Bloom Security Review

Pashov Audit Group

Conducted by: pashov

June 24th, 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 **Bloom** protocol was done by **pashov**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Bloom

Bloom is a protocol that allows lenders & borrowers to deposit tokens, leverage their position and swap it for a US treasury bill ETF ($ib01) which is effectively earning the federal funds rate, while there is a small fee for the treasury. The protocol works in stages (named `Commit`, `ReadyPreHoldSwap`, `PendingPreHoldSwap`, `Holding` etc) and each `BloomPool` has a duration which will follow the underlying ETF duration.

Here is an example use case:

1. Alice (KYC'd) deposits 20k tokens as a borrower. To the protocol, this is a borrow order of 20k * leverage. The 20k deposited tokens will be used as collateral.
2. Bob (anon) deposits 20k * leverage tokens as a lender order.
3. Now Chris comes and deposits 15 tokens as a lender, but since the borrow and lend demand has been matched perfectly his tokens are unused and sent back to him (first come, first serve principle)
4. Commit phase ends, meaning all remaining tokens that didn't have a matched demand (either lender or borrower deposits) can be sent back, and now the pool holds the matched lending/borrowing demand tokens. This is the `ReadyPreHoldSwap` state - the pool holds the funds, they are not swapped for an ETF yet
5. Now the matched demand tokens are used to initiate a swap
6. A whitelisted (separate from the borrower whitelist) address executes an actual swap between the underlying tokens and the "billy token" (the treasury bill ETF token)
7. Now the pool contract will be in a `Holding` state, holding the billy tokens until maturity
8. From there a "post hold swap" can be initiated by anyone after the pool phase ends
9. Again, a whitelisted address executes the actual swap, but this time swapping billy tokens for underlying tokens
10. A swap completion is triggered which calculates the underlying tokens payouts for lenders and borrowers
11. Finally everyone can withdraw their initial deposit + interest (payout)

Observations

To be eligible to be a borrower in Bloom, you go through an off-chain KYC process and then your address gets whitelisted. The reason for this are regulations around purchasing of treasury bills, since the borrowers actually do this.

The economical model is:

- Borrowers are paying an interest rate of the Federal Funds Rate - 1 % (which is around 4.25%)
- Borrowers will be yielding the Federal Funds Rate (which is around 5.25%)
- Basically borrowers will be earning the spread between the Federal Funds Rate and the interest they are paying (5.25 - 4.25 = 1%) multiplied by the leverage they are using (let's say 50x leverage) and subtracting the protocol fees
- Lenders on the other hand will receive the interest from borrowers (as mentioned around 4.25%) minus the protocol fees - the benefit is that they don't have to go through KYC but would still get a yield that is close to the ETF rate
- The protocol earns fees which are percentage of the lender and borrower returns

Threat Model

Privileged Roles & Actors

- Borrower - deposits collateral to do a leverage borrow to earn the spread between interest and ETF rate
- Lender - deposits value for borrowers and earns interest
- SwapFacility Owner - can set spread price and pool address in `SwapFacility`
- Market maker - will be whitelisted in `SwapFacility`, can call `swap` to execute actual swaps between underlying and billy tokens

Security Interview

Q: What in the protocol has value in the market?

A: The BloomPool tokens that are minted and the treasury bill tokens that the protocol holds and swap

Q: In what case can the protocol/users lose money?

A: If they can't withdraw their initial deposit or receive less interest/reward than expected.

Q: What are some ways that an attacker achieves his goals?

A: Exploit interest calculations or force methods to revert.

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 **-** **744ad16b7166bcaca4b74f870a7a851e8bb0c38c**

fixes review commit hash **-** **8fa7239ef0c9bc677b5ef3720ededf88547c97de**

Scope

The following smart contracts were in scope of the audit:

- `BloomPool`
- `BPSFeed`
- `MerkleWhitelist`
- `SwapFacility`
- `interfaces/*`
- `lib/*`

7. Executive Summary

Over the course of the security review, pashov engaged with Bloom to review Bloom. In this period of time a total of **7** issues were uncovered.

Protocol Summary

Protocol Name	Bloom
Date	June 24th, 2023

Findings Count

Severity	**Amount**
Critical	1
High	1
Medium	4
Low	1
Total Findings	**7**

Summary of Findings

ID	Title	Severity	Status
[C-01]	Scaling of underlyingTokenPrice will leave funds stuck in protocol	Critical	Resolved
[H-01]	The protocol does not implement slippage checks on swaps	High	Resolved
[M-01]	The swap mechanism does not have a deadline parameter	Medium	Resolved
[M-02]	Missing price feed validation and usage of a deprecated method can lead to 0 price	Medium	Resolved
[M-03]	Centralization attack vectors are present	Medium	Resolved
[M-04]	Tokens with a fee-on-transfer mechanism will break the protocol	Medium	Resolved
[L-01]	Usage of safeApprove might be incompatible with tokens like USDT	Low	Resolved

8. Findings

8.1. Critical Findings

[C-01] Scaling of `underlyingTokenPrice` will leave funds stuck in protocol

Severity

Impact: High, as all deposited funds will be stuck in the protocol

Likelihood: High, as it will always happen

Description

The `_getTokenPrices` method in `SwapFacility` has the following code:

```
underlyingTokenPrice =
    uint256(IOracle(underlyingTokenOracle).latestAnswer()) *
    1e12;
```

This scaling by `1e12` is an error, because most oracle price feeds in Chainlink (and more specifically, the one that is expected to be used, `USDC/USD`) return an 8 decimals number. Since this `underlyingTokenPrice` value will be divided by the `billyTokenPrice` value which again is in 8 decimals, this will result in a calculation error and overinflation of the `outAmount` in the `_swap` method. Since the `SwapFacility` contract won't be holding so many tokens in its balance, the calls to `swap` will always revert, leaving the `BloomPool` contract in a stuck state - with all deposited funds in it but without an ability to continue further through its phases.

Recommendations

Do not scale the price by `1e12`. Clearly define the price feeds that will be used and if they have different decimals only then scale them to the expected decimals count.

8.2. High Findings

[H-01] The protocol does not implement slippage checks on swaps

Severity

Impact: High, as it can result in a substantial loss of value if there is big price movement

Likelihood: Medium, as slippage is never handled, but it requires specific market conditions

Description

The protocol mentions in its `README` file that the `SwapFacility` has to implement slippage checks, but it doesn't. If a swap transaction is sent to the mempool, but it takes a while until it is executed, it is possible that there was big price movement and the swap returned value is substantially lower than what it was initially expected to be, which will be a value loss for the protocol & its users.

Recommendations

Add a `minOutAmount` parameter to `SwapFacility::_swap` and check that the swap resulted in at least that many tokens, otherwise revert.

8.3. Medium Findings

[M-01] The swap mechanism does not have a deadline parameter

Severity

Impact: High, as the swap might forcefully result in a big slippage (or maximum allowed one)

Likelihood: Low, as it requires special conditions

Description

Swap mechanisms should implement a transaction deadline mechanism, due to the following attack vector:

1. Alice wants to execute a swap, sets slippage to 10% and sends a transaction to the mempool, but with a very low gas fee
2. Miners/validators see the transaction but the fe is not attractive, so the transaction is stale and pending for a long time
3. After a week (let's say) the average gas fees drop low enough for the miners/validators to execute the transaction but the price of the assets has changed drastically
4. Now the value Alice receives is much lower and possibly close to the max slippage she set.

The effects are even worse when there is no slippage as it is the current case in the protocol.

Recommendations

Add a `deadline` timestamp parameter to the `SwapFacility::_swap` method and revert the transaction if the expiry has passed.

[M-02] Missing price feed validation and usage of a deprecated method can lead to 0 price

Severity

Impact: High, as using a 0 price would mess the swap calculations

Likelihood: Low, as it requires a malfunctioning price feed

Description

The `_getTokenPrices` method in `SwapFacility` makes use of the `latestAnswer` method from Chainlink price feeds. The problem is that the NatSpec of `latestAnswer` says this:

> @dev #[deprecated] Use latestRoundData instead. This does not error if no answer has been reached, it will simply return 0. Either wait to point to an already answered Aggregator or use the recommended latestRoundData instead which includes better verification information.```

So currently it is possible that `latestAnswer` returns 0 and the code operates with zero price, leading to miscalculations in the rate of `underlyingToken` to `billyToken` which will lead to a loss of funds.

Recommendations

As pointed out in the comment, use `latestRoundData` instead to query a price feed.

[M-03] Centralization attack vectors are present

Severity

Impact: High, as it can break the protocol for users

Likelihood: Low, as it requires a malicious or a compromised owner

Description

The `owner` of `SwapFacility` can change the `pool` variable any time, meaning it can be set to `address(0)` for example, breaking the protocol's `swap` functionality. Another such issue is that the `setSpreadPrice` method does not do any input validation, meaning the `spreadPrice` can be set to a huge number that is bigger than the token prices, which will make the spread subtraction revert the `swap` transactions every time.

Recommendations

Make `setPool` callable only once and also put an upper bound of the `spreadPrice` value.

[M-04] Tokens with a fee-on-transfer mechanism will break the protocol

Severity

Impact: High, as some users will lose value

Likelihood: Low, as such tokens are not common

Description

The ERC20 logic in `BloomPool` is incompatible with tokens that have a fee-on-transfer mechanism. Such tokens for example is `PAXG`, while `USDT` has a built-in fee-on-transfer mechanism that is currently switched off. One example of this `BloomPool::depositBorrower` where the following code:

```
UNDERLYING_TOKEN.safeTransferFrom(msg.sender, address(this), amount);
```

This will work incorrectly if the token has a fee-on-transfer mechanism - the contract will cache `amount` as its expected added balance, but it will actually add `amount - fee` balance. This will result in a revert in the last person to withdraw tokens out of the contract. Same thing applies for other `transferFrom` calls that transfer tokens into the protocol, for example in `SwapFacility::_swap`.

Recommendations

You should cache the balance before a `transferFrom` to the contract and then check it after the transfer and use the difference between them as the newly added balance. This also requires a `nonReentrant` modifier, as otherwise ERC777 tokens can manipulate this. Another fix is to just document and announce you do not support tokens that can have a fee-on-transfer mechanism.

8.4. Low Findings

[L-01] Usage of `safeApprove` might be incompatible with tokens like `USDT`

Some tokens implement an approval race protection mechanism (`USDT`, `KNC`) which requires the pre-approval allowance to be either zero or `type(uint256).max`. Currently the code in `initiatePreHoldSwap` uses `safeApprove` for the `UNDERLYING_TOKEN`. If the previous allowance was not used until it was 0, then if `USDT` is the `UNDERLYING_TOKEN` the code will revert on the `safeApprove` call. It is recommended to use `forceApprove` from OpenZeppelin 4.9.1's `SafeERC20` so this case is covered properly.