BitGenie
Audit Report

Tue Apr 23 2024

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

ScaleBit

BitGenie Audit Report

1 Executive Summary

1.1 Project Information

Description	BitGenie is the first DEX on BTC L2.
Type	Dex
Auditors	ScaleBit
Timeline	Mon Apr 22 2024 - Tue Apr 23 2024
Languages	Solidity
Platform	Merlin Chain
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/Async-Finance/BitGenie https://github.com/Async-Finance/uni-v2-core https://github.com/Async-Finance/uni-v2-periphery
Commits	https://github.com/Async-Finance/BitGenie: 91c8809c0765bc998bed3f0e5d62be99e609e282 https://github.com/Async-Finance/uni-v2-core: b5bc6c710e58ffb5957746109b6f320667207c8d https://github.com/Async-Finance/uni-v2-periphery: 1ae09cbdfb2d049ba01dbc13ebad9ea7b0cacaee

1.2 Files in Scope

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

ID	File	SHA-1 Hash
UV2R0	contracts/UniswapV2Router02.sol	af24a28533767d1473c734e649ee24682fd99cd9
UQ1X1	contracts/libraries/utils/math/UQ112x112.sol	34c562209526fad66fed9e07ca83bafe69a2f066
CON	contracts/libraries/utils/Context.sol	d25a6cc2f38ad7ab5f4d4ab607050a3f4ed979b0
STR	contracts/libraries/utils/Strings.sol	e7e47797baf283b697111d67b44e7cf1cdb3dd22
ERC2P	contracts/libraries/token/ERC20Permit.sol	45180ea231f22761c5b604b75b2988d8971ed3fb
ERC2	contracts/libraries/token/ERC20.sol	661c6b076d0d172fd07f777396895a7cf994ee83
ERC2WP	contracts/libraries/token/ERC20WithPermit.sol	6a2c73fcd439e34c3d36440680690fb217ae8804
MHE	contracts/libraries/token/helper/MetadataHelper.sol	dacbe5d7cf1d7680767d016f96f2cd9da57dc48c
ECDSA	contracts/libraries/cryptography/ECDSA.sol	99afc2ace3d47da3acb1e99a6dc14c872a7241ed
ERC2TT	contracts/test/ERC20TestToken.sol	30b8d5905e5b9c82aef89bff48ffa89138246940
ERC2WPTT	contracts/test/ERC20WithPermTestToken.sol	190d96e1115616c796faa2494ce2f5ea8160d2b0

IUV2ERC2	contracts/interfaces/IUniswapV2ERC20.sol	79c6d731a2561b08eeb9a0f2a6fa6ac078c60abe
IUV2P	contracts/interfaces/IUniswapV2Pair.sol	a2aefced50af2141ebdb9f531392cffc83074730
IUV2F	contracts/interfaces/IUniswapV2Factory.sol	b279046e1e6a97512b099252eec99e2acbe53c19
IUV2C	contracts/interfaces/IUniswapV2Callee.sol	26f0843e1565071a219fadd9296be56c83e1c435
IERC2M	contracts/interfaces/ERC20/IERC20Metadata.sol	f08bdaa51dcd45b4c38b1c6c94baac7b9099171c
IERC2P	contracts/interfaces/ERC20/IERC20Permit.sol	e9fcefdc75429c3d206abcd6b1f66041ec264c62
IERC2	contracts/interfaces/ERC20/IERC20.sol	902b4299056a1b2965973b9bab490d1834b7f17e
IERC21	contracts/interfaces/IERC20.sol	3afe25d441f3b6efebc220ca23c60d4af49780d8
UV2F	contracts/UniswapV2Factory.sol	2d2346788dff3f122dda206717b740f4aae79759
UV2P	contracts/UniswapV2Pair.sol	84ddae671f9fe9630c514b090046775019ec3f94
STA	contracts/Staking.sol	a20add7a6a3e2b5e5cc141be5500940efd6e4f29

1.3 Issue Statistic

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

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"** and **"Formal Verification"** 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 BitGenie to identify any potential issues and vulnerabilities in the source code of the BitGenie 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 2 issues of varying severity, listed below.

ID	Title	Severity	Status
STA-1	Centralized Risk	Critical	Acknowledged
UV2-1	`UniswapV2Pair` Can Add A Lock Modifier	Informational	Acknowledged

3 Participant Process

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

Admin

- `pause()` : Admin can pause the protocol through this function to temporarily stop all staking and redeeming activities.
- `unpause()` : Admin can resume the protocol activities that were halted by the `pause` function.
- `setHardCap(uint256 _hardCap)` : Admin sets the maximum amount of tokens that can be staked in the pool.
- `setRewardPerTokenPerSecond()` : Admin calculates and sets the rate at which rewards are generated per token per second based on the total balance and duration.
- `setWithdrawTime(uint256 _withdrawTime)` : Admin sets the timestamp after which withdrawal of funds is allowed.
- `setVersion(uint256 _version)` : Admin updates the version of the contract or protocol.
- `withdraw(address token)` : Admin can withdraw pool funds after the set withdrawal time, through this function.

User

- **Stake Operations**:
  - `stake(uint256 amount)` : Users can stake a specified amount of tokens as long as it doesn't exceed the hard cap and the staking period hasn't ended.
  - `unstake(uint256 amount)` : Users can unstake a specified amount of tokens, not exceeding their current staked balance.
  - `redeem()` : Users can redeem their accumulated rewards after the staking period has ended.
- **Liquidity Operations**:
  - `addLiquidity()` : Users can add liquidity to a pair of tokens (tokenA and tokenB). This function requires users to provide both tokens in desired amounts, and it returns the liquidity tokens to the specified recipient address.

- `addLiquidityETH()` : Users can add liquidity using ETH and another token. This function requires the ETH amount to be sent along with the transaction and returns liquidity tokens.
- `removeLiquidity()` : Users can remove their liquidity from a token pair, receiving back both tokens in the pool.
- `removeLiquidityETH()` : Users can remove liquidity that includes ETH, receiving back one token and ETH.
- `removeLiquidityWithPermit()` : Allows for removing liquidity without a separate approval transaction by using a permit (EIP-2612).
- `removeLiquidityETHWithPermit()` : Similar to `removeLiquidityWithPermit` , but for pools involving ETH.
- `removeLiquidityETHSupportingFeeOnTransferTokens()` : Allows removing liquidity involving ETH with tokens that implement transfer fees.
- `removeLiquidityETHWithPermitSupportingFeeOnTransferTokens()` : Combines permit functionality with support for fee-on-transfer tokens in liquidity removal involving ETH.

- **Swap Operations**:
  - `swapExactTokensForTokens()` : Allows users to swap an exact amount of input tokens for a minimum possible output tokens along a specified path of token pairs.
  - `swapTokensForExactTokens()` : Users can swap tokens to receive an exact amount of output tokens, specifying a maximum amount of input tokens that can be spent.
  - `swapExactETHForTokens()` : Swaps an exact amount of ETH for tokens, specifying a path that must start with WETH.
  - `swapTokensForExactETH()` : Users can swap tokens to receive an exact amount of ETH, specifying a maximum input amount and a path ending in WETH.
  - `swapExactTokensForETH()` : Swaps tokens for an exact amount of ETH, providing a minimum amount of ETH to receive.
  - `swapETHForExactTokens()` : Users can send ETH to receive an exact amount of tokens specified by a path beginning with WETH.

- `swapExactTokensForTokensSupportingFeeOnTransferTokens()` : Supports swaps involving tokens with transfer fees.
- `swapExactETHForTokensSupportingFeeOnTransferTokens()` : Allows swapping ETH for tokens that have transfer fees, with a path starting with WETH.
- `swapExactTokensForETHSupportingFeeOnTransferTokens()` : Swaps tokens with transfer fees for an exact amount of ETH.

4 Findings

STA-1 Centralized Risk

Severity: Critical

Status: Acknowledged

Code Location:

contracts/Staking.sol#123-126;

contracts/Staking.sol#131-136

Descriptions:

The smart contract includes a function `setWithdrawTime` that allows the contract owner to arbitrarily set the `withdrawTime`. This function, combined with the `withdraw` function that enables the owner to pull all funds post the designated `withdrawTime`, creates a high-risk scenario where the owner can potentially withdraw all the funds in the pool at any time.

```
  function setWithdrawTime(uint256 _withdrawTime) external onlyOwner {
    withdrawTime = _withdrawTime;
  }
...
  function withdraw(address token) external onlyOwner {
    require(block.timestamp >= withdrawTime, "Cannot withdraw");
    uint256 amount = IERC20(token).balanceOf(address(this));
    IERC20(token).safeTransfer(_msgSender(), amount);
    emit WithdrawEvent(_msgSender(), amount);
  }
```

Suggestion:

Implement a **Time Lock** mechanism for the `setWithdrawTime` function to mitigate the risks associated with this centralized control feature. A time lock would require any changes to `withdrawTime` to be proposed well in advance of being enacted, providing transparency and a buffer period during which stakeholders can react.

UV2-1 `UniswapV2Pair` Can Add A Lock Modifier

Severity: Informational

Status: Acknowledged

Code Location:

contracts/UniswapV2Pair.sol

Descriptions:

In the `UniswapV2Pair` smart contract, a lot of functions will check and set `unlocked` to protect the execution.

However, this method creates a lot of duplicate code where a modifier can solve this problem.

Suggestion:

It is suggested to use the modifier instead of hardcode it.

```
modifier lock() {
      require(unlocked == 1, 'UniswapV2: LOCKED');
      unlocked = 0;
      _;
      unlocked = 1;
  }
```

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.