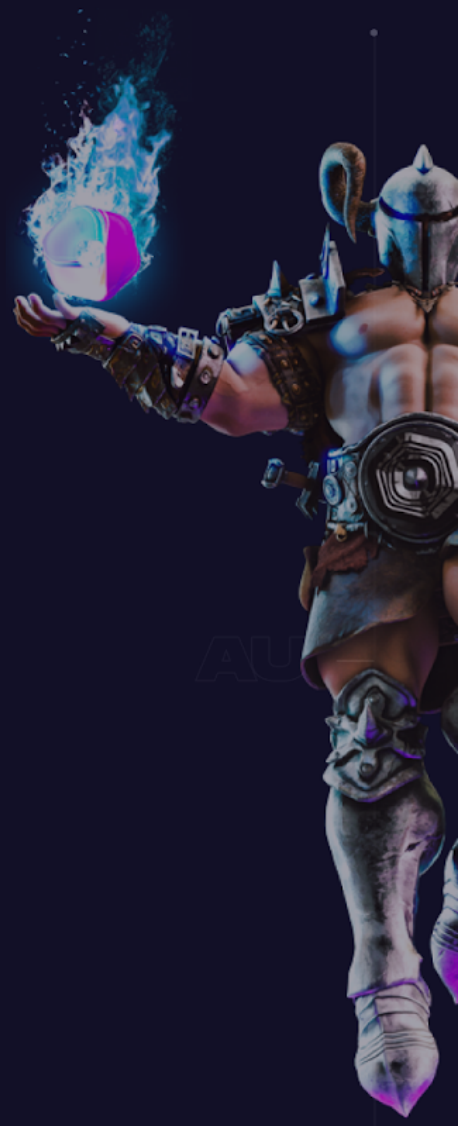
UnoRe

Insured Audit: Code Review & Protocol Security Report



Based

Protocol

Based Markets

Date

1st September 2023

The **Uno Re Watchdog** security research team has completed an initial time-boxed security review of the Based Markets smart contract code and Muon App JS script, with a focus on the security aspects of the application's implementation.

Disclaimer

This report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all the vulnerabilities are fixed - upon the decision of the Customer.

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.

Document Changelog:

1st September 2023	**Audit Kickoff and Scoping**
19th September 2023	**Based Markets Insured Audit Report**

Technical Overview

BASED Markets are Intent-based derivatives. BASED Markets is built on top of SYMM-IO trading engine https://www.symm.io/how-it-works. The Based Market team reached out to us and asked us to audit their:

1. DiBs rewarder (rewards against trading activity) + periphery contracts
2. The Muon app client is responsible for the validity of reward claims.

You can read more about the protocol in the full documentation at https://docs.based.markets/

Threat Model

Internal Security QA

1. Q: Would it be possible for you to provide us with the parameters for the initialization functions in BasedRewarder.sol, such as _muonAppID, _muonPublicKey, and _validMuonGateway?

   A: The initialization variables have been mentioned below:

```
[
    ethers.constants.AddressZero, // based token
   "0x6E40691a5DdC2cBC0F2f998Ca686BDF6C777ee29", // admin
    BigNumber.from("10000000000000000"), // start time.
    0x6914c3af649c285d706d6757dd899d84b606c2da", // valid gateway
    "29996138867610942848855832240712459333931278134263772663951800460922233661812", // app id
     [
      "0x4d8bf64cdc8651641833910995bfe0aed9b61037721f3d2305d1f87e8f3ad815",
      "0",
     ], // public key
  ];
```

2. Q: Could you explain how the value for _reqID, _gatewaySignature, and _sign are obtained in the claim function?

   A: This is a sample Muon request for getting a claim signature. for more information on how this works, ask someone from Muon https://dibs-shield.muon.net/v1/?app=dibsGlobal&method=userVolume&params[projectId]=0x9349663dc03c4dd87f6dc2a1f01460e76a0bfe64f65e0e02a23ba682a102c5b5&params[day]=0&params[pair]=0x0000000000000000000000000000000000000000&params[user]=0x05d1c344bca35880e079b1dfb97d36d9194d4f83

3. Q: What is the purpose of setStartTimestamp? The Based contract fills rewards into BasedRewarder daily for four years, starting on startTimestamp as defined in Based. The startTimestamp on BasedRewarder is changeable by the admin. What changes does the admin seek when calling setStartTimestamp on BasedRewarder, and how does this affect the two startTimestamps?

   A: We want to start the rewards once everything is ready. Its only callable once.

4. Q: Would the shield node and Muon nodes for the BASED team all be using Infura for Ethereum RPC calls as shown in the example .env, ref? Our concern is excessive trust in the RPC provider.

   A: Currently shield node uses public Ankr rpcs.

5. Q: For your use of the Graph, would you be using the Graph's decentralized network or some sort of self-hosted solution? If you have a self-hosted deployment, my concern is excessive trust in the BASE team

   A; Currently, we use Graph's decentralized network.

6. Q: Could you do a simple walk through what these functions are for and how they're used? Presumably, signParams provides the list of params for the Muon nodes of the app's subnet to sign, then the signature should be later aggregated and submitted on-chain for verification. Which application does the function run in? Similarly, what is whoIsWinner used for?

   A: Here is a walkthrough of the mentioned functions:

```
signParams:
Each muon app has two basic functions (onRequest & signParams). When a Muon node receives a
request, onRequest function is called first, and its result is passed to signParams. In signParams,
each Muon node can access the gateway node's result and check its own result against the gateway's,
throwing an error in case of a bad result.

* The process of handling a Muon app request that has a shield node is as follows:

- The shield node receives the request and then forwards it to the shield_forward_url, which is
supposed to be the URL of Muon network.
- After forwarding the request and receiving the response from the Muon network, the shield node runs
the app and checks its own result against the Muon network's result.
- If the result hashes are the same, the shield node generates a signature.

* The gateway node is the first node in the app subnet that receives the request.


whoIsWinner:
It is the logic for choosing the winner from a list of tickets. It takes seed and tickets as inputs, which
are a random number and a list of users, respectively.
The index of the winner is determined by the expression 'seed % tickets.length'.

* Please note that each user can appear in the list multiple times.
```

Note: Additional QA regarding the security of the Muon oracles and the trade2earn contracts case scenarios for various signature types have not been added to the report. However interested researchers can reach out to our team via Uno Re discord server for additional information.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic, and reputation damage of a successful attack
Likelihood - the chance that a particular vulnerability gets discovered and exploited
Severity - the overall criticality of the risk

Security Review Summary

Last Review commit hash Based Markets Contracts and Muon app JS Script file - 4a0eafc48250fb50c2aca0af22763547cb55cdc7

Audit Scope

The following smart contracts and JS scripts were in scope of the audit:

Precommit hash: 08b521e9cc0fb3f5a09db44743b0b4e2537fb5be

- MultiRewarder.sol
- Based.sol
- BasedRewarder.sol
- BasedDeployer.sol
- MuonClient.sol
- MuonClientbase.sol
- SchnorrSECP256K1Verifier.sol
- dibsGlobal.js

The following number of issues were found, categorized by their severity:

- Critical: 1 issue
- High: 0 issues
- Medium: 2 issues
- Low: 0 issue
- Informational: 1 issue

Note: The above summary of report findings at the Pre-Triage stage, most of these issues were addressed/fixed in consecutive stages.

Summary Table of Our Findings

ID	TITLE	Severity	Status
[C-01]	Anyone can freeze depositor's reward	Critical	Fixed
[M-01]	Reward gets stuck when there is no staked BASED token	Medium	Acknowledged
[M-02]	BASED reward gets stuck when there is zero trading volume	Medium	Acknowledged
[I-01A-E]	Misc Informational Items	Informational	Fixed

Triage Fix Comments

[M-01] Reward gets stuck when there is no staked BASED token

Protocol Team; We believe it's not a major issue, and it seems to be just fine.

[M-02] BASED reward gets stuck when there is zero trading volume

Protocol Team: We won't be taking action for this issue in this context. However, the protocol team has taken actions while designing the ecosystem to ensure that this case scenario never happens. This will help in preventing such scenarios.

[I-1A] Storage gaps helps with upgradeability of base contracts

Protocol Team:- There is a good chance that we might want to upgrade MuonClientBase which DibsRewarder is inheriting from. what should the size of the gap be? or how should we determine that?

Audit Team: Openzeppelin adds __gap such that there is 50 slots. "The size of the __gap array is calculated so that the amount of storage used by a contract always adds up to the same number (in this case 50 storage slots)."

Protocol Team:- since MuonClient has no state variables, does it mean this should be the gap: uint256[50] private __gap ?

Audit Team: Yes, that's the convention OpenZeppelin uses.

Note: Some additional triage comments during the discussion of reported issues of Muon oracles and the trade2earn contracts have not been added to the report. However interested researchers can reach out to our team via Uno Re discord server for additional information.

Centralization Risk Areas:

We have also identified several key areas within the protocol that contain centralization risks that need to be made aware to the community and have highlighted them below:

1. Based.sol:
    a. Initialize
2. BasedDeployer.sol:
    a. Deploy
    b. deployMany
3. BasedRewarder.sol:
    a. setBased
4. The dibsGlobal.js (muon app) may or may not work as intended if
    a. Subgraphs are down for any reason
    b. TSS nodes are down
    c. Shield nodes are down

Initial Report Detailed Findings

[C-01] Anyone can freeze depositor's rewards

Issue Description

In MultiRewarder.sol, any user can call notifyRewardAmount to add reward tokens for depositors. based depositors can then receive their portions of the added rewards by calling getReward.

However, a malicious user can add a reward token to intentionally block getReward, eg blocking transfers from MultiRewarder.

Attack Scenario

Alice adds an Attack ERC20 as reward by calling notifyRewardAmount. The Attack's transfer method specifically blocks transfers from the MultiRewarder address.

```
    function transfer(address to, uint256 amount) ... {
        address owner = _msgSender();
            if (owner == multiRewarder) revert Error("PWNED");
        ...
    }

```

Anyone deposited Based tokens into MultiRewarder and tries to getReward, receives the revert above, because getReward loops through all the reward tokens and performs a transfer of reward to the caller.

```
     function getReward() public nonReentrant updateReward(msg.sender) {
        for (uint256 i; i < rewardTokens.length; i++) {
            ...
            if (reward > 0) {
                ...
                  IERC20(_rewardsToken).safeTransfer(msg.sender,reward);
                ...
            }
        }
```

```
    }
```

Because of this action all rewards will be permanently frozen because getReward is the only way of withdrawing.

Recommended Fix

Consider the following options:

1. Using an allowlist for rewardTokens to avoid depending on malicious external contracts. If there is a need to add a new reward token, it can always be done by deploying a new MultiRewarder for that specific reward token(s).
2. Using an allowlist for the callers of notifyRewardAmount, but this adds permissions without offering benefits compared to (1).

[M-01] Reward gets stuck when there is no staked BASED token

Issue Description

In MultiRewarder's notifyRewardAmount ie when someone tops up a new reward, the leftover gets rolled over and gets disbursed in the next 7 days.

```
        } else {
         // @audit Calculate the remaining seconds in this 7-day period.
               uint256 remaining = rewardData[_rewardsToken].periodFinish -
                    block.timestamp;
                // @audit Let `leftover` be integral of rewardRate.
                uint256 leftover = remaining *
                    rewardData[_rewardsToken].rewardRate;
                // @audit Let `rewardRate` be the new reward amount plus
leftover, to be disbursed in the next 7 days.
                rewardData[_rewardsToken].rewardRate =
                    (reward + leftover) /
                    rewardData[_rewardsToken].rewardsDuration;
            }
 ```

rewardRate is the amount of reward per second. However, when there is no staked BASED token, rewardRate does not get disbursed, leading to leftover reward stuck in MultiRewarder.

Another way to explain the issue is that the definition of leftover effectively assumes rewardRate has been applied for every second that has elapsed.

```
// From notifyRewardAmount
uint256 leftover = remaining * rewardData[_rewardsToken].rewardRate;

```

However this is not true when there is no staked BASED token.

```
   function rewardPerToken(
        address _rewardsToken
    ) public view returns (uint256) {
      // @audit If there is no based tokens staked, the function DOES NOT
```

```
accumulate new reward to avoid division by 0.
        if (totalSupply == 0) {
            return rewardData[_rewardsToken].rewardPerTokenStored;
        }
        return
            rewardData[_rewardsToken].rewardPerTokenStored +
            (((lastTimeRewardApplicable(_rewardsToken) -
                rewardData[_rewardsToken].lastUpdateTime) *
            rewardData[_rewardsToken].rewardRate * // @audit rewardRate =
reward per token per second
                1e18) / totalSupply);
    }
```

The impact of this particular issue is met under the condition that when there is no staked BASED token in MultiRewarder. And the max impact in this case would be the fact that the reward rate accumulated over the period of no BASED staking does not get disbursed and gets stuck permanently.

Recommended Fix

Consider tracking the reward credits and debits to define leftover, and this method's potential implications.

```
uint rewardNotified;
uint rewardEarned;
// In notifyRewardAmount
uitn leftover = rewardNotified - rewardEarned;
...
rewardNotified += reward;

// In updateReward
rewardEarned += earned(account, token);
```

POC written by Audit Issue Author

```
// MultiRewarder.poc.t.sol
pragma solidity 0.8.17;
```

```
import {Test, console} from "forge-std/Test.sol";
import {MultiRewarder} from "../../contracts/MultiRewarder.sol";
import {Based} from "../../contracts/Based.sol";
import {StableCoin} from "./StableCoin.sol";

contract ContractBTest is Test {
    Based         based;
    MultiRewarder multiRewarder;
    StableCoin    stableCoin;

     struct Reward {
        uint256 rewardsDuration;
        uint256 periodFinish;
        uint256 rewardRate;
        uint256 lastUpdateTime;
        uint256 rewardPerTokenStored;
    }

    function setUp() public {
      based         = new Based(address(this));
      multiRewarder = new MultiRewarder(address(based));
      stableCoin    = new StableCoin();
    }

    function sendReward(uint reward) private {
      stableCoin.mint(address(this), reward);
      stableCoin.increaseAllowance(address(multiRewarder), reward);
      address[] memory tokens = new address[](1);
      uint[] memory rewards   = new uint[](1);
      tokens[0]               = address(stableCoin);
      rewards[0]              = reward;
      multiRewarder.notifyRewardAmount(tokens, rewards);
   }

    function test_Leftover() public {
      address alice = address(0x1001);
      uint deposit = 100e18;

      based.transfer(alice, deposit);

      // At the beginning of time
      // reward gets sent in and Alice starts earning
```

```
        vm.warp(0);
        sendReward(100e18);

        vm.startPrank(alice);
        based.increaseAllowance(address(multiRewarder), deposit);
        multiRewarder.deposit(deposit, alice);
        vm.stopPrank();

        // At 40% of a week
        // Alice withdraws and therefore stops earning
        vm.warp(0.4 * 1 weeks);
        vm.prank(alice);
        multiRewarder.withdraw(deposit, alice);

        // At the end of the week
        // leftover is 0 but MultiRewarder still has some reward remaining
        // and there is no way withdraw it.
        vm.warp(7 days);
        (, uint periodFinish, uint rewardRate, , ) =
multiRewarder.rewardData(address(stableCoin));

        // IMPORTANT
        // These 2 liens are from `notifyRewardAmount`.
        // This demonstrates:
        // if there is 0 BASED tokens staked, then
        // the reward rate accumulated over the period of no BASED staking,
        // **IS NOT** carried over.
        // Furthermore, there is no way to withdraw this stuck reward.
        uint remaining = periodFinish - block.timestamp;
        uint leftover = remaining * rewardRate;

    * Logs:
        *   periodFinish: 604800
        *   remaining: 0
        *   leftover: 0
        *   rewardRate: 165343915343915
        *   Alice earned: 39999999999999916800
        *   stableCoin.balanceOf: 100000000000000000000
        */
        console.log("periodFinish: %s", periodFinish);
        console.log("remaining: %s", remaining);
        console.log("leftover: %s", leftover);
```

```
      console.log("rewardRate: %s", rewardRate);
      console.log("Alice earned: %s", multiRewarder.earned(alice,
address(stableCoin)));
      console.log("stableCoin.balanceOf: %s",
stableCoin.balanceOf(address(multiRewarder)));
    }
}

```
```
pragma solidity 0.8.17;

import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import {console} from "forge-std/Test.sol";

contract StableCoin is ERC20("stablecoin", "STABLE") {
    function mint(address account, uint amount) external {
      ERC20._mint(account, amount);
    }
  }

```

[M-02] BASED reward gets stuck when there is zero trading volume

Issue Description

In Based.sol, the token used as a reward in BasedRewarder.sol, anyone can trigger a reward fill into BasedRewarder.sol on a daily basis for roughly 4 years.

```
// Based.sol
    function fillDibsRewarder(uint256 day) ... {
        ...
        if (!isRewardMinted[day]) {
            isRewardMinted[day] = true;

            uint256 amount = getDibsRewardAmount(day);
            if (amount > 0) {
                _mint(address(this), amount);
                IDibsRewarder(dibsRewarder).fill(day, amount);
                ...
            }
        }
    }
```

BasedRewarder.sol records the reward amount received for the day.

```
 // BasedRewarder.sol
    function fill(uint256 _day, uint256 _amount) external {
        ...
        totalReward[_day] += _amount;
        ...
    }
```

However, when there is no trading volume for the day, that day's reward will be permanently locked. Because:

1. valid _userVolume and _totalVolume are 0
2. consequently, rewardAmount eligible for the claim is 0, and there is no other way to withdraw totalReward[_day]

```
// BasedRewarder.sol
    function claim(
        uint256 _day,
        uint256 _userVolume,
        uint256 _totalVolume,
        uint256 _sigTimestamp,
        bytes calldata _reqId,
        SchnorrSign calldata _sign,
        ...
    )  ... {
        ...
        verifyTSSAndGW(
            abi.encodePacked(
                PROJECT_ID,
                msg.sender,
                address(0),
                _day,
                _userVolume,
                _totalVolume,
                _sigTimestamp
            ),
            _reqId,
            _sign,
            _gatewaySignature
        );
        uint256 rewardAmount = (totalReward[_day] * _userVolume) /
_totalVolume;

```

The Impact for this issue is that when there is no trading volume for the day, that day's reward will be permanently locked if one user calls fill on Based.sol .

Recommended Fix Type A - moveReward

Consider defining a new function, on BasedReward.sol, moveReward that is gated by a Muon network signature (like claim) attesting the day has zero volume. This function can move some amount of reward from one day to another.

```
 function moveReward(
        uint256 _dayX,
```

```
        uint256 _dayY,
        uint256 _reward,
        uint256 _sigTimestamp,
        bytes calldata _reqId,
        SchnorrSign calldata _sign,
        bytes calldata _gatewaySignature
    );
```

The pro's of this approach:

1. This function does not introduce new permission

The cons of this approach:

1. It introduces new incentives for traders.
2. Let's say there is day X that has zero volume and day Y has positive volume. Traders, knowing they can move reward from day X to day Y, are incentivized (and frontrun other traders) to move reward to day Y

Recommended Fix Type B - burnReward

Similarly, consider defining a new function, on BasedReward.sol, burnReward. It burns totalReward[_day] amount of BASED upon receiving a Muon signature attesting the day has zero volume.

The pro's of this approach:

1. Simpler to implement
2. Benefits all BASED token holders equally

The cons of this approach:

1. Benefits not only traders, but anyone that holds BASED tokens

[I-01A-E] Misc Informational Items

[I-1A] Storage gaps helps with upgradeability of base contracts

Unless you can be confident about not upgrading base contracts, consider adding storage gap in each base contract where there are states declared (eg MuonClientBase), so adding new storage variables in base contracts won't shift storage layout in the derived contracts causing errors.

1. More info on storage gaps
2. Example

[I-1B] These functions should change visibility

1. BasedRewarder:__DiBsRewarder_init should change to private
2. MuonClient:__MuonClient_init should change to internal

As they are called from BasedRewarder: initialize and not meant for the public to call.

[I-1C] Call to empty __AccessControl_init should be removed

The call is to an empty function.

[I-1D] MultiRewarder: based should be set immutable

[I-1E] BasedRewarder:startTimestamp and Based:startTimestamp should ideally be the same

Based and BasedRewarder having 2 different startTimestamps could lead to confusion about what day means.

For example, suppose:

1. Based: start timestamp -> 2023-02-01
2. BasedRewarder: start timestamp -> 2023-03-15
3. there is 100 BASED tokens filled for day 2

This means users will be able to:

1. fill for day 2 starting from the beginning of 2023-02-03
2. claim for day 2 starting from the beginning of 2023-03-17

In this example, it is hard to understand what day means, compared to setting both timestamps to be one and the same (eg 2023-02-03).