `beneficiary_info` in `process_close_transaction_buffer`. The same issue exists for `payer_info` in `process_insert_versioned_transaction_from_buffer`.

Remediation

Fixed in commit `8f3a0c78a2d256122273917b4836a345400910c3`, `4637cfa1673e266e0a4de57b25a60c07fdb070c4`, and `03e8fbf0775375c257448b8aa944c7524f0b96bf`.

APPENDIX

Vulnerability Classification

We rate our issues according to the following scale. Informational issues are reported informally to the developers and are not included in this report.

Severity	Description
Critical	Vulnerabilities that can be easily exploited and result in loss of user funds, or directly violate the protocol's integrity. Immediate action is required.
High	Vulnerabilities that can lead to loss of user funds under non-trivial preconditions, loss of fees, or permanent denial of service that requires a program upgrade. These issues require attention and should be resolved in the short term.
Medium	Vulnerabilities that may be more difficult to exploit but could still lead to some compromise of the system's functionality. For example, partial denial of service attacks, or such attacks that do not require a program upgrade to resolve, but may require manual intervention. These issues should be addressed as part of the normal development cycle.
Low	Vulnerabilities that have a minimal impact on the system's operations and can be fixed over time. These issues may include inconsistencies in state, or require such high capital investments that they are not exploitable profitably.
Informational	Findings that do not pose an immediate risk but could affect the system's efficiency, maintainability, or best practices.

Audit Methodology

Accretion is a boutique security auditor specializing in Solana's ecosystem. We employ a customized approach for each client, strategically allocating our resources to maximize code review effectiveness. Our auditors dedicate substantial time to developing a comprehensive understanding of each program under review, examining design decisions, expected and edge-case behaviors, invariants, optimizations, and data structures, while meticulously verifying mathematical correctness—all within the context of the developers' intentions.

Our audit scope extends beyond on-chain components to include associated infrastructure, such as user interfaces and supporting systems. Every audit encompasses both a holistic protocol design review and detailed line-by-line code analysis.

During our assessment, we focus on identifying:
• Solana-specific vulnerabilities
• Access control issues
• Arithmetic errors and precision loss
• Race conditions and MEV opportunities
• Logic errors and edge cases
• Performance optimization opportunities
• Invariant violations
• Account confusion vulnerabilities
• Authority check omissions
• Token22 implementation risks and SPL-related pitfalls
• Deviations from best practices

Our approach transcends conventional vulnerability classifications. We continuously conduct ecosystem-wide security research to identify and mitigate emerging threat vectors, ensuring our audits remain at the forefront of Solana security practices.