Stakep2p Contracts Security Audit Report

Security Researchers

kenzo, 0xrex

Prepared by: Shred Security

29/09/2025

Table of Contents

- Table of Contents
- About Shred Security
- Protocol Executive Summary
- Disclaimer
- Risk Classification
- Executive Summary
- Findings

About Shred Security

Shred Security provides high quality security audits for blockchain and DeFi protocols across different chains. Our audits consistently uncover high-impact vulnerabilities missed by others, backed by a proven track record of top competition placements and security partnerships with leading protocols.

Learn more about us: shredsec.xyz

Protocol Executive Summary

The P2PSVault contract is a peer-to-peer betting protocol built on Ethereum, allowing users to create bets, place stakes on "YES" or "NO" outcomes, and claim winnings based on resolutions by a designated publisher. It supports public and buddy (eligible addresses) bets, with fees deducted on placements.

Disclaimer

The Shred Security team makes all effort to find as many vulnerabilities in the code in the given time period. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and this report neither endorses any specific project or team nor assures the complete security of the project.

Risk Classification

Likelihood \ Impact	High	Medium	Low
High	High	High/Medium	Medium
Medium	High/Medium	Medium	Medium/Low
Low	Medium	Medium/Low	Low

Executive Summary

The shred security team has conducted the review for 5 days in total. In this period of time, a total of 19 issues were found.

About the Project

Project Name	Stakep2p Contracts
Repository	https://github.com/stakep2p/stake_contracts/tree/main/src
Commit	469b593ff17c
Type of Project	DeFi Protocol, Staking
Lines of Code	300

Audit Timeline

Audit Start	21/09/2025
Audit End	26/09/2025
Report Published	29/09/2025

Vulnerability Summary

Severity	Count	Fixed	Acknowledged
High Risk	2	2	2
Medium Risk	7	3	4
Low Risk	8		8
Informational	2		2
Total	19	5	14

Findings Summary

Issue ID	Description	Severity	Status
H-1	Stuck Funds if Winning Side Has Zero Bets	High	Acknowledged
H-2	Bypass of Access Control for Public Bet Creation	High	Fixed
M-1	Owner Can Effectively Lock Funds by Changing betToken	Medium	Fixed
M-2	No Fee Refund in Draw Outcomes	Medium	Fixed
M-3	Lack of Bet Cancellation or Early Withdrawal	Medium	Acknowledged
M-4	Uninitialized Variables Leading to Potential Failures	Medium	Acknowledged
M-5	Unlimited Buddy Bet Creation Allows Storage Bloating	Medium	Fixed
M-6	With Large `eligibleAddresses`, the `placeBet` might face DOS	Medium	Acknowledged
M-7	No Oracle Integration for bet resolution	Medium	Acknowledged
L-1	Incompatibility with Fee-on-Transfer Tokens	Low	Acknowledged
L-2	Overwriting of betToCreator Mapping	Low	Acknowledged
L-3	Precision Loss and Dust Accumulation	Low	Acknowledged
L-4	No Check for Bet Overlap or Duplicates	Low	Acknowledged
L-5	Unused Variables and Mappings Waste Gas and Storage	Low	Acknowledged
L-6	Redundant Data Storage in BetPlacement	Low	Acknowledged
L-7	betAmounts Not Cleared for Losers in claimWinning	Low	Acknowledged
L-8	No Bulk Claim Function	Low	Acknowledged
I-1	Inconsistent Fee Scaling	Informational	Acknowledged
I-2	Lack of ERC20 Approval Checks	Informational	Acknowledged

Findings

High Severity

[H-1] Stuck Funds if Winning Side Has Zero Bets

Severity Assesment
Impact: High
Likelihood: Medium

Context
In `claimWinning` and bet resolution logic.

Description
If all bets are on one side (e.g., "YES") and the publisher resolves to the opposite side ("NO") with zero volume, the losing side's funds are not distributed (no winners to claim). These funds remain in the contract forever, as there's no sweep function or automatic refund.

```
function claimWinning(uint256 betId) public {
    // ...
    if (bets[betId].outcome == keccak256(abi.encodePacked(side))
|| bets[betId].outcome == keccak256(abi.encodePacked("DRAW"))) {
        // Winner or DRAW: Transfer funds
        uint256 amount = betAmounts[betId][msg.sender][side];
        if (bets[betId].outcome ==
keccak256(abi.encodePacked("DRAW"))) {
            pendingWithdrawals[msg.sender] += amount;
        } else {
            uint256 winningAmount = totalBetAmounts[betId][side];
            uint256 losingAmount = totalBetAmounts[betId]
[oppositeSide[side]];
            uint256 userSharePercentage = (amount * 1e18) /
winningAmount;
            uint256 userWinnings = (userSharePercentage *
losingAmount) / 1e18;
            pendingWithdrawals[msg.sender] += amount +
userWinnings;
        }
        betAmounts[betId][msg.sender][side] = 0;
    }
```

Impact
Permanent loss of user funds in unbalanced bets.

Proof of Concepts
Add this test to `P2PSVault.t.sol`

```
    function test_stuckFundsIfZeroBetsOnWinningSide() public {
        test_createPublicBet();

        // Transfer tokens to users and place bets only on YES
        vm.startPrank(owner);
        USDC.transfer(address(111111), 1e18);
        USDC.transfer(address(333333), 3e18);
        USDC.transfer(address(555555), 5e18);
        vm.stopPrank();

        vm.startPrank(address(111111));
        USDC.approve(address(p2pSVault), 1e18);
        p2pSVault.placeBet(0, 1e18, "YES");
        vm.stopPrank();

        vm.startPrank(address(333333));
        USDC.approve(address(p2pSVault), 3e18);
        p2pSVault.placeBet(0, 3e18, "YES");
        vm.stopPrank();

        vm.startPrank(address(555555));
        USDC.approve(address(p2pSVault), 5e18);
        p2pSVault.placeBet(0, 5e18, "YES");
        vm.stopPrank();

        // Verify total bets: YES = 9e18, NO = 0
        uint256 totalYesBets = p2pSVault.totalBetAmounts(0,
"YES");
        uint256 totalNoBets = p2pSVault.totalBetAmounts(0, "NO");
        console.log("TOTAL YES BET: ", totalYesBets);
        console.log("TOTAL NO BET: ", totalNoBets);
        assertEq(totalYesBets, 9e18 * 975 / 1000, "Incorrect YES
bets total");
        assertEq(totalNoBets, 0, "NO bets should be zero");

        // Contract balance should reflect all bets (9e18)
        uint256 contractBalanceBefore =
USDC.balanceOf(address(p2pSVault));
        console.log("CONTRACT BAL AFTER BETS: ",
contractBalanceBefore);
        assertEq(contractBalanceBefore, 9e18, "Incorrect contract
balance");

        // Resolve bet to NO (zero volume side)
        vm.warp(block.timestamp + 3 days + 1 minutes);
        vm.prank(publisher);
        p2pSVault.resolveBet(0, "NO");

        // Attempt claims by YES bettors (losers)
        vm.startPrank(address(111111));
        p2pSVault.claimWinning(0); // No transfer (not winner, not
```

```
DRAW)
        vm.stopPrank();

        vm.startPrank(address(333333));
        p2pSVault.claimWinning(0); // No transfer
        vm.stopPrank();

        vm.startPrank(address(555555));
        p2pSVault.claimWinning(0); // No transfer
        vm.stopPrank();

        // Verify no funds withdrawn by losers
        assertEq(USDC.balanceOf(address(111111)), 0, "User 1
should have no balance");
        assertEq(USDC.balanceOf(address(333333)), 0, "User 3
should have no balance");
        assertEq(USDC.balanceOf(address(555555)), 0, "User 5
should have no balance");

        // Verify contract still holds funds (9e18)
        uint256 contractBalanceAfter =
USDC.balanceOf(address(p2pSVault));
        console.log("CONTRACT BAL AFTER CLAIMS: ",
contractBalanceAfter);
        assertEq(contractBalanceAfter, 9e18, "Funds should remain
stuck in contract");

        // Fee claim by owner (only fees: 9e18 * 25/1000 =
0.225e18)
        vm.prank(owner);
        p2pSVault.claimFee();
        console.log("FEE COLLECTOR BAL: ",
USDC.balanceOf(address(feeCollector)));
        assertEq(USDC.balanceOf(address(feeCollector)), 9e18 * 25
/ 1000, "Incorrect fee collected");

        // Contract still holds net amounts (9e18 * 975/1000 =
8.775e18)
        uint256 finalContractBalance =
USDC.balanceOf(address(p2pSVault));
        console.log("CONTRACT BAL AFTER FEE CLAIM: ",
finalContractBalance);
        assertEq(finalContractBalance, 9e18 * 975 / 1000, "Net
amounts should remain stuck");
    }
```

Recommended mitigation

Modify the claimWinning function to refund the net bet amount to losers if the winning side has zero volume (i.e., totalBetAmounts[betId][oppositeSide[side]] == 0), treating it like a DRAW for those users.

Client Reponse
Fixed

[H-2] Bypass of Access Control for Public Bet Creation

Severity Assesment
Impact: High
Likelihood: High

Context
`createBet` vs `createBuddyBet` functions.

Description
The `createBet` function is restricted to the `betInitiator` address via the `onlyBetInitiator` modifier, intended for controlled creation of public bets (with empty `eligibleAddresses`). However, any user can call `createBuddyBet` with an empty `eligibleAddresses` array, which effectively creates a public bet since `placeBet` only checks eligibility if `eligibleAddresses.length > 0`. This bypasses the initiator restriction, allowing uncontrolled proliferation of public bets.

Impact
Uncontrolled creation of public bets, potential spam or misuse.

Proof of Concepts
User calls createBuddyBet with empty array, places bet without eligibility check.

Recommended mitigation
Add check in createBuddyBet for non-empty eligibleAddresses.

Client Reponse
Fixed

Medium Severity

[M-1] Owner Can Effectively Lock Funds by Changing betToken

Severity Assesment
Impact: Medium
Likelihood: Medium

Context
The `setBetToken` function can be called by the owner at any time.

Description
The owner can call `setBetToken` at any time, even after bets are placed and funds deposited in the old token. Subsequent claims use the new `betToken` for transfers, but the contract holds balances in the old token, locking user winnings. This leaves old tokens stuck (no sweep mechanism) and could fail claims or transfer unintended tokens if the contract somehow acquires the new token.

Impact
Permanent lock of user funds in the contract, as claims will attempt to transfer the wrong token.

Proof of Concepts
N/A (logical issue). Simulate: Deposit with token A, change to token B, try to claim - transfers token B which contract has 0 balance.

Recommended mitigation
Make `betToken` immutable after initial set or add a token sweep/migration function. Recommend making `setBetToken` one-time (e.g., require `betToken == address(0)`).

Client Reponse
Fixed

[M-2] No Fee Refund in Draw Outcomes

Severity Assesment
Impact: Medium
Likelihood: Medium

Context
In `claimWinning` for DRAW.

Description
In a "DRAW" resolution, users are refunded only their net bet amount (after fees). The fee portion is retained by the contract and added to `pendingFee`. This may be unfair in scenarios where no clear winner/loser exists, as the house still profits despite no resolution. Protocol fees are deducted and kept even on DRAW, where principals are refunded (net amounts only).

Impact
Unfair fee retention in unresolved bets.

Proof of Concepts
Place bet, pay fee, resolve DRAW, get back netAmount, fee kept.

Recommended mitigation
Refund full amount including fee in DRAW.

Client Reponse
Fixed

[M-3] Lack of Bet Cancellation or Early Withdrawal

Severity Assesment
Impact: Medium
Likelihood: Medium

Context
Bet placement and resolution flow.

Description
Once placed, bets are locked until resolution. There's no mechanism for users to withdraw before the duration ends or for creators to cancel bets. This could lead to stuck funds if events change or errors occur.

Impact
Funds stuck in unwanted bets.

Proof of Concepts
User places bet, event cancels, no way to withdraw.

Recommended mitigation
Add withdrawal function with penalty or creator cancel.

Client Reponse
Acknowledged

[M-4] Uninitialized Variables Leading to Potential Failures

Severity Assesment
Impact: Medium
Likelihood: High

Context
Contract constructor and setters.

Description
`betToken`, `publisher`, `betInitiator`, and `feeRecipient` default to address(0). Attempts to place bets or claim fees before setting them will revert (e.g., transfers to/from zero address). While settable by owner, there's no enforcement or event emission on initialization, risking operational errors.

Impact
Contract unusable until properly initialized, potential reverts.

Proof of Concepts
Deploy, placeBet without setBetToken, transferFrom fails.

Recommended mitigation
Initialize in constructor or add require checks.

Client Reponse
Acknowledged

[M-5] Unlimited Buddy Bet Creation Allows Storage Bloating

Severity Assesment
Impact: Medium
Likelihood: Medium

Context
`createBuddyBet` function.

Description
Anyone can call `createBuddyBet` repeatedly with no limits, pushing to the `bets` array and bloating storage. While gas costs limit extreme abuse, it can increase costs for views like `getAllBetsCreated` and fill bet IDs unnecessarily.

Impact
Storage bloat, increased gas for operations, potential griefing.

Proof of Concepts
Attacker creates 1000+ empty bets, bloating array.

Recommended mitigation
Add creation limits or fees for bet creation.

Client Reponse
Fixed

[M-6] With Large `eligibleAddresses`, the `placeBet` might face DOS

Severity Assesment
Impact: Medium
Likelihood: Low

Context
`createBuddyBet` and `isEligible`.

Description
Large `eligibleAddresses` arrays in buddy bets can make `placeBet` unaffordable, as warned. Attackers could create bets with massive arrays to grief.

Impact
Denial of service for eligible users due to high gas.

Proof of Concepts
Create buddy bet with 1000+ addresses, eligible user placeBet loops, high gas/fail.

Recommended mitigation
Cap eligibleAddresses length or use mapping for eligibility.

Client Reponse
Acknowledged

[M-7] No Slippage or Oracle Integration

Severity Assesment
Impact: High
Likelihood: High

Context
Bet resolution by publisher.

Description
Bet odds are dynamic (parimutuel), but no external data feeds for real-world events. Relies on publisher honesty, vulnerable to manipulation.

Impact
Publisher can manipulate outcomes for personal gain.

Proof of Concepts
Publisher sees volumes, resolves to favor own bets.

Recommended mitigation
Integrate oracles (e.g., Chainlink) for objective resolutions.

Client Reponse
Acknowledged

[M-8] No Oracle Integration for bet resolution.

Severity Assesment
Impact: High
Likelihood: High

Context
Bet resolution by publisher.

Description
Bet odds are dynamic (parimutuel), but no external data feeds for real-world events. Relies on publisher honesty, vulnerable to manipulation.

Impact
Publisher can manipulate outcomes for personal gain.

Proof of Concepts
Publisher sees volumes, resolves to favor own bets.

Recommended mitigation
Integrate oracles (e.g., Chainlink) for objective resolutions.

Client Reponse
Acknowledged

Low Severity

[L-1] Incompatibility with Fee-on-Transfer Tokens

Severity Assesment
Impact: Low
Likelihood: Low

Context
In `placeBet`, the full `amount` is transferred via `safeTransferFrom`.

Description
Fees are calculated on `amount`, and net amounts are recorded assuming no token-level deductions. For fee-on-transfer (deflationary) tokens, the contract receives `amount - token_tax`, but records `netAmount = amount - protocol_fee` for payouts. If `token_tax != protocol_fee`, this leads to over-accounting, insufficient balance for claims, and stuck funds.

Impact
Over-accounting or under-accounting of balances, leading to insufficient funds for payouts or excess stuck funds.

Proof of Concepts
Tested conceptually with tax > fee: contract has less than expected, breaking payout invariant (total recorded nets > actual balance). If tax < fee, excess but still mismatch.

Recommended mitigation
Use balance diff checks before/after transfer to handle actual received amount.

Client Reponse
Acknowledged

[L-2] Overwriting of betToCreator Mapping

Severity Assesment
Impact: Low
Likelihood: Low

Context
Bet creation functions.

Description
When a creator (or initiator) creates multiple bets, `betToCreator[msg.sender]` is overwritten with the latest `betId`. This mapping only tracks the most recent bet per address, rendering it useless for historical or multi-bet tracking.

Impact
Poor tracking of creator bets.

Proof of Concepts
Create two bets, betToCreator shows only second.

Recommended mitigation
Use mapping to array of betIds.

Client Reponse
Acknowledged

[L-3] Precision Loss and Dust Accumulation

Severity Assesment
Impact: Low
Likelihood: Low

Context
Winnings calculation in `claimWinning`.

Description
Winnings calculation uses `1e18` for percentage shares, but integer division can lead to rounding errors (e.g., `(userSharePercentage * losingAmount) / 1e18` may leave 1-wei remnants). Over many claims, dust accumulates in the contract with no way to recover it. For small bet amounts (e.g., near wei level), this can cause significant relative losses—e.g., a 1 wei bet in a 3 wei winning pool yields ~0 share of losses due to flooring, leaving funds stuck. Even for larger amounts, cumulative rounding over multiple winners leaves dust unstuck. No sweep function exacerbates this.

Impact
Minor fund accumulation as dust.

Proof of Concepts
N/A

Recommended mitigation
Add dust sweep function for owner or use higher precision.

Client Reponse
Acknowledged

[L-4] No Check for Bet Overlap or Duplicates

Severity Assesment
Impact: Low
Likelihood: Low

Context
Bet creation.

Description
Multiple bets can be created with identical `text`, `image`, and `duration`, leading to user confusion or fragmented liquidity. No uniqueness enforcement exists.

Impact
User confusion, split liquidity.

Proof of Concepts
Create two identical bets, users bet on both.

Recommended mitigation
Add hash check for uniqueness.

Client Reponse
Acknowledged

[L-5] Unused Variables and Mappings Waste Gas and Storage

Severity Assesment
Impact: Low
Likelihood: Low

Context
Various mappings and variables in the contract.

Description

- `userVolume[msg.sender]` incremented in `placeBet` but never read or used.
- `bet.volume` incremented but unused.
- `betToCreator[msg.sender]` set but unused (overwrites on multiple creates).

Impact
Wastes gas on writes and storage.

Proof of Concepts
N/A

Recommended mitigation
Remove unused variables and mappings.

Client Reponse
Acknowledged

[L-6] Redundant Data Storage in BetPlacement

Severity Assesment
Impact: Low
Likelihood: Low

Context
`BetPlacement` struct in `placeBet`.

Description
`BetPlacement` stores copies of `bet.text`, `bet.image`, and `bet.duration` (as

`expiry`), which are already in `bets[betId]`. This increases storage/gas costs in `userBets` array and `userBet` mapping without benefit, as bets are immutable post-creation.

Impact
Unnecessary gas and storage costs.

Proof of Concepts
N/A

Recommended mitigation
Remove redundant fields from BetPlacement.

Client Reponse
Acknowledged

[L-7] `betAmounts` Not Cleared for Losers in `claimWinning`

Severity Assesment
Impact: Low
Likelihood: Low

Context
`claimWinning` for losers.

Description
For losers (outcome != side and != DRAW), `claimWinning` skips transfers but doesn't zero `betAmounts[betId][msg.sender]`. While harmless (nothing to claim, and expired prevents reuse), it leaves stale data.

Impact
Stale data in mappings.

Proof of Concepts
N/A

Recommended mitigation
Add zeroing for cleanliness.

Client Reponse
Acknowledged

[L-8] No Bulk Claim Function

Severity Assesment
Impact: Low
Likelihood: Low

Context
`claimWinning` function.

Description
Users must call `claimWinning` per `betId`, increasing gas for multi-bet users.

Impact
Higher gas costs for users with multiple bets.

Proof of Concepts
N/A

Recommended mitigation
Add a bulk claim function.

Client Reponse
Acknowledged

Informational

[I-1] Inconsistent Fee Scaling

Severity Assesment
Impact: Low
Likelihood: Low

Context
Fee constants and calculation.

Description
The `MAX_FEE_BPS` is set to 50 with a comment indicating 5%, and fees are calculated as `(amount * fee) / 1000`, which aligns with 50 representing 5% (e.g., 50/1000 = 0.05). However, "BPS" typically means basis points (where 50 BPS = 0.5%), creating confusion.

Impact
Developer confusion on fee units.

Proof of Concepts
N/A

Recommended mitigation
Clarify units or rename constant to MAX_FEE_PER_1000.

Client Reponse
Acknowledged

[I-2] Lack of ERC20 Approval Checks

Severity Assesment
Impact: Low
Likelihood: Low

Context
`placeBet` transfer.

Description
Relies on users approving `betToken` transfers, but no on-chain verification of allowances, risking failures if approvals are insufficient.

Impact
Transaction reverts due to user error.

Proof of Concepts
User forgets approval, placeBet reverts.

Recommended mitigation
Add allowance check before transfer.

Client Reponse
Acknowledged