PUMPe
Audit Report

Wed Nov 27 2024

contact@bitslab.xyz https://twitter.com/scalebit_

ScaleBit

PUMPe Audit Report

1 Executive Summary

1.1 Project Information

Description	An ERC20 token called PumpToken.
Type	Token
Auditors	ScaleBit
Timeline	Tue Nov 26 2024 - Wed Nov 27 2024
Languages	Solidity
Platform	Ethereum
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/AnyAxis-Labs/pumpe-sc
Commits	9772831cb0df7bd5b5972afb881025702c46bbae 417c2ae45d629da9b88c2397f31636a67ddaf0e2 cc542372e8ef081f15a220729a37ff89fc73002d f9c2dd9ad4630b080924f76c8b52e8a4972fcdc4

1.2 Files in Scope

The following are the SHA1 hashes of the original reviewed files.

ID	File	SHA-1 Hash
BFO	src/BancorFormula.sol	2fa9b8010b6a3724425ec001109cd4a0dc0b9312
IUV2P	src/interfaces/IUniswapV2Pair.sol	d6c9c14442c460fd79c6caf0d0a364251d09f1dd
IUV2R0	src/interfaces/IUniswapV2Router02.sol	e4289cd4a7d588ea141718c1f2f0dd47ed029718
IWETH	src/interfaces/IWETH.sol	cbea2cc469ce2ac61fdaa40a70d11e5188c1d55d
IUV2F	src/interfaces/IUniswapV2Factory.sol	aa92ec9c013fd63067249e95ea7b3976a854baca
PFA	src/PumpFactory.sol	d0476ae6a44d551bb15d6a13d9422b635b1e1630
BCU	src/BondingCurve.sol	f3ca0188fcf7875ee5ed5a533f959bb68af2ea67
PTO	src/PumpToken.sol	be38153cafc5e8ce2d833fd6cfd4fd6f7ae4907f
POW	src/Math/Power.sol	9c8341652b923ca19094bf98e5370d57f5572e8c
SMA	src/Math/SafeMath.sol	8a91544187994035e23ab45135a840657ba26fae

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	8	5	3
Informational	2	0	2
Minor	2	1	1
Medium	1	1	0
Major	3	3	0
Critical	0	0	0

1.4 ScaleBit Audit Breakdown

ScaleBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow
- Number of rounding errors
- Unchecked External Call
- Unchecked CALL Return Values
- Functionality Checks
- Reentrancy
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic issues
- Gas usage
- Fallback function usage
- tx.origin authentication
- Replay attacks
- Coding style issues

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Quang to identify any potential issues and vulnerabilities in the source code of the PUMPe smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 8 issues of varying severity, listed below.

ID	Title	Severity	Status
PTO-1	BPS Design Does Not Match Actual Code	Major	Fixed
PTO-2	Normal Calls to `list` Result In Asset Locking	Major	Fixed
PTO-3	UniswapV2 Pool Pre-Registration Error	Major	Fixed
PTO-4	Loss of Funds in Case of `pump`	Medium	Fixed
PTO-5	No Handling Fee for Microfinance	Minor	Fixed
PTO-6	If All Tokens Have Been Sold and The TotalSupply and PoolBalance Are 0, the Contract Is Stuck	Minor	Acknowledged
PTO-7	`initialize` Function Are Called Preemptively, Resulting in Subsequent Functions Not Working Properly	Informational	Acknowledged
PTO-8	Purchase Loss Caused by Curve	Informational	Acknowledged

3 Participant Process

Here are the relevant actors with their respective abilities within the PUMPe Smart Contract :

The User

- The `User` can buy `PumpToken` through `buy()` .
- The `User` can sell `PumpToken` through `sell()` .
- The `User` can `pump` their owned `PumpToken` through `pump()` .

The Admin

- The `Admin` or `User` can list eth through `list()` .

4 Findings

PTO-1 BPS Design Does Not Match Actual Code

Severity: Major

Status: Fixed

Code Location:

src/PumpToken.sol#197;

src/PumpToken.sol#198;

src/PumpToken.sol#151-156

Descriptions:

```
uint16 private constant RESERVE_RATE_BPS = 500; // 5% in basis point


   uint16 private constant FEE_RATE_BPS = 100; // 1% in basis point


   uint16 private constant USER_HOLDINGS_TO_PUMP_THRESHOLD_BPS = 5100; // 51%


function _chargeFee(
      uint256 amount
   ) internal returns (uint256 reserved, uint256 fee, uint256 remaining) {
      reserved = amount.mulDiv(RESERVE_RATE_BPS, 1e5);
      fee = amount.mulDiv(FEE_RATE_BPS, 1e5);
      remaining = amount - reserved - fee;
      totalReserve = totalReserve + reserved;


      console.log("after charge: %d, %d, %d", reserved, fee, remaining);
   }
```

The above code, you can see that the comment is written 5%, 1%, 51%, but the actual code in the division of 1e5, when the actual expression of the meaning of the respective 0.5%, 0.1%, 5.1%.

Suggestion:

Proposed replacement of 1e5 with 1e4.

Resolution:

The customer took our advice and fixed the issue.

PTO-2 Normal Calls to `list` Result In Asset Locking

Severity: Major

Status: Fixed

Code Location:

src/PumpToken.sol

Descriptions:

Calling the `list` function requires `LISTING_THRESHOLD = 366.67 ether`, and then creates the `UniswapV2Pair` using only `LISTING_LIQUIDITY = 256.67 ether`, after which the contract shuts down and can't be called any further, and at that point an asset lock is created.

```
function list() external payable onlyWhenLaunching {
    require(msg.value == LISTING_FEE, "Must pay listing fee");
    require(
        totalRaised >= LISTING_THRESHOLD,
        "Total raised must pass the listing threshold"
    );

    IUniswapV2Factory uniswapFactory = IUniswapV2Factory(
        uniswapV2Router.factory()
    );

    address WETH = uniswapV2Router.WETH();
    uniswapPair = IUniswapV2Pair(
        uniswapFactory.createPair(address(this), WETH)
    );

    uint256 tokensToList = LISTING_LIQUIDITY.mulDiv(
        totalSupply(),
        totalRaised
    );
```

```
        _mint(address(uniswapPair), tokensToList); // mint liquidity amount to the pair

        IWETH(WETH).deposit{value: LISTING_LIQUIDITY}();
        assert(IWETH(WETH).transfer(address(uniswapPair), LISTING_LIQUIDITY)); // transfer weth to the pair
        IUniswapV2Pair(uniswapPair).mint(address(uniswapFactory)); // call low level mint function on pair
        _launching = false;

        emit List(tokensToList);
    }
```

Suggestion:

Suggest adding a method to extract assets to prevent asset locking issues after `_launching = false`.

Resolution:

The customer took our advice and fixed the issue.

PTO-3 UniswapV2 Pool Pre-Registration Error

Severity: Major

Status: Fixed

Code Location:

src/PumpToken.sol#163

Descriptions:

In the `list()` function, the following tasks are mainly completed:

step1 Check whether `totalraise` reaches the set threshold value. step2: Create UniswapV2 liquidity pool. step3: Add liquidity to the pool. step4: Change the launching status.

The problem is that the attacker may create a liquidity pool in uniswapV2 in advance, which will cause step2 to report an error because the pool already exists.

```
function createPair(address tokenA, address tokenB) external returns (address pair) {
    require(tokenA != tokenB, 'UniswapV2: IDENTICAL_ADDRESSES');
    (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
    require(token0 != address(0), 'UniswapV2: ZERO_ADDRESS');
    require(getPair[token0][token1] == address(0), 'UniswapV2: PAIR_EXISTS'); // single check
is sufficient
//...
```

As a result, the contract status cannot be changed, liquidity cannot be added, and the contract cannot work properly. This may cause serious problems.

Suggestion:

It is recommended to redesign this part of the code, please consider the situation where the pool is pre-registered, and ensure that:

1. The status change of the pool is in line with expectations.
2. The pool can add liquidity.

Resolution:

The customer took our advice and fixed the issue.

PTO-4 Loss of Funds in Case of `pump`

Severity: Medium

Status: Fixed

Code Location:

src/PumpToken.sol#157

Descriptions:

```
function _buy(uint256 value, address recipient) internal returns (uint256) {
    require(value > 0, "Amount in too small");
    (, , uint256 buyAmount) = _chargeFee(value);
    console.log("actual buy amount: %d", buyAmount);
    uint256 tokensToMint = _bcBuy(buyAmount, recipient);
    totalRaised = totalRaised + buyAmount;
    emit Buy(recipient, buyAmount, tokensToMint);
    return tokensToMint;
}
```

When the above logic is executed, the `_buy` function will be called and all the `totalReserve` will be used to buy pump tokens, and a certain fee will be charged, and then the fee will be added to the `totalReserve`, but will be cleared to 0 afterward, that is to say, a loss of funds of `totalReserve*5%` is incurred.

Suggestion:

A more primitive function is used to implement this method.

Resolution:

The customer took our advice and fixed the issue.

PTO-5 No Handling Fee for Microfinance

Severity: Minor

Status: Fixed

Code Location:

src/PumpToken.sol#197-198

Descriptions:

In this case, when calculating the handling fee, for small amounts of eth, due to the loss of precision in the division, the handling fee will be zero.

```
function _chargeFee(
    uint256 amount
) internal returns (uint256 reserved, uint256 fee, uint256 remaining) {
    reserved = amount.mulDiv(RESERVE_RATE_BPS, 1e5);
    fee = amount.mulDiv(FEE_RATE_BPS, 1e5);
    remaining = amount - reserved - fee;
    totalReserve = totalReserve + reserved;

    console.log("after charge: %d, %d, %d", reserved, fee, remaining);
}
```

Suggestion:

It is recommended that a check be added so that an error can be reported if the handling fee is zero, to avoid such a situation.

Resolution:

The customer took our advice and fixed the issue.

PTO-6 If All Tokens Have Been Sold and The TotalSupply and PoolBalance Are 0, the Contract Is Stuck

Severity: Minor

Status: Acknowledged

Code Location:

src/PumpToken.sol

Descriptions:

This is an edge case - in practice if everyone sells their tokens its probably OK that the contract is dead. However it would be nice to have a work around for this. The `creator` selling all of his tokens and others selling all of their tokens can cause such scenario.

Suggestion:

If `totalSupply` and `poolBalance` are 0 use a function similar to `calculatePurchaceReturn` to compute the number of tokens you would need to sell if you reduce the `poolBalance` from `intialPoolBalance` (`poolBalance` the contract was initialized with) to `msg.value` .

PTO-7 `initialize` Function Are Called Preemptively, Resulting in Subsequent Functions Not Working Properly

Severity: Informational

Status: Acknowledged

Code Location:

src/PumpToken.sol#78

Descriptions:

An attacker can preempt initialization before the administrator's normal initialization causing initialization data such as owner to be modified to unintended values, leading to a number of problems.

Suggestion:

It is recommended that some means be used to prevent grab-and-run behavior from occurring.

PTO-8 Purchase Loss Caused by Curve

Severity: Informational

Status: Acknowledged

Code Location:

src/PumpToken.sol

Descriptions:

```
* Return = _supply * ((1 + _depositAmount / _connectorBalance) ^ (_connectorWeight / 1000000) - 1)
```

According to the purchase formula, we found that in the same state of the pool, it costs more to spend `a` ETH to buy tokens at one time than to spend `b` and `c` ETH to buy tokens twice.(`a=b+c`).This is because the purchase function is not linear. This problem may cause users to have a slight loss when purchasing tokens with a large amount of ETH.

Suggestion:

It is recommended to make sure you are aware of this and it fits your design.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.