GMD Security Review

Pashov Audit Group

Conducted by: pashov

January 19th, 2023

Contents

1. About pashov

Krum Pashov, or **pashov**, is an independent smart contract security researcher. Having found numerous security vulnerabilities in various protocols, he does his best to contribute to the blockchain ecosystem and its protocols by putting time and effort into security research & reviews. Check his previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where I try to find as many vulnerabilities as possible. I 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.

3. Introduction

A time-boxed security review of the **GMD** protocol was done by **pashov**, with a focus on the security aspects of the application's smart contracts implementation.

4. About GMD

The GMD protocol is built on top of GMX and allows you to stake a token (USDC, ETH, BTC) to earn some APY. It fits into the Yield Aggregator category. Users can enter the vault with either a native or an ERC20 asset. The admin takes care of claiming rewards and compounding them. Currently supported assets are `USDC`, `WETH` and `wBTC`.

5. Risk Classification

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

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash **-** **c27c012fde95d9e2ac40fc0fe795489b76284eee**

Scope

The following smart contracts were in scope of the audit:

- `final_vault`

7. Executive Summary

Over the course of the security review, pashov engaged with GMD to review GMD. In this period of time a total of **24** issues were uncovered.

Protocol Summary

Protocol Name	GMD
Date	January 19th, 2023

Findings Count

Severity	**Amount**
Medium	7
Low	6
QA	11
Total Findings	**24**

Summary of Findings

ID	Title	Severity	Status
[M-01]	Hardcoded handling of non-18 decimal token pools limits the interoperability of the protocol	Medium	Resolved
[M-02]	As protocol relies heavily on admin actions, single-step ownership transfer pattern is dangerous	Medium	Resolved
[M-03]	If addPool is called too many times it can brick core functionality	Medium	Resolved
[M-04]	Call to updatePoolRate is missing	Medium	Resolved
[M-05]	Admin privilege actions can be risky for users	Medium	Resolved
[M-06]	Token approvals & allowances management is flawed	Medium	Resolved
[M-07]	Inverted slippage protection approach can lead to problems	Medium	Resolved
[L-01]	Inconsistent input validation	Low	Resolved
[L-02]	Storage variable is only written to but never read from	Low	Resolved
[L-03]	Missing parameter validation in enter	Low	Resolved
[L-04]	The IWETH interface has a method that WETH does not have	Low	Resolved
[L-05]	Code is calling a deprecated method	Low	Resolved
[L-06]	Value of slippage protection arguments is not set	Low	Resolved
[QA-01]	Using SafeMath when compiler is ^0.8.0	QA	Resolved

[QA-02]	leaveETH should not be payable	QA	Resolved
[QA-03]	Unused storage variable	QA	Resolved
[QA-04]	Misleading comments throughout the code	QA	Resolved
[QA-05]	Code is not properly formatted	QA	Resolved
[QA-06]	NatSpec missing from external functions	QA	Resolved
[QA-07]	Comment has no meaning	QA	Resolved
[QA-08]	Mismatch between the filename and the contract name	QA	Resolved
[QA-09]	Remove unused import	QA	Resolved
[QA-10]	Not all require statements have an error string	QA	Resolved
[QA-11]	External calls can be grouped together	QA	Resolved

8. Findings

8.1. Medium Findings

[M-01] Hardcoded handling of non-18 decimal token pools limits the interoperability of the protocol

Severity

Likelihood: Medium, because even though at this point no such pools are added, it is possible that they are in the future

Impact: Medium, because it limits the functionality of the protocol

Description

The `enter` method implements specific handling for `USDC` and `wBTC` tokens, because they have decimals that are not equal to 18. This should be done for all such pools tokens, but since it is hardcoded it is not extensible - for example `USDT` pool can't be added.

Recommendations

Redesign the approach with the decimals that is hardcoded or implement it in an extensible-friendly way for new non-18 decimal token pools.

Discussion

pashov: Client has fixed the issue.

[M-02] As protocol relies heavily on admin actions, single-step ownership transfer

pattern is dangerous

Severity

Likelihood: Low, because it requires admin error when transferring ownership

Impact: High, because it bricks core protocol functionality

Description

Inheriting from OpenZeppelin's `Ownable` contract means you are using a single-step ownership transfer pattern. If an admin provides an incorrect address for the new owner this will result in none of the `onlyOwner` marked methods being callable again. The better way to do this is to use a two-step ownership transfer approach, where the new owner should first claim its new rights before they are transferred.

Recommendations

Use OpenZeppelin's `Ownable2Step` instead of `Ownable`

Discussion

pashov: Client has acknowledged the issue.

[M-03] If `addPool` is called too many times it can brick core functionality

Severity

Likelihood: Low, because it requires a malicious admin or a big admin error

Impact: High, because it bricks core protocol functionality

Description

The `addPool` method pushes an entry to the `poolInfo` array. Methods like `swapGLPout` and `recoverTreasuryTokensFromGLP` have internal calls (`GLPbackingNeeded`) that iterate over the whole array. If too many pools are added then all calls to those methods will need too much gas to iterate over the array and if this cost is over the block gas limit it will lead to a DoS situation of core functionality.

Recommendations

Limit the number of pools that can be added, for example to 50.

Discussion

pashov: Client has fixed the issue.

[M-04] Call to `updatePoolRate` is missing

Severity

Likelihood: Medium, because it happens only for a paused and then resumed pool

Impact: Medium, because it can hardly lead to big losses

Description

Every time the `totalStaked` amount of a pool is updated, the `updatePoolRate` method is called to update the `EarnRateSec`. This is not true for the `pauseReward` method, which calls `updatePool` that changes the `totalStaked` amount. Now if a pool is paused, when it gets resumed again and `updatePool` is called it will calculate less rewards than it should had, because `EarnRateSec` was not updated.

Recommendations

Call `updatePoolRate` after the `updatePool` call in `pauseReward`

Discussion

pashov: Client has fixed the issue.

[M-05] Admin privilege actions can be risky for users

Severity

Likelihood: Low, because it requires a malicious/compromised admin

Impact: High, because it can brick the protocol

Description

The methods `updateOracle`, `updateRouter` and `updateRewardRouter` are admin controllable and callable anytime. Same for the `withdrawable` property of `PoolInfo`. A malicious/compromised admin can either provide non-existing addresses or set the `withdrawable` property to false for all pools, leading to a DoS for users of the protocol.

Recommendations

Consider using an role-based access control approach instead of a single admin role as well as a timelock for important admin actions.

Discussion

pashov: Client has acknowledged the issue.

[M-06] Token approvals & allowances management is flawed

Severity

Likelihood: Medium, because it will be problematic only with special type of tokens

Impact: Medium, because it can lead to a limited loss of funds

Description

There are a few problems related to approvals & allowances in the contract. One is that the `swaptoGLP` method approves another contract to spend tokens, but some tokens (like `USDT`) have approval race condition protection, which requires the allowance before a call to `approve` to already be either 0 or `UINT_MAX`. If this is not the case, the call reverts, which can lead to a DoS situation with `swaptoGLP`. It looks like there was an idea to mitigate this, because at all places (apart from in `convertDust`) after calling the `swaptoGLP` method there is an `approve` call for 0 allowance, but it is done to the wrong address. `swaptoGLP` always approves `poolGLP` but the 0 allowance `approve` call is always to the `_GLPRouter` when the `_GLPRouter` should never have allowance.

Recommendations

Set allowance to zero after each `swaptoGLP` call for the `poolGLP` address

Discussion

pashov: Client has fixed the issue.

[M-07] Inverted slippage protection approach can lead to problems

Severity

Likelihood: Low, because it needs more than one special condition simultaneously

Impact: Medium, because it can lead to limited amount of funds lost from the protocol

Description

Both the `leaveETH` and `leave` methods use the `slippage` storage variable to implement slippage protection for the users leaving the vault. The problem is that the slippage protection is done in an unusual approach which can result in problems. Both methods call the `swapGLPto` method which has the `min_receive` parameter that is passed to the `unstakeAndRedeemGlp` method in `GLPRouter`. The usual approach to slippage protection is to calculate how much tokens you expect to receive after a swap, let's say 100 $TKN, and then apply some slippage tolerance percentage to it - if the tolerance is 5% then the minimum expected tokens received is 95 $TKN. The protocol implemented a different approach, instead of providing a smaller expected received value it actually inflates the value to be sent for the swap.

```
uint256 percentage = 100000 - slippage;
uint256 glpPrice = priceFeed.getGLPprice().mul(percentage).div(100000);
uint256 glpOut = amountOut.mul(10**12).mul(tokenPrice).div(glpPrice).div
  (10**30);
```

As you see, the way it works is "expecting" a lower price of $GLP which means the protocol always sends more $GLP than needed to swap. Now if the slippage protection is bigger than the deposit fee this can be used as a griefing attack vector by depositing and then withdrawing from a vault multiple times to drain the pool's $GLP balance.

Recommendations

Think about redesigning the `leave` methods so that you make the user pay the slippage cost instead of the protocol.

Discussion

pashov: Client has acknowledged the issue.

8.2. Low Findings

[L-01] Inconsistent input validation

`APR` has a maximum value of 1599 when calling `addPool`, but can be 3999 when calling `setAPR`. Also `glpFees` has a maximum value of 700 when adding a pool but when you call the setter method `setGLPFees` it can be 999. Make sure the input validation is consistent throughout the system.

Discussion

pashov: Client has fixed the issue.

[L-02] Storage variable is only written to but never read from

The `GLPbacking` storage variable is only written to in `updateGLPbackingNeeded` but is never actually read from - this also means that the calls to `updateGLPbackingNeeded` are useless as they have no effect. If an external actor wants to get the "GLPbacking" he can just call the `GLPbackingNeeded` view function. Remove the `updateGLPbackingNeeded` method and the `GLPbacking` storage variable, or if some logic was not implemented - add it

Discussion

pashov: Client has fixed the issue.

[L-03] Missing parameter validation in `enter`

`enterETH` has a check if `msg.value > 0` but `enter` does not check if `_amountin > 0`. Add that check.

Discussion

pashov: Client has fixed the issue.

[L-04] The `IWETH` interface has a method that `WETH` does not have

The `safeTransfer` method is not part of the usual `IWETH` interface and is not actually used in the code, so it should be removed.

Discussion

pashov: Client has fixed the issue.

[L-05] Code is calling a deprecated method

The `safeApprove` method from the `SafeERC20` library is deprecated so it should not be used.

Discussion

pashov: Client has acknowledged the issue.

[L-06] Value of slippage protection arguments is not set

The `swaptoGLP` method does a `mintAndStakeGlp` call that has a 0 value for both `_minUsdg` and `_minGlp`. Also, in `recoverTreasuryTokensFromGLP` the `min_receive` parameter of the call to the `swapGLPto` method is 0 as well. This can hardly be exploited by the mechanics/tokenomics of `GLP` but it is still smart to add `minReceive` parameters to be provided by the user from external functions and pass them to the `swapToGLP` calls.

Discussion

pashov: Client has acknowledged the issue.

8.3. QA Findings

[QA-01] Using `SafeMath` when compiler is ^0.8.0

There is no need to use `SafeMath` when compiler is ^0.8.0 because it has built-in under/overflow checks.

Discussion

pashov: Client has acknowledged the issue.

[QA-02] `leaveETH` should not be `payable`

The `leaveETH` method only transfers ETH out, so `payable` keyword should be removed from its signature.

Discussion

pashov: Client has fixed the issue.

[QA-03] Unused storage variable

Storage variable `gdUSDC` is unused and should be removed.

Discussion

pashov: Client has fixed the issue.

[QA-04] Misleading comments throughout the code

Almost all comments that contain the words `usdc` or `gdUSDC` in the code are misleading and stale and should be removed or updated.

Discussion

pashov: Client has acknowledged the issue.

[QA-05] Code is not properly formatted

Run a formatter on the code, for example use the `prettier-solidity` plugin.

Discussion

pashov: Client has fixed the issue.

[QA-06] NatSpec missing from external functions

Add NatSpec docs for all external functions so their intentions and signatures are clear.

Discussion

pashov: Client has acknowledged the issue.

[QA-07] Comment has no meaning

This comment - `// Info of each user that stakes LP tokens.` has no meaning and it looks like it was related to a storage variable that is now gone. Remove it.

Discussion

pashov: Client has fixed the issue.

[QA-08] Mismatch between the filename and the contract name

While the file is named `final_vault.sol` the contract is named `vault` - rename file to `Vault.sol` and contract to `Vault`

Discussion

pashov: Client has fixed the issue.

[QA-09] Remove unused import

Remove the `import "@openzeppelin/contracts/token/ERC20/ERC20.sol";` import as it is unused.

Discussion

pashov: Client has fixed the issue.

[QA-10] Not all `require` statements have an error string

Since the project is using a compiler that is newer than version 0.8.4 it is best to use Solidity Custom Errors for error situations - replace all require statements with such custom errors.

Discussion

pashov: Client has acknowledged the issue.

[QA-11] External calls can be grouped together

The `RewardRouter` smart contract has the `handleRewards` function, which can be used in the place of `cycleRewardsETHandEsGMX` and `_cycleRewardsETH` `methods` - code

Discussion

pashov: Client has fixed the issue.