SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared for: Volta
Prepared by: Sherlock
Lead Security Expert: 0x52
Dates Audited: February 15 - February 19, 2023
Prepared on: March 10, 2023

Introduction

Overcollateralized stablecoin protocol that enables users to borrow VOLT by staking yield generating assets, with 0% interest loans.

Scope

```
voltGNS.sol
VoltaVault.sol
FeesContract.sol
VoltaToken.sol
PriceSource/gDAIPriceSource.sol
PriceSource/VoltGNSPriceSource.sol
```

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
9	3

Issues not fixed or acknowledged

Medium	High
0	0

Security experts who found valid issues

0x52
rvierdiiev
hansfriese
clems4ever
carrot
TrungOre
CodingNameKiki
neumo
yixxas

Issue H-1: voltGNS#withdraw and redeem take fees from the wrong address when caller != owner

Source: https://github.com/sherlock-audit/2023-02-volta-judging/issues/24

Found by

0x52

Summary

ERC4626 vaults allow approved users to withdraw shares on behalf of another user. When this occurs fees are charged to msg.sender rather than the owner.

Vulnerability Detail

https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/voltGNS.sol#L152-L162

sendWithdrawFees and sendRedeemFees transfers fees from msg.sender rather than the specified owner. This is problematic when a user withdraws/redeems on behalf of another user because they will be forced to pay the fees instead.

Impact

Wrong user pays fees when one users withdraws/redeems on behalf of the other

Code Snippet

https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/voltGNS.sol#L152-L162

https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/voltGNS.sol#L164-L173

Tool used

Manual Review

Recommendation

sendWithdrawFees and sendRedeemFees should take shares from `owner` rather than `msg.sender()`

Discussion

0xLoopy

Fixed https://github.com/volta-finance/contracts/pull/9

IAm0x52

Fix looks good. Withdraw fees are now transferred from owner rather than msg.sender

Issue H-2: Price of `voltGNS` share is calculated incorrectly

Source: https://github.com/sherlock-audit/2023-02-volta-judging/issues/16

Found by

CodingNameKiki, TrungOre, hansfriese, 0x52, clems4ever

Summary

Forget to deduct the `boostedGNS` when calculating price

Vulnerability Detail

The price of voltGNS is updated whenever the function `voltGNS.swapToGNS()` is called.

The function firstly calculates the `totalStaked` which is amount of gns staking in `gnsVault` plus with amount of gns which was swapped from `pendingDai`. Then the price will be determined by dividing `totalStaked` with `totalSupply()` of the vault.

At the first glance, this formula seem right when calculating the price using the total underlying assets and the total shares supply. So it means that if I withdraw all the `totalSupply` shares, I can receive all the `totalStaked`. Unfortunately this won't be happened because of the `boostedGNS`. The `boostedGNS` is amount of gns which is deposited by the owner and no share will be minted for the him. Furthermore this amount can be deposited / withdrew at anytime whenever the owner wants.

This `boostedGNS` mechanism can lead to the big issue for the `VoltaVault` contract which uses the `voltGNS` as collateral. When the owner calls `unboostGNS()`, it can make the `voltGNS.price()` lose a lot of value. That will make the big liquidation events occured for a lot of users without noticing them.

The price of a token should depend on the fluctuation of the market, and should not depend on the single individual like owner. Malicious owner can abuse this to make profit.

Impact

When owner calls `unboostGNS()` can incur a immediate liquidation for users without their preparation.

Code Snippet

https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/voltGNS.sol#L201-L203

Tool used

Manual review

Recommendation

Deducting the `boostedGNS` out of `totalStaked` when calculating the price.

Discussion

IAm0x52

Dupe of #21

0xLoopy

Fixed https://github.com/volta-finance/contracts/pull/5

IAm0x52

Fix looks good. Boosted tokens are now correctly accounted for

Issue H-3: Liquidator receives less collateral then he should

Source: https://github.com/sherlock-audit/2023-02-volta-judging/issues/5

Found by

carrot, hansfriese, 0x52, rvierdiiev

Summary

Liquidator receives less collateral then he should

Vulnerability Detail

When `VoltaVault.liquidateVault` is called, then liquidator paysdebtofvault. Later, vault's collateral should be distributed among the liquidator and protocol. https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/VoltaVault.sol#L389-L397

```
uint256 collateralExtract = vaultCollateral[vaultID];

uint liquidatorDiscount = 100e18 * 1e18 / _minimumCollateralPercentage;
uint protocolFees = 1e18 - ((1e18 - liquidatorDiscount) / 2);
uint protocolEarned = collateralExtract * protocolFees / 1e18;


collateral.safeTransfer(treasury, protocolEarned);
vaultCollateral[vaultID] = 0;
collateral.safeTransfer(msg.sender, collateralExtract - protocolEarned);
```

According to the documentation:

> For example, let's consider voltGNS with a minimum ratio of 150%, which is equivalent to a 66% Loan to Value ratio. If the LTV of voltGNS reaches 66%, it becomes eligible for liquidation. In this case, the liquidator would receive half of the 34% discount, or 17%, and the platform would receive the remaining 17% as fees. If the collateral is worth $1000, the liquidator would get it for $830, while the platform would receive $170 in fees.

The problem that in the code, it's protocol who receives $830 and liquidator receives only $170.

You can check it with this simple code in remix which uses code from protocol. Just run this in the remix and see result.

```
contract Test {
    function test() external virtual pure returns (uint256, uint256) {
```

```
        uint256 collateralExtract = 100 ether;
        uint256 ratio = 150 ether;
        uint liquidatorDiscount = 100e18 * 1e18 / ratio;
        uint protocolFees = 1e18 - ((1e18 - liquidatorDiscount) / 2);
        uint protocolEarned = collateralExtract * protocolFees / 1e18;
        return (protocolEarned, collateralExtract - protocolEarned);
        //returns 83333333333333333300 for portocol and 16666666666666666700 for
↪   liquidator
        //so protocol receive more
    }
}
```

Impact

Liquidator lose big part of funds.

Code Snippet

Provided above

Tool used

Manual Review

Recommendation

Swap these amounts and send them to correct recipient.

Discussion

0xLoopy

Fixed https://github.com/volta-finance/contracts/pull/7

IAm0x52

Fixes look good. Liquidation logic has been moved to a separate function liquidateVaultRewards. earnFees calculation has been corrected.

Issue M-1: Depositing GNS might revert before reaching the maximum threshold

Source: https://github.com/sherlock-audit/2023-02-volta-judging/issues/27

Found by

rvierdiiev, clems4ever

Summary

The deposit function is checking the balance after deposit to verify that it does not exceeds the maximum threshold `maxGNSDeposit` set into the contract. However, the check does not take the fees into account. Therefore, the function might revert in some cases even though the final amount of deposited GNS is not exceeding the maximum threshold

Vulnerability Detail

The deposit function is checking the future balance of GNS against the maximum threshold set into the contract. This is done in the function below:

```
function deposit(uint256 assets, address receiver) public virtual override
↪   returns (uint256) {
        require(totalAssets() + assets <= maxGNSDeposited(), "GNS deposits more
↪   than max"); <==========================
    compound();

    assets = sendDepositFees(assets);

        uint256 shares = super.deposit(assets, receiver);

    stakeGNS();

        return shares;
    }
```

However, as we can see, the fees are computed after the check. Therefore the final amount of GNS deposited should be `totalAssets() + assets - fees` instead of `totalAssets() + assets`.

This can lead to situations where the transaction reverts when it should not. Note that this is properly handled in the mint function though because the check is done once every amounts have been taking into account (deposit, fees and rewards).

Impact

`deposit` might revert while it should not.

Code Snippet

https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/voltGNS.sol#L59

https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/voltGNS.sol#L78

Tool used

Manual Review

Recommendation

Replicate what is done in `mint`, i.e., make the check after all amounts are taken into account in the balance.

Discussion

0xLoopy

Fixed https://github.com/volta-finance/contracts/pull/11

IAm0x52

Fix looks good. Deposit limit is now checked after fees are taken to prevent a premature revert.

Issue M-2: gDAIPriceSource and VoltGNSPriceSource will return an incorrect price in the event DAI depegs

Source: https://github.com/sherlock-audit/2023-02-volta-judging/issues/23

Found by

0x52, neumo

Summary

gDAIPriceSource and VoltGNSPriceSource#getPrice returns the share value in terms of DAI. If DAI were to depeg then the vault would continue to give out loans under the assumption that DAI is still worth $1.

Vulnerability Detail

https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/PriceSource/gDAIPriceSource.sol#L17-L19

Here we see that the price source simply returns the current share price in terms of DAI. This works fine if DAI remains pegged to $1. In the event that DAI depegs then the valuation of the gDAI collateral will be completely wrong and will result in the protocol taking on a lot of bad debt.

This issue also affects VoltGNSPriceSource because GNS will be greatly overvalued

Impact

Protocol will take on a large amount of bad debt if DAI depegs

Code Snippet

https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/PriceSource/gDAIPriceSource.sol#L17-L19

https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/PriceSource/VoltGNSPriceSource.sol#L23-L25

Tool used

Manual Review

Recommendation

Use the DAI/USD chainlink oracle to protect against DAI variations from $1

Discussion

0xLoopy

Fixed https://github.com/volta-finance/contracts/pull/8

IAm0x52

Fix looks good. DAI is now valued via a Chainlink oracle

Issue M-3: VoltGNSPriceSource does not consider withdraw fee when calculating worth of VoltGNS collateral

Source: https://github.com/sherlock-audit/2023-02-volta-judging/issues/22

Found by

0x52

Summary

VoltGNS can have a withdraw fee that is taken when it is withdrawn. If this is the case then the value of VoltGNS should take this withdraw fee into consideration to properly value the collateral.

Vulnerability Detail

https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/PriceSource/VoltGNSPriceSource.sol#L23-L25

Here the exchange ratio of the vault and GNS price are used directly without considering the withdraw fee to remove the GNS from the vault.

Impact

Collateral will be valued incorrectly when there is a fee, leading to potentially bad loans

Code Snippet

https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/PriceSource/VoltGNSPriceSource.sol#L12-L26

Tool used

Manual Review

Recommendation

Query the vault to see if there is a withdraw fee. If there is then remove the corresponding amount of value from the collateral price.

Discussion

0xLoopy

Withdrawing fees shouldn't affect voltGNS price.

IAm0x52

Liquidators usually only cares to get their initial asset back (volt, dai, etc) which would require the liquidator to withdraw and trade the GNS hence losing the value of the withdraw fee. Which is why the value of the collateral should be reduced by the withdraw fee.

In a more general sense. Realizing any value from voltGNS would require withdrawing so the fee should be removed from it's valuation.

0xLoopy

I don't agree with assuming that liquidators will for sure swap voltGNS, even if that is the case that should be calculated and handled by the liquidator and not affect the voltGNS price.

hrishibhat

Along with the comments above, Considering the impact of this issue as a low.

IAm0x52

Escalate for 1 USDC

This should be medium. It is nearly identical with the issue awarded in the Isomorph contest

https://github.com/sherlock-audit/2022-11-isomorph-judging/issues/120

sherlock-admin

> Escalate for 1 USDC
>
> This should be medium. It is nearly identical with the issue awarded in the Isomorph contest
>
> https://github.com/sherlock-audit/2022-11-isomorph-judging/issues/120

You've created a valid escalation for 1 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.

After further discussion, and considering the previous precedence, it is agreed that the corresponding amount of value from the collateral price in case of fee, if not could result in bad loans. Quoting the Lead Watson:

> Ultimately the liquidator is responsible for evaluating the profitability but if the value doesn't account for the fee then the window in which it is profitable is smaller than expected and then no one will liquidate it and it will just become bad debt for the protocol

sherlock-admin

> Escalation accepted.
>
> After further discussion, and considering the previous precedence, it is agreed that the corresponding amount of value from the collateral price in case of fee, if not could result in bad loans. Quoting the Lead Watson:
>
> > Ultimately the liquidator is responsible for evaluating the profitability but if the value doesn't account for the fee then the window in which it is profitable is smaller than expected and then no one will liquidate it and it will just become bad debt for the protocol

This issue's escalations have been accepted!

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

IAm0x52

Sponsor has acknowledged the risk associated with this issue and have opted not implement changes to address it.

Issue M-4: swapToGNS doesn't add any slippage protection allowing caller to steal all swapped GNS

Source: https://github.com/sherlock-audit/2023-02-volta-judging/issues/20

Found by

carrot, yixxas, rvierdiiev, 0x52, clems4ever

Summary

The uniswapV3Swap call doesn't implement any kind of slippage protection allowing it to be sandwich attacked.

Vulnerability Detail

https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/voltGNS.sol#L195-L196

We can see here that it calls the swap with not slippage protection at all. This allows the caller to sandwich attack the trade and steal all the DAI being swapped.

Impact

DAI being swapped can be stolen via sandwich attack

Code Snippet

https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/voltGNS.sol#L189-L205

Tool used

Manual Review

Recommendation

Implement a percentage slippage protection using the price of GNS

Discussion

0xLoopy

Fixed https://github.com/volta-finance/contracts/pull/12

yixxas

Escalate for 1 USDC

Considering how assets can be stolen with no precondition and with `minAmount` being hardcoded to 0, there is no way for protocol to fix this slippage issue if deployed, high severity seems to be more appropriate.

sherlock-admin

> Escalate for 1 USDC
>
> Considering how assets can be stolen with no precondition and with `minAmount` being hardcoded to 0, there is no way for protocol to fix this slippage issue if deployed, high severity seems to be more appropriate.

You've created a valid escalation for 1 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 rejected

Considering the fee required to move the pool price up and down, the sandwich attack may not always be profitable. This should be a valid medium

sherlock-admin

> Escalation rejected
>
> Considering the fee required to move the pool price up and down, the sandwich attack may not always be profitable. This should be a valid medium

This issue's escalations have been rejected!

Watsons who escalated this issue will have their escalation amount deducted from their next payout.

IAm0x52

Preliminary review completed. Waiting for response on additional changes

IAm0x52

Fixes look good. Additional changes made here to update the fee calculation

Issue M-5: Protocol fees will make some liquidations unprofitable if debt is too far underwater

Source: https://github.com/sherlock-audit/2023-02-volta-judging/issues/18

Found by

0x52

Summary

When calculating liquidator discount it always uses _minimumCollateralPercentage instead of the the actual collateralization of the debt. If the debt goes too far underwater then the protocol fees will be higher than the actual discount making it unprofitable to liquidate.

Vulnerability Detail

https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/VoltaVault.sol#L389-L393

When calculating liquidator discount it always uses _minimumCollateralPercentage. This means that the liquidatorDiscount will be incorrect if the actual ratio is lower than the minimum ratio. Since it takes fees based on liquidatorDiscount it could cost the liquidator more in VOLT than they would receive in collateral.

Example: Assume _minimumCollateralPercentage = 150%. This gives a liquidator discount of 33%. Now imagine that a position being liquidated only has a collateralization of 110%. This gives a liquidator discount of 9%. Since the protocol takes half, it will take 16.33% of the collateral which will make the liquidation unprofitable.

It might seem impossible for the percentage to get low enough to make this an issue but in the event the sequencer goes down liquidations would be delayed.

Impact

Liquidations are unprofitable if debt is too far underwater

Code Snippet

https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/VoltaVault.sol#L354-L407

Tool used

Manual Review

Recommendation

Use the actual collateral percentage rather than the minimum to calculate the discount.

Discussion

IAm0x52

This is a separate issues from all of the dupes. They point out that `protocolFees` are calculated incorrectly. This issue points out that the 'liquidatorDiscount' is calculated incorrectly.

0xLoopy

Fixed this and #5 https://github.com/volta-finance/contracts/pull/7

IAm0x52

Fixes look good. It addresses the issue of unprofitable liquidations by giving the liquidator the entire vault balance if fees would cause vault to become unprofitable.

Issue M-6: Adversary can DOS vault repayments by making dust repayments

Source: https://github.com/sherlock-audit/2023-02-volta-judging/issues/17

Found by

hansfriese, 0x52

Summary

payBackToken will revert if the repayment amount is greater than the amount of debt owed by the vault. Adversary can DOS repayments by making a dust payment and causing a revert.

Vulnerability Detail

https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/VoltaVault.sol#L261-L273

VoltaVault#payBackToken requires that the repayment amount is less than or equal to the amount of debt owed by the vault. This can be frontrun with dust repayment to cause repayment to revert.

Impact

Adversary can prevent repayment

Code Snippet

https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/VoltaVault.sol#L261-L290

Tool used

Manual Review

Recommendation

If repayment amount is greater than vault debt, reduce the repayment amount to the amount owed by the vault

Discussion

0xLoopy

Fixed https://github.com/volta-finance/contracts/pull/6

IAm0x52

Fix looks good. If user attempts to pay back more than they owe it will payback everything they owe rather than reverting.

Issue M-7: Initial user of voltGNS vault can abuse rounding error to steal tokens

Source: https://github.com/sherlock-audit/2023-02-volta-judging/issues/10

Found by

carrot, yixxas, rvierdiiev, TrungOre, hansfriese, clems4ever

Summary

ERC4626 vault contracts suffer from a commonly known vulnerability due to division rounding as described in this report TOB-YEARN-003. The initial user can massively skew the ratio of shares to assets with a large "donation", and can profit off of later users.

Vulnerability Detail

In VoltGNS vault, the deposit function has no minimum deposit limit or burn. https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/voltGNS.sol#L58-L69 This calls the ERC4626 deposit function which is defined as

```
function deposit(uint256 assets, address receiver) public virtual override
↪  returns (uint256) {
        require(assets <= maxDeposit(receiver), "ERC4626: deposit more than
↪  max");

        uint256 shares = previewDeposit(assets);
        _deposit(_msgSender(), receiver, assets, shares);

        return shares;
    }
```

Which eventually calls `_convertToShares` to calculate the number of shares,

```
function _convertToShares(uint256 assets, Math.Rounding rounding) internal view
↪  virtual returns (uint256 shares) {
        uint256 supply = totalSupply();
        return
            (assets == 0 || supply == 0)
                ? _initialConvertToShares(assets, rounding)
                : assets.mulDiv(supply, totalAssets(), rounding);
    }
```

This function takes into account the `totalAssets()`, which has been overridden and re-defined as https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/voltGNS.sol#L260-L263 returning the sum of current balance and staked amount.

However, due to rounding errors, an attacker who deposits to a fresh vault can skew the ratio by transferring in a large quantity of GNS tokens by following these steps

1. Attacker deposits `1 wei` of GNS token, minting `1 wei` of voltGNS share
2. Attacker then sends a large amount (`1e18 wei`) of GNS tokens to the vault.
3. Normal user comes and deposits `2e18` wei of GNS tokens. Shares calculated = 2e18 * 1 / (1e18+1) = 1 wei
4. User only gets minted `1 wei` of voltGNS token, giving him claim to only half the vault, which has `3e18 wei` tokens, instantly losing him `0.5e18` tokens which is profited by the attacker.

Impact

Exploit by first user of vault. Loss of tokens by other users

Code Snippet

https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/voltGNS.sol#L58-L69

Tool used

Manual Review

Recommendation

Burn the first (`if totalSupply() == 0`) 10000wei of minted shares by sending them to address 0. This will make the cost to pull of this attack unfeasable.

Discussion

0xLoopy

Fixed https://github.com/volta-finance/contracts/pull/4

hrishibhat

Considering this issue a valid medium as it requires the first depositor to be the attacker

IAm0x52

Preliminary review completed. Waiting for response on additional changes

IAm0x52

Secondary fix looks good. With OZ ERC4626 new accounting requiring shares minted (shares > 0) on deposit will prevent loss to legitimate users. Additionally treasury will mint and hold initial shares.

Issue M-8: voltGNS.mint doesn't allow to mint for first depositor

Source: https://github.com/sherlock-audit/2023-02-volta-judging/issues/6

Found by

rvierdiiev

Summary

voltGNS.mint doesn't allow to mint for first depositor

Vulnerability Detail

https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/voltGNS.sol#L71-L83

```
function mint(uint256 shares, address receiver) public virtual override returns
↪  (uint256) {
    require(totalAssets() > 0);
    compound();


    shares = sendMintFees(shares);
    uint256 assets = super.mint(shares, receiver);


    require(totalAssets() <= maxGNSDeposited(), "GNS deposits more than max");


    stakeGNS();


    return assets;
}
```

voltGNS.mint doesn't allow to call itself, when totalAssets is 0. I believe that there is no reasons for that and it just creates bad experience for the first user who will try to mint shares for himself in the voltGNS.

If fore some reasons i am wrong and there is reason why to limit first depositor to call `mint` then there is a simple way to avoid that. Just donate 1 wei of GNS into voltGNS and then call `mint`.

Impact

First depositor can't `mint` shares.

Code Snippet

Provided above

Tool used

Manual Review

Recommendation

Allow deposit when `totalAssets` is 0.

Discussion

0xLoopy

Fixed https://github.com/volta-finance/contracts/pull/2

IAm0x52

Fix looks good. Require statement has been removed

Issue M-9: voltGNS.compound is not called for voltGNS.withdraw and redeem

Source: https://github.com/sherlock-audit/2023-02-volta-judging/issues/3

Found by

clems4ever, carrot, rvierdiiev, hansfriese, neumo

Summary

compound is not called for voltGNS.withdraw and redeem. User lose funds because of that.

Vulnerability Detail

`voltGNS.compound` is harvesting rewards from `gnsVault`, then swap them into GNS and then stake them back to the vault. Because of this `totalAssets` is increasing and stakers earn more GNS. This function is called at the top of `deposit` and `mint` function, but it's not called inside `withdraw` and `redeem`. Because of that, users that are going to withdraw from voltGNS are losing some funds.

Impact

Lose of funds for withdrawers.

Code Snippet

https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/voltGNS.sol#L175-L187 https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/voltGNS.sol#L60 https://github.com/sherlock-audit/2023-02-volta/blob/main/contracts/voltGNS.sol#L85-L116

Tool used

Manual Review

Recommendation

Call `compound` in both `withdraw` and `redeem` functions.

Discussion

0xLoopy

Fixed https://github.com/volta-finance/contracts/pull/1

IAm0x52

Fix looks good. Compound is now called during redeem and withdraw