Amped Protocol Fix Plan (Updated)

Executive Summary

The Amped Protocol contracts (AmpedSwapRouter and AmpedStakingRouter) contain critical architectural flaws that prevent them from functioning as intended. This updated document incorporates feedback from the security audit, correcting inaccuracies in the staking refactor (e.g., method signatures and ERC20 handling), ensuring alignment with the actual RewardTracker implementation (which is ERC20-based with handler privileges for seamless chaining). We have also enhanced code examples, added handler setup requirements, and expanded testing for chaining mechanics.

Current Status: CRITICAL - Contracts will fail in production without fixes
Estimated Timeline: 2-4 weeks for complete refactoring
Risk Level: High - Total protocol failure without fixes
Key Changes in This Version: Accurate staking/unstaking code examples based on RewardTracker mechanics; explicit handler setup; no coverage of external exploits as per confirmation that vulnerable components are disabled.

Critical Issues Overview

1. Fundamental Architecture Misunderstanding

The contracts misunderstand how GMX V1 reward trackers work:

- **Current (Wrong)**: Assumes direct staking without proper ERC20 receipt handling or approvals
- **Correct (GMX V1)**: Trackers are ERC20 tokens; staking mints receipts to users, requiring handler-privileged transfers for chaining without user approvals

2. Oracle Validation Conflicts

- **Issue**: Enforces price oracle validation for 1:1 AMP/AMPED swaps
- **Impact**: Valid swaps fail if oracle prices deviate from exact 1:1

3. Staking Chain Failure

3. Staking Chain Failure

- **Issue**: Missing handler-privileged transfers for ERC20 receipts between trackers
- **Impact**: Staking transactions will revert after first tracker due to approval issues

4. Unstaking Approval Requirements

- **Issue**: Requires users to pre-approve router for AMP tokens
- **Impact**: Breaks "seamless" user experience

Detailed Fix Plan

Phase 1: SwapRouter Fixes (Immediate)

Fix 1.1: Remove Oracle Validation for AMP/AMPED Pairs ✅

Status: Completed

```solidity
// In _validatePriceWithOracle function
// Skip oracle validation for AMP/AMPED pairs (always 1:1)
if ((_tokenIn == ampedToken && _tokenOut == ampToken) ||
    (_tokenIn == ampToken && _tokenOut == ampedToken)) {
    return;
}
```

Rationale: AMP/AMPED are always 1:1 by design. Oracle validation adds unnecessary complexity and potential failure points.

Fix 1.2: Handle Direct 1:1 Swaps When Disabled

Current Issue: When `swapEnabled=false` or no external DEX, swaps revert entirely
Required Fix: Add fallback logic for 1:1 conversion using contract-held liquidity

```solidity
if (!useExternalDex || dexRouter == address(0)) {
    // Direct 1:1 swap for AMP/AMPED pairs
```

```
    if (_tokenIn == ampedToken && _tokenOut == ampToken) {
        // Ensure sufficient liquidity in contract
        require(IERC20(ampToken).balanceOf(address(this)) >= _amountIn, "Ins
        return _amountIn; // 1:1 ratio, transfer from contract reserves
    } else if (_tokenIn == ampToken && _tokenOut == ampedToken) {
        require(IERC20(ampedToken).balanceOf(address(this)) >= _amountIn, "I
        return _amountIn; // 1:1 ratio
    }
    revert("Unsupported pair for internal swap");
}
```

Rationale: Users should always be able to convert between AMP/AMPED even if external swaps are disabled. Use contract reserves for transfers.

Phase 2: StakingRouter Critical Refactor

Fix 2.1: Redesign Staking Chain Architecture

Current Issue: Code attempts improper staking calls without handling ERC20 receipts or approvals

GMX V1 Pattern:

- Trackers are ERC20 tokens; staking mints receipt tokens (e.g., sAMP) to the user.
- For chaining, use router as handler on trackers to transfer receipts without user approvals (via special `transferFrom` logic).
- Sequence: Stake into first tracker (mints to user), handler-transfer receipt to router, approve and stake into next.

Prerequisites:

- Set StakingRouter as handler on all trackers post-deployment: `rewardTracker.setHandler(stakingRouter.address, true)` for each (stakedGmxTracker, bonusGmxTracker, feeGmxTracker).

Required Changes (in `_stakeAmped`):

```
solidity                                   Collapse   Wrap   Copy
```

```
// Approve for first stake
uint256 currentAllowance = IERC20(ampToken).allowance(address(this), stakedG
```

```
if (currentAllowance > 0) IERC20(ampToken).safeApprove(stakedGmxTracker, 0);
IERC20(ampToken).safeApprove(stakedGmxTracker, ampAmount);

// Step 1: Stake AMP into stakedGmxTracker (pulls from this, mints sAMP to _
IRewardTracker(stakedGmxTracker).stakeForAccount(address(this), _account, am

// Step 2: As handler, transfer sAMP from _account to this (no user approval
IERC20(stakedGmxTracker).transferFrom(_account, address(this), ampAmount);

// Approve for next stake
currentAllowance = IERC20(stakedGmxTracker).allowance(address(this), bonusGm
if (currentAllowance > 0) IERC20(stakedGmxTracker).safeApprove(bonusGmxTrack
IERC20(stakedGmxTracker).safeApprove(bonusGmxTracker, ampAmount);

// Step 3: Stake sAMP into bonusGmxTracker (mints bnAMP to _account)
IRewardTracker(bonusGmxTracker).stakeForAccount(address(this), _account, sta

// Step 4: Transfer bnAMP from _account to this
IERC20(bonusGmxTracker).transferFrom(_account, address(this), ampAmount);

// Approve for final stake
currentAllowance = IERC20(bonusGmxTracker).allowance(address(this), feeGmxTr
if (currentAllowance > 0) IERC20(bonusGmxTracker).safeApprove(feeGmxTracker,
IERC20(bonusGmxTracker).safeApprove(feeGmxTracker, ampAmount);

// Step 5: Stake bnAMP into feeGmxTracker (mints fAMP to _account)
IRewardTracker(feeGmxTracker).stakeForAccount(address(this), _account, bonus
```

Rationale: Aligns with ERC20-based trackers. Handler privilege allows seamless receipt transfers. Ensures positions and rewards are credited directly to the user.

Fix 2.2: Fix Unstaking Flow

Current Issue: Router tries to pull AMP from user after unstaking, requiring approval
Required Fix: Unstake directly to router in reverse order (fee → bonus → staked), so router receives tokens for swapping

solidity Collapse Wrap Copy

```
// In _unstakeAmped
```

```
// Step 1: Unstake from feeGmxTracker (burns fAMP from _account, sends bnAMP
IRewardTracker(feeGmxTracker).unstakeForAccount(_account, bonusGmxTracker, _

// Step 2: Unstake from bonusGmxTracker (burns bnAMP from _account, sends sA
IRewardTracker(bonusGmxTracker).unstakeForAccount(_account, stakedGmxTracker

// Step 3: Unstake from stakedGmxTracker (burns sAMP from _account, sends AM
IRewardTracker(stakedGmxTracker).unstakeForAccount(_account, ampToken, _amou

// Now router has AMP, proceed with swap to AMPED and transfer to _account
```

Rationale: Burns are internal (no approvals needed); directing outputs to router makes it seamless.

Fix 2.3: Add Internal 1:1 Conversion

Issue: StakingRouter reverts if swapEnabled=false

Fix: Add direct conversion logic using contract reserves

solidity Collapse Wrap Copy

```
if (!swapEnabled || swapRouter == address(0)) {
    if (tokenIn == ampedToken && tokenOut == ampToken) {
        // Direct 1:1 conversion
        require(IERC20(ampToken).balanceOf(address(this)) >= _amount, "Insuf
        ampAmount = _amount;
        // Transfer AMP from reserves to continue staking
    } else if (tokenIn == ampToken && tokenOut == ampedToken) {
        // Similar for unstake
    }
}
```

Phase 3: Testing & Validation

Test 3.1: Reward Tracker Integration

- Deploy test contracts mimicking GMX V1 trackers
- Verify internal balance updates and ERC20 mint/burn
- Test full staking chain, including handler transfers

Test 3.2: End-to-End Flow Testing

1. User approves AMPED to StakingRouter
2. StakeAmped: AMPED → AMP → Staked in trackers (verify positions credited to user)
3. Earn rewards over time (simulate distributor)
4. UnstakeAmped: Unstake → AMP → AMPED
5. Verify no additional approvals needed beyond initial

Test 3.3: Edge Cases

- Test with swapEnabled=false (using internal 1:1)
- Test with zero oracle prices (ensure bypass works)
- Test with maximum amounts
- Test pause/unpause scenarios
- Test chaining failures (e.g., without handler set) and recoveries

Phase 4: Security Considerations

Risk 4.1: Reentrancy in New Flow

- Maintain nonReentrant modifiers on all external-call functions
- Follow checks-effects-interactions pattern strictly
- Test for reentrancy in handler transfers and staking calls

Risk 4.2: Token Approval Management

- Minimize approval requirements (handler bypasses for receipts)
- Consider EIP-2612 permits for any remaining user approvals
- Clear documentation on required approvals (e.g., initial AMPED only)

Risk 4.3: Handler Privilege Management

- Document risks of handler role (e.g., potential for abuse if compromised)
- Use multisig/timelock for setting handlers

- Add emergency revoke function for handlers

Risk 4.4: Migration Path

- If any users have existing positions, provide migration tools
- Implement emergency withdrawal functions with timelock
- Maintain backward compatibility where possible

Implementation Priority

1. **CRITICAL (Week 1)**:
    - SwapRouter oracle fix ✅
    - SwapRouter disabled swap handling

- Set up handlers on trackers
- Research/confirm tracker interfaces (if needed)

2. **HIGH (Week 2-3)**:
   - Redesign staking chain with ERC20 handling
   - Fix unstaking flow
   - Add internal conversions
3. **TESTING (Week 3-4)**:
   - Deploy to testnet
   - Full integration testing, including chaining
   - Edge case validation
4. **AUDIT (Week 4+)**:
   - Professional security audit focusing on chaining and handlers
   - Fix any findings
   - Final deployment preparation

Success Criteria

1. **Functional Requirements**:
   - Users can stake AMPED and receive staking credit/rewards directly
   - Users can unstake and receive AMPED back

- No manual approval steps except initial AMPED approval
- Works with swapEnabled=true or false
- Seamless chaining without reverts

2. **Security Requirements**:
   - No reentrancy vulnerabilities
   - No approval race conditions
   - Proper access controls (e.g., handlers secured)
   - Emergency pause functionality works
3. **Performance Requirements**:
   - Gas costs comparable to direct GMX interaction
   - No unnecessary token transfers
   - Efficient approval management

Conclusion

The current contracts have fundamental architectural flaws that make them non-functional. The fixes outlined above are not optional enhancements but critical requirements for basic functionality, now aligned with the ERC20-based tracker mechanics.

Key Takeaway: The staking router must be completely redesigned to handle ERC20 receipts via handler privileges. This is a significant refactor, not a minor patch.

Recommendation: Do not deploy current contracts. Implement all fixes and conduct thorough testing before any mainnet deployment.