VITAL BLOCK

Security Assessment

zkSwap Finance

Vital Block Verified on June 29h, 2023

@Vital-Block

@VB_Audit

info@vitalblock.org

www.vitalblock.org

PREPARED FOR:

zkSwap Finance

INTRODUCTION

Auditing Company	**VITAL BLOCK SECURITY**
Client Project	**ZKSWAP FINANCE**
Methodology	Automated Analysis, Manual Code Review
Language	Solidity
License	MIT
Contracts Address	ZFFactory:0x3a76e377ED58c8731F9DF3A36155942438744Ce3 ZFRouter:0x18381c0f738146Fb694DE18D1106BdE2BE040Fa4
Network	**ARBITRUM CHAIN**
Compiler Version	0.8.16
Zksolc Version	v1.3.8
Website	https://zkswap.finance/
Telegram	https://t.me/zkSwap_Announcement
Twitter	https://twitter.com/zkSwap_finance
Discord	https://discord.gg/4eHMumaJDA
Doc	https://zkswapfinance.gitbook.io/zkswap/
Prelim Report Date	June 28TH 2023
Final Report Date	June 29th 2023

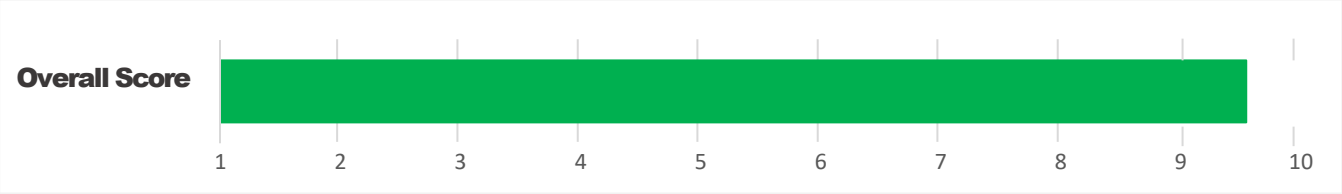
Verify the authenticity of this report on our GitHub Repo: https://www.github.com/vital-block

EXECUTIVE SUMMARY

Vital Block has performed the automated and manual analysis of the ZKSWAP FINANCE Sol code. The code was reviewed for common contract vulnerabilities and centralized exploits. Here's a quick audit summary:

Status	Critical !	Major "	Medium #	Minor $	Unknown %
Open	0	0	0	2	0
Acknowledged	0	0	2	3	1
Resolved	0	0	0	0	0
Noteworty OnlyOwner Privileges	Set Taxes and Ratios, Airdrop, Set Protection Settings, Set Reward Properties, Set Reflector Settings, Set Swap Settings, Set Pair and Router				

ZKSWAP FINANCE Smart contract has achieved the following score: 95.0



Please note that smart contracts deployed on blockchains aren't resistant to exploits, vulnerabilities and/or hacks. Blockchain and cryptography assets utilize new and emerging technologies. These technologies present a high level of ongoing risks. For a detailed understanding of risk severity, source code vulnerability, and audit limitations, kindly review the audit report thoroughly.

Please note that centralization privileges regardless of their inherited risk status - constitute an elevated impact on smart contract safety and security.

SCOPE OF WORK

Vital Block was consulted by ZKSWAP FINANCE to conduct the smart contract audit of its .Sol source code. The audit scope of work is strictly limited to mentioned .SOL file only:

O ZFFactory.Sol
O ZFRouter.Sol

External contracts and/or interfaces dependencies are not checked due to being out of scope.

Verify audited contract's contract address and deployed link below:

Contract Address:	
0x3a76e377ED58c8731F9DF3A36155942438744Ce3	
0x18381c0f738146Fb694DE18D1106BdE2BE040Fa4	
Contract Code: ZFFactory.Sol ZFRouter.Sol	
Project Name	**ZKSWAP FINANCE**
Blockchain	**Abitrum Network**

AUDIT METHODOLOGY

Smart contract audits are conducted using a set of standards and procedures. Mutual collaboration is essential to performing an effective smart contract audit. Here's a brief overview of Vital Block auditing process and methodology:

CONNECT

- The onboarding team gathers source codes, and specifications to make sure we understand the size, and scope of the smart contract audit.

AUDIT

- Automated analysis is performed to identify common contract vulnerabilities. We may use the following third-party frameworks and dependencies to perform the automated analysis:
  - Remix IDE Developer Tool
  - Open Zeppelin Code Analyzer
  - SWC Vulnerabilities Registry
  - DEX Dependencies, e.g., Pancakeswap, Uniswap
- Simulations are performed to identify centralized exploits causing contract and/or trade locks.
- A manual line-by-line analysis is performed to identify contract issues and centralized privileges. We may inspect below mentioned common contract vulnerabilities, and centralized exploits:

Centralized Exploits	○ Token Supply Manipulation ○ Access Control and Authorization ○ Assets Manipulation ○ Ownership Control ○ Liquidity Access ○ Stop and Pause Trading ○ Ownable Library Verification

Common Contract Vulnerabilities	○ Integer Overflow ○ Lack of Arbitrary limits ○ Incorrect Inheritance Order ○ Typographical Errors ○ Requirement Violation ○ Gas Optimization ○ Coding Style Violations ○ Re-entrancy ○ Third-Party Dependencies ○ Potential Sandwich Attacks ○ Irrelevant Codes ○ Divide before multiply ○ Conformance to Solidity Naming Guides ○ Compiler Specific Warnings ○ Language Specific Warnings

REPORT

- The auditing team provides a preliminary report specifying all the checks which have been performed and the findings thereof.
- The client's development team reviews the report and makes amendments to the codes.
- The auditing team provides the final comprehensive report with open and unresolved issues.

PUBLISH

- The client may use the audit report internally or disclose it publicly.

ℹ It is important to note that there is no pass or fail in the audit, it is recommended to view the audit

as an unbiased assessment of the safety of solidity codes.

RISK CATEGORIES

Smart contracts are generally designed to hold, approve, and transfer tokens. This makes them very tempting attack targets. A successful external attack may allow the external attacker to directly exploit. A successful centralization-related exploit may allow the privileged role to directly exploit. All risks which are identified in the audit report are categorized here for the reader to review:

Risk Type	Definition
Critical ! 🔴	These risks could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
Major " 🟠	These risks are hard to exploit but very important to fix, they carry an elevated risk of smart contract manipulation, which can lead to high-risk severity.
Medium # 🟡	These risks should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution. Low-risk re-entrancy-related vulnerabilities should be fixed to deter exploits.
Minor $ 🟢	These risks do not pose a considerable risk to the contract or those who interact with it. They are code-style violations and deviations from standard practices. They should be highlighted and fixed nonetheless.
Unknown % 🟤	These risks pose uncertain severity to the contract or those who interact with it. They should be fixed immediately to mitigate the risk uncertainty.

All statuses which are identified in the audit report are categorized here for the reader to review:

Status Type	Definition
Open	Risks are open.
Acknowledged	Risks are acknowledged, but not fixed.
Resolved	Risks are acknowledged and fixed.

CENTRALIZED PRIVILEGES

Centralization risk is the most common cause of cryptography asset loss. When a smart contract has a privileged role, the risk related to centralization is elevated.

There are some well-intended reasons have privileged roles, such as:

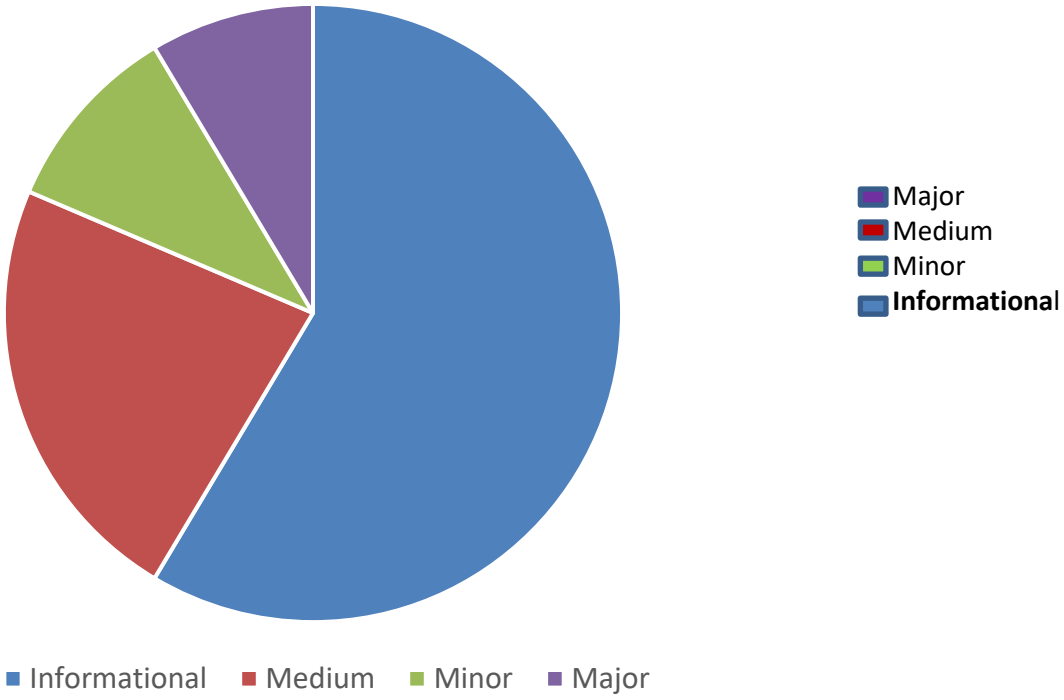
- **Privileged roles can be granted the power to** pause() **the contract in case of an external attack.**
- **Privileged roles can use functions like,** include(), **and** exclude() **to add or remove wallets from fees, swap checks, and transaction limits. This is useful to run a presale and to list on an exchange.**

Authorizing privileged roles to externally-owned-account (EOA) is dangerous. Lately, centralization-related losses are increasing in frequency and magnitude.

- **The client can lower centralization-related risks by implementing below mentioned practices:**
- **Privileged role's private key must be carefully secured to avoid any potential hack.**
- **Privileged role should be shared by multi-signature (multi-sig) wallets.**
- **Authorized privilege can be locked in a contract, user voting, or community DAO can be introduced to unlock the privilege.**
- **Renouncing the contract ownership, and privileged roles.**
- **Remove functions with elevated centralization risk.**

Understand the project's initial asset distribution. Assets in the liquidity pair should be locked. Assets outside the liquidity pair should be locked with a release schedule.

Finding Summary



Status Icon Definitions

✅	Resolved	🚧	In Progress	ℹ️	Ignored (pro)
❌	Not Resolved	❓	Incorrect	🚫	Ignored (con)

Contract Ownership

0x13BD7a61b46950fF0e9b41571Dc4C503eE854042 Is The Owner Of The Contracts.

Summary

- **Owner is not able to change or set taxes**
- **Owner is not able to set a max amount for buys/sells/transfer**
- **Owner is not able to pause trades**
- **Owner is not able to mint new tokens**
- **Owner is not able to blacklist an arbitrary address**

Issues Found

Vital Block Security found that the **ZKSWAP FINANCE** contracts contain no critical issue, no major issues, and 3 minor issue, in addition to 4 informational notes.

We recommend all issues are amended, while the notes are up to the team's discretion, as it refers to best practices.

IRouter01	Interface			
└	factory	External ❗		|NO❗
└	ETH	External ❗		|NO❗
└	addLiquidityETH	External ❗		# |NO❗
└	addLiquidity	External ❗	"	|NO❗
└	swapExactAPTForTokens	External ❗		# |NO❗
└	getAmountsOut	External ❗		|NO❗
└	getAmountsIn	External ❗		|NO❗
IRouter02	Interface	IRouter01		
└	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	"	|NO❗
└	swapExactETHForTokensSupportingFeeOnTransferTokens	External ❗		# |NO❗
└	swapExactTokensForTokensSupportingFeeOnTransferTokens	External ❗	" ! 🛑 |NO❗
└	swapExactTokensForTokens	External ❗	"	|NO❗
Protections	Interface			
└	checkUser	External ❗	" ! 🛑 |NO❗
└	setLaunch	External ❗	" |NO❗
└	setLpPair	External ❗	" |NO❗
└	**ZFRouter**	External ❗	" |NO❗
└	removeSniper	External ❗	" |NO❗
Cashier	Interface			
└	setRewardsProperties	External ❗	"	|NO❗
└	tally	External ❗	" |NO❗
└	load	External ❗		# |NO❗
└	cashout	External ❗	"	|NO❗
└	giveMeWelfarePlease	External ❗	"	|NO❗
└	getTotalDistributed	External ❗		|NO❗
└	getUserInfo	External ❗		|NO❗
└	getUserRealizedRewards	External ❗		|NO❗

| └ | getPendingRewards | External ❗ | | |NO❗ |

| └ | initialize | External ❗ | " | |NO❗ |

| └ | getCurrentReward | External ❗ | | |NO❗ |

||||||

| **SOL** | Implementation | SafeMath |||

| └ | <Constructor> | Public ❗ | | # |NO❗ |

| └ | transferOwner | External ❗ | " | | onlyOwner |

| └ | renounceOwnership | External ❗ | " | | NO❗|

| └ | setOperator | Public ❗ | " | |NO❗ |

| └ | renounceOriginalDeployer | External ❗ | " | |NO❗ |

| └ | <Receive Ether> | External ❗ | | # |NO❗ |

| └ | totalSupply | External ❗ | | |NO❗ |

| └ | decimals | External ❗ | | |NO❗ |

| └ | symbol | External ❗ | | |NO❗ |

| └ | name | External ❗ | | |NO❗ |

| └ | getOwner | External ❗ | | ❗ |NO❗ |

| └ | balanceOf | Public ❗ | | ❗ |NO❗ |

| └ | allowance | External ❗ | | ❗ |NO❗ |

| └ | approve | External ❗ | " | ❗ 🛑 |NO❗ |

| └ | _approve | Internal $ | " | | |

| └ | approveContractContingency | Public ❗ | " | ❗ 🛑 | onlyOwner |

| └ | transfer | External ❗ | " | |NO❗ |

| └ | transferFrom | External ❗ | " | |NO❗ |

| └ | setNewRouter | External ❗ | " | | onlyOwner |

| └ | setLpPair | External ❗ | " | | onlyOwner |

| └ | setInitializers | External ❗ | " | | onlyOwner |

| └ | isExcludedFromFees | External ❗ | | |NO❗ |

| └ | isExcludedFromDividends | External ❗ | | |NO❗ |

| └ | isExcludedFromProtection | External ❗ | | |NO❗ |

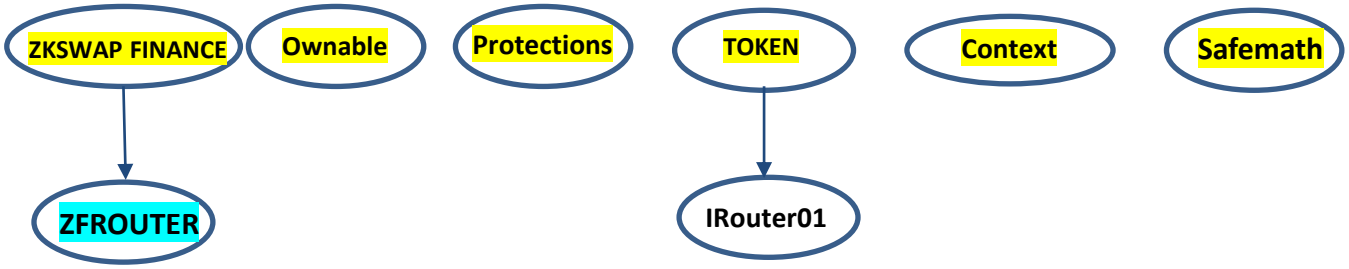
| └ | setDividendExcluded | Public ❗ | " | | onlyOwner |

| └ | setExcludedFromFees | Public ❗ | " | | onlyOwner |

AUDIT SCOPE | ZKSWAP.FINANCE

ID	Repo	Comment	File	SHM211 Checksum
FTM	contracts/ZkSwapFactory.sol	cC51D65	Strings.sol	85f15802c6be0fd50f8632d8433cccc9db6f4b39f9e566d1fa78de54b84bddr54
FRY	contracts/ZkSwapFactory.sol	cC51D53	MetadataHelper.sol	8oippkjjjk96be0fd50f8632d8433cccc9db6f4b39f9e566d1yhhg8765fffckiuybb
FTV	contracts/ZkSwapFactory.sol	cC51D61	ERC20WithPermit.sol	3666778uj908766362fvyga98jdkl88648yhfbqt37409owehbgwhuyyyg223738
FML	contracts/ZkSwapFactory.sol	cC51D76	ReentrancyGuard.sol	98uuyriy399787390uhbiiuhghhdg7guu30oi7799u9359ydfgdgygeigi3ioueyy78
FTR	contracts/ZkSwapFactory.sol	cC51D22	ECDSA.sol	4566efgywqutfeuh87872t15378837983639293763hhegetgjfwjk89336668862
FOP	contracts/ZkSwapFactory.sol	cC51D44	IUniswapV2Callee.sol	546363ttebnve88329973mvvdsggct478153ytgdfdxy792635fgdjgi1900990908
FDP	contracts/ZkSwapFactory.sol	cC51D21	IERC20Permit.sol	835656990327hudbinnjntr6729dchjld0993ytyy3vq63235727879889073
FWY	contracts/ZkSwapFactory.sol	cC51D97	ZFPair.so	cc089692343d1cc36eaf196046d7a528d153abd55ba20e82f1d57c22fcd92675
FKB	contracts/ZkSwapRouter.sol	cC51D76	Math.sol	8448b3af42497f5f74e53424ee3e6c551f51356945108d22a893d608a7990542
FXY	contracts/ZkSwapRouter.sol	cC51D23	IUniswapV2Factory.sol	5c86aa1dd3889db5fcd17a80214b226fc784f268ab9db82df97c1d2459467831
FCB	contracts/ZkSwapRouter.sol	cC51D63	IZFPair.sol	b8244da33db171e5533d77bef4a35703df1de2cebea5f35cb38ce6a26c778cf1
FWO	contracts/ZkSwapRouter.sol	cC51D60	IWETH.sol	3d408b8f2cc56f9699a402b5151de90671de089c3007afc9e4fc867c04152e7c
FGT	contracts/ZkSwapRouter.sol	cC51D54	IERC20Permit.sol	9d751621c3501102e4b50005ca3314ec6e04e6ff8bbb30852d1c7edfff3f8cef
FDF	contracts/ZkSwapRouter.sol	cC51D78	ZFRouterInternal.sol	455687gfesadjknlppiuhhg774580vgfxrki9876dhgvb990lkjhde444566788
FTY	contracts/ZkSwapRouter.sol	cC51D94	IERC20Metadata.sol	566HFFertyuijdsfggtyyyhkhgdrstgioprdetyuuiyyt446789ysghn

INHERITANCE GRAPH



Identifier	Definition	Severity
CEN-12	Centralization privileges of ZKROUTER	Medium # 🟡

Vulnerability 0 : No important security issue detected.
Threat level: Low

```
Home  ZFRouter.sol

268
269     function swapExactTokensForTokens(
270         uint amountIn,
271         uint amountOutMin,
272         address[] calldata path,
273         address to,
274         uint deadline
275     ) external override ensureNotExpired(deadline) returns (uint[] memory amounts) {
276         amounts = ZFLibrary.getAmountsOutUnchecked(factory, amountIn, path); // will fail below if path is invalid
277         // make sure the final output amount not smaller than the minimum
278         require(amounts[amounts.length - 1] >= amountOutMin, 'INSUFFICIENT_OUTPUT_AMOUNT');
279
280         address tokenIn = path[0];
281         address initialPair = ZFLibrary.pairFor(factory, tokenIn, path[1]);
282         TransferHelper.safeTransferFrom(tokenIn, msg.sender, initialPair, amounts[0]);
283         _swapCached(factory, initialPair, amounts, path, to);
284     }
285
286     function swapExactETHForTokens(
287         uint amountOutMin,
288         address[] calldata path,
289         address to,
290         uint deadline
291     ) external override payable ensureNotExpired(deadline) returns (uint[] memory amounts) {
```

FTV-02 POSSIBLE OVERFLOW

Category	Severity	Location	Status
Status Mathematical Operations	**Minor**	contracts/ZkSwapRouter.sol	Acknowledged

Description

In **updateForLiqudity** , the following equation is used inside an unchecked block

```
  ) external override payable ensureNotExpired(deadline) returns (uint
amountTokenInActual, uint amountETHInActual, uint liquidity) {
        address pair;
        (pair, amountTokenInActual, amountETHInActual) = _addLiquidity(token, WETH,
amountTokenInExpected, msg.value, amountTokenInMin, amountETHInMin);
```

Where parameters. **Liquidity** less Used is a this and lessride In is a this.
As these two are multiplied together in an unchecked block, they may overflow.

Recommendation

We recommend either checking for overflow in this case, or ensuring that the PairsIn is close enough it will never causean overflow

FZT-03 POSSIBLE OVERFLOW

Category	Severity	Location	Status
Status Mathematical Operations	**Minor**	contracts/ZkSwapRouter.sol	Acknowledged

Description

In **updateForMinter** , the following equation is used inside an unchecked block

```
            liquidity = IZFPair(pair).mint(to);

            if (msg.value > amountETHInActual) {
          TransferHelper.safeTransferETH(msg.sender, msg.value - amountETHInActual);
        }
         } public fun mint_allowed<CoinType>(minter: address, _mint_amount: u64): u64 {
assert!(!storage::mint_guardian_paused<CoinType>(), EMINT_PAUSED);
```

Minter can not issue more **ZKSWAP** tokens indefinitely.
Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the **ZFROUTER** contract.

Recommendation

We recommend either checking for overflow in this case, or ensuring that the PairsIn is close enough it will never cause an overflow.

TZT-02 POSSIBLE OVERFLOW

Category	Severity	Location	Status
Status Mathematical Operations	**Minor**	contracts/code/zffactory.sol	Acknowledged

Description

There seems to be no way to disable a particular fee by setting spacing to 0. Probably not an issue.

```
function _swapSupportingFeeOnTransferTokens(address initialPair, address[] calldata
path, address _to) internal virtual {
        for (uint i; i < path.length - 1; ) {
            (address input, address output) = (path[i], path[i + 1]);
```

Recommendation

This should be a named constant being equal to 1e6, which occurs in other contracts.

TZT-02 POSSIBLE OVERFLOW

Category	Severity	Location	Status
Bad datatype	**Minor**	contracts/code/ZKSWAPFINANCE.sol	Acknowledged

Description

The name **mapping** is confusing. It would be fine for a getter function but not for a property.

```
mapping(address => mapping(address => bool)) public override isPairIndexed;
    mapping(address => address[]) public override indexedPairs;

    function indexedPairsOf(address account) external view override returns (address[]
memory) {
        return indexedPairs[account];
    }
```

Recommendation

Consider renaming to just "pool" or to something like "poolByTokenPair"

TTV-03 POSSIBLE OVERFLOW

Category	Severity	Location	Status
Bad naming	**Minor**	Contract/ZkSwapFactory.sol	INFORMATIONAL

Description

State **Token** variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
internal virtual returns (address pair, uint amountAInActual, uint amountBInActual) {
        address _factory = factory;
        pair = ZFLibrary.pairFor(_factory, tokenA, tokenB);
        if (pair == address(0)) {
```

Recommendation

Constant state **Token** variables can be useful when the contract wants to ensure that the value of a state Token variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

Vulnerability Scan

REENTRANCY

Severity	Major
Confidence Parameter	Certain

Vulnerability Description

NOTE: In a re-entrance attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls.

Scanning Line:

```
    }

    function indexedPairsRange(address account, uint256 start, uint256 counts) external view override returns (address[] memory) {
        require(counts != 0, "Counts must greater than zero");

        address[] memory pairs = indexedPairs[account];
        require(start + counts <= pairs.length, "Out of bound");

        address[] memory result = new address[](counts);
        for (uint256 i = 0; i < counts; i++) {
            result[i] = pairs[start + i];
        }
        return result;
    }
```

General Detectors

Incorrect Solidity Version
This contract uses an unconventional or very old version of Solidity.

Attention Required

Public Functions Should be Declared External
Some functions in this contract should be declared as external in order to save gas.

Attention Required

State Variables Should be Declared Constant
Some state variables in this contract should be declared as constant

Attention Required

- No vulnerable withdrawal functions found
- No reentrancy risk found
- No locks detected
- Verified source code found
- No mintable risks found
- Users can always transfer their tokens
- Contract cannot be upgraded
- Wallets cannot be blacklisted from transfering the token
- No transfer fees found
- Token can be sold through regular AMMs
- No transfer limits found
- No ERC20 approval vulnerability found
- Contract owner cannot abuse ERC20 approvals
- No ERC20 interface errors found
- No blocking loops found
- No centralized balance controls found
- No transfer cooldown times found
- No approval restrictions found
- No external calls detected
- No dumping risks found
- No compiler version inconsistencies found
- No unchecked call responses found
- No vulnerable self-destruct functions found
- No assertion vulnerabilities found
- No old solidity code found
- No external delegated calls found
- No external call dependency found
- No vulnerable authentication calls found
- No invalid character typos found
- No RTL characters found
- No dead code found
- No risky data allocation found
- No uninitialized state variables found
- No uninitialized storage variables found
- No vulnerable initialization functions found
- No risky data handling found
- No number accuracy bug found
- No out-of-range number vulnerability found

OPTIMIZATIONS | TIDE EXCHANGE

ID	Title	Category	Status
GZT- 007	Logarithm Refinement Optimization	Gas Optimization	Acknowledged
GZT- 323	Checks Can Be Performed Earlier	Gas Optimization	Acknowledged
GZT- 679	Unnecessary Use Of SafeMath	Gas Optimization	Acknowledged
GZT- 122	Struct Optimization	Gas Optimization	Acknowledged
GZT-067	Unused State Variable	Gas Optimization	Acknowledged

Repository:

https://github.com/ZKSwapfinance/

All Audited Files

ZKROUTER.SOL
ZKFACTORY.SOL

Contracts:

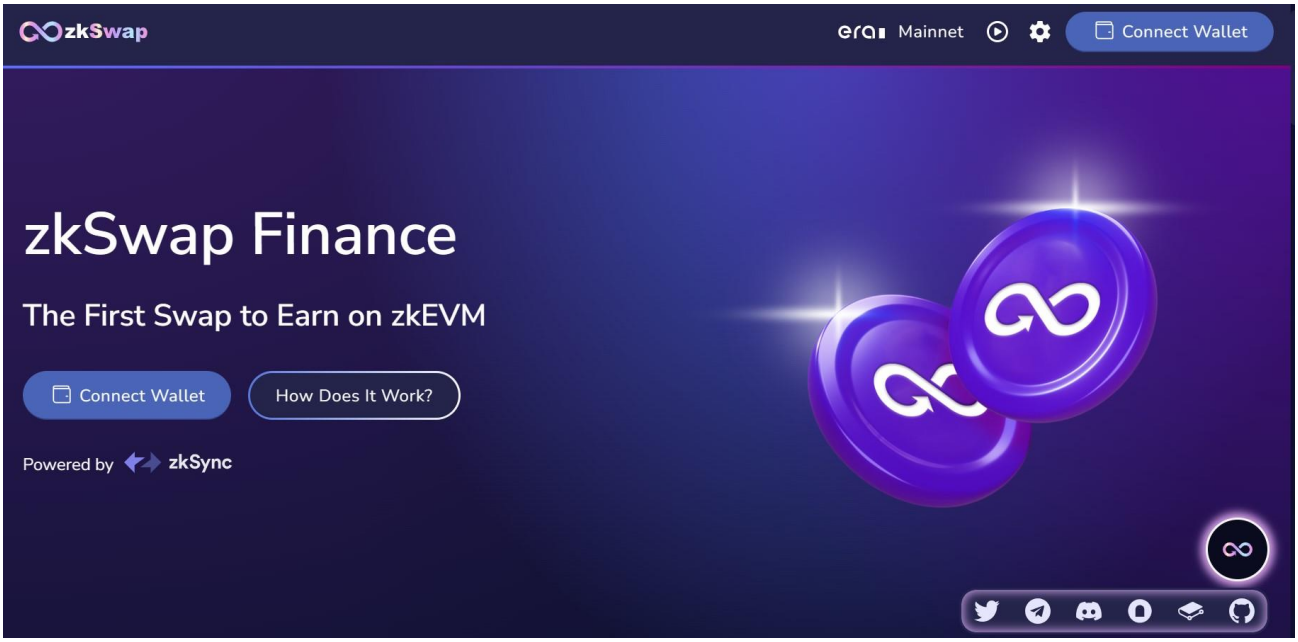
Contract:
Router:: 0x18381c0f738146Fb694DE18D1106BdE2BE040Fa4
factory:: 0x3a76e377ED58c8731F9DF3A36155942438744Ce3

MANUAL REVIEW

ZKSWAP.FINANCE: is the first decentralized Swap-to-Earn DeFi DEX that pioneers a unique incentive model rewarding both liquidity providers and traders. We also have juicy retroactive rewards for early supporters.

Project: ZKSWAP FINANCE
Chain/Standard: Arbitrum Network

Outstanding Features of ZKSWAP FINANCE Is Launching On Arbitrum Network



ISSUES CHECKING STATUS

	Issue Description	Checking Status
1.	Compiler errors.	PASSED
2.	Race Conditions and reentrancy. Cross-Function Race Conditions.	PASSED
3.	Possible Delay In Data Delivery.	PASSED
4.	Oracle calls.	PASSED
5.	Front Running.	PASSED
6.	Sol. Dependency.	PASSED
7.	Integer Overflow And Underflow.	PASSED
8.	DoS with Revert.	PASSED
9.	Dos With Block Gas Limit.	PASSED
10.	Methods execution permissions.	PASSED
11.	Economy Model of the contract.	PASSED
12.	The Impact Of Exchange Rate On the solidity Logic.	PASSED
13.	Private use data leaks.	PASSED
14.	Malicious Event log.	PASSED
15.	Scoping and Declarations.	PASSED
16.	Uninitialized storage pointers.	PASSED
17.	Arithmetic accuracy.	PASSED
18.	Design Logic.	PASSED
19.	Cross-Function race Conditions	PASSED
20.	Save Upon solidity contract Implementation and Usage.	PASSED
21.	Fallback Function Security	PASSED

AUDIT RESULT

PASSED

Identifier	Definition	Severity
TEN-02	Transfers User's Tokens	Minor

```
address tokenIn = path[0];
        address initialPair = ZFLibrary.pairFor(factory, tokenIn, path[1]);
        TransferHelper.safeTransferFrom(tokenIn, msg.sender, initialPair, amounts[0]);
        _swapCached(factory, initialPair, amounts, path, to);
    }
```

Location: ZFROUTER/IERC20.sol

Alleviation:

Any user has the authority to transfer the balance of a user's address if the user has granted allowance. The contract does not subtract the allowance in the transferFrom() method, as a result, the transfer can be repeated until the user's balance go to zero.

RECOMMENDATION

Deployer and/or contract owner private keys are secured carefully.

Please refer to PAGE-09 CENTRALIZED PRIVILEGES for a detailed understanding.

ALLEVIATION

ZKSWAP FINANCE project team understands the centralization risk. Some functions are provided privileged access to ensure a good runtime behaviour in the project

Identifier	Definition	Severity
TDB-12	Third Party Dependencies	Minor

A smart contract is interacting with third-party protocols e.g., Uniswap, Pancakeswap router, cashier contract,

And protections contract. The scope of the audit treats third-party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and exploited. Moreover, upgrades in third parties can create severe impacts, e.g., increased transactional fees, deprecation of previous routers, etc.

RECOMMENDATION

Inspect and validate third party dependencies regularly, and mitigate severe impacts whenever necessary.

DISCLAIMERS

Vital Block Security provides the easy-to-understand audit of Solidity, Move, and Raw source codes (commonly known as smart contracts).

The smart contract for this particular audit was analyzed for common contract vulnerabilities, and centralization exploits. This audit report makes no statements or warranties on the security of the code. This audit report does not provide any warranty or guarantee regarding the absolute bug-free nature of the smart contract analyzed, nor do they provide any indication of the client's business, business model, or legal compliance. This audit report does not extend to the compiler layer, any other areas beyond the programming language, or other programming aspects that could present security risks. Cryptographic tokens are emergent technologies, they carry high levels of technical risks and uncertainty. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. This audit report could include false positives, false negatives, and other unpredictable results.

CONFIDENTIALITY

This report is subject to the terms and conditions (including without limitations, description of services, confidentiality, disclaimer and limitation of liability) outlined in the scope of the audit provided to the client. This report should not be transmitted, disclosed, referred to, or relied upon by any individual for any purpose without InterFi Network's prior written consent.

NO FINANCIAL ADVICE

This audit report does not indicate the endorsement of any particular project or team, nor guarantees its security. No third party should rely on the reports in any way, including to make any decisions to buy or sell a product, service or any other asset. The information provided in this report does not constitute investment advice, financial advice, trading advice, or any other sort of advice and you should not treat any of the report's content as such. This audit report should not be used in any way

to make decisions around investment or involvement. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort.

FOR AVOIDANCE OF DOUBT, SERVICES, INCLUDING ANY ASSOCIATED AUDIT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

TECHNICAL DISCLAIMER

ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, VITAL BLOCK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO SERVICES, AUDIT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, VITAL BLOCK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM THE COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

WITHOUT LIMITING THE FOREGOING, VITAL BLOCK MAKES NO WARRANTY OF ANY KIND THAT ALL SERVICES, AUDIT REPORTS, SMART CONTRACT AUDITS, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET THE CLIENT'S OR ANY OTHER INDIVIDUAL'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE.

TIMELINESS OF CONTENT

The content contained in this audit report is subject to change without any prior notice. Vital Block does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following the publication.

LINKS TO OTHER WEBSITES

This audit report provides, through hypertext or other computer links, access to websites and social accounts operated by individuals other than Vital Block. Such hyperlinks are provided for your reference and convenience only and are the exclusive responsibility of such websites and social accounts owners. You agree that Vital block Security is not responsible for the content or operation of such websites and social accounts and that Vital Block shall have no liability to you or any other person or entity for the use of third-party websites and social accounts. You are solely responsible for determining the extent to which you may use any content at any other websites and social accounts to which you link from the report.

ABOUT VITAL BLOCK

Vital Block provides intelligent blockchain Security Solutions. We provide solidity and Raw Code Review, testing, and auditing services. We have Partnered with 15+ Crypto Launchpads, audited 50+ smart contracts, and analyzed 200,000+ code lines. We have worked on major public blockchains e.g., Ethereum, Binance, Cronos, Doge, Polygon, Avalanche, Metis, Fantom, Bitcoin Cash, Aptos, Oasis, etc.

Vital Block is Dedicated to Making Defi & Web3 A Safer Place. We are Powered by Security engineers, developers, UI experts, and blockchain enthusiasts. Our team currently consists of 5 core members, and 4+ casual contributors.

Website: https://www.Vitalblock.org

Email: info@vitalblock.org

GitHub: https://github.com/vital-block

Telegram (Engineering): https://t.me/vital_block

Telegram (Onboarding): https://t.me/vitalblock_cmo

vital-block

info@vitalblock.org

www.Vitalblock.org

Vital Block Dedicated to securing Public and Private Blockchain Ecosystem