QuillAudits

Audit Report
July, 2024

For

BURVE

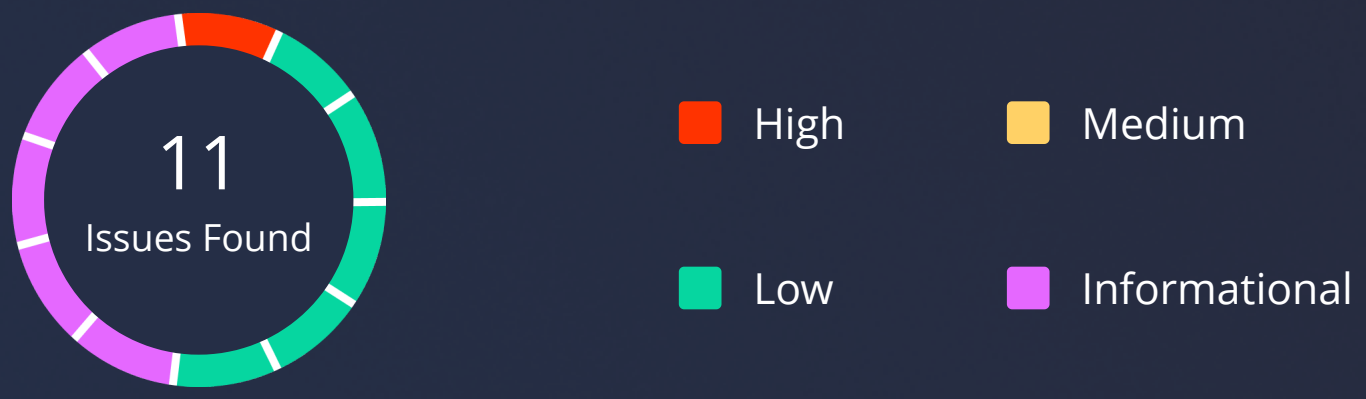
Table of Content

Table of Content

Executive Summary

Project Name	Burve Protocol
Overview	Burve Protocol is an innovative automatic market maker (AMM) developed by Burve Labs, focusing on token fair launches, unilateral liquidity management, and secure lending solutions. It leverages bonding curve theory within blockchain contexts to enhance decentralized finance (DeFi) applications.
Timeline	12th July 2024 to 26th July 2024
Updated Code Received	30th July 2024
Second Review	30th July 2024 - 1st August 2024
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	The scope of this audit was to analyse the Burve Contracts for quality, security, and correctness.
Source Code	https://github.com/BurveProtocol/burve-contracts
Contracts In-Scope	**Branch:** Main **Contracts:** src/abstract/BurveBase.sol src/BurveTokenFactory.sol
Branch	Main
Fixed In	https://github.com/BurveProtocol/burve-contracts/commit/3da806fa3616affb4214d292137778397f39c1ca

Number of Security Issues per Severity



	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	3	4
Partially Resolved Issues	0	0	0	0
Resolved Issues	1	0	2	1

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC721 API violation
- Malicious libraries
- ERC20 API violation
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC's standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Hardhat, Foundry.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

High Severity Issues

1. Existing token hooks are replaceable while adding new

Path

BurveTokenFactory.sol

Function

addHookForToken

Description

The addHookForToken function adds a hook in the token factory against the token address, and the hook registers the token and data. addHooksForToken function adds a list of hooks in the token factory against the token address, and hooks register the token and data.

The following scenario will replace the existing list of hooks stored in the token factory:

- Deploy hook1 and use addHookForToken function to update the tokenHooks mapping.
- Deploy hook2 and hook3. Use addHooksForToken function to update the tokenHooks mapping.

Below is the same scenario presented as a Foundry test

```
function test_addHookForTokens() public {
    deployNewERC20(100, 100, 1000, 0.001 ether);
    LaunchTimeHook hook = new LaunchTimeHook(address(factory));
    deployNewHook(address(hook)); // hook1
    HardcapHook hook2 = new HardcapHook(address(factory));
    deployNewHook(address(hook2)); // hook2
    SBTHook hook3 = new SBTHook(address(factory));
    deployNewHook(address(hook3)); // hook3
    // add hook1
    vm.prank(projectAdmin);
    factory.addHookForToken(address(currentToken), address(hook),
abi.encode(block.timestamp + 1 days));
    address[] memory result = factory.getTokenHooks(address(currentToken));
    console2.log(result[0]);
    assert(result.length == 1); // array length 1
    address[] memory hooks = new address[](2);
    hooks[0] = address(hook2);
```

```
    hooks[1] = address(hook3);
    bytes[] memory datas = new bytes[](2);
    datas[0] = abi.encode(block.timestamp + 1 days);
    datas[1] = abi.encode(block.timestamp + 1 days);
    vm.prank(projectAdmin);
    // add hook2 and hook3
    factory.addHooksForToken(address(currentToken), hooks, datas);
    result = factory.getTokenHooks(address(currentToken));
    console2.log(result[0]);
    console2.log(result[1]);
    assert(result.length != 3); // array is not 3
}
```

This happens because in addHooksForToken function instead of appending the input hooks to existing hooks, input array of hooks is set to the tokenHooks mapping.

```
function addHooksForToken(address token, address[] calldata hooks, bytes[] calldata
datas) external override onlyProjectAdmin(token) {
    require(hooks.length == datas.length);
    tokenHooks[token] = hooks; // @audit should append input hooks to existing hooks
    for (uint256 I = 0; I < hooks.length; I++) {
        require(whitelistHooks[hooks[I]], “not whitelist”);
        IHook(hooks[I]).registerHook(token, datas[I]);
    }
}
```

Recommendation

Consider to loop over the input hook addresses and append them to the tokenHooks mapping.

Status

Resolved

Burve protocol team changed to addHooksForToken function to loop over the input hooks and append them to the tokenHooks mapping.

Low Severity Issues

2. Incorrect comparison operator while checking

Path

BurveTokenFactory.sol

Function

upgradeTokenImplement

Description

In the upgradeTokenImplement function before upgrading there are two checks performed

- upgradeTimelock is not zero
- upgradeTimelock is less than or equal to the current block.timestamp

```
/**
 * @notice when the upgrade requested, admin can upgrade the implement of token after 2 days
 * @param proxyAddress the proxy address of token
 */
function upgradeTokenImplement(address proxyAddress) external payable override onlyRole(PLATFORM_ADMIN_ROLE) {
    // @audit should allow upgrade only after 2 days
    require(upgradeTimelock[proxyAddress] != 0 && upgradeTimelock[proxyAddress] <= block.timestamp, “Upgrade Failed: timelock”);
    ...
}
```

Second check is performed to verify it has been 2 days since the upgrade request. Less than or equal to operator checks if it is been 2 days or it is 2nd day.

Recommendation

Consider changing the comparison operator in the require statement to < instead of <=

Status

Resolved

The burve protocol team removed the upgradeTokenImplement function and changed the logic of upgrading the token implementation.

3. Not emitting LogHookRegistered event

Path

BurveTokenFactory.sol

Function

addHookForToken

Description

While adding a hook in the factory contract LogHookRegistered event is emitted. addHookForToken function emits this event but not the addHooksForToken function.

```
function addHooksForToken(address token, address[] calldata hooks, bytes[] calldata
datas) external override onlyProjectAdmin(token) {
    require(hooks.length == datas.length);
    tokenHooks[token] = hooks;
    for (uint256 I = 0; I < hooks.length; I++) {
        require(whitelistHooks[hooks[I]], “not whitelist”);
        IHook(hooks[I]).registerHook(token, datas[I]);
        // @audit missing LogHookRegistered event emission
    }
}
```

Recommendation

Consider to add the event emission in the above function.

Status

Resolved

addHooksForToken function emits the event.

4. Use _grantRole instead of _setupRole

Path

BurveBase.sol

Description

Since Openzeppelin 4.4.0 version, _setupRole as deprecated in favor of _grantRole.

initialise in BurveBase contract uses _setupRole function to provide FACTORY_ROLE role to token factory address and PROJECT_ADMIN_ROLE role to token's project admin address.

_setupRole(FACTORY_ROLE, factory); // @audit use _grantRole instead of _setupRole
_setupRole(PROJECT_ADMIN_ROLE, token.projectAdmin); // @audit use _grantRole instead of _setupRole

Using deprecated is not a best practice and it might cause issues.

Recommendation

Consider using _grantRole in the constructor to grant role to accounts.

Status

Acknowledged

5. safeApprove deprecated

Path

BurveTokenFactory.sol

Description

While deploying the token, the Burve token factory transfers the raising token from the msg.sender to the factory itself. Then approves the same amount to created token using safeApprove.

The OpenZeppelin SafeERC20 safeApprove function has been deprecated since the 3.1.0 version, as seen in the comments of the OpenZeppelin code. Using this deprecated function can lead to unintended reverts and potentially the locking of tokens. A deeper discussion on the deprecation of this function is in the OZ issue.

Recommendation

Consider to replace safeApprove with safeIncreaseAllowance function.

Status

Acknowledged

6. Check newProjectTreasury is the same as the old one

Path

BurveBase.sol

Description

_setProjectTreasury internal function changes the _projectTreasury address to a new one. It verifies whether the input newProjectTreasury is not a zero address. But it is verifying the existing address and new address are not the same.

```
function _setProjectTreasury(address newProjectTreasury) private {
    // @audit not verifying new and existing project treasury are not same
    require(newProjectTreasury != address(0), “Invalid Address”);
    _projectTreasury = newProjectTreasury;
    emit LogProjectTreasuryChanged(newProjectTreasury);
}
```

If project admin sets new project treasury address same as existing this will consume gas which is unnecessary and also it emits LogProjectTreasuryChanged event which might cause issue other parts of the protocol

Recommendation

Consider to add verification to check new address and existing address are not same while setting new project treasury.

Status

Acknowledged

Informational Issues

7. Use the latest version of the openzeppelin contracts

Path

BurveBase.sol, BurveTokenFactory.sol

Function

addHookForToken

Description

An old version of OpenZeppelin Contracts is being used in the contracts. Since version 5.0.0 has been recently released with improvements such as gas optimizations and removed vulnerabilities

Recommendation

Consider to upgrade the OpenZeppelin library contracts

Status

Acknowledged

8. Use Style Guide to structure the contracts

Path

BurveTokenFactory.sol

Description

Solidity official style guide explains about standard order of layout. Also, function order can be changed based on this. The structure of referred contracts is difficult to read.

Recommendation

Consider to use order of layout documentation to change the structure of the contracts.

Status

Acknowledged

9. Missing error string messages in require

Path

BurveTokenFactory.sol

Function

deployTokenWithHooks, addHooksForToken

Description

BurveTokenFactory contract has many require statements but few are missing error string messages. Including specific, informative error messages in require and revert statements to improve overall code clarity and facilitate troubleshooting whenever a requirement is not satisfied.

```
function deployTokenWithHooks(TokenInfo calldata token, uint256 mintfirstAmount,
address[] calldata hooks, bytes[] calldata datas) public payable returns (address) {
    ...
    require(hooks.length == datas.length); // @audit add error string
    ..
}

function addHooksForToken(address token, address[] calldata hooks, bytes[] calldata datas)
external override onlyProjectAdmin(token) {
    require(hooks.length == datas.length); // @audit add error string
    ...
}
```

Recommendation

Consider to add the error string messages in above require statements.

Status

Acknowledged

10. No event emitted while a hook is removed

Path

BurveTokenFactory.sol

Description

In the factory token project admin can remove a token hook or set of token hooks, this basically updates the contract's state variable tokenHooks mapping. It is a best practice to emit an event whenever a state change happens. Also, this helps front-end or subgraphs if any are used.

Recommendation

Consider to emit an event when a hook is removed.

Status

Resolved

removeHookForToken and removeAllHookForToken functions are removed.

11. Wrong spelling

Path

BurveTokenFactory.sol

Function

BurveBase.sol

Description

In the mint function's require statement, error message is having recieved which is the incorrect spelling for received.

```
function mint(address to, uint payAmount, uint minReceive) public payable virtual
nonReentrant {
    ...
    require(tokenAmount >= minReceive, "Mint: mint amount less than minimal expect
recieved"); //@audit spelling wrong
    ...
}
```

Recommendation

Consider to correct the spelling of recieved to received.

Status

Acknowledged

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the Burve Protocol codebase. We performed our audit according to the procedure described above.

Some issues of High, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Burve Protocol smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Burve Protocol smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the Burve Protocol to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

1000+
Audits Completed

$30B
Secured

1M+
Lines of Code Audited

Follow Our Journey

Audit Report
July, 2024

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com