Adevar Labs

Prepared for:

M0 Foundation

July 02, 2025

Audit Report

Version: Audit report after fix review

Table of Contents

1. Executive Summary

About M0 Extensions

The M0 Extensions protocol enables the creation of yield-bearing wrapped tokens based on the M token. The system consists of two main programs: **m_ext** for managing individual extension tokens with customizable yield distribution, and **ext_swap** for facilitating swaps between different extension tokens. The protocol implements sophisticated yield accrual mechanisms with admin-configurable fees.

Audit Summary

- Number of Findings: 6 total
- Critical: 1
- High: 0
- Medium: 3
- Low: 2
- Number of Enhancement Opportunities: 8

Risk Profile

The following table summarizes the distribution of identified vulnerabilities by risk level:

Risk Level	Count	Fixed	Acknowledged
Critical	1	1	0
High	0	0	0
Medium	3	3	0
Low	3	1	2

Overall Security Posture

The protocol demonstrates solid architectural design with robust mathematical foundations. However, critical access control vulnerabilities and floating-point precision issues pose significant risks. The extensive fuzzing campaign (3M+ executions) validated core invariants remain intact under stress conditions, but identified vulnerabilities require immediate attention before production deployment.

Our analysis included comprehensive mathematical modeling of the **powf** approximation (detailed in Appendix A) and and a fuzzing campaign on a set of 6 core properties we extracted during the manual audit (detailed in Appendix B).

Key Findings Highlight

- **Missing Admin Access Control** in **ext_swap** whitelist operations allows unauthorized modifications
- **Floating-Point Precision Issues** in yield calculations can cause temporary insolvency conditions
- **Retroactive Fee Application** may result in unexpected user losses
- **Front-Running Vulnerability** during initialization (limited impact)

Launch Recommendations

I. **Mandatory Before Launch:**

- Implement admin access controls for all whitelist operations

- Add prospective fee synchronization

II. **Strongly Recommended:**

- Implement comprehensive unit testing using LiteSVM

- Add same-token swap prevention

- Validate Token2022 extension compatibility

- Replace **powf** with deterministic approximation or relax or remove the solvency check for lamport-level differences

III. **Post-Launch Monitoring:**

- Deploy continuous fuzzing in CI/CD pipeline

- Monitor for arbitrage patterns as extension count grows

- Track precision-related errors in production

Audit Scope

- **Repository:** https://github.com/m0-foundation/solana-extensions
- **Commit Hash:** **fc4d9343961092f31b19440915f64a9e5ab00c3e**
- **Files/Modules in Scope:**
- **programs/m_ext/** - Core extension program
- **programs/ext_swap/** - Token swap program

- **Repository:** https://github.com/m0-foundation/solana-m
- **Commit Hash:** **8551bcb5f954a59a7cb677cfc72d244941d09c4b**
- **Files/Modules in Scope:**
- **programs/ext_earn/** - Yield distribution (limited review, PR #125)

- **Exclusions:**
- Frontend components
- Off-chain infrastructure
- Third-party dependencies

2. Assumptions and Considerations

Assumptions and Considerations

The following assumptions and considerations should be understood when reviewing this security assessment report:

Audit Assumptions

I. **Out of scope programs**: Several critical security checks are implemented in the **ext_earn** program (part of **solana-m** repo), which was outside the primary audit scope. We assume these checks function correctly:

- **Index Monotonicity**: The new M index is always greater than or equal to the previous index

- **Yield Transfer Validation**: Proper verification of yield transfers and calculations

II. **Protocol Design Assumptions**:

- Yield generation follows expected patterns without negative yields

Centralization Risks

I. **set_m_mint Function**: This function presents a significant centralization risk as it allows the admin to:

- Change the underlying collateral token

- Potentially affect the economic equivalence between tokens

- Impact all users holding the extension token

While some checks are in place (decimals, vault balance), they do not guarantee full economic equivalence or prevent malicious token substitution.

II. **Admin Privileges**: The protocol admin maintains several powerful capabilities:

- Setting and modifying fees (with retroactive effect)

- Whitelisting/delisting extensions and wrap authorities

- Modifying critical protocol parameters

Technical Considerations

I. **Floating-Point Determinism**: The protocol's use of floating-point operations (**powf**) introduces platform-dependent behavior that conflicts with Solana's deterministic execution requirements.

II. **Token2022 Compatibility**: The protocol's compatibility with certain Token2022 extensions (Memo Transfer, Transfer Fee) has not been fully validated and may cause unexpected behavior.

III. **Scalability Limitations**: The current arbitrage protection mechanism (Jito bundles) has a practical limit of approximately 12 extensions before requiring architectural changes.

Risk Acceptance

By deploying this protocol, the team acknowledges:

- Trust assumptions **solana-m - ext_earn program**
- Centralization risks inherent in admin functions
- Limitations of current arbitrage protection mechanisms

3. Severity Definitions

Each issue identified in this report is assigned a severity level based on two dimensions: **Impact** and **Likelihood**. These dimensions help project our team's understanding of both the potential consequences of a vulnerability and how likely a vulnerability is to be discovered and exploited in the real world.

Impact

Impact reflects the potential consequences of the issue–particularly on **project funds**, **user funds**, and the **availability or integrity** of the protocol.

- **High Impact**: Successful exploitation could result in a complete loss of user or protocol funds, disruption of core protocol functionality, or permanent loss of control over critical components.
- **Medium Impact**: Exploitation could cause significant disruption or partial loss of funds, but not a total compromise. May impact some users or non-core functionality.
- **Low Impact**: The issue has minor or negligible consequences. It may affect edge cases, expose metadata, or degrade performance slightly without putting funds or core logic at serious risk.

Likelihood

Likelihood reflects how easy a vulnerability is to discover and how easy it is to exploit by an attacker, but also how economically attractive the exploit is to an attacker.

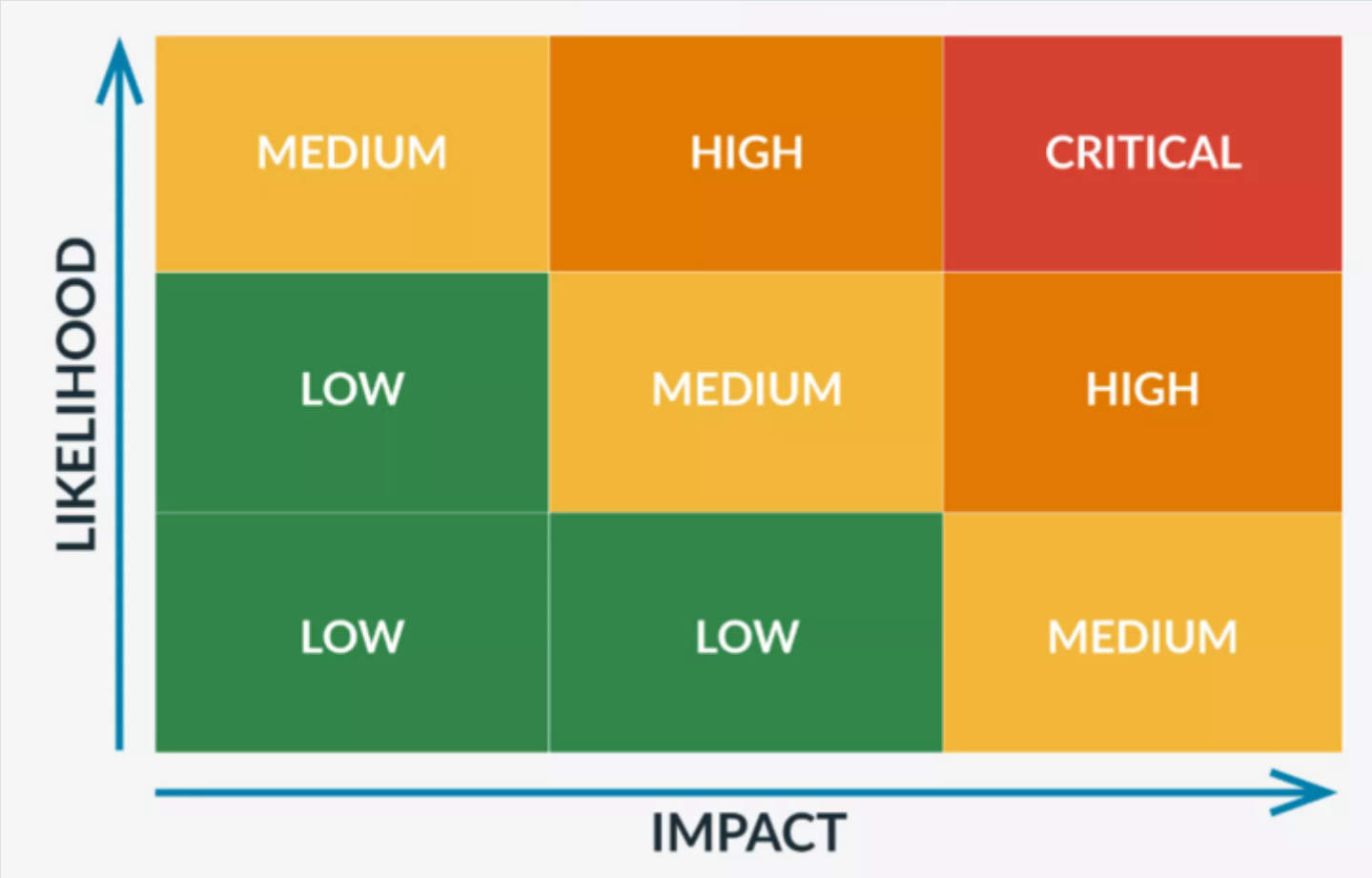
- **High Likelihood**: The vulnerability is trivially discoverable (e.g. via static analysis or public tooling) or by low skilled attackers, and/or it can be exploited by a wide range of actors without privileged access rights.
- **Medium Likelihood**: The issue may require some understanding of the codebase or interaction with specific contracts, but can still be found and exploited with moderate effort.
- **Low Likelihood**: The bug is difficult to detect, requiring deep technical knowledge, specific conditions, or access to privileged information. Exploitation may be impractical or theoretical.

Severity Classification Matrix

By combining **Impact** and **Likelihood**, we assign a severity level using the matrix below:

- **Critical**: High impact + high likelihood (e.g. a bug that could allow anyone to drain protocol funds with minimal effort)
- **High**: High impact with medium likelihood, or medium impact with high likelihood
- **Medium**: Moderate impact and/or discoverability
- **Low**: Minimal impact or unlikely to be exploited

This structured approach helps teams prioritize fixes and mitigate the most dangerous threats first.



Severity Matrix

4. Findings

[Finding #1] [Critical] Missing admin access control for whitelisting operations in ext_swap

Status: Resolved

Impact: High × **Likelihood:** High = **Severity:** Critical

Location: https://github.com/AdevarLabs/solana-extensions/blob/25e29e1c4564b12cb811e9461b44b5305a824ac9/programs/ext_swap/src/instructions/whitelist.rs#L13-L27

>_ whitelist.rs RUST

```rust
    pub admin: Signer<'info>,

    #[account(
        mut,
        seeds = [GLOBAL_SEED],
        bump = swap_global.bump,
        realloc = SwapGlobal::size(
            swap_global.whitelisted_unwrappers.len(),
            swap_global.whitelisted_extensions.len() + 1,
        ),
        realloc::payer = admin,
        realloc::zero = false,
    )]
    pub swap_global: Account<'info, SwapGlobal>,

    pub system_program: Program<'info, System>,
}

impl WhitelistExt<'_> {
```

Description:

The **ext_swap::whitelist** instruction allow M0 admins to whitelist extensions and wrapper/unwrapper pubkeys.
Those whitelist are required during the **swap** instruction for it to execute.
However, there is no access control, effectively allowing anyone to update these whitelists.

The following four functions lack proper admin authorization:
- **WhitelistExt::handler** - Adds extensions to whitelist
- **WhitelistUnwrapper::handler** - Adds unwrappers to whitelist
- **RemoveWhitelistedExt::handler** - Removes extensions from whitelist
- **RemoveWhitelistedUnwrapper::handler** - Removes unwrappers from whitelist

Recommendation:

Add the **has_one = admin @ ExtError::NotAuthorized** Anchor constraints to each **swap_global** accounts

Developer Response:

Fixed.

[Finding #2] [Medium] Retroactive Fee Application

Status: Resolved

Impact: Low × **Likelihood:** High = **Severity:** Medium

Location: https://github.com/AdevarLabs/solana-extensions/blob/main/programs/m_ext/src/instructions/set_fee.rs#L41

>_ set_fee.rs RUST

```
39:         mint::token_program = ext_token_program,
40:     )]
41:     pub ext_mint: InterfaceAccount<'info, Mint>,
42:
43:     /// CHECK: This account is validated by the seed, it stores no data
```

Description:

The **set_fee** function updates the fee but doesn't call **sync_multiplier**, which means accumulated yield gets retroactively affected - The new fee applies to all yield that has accumulated since the last sync. The fee should ideally be applied prospectively by syncing the multiplier first (to lock in the old fee rate for past yield), then updating the fee for future yield.

Recommendation:

A safer implementation would do a **sync_multiplied** first and then set the new fee. This ensures clean separation between old and new fee periods.

Developer Response:

Fixed.

[Finding #3] [Medium] Imprecise multiplier calculus could lead the token to become insolvent

Status: Resolved

Impact: Medium × **Likelihood:** Medium = **Severity:** Medium

Location: https://github.com/AdevarLabs/solana-extensions/blob/main/programs/m_ext/src/utils/conversion.rs#L224-L228

>_ conversion.rs RUST

```
222:
223:         fn calculate_new_multiplier(
224:             last_ext_multiplier: f64,
225:             last_m_multiplier: f64,
226:             new_m_multiplier: f64,
227:             fee_bps: u64,
228:         ) -> Result<f64> {
229:             // Confirm the inputs are in the expected domain.
230:             // These checks ensure that the resultant value is >= 1.0,
```

Description:

There's a corner case when calculating the **last_ext_index** where the value rounds up instead of down, resulting in a (slightly) bigger index than it was supposed to be. This means users could be able to claim a (slightly) better yield than in reality which could, theoretically, result in an insolvent token. The problem occurs because of the usage of float instead of integers in **calculate_new_multiplier**. This round up index would break the solvency check and effectively make the program unusable until either more tokens are deposited as collateral or another index is received.

Recommendation:

One approach is to replace **powf** base calculus, deterministic approximations are to be preferred, like a linear function, or a taylor series approximation (See Annex C). Another approach is to consider a new fee model, where its calculation relies on purely linear function (addressing the base issue). The use of a f64 multiplier seems inevitable, imposed by scaledUi cpi interface, but here it is acceptable to use a (highly precise) approximation since its purpose is purely Ui.

Proof of Concept:

RUST

```rust
// Let's take this input for reference:
let last_ext_index: u64 = 1_000_000_000_001;  // 1e12 + 1
let last_m_index: u64 = 1_000_000_000_002;    // 1e12 + 2
let new_m_index: u64 = 1_000_000_000_003;    // 1e12 + 3
let fee_bps: u64 = 0;

// Let's manually compute what should happen:
// Integer: (1000000000003 * 1000000000001) / 1000000000002
// The multiplier becomes 1.0000000000199995576 which is very close to what WolframAlpha
    returns:
// 1.0000000000019999999999990000000000019999999999960000000000799...
// But when scaling the multiplier back to an index here
ext_global_account.yield_config.last_ext_index = (multiplier * INDEX_SCALE_F64).floor() as u64
    ;

// The value rounds up and becomes: 1000000000002 instead of rounding down to 1000000000001
```

... continued RUST

```
// The effective difference caused by this is $0.2 for a flashloan of $200m. Considering the
    fact that
// there are providers that offer this service with 0% fee (see
// https://docs.marginfi.com/faqs#are-there-fees-on-flashloans), an attacked could do this for
     free
// (with solana tx fee being cheap) and while his motivation might not be to take profit it
    could be to
// grief by breaking the solvency check and putting the program in a stall state. At the same
    time, this
// could happen with usual user activity hence the likelihood is considered high.
```

Developer Response:

Fixed by removing the check and by replacing the last index claimed with the global index.

[Finding #4] [Medium] The swap instruction doesn't explicitly forbid swapping from one token with the same one

Status: Resolved

Impact: Medium × **Likelihood:** Medium = **Severity:** Medium

Location: https://github.com/AdevarLabs/solana-extensions/blob/main/programs/ext_swap/src/instructions/swap.rs#L148-L149

>_ swap.rs RUST

```
146:      */
147:     pub from_token_program: Interface<'info, TokenInterface>,
148:     pub to_token_program: Interface<'info, TokenInterface>,
149:     pub m_token_program: Interface<'info, TokenInterface>,
150:
151:     /*
```

Description:

In the swap instruction, a check is missing that should prevent one user from swapping one token with the same one. The reason why such check would make sense is because if one could swap from one token with the same one, unwrapping + wrapping will happen under the hood and with that, rounding errors might cause the user to lose a small amount of tokens.

Recommendation:

Add a check to make sure the **from** and **to** programs are not the same.

Developer Response:

Fixed.

[Finding #5] [Low] m_ext initialization is vulnerable to front-running

Status: Acknowledged

Impact: Low × **Likelihood:** Low = **Severity:** Low

Location: https://github.com/AdevarLabs/solana-extensions/blob/25e29e1c4564b12cb811e9461b44b5305a824ac9/programs/m_ext/src/instructions/initialize.rs#L173-L183

>_ initialize.rs RUST

```
171:
172:         // Initialize the ExtGlobal account
173:         ctx.accounts.global_account.set_inner(ExtGlobal {
174:             admin: ctx.accounts.admin.key(),
175:             ext_mint: ctx.accounts.ext_mint.key(),
176:             m_mint: ctx.accounts.m_mint.key(),
177:             m_earn_global_account: ctx.accounts.m_earn_global_account.key(),
178:             bump: ctx.bumps.global_account,
179:             m_vault_bump: ctx.bumps.m_vault,
180:             ext_mint_authority_bump: ctx.bumps.ext_mint_authority,
181:             yield_config,
182:             wrap_authorities,
183:         });
184:
185:         // Set the ScaledUi multiplier to 1.0
```

Description:

The **initialize** instruction is potentially vulnerable to front-running during the deployment process. There is a time-window between when the ext_mint token is created and when the **initialize** instruction is called to set up the global account.
As the **admin** parameter in the instruction is not validated against any predetermined deployer address, anyone can call **initialize** and set himself as the **admin** of the **global account** for this program.
This only represent a grieving risk as no funds are at risk.
The deployer will be forced to re-deploy the program and retry the initialization process.

Recommendation:

Make sure to bundle both mint creation and initialization in a single transaction to ensure that the initialization instruction cannot be front-ran.
Alternatively, verify the **admin** Signer against a verified list to ensure only authorized address can call the instruction.

Developer Response:

Noted but not concerned.

[Finding #6] [Low] Misaligned Anchor constraint of source token account in swap instruction

Status: Resolved

Impact: Low × **Likelihood:** Low = **Severity:** Low

Location: https://github.com/AdevarLabs/solana-extensions/blob/main/programs/ext_swap/src/instructions/swap.rs#L75

>_ swap.rs RUST

```
73:         token::mint = from_mint,
74:         token::token_program = from_token_program,
75:     )]
76:     pub from_token_account: Box<InterfaceAccount<'info, TokenAccount>>,
77:     #[account(
```

Description:

In the swap instruction, an anchor misaligned constraint is placed on the **from_token_account**. It specifies that it must be owned by **to_token_program** instead of **from_token_program**.

Recommendation:

Replace **to_token_program** with **from_token_program** in the anchor constraint.

Developer Response:

Fixed.

5. Enhancement Opportunities

[Enhancement #1] Missing check on the new M index to be greater or equal comparing to previous

Status: Acknowledged

Location: https://github.com/AdevarLabs/solana-extensions/blob/main/programs/m_ext/src/utils/conversion.rs#L224-L236

>_ conversion.rs RUST

```
222:
223:         fn calculate_new_multiplier(
224:             last_ext_multiplier: f64,
225:             last_m_multiplier: f64,
226:             new_m_multiplier: f64,
227:             fee_bps: u64,
228:         ) -> Result<f64> {
229:             // Confirm the inputs are in the expected domain.
230:             // These checks ensure that the resultant value is >= 1.0,
231:             // are allowable values to set as the Token2022 Scaled UI multiplier,
232:             // and the ext multiplier is monotonically increasing.
233:             // While having the last ext multiplier <= last m multiplier isn't strictl
        y necessary,
234:             // it arises naturally from our construction and provides a good sanity ch
        eck.
235:             if last_ext_multiplier < 1.0 ||
236:                last_m_multiplier < last_ext_multiplier ||
237:                new_m_multiplier < last_m_multiplier ||
238:                new_m_multiplier > 100.0 || // we set a high, but finite upper bound on
         the multiplier to ensure it (or the other multipliers) don't lead to overflow.
```

Description:

The system blindly trusts that the underlying M protocol's index will always increase but there's no check for this. If, by accident, the index set by ext_earn is smaller than the previous value, this would mean users will lose funds, because it will essentially mean that in between the two indexes the yield was negative, but the system is designed to be positive-yield-only.

Potential Benefit:

Adding a check in this program improves the reliability of it in the sense that if the earn program gets updated and by mistake, a smaller index is pushed, this program will detect the mistake and not distribute yield. It is not a finding because the other program handles this but another check might come in handy in the future.

Recommendation:

After this check: **if latest_m_multiplier == cached_m_multiplier {** add a new check that makes sure the **latest_m_multiplier** is greater or equal than cached_m_multiplier. A max value increase for the index could be checked also to improve resilience. Sanity checks can be made later too on the new vs old multiplier.

Developer Response:

Noted but not concerned (will also be moving away from this index soon).

[Enhancement #2] Consider long term solutions against ext<->ext arbitrage

Status: Acknowledged

Location: General advice

Description:

For the time being, the solution against arbitrage of token yield distribution is to ensure that all M Extensions yield get updated in a **(claimFor + sync)** Jito bundle, making it impossible for an external actor to sandwich to swap in and out two extension, and generate an instant profit off the yield increase.
This solution is viable until the number of M Extension grow to a point it will not fit into a single Jito bundle (limited to 5 transactions).
A transaction can hold up to 35 accounts as of today, while the **claimFor + sync** instructions require 7 accounts each. With 5 transactions, this mean it is possible to fit 12 **claimFor + sync** bundle, or 12 M Extensions.
Once this limit is exceeded, yield updates will need to be separated in two different bundle, opening the possibility to arbitrage yield differences between two extensions.

Potential Benefit:

Preventing M Extension arbitrage to protect users.

Recommendation:

Ensuring that the bundles are executed from the higher yield increase to the lower ones would make the arbitrage operation unprofitable.
Another solution would be to add an unwrapping fee equal to 1 period of yield increase, but this would force legitimate users to keep the token at least for one yield period at least to earn profit.

Developer Response:

Noted that this is an issue but not trying to solve it this cycle.

[Enhancement #3] ext_swap program could use ExtGlobal saved bump for the PDA generation

Status: Resolved

Location: **swap**, **wrap** and **unwrap** instructions in **ext_swap** program

Description:

The **swap**, **wrap**, and **unwrap** instructions in the **ext_swap** program currently derive PDA bumps dynamically for vault and mint authority accounts.
The **ExtGlobal** struct already stores these bump values (**m_vault_bump** and **ext_mint_authority_bump**), but the instructions are not using them.

Potential Benefit:

Reduces compute units by eliminating redundant bump derivation operations during instruction execution.

Recommendation:

Modify the account constraints to use the saved bump values from **ExtGlobal** instead of deriving them dynamically.

Developer Response:

Cannot do this due to an anchor limitation (parsing the ExtGlobal account on ext_swap results in an 'owner' check which isn't dynamic. We could implement a program interface like the token program but would require a program upgrade anytime we add an extension).

[Enhancement #4] Missing Token extensions and configuration validations for set_m_mint instruction

Status: Resolved

Location: https://github.com/AdevarLabs/solana-extensions/blob/25e29e1c4564b12cb811e9461b44b5305a824ac9/programs/m_ext/src/instructions/set_m_mint.rs#L54-L75

>_ set_m_mint.rs RUST

```rust
52: }
53:
54: impl SetMMint<'_> {
55:     // This instruction allows the admin to set a new mint for the m_mint in the global account.
56:     // The new mint must be a valid mint with the same decimals as the existing m_mint.
57:     // Additionally, the new vault ATA for the new mint must contain at least as many tokens
58:     // as the existing vault ATA to ensure the extension remains fully collateralized.
59:     pub fn validate(&self) -> Result<()> {
60:         // Validate that the vault ATA for the new mint contains as many tokens
61:         // as the existing vault ATA so that the extension remains fully collateralized
62:         if self.new_vault_m_token_account.amount < self.vault_m_token_account.amount {
63:             return err!(ExtError::InsufficientCollateral);
64:         }
65:
66:         Ok(())
67:     }
68:
69:     #[access_control(ctx.accounts.validate())]
70:     pub fn handler(ctx: Context<Self>) -> Result<()> {
71:         // Set the new mint
72:         ctx.accounts.global_account.m_mint = ctx.accounts.new_m_mint.key();
73:
74:         Ok(())
75:     }
76: }
```

Description:

the M token, which is used as the underlying token by integrators for their M Extension, is transferred to/from a M vault during wrap and unwrap operations.
The **set_m_mint** instruction could be used either if the M0 M Token gets upgraded, or if integrators wish to leave the M0 ecosystem.

As the **m_ext** program accepts Token2022 to be used for **m_mint**, it is important to acknowledge that some Token 2022 extensions are incompatible with the current implementation:
- Memo Transfer: which require a memo for every transfer (if no memo the tx reverts), but the current implementation do not accept memos
- Transfer Fee, the wrap/unwrap operation do not take the Token2022 extension fee into account when computing the amount to mint/burn. This might result in the draining of the M vault over time.

Potential Benefit:

Correct functioning of the program if a new m_mint is set.

Recommendation:

Add validation in **set_m_mint** to reject tokens with Memo Transfer and Transfer Fee extensions until proper support is implemented.
Alternatively, add a warning to the natspec/documentation for integrators to be aware of the incomptability.

Developer Response:

The **set_m_mint** instruction was removed.

[Enhancement #5] Introduce sanity checks on amount in wrap/unwrap/swap instructions

Status: Resolved

Location: https://github.com/AdevarLabs/solana-extensions/blob/main/programs/ext_swap/src/instructions/swap.rs#L164

>_ swap.rs RUST

```rust
162:
163: impl<'info> Swap<'info> {
164:     fn validate(
165:         &self,
166:         remaining_accounts: &[AccountInfo<'_>],
```

Description:

Both swap and ext programs lack of sanity checks on **amount**. The program could make an early exit when:**amount == 0** or **amount > owned_amount**. This check can help avoid future possible mistakes, for example when one does a CPI call that usually ends with **)?;** but by mistake the question mark is not included, **);**.

Potential Benefit:

Adding those sanity checks would ensure an early exit for transactions with invalid parameters hence saving some gas. The program would have failed and exited anyway, when a transfer of 0 amount would have inevitably taken place but it would have consumed more CU at that point and failing might cause user annoyance.

Recommendation:

Add sanity checks for input parameter **amount**.

Developer Response:

Added checks on part of the instructions (as for validating amount, we want to let the ext program do that as amounts are specified in $M and require converting, we don't want to do that in both programs).

[Enhancement #6] ClaimFees instruction should not be included in no-yield feature build

Status: Resolved

Location:
https://github.com/AdevarLabs/solana-extensions/blob/main/programs/m_ext/src/lib.rs#L82

>_ lib.rs RUST

```rust
80:     // Wrap authority instructions
81:
82:     pub fn wrap(ctx: Context<Wrap>, amount: u64) -> Result<()> {
83:         Wrap::handler(ctx, amount)
84:     }
```

Description:

The fees are taken from the yield it makes so **ClaimFees** instruction should not be part of thethe **no-yield** build

Potential Benefit:

Reducing the program size and potential attack surface.

Recommendation:

Add the instruction to the build only when scaledUi is enabled,**#[cfg(feature = "scaled-ui")]**, similar to how **SetFee** instruction is handled

Developer Response:

no-yield actually means 100% fee so it makes sense to keep the function.

[Enhancement #7] Setting a new mint doesn't guarantee economic equivalence between old/new tokens

Status: Resolved

Location: https://github.com/AdevarLabs/solana-extensions/blob/25e29e1c4564b12cb811e9461b44b5305a824ac9/programs/m_ext/src/instructions/set_m_mint.rs#L55-L66

>_ set_m_mint.rs RUST

```rust

impl SetMMint<'_> {
    // This instruction allows the admin to set a new mint for the m_mint in the global account.
    // The new mint must be a valid mint with the same decimals as the existing m_mint.
    // Additionally, the new vault ATA for the new mint must contain at least as many tokens
    // as the existing vault ATA to ensure the extension remains fully collateralized.
    pub fn validate(&self) -> Result<()> {
        // Validate that the vault ATA for the new mint contains as many tokens
        // as the existing vault ATA so that the extension remains fully collateralized
        if self.new_vault_m_token_account.amount < self.vault_m_token_account.amount {
            return err!(ExtError::InsufficientCollateral);
        }

        Ok(())
    }

```

Description:

The checks in **set_m_mint** do not ensure the extension remains fully collateralized after the change, as the comments explicitly state. They check the amount of tokens that are in the new vault and the decimals of the token, but they must be considered best effort checks, other factors like supply and price are left unchecked. Hence, although the present checks are a nice to have, they do not guarantee economic equivalence between the old/new tokens. It is also worth noting that this function comes a centralization risk.

Potential Benefit:

Acknowledging that the current checks do not ensure full collateralization might open up the possibility of either adding more checks or just updating documentation that there is a centralization risk attached to the function and that it must be used with great care.

Recommendation:

sync_multiplier could be called to distribute all yield that has accumulated since the last sync, yield that came from the old mint. This ensures clean separation between old and new token yield periods. Alternatively, you can check all of the other properties (supply, price, etc) to ensure a better matching between collateral tokens.

Developer Response:

The **set_m_mint** instruction was removed.

[Enhancement #8] Decoding error in RemoveWrapAuthority

Status: Acknowledged

Location: https://github.com/m0-foundation/solana-m/blob/develop/programs/ext_earn/src/instructions/admin/remove_wrap_authority.rs#L18

>_ remove_wrap_authority.rs RUST

```
16:         has_one = admin @ ExtError::NotAuthorized,
17:     )]
18:     pub global_account: Account<'info, ExtGlobal>,
19: }
20:
```

Description:

The PR introduces a new structure for the **ExtGlobal** account. The **AddWrapAuthority** instruction, besides adding a new wrap authority, it migrates the **ExtGlobal** from the old structure to the new structure. The problem is that **RemoveWrapAuthority** assumes the **ExtGlobal** structure has already been migrated.

Potential Benefit:

Although logical from a point of view, you have to add before you remove, a call to **RemoveWrapAuthority** by mistake before adding any wrap authorities will result in a decoding error.

Recommendation:

Make a function that just migrates the account from the old structure to the new structure and make it the first thing that is called before any decoding / parsing of memory.

Developer Response:

Not concerned.

About Us

Adevar Labs is a boutique blockchain security firm specializing in web3 audits.

Built by a mix of experienced professionals in traditional security and crypto natives who have contributed to some of the most critical projects in blockchain infrastructure.

Our team's background spans companies like Bitdefender, Asymmetric Research, Quantstamp, Chainproof, and Juicebox, and includes experience securing smart contracts, bridges, and L1 and L2 protocols across ecosystems like Solana, Ethereum, Polkadot, Cosmos, and MultiversX.

With over 100 audits completed and a portfolio that includes custom fuzzers, exploit modeling, and runtime testing frameworks, Adevar Labs brings both depth and precision to every engagement.

Our auditors have discovered critical vulnerabilities, built high-impact tooling, and placed in top positions in premier audit competitions including Code4rena and Sherlock.

Team members hold distinctions such as PhDs in software protection, and key roles at Fortune 500 companies, and leadership of flagship conferences like ETH Bucharest.

With team members having publications with over 1,100 academic citations and trusted by projects securing over $500M in on-chain value, Adevar Labs blends elite technical rigor with real-world security impact.

We also collaborate with some of the best independent security researchers in the web3 space.

Projects may optionally request specific contributors to be part of their audit.

Audit Methodology

Our audit methodology is specialized to provide thorough security assessments of Solana programs. We focus explicitly on rigorous manual analysis, detailed threat modeling, and careful validation of implemented fixes to ensure your programs operate securely and as intended.

1. Program Context and Architecture Analysis

Our auditors begin by deeply examining your Solana program's documentation, intended functionality, and account design. We meticulously map out program interactions, instruction processing flows, and state management logic. Special attention is given to understanding how your program interfaces with critical Solana system programs, such as the SPL Token Program, Stake Program, and System Program. Last but not least we check external integrations with other Solana projects and verify if the inputs and return values are handled properly.

2. Threat Modeling

We conduct targeted threat modeling tailored specifically for Solana's execution environment. We carefully define attacker capabilities and identify potential vulnerabilities that may arise from Solana-specific issues, including but not limited to:

- Unauthorized account data manipulation
- Improper ownership or signer verification
- Misuse of Program-Derived Addresses (PDAs)
- Incorrect use of Cross-Program Invocations (CPI)
- Failure to adequately handle account privileges or account states

- Risks stemming from rent-exemption and account initialization logic

3. In-depth Manual Security Review

Our experienced auditors perform an extensive manual security review of your Rust-based Solana programs. This involves a comprehensive line-by-line inspection of source code, focusing on common Solana vulnerabilities including, but not limited to:

- Missing or insufficient ownership checks
- Inadequate signer checks
- Incorrect handling of CPI calls and invocation privileges
- Arithmetic and integer overflow or underflow errors
- Unsafe deserialization and serialization of account data structures
- Improper token transfers and SPL-token logic issues
- Logic flaws in financial operations or state transitions
- Edge-case handling in instruction input validation
- Potential denial-of-service vectors related to transaction execution and account handling

During this stage, we clearly document any discovered vulnerabilities, including detailed descriptions, precise severity ratings, and recommendations for secure implementation. In situations where it is not clear how the vulnerability might be exploited we may also include a detailed proof-of-concept exploit including code snippets and instructions on how the exploit could be performed.

4. Detailed Fix Review and Validation

After the initial audit and your team's subsequent remediation efforts, we perform a comprehensive fix review to ensure the vulnerabilities identified have been effectively resolved.

We verify each fix individually, confirming that:

- Corrections effectively eliminate the security risks
- Changes do not inadvertently introduce new vulnerabilities or regressions
- The fixes align closely with Solana best practices and secure coding guidelines

Our detailed validation ensures that security improvements are robust, complete, and aligned with best practices specific to the Solana development ecosystem.

Confidentiality Notice

This report, including its content, data, and underlying methodologies, is subject to the confidentiality and feedback provisions in your agreement with Adevar Labs. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Adevar Labs.

Legal Disclaimer

The review and this report are provided by Adevar Labs on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Adevar Labs disclaims all warranties, expressed or implied, in connection with this report, its content, and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

You agree that access to and/or use of the report and other results of the review, including any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided.

You acknowledge that blockchain technology remains under development and is subject to unknown risks and flaws. Adevar Labs does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open-source or third-party software, code, libraries, materials, or information accessible through the report. As with the purchase or use of a product or service in any environment, you should use your best judgment and exercise caution where appropriate.

Appendix A: Approximation of powf usage

Core Concern

During our analysis, we identified a scenario in which the index, used as the base for yield calculations, can round up due to floating-point imprecision, potentially triggering a short-term "insolvency" condition. While the discrepancy is corrected during subsequent updates and does not accumulate over time, it can cause temporary state inconsistencies that may stall program execution.

The root cause lies in the use of floating-point operations, specifically the computation of the form:

multiplier = growth.powf(1 - fee)

This expression introduces non-determinism due to platform-dependent floating-point behavior, which is problematic in a deterministic execution environment like Solana's.

Explored Solutions

We explored the following solutions:

Deterministic Approximation

Replace **growth.powf(1 - fee)** with a deterministic approximation, such as a Taylor series or a piecewise polynomial. The key requirement is that the approximation must:

- Be precise (low relative error), and
- Always underestimate the exact value, to prevent insolvency by over-crediting.

Redesign Fee Structure

Avoid exponentiation altogether by adopting a linear or logarithmic fee model. While mathematically less elegant than exponential decay, this eliminates floating-point operations entirely and ensures deterministic execution.

Relax Solvency Check

Loosen or remove the current solvency check that stalls execution when rounding introduces small imbalances. Since discrepancies occur at the lamport level and self-correct, this could be a safe tradeoff in practice, though it does redefine what "insolvency" means in this context.

Taylor Series Approximation

We evaluated the feasibility of approximating **x^y** (i.e., **growth.powf(1 - fee)**) using a truncated Taylor series. This approach is particularly effective when:

- The exponent **y = 1 - fee** lies within the range (0, 1]

- The base x (the growth factor) lies in the range [1, 1.1], corresponding to a 0-10% yield range

Observations

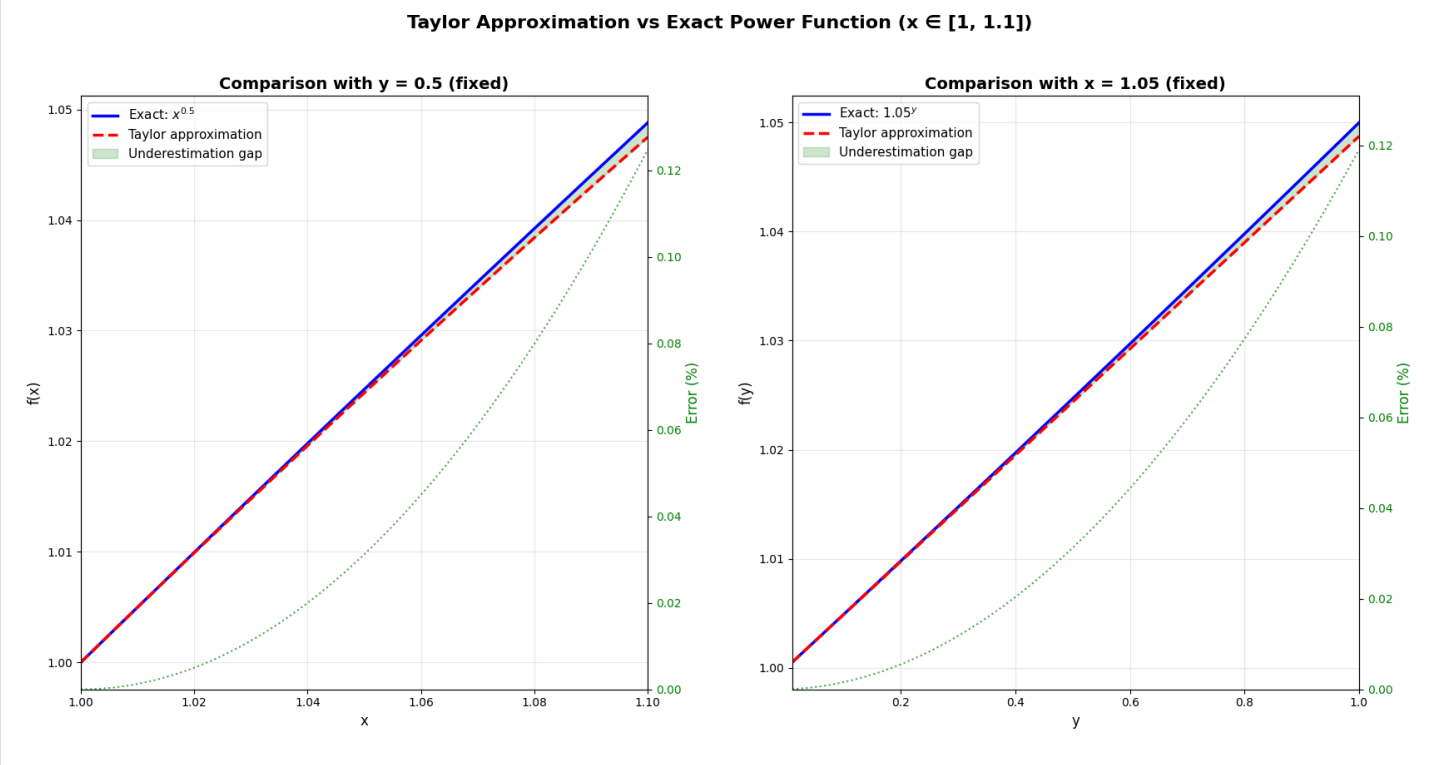
- The Taylor series always underestimates the true value of x^y within the tested domain
- The function is safe to use in a deterministic Solana program
- Error characteristics for the domain x lies in [1, 1.1] and y lies in [0.01, 1]:

- Maximum error: 0.454545%

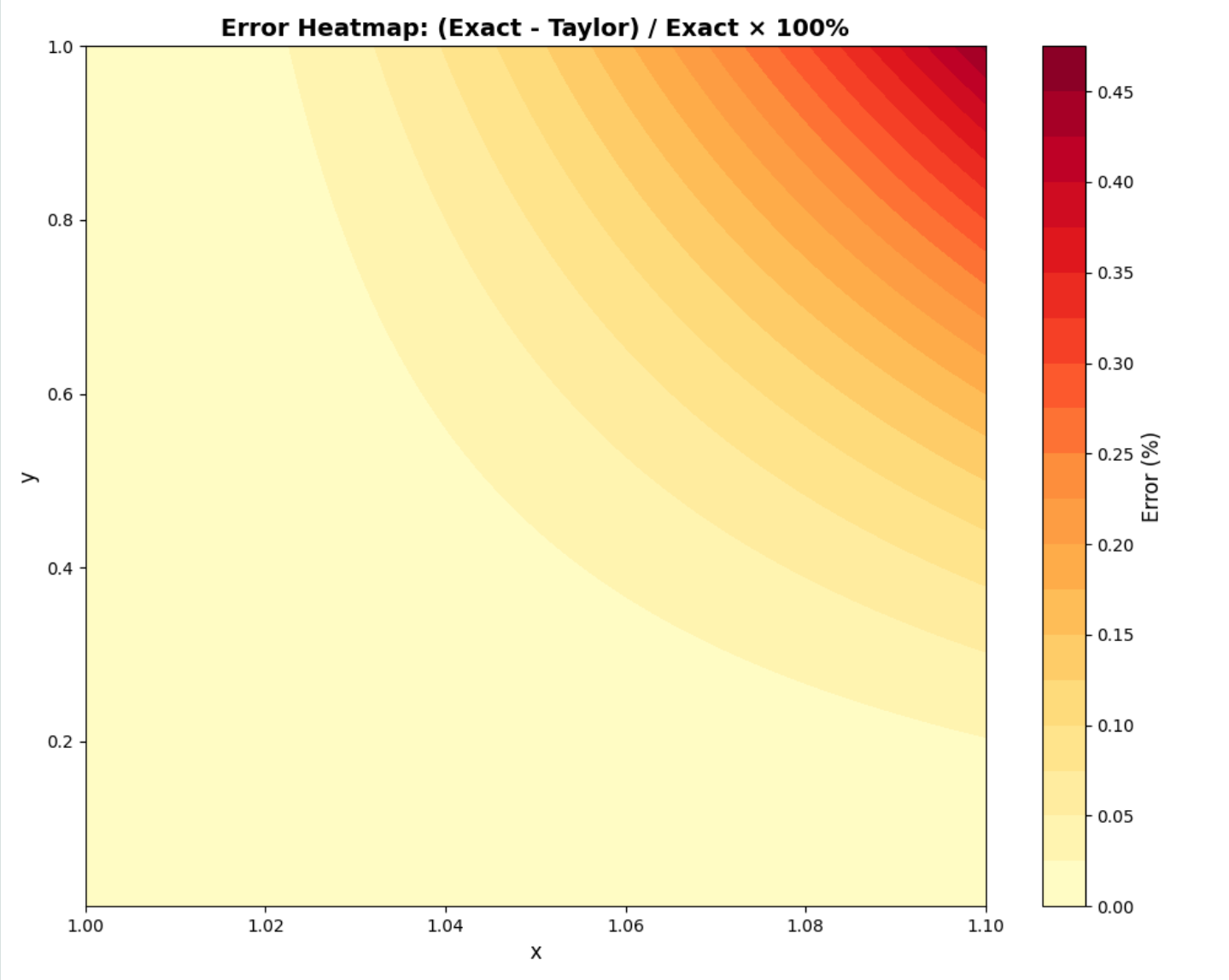
- Mean error: 0.054187%

- Accuracy increases sharply as x → 1

Visualizations



Taylor Approximation vs Exact Power Function



Error Heatmap

Appendix B: Fuzzing Campaign Analysis for m_ext program

Introduction

This section presents a comprehensive analysis of an extensive fuzzing campaign conducted on the m_ext program as part of the security audit. The campaign utilized advanced coverage-guided fuzzing techniques to systematically test the program's resilience against realistic and edge cases. During our manual audit we identified the code invariants and properties presented below and using a custom harness we verified if the fuzzer can find any inputs that can violate these invariants/properties.

Fuzzing Strategy and Implementation

Action Space Definition

The fuzzer targets seven distinct protocol actions:

Action	Description	Input Parameters	Complexity
Wrap	Convert M tokens to EXT tokens	User index, Amount	Medium
Unwrap	Convert EXT tokens to M tokens	User index, Amount	Medium
WrapUnwrap	Wrap followed by unwrap	User index, Amount	High
UpdateLastClaimIndex	Modify yield distribution index	New index delta	Low
ClaimFees	Admin fee collection	None	Low
DoubleClaimFees	Consecutive fee claims	None	Medium
UnwrapAllForAllUsers	Batch unwrap operation	None	High

Input Generation Strategy

```rust
// Input byte structure:
// Byte 0: Action selector (0-6)
// Bytes 1-8: Action-specific parameters
//
// Example: Wrap action
// [0x00, 0x01, 0xE8, 0x03, 0x00, 0x00, 0x00, 0x00, 0x00]
//   ^     ^    ^------ Amount
//   |     +--- User index: 1
//   +--- Action: Wrap (0)
```

Fuzzing Events and Monitoring

Event Types Tracked

Event Type	Purpose	Key Metrics
Wrap	Token conversion tracking	M amount, EXT amount, multiplier
Unwrap	Reverse conversion tracking	EXT unwrapped, M received, fees
WrapUnwrap	Round-trip consistency	Initial/final balances
UpdateLastClaimIndex	Yield distribution	Old/new index, yield amount
ClaimFees	Fee collection	Admin balance, fees claimed
DoubleClaimFees	Fee collection	First/second claim amounts
UnwrapAllForAllUsers	Batch operation safety	User count, total unwrapped

Invariant Testing

Critical Invariants Monitored

Invariant 1: Vault Solvency

```
vault_balance >= principal_to_amount_down(total_ext_supply, current_multiplier) - 2
```

Purpose: Ensures the vault always maintains sufficient M tokens to back all outstanding EXT tokens at the current exchange rate.

Invariant 2: Index Monotonicity

```
current_index >= INITIAL_INDEX (1e12)
```

Purpose: Prevents index manipulation that could result in negative yields or protocol exploitation.

Invariant 3: Fee Claim Idempotency

```
second_claim_amount == 0 (for consecutive claims)
```

Purpose: Ensures fees cannot be claimed multiple times for the same period.

Invariant 4: Conservation of Value

```
sum(user_m_tokens) + vault_m_tokens == initial_total_m_supply + minted_yield
```

Purpose: Verifies no tokens are created or destroyed outside of authorized mint operations.

Invariant Checking Implementation

```rust
pub fn check_basic_invariants(&mut self) -> Result<(), String> {
    // Vault solvency check
    let vault_balance = self.get_token_balance(&self.vault_m_token_account)?;
    let total_ext_supply = self.get_total_ext_supply()?;
    let multiplier = self.get_ext_multiplier()?;

    let required_backing = Self::principal_to_amount_down(total_ext_supply, multiplier)?;
    if vault_balance < required_backing - 2 {
        return Err(format!("CRITICAL: Vault undercollateralized"));
    }

    // Additional invariant checks...
    Ok(())
}
```

RUST

Results and Findings

1. **Invariant Preservation:** All critical invariants maintained across 3M executions
2. **Mathematical Soundness:** No arithmetic errors or precision loss vulnerabilities
3. **State Machine Integrity:** Valid state transitions only

Recommendations

Despite the positive results, we recommend:

1. **Testing Methodology:** Create a suite of unit tests using LiteSVM.
2. **Continuous Fuzzing:** Integrate fuzzing into CI/CD pipeline
3. **Expanded Corpus:** Add more complex multi-user scenarios