QuillAudits

Audit Report
September, 2022

For

CREPE

Table of Content

Table of Content

Executive Summary

Project Name Crepe

Overview Crepe project is a token offering using IDO that raises USDC capital. The auction amount of the Crepe Token is 200 million with no lock up or incentives. Current token price will be determined by the relative USDC TWAP price on Uniswap. Project has vested token wallets for a certain period of time in a time-sliced way.

Scope of Audit The scope of this audit was to analyse Crepe smart contract's codebase for quality, security, and correctness. This included testing of smart contracts to ensure proper logic was followed, manual analysis ,checking for bugs and vulnerabilities, checks for dead code, checks for code style, security and more. The audited contracts are as follows:

- CrepIDO.sol
- CrepeVesting.sol
- TokenERC20.sol

Branch: master
Commit: 5d3825dd71ad0ed0ff486e8780d7ba88834266a7
Fixed In: 8dedd5841be28bf0aa288b2f481eecaf4790e430

High Medium Low Informational

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

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.

Contracts

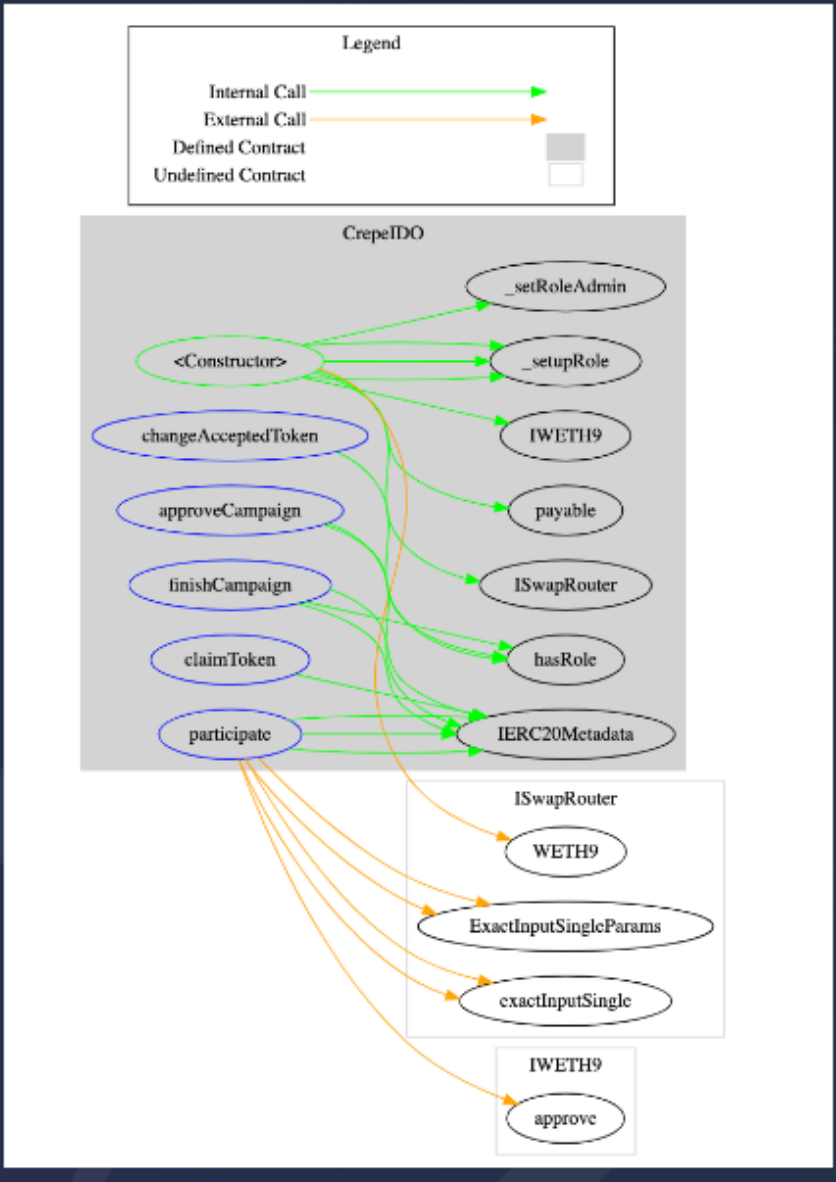
Contract	Lines	Complexity Score	Capabilities
contracts/interfaces/IWETH9.sol	9	13	payable functions
contracts/interfaces/ISwapRouter.sol	30	10	payable functions
contracts/CrepeVesting.sol	168	49	hash functions
contracts/CrepeIDO.sol	194	88	payable functions, hash functions
contracts/TokenERC20.sol	14	5	
TOTALS	415	165	payable functions, hash functions

Dependencies

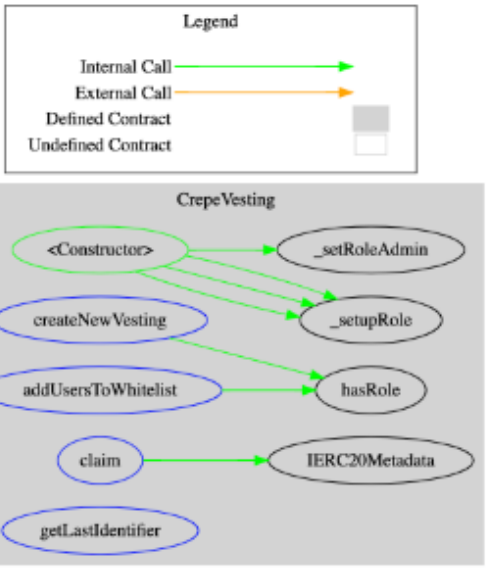
Dependency / Import Path	Count
@openzeppelin/contracts/access/AccessControl.sol	2
@openzeppelin/contracts/token/ERC20/ERC20.sol	1
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	2
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	2
@uniswap/v3-core/contracts/interfaces/callback/IUniswapV3SwapCallback.sol	1

Call Graphs

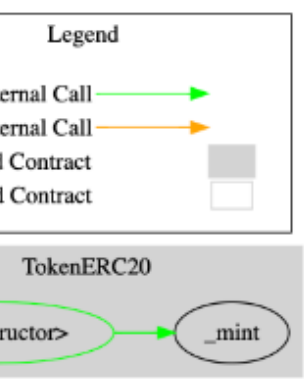
contracts/CrepeIDO.sol



contracts/CrepeVesting.sol



contracts/TokenERC20.sol



Tests

```
(node:20169) ExperimentalWarning: stream/web is an experimental feature. This
feature could change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
    ✔ STEP1. Deploy Vesting (952ms)
    ✔ STEP2. Deploy some tokens (984ms)
    ✔ STEP3. Checking access control for ADMIN_ROLE (102ms)
    ✔ STEP4. Grant role for contract controller
    ✔ STEP5. Check last identifier without campaigns
    ✔ STEP6. Start First Vesting
    ✔ STEP7. Checking requires for createNewVesting (46ms)
    ✔ STEP8. Adding users to whitelist for first campaign (83ms)
    ✔ STEP9. Checking requires for addUsersToWhitelist (48ms)
    ✔ STEP10. Checking require by time
    ✔ STEP11. Time movement in EVM
    ✔ STEP12. Approve tokens for claiming
    ✔ STEP13. Checking balance of vesting after approve tokens for token 1
    ✔ STEP14. Checking requires before claiming
    ✔ STEP15. Time movement in EVM (15 days)
    ✔ STEP16. Claiming tokens after 25% of duration vesting time
    ✔ STEP17. Checking balance after claim
    ✔ STEP18. Time movement in EVM (9 days)
    ✔ STEP19. Claiming tokens after 40% of duration vesting time (66ms)
    ✔ STEP20. Checking balance after claim
    ✔ STEP21. Time movement in EVM (21 days)
    ✔ STEP22. Claiming tokens after 75% of duration vesting time (78ms)
    ✔ STEP23. Checking balance after claim
    ✔ STEP24. Time movement in EVM (15 days)
    ✔ STEP25. Claiming tokens after 100% of duration vesting time (94ms)
    ✔ STEP26. Checking require after full claim (64ms)
    ✔ STEP27. Checking balances after full claim
    ✔ STEP28. Start Second Vesting
    ✔ STEP29. Adding users to whitelist for second campaign
    ✔ STEP30. Time movement in EVM
    ✔ STEP31. Approve tokens for claiming
    ✔ STEP32. Time movement in EVM (60 days)
    ✔ STEP33. Claiming tokens after 33.33% of duration vesting time (82ms)
    ✔ STEP34. Checking balance after claim
    ✔ STEP35. Time movement in EVM (60 days)
    ✔ STEP36. Claiming tokens after 66.66% of duration vesting time (80ms)
    ✔ STEP37. Checking balance after claim
    ✔ STEP38. Time movement in EVM (60 days)
    ✔ STEP39. Claiming tokens after 100% of duration vesting time (103ms)
    ✔ STEP40. Checking balance after claim
    ✔ STEP41. Checking vesting array by name vesting


  41 passing (9s)
```

```
  IDO test
(node:20209) ExperimentalWarning: stream/web is an experimental feature. This
feature could change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
    ✔ STEP1. Initializing of Router, USDC, WETH (68ms)
    ✔ STEP2. Purchase of WETH for MATIC (15603ms)
    ✔ STEP3. Deploy Crepe Token (435ms)
    ✔ STEP4. Deploying IDO (457ms)
    ✔ STEP5. Add accepted address (147ms)
    ✔ STEP6. Start IDO
    ✔ STEP7. Purchase for WETH (6788ms)
    ✔ STEP8. Purchase for MATIC (25634ms)
    ✔ STEP9. Finishing IDO
    ✔ STEP10. Approving campaign (42ms)
    ✔ STEP11. Start claiming Crepe tokens
    ✔ STEP12. Claiming Crepe tokens (92ms)


  12 passing (53s)
```

Manual Testing

High Severity Issues

No issues found

Medium Severity Issues

1. Reentrancy

Description
Several of the following functions in the contracts may be prone to reentrancy attacks.

CrepeIDO.sol function participate(address,uint256,uint256) line 76-139
State variables Users, TotalAccumulated and event Participated line 136-138 only updated after external call safeTransferfrom()which may be reentered by contracts. This can result in variables and events being updated for only the last reentry and missing critical updates on previous entries in participation impacting the proper accounting, logging and reporting of the contracts.
CrepeVesting.sol function claim() line 129-160 Event TokenClaimed line 152 emitted after safeTransferfrom()which may be reentered by contracts. This can result in the event being updated for only the last reentry and missing critical updates on previous entries.

Remediation
It is recommended to follow CEI (checks -effects-interactions) pattern in especially functions with external calls. This implies state updates, events emission must ideally occur before external calls. Additionally may make use of OpenZeppelin Reentrancy guard but awareness that it may result in additional gas costs so ensure CEI pattern is implemented right.

Auditor's Response: Open Zeppelin Reentrancy Guard Implemented

Status
Resolved

Low Severity Issues

2. Centralization risks and overpowered access control roles

Description
Contracts CrepeIDO.sol and CrepeVesting.sol have access control roles assigned that control critical functions and operations of the contracts. However there is no indication if these accounts will be single addresses or multisigs. If single addresses this can bring about centralization risks. Whilst access control reduces the risks by having multiple roles etc, it appears in these contracts CONTRACT_CONTROL, ADMIN roles etc are set up as the same account, which is the msg.sender() the deployer

Remediation
It is recommended to document roles, how they will work, risks of access roles, process of changing roles etc. It may be prudent to have different accounts playing different roles in various Access aspects for contracts or document this if intention. It may be prudent to have multisig account control various admin and roles for contracts.

Auditor's Response: Power over contracts after deployment to pass to addresses by customer see contracts Read.me file.

Status
Acknowledged

3. Unchecked return values

Description
Certain operations in functions are not checking the return values. Function participate() ignores the return values from calling WETH
CrepeIDO.sol line 93 WETH.approve(address(router), msg.value);
Above call to .approve() on WETH returns boolean but is not checked.

Remediation
It is recommended to check all return values from external calls were applicable require(WETH.approve(address(router), msg.value, “approval failed”)

Auditor's Response: Fixed with require check

Status
Resolved

4. Missing events

Description
Some critical actions and functions are missing events e.g
CrepeIDO.sol line 161 function changeAcceptedToken(address _token)
Above function is a critical function that changes the authorised tokens by the authorised role. Alerting of change is critical and adds transparency, allows offline monitoring and logging tools to report such changes to bring awareness to stakeholders.

CrepeIDO.sol line 188 function finishCampaign(address _recipient)
Above function is a critical function that transfers collected USDC after ending the participation. It is a critical function that may require reporting and logging via events for off chain tooling to allow for greater transparency, records and more.

Remediation
It is recommended to add events for the identified cases and any other cases as needed. It is important to emit events for critical state changes especially authorised role only functions, critical change of parameters or functioning of contracts.

Auditor's Response: Appropriate events emitted

Status
Resolved

5. Gas limit risks due to loops

Description
Looping over large arrays may be computationally intensive resulting in costly operations and potential gas limit and failed transactions.
CrepeVesting.sol line 119-21 in function addUsersToWhitelist()
Loops over an array of users address[] memory _users

Remediation
Although the above function is low risk since users array size is not determined by contracts users and function is called by Controller role, however for the controller it still can result in gas limits or expensive gas operations if impact of size of address[] memory _users,is not understood. It may be prudent to whitelist users using efficient techniques such as merkle trees. However just noting, documenting this for the Contract Controller role may be sufficient.

Auditor's Response: Acknowledged that it is not expected for the numbers to be whitelisted to be very large and the function to be frequently called. Additionally, administrators will be made aware of limitations of large size

Status
Acknowledged

Informational Issues

6. Missing events

Description
Project does not have documentation such as the Read.me file is empty. Additional documentation geared for stakeholders and or developers outlining aspects such as processes, flow, dynamics, treasury vaults, multisigs e.g timings etc. A Read.me file with sufficient information will help understand how projects can be run, dependencies, deployments, key development issues etc.

Remediation
It is recommended to add abovementioned documentation to the project

Auditor's Response: Read.me populated. Access and admin control explained in the Read.me file

Status
Resolved

7. Variables that can be declared immutable

Description
Variables below are assigned during the contract creation phase, but remain constant throughout the life-time of a deployed contract
CrepeIDO.sol line 16 uint256 public StartIn;
CrepeIDO.sol line 17 uint256 public EndIn;
CrepeIDO.sol line 18 uint256 public UnlockClaimTime;
CrepeIDO.sol line 25 address public USDC;
CrepeIDO.sol line 26 address public crepe;

Remediation
It is recommended to make the above variables immutable to save on gas costs

Auditor's Response: Variables made immutable

Status
Resolved

Closing Summary

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

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of Crepe contracts. 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 multistep process. One audit cannot be considered enough. We recommend that the Crepe 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
July, 2022

For

CREPE

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com