SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for:	Sentiment
Prepared by:	Sherlock
Lead Security Expert:	obront
Dates Audited:	January 18 - January 21, 2023
Prepared on:	February 2, 2023

Introduction

Sentiment is a permissionless undercollateralised onchain credit protocol that allows users to lend and borrow assets with increased capital efficiency and deploy them across DeFi.

Scope

Changes made after the following contests

- Sentiment
- Sentiment Update
- Sentiment Update #2

Oracles - https://github.com/sentimentxyz/oracle/pull/49

- GLP Oracle - Price oracle for staked GLP
- PLVGLPOracle - Modified version of ERC4626 Oracle to price PlvGLP

Controllers - https://github.com/sentimentxyz/controller/pull/55

- BaseController - Acts as a no action controller, used to approve GMX Manager as spender
- RewardRouter - Controller for GMX reward router v1
- RewardRouterV2 - Controller for GMX reward router v2
- PLVGLPController - Controller for PlvGlp

Rage Trade DN Vault controller - https://github.com/sentimentxyz/controller/pull/52

- Controller to interact with rage trade jr vault (Risk on strategy)

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.

Issues found

Medium	High
6	0

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

obront	GalloDaSballo	0xdeadbeef

Issue M-1: Risk with Liquidation - Because of partnership requirement, caller may be unable to redeem during liquidation making it less likely for them to be willing to perform the liquidation

Source: https://github.com/sherlock-audit/2023-01-sentiment-judging/issues/20

Found by

GalloDaSballo

Summary

Due to the approval system with pvGLP, liquidations may be less likely

Vulnerability Detail

In times of intense price action, a liquidation may have to be performed on pvGLP.

The protocol will offer `liquidate` which will sweep funds out, this is fine and will work as intended because it relies on `transferFrom`.

However, a liquidator will receive the vault token, and may be unable to redeem it.

That's because redemptions have to be performed via plvGLPdepositor which may not have approved the liquidators account.

This will make it less likely for liquidators to perform the operation as it may force either a manual operation (redemption can be performed by any EOA), or it will require further setup, reducing the number of operators willing to perform the liquidation in the time of need.

Impact

Code Snippet

```
function _isEligibleSender() private view {
  if (
    msg.sender != tx.origin && whitelist.isWhitelisted(msg.sender) == false &&
 ↪  partners[msg.sender].isActive == false
  ) revert UNAUTHORIZED();
}
```

Tool used

Manual Review

Recommendation

Discussion

r0ohafza

Will be communication with the plutus team and update here accordingly to validate the issue.

zobront

> Will be communication with the plutus team and update here accordingly to validate the issue.

This seems to just be missing the fact that Sentiment accounts return `true` for `whitelist.isWhitelisted()`, so this isn't an issue.

r0ohafza

> > Will be communication with the plutus team and update here accordingly to validate the issue.
>
> This seems to just be missing the fact that Sentiment accounts return `true` for `whitelist.isWhitelisted()`, so this isn't an issue.

The scenario you are referring to is of an account withdraw/redeem, but when an account is liquidated all plvGLP shares are transferred to the liquidator. This liquidator will not be able to redeem the shares and repay a flashloan used to repay the account debt.

hrishibhat

Considering this issue as a valid medium.

Issue M-2: `PreviewRedeem` may under-price the value of the asset

Source: https://github.com/sherlock-audit/2023-01-sentiment-judging/issues/19

Found by

GalloDaSballo

Summary

previewRedeem will return an incorrect result based on `address(0)`

If you get the partnership the fee changes, the address could change the value This may enable: Unfairer (bps wise), liquidations when they shouldn't happen, also will enable marginally higher profit for liquidators as they may be able to benefit from the reduction of the fee

Vulnerability Detail

Impact

A user may get liquidated earlier, and the accounting would be incorrect

Code Snippet

https://arbiscan.io/address/0x13f0d29b5b83654a200e4540066713d50547606e#code

```
function previewRedeem(address _addr, uint256 _shares)
  external
  view
  returns (
    uint256 _exitFeeLessRebate,
    uint256 _rebateAmount,
    uint256 _assetsLessFee
  )
{
  PartnerInfo memory partner = partners[_addr];
  uint256 exitFee = partner.isActive ? partner.exitFee : defaultExitFee;
  uint256 rebate = partner.isActive ? partner.rebate : defaultVaultRebate;
  uint256 assets = IERC4626(vault).previewRedeem(_shares);

  uint256 _exitFee;
  (_exitFee, _assetsLessFee) = _calculateFee(assets, exitFee);
  (_rebateAmount, _exitFeeLessRebate) = _calculateFee(_exitFee, rebate);
}
```

Tool used

Manual Review

Recommendation

Use the account to determine the price

Issue M-3: No check if Arbitrum L2 sequencer is down in Chainlink feeds

Source: https://github.com/sherlock-audit/2023-01-sentiment-judging/issues/16

Found by

0xdeadbeef

Summary

Using Chainlink in L2 chains such as Arbitrum requires to check if the sequencer is down to avoid prices from looking like they are fresh although they are not.

The bug could be leveraged by malicious actors to take advantage of the sequencer downtime.

Vulnerability Detail

The new `GLPOracle` is used the get the the price of GLP. There is no check that the sequencer is down: https://github.com/sherlock-audit/2023-01-sentiment/blob/main/oracle/src/gmx/GLPOracle.sol#L47

```
function getEthPrice() internal view returns (uint) {
    (, int answer,, uint updatedAt,) =
        ethUsdPriceFeed.latestRoundData();

    if (block.timestamp - updatedAt >= 86400)
        revert Errors.StalePrice(address(0), address(ethUsdPriceFeed));

    if (answer <= 0)
        revert Errors.NegativePrice(address(0), address(ethUsdPriceFeed));

    return uint(answer);
}
```

Impact

The impact depends on the usage of the GLP. If it is used as part of the collateral for lenders:

- Users can get better borrows if the price is above the actual price
- Users can avoid liquidations if the price is under the actual price

Code Snippet

Tool used

VS Code

Manual Review

Recommendation

It is recommended to follow the code example of Chainlink: https://docs.chain.link/data-feeds/l2-sequencer-feeds#example-code

Issue M-4: GMX Reward Router's claimForAccount() can be abused to incorrectly add WETH to tokensIn

Source: https://github.com/sherlock-audit/2023-01-sentiment-judging/issues/10

Found by

obront

Summary

When `claimFees()` is called, the Controller automatically adds WETH to the user's account. However, in the case where no fees have accrued yet, there will not be WETH withdrawn. In this case, the user will have WETH added as an asset in their account, while they won't actually have any WETH holdings.

Vulnerability Detail

When a user calls the GMX Reward Router's `claimFees()` function, the RewardRouterController confirms the validity of this call in the `canCallClaimFees()` function:

```
function canCallClaimFees()
    internal
    view
    returns (bool, address[] memory, address[] memory)
{
    return (true, WETH, new address[](0));
}
```

This function assumes that any user calling `claimFees()` will always receive `WETH`. However, this is only the case if their stake has been accruing.

Imagine the following two actions are taken in the same block:

- Deposit assets into GMX staking
- Call claimFees()

The result will be that `claimFees()` returns no `WETH`, but `WETH` is added to the account's asset list.

The same is true if a user performs the following three actions:

- Call claimFees()
- Withdraw all ETH from the WETH contract
- Call claimFees() again

Impact

A user can force their account into a state where it has WETH on the asset list, but doesn't actually hold any WETH.

This specific Impact was judged as Medium for multiple issues in the previous contest:

- https://github.com/sherlock-audit/2022-11-sentiment-judging/issues/20
- https://github.com/sherlock-audit/2022-11-sentiment-judging/issues/7

Code Snippet

https://github.com/sherlock-audit/2023-01-sentiment/blob/main/controller-55/src/gmx/RewardRouterController.sol#L54-L60

Tool used

Manual Review

Recommendation

The best way to solve this is actually not at the Controller level. It's to solve the issue of fake assets being added once and not have to worry about it on the Controller level in the future.

This can be accomplished in `AccountManager.sol#_updateTokensIn()`. It should be updated to only add the token to the assets list if it has a positive balance, as follows:

```
function _updateTokensIn(address account, address[] memory tokensIn)
    internal
{
    uint tokensInLen = tokensIn.length;
    for(uint i; i < tokensInLen; ++i) {
-        if (IAccount(account).hasAsset(tokensIn[i]) == false)
+        if (IAccount(account).hasAsset(tokensIn[i]) == false &&
 ↪  IERC20(token).balanceOf(account) > 0)
            IAccount(account).addAsset(tokensIn[i]);
    }
}
```

However, `_updateTokensIn()` is currently called before the function is executed in `exec()`, so that would need to be changed as well:

```
function exec(address account, address target, uint amt, bytes calldata data)
 ↪  external onlyOwner(account) {
    bool isAllowed;
    address[] memory tokensIn;
    address[] memory tokensOut;
    (isAllowed, tokensIn, tokensOut) = controller.canCall(target, (amt > 0),
 ↪  data);
    if (!isAllowed) revert Errors.FunctionCallRestricted();
-    _updateTokensIn(account, tokensIn);
    (bool success,) = IAccount(account).exec(target, amt, data);
    if (!success)
        revert Errors.AccountInteractionFailure(account, target, amt, data);
+    _updateTokensIn(account, tokensIn);
    _updateTokensOut(account, tokensOut);
    if (!riskEngine.isAccountHealthy(account))
        revert Errors.RiskThresholdBreached();
}
```

While this fix does require changing a core contract, it would negate the need to worry about edge cases causing incorrect accounting of tokens on any future integrations, which I think is a worthwhile trade off.

This accuracy is especially important as Sentiment becomes better known and integrated into the Arbitrum ecosystem. While I know that having additional assets doesn't cause internal problems at present, it is hard to predict what issues inaccurate data will cause in the future. Seeing that Plutus is checking Sentiment contracts for their whitelist drove this point home — we need to ensure the data stays accurate, even in edge cases, or else there will be trickle down problems we can't currently predict.

Issue M-5: Using one controller for two addresses could risk signature collisions

Source: https://github.com/sherlock-audit/2023-01-sentiment-judging/issues/9

Found by

obront

Summary

The DNGMXVaultController is intended to be used as a controller for two separate contracts when interacting with Rage Trade. While individual contracts check for signature collisions, there is no protection in this Controller. If the contracts end up with functions with colliding signatures, this may enable users to call illegal functions and get around the protocol safeguards.

Vulnerability Detail

While most of Sentiment's Controllers correspond to one contract, the DNGMXVaultController is one controller that will support two separate contracts: `DepositPeriphery` and `WithdrawPeriphery`.

While signature collisions are unlikely, they do happen. For this reason, the Solidity compiler stops code from compiling in the case of a collision. This ensures that there will not be issues in any EVM bytecode. Because Sentiment controllers approve function calls based on signatures, they can be sure that, for any given contract, they will only be approving the function they are intending.

However, once multiple contracts are managed by one controller, there are no compiler checks across these contracts for colliding signatures. The result is that there is the potential for a function signature on one contract that is approved, due to a matching signature on the other contract.

Impact

There is the potential for users to get access to non-permitted functions if there is a signature collision.

This is a security risk in this specific case, but is also a more global suggestion for Sentiment. While two contracts on one Controller one time is unlikely to cause a problem, the practice of loading multiple contracts onto on Controller should be avoided, as the risks will increase as this is performed.

Code Snippet

These three functions do not exist on the same contract:

https://github.com/sherlock-audit/2023-01-sentiment/blob/main/controller-52/src/rage/DNGMXVaultController.sol#L15-L22

Tool used

Manual Review

Recommendation

Split DNGMXVaultController into two files, one for each of the two contracts that will be interacted with.

Going forward, continue to uphold the practice of always having one Controller per contract, unless the two contracts are identical and non-upgradeable.

Issue M-6: All Rage Trade functions allow sending tokens to a different address, leading to incorrect tokensIn

Source: https://github.com/sherlock-audit/2023-01-sentiment-judging/issues/5

Found by

obront

Summary

All three approved functions on Rage Trade (`depositTokens()`, `withdrawTokens()` and `redeemTokens()`) allow for a `receiver` argument to be included, which sends the resulting tokens to that address. The corresponding Controller assumes that all tokens are withdrawn to the Sentiment account that called the function.

Vulnerability Detail

The three functions that can be called on Rage Trade have the following signatures:

- depositToken(address token, address receiver, uint256 amount)
- redeemToken(address token, address receiver, uint256 amount)
- withdrawToken(address token, address receiver, uint256 amount)

Each of these functions contains a `receiver` argument, which can be passed any address that will receive the outputted tokens.

The DNGMXVaultController incorrectly assumes in all cases that the outputted tokens will be received by the Sentiment account in question, regardless of what is entered as a receiver.

Impact

Accounting on user accounts can be thrown off (intentionally or unintentionally), resulting in mismatches between their assets array and hasAsset mapping and the reality of their account.

This specific Impact was judged as Medium for multiple issues in the previous contest:

- https://github.com/sherlock-audit/2022-11-sentiment-judging/issues/20
- https://github.com/sherlock-audit/2022-11-sentiment-judging/issues/7

Code Snippet

https://github.com/sherlock-audit/2023-01-sentiment/blob/main/controller-52/src/rage/DNGMXVaultController.sol#L60-L73

https://github.com/sherlock-audit/2023-01-sentiment/blob/main/controller-52/src/rage/DNGMXVaultController.sol#L75-L88

Tool used

Manual Review

Recommendation

When decoding the data from the call to Rage Trade, confirm that receiver == msg.sender. Here's an example with the deposit function:

```
function canDeposit(bytes calldata data)
    internal
    view
    returns (bool, address[] memory, address[] memory)
{
    (address token, address receiver,) = abi.decode(
        data, (address, address, uint256)
    );
+   if (receiver != msg.sender) return (true, vault, new address[](0));

    address[] memory tokensOut = new address[](1);
    tokensOut[0] = token;

    return (true, vault, tokensOut);
}
```

Discussion

r0ohafza

Agree with the issue mentioned. Disagree with the fix provided since the receiver can be any other account and still lead to an accounting error, I think the recommended fix mentioned by @zobront on issue #10 will resolve this as well.