SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for:	Sentiment
Prepared by:	Sherlock
Lead Security Expert:	obront
Dates Audited:	January 18 - January 21, 2023
Prepared on:	June 14, 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

For the detailed scope, see the contest details.

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
7	0

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

obront
GalloDaSballo
0xdeadbeef
Bahurum

Issue M-1: 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 The corresponding Controller assumes that all tokens are withdrawn to the Sentiment account that called the function.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.

bahurum

Escalate for 50 USDC. I believe that this issue is Low severity. I filed it as a low severity myself (https://github.com/sherlock-audit/2023-01-sentiment-judging/issues/26). See the 3rd point in the Vulnerability Detail section. This issue will never lead to loss or lockup of funds. So by Sherlock judging criteria this is Low severity. Otherwise, please provide a scenario where this issue leads to a loss or lockup of

funds. The watson mentions some similiar issues previously judged as Medium. I didn't see those before or I would have escalated them as well for the same reason.

sherlock-admin

> Escalate for 50 USDC. I believe that this issue is Low severity. I filed it as a low severity myself (https://github.com/sherlock-audit/2023-01-sentiment-judging/issues/26). See the 3rd point in the Vulnerability Detail section. This issue will never lead to loss or lockup of funds. So by Sherlock judging criteria this is Low severity. Otherwise, please provide a scenario where this issue leads to a loss or lockup of funds. The watson mentions some similiar issues previously judged as Medium. I didn't see those before or I would have escalated them as well for the same reason.

You've created a valid escalation for 50 USDC!

To remove the escalation from consideration: Delete your comment. To change the amount you've staked on this escalation: Edit your comment **(do not create a new comment)**.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

hrishibhat

Escalation accepted

While the above issue is not considered a low in this case, considering issue #26 as a valid medium based on further discussions in issue #10

sherlock-admin

> Escalation accepted
>
> While the above issue is not considered a low in this case, considering issue #26 as a valid medium based on further discussions in issue #10

This issue's escalations have been accepted!

Contestants' payouts and scores will be updated according to the changes made on this issue.

Issue M-2: 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.

Discussion

bahurum

Escalate for 50 USDC. This issue should be low/informational. I have flagged the issue but as Informational (https://github.com/sherlock-audit/2023-01-sentiment-judging/issues/25), since the two target contracts are non-upgradeable and new functions can never be added and signature collision is impossible. Here are the contracts on chain: depositPeriphery withdrawPeriphery Note that they do not share any of the 3 function signatures of `DNGMXVaultController` and they never will since they are non-upradeable.

sherlock-admin

> Escalate for 50 USDC. This issue should be low/informational. I have flagged the issue but as Informational (https://github.com/sherlock-audit/2023-01-sentiment-judging/issues/25), since the two target contracts are non-upgradeable and new functions can never be added and signature collision is impossible. Here are the contracts on chain: depositPeriphery withdrawPeriphery Note that they do not share any of the 3 function signatures of `DNGMXVaultController` and they never will since they are non-upradeable.

You've created a valid escalation for 50 USDC!

To remove the escalation from consideration: Delete your comment. To change the amount you've staked on this escalation: Edit your comment **(do not create a new comment)**.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

zobront

Baharum's issue did not identify the signature collision as a risk for combining contracts into one Controller, and simply said that it would be confusing, which is why it was identified as Informational.

The issue here points out that the current Controller violates an explicit practice that Sentiment must be enforcing on all contracts to keep their protocol safe. Just because the collision doesn't happen in this specific case, it doesn't change the fact that Sentiment is violating a security principle that they should be upholding consistently.

To drive the point home, I do not believe Sentiment would get their contracts re-audited if they added one function to an individual Controller. So this type of issue needs to be caught NOW so that they don't set themselves up to be in a situation where adding one innocent, safe function later ends up causing a catastrophic problem.

zobront

Fix confirmed.

hrishibhat

Escalation accepted.

After further internal discussion, considering this issue as informational as there are no collision risks with current implementation and any changes to the code must undergo an audit process.

sherlock-admin

> Escalation accepted.
>
> After further internal discussion, considering this issue as informational as there are no collision risks with current implementation and any changes to the code must undergo an audit process.

This issue's escalations have been accepted!

Contestants' payouts and scores will be updated according to the changes made on this issue.

Issue M-3: 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.

Discussion

bahurum

Escalate for 50 USDC. I believe that this issue is Low severity. I filed a list of issues with same impact as a Low severity submission (https://github.com/sherlock-audit/2023-01-sentiment-judging/issues/26). This issue will never lead to loss or lockup of funds. So by Sherlock judging criteria this is Low severity. Otherwise, please provide a scenario where this issue leads to a loss or lockup of funds. The watson mentions some similiar issues judged as Medium in previous contests. I didn't see those before or I would have escalated them as well for the same reason.

sherlock-admin

> Escalate for 50 USDC. I believe that this issue is Low severity. I filed a list of issues with same impact as a Low severity submission (https://github.com/sherlock-audit/2023-01-sentiment-judging/issues/26). This issue will never lead to loss or lockup of funds. So by Sherlock judging criteria this is Low severity. Otherwise, please provide a scenario where this issue leads to a loss or lockup of funds. The watson mentions some

> similiar issues judged as Medium in previous contests. I didn't see those before or I would have escalated them as well for the same reason.

You've created a valid escalation for 50 USDC!

To remove the escalation from consideration: Delete your comment. To change the amount you've staked on this escalation: Edit your comment **(do not create a new comment)**.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

zobront

Baharum is correct that within the protocol, as it stands, there are no risks of an account having an incorrect Assets list.

But Sentiment is becoming increasingly incorporated into the Arbitrum DeFi ecosystem. Sentiment purports that their account Asset lists are accurate and they go to extreme lengths with the Controller setup (and take on additional security risk) to track tokens in and out to keep it accurate. If it didn't matter, they could just use the list of all allowed tokens when taking operations on an account.

As I stated in my Recommendations:

> [W]e need to ensure the data stays accurate, even in edge cases, or else there will be trickle down problems we can't currently predict.

I believe this is a valid Medium, and is the kind of thing that Sentiment needs to understand when releasing new Controllers with the explicit goal of keeping Asset lists accurate as users interact with a given protocol.

bahurum

Thank you for your reply.

While I agree that data accuracy is desired, I'd like to insist that IMO the impact of this issue is not within Sherlock's definition of Medium severity issue, since there is no path to loss of funds, unless the contrary is shown.

For integration, if an asset with zero balance is in an account's asset list, I don't see that as a problem since an external protocol can check if the balance is positive if a zero balance ever causes issues.

If Sherlock thinks differently then I encourage them to reconsider my Low severity issue #26 along with your 2 issues #10 and #5 as a Medium.

zobront

I agree that your Low should be Medium.

hrishibhat

Escalation accepted.

Considering issue #26 as a valid medium in this case

sherlock-admin

> Escalation accepted.
>
> Considering issue #26 as a valid medium in this case

This issue's escalations have been accepted!

Contestants' payouts and scores will be updated according to the changes made on this issue.

Issue M-4: 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

Discussion

zobront

Fix confirmed.

Issue M-5: `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-6: Risk with 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 plvGLP depositor 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-7: Tokens not owned by an account can be added as an asset to the account

Source: https://github.com/sherlock-audit/2023-01-sentiment-judging/issues/26

Found by

Bahurum

Summary

In the controllers `RewardRouterController`, `RewardRouterV2Controller` and `DNGMXVaultController` the function `canCall` can return in `tokenIn` a token address that has actually not been received by the account. If the account did not have the token before, than the token is added to the asset list of the account even if the account does not hold the token at all.

Vulnerability Detail

- in `RewardRouterController`: in `canCallCompound()`, `WETH` is added to `tokensIn` but no tokens are sent to the account as a result of the call to the Reward Router's function `compound()`
- in `RewardRouterV2Controller`: in `canCallRedeem()` the token redeemed is added to `tokensIn`, but the router's function `unstakeAndRedeemGlp()` allows to send the tokens to a 3rd party receiver instead of the caller. In such a case, nop tokens are sent to the account.
- in `DNGMXVaultController`: in `canWithdraw()` the token redeemed is added to `tokensIn`, but the DN GMX vault's functions `redeemToken()` and `withdrawToken()` allow to send the tokens to a 3rd party receiver instead of the caller. In such a case, nop tokens are sent to the account.

Impact

There can be tokens in the list of assets of an account that the account doesn't actually hold. Note that this does not pose any issues for the calculation of collateral.

Code Snippet

https://github.com/sherlock-audit/2023-01-sentiment/blob/main/controller-55/src/gmx/RewardRouterController.sol#L67

https://github.com/sherlock-audit/2023-01-sentiment/blob/main/controller-55/src/gmx/RewardRouterV2Controller.sol#L88

https://github.com/sherlock-audit/2023-01-sentiment/blob/main/controller-55/src/gmx/RewardRouterV2Controller.sol#L88

Tool used

Manual Review

Recommendation

No particular reccommendation.