QuillAudits

Audit Report
September, 2022

For

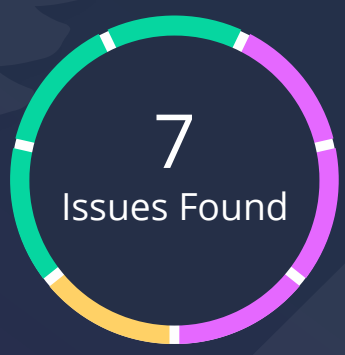
PLÜG

Table of Content

Table of Content

Executive Summary

Project Name	Plug Exchange
Timeline	2nd August, 2022 to 25th August, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze Plug Exchange codebase for quality, security, and correctness.
Contracts Under Audit Scope	1]PlugRouterUpgradeable.sol 2] Adapters Folder:- AdapterBase.sol CelerAdapter.sol HopAdapter.sol HyphenApapter.sol MultichainAdapter.sol PolygonBridgeApdater.sol SynapseAdapter.sol WormholeAdapter.sol 3]TransferHelpers
Fixed in	*https://github.com/PlugExchange/plug-smart-contracts* Commit-id: f6b7836091207e1b09af55ae0b60c5b07a9cb2c3 *https://github.com/PlugExchange/plug-smart-contracts/commit/b1f2fc91f2e353b8fcaa19db1ea60ebb7e5d589d*



High Medium Low Informational

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

Types of Severities

High

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

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

Low-level severity issues can cause minor impact and or 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 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.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Techniques and Methods

Throughout the audit of smart contract, 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 analysed 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

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 analysed, 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 behaviour 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, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Contract - PlugRouterUpgradeable

High Severity Issues

No issues found

Medium Severity Issues

A1. Compilation not successful

```
yarn run v1.22.18
$ hardhat compile
DeclarationError: Declaration "TransferHelpers" not found in "contracts/libraries/TransferHelpers.sol" (referenced as "./libraries/TransferHelpers.sol").
 --> contracts/PlugRouterUpgradeable.sol:7:1:
  |
7 | import {TransferHelpers} from './libraries/TransferHelpers.sol';
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^


Error HH600: Compilation failed

For more info go to https://hardhat.org/HH600 or run Hardhat with --show-stack-traces
error Command failed with exit code 1.
info Visit https://yarnpkg.com/en/docs/cli/run for documentation about this command.
```

Description

The compiler is throwing an error on compiling the contracts

Remediation

Change the imports to the correct contract name.

Status

Resolved

Low Severity Issues

A2. Add Check for msg.value for native and ERC-20 token transactions.

Description

The check is needed to avoid extra ETH being sent to the contract at:-
Line 285:

```
function _swap(
    address _fromToken,
    uint256 _amount,
    bytes4 _exchangeId,
    bytes calldata _swapCallData,
    bool _feeFlag
) internal whenNotPaused returns (address outToken, uint256 swapedAmount) {
```

```
function _deposit(
    address recipient,
    address token,
    address bridgeAdapter,
    uint256 amount,
    bytes calldata bridgeCallData
) internal whenNotPaused returns (uint256 toChainId) {
```

Remediation

The functions should check if the msg.value is equal to 0 if it is an ERC20 token else msg.value should be equal to the amount specified by the users.

```
if(_fromToken == NATIVE_TOKEN_ADDRESS) {
    require(amount == msg.value);
} else {
    require(msg.value == 0);
}
```

Plug Exchange Team Comment: This can not be addressed as sometimes few of the bridges used to ask for extra eth to send transactions on the other chains. (We are transferring the remaining amount to users in this case after transaction execution.)

Status

Acknowledged

Informational Issues

A3. Function input parameters lack check

```
function _setSwapFeeConfig(uint256 _swapFeePercentage, address _swapFeeCollector)
internal {
   swapFeeConfig = SwapFeeConfig({swapFeePercentage: _swapFeePercentage,
swapFeeCollector: _swapFeeCollector});
 }
```

Description

The contract allows the contract owner to update fee settings without a limit check.

Remediation

It is recommended to add checks to avoid invalid values hence reverting before the business logic executes.

Status

Resolved

B. Contract - Adapters

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

B1. Check msg.value for native and ERC-20 token transactions.

Description

The check is needed to avoid extra ETH being sent to the contract.

Remediation

The functions should check if the msg.value is equal to 0 if it is an ERC20 token else msg.value should be equal to the amount specified by the users.

```
if(_fromToken == NATIVE_TOKEN_ADDRESS) {
    require(amount == msg.value);
} else {
    require(msg.value == 0);
}
```

Status

Acknowledged

B2. Function input parameters lack check

Description

The functions lack validations on input parameters like while updating important external addresses.

Line: 20

```
constructor(address _plugRouter, address _bridgeRouter) AdapterBase(_plugRouter) {
  _cBridgeRouter = _bridgeRouter;
}
function setCbridgeRouter(address _newCBridgeRouter) external onlyOwner {
  _cBridgeRouter = _newCBridgeRouter;
}
```

Remediation

It is recommended to add checks to avoid invalid values hence reverting before the business logic executes

Status

Resolved

Informational Issues

B3. General Recommendation

Description

The contracts do not follow naming conventions and the official solidity style guide. It is recommended to improve the readability and code quality of the contracts.

Status

Acknowledged

C. Contract - TransferHelpers

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

C1. General Recommendation

Description

The contract’s name is different from the file name, it should be corrected to improve the readability and code quality of the contracts as suggested in soli.

Status

Acknowledged

Functional Testing

Some of the tests performed are mentioned below

- ✓ Should be able to deploy the contracts.
- ✓ Should be able to deposit the tokens to the PlugRouterUpgradeable contract.
- ✓ Should be able to deposit the Native tokens to the PlugRouterUpgradeable contract.
- ✓ Should be able to swap the tokens through the PlugRouterUpgradeable contract.
- ✓ Should be able to swap the Native tokens through the PlugRouterUpgradeable contract.
- ✓ Should be able to do a cross-chain swap through the PlugRouterUpgradeable contract.
- ✓ Should revert if the operation is locked.
- ✓ Should revert if the contract is paused.
- ✓ Should revert if onlyOwner functions are not called by the owner.
- ✓ Should revert if adapters are not called by PlugRouter.
- ✓ Should allow withdrawal of tokens and ETH stuck in the contract.
- ✓ Should allow the owner to update routers and bridge configs as needed.

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 Plug Exchange. We performed our audit according to the procedure described above.

Some issues of 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 Plug Exchange Team Fixed Some issues and Acknowledged others.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the Plug Exchange Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multi-step process. One audit cannot be considered enough. We recommend that the Plug Exchange team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

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.

500+
Audits Completed

$15B
Secured

500K
Lines of Code Audited

Follow Our Journey

Audit Report
September, 2022

For

PLÜG

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com