



AUDIT REPORT

June 2025

For




vPOP

Table of Content

Executive Summary	04
Number of Security Issues per Severity	06
Summary of Issues	07
Checked Vulnerabilities	09
Techniques and Methods	11
Types of Severity	13
Types of Issues	14
Severity Matrix	15
 Critical Severity Issues	16
1. Market Creator Can Steal All User Funds by Manipulating Reveal Duration	16
2. Market Creators Can Manipulate Reward Distribution Through Improper minWager Setting	18
 High Severity Issues	19
1. Users can unintentionally lose excess ETH due to lack of refund logic in market creation	19
 Medium Severity Issues	20
1. Market Participants Can Permanently Lose Funds When No Commitments Are Revealed	20
2. Malicious User Can Manipulate Market Consensus Through Weight-Based Position Manipulation and Steal Legitimate User Funds	21
3. Contract Owner Can Cause Financial Loss to Early Committers in Whitelisted Markets	24
4. Fee-on-Transfer Tokens Can Disrupt Market Accounting	25



Table of Content

 Low Severity Issues	26
1. Callers Can Lock Ether While Using ERC20 Tokens in addWinnings Function	26
2. Contract Lacks Ether Recovery Mechanism Risking Permanent Fund Loss	27
3. Use Ownable2Step version rather than Ownable version	28
Functional Tests	29
Threat Model	31
Automated Tests	32
Closing Summary & Disclaimer	32



Executive Summary

Project Name	VPOP
Protocol Type	Prediction Market
Project URL	https://vpop.wtf/
Overview	<p>VPOP is a prediction market contract that allows users to create markets around numerical predictions within specified bounds. The system operates in two phases: a commit phase where users submit hashed predictions with wagers, and a reveal phase where they disclose their actual positions. Winners are determined by proximity to the weighted consensus position, with a configurable winning percentile threshold. The contract supports both ETH and ERC20 tokens, implements whitelist functionality via Merkle proofs, and charges platform, creator, and "ape" fees on wagers. Markets can be permissioned (owner-only creation) or public, with optional market creation fees. The consensus mechanism calculates a weighted average of all revealed positions, with earlier commits receiving higher weights based on a decay factor.</p>
Audit Scope	The scope of this Audit was to analyze the VPOP Smart Contracts for quality, security, and correctness.
Source Code link	https://github.com/internetmoney-gg/apepop/blob/main/contracts/vpop.sol
Branch	Main
Contracts in Scope	vpop.sol
Commit Hash	35d12486b1ffcf0f62402a4cf59d532094788f71
Language	Solidity
Blockchain	Ethereum , Base
Method	Manual Analysis, Functional Testing, Automated Testing



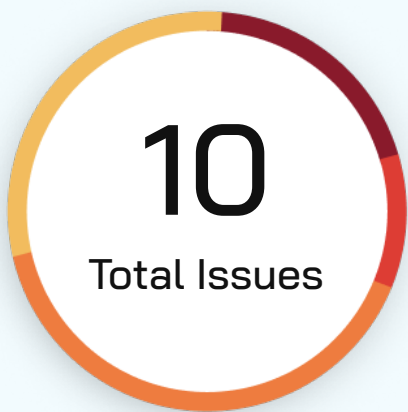
Review 1	17th June 2025 - 27th June 2025
Updated Code Received	30th June 2025
Review 2	30th June 2025
Fixed In	a0d9cfb7c69356787c73d0357206641034ccfd70

Verify the Authenticity of Report on QuillAudits Leaderboard:

<https://www.quillaudits.com/leaderboard>



Number of Issues per Severity



Critical	2 (20%)
High	1 (10%)
Medium	4 (40%)
Low	3 (30%)
Informational	0 (0%)

		Severity				
		Critical	High	Medium	Low	Informational
Issues	Open	0	0	0	0	0
	Acknowledged	0	0	2	1	0
	Partially Resolved	0	0	0	0	0
	Resolved	2	1	2	2	0



Summary of Issues

Issue No.	Issue Title	Severity	Status
1	Market Creator Can Steal All User Funds by Manipulating Reveal Duration	Critical	Resolved
2	Market Creators Can Manipulate Reward Distribution Through Improper minWager Setting	Critical	Resolved
3	Users can unintentionally lose excess ETH due to lack of refund logic in market creation	High	Resolved
4	Market Participants Can Permanently Lose Funds When No Commitments Are Revealed	Medium	Acknowledged
5	Malicious User Can Manipulate Market Consensus Through Weight-Based Position Manipulation and Steal Legitimate User Funds	Medium	Acknowledged
6	Contract Owner Can Cause Financial Loss to Early Committers in Whitelisted Markets	Medium	Resolved
7	Fee-on-Transfer Tokens Can Disrupt Market Accounting	Medium	Resolved
8	Callers Can Lock Ether While Using ERC20 Tokens in addWinnings Function	Low	Resolved



Issue No.	Issue Title	Severity	Status
9	Contract Lacks Ether Recovery Mechanism Risking Permanent Fund Loss	Low	Resolved
10	Use Ownable2Step version rather than Ownable version	Low	Acknowledged



Checked Vulnerabilities

✓ Access Management

✓ Arbitrary write to storage

✓ Centralization of control

✓ Ether theft

✓ Improper or missing events

✓ Logical issues and flaws

✓ Arithmetic Computations
Correctness

✓ Race conditions/front running

✓ SWC Registry

✓ Re-entrancy

✓ Timestamp Dependence

✓ Gas Limit and Loops

✓ Exception Disorder

✓ Gasless Send

✓ Use of tx.origin

✓ Malicious libraries

✓ Compiler version not fixed

✓ Address hardcoded

✓ Divide before multiply

✓ Integer overflow/underflow

✓ ERC's conformance

✓ Dangerous strict equalities

✓ Tautology or contradiction

✓ Return values of low-level calls



✓ Missing Zero Address Validation

✓ Private modifier

✓ Revert/require functions

✓ Multiple Sends

✓ Using suicide

✓ Using delegatecall

✓ Upgradeable safety

✓ Using throw

✓ Using inline assembly

✓ Style guide violation

✓ Unsafe type inference

✓ Implicit visibility level



Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code
- Use of best practices
- Code documentation and comments, match logic and expected behavior
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper
- Implementation of ERC standards
- Efficient use of gas
- Code is safe from re-entrancy and other vulnerabilities

The following techniques, methods, and tools were used to review all the smart contracts:

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.



Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms Used for Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity Static Analysis.



Types of Severity

Every issue in this report has been assigned to a severity level. There are five levels of severity, and each of them has been explained below.

■ **Critical: Immediate and Catastrophic Impact**

Critical issues are the ones that an attacker could exploit with relative ease, potentially leading to an immediate and complete loss of user funds, a total takeover of the protocol's functionality, or other catastrophic failures. Critical vulnerabilities are non-negotiable; they absolutely must be fixed.

■ **High (H): Significant Risk of Major Loss or Compromise**

High-severity issues represent serious weaknesses that could result in significant financial losses for users, major malfunctions within the protocol, or substantial compromise of its intended operations. While exploiting these vulnerabilities might require specific conditions to be met or a moderate level of technical skill, the potential damage is considerable. These findings are critical and should be addressed and resolved thoroughly before the contract is put into the Mainnet.

■ **Medium (M): Potential for Moderate Harm Under Specific Circumstances**

Medium-severity bugs are loopholes in the protocol that could lead to moderate financial losses or partial disruptions of the protocol's intended behavior. However, exploiting these vulnerabilities typically requires more specific and less common conditions to occur, and the overall impact is generally lower compared to high or critical issues. While not as immediately threatening, it's still highly recommended to address these findings to enhance the contract's robustness and prevent potential problems down the line.

■ **Low (L): Minor Imperfections with Limited Repercussions**

Low-severity issues are essentially minor imperfections in the smart contract that have a limited impact on user funds or the core functionality of the protocol. Exploiting these would usually require very specific and unlikely scenarios and would yield minimal gain for an attacker. While these findings don't pose an immediate threat, addressing them when feasible can contribute to a more polished and well-maintained codebase.

■ **Informational (I): Opportunities for Improvement, Not Immediate Risks**

Informational findings aren't security vulnerabilities in the traditional sense. Instead, they highlight areas related to the clarity and efficiency of the code, gas optimization, the quality of documentation, or adherence to best development practices. These findings don't represent any immediate risk to the security or functionality of the contract but offer valuable insights for improving its overall quality and maintainability. Addressing these is optional but often beneficial for long-term health and clarity.



Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Severity Matrix

		Impact		
		High	Medium	Low
Likelihood	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

Likelihood

- **High** - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- **Medium** - only a conditionally incentivized attack vector, but still relatively likely.
- **Low** - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.



Critical Severity Issues

Market Creator Can Steal All User Funds by Manipulating Reveal Duration

Resolved

Path

contracts/vpop.sol

Function Name

initializeMarket()

Description

The VPOP contract contains a critical vulnerability in the market creation and resolution mechanism that allows malicious market creators to steal all user funds by manipulating the reveal duration parameter. When creating a market through the initializeMarket function, creators can set an extremely short `_revealDuration` (such as 5 seconds) while setting a reasonable `_commitDuration` to attract users. This creates a scenario where users have sufficient time to commit their wagers during the commitment phase, but an impossibly short window to reveal their commitments.

The vulnerability manifests in the resolve function's logic, which allows resolution when either all commitments are revealed or the reveal phase has ended. The critical flaw lies in the condition `bool revealPhaseEnded = block.timestamp > market.createdAt + market.commitDuration + market.revealDuration` combined with the requirement `require(allRevealed || revealPhaseEnded, "Market not ready for resolution")`. A malicious creator can exploit this by being the only participant who successfully reveals their commitment within the artificially short reveal window, then immediately calling resolve to become the sole winner eligible to claim the entire prize pool through the claim function.

The contract calculates winnings based on the proportion of winning wagers using `winnings = Math.mulDiv(commitment.wager, consensus.totalWinnings, consensus.winningWagers)`. When only the creator's commitment is revealed and wins, they receive 100% of the totalWinnings, which represents the sum of all user wagers minus fees. This mechanism effectively allows creators to siphon funds from legitimate users who are unable to reveal their commitments due to the manipulated timing.

Impact

This vulnerability enables market creators to execute sophisticated rug pulls with guaranteed success. The financial impact is severe, as creators can drain the entire prize pool accumulated from user wagers. For example, if 100 users each wager 1 ETH in a market with a 1000 ETH total prize pool, a malicious creator wagering just 1 ETH can steal the entire 880 ETH remaining after fees (assuming 12% total fees) by manipulating the reveal duration to 5 seconds, revealing their own commitment within this window, and immediately resolving the market as the sole winner.



Recommendation

The primary remediation requires implementing a minimum reveal duration constraint in the `initializeMarket` function to prevent creators from setting unreasonably short reveal periods.

Specific Fixed In Commit

[a0d9cfb](#)

Protocol team's Response

Fixed by setting minimum reveal duration to 30 minutes (1800 seconds)

QuillAudits team's Response

This Issue has been fixed properly.



Market Creators Can Manipulate Reward Distribution Through Improper minWager Setting

Resolved

Path

contracts/vpop.sol

Function Name

initializeMarket()

Description

The vulnerability stems from an incorrect coupling between the market creation parameters and reward distribution logic in the VPOP contract. During market initialization via the initializeMarket function, creators can set `_minWager` to zero for non-whitelisted markets. This parameter setting improperly triggers equal distribution of rewards in the claim function through the conditional branch if `(market.minWager > 0)`, despite the market not being an actual whitelisted market (which would be properly indicated by `whitelistRoots[marketId] != bytes32(0)`).

The core issue arises from the contract's assumption that zero minimum wager automatically indicates a whitelisted market, when in reality this should be explicitly verified against the whitelist root hash. This flawed assumption allows malicious creators to artificially force equal reward distribution regardless of participants' actual wager amounts, breaking the intended proportional reward mechanism.

Impact

This vulnerability enables market creators to systematically distort the reward distribution mechanism in their favor. By setting `minWager` to zero for non-whitelisted markets, creators can ensure all winning participants receive equal payouts regardless of their individual contributions. This manipulation particularly benefits creators who may wager minimal amounts while receiving equal shares of the total rewards pool.

The financial impact scales directly with the market size, potentially allowing creators to siphon significant value from legitimate participants.

Recommendation

The solution requires decoupling the reward distribution logic from the minimum wager parameter and properly tying it to the actual whitelist status. The claim function should be modified to explicitly check the whitelist status through the root hash rather than inferring it from the minimum wager. The proportional reward calculation should become the default case, with equal distribution reserved only for properly configured whitelisted markets.

Specific Fixed In Commit

[a0d9cfb](#)

Protocol team's Response

Fixed by explicitly checking whitelist status through root hash in the claim function, as per recommendation.

QuillAudits team's Response

This Issue has been fixed properly.



High Severity Issues

Users can unintentionally lose excess ETH due to lack of refund logic in market creation

Resolved

Path

contracts/vpop.sol

Function

initializeMarket()

Description

In the initializeMarket function of the contract, users are allowed to create a prediction market by submitting a transaction along with an ETH payment (msg.value) to cover the marketCreateFee. The contract verifies that msg.value is at least equal to marketCreateFee, and if so, transfers exactly marketCreateFee to the contract owner. However, the function does not contain any logic to refund any excess ETH sent by mistake—i.e., when msg.value > marketCreateFee. This can lead to unintentional loss of funds by users who miscalculate the fee

Impact

Users who send more than the required marketCreateFee during the initializeMarket call will permanently lose the excess ETH, resulting in unintended donations to the contract.

Remediation

To ensure proper fund handling, the function should explicitly refund any excess ETH to the sender after transferring the marketCreateFee to the owner.

Specific Fixed In Commit

[a0d9cfb](#)

Protocol team's Response

Fixed by refunding excess ETH

QuillAudits team's Response

This Issue has been fixed properly.



Medium Severity Issues

Market Participants Can Permanently Lose Funds When No Commitments Are Revealed

Acknowledged

Path

`src/vpop.sol`

Function

`reveal()`

Description

The vulnerability manifests in the market state transition logic within the `resolve()` function, which contains a strict requirement that at least one commitment must be revealed (`require(consensus.revealedCommitments > 0)`). This creates a protocol deadlock when the reveal period concludes without any participant successfully revealing their position. The condition arises because the contract lacks alternative state transition paths or fallback mechanisms to handle this edge case. The market becomes permanently stuck in an unresolved state, with all participant funds remaining locked in contract storage but inaccessible through normal protocol operations. This failure mode is particularly concerning because it can occur organically through normal protocol use when participants forget to reveal, encounter technical issues, or simply choose not to participate in the reveal phase.

Protocol Team Response

Due to autoreveal functionality in our dApp, this is trivial and should never occur.

Malicious User Can Manipulate Market Consensus Through Weight-Based Position Manipulation and Steal Legitimate User Funds

Acknowledged

Path

contracts/vpop.sol

Function

commit()

Description

The VPOP contract contains a critical vulnerability in its consensus mechanism that allows malicious users to manipulate market outcomes through strategic high-value commitments at extreme positions. The vulnerability stems from the weighted consensus calculation logic in the reveal function, where the consensus position is determined by $\text{consensusPosition} = \text{weightedSum} / \text{totalWeight}$. This calculation gives disproportionate influence to users who commit larger wagers, as their weight directly correlates to their wager amount through the formula $\text{weight} = \text{wager} * (10000 - \text{decay}) / 10000$.

The attack vector exploits markets with low participation and minimal decay factors. A malicious actor can commit an extremely large wager (such as 5x the total existing wagers) at an extreme position like the upperBound or lowerBound. When they reveal their commitment, their massive weight shifts the consensus position dramatically toward their chosen extreme. The consensus calculation $\text{marketConsensus}[\text{marketId}].\text{weightedSum} += \text{position} * \text{commitment.weight}$ means their large weight multiplied by the extreme position value dominates the weighted sum, effectively controlling the final consensus position.

Once the malicious user controls the consensus position, they can ensure their commitment wins by being closest to the manipulated consensus. The resolution logic in the resolve function determines winners based on distance from the consensus position through $\text{distance} = |\text{commitment.position} - \text{consensus.consensusPosition}|$. By artificially shifting the consensus to their extreme position, the attacker guarantees they have the smallest distance and thus qualify as a winner. They can then claim a disproportionate share of the total winnings through the claim function, which distributes prizes based on the proportion of winning wagers.

Impact

Legitimate users who make thoughtful predictions based on actual market analysis lose their funds to manipulative actors who exploit the weight-based consensus system. The attack can be repeated across multiple markets, systematically draining funds from honest participants and destroying platform credibility.

Numeric Example

Market Setup (Optimized for Attack)



Market Bounds: lowerBound = 0, upperBound = 10000
 Decay Factor: 200 (2% decay factor - very low)
 Commit Duration: 7200 seconds (2 hours)
 Platform Fee: 8% (800 basis points)
 Creator Fee: 2% (200 basis points)
 Ape Fee: 2% (200 basis points)
 Total Fees: 12%
 Winning Percentile: 2000 (20% - higher percentage of winners)

Legitimate Users' Commitments (Small Scale Market)

5 legitimate users each commit different amounts at clustered positions (typical for prediction markets):

User	Wager	Position	Commit Time	Weight (ETH)
User1	0.5 ETH	4800	0s	0.5 ETH
User2	0.8 ETH	4900	1440s	0.7968 ETH
User3	1.2 ETH	5000	2880s	1.1904 ETH
User4	0.7 ETH	5100	4320s	0.6916 ETH
User5	0.3 ETH	5200	5760s	0.2952 ETH

Legitimate Users' Financial Impact

Fee calculations per user:

User1: Platform: 0.04, Creator: 0.01, Ape: 0.01 → Net: 0.44 ETH
 User2: Platform: 0.064, Creator: 0.016, Ape: 0.016 → Net: 0.704 ETH
 User3: Platform: 0.096, Creator: 0.024, Ape: 0.024 → Net: 1.056 ETH
 User4: Platform: 0.056, Creator: 0.014, Ape: 0.014 → Net: 0.616 ETH
 User5: Platform: 0.024, Creator: 0.006, Ape: 0.006 → Net: 0.264 ETH

Total legitimate investment: 3.5 ETH

Total legitimate prize pool: $0.44 + 0.704 + 1.056 + 0.616 + 0.264 = 3.08$ ETH

Legitimate Consensus Calculation (Before Attack)

Total Weight: $0.5 + 0.7968 + 1.1904 + 0.6916 + 0.2952 = 3.4740$ ETH

Weighted Sum:

User1: $4800 \times 0.5 = 2400$
 User2: $4900 \times 0.7968 = 3904.32$
 User3: $5000 \times 1.1904 = 5952$
 User4: $5100 \times 0.6916 = 3527.16$
 User5: $5200 \times 0.2952 = 1535.04$
 Total: $2400 + 3904.32 + 5952 + 3527.16 + 1535.04 = 17,318.52$

Natural Consensus: $17,318.52 / 3.4740 = 4,986.34 \approx 4986$

The attacker observes this small, clustered market and executes a precision attack:

Wager: 10 ETH (approximately 2.86x the total legitimate wagers)
 Position: 0 (lowerBound - maximum distance from cluster)
 Commit Time: 7180s (20 seconds before commit phase ends)
 Elapsed Time: 7180s

Attacker's Weight Calculation

decay = $(200 \times 7180) / 7200 = 199.44 \approx 199$

weight = $10 \times (10000 - 199) / 10000 = 10 \times 9801 / 10000 = 9.801$ ETH

Attacker's Financial Impact

Platform fee: $10 \times 800/10000 = 0.8$ ETH

Creator fee: $10 \times 200/10000 = 0.2$ ETH

Ape fee: $10 \times 200/10000 = 0.2$ ETH

Net to prize pool: $10 - 1.2 = 8.8$ ETH

Total prize pool after attack: $3.08 + 8.8 = 11.88$ ETH

Manipulated Consensus Calculation

New Totals After Attack

Total Weight: $3.4740 + 9.801 = 13.275$ ETH

New Weighted Sum: $17,318.52 + (0 \times 9.801) = 17,318.52 + 0 = 17,318.52$

Manipulated Consensus: $17,318.52 / 13.275 = 1,305.01 \approx 1305$

Winning Threshold Analysis

Since only the attacker has a reasonable distance (1305) and all legitimate users are extremely far (3400+), the attacker will be the primary winner regardless of the exact threshold.

For 2 winners, the threshold would need to be set around 3495 to include User1, but this gives the attacker an even larger advantage.

Optimal threshold for attacker: Any value ≥ 1305 but < 3495

Financial Impact Analysis

Scenario 1: Threshold = 3500 (2 winners: Attacker + User1)

Winning wagers: 10 (Attacker) + 0.5 (User1) = 10.5 ETH

Prize distribution:

User1: $(0.5/10.5) \times 11.88 = 0.567$ ETH

Attacker: $(10/10.5) \times 11.88 = 11.314$ ETH

Scenario 2: Threshold = 1305 (1 winner: Attacker only)

Winning wagers: 10 ETH (Attacker only)

Prize distribution:

Attacker: $(10/10) \times 11.88 = 11.88$ ETH

Recommendation

The remediation requires implementing comprehensive logic changes to prevent weight concentration attacks or we should increase the decay factor such that an attacker performing this attack can never get profitable.

Protocol team;s Response

Such an attack requires the attacker to risk their full wager. The only way to guarantee that the attacker's bet will be within the winning percentile without knowing the positions of the other bets is to set $\text{wager} = \text{pot size} * \text{range}$ (number of possible positions). This is only an issue in trivial edge cases where pot size and range are very low, making this feasible.

QuillAudits team;s Response

This issue can happen during low pot size and when decay factor is not that big and there is high risk for the attacker as well.



Contract Owner Can Cause Financial Loss to Early Committers in Whitelisted Markets

Resolved

Path

src/vpop.sol

Function

reveal()

Description

The vulnerability arises in the interaction between three key functions: `initializeMarket()`, `updateWhitelistRoot()`, and `commit()`. When a whitelisted market is created, the contract initially has no whitelist root set. Users can commit to these markets before the owner sets the whitelist root via `updateWhitelistRoot()`. In whitelisted markets, the `commit()` function enforces a fixed wager of 100,000 units and bypasses normal fee distribution, instead splitting rewards equally among all qualifying participants. However, if the owner updates the whitelist root after some users have already committed, these early committers would face loss as the mechanism in `resolve()` divides winnings equally among qualifying commitments, creating an unfair distribution where some participants lose funds while others benefit disproportionately. So to prevent this transaction should be bundled should be used.

Protocol Team Response

Atomic bundle will be used when creating whitelist markets to prevent this scenario.



Fee-on-Transfer Tokens Can Disrupt Market Accounting

Resolved

Path

src/vpop.sol

Function

commit()

Description

The VPOP contract allows users to create markets with arbitrary ERC20 tokens, including fee-on-transfer tokens (tokens that deduct a fee on transfers). The issue arises in the commit() function, where the contract assumes the received token amount matches the wager value. However, if a fee-on-transfer token is used, the actual balance received by the contract will be less than the wager due to the transfer fee.

The contract calculates fees (platformFee, creatorFee, apeFee) and transfers them based on the wager amount, but the remaining balance (winnings) is stored in marketConsensus[marketId].totalWinnings. If the token deducts a fee, the contract's balance will be less than expected, leading to incorrect accounting. Later, during claim(), users may receive incorrect payouts or transactions may fail due to insufficient balance.

Specific Fixed In Commit

[a0d9cfb](#)

Protocol team's Response

Fixed by checking contract balance before and after transfer, and setting the actual wager to balanceAfter - balanceBefore, accounting for fee on transfer tokens.



Low Severity Issues

Callers Can Lock Ether While Using ERC20 Tokens in addWinnings Function

Resolved

Path

src/vpop.sol

Function

addWinnings()

Description

The addWinnings function allows users to contribute additional winnings to a market, supporting both ERC20 tokens and native Ether. However, the function fails to validate whether msg.value is zero when processing ERC20 token transfers. This oversight could lead to accidental or malicious locking of Ether in the contract when users intend to only transfer ERC20 tokens.

The issue arises because the function checks msg.value only in the Ether transfer branch (market.token == address(0)), but does not enforce msg.value == 0 when executing ERC20 transfers

Specific Fixed In Commit

[a0d9cfb](#)

Protocol team's Response

Fixed by requiring msg.value == 0 in addWinnings for ERC20 based markets.



Contract Lacks Ether Recovery Mechanism Risking Permanent Fund Loss

Resolved

Path

src/vpop.sol

Function

constructor()

Description

The VPOP contract's constructor is payable, allowing it to receive Ether during deployment. However, the contract lacks functionality to withdraw or utilize these funds. Consequently, any Ether sent during deployment becomes permanently locked in the contract.

Specific Fixed In Commit

[a0d9cfb](#)

Protocol team's Response

Fixed by making constructor non-payable.



Use Ownable2Step version rather than Ownable version

Acknowledged

Path

src/vpop.sol

Function

constructor()

Description

Ownable2Step prevent the contract ownership from mistakenly being transferred to an address that cannot handle it (e.g. due to a typo in the address), by requiring that the recipient of the owner's permissions actively accept via a contract call of its own.

Consider using Ownable2Step from OpenZeppelin Contracts to enhance the security of your contract ownership management. This contract prevents the accidental transfer of ownership to an address that cannot handle it, such as due to a typo, by requiring the recipient of owner permissions to actively accept ownership via a contract call.

Protocol team's Response

Ownership will be handled carefully by team via multi-sig.

Functional Tests

Some of the tests performed are mentioned below:

- ✓ Initial owner should be set correctly
- ✓ Initial fee rates should be set correctly
- ✓ Only owner can update platform settings
- ✓ Platform settings update should change fee rates correctly
- ✓ Only owner can update whitelist root
- ✓ Market creation fee should be transferred to owner when required
- ✓ Market creation should store all parameters correctly
- ✓ Commitment should fail with wager below minimum
- ✓ Commitment should verify whitelist when required
- ✓ Commitment should transfer correct fees for ETH markets
- ✓ Commitment should transfer correct fees for ERC20 markets
- ✓ Commitment should calculate weight with decay correctly
- ✓ Reveal should fail outside reveal phase
- ✓ Reveal should fail for already revealed commitment
- ✓ Reveal should fail with invalid position
- ✓ Reveal should verify commitment hash matches
- ✓ Reveal should update market consensus correctly
- ✓ Resolution should fail before reveal phase ends
- ✓ Resolution should fail if no commitments revealed
- ✓ Resolution should calculate consensus position correctly
- ✓ Resolution should set winning threshold correctly
- ✓ Resolution should mark market as resolved
- ✓ Resolution should calculate winning wagers correctly
- ✓ Claiming Tests
- ✓ Claim should fail for non-existent market



- ✓ Claim should fail for unresolved market
- ✓ Claim should fail for unrevealed commitment
- ✓ Claim should fail for non-winning position
- ✓ Claim should fail for already claimed commitment
- ✓ Claim should transfer correct ETH winnings
- ✓ Claim should transfer correct ERC20 winnings
- ✓ Claim should mark commitment as claimed
- ✓ addWinnings should increase total winnings for ETH
- ✓ addWinnings should increase total winnings for ERC20
- ✓ addWinnings should fail with incorrect ETH amount
- ✓ Market state transitions should follow correct timeline
- ✓ Multiple commitments should affect consensus correctly
- ✓ Edge cases in weight calculation should be handled



Threat Model

Contract	Function	Threats
VPOP	updatePlatformSettings	Fee manipulation by owner Front-running attacks when changing fee rates
	updateWhitelistRoot	Front-running attacks when changing whitelist Invalid Merkle root could block legitimate users
	addWinnings	Incorrect ETH amount validation
	initializeMarket	Parameter manipulation Malicious IPFS hash injection Reveal duration manipulation
	commit	Commitment phase timing attacks Fee calculation errors
	reveal	Reveal phase timing attacks Position out-of-bounds attacks Weighted sum calculation errors
	resolve	Consensus position calculation errors Griefing by resolving with no reveals
	claim	Incorrect winnings calculation



Automated Tests

No major issues were found. Some false-positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of VPOP. We performed our audit according to the procedure described above.

Issues of Critical , High , Medium and Low severity were found. Vpop team acknowledged three and resolved the rest of the issues

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.



About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over \$30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

**7+**

Years of Expertise

1M+

Lines of Code Audited

\$30B+

Secured in Digital Assets

1400+

Projects Secured

Follow Our Journey

AUDIT REPORT

June 2025

For



vPOP



Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillaudits.com