Melee Prediction Market Security Review

AUDITED BY LAMSY

15 May 2025 - 19 May 2025

- Twitter: @lamsy
- Email: olamide.adetula.1744@gmail.com
- Telegram: Lamsy
- GitHub: LamsyA
- Discord: LamsyA

Protocol Summary

The Melee Prediction Market is a decentralized platform built on Solana that enables users to create and participate in prediction markets. The protocol implements a bonding curve mechanism for token pricing, where users can buy shares in different outcomes of a market. When a market resolves, winners can claim rewards proportional to their share of the winning outcome.

Key features of the Melee Prediction Market include:

- Market creation with customizable parameters
- Role-based access control for market creators and administrators
- Bonding curve pricing mechanism for share purchases
- Market resolution and reward distribution
- Void market handling for cancelled markets

The core components of the protocol are:

- **Market**: Manages market creation, buying, and claiming
- **User**: Handles user roles and permissions
- **Config**: Stores protocol configuration parameters

The protocol aims to provide a transparent and efficient platform for prediction markets on Solana, with proper economic incentives for participants.

- Reviewed Version: https://github.com/melee-markets/melee-solana-programs
- commit hash: 2a5c328
- Fix commit hash: c81235f

Disclaimer

This security review is not to be considered as a security guarantee. The review is conducted to the best of my knowledge according to industry best practices and understanding of the codebase at the time of review.

Audit Details

Impact	High	Medium	Low	Informational
High	0	0	0	0
Medium	0	5	0	0
Low	0	0	1	0
Informational	0	0	0	2

Scope

The audit covers the core smart contracts of the Melee Prediction Market protocol

Files in Scope

```
Files in Scope:
```

```
* predict-market/programs/predict-market/src/instructions/market_buy.rs
* predict-market/programs/predict-market/src/instructions/market_claim.rs
* predict-market/programs/predict-market/src/instructions/market_resolve.rs
* predict-market/programs/predict-market/src/instructions/market_close.rs
* predict-market/programs/predict-market/src/instructions/config_update.rs
* predict-market/programs/predict-market/src/instructions/market_void.rs
* predict-market/programs/predict-market/src/instructions/market_create.rs
* predict-market/programs/predict-market/src/instructions/user_role_setup.rs
* predict-market/programs/predict-market/src/instructions/user_account_init.rs
* predict-market/programs/predict-market/src/instructions/user_account_close.rs
* predict-market/programs/predict-market/src/instructions/loser_claim_after_market_close.rs
* predict-market/programs/predict-market/src/instructions/melee_init.rs
```

Issues found

The audit identified several issues categorized by their severity:

Medium Severity

- **Fee Distribution in Void Markets**: The `market_resolve` function lacked special handling for void markets, allowing a malicious market creator to exploit the system by voiding markets when they're losing and collecting both fees and refunds.
- **Missing Outcome Index Validation in Market Resolve**: The `market_resolve` function did not validate that the `resolved_outcome_index` was within the bounds of the market's outcomes array.
- **Incorrect Fee Parameter in Void Market Refunds**: The void market refund calculation used `creator_rewards_bips` instead of `trading_fee_bips`, potentially leading to incorrect refund amounts.
- **Missing Outcome Index Validation**: The code accessed `outcomes[outcome_index]` without validating that the index was within bounds.
- **No Minimum Purchase Amount**: No minimum limit was set on the `amount` parameter, allowing dust transactions.

Low Severity

- **Unchecked Addition in Token Transfer**: The code used `unwrap()` on a `checked_add` result which could panic under extreme conditions.

Informational

- **Unchecked Addition in `update_buyer`**: The `update_buyer` function used unchecked addition which could theoretically overflow.
- **Inconsistent Logging**: The code used a mix of `msg!` and `.log()` for logging, making debugging harder.

Findings

Medium

Fee Distribution in Void Markets

Summary

The `market_resolve` function lacked special handling for void markets, allowing a malicious market creator to exploit the system by voiding markets when they're losing and collecting both fees and refunds.

Finding Description

In the `market_resolve` function, fees were distributed to the creator and treasury regardless of the `resolved_outcome_index` value:

```rust
// Calculate and distribute fees
let fees = self.outcome_escrow.amount - total_tokens_spent_on_outcomes;
let creator_rewards_fee = fees * creator_rewards_bips / 10_000;
let admin_collect_fee = fees - creator_rewards_fee;

// Transfer to creator and treasury
transfer(cpi_ctx, creator_rewards_fee)?;
transfer(ctx, admin_collect_fee)?;

// Set resolved outcome index
self.market.resolved_outcome_index = Some(resolved_outcome_index);
```

This created an exploitable scenario where a market creator could:

1. Create a market and place bets on one or more outcomes
2. Wait for the market to end
3. If their chosen outcome lost, resolve the market with `VOID_MAGIC_NUMBER` (255)
4. Collect creator rewards from the fee distribution
5. Also claim a full refund for their losing positions through `buyer_withdraw_for_void_market`

Impact Explanation

This vulnerability had significant economic implications:

1. A malicious market creator could avoid losses by voiding markets when their bets were losing
2. They would receive double benefits: creator rewards from fee distribution and full refunds for their positions
3. This created a perverse incentive for market creators to void markets selectively based on their own positions
4. If many users tried to claim refunds, there might not be enough tokens in the escrow due to the fees already distributed

Likelihood Explanation

Medium likelihood as it requires a market creator to intentionally exploit the system, but the economic incentive to do so is significant, especially for markets with large amounts at stake.

Recommendation

Modified `market_resolve` to skip fee distribution for void markets:

```rust
pub fn market_resolve(&mut self, resolved_outcome_index: u8) -> Result<()> {
    // Validate outcome index
    require!(
        resolved_outcome_index < self.market.outcomes.len() as u8 ||
        resolved_outcome_index == VOID_MAGIC_NUMBER,
        MarketError::InvalidOutcomeIndex
    );

    // Other validations...

    // If this is a void market, don't distribute fees
    if resolved_outcome_index == VOID_MAGIC_NUMBER {
        // Just set the resolved outcome index and return
        self.market.resolved_outcome_index = Some(resolved_outcome_index);

        // Emit an event for the void market resolution
        emit!(MarketResolvedEvent {
            market: self.market.key(),
            resolved_outcome_index,
            is_void: true,
            creator_rewards_fee: 0,
            admin_collect_fee: 0,
        });

        return Ok(());
    }

    // Regular market resolution logic continues here...
}
```

Melee: fixed in commit c81235f84af3510a9c38fe9b62b1922f33a5ca21
Lamsy: Verified

Missing Outcome Index Validation in Market Resolve

Summary

The `market_resolve` function did not validate that the `resolved_outcome_index` was within the bounds of the market's outcomes array.

Finding Description

In the `market_resolve` function, the code set the resolved outcome index without validating that it was within bounds:

```
// No validation before setting
self.market.resolved_outcome_index = Some(resolved_outcome_index);
```

If an out-of-bounds index was provided, it would create an invalid market state that could cause subsequent claim operations to fail with runtime errors.

Impact Explanation

This vulnerability could lead to:

1. Invalid market states where the resolved outcome index points to a non-existent outcome
2. Runtime panics when users try to claim their winnings
3. Potential fund loss if users are unable to claim their winnings due to the invalid state

Likelihood Explanation

Low likelihood as it requires a market creator to intentionally or accidentally provide an invalid index. The impact would affect all users who participated in the market.

Recommendation

Added validation to ensure the outcome index is within bounds or is the special void market value:

```
require!(
    resolved_outcome_index < self.market.outcomes.len() as u8 ||
    resolved_outcome_index == VOID_MAGIC_NUMBER,
    MarketError::InvalidOutcomeIndex
);
```

Melee: fixed in commit c81235f84af3510a9c38fe9b62b1922f33a5ca21
Lamsy: Verified

Incorrect Fee Parameter in Void Market Refunds

Summary

The void market refund calculation used `creator_rewards_bips` instead of `trading_fee_bips`, potentially leading to incorrect refund amounts.

Finding Description

In the `buyer_withdraw_for_void_market` function, the code calculated refunds using `creator_rewards_bips` instead of `trading_fee_bips`:

```
let trading_fee_bips = u128::from(self.market.config.creator_rewards_bips); // Incorrect parameter
let trading_fee = u128::from(self.buyer_buy_account.tokens_spent)
    .checked_mul(trading_fee_bips)
    .ok_or(MarketError::ArithmeticOverflow)?
    .checked_div(10_000)
    .ok_or(MarketError::ArithmeticUnderflow)?;
```

This created a discrepancy between the fee charged during market buy (based on `trading_fee_bips`) and the amount refunded in void markets (based on `creator_rewards_bips`).

Impact Explanation

This vulnerability had significant economic implications:

1. If `creator_rewards_bips < trading_fee_bips` (the typical case), users would be undercompensated in void markets, receiving less than they originally paid in fees.
2. If `creator_rewards_bips > trading_fee_bips` (an invalid configuration), users could profit from void markets, creating a perverse incentive for market manipulation.
3. The remaining fee difference would stay in the escrow and could potentially be claimed by the market creator when closing the market, creating an incentive to void markets.

Likelihood Explanation

Medium likelihood as this would affect all void market scenarios, which are a normal part of prediction market operations. The economic impact would be proportional to the difference between `trading_fee_bips` and `creator_rewards_bips`.

Recommendation

Fixed the void market refund calculation to use the correct fee parameter:

```
let trading_fee_bips = u128::from(self.market.config.trading_fee_bips); // Correct parameter
let trading_fee = u128::from(self.buyer_buy_account.tokens_spent)
    .checked_mul(trading_fee_bips)
    .ok_or(MarketError::ArithmeticOverflow)?
    .checked_div(10_000)
    .ok_or(MarketError::ArithmeticUnderflow)?;
```

Additionally, added a validation check in market creation to ensure the fee parameters maintain the correct relationship:

```
require!(
    self.config.creator_rewards_bips <= self.config.trading_fee_bips,
    MarketError::InvalidCreatorRewardsBips
);
```

Melee: fixed in commit c81235f84af3510a9c38fe9b62b1922f33a5ca21
Lamsy: Verified

Missing Outcome Index Validation

Summary

The code accessed `outcomes[outcome_index]` without validating that the index was within bounds.

Finding Description

The `market_buy` function accepted an `outcome_index` parameter without validating that it was within the bounds of the market's outcomes array. If a malicious user provided an out-of-bounds index, it would cause a runtime panic when accessing `self.market.outcomes[outcome_index as usize]`, potentially halting the program.

Impact Explanation

This vulnerability could lead to a denial of service by causing the transaction to fail with a runtime error rather than a proper error message. It affects the reliability of the system but doesn't directly lead to fund loss.

Likelihood Explanation

Low likelihood as it requires a malicious user to intentionally provide an invalid index, and most legitimate clients would use valid indices.

Recommendation

Added validation to ensure the outcome index is within the valid range:

```
require!(
    outcome_index < self.market.outcomes.len() as u8,
    MarketError::InvalidOutcomeIndex
);
```

Melee: fixed in commit c81235f84af3510a9c38fe9b62b1922f33a5ca21
Lamsy: Verified

No Minimum Purchase Amount

Summary

No minimum limit was set on the `amount` parameter, allowing dust transactions.

Finding Description

The `market_buy` function accepted any non-zero amount for purchasing shares, which could lead to dust attacks where users make many tiny purchases to bloat the system with minimal economic value.

Impact Explanation

This could lead to blockchain bloat, increased storage costs, and inefficient use of computational resources. It also creates economic inefficiencies where transaction fees might exceed the value being traded.

Likelihood Explanation

Medium likelihood as there's no economic incentive to prevent users from making very small purchases that could clog the system.

Recommendation

Added a minimum purchase amount check:

```
require!(amount >= MIN_PURCHASE_AMOUNT, MarketError::AmountTooSmall);
```

And defined the constant:

```
pub const MIN_PURCHASE_AMOUNT: u64 = 1_000; // Minimum purchase amount (adjust based on token decimals)
```

Melee: fixed in commit c81235f84af3510a9c38fe9b62b1922f33a5ca21
Lamsy: Verified

Low

Unchecked Addition in Token Transfer

Summary

The code used `unwrap()` on a `checked_add` result which could panic under extreme conditions.

Finding Description

When calculating the total transfer amount (`spent + fee`), the code used `.unwrap()` which would panic if the addition overflowed. While this is extremely unlikely due to prior checks, it represents a potential vulnerability.

Impact Explanation

In the highly unlikely event of an overflow, the program would panic rather than returning a proper error, potentially causing transaction failure.

Likelihood Explanation

Very low likelihood as it would require amounts close to u64::MAX, which are economically infeasible.

Recommendation

Replaced with proper error handling:

```
let transfer_amount = spent
    .checked_add(fee)
    .ok_or(MarketError::ArithmeticOverflow)?;
```

Melee: fixed in commit c81235f84af3510a9c38fe9b62b1922f33a5ca21
Lamsy: Verified

Informational

Unchecked Addition in `update_buyer`

Summary

The `update_buyer` function used unchecked addition which could theoretically overflow.

Finding Description

When updating the buyer's shares and tokens spent, the code used standard addition operators (`+=`) rather than checked addition. While upstream checks make overflow extremely unlikely, this is inconsistent with the defensive programming approach used elsewhere.

Impact Explanation

In the highly unlikely event of an overflow, the program would silently produce incorrect results rather than returning an error.

Likelihood Explanation

Very low likelihood as it would require amounts close to u64::MAX, which are economically infeasible given the constraints of the bonding curve.

Recommendation

Use checked addition for consistency:

```
self.buyer_account.shares = self.buyer_account.shares
    .checked_add(shares)
    .ok_or(MarketError::ArithmeticOverflow)?;
self.buyer_account.tokens_spent = self.buyer_account.tokens_spent
    .checked_add(amount)
    .ok_or(MarketError::ArithmeticOverflow)?;
```

Melee: fixed in commit c81235f84af3510a9c38fe9b62b1922f33a5ca21
Lamsy: Verified

Inconsistent Logging

Summary

The code used a mix of `msg!` and `.log()` for logging, making debugging harder.

Finding Description

The logging approach was inconsistent, using both direct key logging (`self.market.to_account_info().key().log()`) and structured events. This makes it harder to parse logs and track events consistently.

Impact Explanation

This is a code quality issue that affects debugging and monitoring capabilities but doesn't introduce security vulnerabilities.

Recommendation

Standardized the logging approach by using structured events:

```
emit!(MarketBuyEvent {
    market: self.market.key(),
    buyer: self.buyer.key(),
    outcome_index,
    shares,
    spent,
    fee,
});

// For regular market claims
emit!(MarketClaimEvent {
    market: self.market.key(),
    buyer: self.buyer.key(),
    outcome_index,
    shares: self.buyer_buy_account.shares,
    winning_amount,
    is_void_market: false,
});

// For void market claims
emit!(MarketClaimEvent {
    market: self.market.key(),
    buyer: self.buyer.key(),
    outcome_index,
    shares: self.buyer_buy_account.shares,
    winning_amount: refund as u64,
    is_void_market: true,
});
```

Melee: fixed in commit c81235f84af3510a9c38fe9b62b1922f33a5ca21
Lamsy: Verified