// Security Assessment

01.26.2026 - 01.27.2026

# Glow Margin "Diff Audit"

*Blueprint Finance*

HALBORN

# Glow Margin "Diff Audit" - Blueprint Finance

Prepared by: HALBORN

Last Updated 02/11/2026

Date of Engagement: January 26th, 2026 - January 27th, 2026

## Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

| ALL FINDINGS | CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL |
|---|---|---|---|---|---|
| 2 | 0 | 0 | 0 | 1 | 1 |

## TABLE OF CONTENTS

## 1. Introduction

`Blueprint Finance` engaged Halborn to conduct a security assessment on their `Glow` program beginning on January 26th, 2026 and ending on January 27th, 2026. The security assessment was scoped to the smart contracts provided in the GitHub repository glow-v1, commit hashes, and further details can be found in the Scope section of this report.

The `Blueprint Finance team` is releasing updates to their Glow Solana programs. Glow is a DeFi protocol consisting of three main programs: `margin-pool` (lending and borrowing functionality), `margin` (margin account management and liquidation), and `vault` (user deposit and withdrawal management). This differential audit covers security improvements and bug fixes across all three programs.

## 2. Assessment Summary

Halborn was provided 2 business days 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 Program.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks, which were partially addressed by the `Blueprint Finance team`:

- `Refactor fee accounting to separate protocol reserves from depositor accounting following industry standard patterns`
- `Add explicit airspace validation to instructions for defense-in-depth consistency`

## 3. Test Approach And Methodology

Halborn performed a combination of manual review and security testing based on scripts to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Differences analysis using GitLens to have a proper view of the differences between the mentioned commits
- Graphing out functionality and programs logic/connectivity/functions along with state changes

## 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)<br>Specific (AO:S) | 1<br>0.2 |
| Attack Cost (AC) | Low (AC:L)<br>Medium (AC:M)<br>High (AC:H) | 1<br>0.67<br>0.33 |

| EXPLOITABILITY METRIC ($M_E$) | METRIC VALUE | NUMERICAL VALUE |
|---|---|---|
| Attack Complexity (AX) | Low (AX:L)<br>Medium (AX:M)<br>High (AX:H) | 1<br>0.67<br>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 (C:N)<br>Low (C:L)<br>Medium (C:M)<br>High (C:H)<br>Critical (C:C) | 0<br>0.25<br>0.5<br>0.75<br>1 |

| IMPACT METRIC ($M_I$) | METRIC VALUE | NUMERICAL VALUE |
|---|---|---|
| Integrity (I) | None (I:N)<br>Low (I:L)<br>Medium (I:M)<br>High (I:H)<br>Critical (I:C) | 0<br>0.25<br>0.5<br>0.75<br>1 |
| Availability (A) | None (A:N)<br>Low (A:L)<br>Medium (A:M)<br>High (A:H)<br>Critical (A:C) | 0<br>0.25<br>0.5<br>0.75<br>1 |
| Deposit (D) | None (D:N)<br>Low (D:L)<br>Medium (D:M)<br>High (D:H)<br>Critical (D:C) | 0<br>0.25<br>0.5<br>0.75<br>1 |
| Yield (Y) | None (Y:N)<br>Low (Y:L)<br>Medium (Y:M)<br>High (Y:H)<br>Critical (Y:C) | 0<br>0.25<br>0.5<br>0.75<br>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)<br>Partial (R:P)<br>Full (R:F) | 1<br>0.5<br>0.25 |
| Scope ($s$) | Changed (S:C)<br>Unchanged (S:U) | 1.25<br>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

### REPOSITORY

(a) Repository: glow-v1

(b) Assessed Commit ID: 4a92bf5

(c) Items in scope:

- programs/margin-pool/src/instructions/close_loan.rs
- programs/margin-pool/src/instructions/margin_borrow.rs
- programs/margin-pool/src/instructions/margin_borrow_v2.rs
- programs/margin-pool/src/instructions/margin_refresh_position.rs
- programs/margin-pool/src/instructions/margin_repay.rs
- programs/margin-pool/src/instructions/register_loan.rs
- programs/margin-pool/src/instructions/withdraw.rs
- programs/margin-pool/src/instructions/withdraw_fees.rs
- programs/margin/src/instructions/liquidator_invoke.rs
- programs/margin/src/instructions/oracle/create_price_feed.rs
- programs/margin/src/instructions/oracle/refresh_price_feed.rs
- programs/margin/src/instructions/positions/transfer_deposit.rs
- programs/margin/src/lib.rs
- programs/margin/src/state/price_feed.rs
- programs/vault/src/instructions/operator/operator_deposit_to_vault.rs
- programs/vault/src/instructions/operator/operator_transfer_from_margin.rs
- programs/vault/src/instructions/operator/operator_transfer_to_margin.rs
- programs/vault/src/instructions/operator/operator_withdraw_from_vault.rs
- programs/vault/src/instructions/user/cancel_vault_pending_withdrawal.rs
- programs/vault/src/instructions/user/deposit.rs
- programs/vault/src/instructions/user/execute_vault_withdrawal.rs
- programs/vault/src/instructions/user/initiate_withdrawal.rs
- programs/vault/src/state/operator.rs
- programs/vault/src/state/vault_user.rs

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

### FILE

(a) Submitted File: glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad.zip

(b) Items in scope:

- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/margin/src/instructions/oracle/create_price_feed.rs

- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/margin/src/instructions/oracle/refresh_price_feed.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/margin/src/instructions/positions/transfer_deposit.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/margin/src/instructions/liquidator_invoke.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/margin/src/state/price_feed.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/margin/src/lib.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/margin-pool/src/instructions/close_loan.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/margin-pool/src/instructions/margin_borrow_v2.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/margin-pool/src/instructions/margin_borrow.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/margin-pool/src/instructions/margin_refresh_position.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/margin-pool/src/instructions/margin_repay.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/margin-pool/src/instructions/register_loan.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/margin-pool/src/instructions/withdraw_fees.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/margin-pool/src/instructions/withdraw.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/vault/src/instructions/operator/operator_deposit_to_vault.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/vault/src/instructions/operator/operator_transfer_from_margin.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/vault/src/instructions/operator/operator_transfer_to_margin.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/vault/src/instructions/operator/operator_withdraw_from_vault.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/vault/src/instructions/user/cancel_vault_pending_withdrawal.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/vault/src/instructions/user/deposit.rs

- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/vault/src/instructions/user/execute_vault_withdrawal.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/vault/src/instructions/user/initiate_withdrawal.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/vault/src/state/operator.rs
- /glow-v1-4a92bf54e83b4fa64ceed4fcafd17600217a90ad/programs/vault/src/state/vault_user.rs

REMEDIATION COMMIT ID:

- 8e3c479

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

## 6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

| CRITICAL | HIGH | MEDIUM | LOW |
|---|---|---|---|
| 0 | 0 | 0 | 1 |

| INFORMATIONAL |
|---|
| 1 |

| SECURITY ANALYSIS | RISK LEVEL | REMEDIATION DATE |
|---|---|---|
| PROTOCOL FEE ACCOUNTING MIXED WITH DEPOSITOR ACCOUNTING DEVIATES FROM INDUSTRY STANDARD | LOW | RISK ACCEPTED - 02/05/2026 |
| MISSING AIRSPACE VALIDATION IN MARGIN POOL INSTRUCTIONS VIOLATES DEFENSE-IN-DEPTH PATTERN | INFORMATIONAL | SOLVED - 02/06/2026 |

# 7. FINDINGS & TECH DETAILS

## 7.1 PROTOCOL FEE ACCOUNTING MIXED WITH DEPOSITOR ACCOUNTING DEVIATES FROM INDUSTRY STANDARD

// LOW

### Description

The margin-pool program implements a lending pool where depositors provide liquidity and earn interest from borrowers, while a portion of the accrued interest is collected as protocol fees.

Fee management is handled through `collect_accrued_fees()`, which converts accumulated fees into deposit notes, and `withdraw_fees()`, which allows the fee owner to redeem those notes for underlying tokens.

The implementation introduces accounting risk by merging protocol fee accounting with depositor share accounting. Fees are converted into `deposit_notes`, the same asset used to represent depositor ownership, instead of being tracked separately as protocol reserves, as commonly implemented in lending protocols such as Compound V2.

Because fees increase the total `deposit_notes`, subsequent fee withdrawals directly impact the same `deposit_tokens` and `deposit_notes` variables that determine depositor balances. This design breaks the logical separation between protocol-owned funds and user-owned liquidity, increasing the likelihood of accounting inconsistencies and unintended value transfer under certain conditions.

*programs/margin-pool/src/instructions/withdraw_fees.rs*

Copy Code

```
101 pub fn withdraw_fees_handler(ctx: Context<WithdrawFees>) -> Result<()> {
102     let pool = &mut ctx.accounts.margin_pool;
103
104     let fee_notes = ctx.accounts.fee_destination.amount;
105
106     let claimed_amount = pool.convert_amount(Amount::notes(fee_notes), PoolAction::Withdraw)?;
107     pool.withdraw(&claimed_amount)?;
108     ...
```

While the current implementation is mathematically functional and depositors are not materially harmed, the mixing of fee accounting with depositor accounting deviates from the industry standard established by Compound V2, one of the most widely adopted and audited lending protocols in DeFi.

This design choice can lead to minor side effects such as small rounding variations in the exchange rate during fee operations and reduced transparency in distinguishing protocol-owned assets from user-owned assets.

The primary concern is the architectural deviation from proven patterns that separate reserve accounting from supplier share accounting.

## BVSS

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

## Recommendation

It is recommended to refactor the fee accounting to follow the industry standards. This involves:
1. **Maintain fees as underlying tokens**: Keep protocol fees in a separate `total_reserves` variable denominated in underlying tokens, not deposit notes.
2. **Modify exchange rate formula**: Update the exchange rate calculation to explicitly exclude reserves:
3. **Implement direct reserve withdrawal**: Create a `reduce_reserves()` function that directly decrements `total_reserves` and transfers tokens without affecting `deposit_notes`.

## Remediation Comment

**RISK ACCEPTED:** The **Blueprint Finance team** accepted the risk of this finding, as the fix would require a complex migration of account data to a new margin pool layout—a move they currently deem **unnecessary** by now.

## References

compound-finance/compound-protocol/contracts/CToken.sol

## 7.2 MISSING AIRSPACE VALIDATION IN MARGIN POOL INSTRUCTIONS VIOLATES DEFENSE-IN-DEPTH PATTERN

// INFORMATIONAL

### Description

The margin-pool program provides lending and borrowing functionality for margin accounts within isolated airspaces.

Each airspace operates as an independent environment with its own set of pools and margin accounts. To maintain proper isolation, instructions that interact with both margin accounts and margin pools should validate that both entities belong to the same airspace.

Several instructions in the margin-pool program like `margin_borrow`, `margin_borrow_v2`, `register_loan`, and `withdraw`, correctly implement airspace validation by checking that `margin_account.airspace == margin_pool.airspace`.

However, three other instructions (`margin_repay`, `close_loan`, and `margin_refresh_position`) do not perform this validation, as shown in the code snippets below:

*programs/margin-pool/src/instructions/margin_repay.rs*

Copy Code

```
pub fn margin_repay_handler(
    ctx: Context<MarginRepay>,
    change_kind: ChangeKind,
    amount: u64,
) -> Result<()> {
    let change = TokenChange {
        kind: change_kind,
        tokens: amount,
    };
    ...
```

For instance, the `margin_refresh_position` instruction demonstrates why defense-in-depth validation is important.

Unlike `margin_repay` and `close_loan` which require `margin_account` to be a signer (restricting calls to `adapter_invoke` with owner signature), `margin_refresh_position` is fully permissionless - any caller can invoke it on any margin account with any pool.

A PoC showed that `margin_refresh_position` accepts pools from different airspaces (missing validation), but no data corruption occurs because each pool's mints are unique (PDA-derived) and `apply_changes` only updates positions that already exist in the account.

This represents a "near miss" - the attack vector exists and the unauthorized cross-airspace call succeeds, but is neutralized by unrelated internal logic. This creates a fragile security posture where:

- Future changes to position handling could inadvertently make the attack exploitable

- The inconsistent validation pattern across instructions makes the codebase harder to audit
- Relying on implicit protections instead of explicit validation violates defense-in-depth principles

For `margin_repay` and `close_loan`, the signer requirement provides stronger protection, but the lack of explicit airspace validation still represents an inconsistency with the pattern established by `margin_borrow`, `margin_borrow_v2`, `register_loan`, and `withdraw`.

## Proof of Concept

### Test Scenario

1. Bob owns a margin account in airspace A with positions in pool1 (airspace A)
2. An attacker calls `margin_refresh_position` via `accounting_invoke` with Bob's margin account but using pool2 from airspace B
3. The transaction executes successfully - no `PoolPermissionDenied` error is returned

The attack transaction succeeds because there is no airspace validation. However, no actual data corruption occurs due to internal safeguards:
1. **Unique pool mints**: Each pool derives its `deposit_note_mint` and
2. `loan_note_mint` from its own address via PDA seeds `[margin_pool, 'deposit-notes'/'loan-notes']`. Pool1 and pool2 have completely different mint addresses.
3. **Position matching in** `apply_changes`: The function only updates positions that already exist in the margin account. Since Bob's account only has positions for pool1's mints, and pool2 returns price updates for pool2's mints, the `apply_changes` function finds no matching positions to update.

### PoC Code

Copy Code

```
// Build margin_refresh_position instruction using pool2 (from airspace 2)
// but targeting bob_margin_account (from airspace 1)
let refresh_ix = pool2.margin_refresh_position(
    bob_margin_account,  // Victim's margin account in Airspace 1
    oracle_address,      // Oracle
    None,                // No redemption oracle
);

// The attacker (NOT Bob) calls accounting_invoke on Bob's margin account
// accounting_invoke doesn't require the margin account owner's signature!
let attack_ix = accounting_invoke(
    airspace1,           // Bob's airspace (required for adapter lookup)
    bob_margin_account,  // Victim's margin account
    refresh_ix,          // The cross-airspace refresh instruction
);

// Execute the attack
let attack_result = attack_ix
    .with_signer(&attacker)
    .send_and_confirm(&ctx.rpc())
    .await;

// ===== STEP 9: Verify the attack succeeded (vulnerability exists) =====
// If the vulnerability EXISTS, the transaction should succeed
assert!(
    attack_result.is_ok(),
    "VULNERABILITY CONFIRMED: Attack transaction succeeded. Error: {:?}",
    attack_result.err()
);
println!("Attack transaction succeeded - vulnerability confirmed!");
```

## Evidence

```
AAAAAPj///88AAAAAAAAAA==
[2026-01-28T16:28:57.427105000Z DEBUG solana_runtime::message_processor::stable_log] Program CWPeEXnSpELj7tSz9W4oQA
GGRbavBtdnhY2bWMyPoo1 success
[2026-01-28T16:28:57.427708000Z DEBUG solana_runtime::message_processor::stable_log] Program data: 1g5z9jARO/4=
[2026-01-28T16:28:57.427785000Z DEBUG solana_runtime::message_processor::stable_log] Program GLoWMgcn3VbyFKiC2FGMgf
KxYSyTJS7uKFwKY2CSkq9X consumed 49891 of 200000 compute units
[2026-01-28T16:28:57.427808000Z DEBUG solana_runtime::message_processor::stable_log] Program GLoWMgcn3VbyFKiC2FGMgf
KxYSyTJS7uKFwKY2CSkq9X success
Attack transaction succeeded - vulnerability confirmed!
Bob's deposit position price timestamp AFTER attack: 1769617736
```

### BVSS

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

### Recommendation

It is recommended to add explicit airspace validation to all three `margin_repay`, `close_loan`, and `margin_refresh_position` instructions for consistency with the existing pattern.

### Remediation Comment

**SOLVED:** The **Blueprint Finance team** solved the issue by implementing the suggested changes.

### Remediation Hash

https://github.com/Blueprint-Finance/glow-v1/commit/8e3c4795e5d43bc857e025da230fe4fb831f2c88

# 8. AUTOMATED TESTING

## Description

Halborn used automated security scanners to assist with the 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 reviewers 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.

## Results

| ID | Package | Short Description |
|---|---|---|
| RUSTSEC-2024-0344 | curve25519-dalek | Timing variability in curve25519-dalek's |
| 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.