// Security Assessment

06.17.2025 - 06.23.2025

Solana Extensions

MO

HALBORN

Solana Extensions - M0

Prepared by: HALBORN

Last Updated 07/02/2025

Date of Engagement: June 17th, 2025 - June 23rd, 2025

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
5	0	1	0	1	3

TABLE OF CONTENTS

1. Introduction

`The M0 team` engaged `Halborn` to conduct a security assessment of their **m_ext and ext_swap Solana programs** beginning on June 17, 2025, and ending on June 23, 2025. The security assessment was scoped to the Solana Programs provided in solana-extensions GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

M0 is a universal stablecoin platform that enables developers to create their own application-specific digital dollars and integrate them into any use case.

The scope of this assessment included the following two programs:

- `m_ext` – The canonical implementation of an M Extension on Solana. It allows anyone to deploy and initialize a token as an M extension. The program provides instructions for managing the extension and claiming fees. It includes feature flags to select the type of yield distribution: no yield or yield earned by the underlying $M backing.
- `ext_swap` – Facilitates swapping between M Extensions through wrapping and unwrapping operations. Holders can move between Extensions without fees or price impact, enabling Extensions to share liquidity.

2. Assessment Summary

`Halborn` was provided 1 week for the engagement and assigned one full-time security engineer to review the security of the Solana Programs in scope. The engineer is a blockchain and smart contract security expert with advanced smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the Solana Programs.
- Ensure that smart contract functionality operates as intended.

In summary, `Halborn` identified some improvements to reduce the likelihood and impact of risks, which were mostly addressed by the `M0 team`. The main ones were the following:

- `Ensuring that only a designated authority is allowed to manipulate extensions and unwrappers whitelists.`
- `Implement access control during programs initialization.`

3. Test Approach And Methodology

Halborn performed a combination of a manual review of the source code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the program assessment. While manual testing is recommended to uncover flaws in business logic, processes, and implementation; automated testing techniques help enhance coverage of programs and can quickly identify items that do not follow security best practices.

The following phases and associated tools were used throughout the term of the assessment :

- Research into the architecture, purpose, and use of the platform.

- Manual program source code review to identify business logic issues.

- Mapping out possible attack vectors

- Thorough assessment of safety and usage of critical Rust variables and functions in scope that could lead to arithmetic vulnerabilities.

- Scanning dependencies for known vulnerabilities (`cargo audit).`

- Local runtime testing (`anchor test`)

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4

SEVERITY	SCORE VALUE RANGE
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: solana-extensions

(b) Assessed Commit ID: 25e29e1

(c) Items in scope:

- ./Cargo.toml
- ./Anchor.toml
- ./programs/ext_swap/Cargo.toml
- ./programs/ext_swap/Xargo.toml
- ./programs/ext_swap/src/instructions/unwrap.rs
- ./programs/ext_swap/src/instructions/whitelist.rs
- ./programs/ext_swap/src/instructions/swap.rs
- ./programs/ext_swap/src/instructions/initialize.rs
- ./programs/ext_swap/src/instructions/mod.rs
- ./programs/ext_swap/src/instructions/wrap.rs
- ./programs/ext_swap/src/lib.rs
- ./programs/ext_swap/src/state.rs
- ./programs/ext_swap/src/errors.rs
- ./programs/m_ext/Cargo.toml
- ./programs/m_ext/Xargo.toml
- ./programs/m_ext/src/instructions/unwrap.rs
- ./programs/m_ext/src/instructions/set_m_mint.rs
- ./programs/m_ext/src/instructions/initialize.rs
- ./programs/m_ext/src/instructions/set_fee.rs
- ./programs/m_ext/src/instructions/sync.rs
- ./programs/m_ext/src/instructions/mod.rs
- ./programs/m_ext/src/instructions/wrap.rs
- ./programs/m_ext/src/instructions/claim_fees.rs
- ./programs/m_ext/src/instructions/manage_wrap_authority.rs
- ./programs/m_ext/src/constants.rs
- ./programs/m_ext/src/lib.rs
- ./programs/m_ext/src/utils/conversion.rs
- ./programs/m_ext/src/utils/token.rs
- ./programs/m_ext/src/utils/mod.rs
- ./programs/m_ext/src/state.rs
- ./programs/m_ext/src/errors.rs

Out-of-Scope: Third party dependencies and economic attacks.

REMEDIATION COMMIT ID:

- dc9cf3b
- 51b65b9
- 6258a7c
- d77ff5c

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	1	0	1	3

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
MISSING ACCESS CONTROL ALLOWS WHITELIST MANIPULATION	HIGH	SOLVED - 06/23/2025
RISK OF FRONT-RUNNING DURING PROGRAM INITIALIZATION	LOW	RISK ACCEPTED - 06/25/2025
INCONSISTENT DOCUMENTATION	INFORMATIONAL	SOLVED - 06/25/2025
PASSING UNNECESSARY SYSTEM PROGRAM ACCOUNT	INFORMATIONAL	SOLVED - 06/24/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
INCORRECT TOKEN PROGRAM CONSTRAINT	INFORMATIONAL	SOLVED - 06/23/2025

7. FINDINGS & TECH DETAILS

7.1 MISSING ACCESS CONTROL ALLOWS WHITELIST MANIPULATION

// HIGH

Description

The `ext_swap` program provides four instructions — `whitelist_extension`, `remove_whitelisted_extension`, `whitelist_unwrapper`, and `remove_whitelisted_unwrapper` — that are used to manage a whitelist of approved extensions and users with unwrap permissions. These instructions are intended to be restricted to a designated authority.

However, the program does not enforce this access control, allowing **any user** to invoke these instructions and modify the whitelist.

This missing validation effectively defeats the purpose of the whitelist, as it allows unauthorized users to arbitrarily add or remove extensions and unwrappers, bypassing intended program restrictions and undermining its security model.

whitelist.rs

```
8  #[derive(Accounts)]
9  pub struct WhitelistExt<'info> {
10     #[account(mut)]
11     pub admin: Signer<'info>,
12
13     #[account(
14         mut,
15         seeds = [GLOBAL_SEED],
16         bump = swap_global.bump,
17         realloc = SwapGlobal::size(
18             swap_global.whitelisted_unwrappers.len(),
19             swap_global.whitelisted_extensions.len() + 1,
20         ),
21         realloc::payer = admin,
22         realloc::zero = false,
23     )]
24     pub swap_global: Account<'info, SwapGlobal>,
25
26     pub system_program: Program<'info, System>,
27 }
```

Proof of Concept

Invoke the `whitelist_extension` instruction and sign it by an arbitrary user.

```
it("add to ext whitelist", async () => {
      await sendTransaction(
        program.methods
          .whitelistExtension(earn.programId)
          .accounts({
```

```
            admin: swapper.publicKey, // incorrect admin
          })
          .transaction(),
        [swapper] // incorrect admin
      );

      const { whitelistedExtensions } = await program.account.swapGlobal.fetch(
        PublicKey.findProgramAddressSync(
          [Buffer.from("global")],
          program.programId
        )[0]
      );

      // Validate the extension was added
      expect(whitelistedExtensions).toHaveLength(1);
      expect(whitelistedExtensions[0].toBase58()).toBe(
        earn.programId.toBase58()
      );
    });
```

The test passes when it is expected to fail:

```
PASS tests/unit/ext_swap.test.ts
  extension swap tests
    initialize swap programs
      ✓ create mints (38 ms)
      ✓ initialize earn program (31 ms)
      ✓ initialize extension programs (15 ms)
    configure
      ✓ initialize config (3 ms)
      ✓ re-initialize config revert (2 ms)
      ✓ add to ext whitelist (4 ms)
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:H/D:N/Y:N (7.5)

Recommendation

To address this issue, it is recommended to implement proper access control for all four instructions, ensuring that only a designated authority is allowed to invoke them.

Remediation Comment

SOLVED: The **MO team** resolved this finding by correctly implementing access control and verifying that only an authorized administrator can manage the whitelists.

Remediation Hash

https://github.com/m0-foundation/solana-extensions/commit/dc9cf3b4bb83b6aaabb1aa27b442259b35771e45

7.2 RISK OF FRONT-RUNNING DURING PROGRAM INITIALIZATION

// LOW

Description

The `m_ext::initialize` and `ext_swap::initialize` instructions are designed to initialize the respective programs and configure their settings.

However, the instructions currently lack proper access control—they do not restrict or validate the signer. As a result, any user can invoke it, creating a risk that a malicious actor could front-run the intended administrator and initialize the program with unauthorized settings.

m_ext/src/instructions/initialize.rs

```
29  #[derive(Accounts)]
30  #[instruction(wrap_authorities: Vec<Pubkey>)]
31  pub struct Initialize<'info> {
32      #[account(mut)]
33      pub admin: Signer<'info>,
34
35      #[account(
36          init,
37          payer = admin,
38          space = ExtGlobal::size(
39              wrap_authorities.len()
40          ),
41          seeds = [EXT_GLOBAL_SEED],
42          bump
43      )]
44      pub global_account: Account<'info, ExtGlobal>,
```

m_swap/src/instructions/initialize.rs

```
5   #[derive(Accounts)]
6   pub struct InitializeGlobal<'info> {
7       #[account(mut)]
8       pub admin: Signer<'info>,
9
10      #[account(
11          init,
12          payer = admin,
13          space = SwapGlobal::size(0,0),
14          seeds = [GLOBAL_SEED],
15          bump,
16      )]
17      pub swap_global: Account<'info, SwapGlobal>,
18
19      pub system_program: Program<'info, System>,
20  }
```

BVSS

AO:A/AC:L/AX:L/R:P/S:U/C:N/A:M/I:L/D:L/Y:N (3.1)

Recommendation

To address this issue, it is recommended to implement a proper access control and validate that the signer is a known authorized address.

Remediation Comment

RISK ACCEPTED: The **MO team** acknowledged this finding but has decided to keep the current implementation.

7.3 INCONSISTENT DOCUMENTATION

// INFORMATIONAL

Description

Both programs are well documented. However, there are several inconsistencies between the provided documentation and the actual implementation:

- In the provided Notion documentation, the description of the amount argument in the `m_ext` 's program `unwrap` instruction says that it is the ""ui amount" of $EXT tokens to unwrap into $M", however in the code, it is the other way around being it the maximal $M amount used to calculate the amount of $EXT tokens to unwrap.
- The code comment in the `m_ext` 's program `wrap` instruction on line 85 states, that "all extensions may not be token2022". However currently all extension mints are required to be token2022.
- The code comment in the `m_ext` 's program `set_fee` instruction on line 26 states, that "The fee must be between 0 and 100 bps (inclusive).". However the fee can be between 0 and 10000 bps (inclusive).

wrap.rs

```
85 | // we have duplicate entries for the token2022 program since the interface needs to be consistent, an
86 | // additionally, it allows us to change the m token program in the future without breaking the interf
87 | pub m_token_program: Program<'info, Token2022>,
```

set_fee.rs

```
24 | impl SetFee<'_> {
25 |     // This instruction allows the admin to set a new fee in basis points (bps).
26 |     // The fee must be between 0 and 100 bps (inclusive).
27 |     // If the fee is set to 0, it effectively disables the fee.
28 |     // If the fee is set to 100, it means the entire amount is taken as a fee.
29 |     // Any value above 100 bps will result in an error.
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

To address this finding, it is recommended to update the provided documentation and code comments accordingly.

Remediation Comment

SOLVED: The **MO team** solved this finding and correctly updated the documentation and the comments in the provided code.

Remediation Hash

https://github.com/m0-foundation/solana-extensions/commit/51b65b9ba5c28db7f429a7975b4b7f923192fa36

7.4 PASSING UNNECESSARY SYSTEM PROGRAM ACCOUNT

// INFORMATIONAL

Description

The `remove_whitelisted_ext` and `remove_whitelisted_unwrapper` instructions enable an authorized user to remove specific extensions and unwrapper addresses from the whitelist. Both instructions require the system program account to be passed; however, this account is not utilized and is therefore unnecessary.

whitelist.rs

```
pub struct RemoveWhitelistedExt<'info> {
    #[account(mut)]
    pub admin: Signer<'info>,

    #[account(
        mut,
        seeds = [GLOBAL_SEED],
        bump = swap_global.bump,
    )]
    pub swap_global: Account<'info, SwapGlobal>,

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

whitelist.rs

```
pub struct RemoveWhitelistedUnwrapper<'info> {
    #[account(mut)]
    pub admin: Signer<'info>,

    #[account(
        mut,
        seeds = [GLOBAL_SEED],
        bump = swap_global.bump,
    )]
    pub swap_global: Account<'info, SwapGlobal>,

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

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

To address this finding, it is recommended to assess whether these instructions require reallocating the `SwapGlobal` data when removing entries from the whitelists. If reallocation is unnecessary, the system program account should be removed.

Remediation Comment

SOLVED: The **M0 team** resolved this finding by removing the unnecessary system program account.

Remediation Hash

https://github.com/m0-foundation/solana-extensions/commit/6258a7cf900991f2506697d6d4199540b1bf5ea2

7.5 INCORRECT TOKEN PROGRAM CONSTRAINT

// INFORMATIONAL

Description

The `swap` instruction requires the correct token program to be specified for each token mint (`from_mint` , `to_mint` , and `m_mint`), and it verifies that the associated token accounts are owned by the corresponding token program. However, the `from_token_account` is incorrectly validated against the token program of `to_mint` (`to_token_program`) instead of its own associated program.
Currently, this does not cause any issues because both `from_mint` and `to_mint` must be Token-2022. Nevertheless, if support for both the legacy SPL Token program and Token-2022 is introduced in the future, this mismatch could result in unexpected errors.

swap.rs

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

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:N/D:N/Y:N (0.0)

Recommendation

To resolve this issue, it is recommended to validate the `from_token_account` against the `from_token_program` .

Remediation Comment

SOLVED: The **MO team** resolved this finding by validating the `from_token_account` against the correct token program.

Remediation Hash

https://github.com/m0-foundation/solana-extensions/commit/d77ff5c07db11d493810de4275eb6ee36cc372d3

8. AUTOMATED TESTING

Static Analysis Report

Description

Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used was `cargo audit`, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in `https://crates.io` are stored in a repository named The RustSec Advisory Database. `cargo audit` is a human-readable version of the advisory database which performs a scanning on Cargo.lock. Security Detections are only in scope. All vulnerabilities shown here were already disclosed in the above report. However, to better assist the developers maintaining this code, the auditors are including the output with the dependencies tree, and this is included in the cargo audit output to better know the dependencies affected by unmaintained and vulnerable crates.

Cargo Audit Results

ID	CRATE	DESCRIPTION
RUSTSEC-2024-0344	curve25519-dalek	Timing variability in `curve25519-dalek`'s `Scalar29::sub` / `Scalar52::sub`
RUSTSEC-2022-0093	ed25519-dalek	Double Public Key Signing Function Oracle Attack on `ed25519-dalek`

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.