BuyBurner Security Review

Pashov Audit Group

Conducted by: T1MOH, MrPotatoMagic, Dan Ogurtsov

December 14th - December 15th

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our 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 we try to find as many vulnerabilities as possible. We 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 **avierauy/buyburner** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About BuyBurner

BuyBurner contract allows an approved burner to buy BEAM tokens from Uniswap using USDC or ETH, then burn them to reduce supply and support deflation. Admins can also withdraw unused tokens or ETH.

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 - bb4e8a812e6bda4838e84d0bae1b93a6896ffca3

fixes review commit hash - 4f22ab9be35552003572733321 12ff1429d9f3a8

Scope

The following smart contracts were in scope of the audit:

- `BuyBurner`

7. Executive Summary

Over the course of the security review, T1MOH, MrPotatoMagic, Dan Ogurtsov engaged with BuyBurner to review BuyBurner. In this period of time a total of **3** issues were uncovered.

Protocol Summary

Protocol Name	BuyBurner
Repository	https://github.com/avierauy/buyburner
Date	December 14th - December 15th
Protocol Type	Token supply management

Findings Count

Severity	**Amount**
High	1
Low	2
Total Findings	**3**

Summary of Findings

ID	Title	Severity	Status
[H-01]	Manipulatable oracle and breaking slippage protection	High	Resolved
[L-01]	Add safety checks on slippagePercent in function buy()	Low	Resolved
[L-02]	block.timestamp used as swap deadline offers no protection	Low	Resolved

8. Findings

8.1. High Findings

[H-01] Manipulatable oracle and breaking slippage protection

Severity

Impact: High

Likelihood: Medium

Description

The buying of BEAM by the protocol can be affected by external parties. The expected amount of BEAM received is calculated as `UNISWAP_V2_ROUTER_02.getAmountsOut(...)` which depends on the token reserves in related pairs.

```
function _getTokenAmountPrice
      (uint256 tokenAmount, address token0, address token1)
        internal
        view
        returns (uint256)
    {
        ...

@>      uint256[] memory returnedParam = UNISWAP_V2_ROUTER_02.getAmountsOut
  (tokenAmount, path);

        return returnedParam[1];
    }
...
    function _getTokenAmountPrice3
      (uint256 tokenAmount, address token0, address token1, address token2)
        internal
        view
        returns (uint256)
    {
        ...

@>      uint256[] memory returnedParam = UNISWAP_V2_ROUTER_02.getAmountsOut
  (tokenAmount, path);

        return returnedParam[2];
    }
```

These reserves are easily manipulatable. An attacker can frontrun with a huge swap and set almost any price in pairs, thus influencing `_getAmountOutMin()`.

slippagePercent in this calculation is just a fixed number giving no protection from the manipulation described above.

As a result, it's easy to make the protocol buy almost zero BEAMs on each swap.

But in practice, for an attacker it would be more profitable to find the exact size of the sandwich attack to maximize profit (the calculation can be found here) to steal tokens on each trade.

Recommendations

Consider either using less manipulatable oracles (like TWAP from the same pairs, or even Chainlink) or consider inputting the **amount** out of BEAM expected from `buy()`, not the slippage percentage.

8.2. Low Findings

[L-01] Add safety checks on slippagePercent in function buy()

In function buy(), we should add two safety checks on slippagePercent:

1. We should ensure it is not greater than 10000.
2. We should ensure that the BUYER_ROLE cannot misuse the parameter to set slippage to the absolute maximum i.e. 10000. For this, consider allowing the DEFAULT_ADMIN_ROLE to set an allowable limit that the BUYER_ROLE cannot exceed.

```
function buy(
  uint256amountIn,
  uint256slippagePercent,
  booluseUSDC
) external onlyRole(BUYER_ROLE
```

[L-02] `block.timestamp` used as swap deadline offers no protection

`_swapETHForBEAM()` and `_swapUSDCforBEAM()` perform a swap with `deadline = block.timestamp()`. The block the transaction is eventually put into will be `block.timestamp` so this offers no protection.

Recommendation: pass deadline as the function argument.