Report

v. 1.0

Customer

Compound Labs

Smart Contract Audit

Quark

9th April 2024

Report prepared by

ABDK Consulting

Contents

1 Changelog **5**

2 Introduction **6**

3 Project scope **7**

4 Methodology **8**

5 Our findings **9**

6 Major Issues **10**
CVF-4. FIXED 10
CVF-6. FIXED 10
CVF-9. INFO 11

7 Moderate Issues **12**
CVF-1. INFO 12
CVF-5. INFO 12
CVF-7. INFO 13
CVF-8. INFO 13
CVF-11. INFO 14
CVF-12. FIXED 14
CVF-13. INFO 15
CVF-14. INFO 16
CVF-15. INFO 16
CVF-16. INFO 17

8 Recommendations **18**
CVF-2. INFO 18
CVF-3. INFO 18
CVF-17. INFO 19
CVF-18. INFO 19
CVF-19. INFO 20
CVF-20. INFO 20
CVF-21. INFO 21
CVF-22. INFO 21
CVF-23. INFO 21
CVF-24. INFO 22
CVF-25. INFO 22
CVF-26. INFO 22
CVF-27. INFO 23
CVF-28. FIXED 23
CVF-29. INFO 23
CVF-30. INFO 24

CVF-31. INFO . . . 25
CVF-32. INFO . . . 26
CVF-33. INFO . . . 26
CVF-34. INFO . . . 27
CVF-35. INFO . . . 27
CVF-36. INFO . . . 27
CVF-37. INFO . . . 28
CVF-38. INFO . . . 28
CVF-39. INFO . . . 28
CVF-40. INFO . . . 29
CVF-41. INFO . . . 29
CVF-42. INFO . . . 29
CVF-43. INFO . . . 30
CVF-44. INFO . . . 30
CVF-45. INFO . . . 30
CVF-46. INFO . . . 31
CVF-47. INFO . . . 31
CVF-48. FIXED . . . 31
CVF-49. INFO . . . 32
CVF-50. INFO . . . 32
CVF-51. INFO . . . 32
CVF-52. INFO . . . 33
CVF-54. INFO . . . 33
CVF-55. INFO . . . 34
CVF-56. INFO . . . 34
CVF-57. INFO . . . 34
CVF-58. INFO . . . 35
CVF-59. INFO . . . 35
CVF-60. INFO . . . 35
CVF-61. INFO . . . 36
CVF-62. FIXED . . . 36
CVF-63. FIXED . . . 36
CVF-64. FIXED . . . 37
CVF-65. INFO . . . 37
CVF-66. INFO . . . 37
CVF-67. INFO . . . 38
CVF-68. FIXED . . . 38
CVF-69. FIXED . . . 38
CVF-70. INFO . . . 39
CVF-71. INFO . . . 39
CVF-72. INFO . . . 39
CVF-73. FIXED . . . 40
CVF-74. FIXED . . . 40
CVF-75. INFO . . . 40
CVF-76. FIXED . . . 41
CVF-77. INFO . . . 41

CVF-78. INFO 41
CVF-80. FIXED 42
CVF-81. FIXED 42
CVF-82. INFO 42
CVF-83. INFO 43
CVF-84. INFO 43

1 Changelog

#	Date	Author	Description
0.1	09.04.24	A. Zveryanskaya	Initial Draft
0.2	09.04.24	A. Zveryanskaya	Minor revision
1.0	09.04.24	A. Zveryanskaya	Release

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting (Mikhail Vladimirov and Dmitry Khovratovich) at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.

Compound Labs is a technology company building infrastructure for the future of finance.

3 Project scope

We were asked to review:

- Original Code
- Code with Fixes

Files:

codejar/		
CodeJar.sol	CodeJarStub.sol	
quark-core/		
QuarkScript.sol	QuarkStateManager.sol	QuarkWallet.sol
quark-core/periphery/		
BatchExecutor.sol		
quark-core-scripts/lib/		
UniswapFactory Address.sol		
quark-core-scripts/		
Ethcall.sol	Multicall.sol	UniswapFlashLoan.sol
UniswapFlashSwap ExactOut.sol		
quark-proxy/		
QuarkMinimalProxy.sol	QuarkWalletProxy Factory.sol	
terminal-scripts/interfaces/		
IComet.sol	ICometRewards.sol	
terminal-scripts/		
TerminalScript.sol		

4 Methodology

The methodology is not a strict formal procedure, but rather a selection of methods and tactics combined differently and tuned for each particular project, depending on the project structure and technologies used, as well as on client expectations from the audit.

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows best code practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places as well as their visibility scopes and access levels are relevant. At this phase, we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and done properly. At this phase, we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check if code actually does what it is supposed to do, if that algorithms are optimal and correct, and if proper data types are used. We also make sure that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

We classify issues by the following severity levels:

- **Critical issue** directly affects the smart contract functionality and may cause a significant loss.
- **Major issue** is either a solid performance problem or a sign of misuse: a slight code modification or environment change may lead to loss of funds or data. Sometimes it is an abuse of unclear code behaviour which should be double checked.
- **Moderate issue** is not an immediate problem, but rather suboptimal performance in edge cases, an obviously bad code practice, or a situation where the code is correct only in certain business flows.
- **Minor issues** contain code style, best practices and other recommendations.

5 Our findings

We found 3 major, and a few less important issues.

	Info	Fixed
Major	1	2
Moderate	9	1

Fixed 3 out of 13 issues

6 Major Issues

CVF-4 FIXED

- **Category** Suboptimal
- **Source** BatchExecutor.sol

Description Ignoring operation results could make it harder to investigate errors. Proceeding after a failed operation could cause problems in case operations within a batch relate to each other.

Recommendation Consider emitting events with operation results. Also, consider adding an extra field into the "OperationParams" structure to specify what to do in case particular operation failed. Possible values for such parameter could be: revert the whole batch, terminate batch, or proceed.

Client Comment *We added a flag to toggle on/off partial failures when batch executing operations. As for events, we purposely avoided events to make this more gas efficient. This is fine because we can trace the transaction to debug any errors.*

```
39 // We purposely ignore success and return values since the
   ↪ BatchExecutor will most likely be called by an EOA
```

CVF-6 FIXED

- **Category** Unclear behavior
- **Source** QuarkStateManager.sol

Description This loop shouldn't be executed in case "bucketNonces" equals "type(uint256).max".

Client Comment *Adopted the suggestion. Though this is misclassified as "Unclear behavior" when it should be "Suboptimal".*

```
71 for (uint256 maskOffset = 0; maskOffset < 256;) {
```

CVF-9 INFO

- **Category** Flaw
- **Source** QuarkWallet.sol

Description This makes the following two situation indistinguishable: The “EmptyCode” error thrown from this function or the same error thrown from a called script.

Recommendation Consider wrapping the revert reasons of a called script into a named error.

Client Comment *We used to wrap errors, but decided to no longer wrap errors to make decoding easier and avoid nested wrapped errors.*

308
```
revert EmptyCode();
```

328
```
    revert(add(returnData, 0x20), returnSize)
```

7 Moderate Issues

CVF-1 INFO

- **Category** Flaw
- **Source** QuarkWallet.sol

Description The QuarkWalletStandalone contract makes authorization claims that it doesn't enforce. The claims enforced in other contract, and the relationship between these two places is quite weak and unclear.

Client Comment *The issue seems more informational (e.g. unclear documentation). The contract does enforce authorization claims. It inherits QuarkWallet, which enforces the authorization.*

```
372 * @param signer_ The address that is allowed to sign QuarkOperations
      ↪  for this wallet
    * @param executor_ The address that is allowed to directly execute
      ↪ Quark scripts for this wallet
```

CVF-5 INFO

- **Category** Suboptimal
- **Source** QuarkStateManager.sol

Description Linear search is extremely inefficient and this loop won't end in any reasonable time.

Recommendation Consider using a more efficient approach. For example implement a bitmap of bucket indexes that don't have free nonces.

Client Comment *These are only meant to be called off-chain, so we would rather avoid the extra complexity.*

```
68 for (uint256 bucket = 0; bucket <= type(uint88).max;) {
```

CVF-7 INFO

- **Category** Suboptimal
- **Source** QuarkStateManager.sol

Description This loop linearly iterates through “zero” bits of “bucketNonces”, which is inefficient.

Recommendation It would be more efficient to invert “bucketNonces” and then use the Brian Kernighan algorithm to iterate through “one” bits.

Client Comment *These are only meant to be called off-chain, so we would rather avoid the extra complexity.*

71
```
for (uint256 maskOffset = 0; maskOffset < 256;) {
```

CVF-8 INFO

- **Category** Suboptimal
- **Source** QuarkWallet.sol

Description There is a common optimization when domain separator is pre-calculated in constructor and stored in an immutable variable along with check ID. In case actual chain ID is the same as the one stored in constructor, the stored domain separator is used, otherwise, domain separator is calculated again. The domain separator logic should be implemented in a proxy

Client Comment *Greatly increases deploy costs of proxies which we want to avoid.*

210
```
return keccak256(
    abi.encode(DOMAIN_TYPEHASH, keccak256(bytes(NAME)), keccak256(
       ↪ bytes(VERSION)), block.chainid, address(this))
);
```

CVF-11 INFO

- **Category** Flaw
- **Source** UniswapFactoryAddress.sol

Description The result of this function may change after deployment in case chain ID will change. This could lead to weird results.

Recommendation Consider just passing the Uniswap factory address as a constructor argument to the contracts that need such address, and storing in an immutable variable.

Client Comment *Acknowledged. Changing the chain id will cause the script to revert (assuming chain id is changed to a new chain id that does not exist).*

```
13 function getAddress() internal view returns (address) {
```

CVF-12 FIXED

- **Category** Unclear behavior
- **Source** Multicall.sol

Description The reasoning behind this check is unclear.

Recommendation Consider explaining in a comment. If the purpose of this check is to ensure the “run” function is only executed via “delegatecall”, then it would be more efficient and more reliable to store the main contract address in an immutable variable and compare “this” address to that stored value.

Client Comment *Added documentation. It was initially done in this way because CodeJar did not support custom constructors. However, since we modified CodeJar to support constructors, we can now simplify the check using an immutable variable.*

```
39 if (address(this) == thisAddress) {
40     revert InvalidCallContext();
   }
```

CVF-13 INFO

- **Category** Flaw
- **Source** QuarkStateManager.sol

Description A script may clear nonce and then set another script address for the same nonce before returning control. In such a case, the other script will be overwritten here with the original script.

Recommendation Consider forbidding such scenario.

Client Comment *This edge-case seems benign and a proper fix would add ~15k gas of overhead, so we are deciding not to prevent this. However, we have added a test case to showcase this edge-case in action.*

172
```
// if a nonce was cleared, set the nonceScriptAddress to lock nonce
   ↪ re-use to the same script address
if (cachedScriptAddress == address(0) && !isNonceSetInternal(msg.
   ↪ sender, bucket, setMask)) {
    nonceScriptAddress[msg.sender][nonce] = scriptAddress;
```

CVF-14 INFO

- **Category** Suboptimal
- **Source** QuarkScript.sol

Description Obtaining a state manager here is an expensive external call, while state manager is an immutable variable in “QuarkWallet”.

Recommendation Consider adding an abstract function “stateManager” into this abstract contract, so inherited contracts could implement it efficiently.

Client Comment *Contracts (”Quark scripts”) that inherit this contract also don’t have access to the immutable variable. The ‘QuarkWallet‘ is not the contract that inherits this abstract contract.*

```
60 self.stateManager().write(self.CALLBACK_KEY(), bytes32(uint256(
   ↪ uint160(self.stateManager().getActiveScript()))));
```

```
65 self.stateManager().write(self.CALLBACK_KEY(), bytes32(0));
```

```
69 return QuarkWallet(payable(address(this))).stateManager().clearNonce
   ↪ ();
```

```
81 return QuarkWallet(payable(address(this))).stateManager().read(key);
```

```
93 return QuarkWallet(payable(address(this))).stateManager().write(key,
   ↪  value);
```

CVF-15 INFO

- **Category** Suboptimal
- **Source** CodeJarStub.sol

Description Implementing this contract in Solidity makes things messy and unreliable.

Recommendation Consider either implementing directly in EVM bytecode (see this gist: https://gist.github.com/3sGgpQ8H/7425157593ffaba3568480470c5f2bad), or passing the byte code as a normal constructor argument.

Client Comment *We were deploying via a constructor, not a deploy function. This is now an outdated version of CodeJar.*

```
9 contract CodeJarStub {
```

CVF-16 INFO

- **Category** Suboptimal
- **Source** CodeJarStub.sol

Description Relying on a particular code size is very error-prone, as code size could be affected by compiler version and settings.

Recommendation Consider refactoring.

Client Comment *No longer relevant since CodeJar has been re-designed.*

```
39 let programSz := 20 // It's magic. It's pure darned magic. Please
      ↪ don't look behind the curtain.
```

8 Recommendations

CVF-2 INFO

- **Category** Unclear behavior
- **Source** Ethcall.sol

Recommendation Consider declaring this function as “payable” so it would be possible to provide the call value along with the call.

Client Comment *This is a Quark script that will be delegatecall/callcode’d into, so there is no need to make it ‘payable‘.*

```
18 function run(address callContract, bytes calldata callData, uint256
   ↪ callValue) external returns (bytes memory) {
```

CVF-3 INFO

- **Category** Suboptimal
- **Source** UniswapFlashSwapExactOut.sol

Recommendation This calculation could be performed by the caller. No need to do it here.

Client Comment *This is a standard implementation of calling the Uniswap ‘swap‘ function. The front-end should be allowed to pass in 0 to set a default value.*

```
57 payload.sqrtPriceLimitX96 == 0
       ? (zeroForOne ? MIN_SQRT_RATIO + 1 : MAX_SQRT_RATIO - 1)
       : payload.sqrtPriceLimitX96,
```

CVF-17 INFO

- **Category** Procedural
- **Source** IComet.sol

Description Specifying a particular compiler versions makes it harder to migrate to newer versions. Consider specifying as ”^0.8.0”.

Recommendation Laso relevant for: ICometRewards.sol, TerminalScript.sol, QuarkMinimalProxy.sol, QuarkWalletProxyFactory.sol, UniswapFactoryAddress.sol, Ethcall.sol, Multicall.sol, UniswapFlashLoan.sol, UniswapFlashSwapExactOut.sol, QuarkStateManager.sol, QuarkWallet.sol, QuarkScript.sol, CodeJar.sol, CodeJarStub.sol, BatchExecutor.sol.

Client Comment *We prefer to be more specific about the compiler version.*

```
2 pragma solidity 0.8.23;
```

CVF-18 INFO

- **Category** Bad datatype
- **Source** IComet.sol

Recommendation The return type should be “IERC20”.

Client Comment *The datatype is correct.*

```
7 function baseToken() external view returns (address);
```

CVF-19 INFO

- **Category** Bad datatype
- **Source** IComet.sol

Recommendation The type for the “asset” arguments should be “IERC20”.

Client Comment *The datatype is correct.*

```
9  function supply(address asset, uint256 amount) external;

11 function supplyTo(address dst, address asset, uint256 amount)
   ↪ external;

13 function supplyFrom(address from, address dst, address asset,
   ↪ uint256 amount) external;

15 function withdraw(address asset, uint256 amount) external;

17 function withdrawTo(address to, address asset, uint256 amount)
   ↪ external;

19 function withdrawFrom(address src, address to, address asset,
   ↪ uint256 amount) external;

29 function collateralBalanceOf(address account, address asset)
   ↪ external view returns (uint128);

31 function getAssetInfoByAddress(address asset) external view returns
   ↪ (AssetInfo memory);
```

CVF-20 INFO

- **Category** Bad datatype
- **Source** IComet.sol

Recommendation The argument type should be more specific.

Client Comment *The datatype is correct.*

```
23 function getPrice(address priceFeed) external view returns (uint256)
   ↪ ;
```

CVF-21 INFO

- **Category** Documentation
- **Source** IComet.sol

Description The number format of the returned value is unclear.

Recommendation Consider documenting.

Client Comment *This is an interface from an external codebase.*

```
23 function getPrice(address priceFeed) external view returns (uint256)
       ↪ ;
```

CVF-22 INFO

- **Category** Suboptimal
- **Source** IComet.sol

Description Declaring a top-level structure in a file named after an interface makes it harder to navigate through code.

Recommendation Consider either moving the struct declaration into the interface or moving it into a separate file.

Client Comment *This is an interface from an external codebase.*

```
40 struct AssetInfo {
```

CVF-23 INFO

- **Category** Bad datatype
- **Source** IComet.sol

Recommendation The type for this field should be “IERC20”.

Client Comment *The datatype is correct.*

```
42 address asset;
```

CVF-24 INFO

- **Category** Bad datatype
- **Source** IComet.sol

Recommendation The type for this field should be more specific.

Client Comment *The datatype is correct.*

```
43 address priceFeed;
```

CVF-25 INFO

- **Category** Documentation
- **Source** IComet.sol

Description The number format for these fields is unclear.

Recommendation Consider documenting.

Client Comment *This is an interface from an external codebase.*

```
45 uint64 borrowCollateralFactor;
   uint64 liquidateCollateralFactor;
   uint64 liquidationFactor;
```

CVF-26 INFO

- **Category** Bad datatype
- **Source** ICometRewards.sol

Recommendation The type for the “comet” argument should be “IComet”.

Client Comment *The datatype is correct.*

```
5 function claim(address comet, address src, bool shouldAccrue)
     ↪ external;
```

CVF-27 INFO

- **Category** Suboptimal
- **Source** ICometRewards.sol

Recommendation This function should return the claimed amount.

Client Comment *This is an interface of a third-party contract that exists outside of Quark. We cannot just add a return value to it.*

```
5 function claim(address comet, address src, bool shouldAccrue)
    ↪ external;
```

CVF-28 FIXED

- **Category** Procedural
- **Source** TerminalScript.sol

Recommendation This library should be moved into a separate file named "TerminalErrors.sol".

Client Comment *Done. We moved the library into a separate file called "LegendErrors.sol". "TerminalScripts.sol" has been renamed to "LegendScripts.sol".*

```
13 library TerminalErrors {
```

CVF-29 INFO

- **Category** Suboptimal
- **Source** TerminalScript.sol

Recommendation This error could be made more useful by adding certain parameters into it.

Client Comment *The custom error name is self-evident.*

```
14 error InvalidInput();
```

CVF-30 INFO

- **Category** Procedural
- **Source** TerminalScript.sol

Recommendation This contract should be moved into a separate file named “CometSupplyActions.sol”.

Client Comment *We prefer to keep all the Legend scripts in one file.*

```
19 contract CometSupplyActions {
```

CVF-31 INFO

- **Category** Bad datatype
- **Source** TerminalScript.sol

Recommendation The type for the “comet” arguments should be “IComet“.

Client Comment *The datatype is correct.*

```
28  function supply(address comet, address asset, uint256 amount)
       ↪ external {

40  function supplyTo(address comet, address to, address asset, uint256
       ↪ amount) external {

53  function supplyFrom(address comet, address from, address to, address
       ↪  asset, uint256 amount) external {

64  function supplyMultipleAssets(address comet, address[] calldata
       ↪ assets, uint256[] calldata amounts) external {

88  function withdraw(address comet, address asset, uint256 amount)
       ↪ external {

99  function withdrawTo(address comet, address to, address asset,
       ↪ uint256 amount) external {

111 function withdrawFrom(address comet, address from, address to,
       ↪ address asset, uint256 amount) external {

121 function withdrawMultipleAssets(address comet, address[] calldata
       ↪ assets, uint256[] calldata amounts) external {

235 function claim(address cometRewards, address comet, address
       ↪ recipient) external {

245     address comet,

270 function run(address comet, address[] calldata assets, uint256[]
       ↪ calldata amounts, address baseAsset, uint256 repay)
```

CVF-32 INFO

- **Category** Bad datatype
- **Source** TerminalScript.sol

Recommendation The type for the “asset” arguments should be “IERC20”.

Client Comment *The datatype is correct.*

```
28 function supply(address comet, address asset, uint256 amount)
   ↪ external {
```

```
40 function supplyTo(address comet, address to, address asset, uint256
   ↪ amount) external {
```

```
53 function supplyFrom(address comet, address from, address to, address
   ↪  asset, uint256 amount) external {
```

```
88 function withdraw(address comet, address asset, uint256 amount)
   ↪ external {
```

```
99 function withdrawTo(address comet, address to, address asset,
   ↪ uint256 amount) external {
```

```
111 function withdrawFrom(address comet, address from, address to,
   ↪ address asset, uint256 amount) external {
```

CVF-33 INFO

- **Category** Bad datatype
- **Source** TerminalScript.sol

Recommendation The type for the “assets” arguments should be “IERC20[]”.

Client Comment *The datatype is correct.*

```
64 function supplyMultipleAssets(address comet, address[] calldata
   ↪ assets, uint256[] calldata amounts) external {
```

```
121 function withdrawMultipleAssets(address comet, address[] calldata
   ↪ assets, uint256[] calldata amounts) external {
```

```
270 function run(address comet, address[] calldata assets, uint256[]
   ↪ calldata amounts, address baseAsset, uint256 repay)
```

CVF-34 INFO

- **Category** Suboptimal
- **Source** TerminalScript.sol

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays. This would also make the length check unnecessary.

Client Comment *We prefer to keep these as separate arrays. Combining these into a struct complicates the calldata.*

```
64 function supplyMultipleAssets(address comet, address[] calldata
   ↪ assets, uint256[] calldata amounts) external {
```

CVF-35 INFO

- **Category** Procedural
- **Source** TerminalScript.sol

Recommendation This contract should be moved into a separate file named "CometWithdrawActions.sol".

Client Comment *We prefer to keep all the Legend scripts in one file.*

```
79 contract CometWithdrawActions {
```

CVF-36 INFO

- **Category** Suboptimal
- **Source** TerminalScript.sol

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays. This would also make the length check unnecessary.

Client Comment *We prefer to keep these as separate arrays. Combining these into a struct complicates the calldata.*

```
121 function withdrawMultipleAssets(address comet, address[] calldata
    ↪ assets, uint256[] calldata amounts) external {
```

CVF-37 INFO

- **Category** Procedural
- **Source** TerminalScript.sol

Recommendation This contract should be moved into a separate file named “Uniswap-SwapActions.sol”.

Client Comment *We prefer to keep all the Legend scripts in one file.*

```
135 contract UniswapSwapActions {
```

CVF-38 INFO

- **Category** Bad datatype
- **Source** TerminalScript.sol

Recommendation The type for these fields should be “IERC20”.

Client Comment *The datatype is correct. The official Uniswap contracts uses ‘address‘ as well.*

```
141 address tokenFrom;
```

```
153 address tokenFrom;
```

CVF-39 INFO

- **Category** Procedural
- **Source** TerminalScript.sol

Recommendation This contract should be moved into a separate file named “TransferActions.sol”.

Client Comment *We prefer to keep all the Legend scripts in one file.*

```
202 contract TransferActions is QuarkScript {
```

CVF-40 INFO

- **Category** Bad datatype
- **Source** TerminalScript.sol

Recommendation The type for the “token” argument should be “IERC20”.

Client Comment *The datatype is correct.*

```
211 function transferERC20Token(address token, address recipient,
      ↪ uint256 amount) external onlyWallet {
```

CVF-41 INFO

- **Category** Procedural
- **Source** TerminalScript.sol

Recommendation This contract should be moved into a separate file named “CometClaimRewards.sol”.

Client Comment *We prefer to keep all the Legend scripts in one file.*

```
228 contract CometClaimRewards {
```

CVF-42 INFO

- **Category** Bad datatype
- **Source** TerminalScript.sol

Recommendation This contract should be moved into a separate file named “CometSupplyMultipleAssetsAndBorrow.sol”.

Client Comment *We prefer to keep all the Legend scripts in one file.*

```
240 contract CometSupplyMultipleAssetsAndBorrow {
```

CVF-43 INFO

- **Category** Suboptimal
- **Source** TerminalScript.sol

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays. This would also make the length check unnecessary.

Client Comment *We prefer to keep these as separate arrays. Combining these into a struct complicates the calldata.*

```
246 address[] calldata assets,
    uint256[] calldata amounts,
```

CVF-44 INFO

- **Category** Procedural
- **Source** TerminalScript.sol

Recommendation This contract should be moved into a separate file named "CometRepayAndWithdrawMultipleAssets.sol".

Client Comment *We prefer to keep all the Legend scripts in one file.*

```
266 contract CometRepayAndWithdrawMultipleAssets {
```

CVF-45 INFO

- **Category** Suboptimal
- **Source** TerminalScript.sol

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays. This would also make the length check unnecessary.

Client Comment *We prefer to keep these as separate arrays. Combining these into a struct complicates the calldata.*

```
270 function run(address comet, address[] calldata assets, uint256[]
      ↪ calldata amounts, address baseAsset, uint256 repay)
```

CVF-46 INFO

- **Category** Bad datatype
- **Source** QuarkMinimalProxy.sol

Recommendation The type for this variable should be “QuarkWallet”.

Client Comment *The datatype is correct.*

```
12 address internal immutable walletImplementation;
```

CVF-47 INFO

- **Category** Bad datatype
- **Source** QuarkMinimalProxy.sol

Recommendation The type for the “implementation” argument should be “QuarkWallet”.

Client Comment *The datatype is correct.*

```
20 constructor(address implementation_, address signer_, address
   ↪ executor_) {
```

CVF-48 FIXED

- **Category** Procedural
- **Source** QuarkMinimalProxy.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

Client Comment *Added a comment.*

```
45 receive() external payable {}
```

CVF-49 INFO

- **Category** Bad datatype
- **Source** QuarkWalletProxyFactory.sol

Recommendation The type for the “walletAddress” parameter should be “QuarkWallet”.

Client Comment *The datatype is correct.*

```
14 event WalletDeploy(address indexed signer, address indexed executor,
   ↪  address walletAddress, bytes32 salt);
```

CVF-50 INFO

- **Category** Bad datatype
- **Source** QuarkWalletProxyFactory.sol

Recommendation The type for this variable should be “QuarkWallet”.

Client Comment *The datatype is correct.*

```
23 address public immutable walletImplementation;
```

CVF-51 INFO

- **Category** Bad datatype
- **Source** QuarkWalletProxyFactory.sol

Recommendation The argument type should be “QuarkWallet”.

Client Comment *The datatype is correct.*

```
26 constructor(address walletImplementation_) {
```

CVF-52 INFO

- **Category** Bad datatype
- **Source** QuarkWalletProxyFactory.sol

Recommendation The return type should be “QuarkWallet”.

Client Comment *The datatype is correct.*

```
69 function create(address signer, address executor, bytes32 salt)
      ↪ public returns (address payable) {
```

```
128 function walletAddressFor(address signer, address executor) external
      ↪  view returns (address payable) {
```

```
140 function walletAddressForSalt(address signer, address executor,
      ↪ bytes32 salt) public view returns (address payable) {
```

CVF-54 INFO

- **Category** Bad datatype
- **Source** UniswapFactoryAddress.sol

Recommendation The type for these constants should be “IUniswapV3Factory”.

Client Comment *The datatype is correct.*

```
6 address internal constant MAINNET = 0
     ↪ x1F98431c8aD98523631AE4a59f267346ea31F984;
  address internal constant CELO = 0
     ↪ xAfE208a311B21f13EF87E33A90049fC17A7acDEc;
  address internal constant BNB = 0
     ↪ xdB1d10011AD0Ff90774D0C6Bb92e5C5c8b4461F7;
  address internal constant BASE = 0
     ↪ x33128a8fC17869897dcE68Ed026d694621f6FDfD;
```

CVF-55 INFO

- **Category** Bad datatype
- **Source** UniswapFactoryAddress.sol

Recommendation The return type should be “IUniswapV3Factory”.

Client Comment *The datatype is correct.*

```
13 function getAddress() internal view returns (address) {
```

CVF-56 INFO

- **Category** Readability
- **Source** UniswapFactoryAddress.sol

Recommendation Should be “else if”.

Client Comment *This is a stylistic suggestion that we prefer not to adopt since each case is a one-line return statement.*

```
15 if (block.chainid == 10) return MAINNET; // Optimism mainnet
   if (block.chainid == 42161) return MAINNET; // Arbitrum mainnet
   if (block.chainid == 137) return MAINNET; // Polygon mainnet
   if (block.chainid == 5) return MAINNET; // Goerli testnet
   if (block.chainid == 56) return BNB; // Binance Smart Chain mainnet
20 if (block.chainid == 8453) return BASE; // Base mainnet
   if (block.chainid == 42220) return CELO; // Celo mainnet
```

CVF-57 INFO

- **Category** Readability
- **Source** UniswapFactoryAddress.sol

Recommendation Should be “else revert”.

Client Comment *This is a stylistic suggestion that we prefer not to adopt.*

```
22 revert UnrecognizedChain(block.chainid);
```

CVF-58 INFO

- **Category** Suboptimal
- **Source** Multicall.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

Client Comment *The custom error name is self-evident.*

```
11 error InvalidCallContext();
   error InvalidInput();
```

CVF-59 INFO

- **Category** Suboptimal
- **Source** Multicall.sol

Recommendation It would be more efficient to pass a single array of structs with two fields, rather than two parallel arrays. This would also make the length check unnecessary.

Client Comment *We prefer to keep these as separate arrays. Combining these into a struct complicates the calldata.*

```
32 function run(address[] calldata callContracts, bytes[] calldata
      ↪ callDatas) external returns (bytes[] memory) {
```

CVF-60 INFO

- **Category** Suboptimal
- **Source** UniswapFlashLoan.sol

Recommendation This error could be made more useful by adding certain parameters into it.

Client Comment *The custom error name is self-evident.*

```
16 error InvalidCaller();
```

CVF-61 INFO

- **Category** Bad datatype
- **Source** UniswapFlashLoan.sol

Recommendation The type for these fields should be “IERC20”.

Client Comment *The datatype is correct.*

```
29 address token0;
30 address token1;
```

CVF-62 FIXED

- **Category** Suboptimal
- **Source** UniswapFlashLoan.sol

Recommendation These two assignments could be merged into one.

```
47 (payload.token0, payload.token1) = (payload.token1, payload.token0);
   (payload.amount0, payload.amount1) = (payload.amount1, payload.
      ↪ amount0);
```

CVF-63 FIXED

- **Category** Suboptimal
- **Source** UniswapFlashLoan.sol

Recommendation The expression “input.amount0 + fee0” is calculated twice.

```
88 if (input.amount0 + fee0 > 0) {
       IERC20(input.poolKey.token0).safeTransfer(address(pool), input.
          ↪ amount0 + fee0);
```

CVF-64 FIXED

- **Category** Suboptimal
- **Source** UniswapFlashLoan.sol

Recommendation The expression “input.amount1 + fee1” is calculated twice.

```
92 if (input.amount1 + fee1 > 0) {
       IERC20(input.poolKey.token1).safeTransfer(address(pool), input.
          ↪ amount1 + fee1);
```

CVF-65 INFO

- **Category** Procedural
- **Source** UniswapFlashSwapExactOut.sol

Recommendation Consider importing these values from the original “TickMath” library.

Client Comment *We prefer to hardcode these values instead of importing another third-arty library.*

```
20 uint160 internal constant MIN_SQRT_RATIO = 4295128739;
```

```
23 uint160 internal constant MAX_SQRT_RATIO =
      ↪ 1461446703485210103287273052203988822378723970342;
```

CVF-66 INFO

- **Category** Suboptimal
- **Source** UniswapFlashSwapExactOut.sol

Recommendation This error could be made more useful by adding certain parameters into it.

Client Comment *The custom error name is self-evident.*

```
25 error InvalidCaller();
```

CVF-67 INFO

- **Category** Bad datatype
- **Source** UniswapFlashSwapExactOut.sol

Recommendation The type for these fields should be “IERC20”.

Client Comment *The datatype is correct.*

```
36 address tokenIn;
   address tokenOut;
```

CVF-68 FIXED

- **Category** Procedural
- **Source** QuarkStateManager.sol

Recommendation This interface should be defined in a separate file named “IExecutor.sol”.

Client Comment *Moved ‘IExecutor‘ into a separate file called ‘IQuarkWallet.sol‘.*

```
4 interface IExecutor {
```

CVF-69 FIXED

- **Category** Documentation
- **Source** QuarkStateManager.sol

Description The semantics of the returned value is unclear.

Recommendation Consider giving a descriptive name to the returned value and/or documenting.

Client Comment *Documented the return value with a natspec comment.*

```
7 returns (bytes memory);
```

CVF-70 INFO

- **Category** Bad naming
- **Source** QuarkStateManager.sol

Recommendation Events are usually named via nouns, such as “ClearedNounce”.

Client Comment *We prefer to keep these events as they currently are.*

```
17 event ClearNonce(address indexed wallet, uint96 nonce);
```

CVF-71 INFO

- **Category** Suboptimal
- **Source** QuarkStateManager.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

Client Comment *The custom error names are self-evident.*

```
19 error NoActiveNonce();
20 error NoUnusedNonces();
   error NonceAlreadySet();
   error NonceScriptMismatch();
```

CVF-72 INFO

- **Category** Suboptimal
- **Source** QuarkStateManager.sol

Recommendation 248 bits would be enough for a bucket index.

Client Comment *256 bits is a single word size. Using 248 bits would likely cost more gas due to the extra conversion.*

```
31 mapping(address wallet => mapping(uint256 bucket => uint256 bitset))
      ↪  public nonces;
```

CVF-73 FIXED

- **Category** Documentation
- **Source** QuarkStateManager.sol

Description This comment is confusing.

Recommendation Consider removing it.

Client Comment *Removed.*

```
100 // the last 20 bytes is the address
```

CVF-74 FIXED

- **Category** Procedural
- **Source** QuarkStateManager.sol

Recommendation This check should be done before calling “setNonceInternal”.

Client Comment *Adopted the suggestion to have checks be before the effects.*

```
162 if ((cachedScriptAddress != address(0)) && (cachedScriptAddress !=
    ↪ scriptAddress)) {
```

CVF-75 INFO

- **Category** Procedural
- **Source** QuarkWallet.sol

Recommendation This library should be defined in a separate file named “QuarkWallet-Metadata.sol”.

Client Comment *We prefer to keep QuarkWalletMetadata in the same file as QuarkWallet because the metadata is tied to this implementation of the wallet.*

```
16 library QuarkWalletMetadata {
```

CVF-76 FIXED

- **Category** Procedural
- **Source** QuarkWallet.sol

Recommendation This interface should be defined in a separate file named “HasSignerExecutor.sol”.

Client Comment *Moved ‘HasSignerExecutor‘ into a separate file.*

41
```
interface HasSignerExecutor {
```

CVF-77 INFO

- **Category** Suboptimal
- **Source** QuarkWallet.sol

Recommendation These errors could be made more useful by adding certain parameters into them.

Client Comment *The custom error names are self-evident.*

53
```
error AmbiguousScript();
error BadSignatory();
error EmptyCode();
error InvalidEIP1271Signature();
error InvalidSignature();
error NoActiveCallback();
error SignatureExpired();
```

CVF-78 INFO

- **Category** Bad naming
- **Source** QuarkWallet.sol

Recommendation Events are usually named via nouns, such as “QuarkScriptExecution”.

Client Comment *We prefer to keep these events as they currently are.*

66
```
event ExecuteQuarkScript(address indexed executor, address indexed
    ↪ scriptAddress, uint96 indexed nonce, ExecutionType
    ↪ executionType);
```

CVF-80 FIXED

- **Category** Procedural
- **Source** QuarkWallet.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

Client Comment *Added a comment.*

```
355 receive() external payable {}
```

CVF-81 FIXED

- **Category** Procedural
- **Source** QuarkWallet.sol

Recommendation This contract should be defined in a separate file named "QuarkWallet-Standalone.sol".

Client Comment *Moved 'QuarkWalletStandalone' into a separate file.*

```
363 contract QuarkWalletStandalone is QuarkWallet, HasSignerExecutor {
```

CVF-82 INFO

- **Category** Suboptimal
- **Source** QuarkScript.sol

Description Reporting the same error in different situations could make problem investigations harder.

Recommendation Consider reporting different errors.

Client Comment *This custom error correctly conveys that the error stems from reentrancy protection.*

```
20 revert ReentrantCall();
```

```
44 revert ReentrantCall();
```

CVF-83 INFO

- **Category** Bad datatype
- **Source** CodeJar.sol

Recommendation The type for this variable should be ”address” and the deployed contract may implement arbitrary API.

Client Comment *This code no longer exists after the recent changes to CodeJar.*

```
27 CodeJarStub script;
```

CVF-84 INFO

- **Category** Suboptimal
- **Source** CodeJar.sol

Description The first part of the condition is redundant.

Recommendation Consider removing it.

Client Comment *This code no longer exists after the recent changes to CodeJar.*

```
48 return codeAddress.code.length != 0 && codeAddress.codehash ==
   ↪ keccak256(code);
```

ABDK
Consulting

About us

Established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function.

The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

Contact

Email

dmitry@abdkconsulting.com

Website

abdk.consulting

Twitter

twitter.com/ABDKconsulting

LinkedIn

linkedin.com/company/abdk-consulting