Report

v. 2.0

Customer

Uniswap

Smart Contract Audit

Universal Router

17th November 2022

Report prepared by

ABDK Consulting

Contents

1 Changelog

#	Date	Author	Description
0.1	16.11.22	A. Zveryanskaya	Initial Draft
0.2	16.11.22	A. Zveryanskaya	Minor revision
1.0	16.11.22	A. Zveryanskaya	Release
1.1	17.11.22	A. Zveryanskaya	CVF-10,12,82,83 marked as fixed
2.0	17.11.22	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.

The Uniswap Protocol is an open-source protocol for providing liquidity and trading ERC20 tokens on Ethereum. It eliminates trusted intermediaries and unnecessary forms of rent extraction, allowing for safe, accessible, and efficient exchange activity. The protocol is non-upgradable and designed to be censorship resistant.

3 Project scope

We were asked to review:

- Original Repository
- Fix Repository

Files:

/			
	Router.sol		
base/			
	Dispatcher.sol	RewardsCollector.sol	RouterCallbacks.sol
interfaces/			
	IRewardsCollector.sol	IRouter.sol	
interfaces/external/			
	IWETH9.sol	ICryptoPunksMarket.sol	
libraries/			
	Commands.sol	Constants.sol	
modules/			
	Payments.sol	Permit2Payments.sol	
modules/uniswap/v2/			
	UniswapV2Library.sol	V2SwapRouter.sol	BytesLib.sol
	V3Path.sol	V3SwapRouter.sol	UniswapPoolHelper.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.

5 Our findings

We found 3 critical, 11 major, and a few less important issues. All identified Critical issues have been fixed. We have checked that the Major issues that have not been fixed, do not pose a threat (see client's comments in each case).

Issues / Severity	Active	Fixed
Critical	0	3
Major	2	8
Moderate	1	2
Minor	76	9

Fixed 21 out of 102 issues

6 Critical Issues

CVF-54. FIXED

- **Category** Flaw
- **Source** V3SwapRouter.sol

Description This function doesn't verify the caller, so anyone can call it with arbitrary payer address and take funds approved by the payer to the router.

Recommendation Consider verifying the caller as the original Uniswap V3 router does: https://github.com/Uniswap/v3-periphery/blob/main/contracts/SwapRouter.sol#L65

```
function uniswapV3SwapCallback(int256 amount0Delta, int256
    ↪ amount1Delta, bytes calldata data) external {
```

CVF-60. FIXED

- **Category** Flaw
- **Source** V3SwapRouter.sol

Description Here, the "amountIn" value is the input amount of the last swap, while "amountInMaximum" is the maximum input amount for the first swap, thus comparing them doesn't make make sense.

```
if (amountIn > amountInMaximum) revert V3TooMuchRequested();
```

CVF-88. FIXED

- **Category** Overflow/Underflow
- **Source** Permit2Payments.sol

Description Overflow is possible when converting "amount" to "uint256".

```
else permit2TransferFrom(token, payer, recipient, uint160(amount));
```

7 Major Issues

CVF-6. FIXED

- **Category** Unclear behavior
- **Source** V2SwapRouter.sol

Description This line would throw in case “path” has less than two elements.

Recommendation Consider reverting with a meaningful error in such a case.

```
UniswapV2Library.pairAndToken0For(V2_FACTORY, PAIR_INIT_CODE_HASH, path[0], path[1]);
```

CVF-7. FIXED

- **Category** Overflow/Underflow
- **Source** V2SwapRouter.sol

Description Underflow would be possible here unless the previous line implicitly checked the “path” length. Such implicit dependencies make code more error-prone.

Recommendation Consider explicitly performing all necessary checks.

```
for (uint256 i; i < path.length - 1; i++) {
```

CVF-25. FIXED

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

Description These function don't handle special recipient addresses such as "MSG_SENDER" or "ADDRESS_THIS".

Recommendation Consider handling them.

```
function pay(address token, address recipient, uint256 value)
   ↪ internal {
```

```
function payPortion(address token, address recipient, uint256 bips)
   ↪ internal {
```

```
function sweep(address token, address recipient, uint256
   ↪ amountMinimum) internal {
```

```
function sweepERC721(address token, address recipient, uint256 id)
   ↪ internal {
```

```
function sweepERC1155(address token, address recipient, uint256 id,
   ↪ uint256 amount) internal {
```

```
function wrapETH(address recipient, uint256 amount) internal {
```

```
function unwrapWETH9(address recipient, uint256 amountMinimum)
   ↪ internal {
```

CVF-31. FIXED

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

Description Unlike the "sweep" function this function treats the passed amount as an exact amount to be transferred rather than as a minimum.

Recommendation Consider not used "sweep" word here to avoid confusion.

```
function sweepERC1155(address token, address recipient, uint256 id,
   ↪ uint256 amount) internal {
```

CVF-32. INFO

- **Category** Flaw
- **Source** Payments.sol

Description The returned value is ignored.

Recommendation Consider explicitly requiring the returned value to be true.

Client Comment *WETH always reverts on a bad transfer and never returns false.*

```
IWETH9(Constants.WETH9).transfer(recipient, amount);
```

CVF-57. FIXED

- **Category** Suboptimal
- **Source** V3SwapRouter.sol

Description This allocates a new bytes array and copies the remaining path into it. This is suboptimal.

Recommendation Consider modifying the existing array in place.

```
path = path.skipToken();
```

CVF-67. INFO

- **Category** Overflow/Underflow
- **Source** V3Path.sol

Description Underflow is possible here.

Recommendation Consider using safe math.

Client Comment *Given that skipToken is only called when hasMultiplePools==true this overflow can never occur.*

```
return path.slice(NEXT_OFFSET, path.length - NEXT_OFFSET);
```

CVF-69. FIXED

- **Category** Suboptimal
- **Source** BytesLib.sol

Description These conditions may never be true, as Solidity 0.8.0+ automatically performs overflow checks.

Recommendation Consider wrapping these conditions into an “unchecked” block.

```
if (_length + 31 < _length) revert SliceOverflow();
if (_start + _length < _start) revert SliceOverflow();
```

```
if (_start + 20 < _start) revert ToAddressOverflow();
```

```
if (_start + 3 < _start) revert ToUint24Overflow();
```

CVF-80. FIXED

- **Category** Suboptimal
- **Source** Dispatcher.sol

Description A linear chain of conditional statements is inefficient.

Recommendation Consider using a tree of conditional statements: if (comment < ...) { if (command < ...) { ... } else { ... } else { if (command < ...) { ... } else { ... } }

```
if (command == Commands.PERMIT2_PERMIT) {
```

```
} else if (command == Commands.PERMIT2_PERMIT_BATCH) {
```

```
} else if (command == Commands.PERMIT2_TRANSFER_FROM) {
```

```
} else if (command == Commands.PERMIT2_TRANSFER_FROM_BATCH) {
```

CVF-81. FIXED

- **Category** Suboptimal
- **Source** Dispatcher.sol

Description Decoding a single bytes array from a bytes array looks like waste of gas.

Recommendation Consider using “inputs” as is.

```
(bytes memory data) = abi.decode(inputs, (bytes));
```

```
(bytes memory data) = abi.decode(inputs, (bytes));
```

```
(bytes memory data) = abi.decode(inputs, (bytes));
```

8 Moderate Issues

CVF-48. INFO

- **Category** Flaw
- **Source** RewardsCollector.sol

Description The returned value is ignored.

Recommendation Consider explicitly checking that the returned value is true.

Client Comment *Given that skipToken is only called when hasMultiplePools==true this overflow can never occur.*

```
looksRareToken.transfer(routerRewardsDistributor, balance);
```

CVF-63. FIXED

- **Category** Overflow/Underflow
- **Source** V3Path.sol

Description Underflow is possible here.

```
return ((path.length - ADDR_SIZE) / NEXT_OFFSET);
```

CVF-75. FIXED

- **Category** Procedural
- **Source** RouterCallbacks.sol

Description According to the EIP-165 standard, this function should return true for interfaceId=0×01ffc9a7 which is the EIP-165 interface ID.

```
function supportsInterface(bytes4 interfaceId) external pure returns
    ↪ (bool) {
```

9 Minor Issues

CVF-1. INFO

- **Category** Bad datatype
- **Source** V2SwapRouter.sol

Recommendation The type of this variable should be “IUniswapV2Factory”.

Client Comment *Most of the ”bad datatype” changes either increase gas or are incompatible with the datatypes that we use in imported libraries from other uniswap versions - we use address in those.*

```
address internal immutable V2_FACTORY;
```

CVF-2. INFO

- **Category** Readability
- **Source** V2SwapRouter.sol

Description UPPER_CASE is normally used for constants.

Recommendation Consider using camelCase.

```
bytes32 internal immutable PAIR_INIT_CODE_HASH;
```

CVF-3. INFO

- **Category** Procedural
- **Source** V2SwapRouter.sol

Recommendation These errors could be make more useful by adding some parameters to them, such as the received and the desired amounts.

```
error V2TooLittleReceived();
error V2TooMuchRequested();
```

CVF-4. INFO

- **Category** Bad datatype
- **Source** V2SwapRouter.sol

Recommendation The type of the “v2Factory” argument should be “IUniswapV2Factory”.

```
17 constructor(address v2Factory, bytes32 pairInitCodeHash, address
   ↪ permit2) Permit2Payments(permit2) {
```

CVF-5. INFO

- **Category** Bad datatype
- **Source** V2SwapRouter.sol

Recommendation The type of the “path” argument should be “IERC20[]”.

```
22 function _v2Swap(address[] memory path, address recipient) private {
```

```
46 function v2SwapExactInput(uint256 amountOutMin, address[] memory
   ↪ path, address recipient) internal {
```

```
58     address[] memory path,
```

CVF-8. FIXED

- **Category** Suboptimal
- **Source** V2SwapRouter.sol

Description The expression “path.length - 1” is calculated on every loop iteration.

Recommendation Consider calculating once and reusing.

```
27 for (uint256 i; i < path.length - 1; i++) {
```

CVF-9. INFO

- **Category** Readability
- **Source** V2SwapRouter.sol

Description The “i” variable is implicitly initialized with zero value.

Recommendation Consider initializing explicitly for readability.

```solidity
for (uint256 i; i < path.length - 1; i++) {
```

CVF-10. FIXED

- **Category** Procedural
- **Source** V2SwapRouter.sol

Description The expression “path.length - 2” is calculated on every loop iteration.

Recommendation Consider calculating once before the loop.

```solidity
(nextPair, token0) = i < path.length - 2
```

CVF-11. INFO

- **Category** Unclear behavior
- **Source** V2SwapRouter.sol

Description These lines will throw in case “path” is empty.

Recommendation Consider reverting with a meaningful error in such a case.

```solidity
uint256 balanceBefore = IERC20(path[path.length - 1]).balanceOf(
  ↪ recipient);
```

```solidity
payOrPermit2Transfer(path[0], payer, pair, amountIn);
```

CVF-12. FIXED

- **Category** Procedural
- **Source** V2SwapRouter.sol

Description The expression “path[path.length - 1]” is calculated twice.

Recommendation Consider calculating once and reusing.

```
uint256 balanceBefore = IERC20(path[path.length - 1]).balanceOf(
   ↪ recipient);
```

```
uint256 amountOut = IERC20(path[path.length - 1]).balanceOf(
   ↪ recipient) - balanceBefore;
```

CVF-13. INFO

- **Category** Procedural
- **Source** UniswapV2Library.sol

Description Specifying an unbounded compiler versions range is a bad practice as one cannot guarantee compatibility with future major versions.

Recommendation Consider specifying as “^0.8.0”.

```
pragma solidity >=0.5.0;
```

CVF-14. INFO

- **Category** Bad datatype
- **Source** UniswapV2Library.sol

Recommendation The type of the "factory" argument should be "IUniswapV2Factory", the type of the "tokenA" and "tokenB" arguments should be "IERC20".

```
function pairFor(address factory, bytes32 initCodeHash, address
    ↪ tokenA, address tokenB)
```

```
function pairAndToken0For(address factory, bytes32 initCodeHash,
    ↪ address tokenA, address tokenB)
```

```
function pairForPreSorted(address factory, bytes32 initCodeHash,
    ↪ address token0, address token1)
```

```
function pairAndReservesFor(address factory, bytes32 initCodeHash,
    ↪ address tokenA, address tokenB)
```

CVF-15. INFO

- **Category** Bad datatype
- **Source** UniswapV2Library.sol

Recommendation The return type should be "IUniswapV2Pair".

```
returns (address pair)
```

```
returns (address pair)
```

CVF-16. INFO

- **Category** Bad datatype
- **Source** UniswapV2Library.sol

Recommendation The types of the returned values should bbe "IUniswapV2Pair" and "IERC20" respectively.

```
returns (address pair, address token0)
```

CVF-17. INFO

- **Category** Bad datatype
- **Source** UniswapV2Library.sol

Recommendation The type of the “pair” returned value should be “IUniswapV2Pair”.

```
returns (address pair, uint256 reserveA, uint256 reserveB)
```

CVF-18. INFO

- **Category** Bad datatype
- **Source** UniswapV2Library.sol

Recommendation The values 997 and 1000 should be named constants.

```
uint256 amountInWithFee = amountIn * 997;
```

```
uint256 denominator = reserveIn * 1000 + amountInWithFee;
```

```
uint256 numerator = reserveIn * amountOut * 1000;
uint256 denominator = (reserveOut - amountOut) * 997;
```

CVF-19. INFO

- **Category** Unclear behavior
- **Source** UniswapV2Library.sol

Description This line seems to simulate rounding up by just incrementing the result.

Recommendation Consider doing proper rounding up.

```
amountIn = (numerator / denominator) + 1;
```

CVF-20. INFO

- **Category** Bad datatype
- **Source** UniswapV2Library.sol

Recommendation The type of the “factory” argument should be “IUniswapV2Factory”. The type of the “path” argument should be “IERC20[]”.

```
78 function getAmountInMultihop(address factory, bytes32 initCodeHash,
     ↪ uint256 amountOut, address[] memory path)
```

CVF-21. INFO

- **Category** Bad datatype
- **Source** UniswapV2Library.sol

Recommendation The type of the “pair” returned value should be “IUniswapV2Pair”.

```
81 returns (uint256 amount, address pair)
```

CVF-22. INFO

- **Category** Bad naming
- **Source** UniswapV2Library.sol

Description The semantics of the returned values is unclear.

Recommendation Consider giving the returned values more specific names such as “amountIn” and “firstPair” or describing them in the documentation comment.

```
81 returns (uint256 amount, address pair)
```

CVF-23. INFO

- **Category** Suboptimal
- **Source** Payments.sol

Recommendation These errors could be made more useful by adding some parameters to them, such as the token address, the desired token or ether balance, or the bips passed.

```
error InsufficientToken();
error InsufficientETH();
error InvalidBips();
```

CVF-24. INFO

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

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

```
function pay(address token, address recipient, uint256 value)
  ↪ internal {
```

```
function payPortion(address token, address recipient, uint256 bips)
  ↪ internal {
```

```
function sweep(address token, address recipient, uint256
  ↪ amountMinimum) internal {
```

CVF-26. INFO

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

Description Reverting in case “bips” is zero looks odd.

Recommendation Consider doing nothing in such a case.

```
if (bips == 0 || bips > 10_000) revert InvalidBips();
```

CVF-27. INFO

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

Recommendation This constant must be named.

```
if (bips == 0 || bips > 10_000) revert InvalidBips();
```

CVF-28. INFO

- **Category** Procedural
- **Source** Payments.sol

Recommendation Brackets are redundant.

```
uint256 amount = (balance * bips) / FEE_BIPS_BASE;
```

```
uint256 amount = (balance * bips) / FEE_BIPS_BASE;
```

CVF-29. INFO

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

Recommendation The type of the “token” argument should be “IERC721”.

```
function sweepERC721(address token, address recipient, uint256 id)
    ↪ internal {
```

CVF-30. INFO

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

Recommendation The type of the “token” argument should be “IERC1155”.

```
function sweepERC1155(address token, address recipient, uint256 id,
    ↪ uint256 amount) internal {
```

CVF-33. FIXED

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

Recommendation The type of this argument should b be "Permit" or an interface extracted from it.

```
address permit2,
```

CVF-34. INFO

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

Recommendation The types of these arguments could be more specific.

```
address routerRewardsDistributor,
address looksRareRewardsDistributor,
```

CVF-35. FIXED

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

Recommendation The type of this argument should be "IERC20".

```
address looksRareToken,
```

CVF-36. INFO

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

Recommendation The type of this argument should be "IUniswapV2Factory".

```
address v2Factory,
```

CVF-37. FIXED

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

Recommendation The type of this argument should be “IUniswapV3Factory”.

```
22 address v3Factory,
```

CVF-38. INFO

- **Category** Documentation
- **Source** Router.sol

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

```
28 {}
```

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

CVF-39. INFO

- **Category** Suboptimal
- **Source** Router.sol

Description Having a separate arrays for command opcodes and command arguments is inefficient.

Recommendation Consider packing into a single bytes array, and use slices to extract portions of said array without copying.

```
40 function execute(bytes calldata commands, bytes[] calldata inputs)
     ↪ public payable {
```

CVF-41. FIXED

- **Category** Suboptimal
- **Source** Router.sol

Recommendation Conversion to “uint256” is redundant, as compiler would perform it automatically.

```
49 uint256 commandType = uint256(uint8(command & Commands.
     ↪ COMMAND_TYPE_MASK));
```

CVF-42. INFO

- **Category** Suboptimal
- **Source** Commands.sol

Description The list of commands lack flow control facilities that would significantly extend capabilities of the commands language.

Recommendation Consider adding commands or flags to execute command only if the previous command was successful/unsuccessful, commands to jump to jump to another position in the program. Command to succeed only when the current balance in a certain token is in certain range etc.

```
6 library Commands {
```

CVF-43. INFO

- **Category** Suboptimal
- **Source** Commands.sol

Recommendation This list of commands could be replaced by a enum for automatic numbering.

```
uint256 constant PERMIT2_PERMIT = 0x00;
uint256 constant TRANSFER = 0x01;
uint256 constant V3_SWAP_EXACT_IN = 0x02;
uint256 constant V3_SWAP_EXACT_OUT = 0x03;
uint256 constant V2_SWAP_EXACT_IN = 0x04;
uint256 constant V2_SWAP_EXACT_OUT = 0x05;
uint256 constant SEAPORT = 0x06;
uint256 constant WRAP_ETH = 0x07;
uint256 constant UNWRAP_WETH = 0x08;
uint256 constant SWEEP = 0x09;
uint256 constant NFTX = 0x0a;
uint256 constant LOOKS_RARE_721 = 0x0b;
uint256 constant X2Y2_721 = 0x0c;
uint256 constant LOOKS_RARE_1155 = 0x0d;
uint256 constant X2Y2_1155 = 0x0e;
uint256 constant FOUNDATION = 0x0f;
uint256 constant PAY_PORTION = 0x10;
uint256 constant SWEEP_ERC721 = 0x11;
uint256 constant SUDOSWAP = 0x12;
uint256 constant NFT20 = 0x13;
uint256 constant OWNERSHIP_CHECK_721 = 0x14;
uint256 constant OWNERSHIP_CHECK_1155 = 0x15;
uint256 constant CRYPTOPUNKS = 0x16;
uint256 constant PERMIT2_TRANSFER_FROM = 0x17;
uint256 constant PERMIT2_TRANSFER_FROM_BATCH = 0x18;
uint256 constant PERMIT2_PERMIT_BATCH = 0x19;
uint256 constant SWEEP_ERC1155 = 0x1a;
```

CVF-44. INFO

- **Category** Procedural
- **Source** RewardsCollector.sol

Description The compiler cversion requirement specified here is inconsistent with requirements in other files.

Recommendation Consider changing to “^0.8.0” or to “^0.8.17”.

```
pragma solidity ^0.8.15;
```

CVF-45. INFO

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

Recommendation The type of these variables could be more specific.

```
address public immutable routerRewardsDistributor;
address public immutable looksRareRewardsDistributor;
```

CVF-46. INFO

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

Recommendation The type of this variable should be “IERC20”.

```
ERC20 public immutable looksRareToken;
```

CVF-47. INFO

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

Recommendation The argument types could be more specific, in particular, the type of the “_looksRareToken” should be “IERC20”.

```
constructor(address _routerRewardsDistributor, address
    ↪ _looksRareRewardsDistributor, address _looksRareToken) {
```

CVF-49. INFO

- **Category** Suboptimal
- **Source** V3SwapRouter.sol

Recommendation These errors could be made more useful by adding some parameters to them.

```
error InvalidSwap();
error V3TooLittleReceived();
error V3TooMuchRequested();
error V3InvalidAmountOut();
```

CVF-50. INFO

- **Category** Bad datatype
- **Source** V3SwapRouter.sol

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

```
address token0;
address token1;
```

CVF-51. INFO

- **Category** Bad datatype
- **Source** V3SwapRouter.sol

Recommendation The type of this variable should be “IUniswapV3Factory”.

```
address internal immutable V3_FACTORY;
```

CVF-52. INFO

- **Category** Procedural
- **Source** V3SwapRouter.sol

Recommendation UPPER_CASE is normally used for constants, consider using camelCase instead.

```
address internal immutable V3_FACTORY;
bytes32 internal immutable POOL_INIT_CODE_HASH_V3;
```

CVF-53. INFO

- **Category** Bad datatype
- **Source** V3SwapRouter.sol

Recommendation The type of the “v3Factory” argument should be “IUniswapV3Factory”.

```
constructor(address v3Factory, bytes32 poolInitCodeHash) {
```

CVF-55. INFO

- **Category** Documentation
- **Source** V3SwapRouter.sol

Description This comment sounds confusing and doesn’t explain anything.

Recommendation Consider elaborating more.

```
// because exact output swaps are executed in reverse order, in this
   ↪  case tokenOut is actually tokenIn
```

CVF-56. INFO

- **Category** Unclear behavior
- **Source** V3SwapRouter.sol

Description This function should return the actual output amount.

```
function v3SwapExactInput(
```

CVF-58. INFO

- **Category** Unclear behavior
- **Source** V3SwapRouter.sol

Description This function should return the actual input amount consumed.

```
function v3SwapExactOutput(
```

CVF-59. INFO

- **Category** Overflow/Underflow
- **Source** V3SwapRouter.sol

Description Underflow is possible here.

Recommendation Consider using safe conversion.

```
127 ? (uint256(amount0Delta), uint256(-amount1Delta))
    : (uint256(amount1Delta), uint256(-amount0Delta));
```

CVF-61. INFO

- **Category** Procedural
- **Source** V3Path.sol

Description Specifying an unbouded range of compiler versions is a bad practice and one cannot guarantee compatibility with future major versions.

Recommendation Consider specifying as “^0.6.0 || ^0.7.0 || ^0.8.0”.

```
2 pragma solidity >=0.6.0;
```

CVF-62. INFO

- **Category** Suboptimal
- **Source** V3Path.sol

Description This function doesn’t check that “path.length” is valid.

Recommendation Consider adding appropriate checks.

```
35 function numPools(bytes memory path) internal pure returns (uint256)
     ↪ {
```

CVF-64. INFO

- **Category** Bad datatype
- **Source** V3Path.sol

Recommendation The type of the token returned values should be “IERC20”.

```
function decodeFirstPool(bytes memory path) internal pure returns (
   ↪ address tokenA, address tokenB, uint24 fee) {
```

```
function decodeFirstToken(bytes memory path) internal pure returns (
   ↪ address tokenA) {
```

CVF-65. FIXED

- **Category** Suboptimal
- **Source** V3Path.sol

Description Each of these calls perform a separate length check inside.

Recommendation Consider refactoring to perform only a single check before the calls.

```
tokenA = path.toAddress(0);
fee = path.toUint24(ADDR_SIZE);
tokenB = path.toAddress(NEXT_OFFSET);
```

CVF-66. FIXED

- **Category** Suboptimal
- **Source** V3Path.sol

Description This function allocates a new array and copies data into it. Using this function to iterate through a path is inefficient.

Recommendation Consider iterating in place.

```
function skipToken(bytes memory path) internal pure returns (bytes
   ↪ memory) {
```

CVF-68. INFO

- **Category** Suboptimal
- **Source** BytesLib.sol

Recommendation These error could be make more useful by adding some parameters to them.

```
error SliceOverflow();
error SliceOutOfBounds();
error ToAddressOverflow();
error ToAddressOutOfBounds();
error ToUint24Overflow();
error ToUint24OutOfBounds();
```

CVF-70. INFO

- **Category** Procedural
- **Source** BytesLib.sol

Description The expression "_start + _length" is calculated twice.

Recommendation Consider calculating once and reusing.

```
if (_start + _length < _start) revert SliceOverflow();
if (_bytes.length < _start + _length) revert SliceOutOfBounds();
```

CVF-71. FIXED

- **Category** Procedural
- **Source** BytesLib.sol

Description The expression "lengthmod + 0×20 * iszero(lemgthmod)" is calculated twice.

Recommendation Consider calculating once and reusing.

```
let mc := add(add(tempBytes, lengthmod), mul(0x20, iszero(lengthmod)
   ↪ ))
```

```
for { let cc := add(add(add(_bytes, lengthmod), mul(0x20, iszero(
   ↪ lengthmod))), _start) } lt(mc, end) {
```

CVF-72. INFO

- **Category** Suboptimal
- **Source** BytesLib.sol

Recommendation Using the identity precompile to copy a memory range would be cheaper than copying word by word. Se the following answer for code example: https://ethereum.stackexchange.com/a/76766

```
for { let cc := add(add(add(_bytes, lengthmod), mul(0x20, iszero(
    ↪ lengthmod))), _start) } lt(mc, end) {
```

CVF-73. INFO

- **Category** Suboptimal
- **Source** BytesLib.sol

Recommendation Right shift would be more efficient than division.

```
tempAddress := div(mload(add(add(_bytes, 0x20), _start)), 0
    ↪ x1000000000000000000000000)
```

CVF-74. INFO

- **Category** Suboptimal
- **Source** BytesLib.sol

Recommendation This could be simplified as: temp\Address := mload(add(add(_bytes, 0×14), _start)

```
tempAddress := div(mload(add(add(_bytes, 0x20), _start)), 0
    ↪ x1000000000000000000000000)
```

CVF-76. INFO

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

Recommendation The type of this argument should be “Permit2” or an interface derived from it.

```
address permit2,
```

CVF-77. INFO

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

Recommendation The type of this argument should be “IUniswapV2Factory”.

```
address v2Factory,
```

CVF-78. INFO

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

Recommendation The type of this argument should be “IUniswapV3Factory”.

```
address v3Factory,
```

CVF-79. INFO

- **Category** Procedural
- **Source** Dispatcher.sol

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

```
) V2SwapRouter(v2Factory, pairInitCodeHash, permit2) V3SwapRouter(
   ↪ v3Factory, poolInitCodeHash) {}
```

CVF-82. FIXED

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

Description Special amount value “CONTRACT_BALANCE” is not handled for decoded amounts.

Recommendation Consider handling it for all amounts.

```
permit2TransferFrom(token, msg.sender, recipient, amount);
```

```
v3SwapExactInput(recipient, amountIn, amountOutMin, path, payer);
```

```
Payments.sweepERC1155(token, recipient, id, amount);
```

CVF-83. FIXED

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

Description Special recipients “MSG_SENDER” and “ADDRESS_THIS” are not handled.

Recommendation Consider handling them in all cases.

```
45  permit2TransferFrom(token, msg.sender, recipient, amount);
```

```
53  Payments.pay(token, recipient, value);
```

```
57  v2SwapExactInput(amountOutMin, path, recipient);
```

```
62  v2SwapExactOutput(amountOut, amountInMax, path, recipient, payer);
```

```
67  v3SwapExactInput(recipient, amountIn, amountOutMin, path, payer);
```

```
72  v3SwapExactOutput(recipient, amountOut, amountInMax, path, payer);
```

```
98      ICryptoPunksMarket(Constants.CRYPTOPUNKS).transferPunk(recipient, punkId);
```

```
104 Payments.sweep(token, recipient, amountMin);
```

```
107 Payments.sweepERC721(token, recipient, id);
```

```
111 Payments.sweepERC1155(token, recipient, id, amount);
```

```
114 Payments.wrapETH(recipient, amountMin);
```

```
117 Payments.unwrapWETH9(recipient, amountMin);
```

```
120 Payments.payPortion(token, recipient, bips);
```

CVF-84. INFO

- **Category** Suboptimal
- **Source** Dispatcher.sol

Description This should be performed only when recipient != this.

```
if (success) ERC721(token).safeTransferFrom(address(this), recipient
   ↪ , id);
```

```
if (success) ERC1155(token).safeTransferFrom(address(this),
   ↪ recipient, id, amount, new bytes(0));
```

CVF-85. INFO

- **Category** Bad datatype
- **Source** Permit2Payments.sol

Recommendation The type of this variable should be “Permit2” or an interface derived from it.

```
address immutable PERMIT2;
```

CVF-86. INFO

- **Category** Bad datatype
- **Source** Permit2Payments.sol

Recommendation The argument type should be “Permit2” or an interface derived from it.

```
constructor(address permit2) {
```

CVF-87. INFO

- **Category** Bad datatype
- **Source** Permit2Payments.sol

Recommendation The type of the “token” arguments should be “IERC20”.

```
function permit2TransferFrom(address token, address from, address to
    ↪ , uint160 amount) internal {
```

```
function payOrPermit2Transfer(address token, address payer, address
    ↪ recipient, uint256 amount) internal {
```

CVF-89. INFO

- **Category** Procedural
- **Source** UniswapPoolHelper.sol

Description This import is not used.

```
import '@uniswap/v2-core/contracts/interfaces/IUniswapV2Pair.sol';
```

CVF-90. INFO

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

Recommendation The type of the “factory” argument should be “IUniswapV2Factory”.

```
function computePoolAddress(address factory, bytes memory identifier
    ↪ , bytes32 initCodeHash)
```

CVF-91. INFO

- **Category** Suboptimal
- **Source** UniswapPoolHelper.sol

Recommendation The “identifier” argument should be replaced with a pair of token arguments.

```
function computePoolAddress(address factory, bytes memory identifier
    ↪ , bytes32 initCodeHash)
```

CVF-92. INFO

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

Recommendation The return type should be “IUniswapV2Pair”.

```
returns (address pool)
```

CVF-93. INFO

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

Recommendation The type of arguments and returned values should be “IERC20”.

```
function sortTokens(address tokenA, address tokenB) internal pure
    ↪ returns (address token0, address token1) {
```

CVF-94. INFO

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

Recommendation The type of this constant should be “IERC20”.

```
address internal constant ETH = address(0);
```

CVF-95. INFO

- **Category** Suboptimal
- **Source** Constants.sol

Recommendation These constants could be turned into immutable variables by turning this library into an abstract contract.

```
/// @dev The constants below here might vary between chains. They
    ↪ cannot be immutables as this is a library
```

CVF-96. FIXED

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

Recommendation The type of this constant should be “IWETH9”.

```
address internal constant WETH9 = 0
    ↪ xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2;
```

CVF-97. INFO

- **Category** Suboptimal
- **Source** IRouter.sol

Recommendation These errors could be make more usefule by adding some parameters to them.

```
error TransactionDeadlinePassed();
```

```
error LengthMismatch();
```

CVF-98. INFO

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

Recommendation The return types couyld be more specific.

```
function routerRewardsDistributor() external returns (address);
function looksRareRewardsDistributor() external returns (address);
```

CVF-99. INFO

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

Recommendation The return type should be “IERC20”.

```
function looksRareToken() external returns (ERC20);
```

CVF-100. INFO

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

Description This function should return the rewards amount claimed.

```
10 function collectRewards(bytes calldata) external;
```

CVF-101. INFO

- **Category** Procedural
- **Source** ICryptoPunksMarket.sol

Description This version requirement is inconsistent with version requirements in other files.

Recommendation Consider changing to “^0.8.0” or to “^0.8.17” for consistency.

```
2 pragma solidity ^0.8.4;
```

CVF-102. INFO

- **Category** Documentation
- **Source** IWETH9.sol

Description The argument is not documented.

Recommendation Consider documenting.

```
12 function withdraw(uint256) external;
```

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