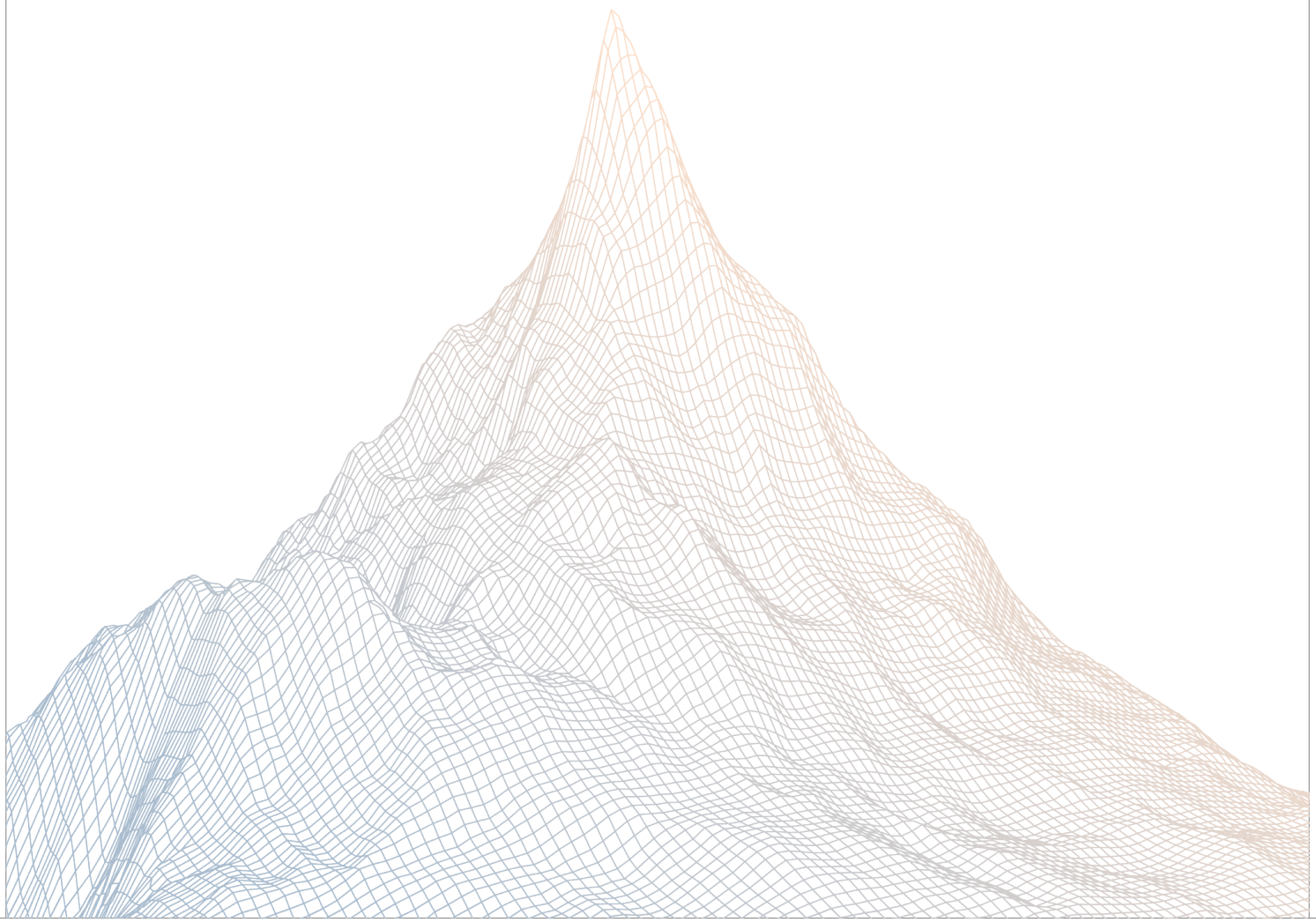
Zenith

Bridge

Smart Contract Security Assessment

VERSION 1.1



AUDIT DATES: April 21th to April 22nd, 2025

AUDITED BY: J4X
peakbolt

Contents

1 Introduction

1.1 About Zenith

Zenith is an offering by Code4rena that provides consultative audits from the very best security researchers in the space. We focus on crafting a tailored security team specifically for the needs of your codebase.

Learn more about us at https://code4rena.com/zenith.

1.2 Disclaimer

This report reflects an analysis conducted within a defined scope and time frame, based on provided materials and documentation. It does not encompass all possible vulnerabilities and should not be considered exhaustive.

The review and accompanying report are presented on an "as-is" and "as-available" basis, without any express or implied warranties.

Furthermore, this report neither endorses any specific project or team nor assures the complete security of the project.

1.3 Risk Classification

SEVERITY LEVEL	IMPACT: HIGH	IMPACT: MEDIUM	IMPACT: LOW
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

2

Executive Summary

2.1 About Bridge

Bridge gives everyone access to world-class financial services. We believe stablecoins will transform and improve global money movement. Bridge creates the infrastructure necessary for builders to take full advantage of this new medium.

Since launching 18 months ago, we've provided millions with faster and cheaper access to cross-border payments, enabled governments and aid agencies to more efficiently distribute funds to thousands, and given millions more true economic choice, enabling them to easily save and spend in USD or EUR.

2.2 Scope

The engagement involved a review of the following targets:

Target	bridge-cards-programs
Repository	https://github.com/withbridge/bridge-cards-programs
Commit Hash	`b1acaf849dc7f5622ea25702bc0728e7310c12f2`
Files	`programs/bridge_cards/*`

2.3 Audit Timeline

April 21, 2025	Audit start
April 22, 2025	Audit end
April 28, 2025	Report published

2.4 Issues Found

SEVERITY	COUNT
Critical Risk	0
High Risk	0
Medium Risk	2
Low Risk	4
Informational	3
Total Issues	**9**

3

Findings Summary

ID	Description	Status
M-1	Each Debitor is not mapped to the (merchantId, mint) pair	Resolved
M-2	Duplicate debit_user() could be triggered due to lack of mapping to card payment	Acknowledged
L-1	close_account_and_transfer_lamports() allows for revival attacks	Resolved
L-2	Unchecked math could lead to overflows	Resolved
L-3	initialize() can be called by anyone after deployment to gain admin rights	Resolved
L-4	per_transfer_limit can be bypassed by executing multiple debit_user() IX in the same TX	Acknowledged
I-1	Bumps can be saved to reduce CU	Resolved
I-2	UserDelegateAddedOrUpdated emits wrong merchant_pda	Resolved
I-3	debit_user should emit event for easier monitoring	Resolved

4

Findings

4.1 Medium Risk

A total of 2 medium risk findings were identified.

[M-1] Each Debitor is not mapped to the (merchantId, mint) pair

SEVERITY: Medium	IMPACT: Medium
STATUS: Resolved	LIKELIHOOD: Medium

Target

- add_or_update_merchant_debitor.rs#L78-L89

Description:

As indicated in the docs, one of the invariant is that the debitors are pair to a (merchantId, mint) pair,

```
Debitors are strictly scoped to their (merchantId, mint) pair.
```

However, the `MerchantDebitorState` PDA is only seeded by the `merchant_id` and does not include the token mint. This violates the above invariant.

```
/// PDA storing the debitor's authorization state
/// Seeds: [MERCHANT_DEBITOR_SEED, merchant_id, debitor]
/// Space: Discriminator + Boolean
/// Required permissions: Mutable if new, Read-only if existing
#[account(
    init_if_needed,
    payer = payer,
    space = MerchantDebitorState::DISCRIMINATOR.len() +
    MerchantDebitorState::INIT_SPACE,
    seeds = [
        MERCHANT_DEBITOR_SEED,
        &merchant_id.to_le_bytes(),
        &debitor.key().as_ref(),
    ],
    bump
)]
```

```
pub debitor_state: Account<'info, MerchantDebitorState>,
```

Recommendations:

Add the token mint to the seed for the `MerchantDebitorState` PDA.

Bridge: Resolved with @809d0b0b27...

Zenith: Verified.

[M-2] Duplicate `debit_user()` could be triggered due to lack of mapping to card payment

SEVERITY: Medium	IMPACT: Medium
STATUS: Acknowledged	LIKELIHOOD: Low

Target

- debit_user.rs#L116

Description:

When a card payment is made, the debitor will call `debit_user()` with the corresponding parameters to perform the payment on-chain. However, as the program allows more than one debitors to be authorized to call `debit_user()` for each merchant, it is possible for duplicate `debit_user()` to be triggered for the same card payment. That is because there are no mapping of the card payment to the `debit_user()` tx that prevent duplicate calls.

This scenario could occur if there are multiple debitors deployed for redundancy purpose. For example, a secondary debitor could take over when the primary debitor is down, and will perform the `debit_user()` without knowing if it has already been triggered. The occurrence of this issue depends on how the debitors are setup and whether there are off-chain tracking to prevent duplicate `debit_user()`.

Recommendations:

One possible solution is to implement a strictly incrementing nonce for each merchant (e.g. `merchant_nonce`), that will be increment off-chain for each card payment and serves as an unique id for each card payment.

The debitor can provide this `merchant_nonce` in `debit_user()`, which validates that `merchant_nonce == stored_merchant_nonce + 1`. If it is valid, the `stored_merchant_nonce` will be incremented on-chain as well. Note that `stored_merchant_nonce` has to be incremented even when the transfer fails, to ensure that the nonce is synchronized with the off-chain nonce.

Bridge: We can enforce this on the backend.

Zenith: This will be resolved at the backend to prevent duplicate calls by multiple Debitors.

4.2 Low Risk

A total of 4 low risk findings were identified.

[L-1] `close_account_and_transfer_lamports()` allows for revival attacks

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: High

Target

- programs/bridge_cards/src/utils.rs#L4-L13

Description:

The `close_account_and_transfer_lamports` function implements an unsafe way of closing an account.

```
pub fn close_account_and_transfer_lamports<'info>(
    account_to_close: &AccountInfo<'info>,
    recipient: &AccountInfo<'info>,
) -> Result<()> {
    // Transfer all lamports from the account to the recipient
    let lamports = account_to_close.lamports();
    **account_to_close.try_borrow_mut_lamports()? = 0;
    **recipient.try_borrow_mut_lamports()? += lamports;
    Ok(())
}
```

Reducing the lamports without reassigning/reallocating account allows for a revival attack.

Recommendations:

We recommend using anchors `close` constraint instead.

Bridge: Resolved with PR-12

Zenith: Verified

[L-2] Unchecked math could lead to overflows

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: High

Target

- programs/bridge_cards/src/state.rs#L58

Description:

The calculation of the transfer amount limit uses unchecked math:

```
if self.period_transferred_amount + amount > self.period_transfer_limit {
    return Err(ErrorCode::ExceedsTransferLimitPerPeriod.into());
}
```

This could allow for a potential overflow on tokens with a very high volume.

Recommendations:

We recommend using checked math.

Bridge: Resolved with PR-13

Zenith: Verified

[L-3] initialize() can be called by anyone after deployment to gain admin rights

SEVERITY: Low	IMPACT: Low
STATUS: Resolved	LIKELIHOOD: Low

Target

- initialize.rs#L36-L40

Description:

`initialize()` is called after deployment by the admin to init the program state PDA and then assign the admin pubkey.

However, there are no access control for it, which then allows anyone to call `initialize()` and gain the admin privilege. This will then require a re-deployment of the program.

Recommendations:

Consider gating `initialize()` using a constant admin key pair as below,

```
#[program]
pub mod bridge_cards {
    use super::*;
    pub const ADMIN_ID: Pubkey = pubkey!("Change this to admin pubkey");

    ...
}

pub struct Initialize<'info> {
   #[account(mut)]
    #[account(
        mut,
        address = crate::ADMIN_ID @ ErrorCode::InvalidAdmin
    )]
    pub payer: Signer<'info>,
}
```

Bridge: Resolved with @280cb0b864... and @09166b3898....

Zenith: Resolved by gating `initialize()` with keypair.

[L-4] `per_transfer_limit` can be bypassed by executing multiple `debit_user()` IX in the same TX

SEVERITY: Low	IMPACT: Low
STATUS: Acknowledged	LIKELIHOOD: Medium

Target

- state.rs#L49-L51

Description:

A `per_transfer_limit` is imposed for `debit_user()` to limit the amount of tokens that can be transferred in a single transaction.

However, this limit can be bypassed by executing multiple `debit_user()` instructions within the same transaction.

Though, the impact is limited as the transferred is still capped by the `period_transferred_amount`.

```
impl UserDelegateState {
    pub fn validate_debit_and_update(&mut self, amount: u64, current_time:
    u64) -> Result<()> {
        if amount > self.per_transfer_limit {
            return Err(ErrorCode::ExceedsMaxTransferLimit.into());
        }
        if current_time - self.period_timestamp_last_reset
            > self.transfer_limit_period_seconds as u64
        {
            self.period_transferred_amount = 0;
            self.period_timestamp_last_reset = current_time;
        }
        if self.period_transferred_amount +
    amount > self.period_transfer_limit {
            return Err(ErrorCode::ExceedsTransferLimitPerPeriod.into());
        }
        self.period_transferred_amount += amount;
        Ok(())
    }
}
```

Recommendations:

Consider imposing `per_transfer_limit` for each slot instead of each instruction.

Bridge: Acknowledged.

Zenith: This is mitigated by backend, which will trigger `debit_user()` with a processing time that is greater than slot time (400ms), preventing multiple `debit_user()` within a block. This provides ample time for the backend to detect the failed transfer in `debit_user()` from the frontrunning scenario and block the card before the malicious user can continue to trigger another `debit_user()`.

4.3 Informational

A total of 3 informational findings were identified.

[I-1] Bumps can be saved to reduce CU

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: High

Target

Description:

The `bump` constraint is used for every account usage inside the codebase. This requires the caller to recalculate the canonical bump for the account.

Recommendations:

We recommend saving the bumps inside the accounts to save CU.

Bridge: Resolved with PR-20

Zenith: Verified

[I-2] `UserDelegateAddedOrUpdated` emits wrong `merchant_pda`

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: High

Target

- programs/bridge_cards/src/instructions/add_or_update_user_delegate.rs#L147

Description:

The `UserDelegateAddedOrUpdated` event which is emitted in the `add_or_update_user_delegate` IX contains a wrong entry in the `merchant_pda` field.

```
// Emit event for indexing and notifications
emit!(UserDelegateAddedOrUpdated {
    merchant_pda: ctx.accounts.user_delegate_account.key(),
    user_delegate: ctx.accounts.user_delegate_account.key(),
});
```

Recommendations:

We recommend either removing the `merchant_pda` or passing the merchant account in and adding it there.

Bridge: Resolved with PR-19

Zenith: Verified

[I-3] `debit_user` should emit event for easier monitoring

SEVERITY: Informational	IMPACT: Informational
STATUS: Resolved	LIKELIHOOD: High

Target

- programs/bridge_cards/src/instructions/debit_user.rs#L116

Description:

The `debit_user` IX is sued so that the delegate can withdraw funds on the users behalf. However it currently does not emit any events so this can be easily monitored off-chain.

Recommendations:

We recommend adding an additional event to ensure for easier monitoring.

Bridge: Resolved with PR-18

Zenith: Verified