QuillAudits

Audit Report
October, 2023

For

ZOKSH

Table of Content

Executive Summary 03

Number of Security Issues per Severity 04

Checked Vulnerabilities 05

Techniques and Methods 06

Types of Severity 07

Types of Issues 07

High Severity Issues 08

A.1: Admin fees can be bypassed via refund() by merchants 08

A.2: Initial fee amount can cause a loss of funds to merchants: POC 08

Medium Severity Issues 10

A.3: Use .call( ) instead of .send( ) 10

Low Severity Issues 11

A.4: Unit testing 11

A.5: Privilege delegation 11

Informational Issues 12

A.6: Unused contract variables and libraries 12

A.7: Naming Convention 12

A.8: GAS - Function visibility modifier can be declared external 13

A.9: Event indexing and emission 13

Table of Content

Functional Tests 14

Closing Summary 15

Executive Summary

Project Name	ZokshPay
Overview	The ZokshPay repo contains 3 contracts, GatewayAccess - an admin contract for access control which sets the admin and merchant addresses, GatewayFactory contract that is responsible for setting fees, deploying/creating routes by merchants or by the admins on behalf of the merchant, and the GatewayRoute contract.
Timeline	17th July 2023 - 25th July 2023
Method	Manual Review, Automated Review, Functional Testing
Audit Scope	The scope of this audit was to analyze ZokshPay's codebase for quality, security, and correctness.
Branch Name	Main
Commit Hash	5804421a59a0ab228f440e825788ec3301299872
Contracts in Scope	• GatewayAccess.sol • GatewayRoute.sol • GatewayFactory.sol
Fixed In	**Remediation Review:** 14th September, 2023 **Branch Name:** audit_v2 **Commit Hash:** ef0f47be10e431f68b9b1aef2f3aa7fd039399b2

Number of Security Issues per Severity

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	0	2
Partially Resolved Issues	1	0	1	1
Resolved Issues	1	1	1	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
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of call
- 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 behaviour.
- Token distribution and calculations are as per the intended behaviour 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.

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, Foundry, Solhint.

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

A.1: Admin fees can be bypassed via refund() by merchants

Description

In contract GatewayRoute the merchants are able to take their share of fare by calling the **settle()** function. The other function **refund()** is used for refunding the users if the wrong purchase was made or for some reason where the user/customer didn't like the product. But the merchant can pass his other address as the recipient to take all the money which allows the merchant to bypass Zoksh's fees.

Line

56-60

Functions - setMaxInvestment & setMinInvestment

```
0 references | Control flow graph | 82ad6f35 | ftrace | funcSig
function refund(address token1, address payable recipient1, uint amount1) external payable onlyMerchant tokenSupported(token1) {
    require(_getBalance(token1) >= amount1, "PG: INSUFFICIENT_BALANCE");
    _refund(token1, recipient1, amount1);
}
```

Remediation

To resolve the issue please make sure to update with appropriate logic. On the other hand, the team can make sure that the recipient's address should not be equal to the merchant's/ manager's address. Also, the refund process can check if the address which is to be refunded has a transaction with the merchant.

Status

Partially Resolved

Zoksh Team Comment

There is off-chain logic which ensures that only transactions processed by the contract in the past is used for refund. Meanwhile we'll add additional check to ensure that recipient address for refund is not the merchant settlement wallet address in on-chain logic.

A.2: Initial fee amount can cause a loss of funds to merchants: POC

Description

The GatewayFactory contract has the variable **setFee** initialized to 100. From the calculations in **_settle()** on L112 this would cause 100% of the amount to be deducted as fees, this way preventing the merchant from withdrawing any amounts and transferring all tokens to the **gatewayAddr**.

A.2: Initial fee amount can cause a loss of funds to merchants: **POC**

Currently in the **setFee()** function the admin can pass in any value to adjust the fee and the merchant uses **confirmFee()** to accept the adjusted fee variable. If the fee set is much higher than expected, it can limit user interaction with the protocol.

Line

112

```
function _settle(address token↑, address payable merchantAddr↑) internal {
  require(merchantExists[merchantAddr↑], "PG: ADDRESS_NOT_WHITELISTED");

  uint256 bal = _getBalance(token↑);
  uint256 feeAmount = (bal * fee) / 100;
  uint256 merAmount = bal - feeAmount;

  emit PaymentSettled(merchantAddr↑, merAmount, token↑);

  if (token↑ == address(0)) {
    if (merAmount > 0)
      require(merchantAddr↑.send(merAmount), "PG: ETH_TRANSFER_FAILED");
    if (feeAmount > 0)
      require(gatewayAddr.send(feeAmount), "PG: ETH_TRANSFER_FAILED");
  } else {
    if (merAmount > 0) IERC20(token↑).safeTransfer(merchantAddr↑, merAmount);
    if (feeAmount > 0) IERC20(token↑).safeTransfer(gatewayAddr, feeAmount);
  }
}
```

Remediation

Ensure proper fee calculations with percentages by using basis points and adjusting the **_settle()** function. Also, consider including input validation checks in the **setFee()** function in GatewayRoute as well as **deployRoute()** and **createRoute()** in GatewayFactory.

Status

Resolved

Zoksh's Team Comment

The initial fee is set to 1%, 100 represents 2 decimal points.

Auditor's Comment

As regards the fee issue (ref. L107),
if fee is 100 (representing 1% as you described), consider this:
bal = 10000, fee = 100
feeAmount = 10000 * 100 / 100 = 10000
Do you see why it's an issue? This is why the recommendation for basis points was raised.

A.2: Initial fee amount can cause a loss of funds to merchants: **POC**

Zoksh Team Comment

This bug in the calculation was fixed long back but we missed the merge to the audit branch.

We have updated the formula in the code along with the other changes.

Auditor's Comment

Fixed in this **commit** to the repo.

Medium Severity Issues

A.3: Use .call( ) instead of .send( )

Description

In GatewayRoute, function **settle()** and **refund()** send is used for transferring the ether. Future changes or forks in Ethereum can result in higher gas fees than send provides. The **.send()** creates a hard dependency on 2300 gas units being appropriate now but not in the future.

This may also affect functionality when a multi-sig wallet is used for such interactions.

Remediation

Please change all the instances of the code where send is use to call
(bool success,) = payable(recipient).call{value: msg.value}("")
require(!success, "payment failed")

Status

Resolved

Low Severity Issues

A.4: Unit testing

Description

It is highly recommended to have above 95% of code functionality tested before going into production so as to catch bugs that could be introduced from user input as well as return values from function calls.

Remediation

Consider using recent development frameworks like Hardhat or Foundry to write tests that cover interactions in the codebase and adequately test for all branches in the code.

Status

Partially Resolved

A.5: Privilege delegation

Description

From GatewayFactory, the creation of a route can happen with **createRoute()** or **deployRoute()**. When routes are created using **deployRoute()** the merchant address and manager address are the same **(msg.sender)**, but with **createRoute()** the manager can be any value passed in. The issue here is that the merchant no longer has the privileges to call **onlyMerchant()** functions if a separate manager is set when calling **createRoute()**.

Remediation

If merchants are sub-contracting their privileges to managers, they should still be able to call the **onlyMerchant()** protected functions. This could be an application for the merchantWhitelist array that is unused.

Status

Resolved

Zoksh Team Comment

deployRoute is available to whitelisted merchant wallets to sign and setup contracts themselves. createRoute is available only for cases, where a merchant coming from web 2 background, is not comfortable with setting up wallet and signing transactions, and requires initial setup be created by Zoksh team for them. In which case both address will always be same.

Informational Issues

A.6: Unused contract variables and libraries

Description

Some variables declared in the GatewayFactory and GatewayRoute contracts are unused and can be removed to reduce the contract size and clean up the codebase. In GatewayFactory, the **owner** variable declared on **L8** is unused. In GatewayRoute, the **blocked** boolean variable is unused as well.

The SafeMath library is also imported into the GatewayRoute contract on **L3** but is never used.

merchantWhitelist() also has methods created around it but is unused as well. It would be expected that this whitelist is checked to make sure that merchant addresses are whitelisted before they are used in transactions.

Remediation

Consider removing unused variables and libraries to avoid increasing contract size and deployment costs.

Status

Acknowledged

A.7: Naming Convention

Description

The naming convention used in the codebase is not clear and reduces the readability of the entire codebase. The terms manager and merchant are used concurrently but they could refer to different entities.

In GatewayFactory, the **onlyManager()** modifier checks that the caller is the same as the **merchantAddr** address provided in the Administrable (GatewayAccess) contract. Since the check is for the merchant and not the manager, it can be changed to **onlyMerchant()**.

In GatewayRoute, the **onlyMerchant()** modifier checks that the caller is the same as the **managerAddr** address provided in the creation of the GatewayRoute contract. Since the check is for the manager and not the manager, it can be changed to **onlyManager()**.

A.7: Naming Convention

Remediation

Follow industry standard naming conventions and best practices to make code easier to read for all users.

Status

Resolved

A.8: GAS - Function visibility modifier can be declared external

Description

Functions with an external visibility modifier are not called directly in the contract and are cheaper than public functions are. For functions that are not to be called internally, it is best practice to make them external for gas savings.

In GatewayRoute, **getBlocked()**, **getManager()**, **getGatewayAddr()** and **getAdminContract()** can be declared external. In GatewayAddress, **getGatewayAddress()** and **getAdminContract()** can be declared external.

Remediation

Consider making external-facing functions not used within the contract external instead of public to save some gas on the function calls for users.

Status

Partially Resolved

A.9: Event indexing and emission

Description

For ease of searching and the use of monitoring tools the following events in GatewayRoute can be indexed, PaymentSettled(), PaymentRefunded(), ManagerUpdated(). In the emission of events as well, there is no need to cast an address to payable.
In GatewayFactory, the RouteCreated event has a 4th event topic without a name.

Remediation

Consider adjusting the events as described above.

Status

Acknowledged

Functional Tests

Some of the tests performed are mentioned below:

- [PASS] test Should Fail Merchants Will Be Able To Create Route For Less Fees()
- [PASS] test settle The Funds()
- [PASS] test refund The Funds To Customer()
- [PASS] test Merchants Try To Bypass The Admin Fees()
- [PASS] test Merchant Is Able To Whitelist Address()
- [PASS] test check All Getter Functions()

Closing Summary

In this report, we have considered the security of the ZokshPay codebase. 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. In The End, the Zoksh Team has Resolved most of the issues and Acknowledged others.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in ZokshPay 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 ZokshPay 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 ZokshPay 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
October, 2023

For

ZOKSH

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com