QuillAudits

Audit Report
December, 2023

For

GIE

Table of Content

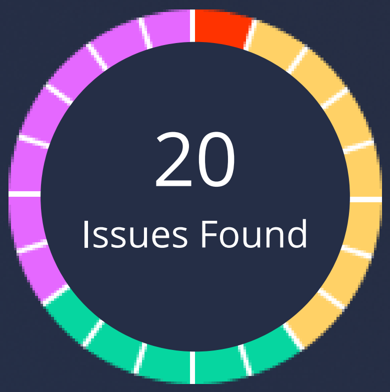
Table of Content

Table of Content

Executive Summary

Project Name	GIE Crypto
Project URL	*https://www.giecrypto.com/*
Overview	It is custom router contract. the swap functions take fees for trades/swaps and some functions takes donations. There's one owner only swap function which allows owner to trade without fees.
Audit Scope	*https://github.com/GIE-Crypto-Exchange/contract-solidity-gie/tree/4b241bfa99e297dd466790caa522a6997f57ba60*
Contracts in Scope	TokenSwap, GIE Token
Commit Hash	*4b241bfa99e297dd466790caa522a6997f57ba60*
Language	Solidity
Blockchain	Polygon
Method	Manual Testing, Automated Tests, Functional Testing
Review 1	3rd October 2023 - 27th October 2023
Updated Code Received	4th December 2023
Review 2	7th December 2023 - 11th December 2023
Fixed In	*https://github.com/GIE-Crypto-Exchange/contract-solidity-gie/pull/15*

Number of Security Issues per Severity



🟥 High 🟨 Medium

🟩 Low 🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	1	1	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	6	5	7

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
- ✓ ERC20 API violation
- ✓ Malicious libraries
- ✓ 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-20 token 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

Remix IDE, Truffle, Solhint, Mythril, Slither, Solidity Statistic Analysis.

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.

A. Contract - TokensSwap

High Severity Issues

A.1 Use of flashloan lending to get more discount

Description

calculateDiscountPercent() takes the user's current balance into account for returning discountPercent.

But here the user(s) can use flashloan/lending to borrow tokens to show the increased amount of balance and get more discount.

Example

1. Malicious user can write a smart contract which will take flashloan for GIEToken.
2. And will use swap function to swap tokens. While calculating the discount calculateDiscountPercent will call balanceOf() to get the balance of the GIE token for that account and it will get an increased amount which is actually a flashloaned amount.
3. In this way users can get more discount based on the flashlaoned amount.

Remediation

Use different implementation e.g **ERC20Snapshot** for GIE token where balanceOfAt can be used to check balance account later by TokensSwap while discount calculation. the snapshot id can be selected randomly and can be set in TokensSwap while deploying which can be used everytime for balanceOfAt().

Status

Acknowledged

Medium Severity Issues

A.2 Fees can be manipulated maliciously

Description

1. Sandwiching own transaction when swap function takes path and feePath:
   Attacker can Sandwich the swap transaction in this way the getAmountsOut() call for feePath on L1135 in _swapPreCheck() will return less amount than what it should return for non manipulated pools. It would be profitable to the attacker if the attacker is swapping a big amount and the fee is big and the profit made by saving fees can exceed the cost of sandwiching his own swap transaction where he will manipulate pairs in feePath so that getAmountsOut() will return less amount.
2. Entering different path in the way where feePath[feePath.length-1] would be weth but middle token addresses would be the token address which would give really less weth amount back. in short the path for pair where the weth would be costly as compared to other token can decrease the fee.

Remediation

The solution should be implemented where the feePath would be the same path that the user is using to exchange the token instead of having an extra feePath variable. Having the same path array would decrease the chance of manipulation by the user as the user would be expecting output tokens for the entered path variable so the chances of manipulation would be less because manipulating pools will also affect the output tokens they would receive.

It should be noted that the user can still manipulate pools before the swap transaction even after using the path as the feePath for _swapPreCheck() but the possibility of profit decreases as mentioned above.

Status

Resolved

A.3 swapExactETHToUSDCForConversion() doesn't take amountOutMin

Description

swapExactETHToUSDCForConversion() doesn’t take amountOutMin which is normally used to check the minimum amount of output tokens that must be received. So it can happen that the swapExactETHToUSDCForConversion() call transaction can get frontran intentionally or unintentionally and the user will receive a very less amount of tokens than what they should have received.

Remediation

Use the amountOutMin parameter so that it can be checked in a similar way the swapExactTokensForTokens() checks it on L634 to check that the slippage is in certain limit.

Status

Resolved

A.4 Dust/extra matic amount should be sent back to the user

Description

swapExactETHToUSDCForDonation() and swapExactETHToUSDCForConversion() take native tokens to get USDC for exchange. If the user sends more than required value of matic tokens then the extra matic tokens should be sent back to the user. It can be done by checking if msg.value is greater than amount[0]/amountIn then the extra can be transferred back to the user.

Remediation

Consider sending dust/extra native token amount back to the user as it is getting send in swapETHForExactTokens().

Status

Resolved

A.5 Fee can be decreased close to zero

Description

In swapETHForExactTokens(), swapExactETHForTokens() malicious user can pass a very small amountIn so that the calculated app fee would be zero or close to zero.

Passing a different/very low amountIn won't affect other logic as amountOut getting passed in getAmountsIn() is different than amountIn argument.

Remediation

Swap _swapPreCheck() call and getAmountsOut() lines in swapETHForExactTokens() function and use amounts[0] as amountIn param for _swapPreCheck().

For swapExactETHForTokens on L834 use amountIn as amountIn param of getAmountsOut().In this way even if user enters less amountIn he would be getting less amount out because amountIn will get used in getAmountsOut().

Status

Resolved

A.6 Incorrect amountIn is getting passed

Description

In swapTokensForExactTokens() and swapTokensForExactETH() amountIn param of _swapPreCheck() should not be passed as amountInMax, because amountInMax is the max amount that user would be willing to give in and amounts[0] (amounts is output of getAmountsIn) would be the amount that user needs to actually give in. So here amountInMax can be more than amount[0] (amount needed for getting amountOut). In this way user will use more than needed tokens.

Remediation

Instead of passing amountIn as amountInMax in _swapPreCheck, amounts[0] should be passed in these functions: swapTokensForExactTokens and swapTokensForExactETH.

Status

Resolved

A.7 Centralization of control

Description

The contract contains many functions which are owner only and can be used to change some important state e.g setDonationFees(), setGieAppFees(), updateDiscount(). These functions can directly affect the functionality for normal users.

Malicious owner can use these functions maliciously.

Remediation

Use a multisig wallet for the owner address so in case of compromise the risk can be mitigated.

Status

Acknowledged

Low Severity Issues

A.8: Ownership Transfer should be a Two-way Process

Description

Owner is responsible for critical functionalities in the contract, it is important to stress the need for the use of a two-way process on the transfer of ownership. When the current owner invokes the transferOwnership function, passing the parameter of the new address, this checks the new address it is not an address zero, hence would revert. But the issue arises when the address passed was a wrong address, this would not be redeemable anymore.

Remediation

Use the Openzeppeling **Ownable2Step** to remedy the issue of instantaneous transfer to the wrong address. This way, the assigned address would claim ownership first, before the completion of ownership transfer.

Status

Resolved

A.9 Add require check

Description

It should be checked that discountPercent should be always <= 100 as there would be subtraction of this amount from 100 in calculateFeesForTransaction() on L1298 while calculating gieAppFee.

Remediation

Add require check in updateDiscount() to check that discountPercent is in limit.

Status

Resolved

A.10 Lack of support for fee on transfer tokens

Description

TokensSwap contract doesn't support tokens with fees on transfer. there are no supportingFeeOnTransfer functions implemented such as https://github.com/Uniswap/v2-periphery/blob/master/contracts/UniswapV2Router02.sol#L339-L400

Remediation

Verify that support for fee on transfer tokens is not required.

GIE Team's Comment

As of now this functionality is not needed.

Status

Resolved

A.11 Unnecessary sending of the dust value

Description

In swapExactETHForTokens() the dust/extra value is getting sent but while passing amountIn to getAmountsOut() it is passing (msg.value - swapUtils.appFeeInEther) and while sending dust value it is checking that (msg.value > (swapUtils.appFeeInEther + amounts[0])) on L841 so here lets say msg.value was 10e18 (including appFeeInEther=2e18) and it passed (msg.value - swapUtils.appFeeInEther) that is 8e18. So while sending dust matic it checks 10e18 is greater than (swapUtils.appFeeInEther + amounts[0]) which would be 10e18 (i.e the same number).

Remediation

Consider verifying the mentioned scenario and remove the code block for sending extra matic value back for mentioned function.

Status

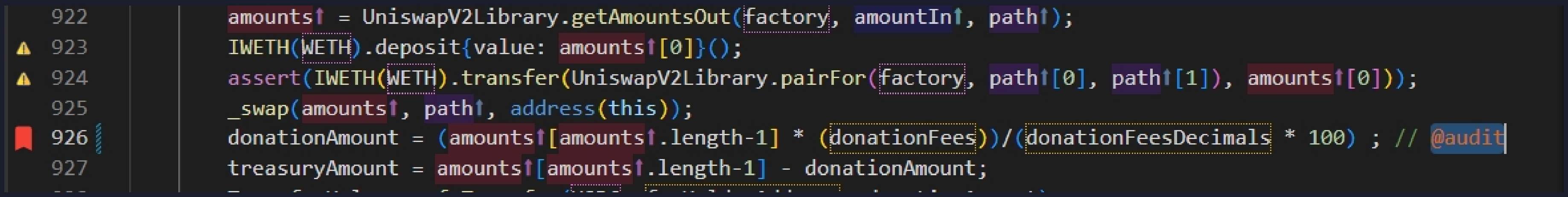
Resolved

A.12 underflow ErrorMessage check

Line

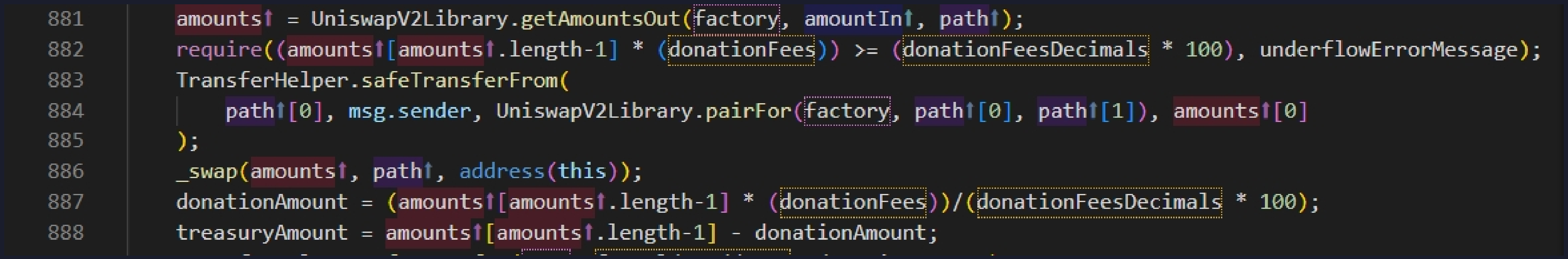
911

Function - swapExactETHToUSDCForDonation



```
        amounts = UniswapV2Library.getAmountsOut(factory, amountIn, path);
        IWETH(WETH).deposit{value: amounts[0]}();
        assert(IWETH(WETH).transfer(UniswapV2Library.pairFor(factory, path[0], path[1]), amounts[0]));
        _swap(amounts, path, address(this));
        donationAmount = (amounts[amounts.length-1] * (donationFees))/(donationFeesDecimals * 100) ; // @audit
        treasuryAmount = amounts[amounts.length-1] - donationAmount;
```

Description



```
    amounts = UniswapV2Library.getAmountsOut(factory, amountIn, path);
    require((amounts[amounts.length-1] * (donationFees)) >= (donationFeesDecimals * 100), underflowErrorMessage);
    TransferHelper.safeTransferFrom(
        path[0], msg.sender, UniswapV2Library.pairFor(factory, path[0], path[1]), amounts[0]
    );
    _swap(amounts, path, address(this));
    donationAmount = (amounts[amounts.length-1] * (donationFees))/(donationFeesDecimals * 100);
    treasuryAmount = amounts[amounts.length-1] - donationAmount;
```

There is a require check on the expected numerators of the donationAmount calculation before making the calculation. However this check was not done for the donationAmount calculation in swapExactETHToUSDCForDonation().

Remediation

Add the check before calculating donationAmount, as correctly done in swapExactTokensToUSDCForDonation().

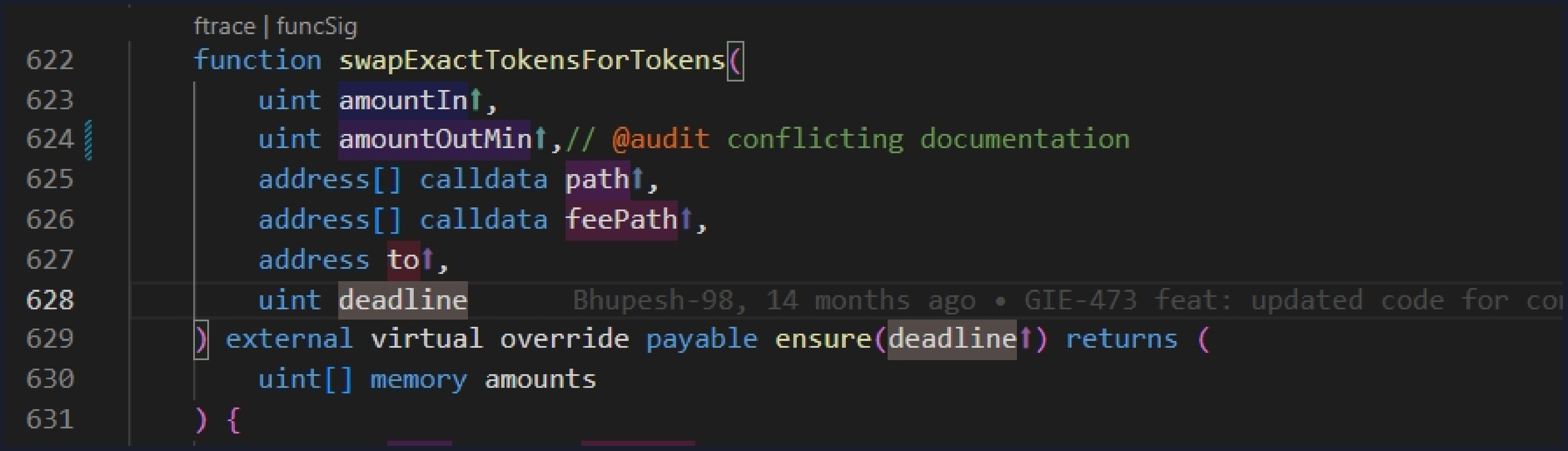
Status

Resolved

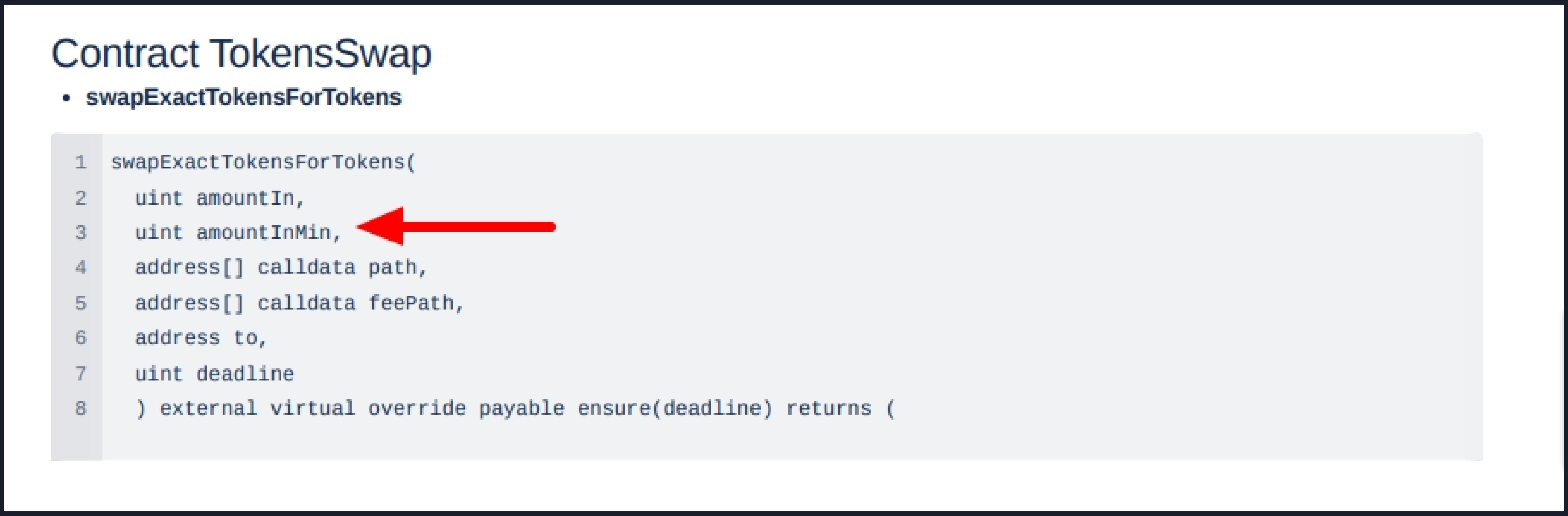
Informational Issues

A.13 Conflicting Code Documentation

Functions - swapExactTokensForTokens, swapETHForExactTokens



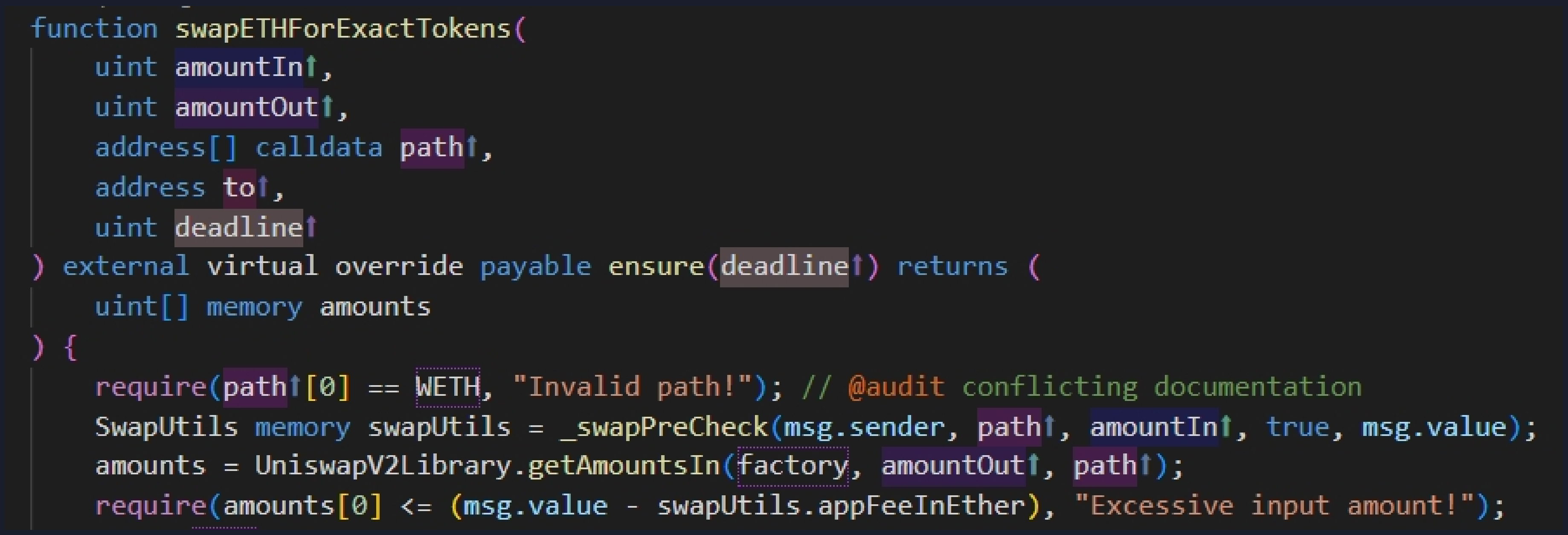
```
ftrace | funcSig
function swapExactTokensForTokens(
    uint amountIn,
    uint amountOutMin,// @audit conflicting documentation
    address[] calldata path,
    address[] calldata feePath,
    address to,
    uint deadline        Bhupesh-98, 14 months ago • GIE-473 feat: updated code for co
) external virtual override payable ensure(deadline) returns (
    uint[] memory amounts
) {
```



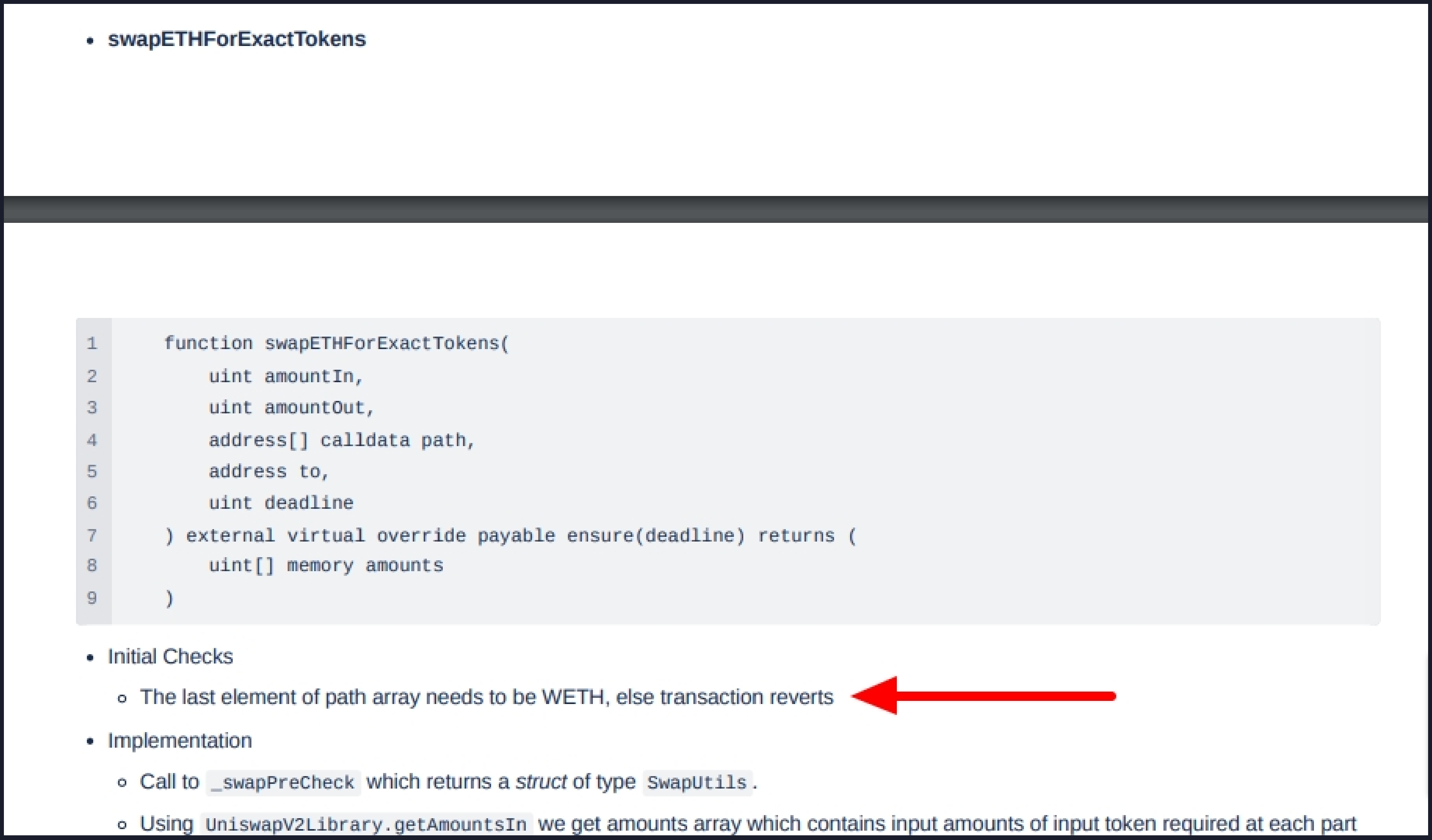
Contract TokensSwap

- **swapExactTokensForTokens**

```
swapExactTokensForTokens(
  uint amountIn,
  uint amountInMin,
  address[] calldata path,
  address[] calldata feePath,
  address to,
  uint deadline
  ) external virtual override payable ensure(deadline) returns (
```



```
function swapETHForExactTokens(
    uint amountIn,
    uint amountOut,
    address[] calldata path,
    address to,
    uint deadline
) external virtual override payable ensure(deadline) returns (
    uint[] memory amounts
) {
    require(path[0] == WETH, "Invalid path!"); // @audit conflicting documentation
    SwapUtils memory swapUtils = _swapPreCheck(msg.sender, path, amountIn, true, msg.value);
    amounts = UniswapV2Library.getAmountsIn(factory, amountOut, path);
    require(amounts[0] <= (msg.value - swapUtils.appFeeInEther), "Excessive input amount!");
```



- **swapETHForExactTokens**

```
function swapETHForExactTokens(
    uint amountIn,
    uint amountOut,
    address[] calldata path,
    address to,
    uint deadline
) external virtual override payable ensure(deadline) returns (
    uint[] memory amounts
)
```

- Initial Checks
  - The last element of path array needs to be WETH, else transaction reverts
- Implementation
  - Call to `_swapPreCheck` which returns a *struct* of type `SwapUtils`.
  - Using `UniswapV2Library.getAmountsIn` we get amounts array which contains input amounts of input token required at each part

Description

There are some conflicting code documentation to the current implementation.

Remediation

Ensure the code documentation is consistent with code implementation.

Status

Resolved

A.14 Redundant function

Description

swapExactTokensToUSDCForDonation() is redundant because there's other function which allows to swap exact token amount for tokens e.g swapExactTokensForTokens() so here the path[last element] can be a UDSC which fulfills the requirement of swapping USDC.

For swapExactETHToUSDCForDonation(), swapExactETHForTokens() is already present which can swap exact eth for tokens.

Remediation

Verify that its intentional to add these extra functions with different fee type (i.e donationFees).

GIE Team’s Comment

This has specific use when users want to donate then they will use this function. These are not redundant because addresses, and fees is different from other function.

Status

Resolved

A.15 User can choose one function to reduce fees over other

Description

Users have option to choose swapExactTokensToUSDCForDonation() over swapExactTokensForTokens() in case the fees for swapExactTokensForTokens() are more than swapExactTokensToUSDCForDonation() and vice versa.

Remediation

Confirm that the team is aware about this type scenarios.

GIE Team’s Comment

With the swapExactTokensToUSDCForDonation function all the tokens will be transferred to "to" address and to addresses will be whitelisted.

Status

Resolved

A.16 Transaction can be reverted instead of taking fees

Description

In swapExactTokensToUSDCForDonation() the require statement should be added for require(path[0] == USDC,"err") instead of taking fees and sending remaining tokens back to the user.

Remediation

This can be achieved by removing if{} and adding suggested require check for checking path[0] == USDC.

GIE Team’s Comment

If has different functionality we can't remove this.This was not the valid issue.

Status

Resolved

A.17 Confirm that the contract doesn’t need to have functionality for adding liquidity

Description

Confirm that TokensSwap contract doesn't need “add liquidity” functionality because it would be using already created pairs and using other router contract liquidity can be added.

Remediation

Confirm that the add liquidity functionality is not required.

GIE Team’s Comment

As of now this functionality is not needed.

Status

Resolved

A.18 Condition statement in updateDiscount() can be better adjusted for easy readability

Description

Code statement implementation can be better adjusted.

Remediation

This condition statement in updateDisount() function with redundant _updateDiscount:

```
if(_tierNo > 1 && _tierNo < 4){
_preUpdateDiscountCheck(_tierNo, _amount, _discountPercent, 0);
_preUpdateDiscountCheck(_tierNo, _amount, _discountPercent, 2);
_updateDiscount(_tierNo, _amount, _discountPercent);
} else if(_tierNo == 1){
_preUpdateDiscountCheck(_tierNo, _amount, _discountPercent, 0);
_updateDiscount(_tierNo, _amount, _discountPercent);
} else{
```

```
_preUpdateDiscountCheck(_tierNo, _amount, _discountPercent, 2);
_updateDiscount(_tierNo, _amount, _discountPercent);
}
```

Can be better simplified to this:

```
if(_tierNo > 1 && _tierNo < 4){
_preUpdateDiscountCheck(_tierNo, _amount, _discountPercent, 0);
_preUpdateDiscountCheck(_tierNo, _amount, _discountPercent, 2);
} else if(_tierNo == 1){
_preUpdateDiscountCheck(_tierNo, _amount, _discountPercent, 0);
} else{
_preUpdateDiscountCheck(_tierNo, _amount, _discountPercent, 2);
}
_updateDiscount(_tierNo, _amount, _discountPercent);
```

Status

Resolved

A.19 Init code hash needs to be changed in case of deploying contract on other chain

Description

Init code hash need to be changed in case the contract would be deployed on another blockchain where the factory contract would be using different creation code while creating pairs as pairFor also needs to use the same init code hash.

Remediation

Care needs to be taken while deploying on other blockchain as suggested.

GIE Team’s Comment

We are planning to deploy our contract on Polygon only , later if require we will change the code hash for different chain.

Status

Resolved

B. Contract - GIEToken

High Severity Issues

No issues were found.

Medium Severity Issues

B.1 Centralization of control

Description

Owner can burn the tokens from anyone's account.

Remediation

Remove the burn() functionality.

Status

Resolved

Low Severity Issues

No issues were found.

Informational Issues

No issues were found.

Functional Tests

Some of the tests performed are mentioned below:

TokensSwap

- ✓ Should be able to swap with swapExactTokensForTokens
- ✓ Should be able to swap with swapTokensForExactTokens
- ✓ Should be able to swap with swapTokensForExactETH
- ✓ Should be able to swap with swapExactTokensForETH
- ✓ Should be able to swap with swapETHForExactTokens
- ✓ Should be able to swap with swapExactETHForTokens
- ✓ Should be able to swap with swapExactTokensToUSDCForDonation
- ✓ Should be able to swap with swapExactETHToUSDCForDonation
- ✓ Only owner should be able to swap with swapExactETHToUSDCForConversion
- ✓ Only owner should be able to call setDonationFees
- ✓ Only owner should be able to call setGieAppFees
- ✓ Only owner should be able to call setFeeHolderAddress
- ✓ Swap functions should be able to send fees to feeHolderAddress except swapExactETHToUSDCForConversion

GIEToken

- ✓ Should be able to mint tokens
- ✓ Should be able to transfer tokens
- ✓ Should be able to approve tokens
- ✓ Should be able to transfer approved tokens using transferFrom
- ✓ minting should increase the totalsupply
- ✓ Should revert when spender does not have enough allowance
- ✓ Should revert when sender does not have enough balance while sending tokens

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 GIE Crypto. We performed our audit according to the procedure described above.

Some issues of High, Medium, 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 GIE Crypto 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 GIE Crypto 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 GIE Crypto to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

850+
Audits Completed

$30B
Secured

$30B
Lines of Code Audited

Follow Our Journey

Audit Report
December, 2023

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com