macro

Compound Quark A-2

Security Audit

May 20, 2024
Version 1.0.0

Presented by 0xMacro

Table of Contents

Introduction

This document includes the results of the security audit for Compound's smart contract code as found in the section titled ‘Source Code’. The security audit was performed by the Macro security team from April 30, 2024 to May 6, 2023. For a total of three audit days.

The purpose of this audit is to review the source code of certain Compound Solidity contracts, and provide feedback on the design, architecture, and quality of the source code with an emphasis on validating the correctness and security of the software in its entirety.

Disclaimer: While Macro’s review is comprehensive and has surfaced some changes that should be made to the source code, this audit should not solely be relied upon for security, as no single audit is guaranteed to catch all possible bugs.

Overall Assessment

The following is an aggregation of issues found by the Macro Audit team:

Severity	Count	Acknowledged	Won't Do	Addressed
Low	1	-	-	1
Code Quality	6	-	1	5

Compound was quick to respond to these issues.

Specification

Our understanding of the specification was based on the following sources:

- Discussions with the Compound team.
- Available public documentation and provided docs for the specific release.

Trust Model, Assumptions, and Accepted Risks (TMAAR)

Quark is an Ethereum smart contract wallet system designed to run custom code with each transaction.

Entities

- **Quark Wallet** – A programmable wallet that executes QuarkOperations containing contract code (or corresponding code reference) and calldata for triggering function from the provided contract code.
- **Quark State Manager** – A contract that provides isolated storage for each Quark wallet.

- **Code Jar** – Manages references to previously deployed and deploys new contract code to be used in future Quark operations.
- **Wallet Proxy Factory** – Enables efficient deploy of new Quark Wallet instance at a counterfactual address derived from an owner EOA based on the minimal proxy contract architecture.
- **Quark Script** – An extensible contract that exposes helper functions for other Quark scripts to inherit from. The helper functions enable callbacks, allow replay of QuarkOperations, and reading from and writing to an isolated storage space in the QuarkStateManager.
- **Core Scripts** – A set of important scripts that should be deployed via CodeJar to cover essential operations that a large number of QuarkOperations will likely use. Examples of Core Scripts include multicall, ethcall, and flashloans with callbacks.
- **User Scripts** – An unbounded and unpermissioned set of arbitrary scripts deployed by the users.

Trust Model

- Only signer of the wallet instance can authorize quark operation execution for the wallet.
- Only executor of the wallet instance can run quark operation directly on the wallet.
- Quark wallet instance related custom state is managed and available through the quark state manager.
- Quark wallet instance related custom state cannot be modified outside of the execution context of the particular wallet instance.
- Code Jar deploys the runtime code referenced by operations (if it hasn't been already deployed) and returns reference to the correct code.

- Quark Script provides properly implemented helper functions for Quark script authors to follow best practices in developing User scripts.
- Core scripts securely and properly implement corresponding operations by relying on Quark Script provided functionality.

General Assumptions

- Quark is an extensible framework that relies on plugins in the form of Scripts (Core/User) to enable flexible runtime operations.
- User scripts are arbitrary code components that are executed in the context of a specific Quark Wallet instance. They may transfer assets owned by the wallet (and have capabilities as arbitrary smart contracts). Therefore, end users need to verify their proper operation.
- New user scripts will follow QuarkWallet's guidelines for Quark user script development to ensure proper execution (usage of allowCallbacks, allowReplay, read, write for isolated storage, nonReentrant, onlyWallet, etc).
- New user scripts interacting with external systems will follow security best practices and be reviewed within a specific deployment context to ensure a lack of security issues.
- Security model for user script used within the context of QuarkWallet is similar as for an individual smart contract with the same functionality. Therefore, user script needs to be properly implemented to avoid related security risks.

Accepted Risks

- If users execute insecure arbitrary user scripts in the context of their Quark wallet instance they may lose assets owned by the corresponding wallet.
- If users reference user script which may be destroyed by others they may need to pay higher cost for their Quark operation execution since contract code will need to be redeployed.

- Valid signatures for replayable operations can be replayed by unexpected third parties arbitrary number of times. User scripts should properly handle this edge case.
- Signatures for previously failed operations may unexpectedly be successfully reused in the future. User scripts should provide ability for invalidating previous signatures.
- MultiQuark operations intended to be executed cross chain cannot guarantee the proper execution order given the conditional isolation for different chains.

Source Code

The following source code was reviewed during the audit:

- **Repository:** quark
- Initial Commit Hash **(Majority of contracts):**
  `9d8f07d571298b448852bf63997b9b5278725f3e`
- Commit Hash **(QuoteCall script addition):**
  `9b334b0239c6a10a7a34578948ca6dc86db444e0`

This security review focused primarily on general refactoring of the QuarkWallet directory structure, the introduction of MultiQuark operations, and the addition of Paycall and Quotecall scripts. More details on the specific contracts are provided below. Additionally, the script deployment was adapted to deploy the runtime code.

Specifically, we audited the following contracts with the `9d8f07d571298b448852bf63997b9b5278725f3` commit hash:

Contract	SHA256
./src/codejar/src/CodeJar.sol	75ae09c6f57e61f1ecf7e4c55b92e1ab174adfb6f3480ac848aaafee7182ac1b
./src/codejar/src/CodeJarFactory.sol	41e3f715a0a19ba364917173d1b1c712d75b497715ead351dc1d04814e57e59f
./src/quark-core/src/QuarkScript.sol	65ea45b22d7a1a5b1e5cf27bb9e83d613c7e6f6d36c18f10119e969cfafaab8d
./src/quark-core/src/QuarkStateManager.sol	648dfe12d713a48252489aa3acef75d34114a5cde84a348f18ff4fe7c1696fa9
./src/quark-core/src/QuarkWallet.sol	00a20445cf77ba6560658b7c6a92f7e160af27990b69f8b5c4e1ecb7fa2d34f4

Contract	SHA256
./src/quark-core/src/QuarkWalletStandalone.sol	56a75e4ab5679c173b4e5cc54f0e1bb8145d04eecceee1dfaf5fea62ffad8f36
./src/quark-core/src/interfaces/IHasSignerExecutor.sol	15921dc30d4be0b2a38a4c470bd2c350d9eca637ea3db6af125006cbea2ff2ae
./src/quark-core/src/interfaces/IQuarkWallet.sol	8ba603e452ad9699eb7b069cf1e917fbbdec00ea760aed0e3b4386f90ac87dba
./src/quark-core/src/periphery/BatchExecutor.sol	4a972442dbf3b4dd0e3018d33854536b539af2a228bdb343e6a7fcdce5c23f62
./src/quark-core-scripts/src/Ethcall.sol	7f11e6ebf4ea32a9e845a68a56661c6319cac0001552d4e0c96a71544a9d6272
./src/quark-core-scripts/src/Multicall.sol	af1fa4abeb7158163094c3b74e41e8617a6ae580166cb9677807b4ddd12a8dd6
./src/quark-core-scripts/src/Paycall.sol	0e5708eeb5f1dacbbcb32c790615098b5cc51c6ccacd93e35c769adb0ea96c57
./src/quark-core-scripts/src/UniswapFlashLoan.sol	3aa07833c47e7c9f87ac769e9e3cbeeb044ba24a61cd818ca69880a027a03b23
./src/quark-core-scripts/src/UniswapFlashSwapExactOut.sol	07c275e0d2f004e8228c65794f3aa6c7ccae47ce6af3f6d7c93fa61950c86781
./src/quark-core-scripts/src/lib/UniswapFactoryAddress.sol	570e490638d8ffc7c5307b445dd9b3bb2574b18dd2ae8373914d466d331b4ebb
./src/quark-core-scripts/src/vendor/chainlink/AggregatorV3Interface.sol	14d1af5988ba1a486deaa8f81f42a7992de22d0b104beb21086f3de44107134b
./src/quark-core-scripts/src/vendor/uniswap_v3_periphery/PoolAddress.sol	6ae36efb47db4d2e41f28a6d3c13e8d21cb434b6fad2c744a4712bc197b3664b

Contract	SHA256
./src/quark-factory/src/QuarkFactory.sol	`ac13523967b68f0f904e2bc0057f5ee5d4f78e008d890344c5cd6f3b38f7b365`
./src/quark-proxy/src/QuarkMinimalProxy.sol	`f2f1d6eb3dd11b333682d95de9eee80281f7a2314a1a40be2172aac1e706561f`
./src/quark-proxy/src/QuarkWalletProxyFactory.sol	`2c9b11a026ecda84f085c11e7b0bd6166a4cdc339e6166b048b551bc77ed37ff`

Additionally, we audited the following contracts with the `9b334b0239c6a10a7a34578948ca6dc86db444e` commit hash. (QuoteCall addition and minor Paycall changes):

Contract	SHA256
./src/quark-core-scripts/src/Paycall.sol	`67eda1f0702d5d0c2e055951af45bb49501f055768b62ca5191298c5e2e5fd47`
./src/quark-core-scripts/src/Quotecall.sol	`1aed845d0dcd46bce62a007d7740930c6ea092751b9366788e4ee208cc28fddd`

Note: This document contains an audit solely of the Solidity contracts listed above. Specifically, the audit pertains only to the contracts themselves, and does not pertain to any other programs or scripts, including deployment scripts.

Issue Descriptions and Recommendations

Click on an issue to jump to it, or scroll down to see them all.

Security Level Reference

We quantify issues in three parts:

1. The high/medium/low/spec-breaking **impact** of the issue:
    - How bad things can get (for a vulnerability)
    - The significance of an improvement (for a code quality issue)
    - The amount of gas saved (for a gas optimization)
2. The high/medium/low **likelihood** of the issue:
    - How likely is the issue to occur (for a vulnerability)
3. The overall critical/high/medium/low **severity** of the issue.

This third part – the severity level – is a summary of how much consideration the client should give to fixing the issue. We assign severity according to the table of guidelines below:

Severity	Description
(C-x) Critical	We recommend the client **must** fix the issue, no matter what, because not fixing would mean **significant funds/assets WILL be lost.**
(H-x) High	We recommend the client **must** address the issue, no matter what, because not fixing would be very bad, *or* some funds/assets will be lost, *or* the code's behavior is against the provided spec.
(M-x) Medium	We recommend the client to **seriously consider** fixing the issue, as the implications of not fixing the issue are severe enough to impact the project significantly, albiet not in an existential manner.
(L-x) Low	The risk is small, unlikely, or may not relevant to the project in a meaningful way. Whether or not the project wants to develop a fix is up to the goals and needs of the project.
(Q-x) Code Quality	The issue identified does not pose any obvious risk, but fixing could improve overall code quality, on-chain composability, developer ergonomics, or even certain aspects of protocol design.
(I-x) Informational	Warnings and things to keep in mind when operating the protocol. No immediate action required.
(G-x) Gas Optimizations	The presented optimization suggestion would save an amount of gas significant enough, in our opinion, to be worth the development cost of implementing it.

Issue Details

~~L-1~~ Oracle data feeds lack sanity checks

TOPIC	STATUS	IMPACT	LIKELIHOOD
Oracle	Fixed	Low	Low

`Quotecall` and `Paycall` scripts use Chainlink data feeds to fetch the native chain asset prices and calculate the gas amount needed for the transaction. However, there are no sanity checks implemented to the returned answer in the `latestRoundData()` call.

Remediations to Consider

Consider adding proper input validation to ensure the returned price.

~~Q-1~~ Encode static type array without looping

TOPIC	STATUS	QUALITY IMPACT
Optimization	Fixed	Medium

Consider encoding the `opDigest` array using `encodePacked` directly in `getDigestForMultiQuarkOperation()` to avoid looping an static type dynamic array:

```
function getDigestForMultiQuarkOperation(bytes32[] memory opDigests) public pu
    bytes memory encodedOpDigests;
```

```
        for (uint256 i = 0; i < opDigests.length; ++i) {
            encodedOpDigests = abi.encodePacked(encodedOpDigests, opDigests[i]);
        }

        bytes32 structHash = keccak256(abi.encode(MULTI_QUARK_OPERATION_TYPEHASH, k
        return keccak256(abi.encodePacked("\x19\x01", MULTI_QUARK_OPERATION_DOMAIN_
    }
```

Reference: QuarkWallet.sol#L294-302

Suggested code snippet:

```
    function getDigestForMultiQuarkOperation(bytes32[] memory opDigests) public pur
        bytes32 encodedOpDigests = keccak256(abi.encodePacked(opDigests));
        bytes32 structHash = keccak256(abi.encode(MULTI_QUARK_OPERATION_TYPEHASH, e
        return keccak256(abi.encodePacked("\x19\x01", MULTI_QUARK_OPERATION_DOMAIN_
    }
```

~~Q-2~~ Use consistent counter increments in loops

TOPIC	STATUS	QUALITY IMPACT
Coding Standards	Fixed	Low

The `Multicall` core script uses an `unchecked` block to increment the loop iteration variable. Consider increasing this in the `for` statement declaration to keep it consistent with the rest of the codebase.

~~Q-3~~ Native token decimals can be declared as constant

TOPIC	STATUS	QUALITY IMPACT
Coding Standards	Fixed	Low

Consider declaring the native token decimals as a constant in `Paycall` and `Quotecall` scripts.

```
// Note: Assumes the native token has 18 decimals
divisorScale = 10
    ** uint256(
        18 + AggregatorV3Interface(nativeTokenBasedPriceFeedAddress).decimals()
            - IERC20Metadata(paymentTokenAddress).decimals()
    );
```

Reference: Paycall.sol#L53-58

~~Q-4~~ Inaccurate comments

TOPIC	STATUS	QUALITY IMPACT
Documentation	Fixed	Low

- `codeExist()` function in `CodeJar.sol` refers to runtime code as the input parameter `code` :

  ```
  * @param code The runtime bytecode of the code to check
  ```

  Reference: CodeJar.sol#L47

Q-5 No input validation for `maxDeltaPercentage_`

TOPIC	STATUS	QUALITY IMPACT
Input Validation	Wont Do	Low

In `QuoteCall` script, the parameter `maxDeltaPercentage` set during the contract's initialization is not bounded to any value. If it is set to a value high value, it could spend a considerably higher amount of tokens than the actual gas spent. Consider limiting this value to a fair amount, for example, `1e18`, which is a 100% variation of the actual amount spent.

RESPONSE BY COMPOUND

> We would rather not explicitly enforce a bound on this value, as any bounds we pick would be subjective and arbitrary.

~~Q-6~~ `GAS_OVERHEAD` should be adapted for different scripts

TOPIC	STATUS	QUALITY IMPACT
Gas	Fixed	Low

Updated the constant to a more accurate value. Both scripts use roughly the same amount of gas, so have the same constant value.

`Paycall` and `Quotecall` scripts declare a `GAS_OVERHEAD` amount to account for the gas used by the contract itself outside the gas metrics being tracked in the `run()` logic. However, both scripts have the same value being used, even if `Quotecall` has less untracked logic in it. Consider adapting this constant for different scripts.

Disclaimer

Macro makes no warranties, either express, implied, statutory, or otherwise, with respect to the services or deliverables provided in this report, and Macro specifically disclaims all implied warranties of merchantability, fitness for a particular purpose, noninfringement and those arising from a course of dealing, usage or trade with respect thereto, and all such warranties are hereby excluded to the fullest extent permitted by law.

Macro will not be liable for any lost profits, business, contracts, revenue, goodwill, production, anticipated savings, loss of data, or costs of procurement of substitute goods or services or for any claim or demand by any other party. In no event will Macro be liable for consequential, incidental, special, indirect, or exemplary damages arising out of this agreement or any work statement, however caused and (to the fullest extent permitted by law) under any theory of liability (including negligence), even if Macro has been advised of the possibility of such damages.

The scope of this report and review is limited to a review of only the code presented by the Compound team and only the source code Macro notes as being within the scope of Macro's review within this report. This report does not include an audit of the deployment scripts used to deploy the Solidity contracts in the repository corresponding to this audit. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. In this report you may through hypertext or other computer links, gain access to websites operated by persons other than Macro. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such websites' owners. You agree that Macro is not responsible for the content or operation of such websites, and that Macro shall have no liability to your or any other person or entity for the use of third party websites. Macro assumes no responsibility for the use of third party software and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.