// Security Assessment

02.24.2025 - 03.11.2025

Magic Block - Delegation Program

Magic Block

HALBORN

Magic Block - Delegation Program - Magic Block

Prepared by: HALBORN

Last Updated 04/25/2025

Date of Engagement: February 24th, 2025 - March 11th, 2025

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
5	0	0	0	2	3

TABLE OF CONTENTS

1. Introduction

`Magic Block` engaged `Halborn` to conduct a security assessment on their `Delegation` **Solana program** beginning on **March 10th, 2025**, and ending on **March 28th, 2025**. The security assessment was scoped to the Solana program provided in magicblock-labs/delegation-program GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

This Delegation Program lets Solana smart contracts hand off their PDAs to an off-chain "ephemeral rollup" for fast updates. While delegated, the program enforces ownership locks and tracks changes, then finalizes them on-chain so the mainnet state stays consistent.

It supports creating ephemeral balances (temporary escrow accounts), delegating/undelegating program accounts, committing off-chain data, and finalizing rollup results. Fees are handled through two vaults—one for protocol fees and one for each validator's fees—both governed by an admin or the program upgrade authority.

This design gives high-throughput advantages without compromising on Solana's security model, enabling dApps to scale off-chain while retaining the trust guarantees and atomic settlement of L1.

2. Assessment Summary

Halborn was provided **15 days** for the engagement and assigned one full-time security engineer to review the security of the Solana Program 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 Delegation Solana Program.
- Ensure that the program's functionality operates as intended.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were addressed or acknowledged by the Magic Block team:

- Consider using a multi-signature wallet or multi-step upgradeable authority approach for enhanced resilience. Additionally, store the admin key in a config account that can be updated, rather than hard-coding a Pubkey as a protocol constant.
- Implement robust verification logic in verify.rs (e.g., cryptographic proof checks, signature validations, or protocol invariants). Additionally, remove the TODO placeholder once real checks are in place. Ensure the function fails if the new state data is malformed or violates any business-critical constraints.
- Add an authorization check so only a recognized admin or multi-signature wallet can create the protocol fees vault.

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 (`solana-test-framework`).

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
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
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

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
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
Informational	0 - 1.9

5. SCOPE

FILES AND REPOSITORY

(a) Repository: delegation-program

(b) Assessed Commit ID: 7fc0ae9

(c) Items in scope:

- src/instruction_builder/finalize.rs
- src/instruction_builder/init_validator_fees_vault.rs
- src/instruction_builder/commit_state.rs
- src/instruction_builder/delegate.rs
- src/instruction_builder/protocol_claim_fees.rs
- src/instruction_builder/validator_claim_fees.rs
- src/instruction_builder/close_ephemeral_balance.rs
- src/instruction_builder/commit_state_from_buffer.rs
- src/instruction_builder/whitelist_validator_for_program.rs
- src/instruction_builder/close_validator_fees_vault.rs
- src/instruction_builder/init_protocol_fees_vault.rs
- src/instruction_builder/undelegate.rs
- src/instruction_builder/top_up_ephemeral_balance.rs
- src/instruction_builder/delegate_ephemeral_balance.rs
- src/error.rs
- src/lib.rs
- src/processor/finalize.rs
- src/processor/init_validator_fees_vault.rs
- src/processor/commit_state.rs
- src/processor/delegate.rs
- src/processor/protocol_claim_fees.rs
- src/processor/validator_claim_fees.rs
- src/processor/close_ephemeral_balance.rs
- src/processor/commit_state_from_buffer.rs
- src/processor/utils/loaders.rs
- src/processor/utils/verify.rs
- src/processor/utils/curve.rs
- src/processor/utils/pda.rs
- src/processor/whitelist_validator_for_program.rs
- src/processor/close_validator_fees_vault.rs
- src/processor/init_protocol_fees_vault.rs
- src/processor/undelegate.rs
- src/processor/top_up_ephemeral_balance.rs
- src/processor/delegate_ephemeral_balance.rs
- src/processor/utils/verify.rs
- src/state/delegation_metadata.rs

- src/state/commit_record.rs
- src/state/utils/try_from_bytes.rs
- src/state/utils/to_bytes.rs
- src/state/utils/discriminator.rs
- src/state/program_config.rs
- src/state/delegation_record.rs
- src/discriminator.rs
- src/args/commit_state.rs
- src/args/delegate.rs
- src/args/validator_claim_fees.rs
- src/args/whitelist_validator_for_program.rs
- src/args/top_up_ephemeral_balance.rs
- src/args/delegate_ephemeral_balance.rs
- src/pda.rs

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

REMEDIATION COMMIT ID:

- https://github.com/magicblock-labs/delegation-program/pull/72/commits/ed1b60da882f8e52fc786c8ada4b272bde8b05f8
- https://github.com/magicblock-labs/delegation-program/pull/72/commits/a7e549d1e76f363f51f9bf9422114cf07d93ced7

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

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	2

INFORMATIONAL
3

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
SINGLE HARD-CODED ADMIN KEY	LOW	SOLVED - 04/02/2025
UNIMPLEMENTED LOGIC IN "VERIFY"	LOW	SOLVED - 04/02/2025
UNRESTRICTED CREATION OF PROTOCOL FEES VAULT	INFORMATIONAL	ACKNOWLEDGED - 04/02/2025
HARD-CODED PROTOCOL_FEES_PERCENTAGE	INFORMATIONAL	ACKNOWLEDGED - 04/02/2025
INSUFFICIENT VALIDATION IN COMMIT_STATE	INFORMATIONAL	SOLVED - 04/02/2025

7. FINDINGS & TECH DETAILS

7.1 SINGLE HARD-CODED ADMIN KEY

// LOW

Description

The `ADMIN_PUBKEY` is a constant defined in `consts.rs`. Multiple parts of the code enforce admin.key == ADMIN_PUBKEY, meaning exactly one globally recognized admin can create validator fee vaults. This centralizes the trust model. if the admin key is lost or compromised, the entire mechanism is compromised (no new validator vaults, or malicious vault creation).

More specifically, the check is present in `close_validator_fees_vault`, `init_validator_fees_vault`, `protocol_claim_fees` and `whitelist_validator_for_program`.

```
    // Check if the admin is the correct one
    if !admin.key.eq(&ADMIN_PUBKEY) {
        msg!(
            "Expected admin pubkey: {} but got {}",
            ADMIN_PUBKEY,
            admin.key
        );
        return Err(Unauthorized.into());
    }
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (2.5)

Recommendation

Consider using a multi-signature wallet or multi-step upgradeable authority approach for enhanced resilience. Additionally, store the admin key in a config account that can be updated, rather than hard-coding a Pubkey as a protocol constant.

Remediation Comment

SOLVED: The **Magic Block team** has solved the issue by removing the hard-coded admin Pubkey and replacing it for the program's `upgrade authority` Pubkey.

Remediation Hash

https://github.com/magicblock-labs/delegation-program/pull/72/commits/ed1b60da882f8e52fc786c8ada4b272bde8b05f8

7.2 UNIMPLEMENTED LOGIC IN "VERIFY"

// LOW

Description

The `verify_state` function contains placeholder logic. It literally does nothing at this stage. Leaving it unimplemented could allow invalid states or malicious updates to go undetected.

```
/// Verify the committed state
#[inline(always)]
pub fn verify_state(
    _authority: &AccountInfo,
    _delegation_record: &DelegationRecord,
    _commit_record: &CommitRecord,
    _commit_state_account: &AccountInfo,
) -> ProgramResult {
    // TODO: Temporary relying on the assumption than the validator fees vault exists (as it was created by the admin)
    Ok(())
}
```

BVSS

AO:A/AC:L/AX:L/R:N/S:U/C:N/A:N/I:L/D:N/Y:N (2.5)

Recommendation

Implement robust verification logic in `verify.rs` (e.g., cryptographic proof checks, signature validations, or protocol invariants). Additionally, remove the `TODO` placeholder once real checks are in place. Ensure the function fails if the new state data is malformed or violates any business-critical constraints.

Remediation Comment

SOLVED: The **Magic Block team** has solved the issue as recommended.

Remediation Hash

https://github.com/magicblock-labs/delegation-program/pull/72/commits/a7e549d1e76f363f51f9bf9422114cf07d93ced7

7.3 UNRESTRICTED CREATION OF PROTOCOL FEES VAULT

// INFORMATIONAL

Description

In `init_protocol_fees_vault` instruction, any user can create the fees vault if it does not exist. The code does not check for an authorized caller. This might be acceptable in the current context, but if the protocol expects only an admin or specific entity to perform initialization, additional checks are required.

```
pub fn process_init_protocol_fees_vault(
    _program_id: &Pubkey,
    accounts: &[AccountInfo],
    _data: &[u8],
) -> ProgramResult {
    // Load Accounts
    let [payer, protocol_fees_vault, system_program] = accounts else {
        return Err(ProgramError::NotEnoughAccountKeys);
    };

    load_signer(payer, "payer")?;
    load_program(system_program, system_program::id(), "system program")?;

    let bump_fees_vault = load_uninitialized_pda(
        protocol_fees_vault,
        fees_vault_seeds!(),
        &crate::id(),
        true,
        "fees vault",
    )?;

    // Create the fees vault account
    create_pda(
        protocol_fees_vault,
        &crate::id(),
        8,
        fees_vault_seeds!(),
        bump_fees_vault,
        system_program,
        payer,
    )?;

    Ok(())
}
```

BVSS

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

Recommendation

It is recommended to add an authorization check so only a recognized admin or multi-signature wallet can create the protocol fees vault. If open creation is intended, document it clearly.

Remediation Comment

ACKNOWLEDGED: The **Magic Block team** has acknowledged this finding, adding to the documentation that `init_protocol_fee_vault` is safe to be executed permissionless.

7.4 HARD-CODED PROTOCOL_FEES_PERCENTAGE

// INFORMATIONAL

Description

The `PROTOCOL_FEES_PERCENTAGE` is a compile-time constant, defined in `constants.rs`. If the protocol needs to adjust fee rates in the future, it can't do so without re-deploying or upgrading the program. This is fine for a static approach but can be limiting if the business logic changes. The `validator_claim_fees` instruction relies on this hard-coded parameter, as follows:

```
    // Calculate fees and remaining amount
    let protocol_fees = (amount * u64::from(PROTOCOL_FEES_PERCENTAGE)) / 100;
```

BVSS

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

Recommendation

It is recommended to store the fee percentage in a config account that an authorized admin can update. If the protocol demands a fixed rate forever, leave it as it is, but document the immutability.

Remediation Comment

ACKNOWLEDGED: The `MagicBlock` team has acknowledged this finding, as per design the protocol fees should remain constant and public.

7.5 INSUFFICIENT VALIDATION IN COMMIT_STATE

// INFORMATIONAL

Description

In the commit_state instruction (see `instruction_builder/commit_state.rs` and `processor/commit_state.rs`), there is a `// TODO` comment referencing "sufficient validator stake" or deeper validation. This indicates the program defers critical checks (e.g., verifying the data in `commit_state_bytes`, ensuring the lamports changes align with protocol rules, or validating off-chain proofs). Without these checks, malicious or incorrect state could be committed.

```
    // Copy the new state to the initialized PDA
    let mut commit_state_data = args.commit_state_account.try_borrow_mut_data()?;
    (*commit_state_data).copy_from_slice(args.commit_state_bytes);

    // TODO - Add additional validation for the commitment, e.g. sufficient validator stake
```

BVSS

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

Recommendation

It is recommended to implement additional logic to verify any off-chain proof or stake data in `commit_state_bytes`.

Remediation Comment

SOLVED: The **Magic Block team** has solved the issue as recommended.

Remediation Hash

https://github.com/magicblock-labs/delegation-program/pull/72/commits/a7e549d1e76f363f51f9bf9422114cf07d93ced7

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	DESCCRIPTION
RUSTSEC-2024-0093	ed25519-dalek	Double Public Key Signing Function Oracle Attack on `ed255109-dalek`
RUSTSEC-2024-0344	curve25519-dalek	Timing variability in `curve25519-dalek` 's Scalar29::sub/Scalar52::sub
RUSTSEC-2025-0004	openssl	`ssl::select_next_proto` use after free.

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.