Competitive Security Assessment

ZKSpace

Jun 8th, 2023

Secure3

secure3.io

Summary

ZKSwap is the first Layer 2 AMM DEX powered by ZK-Rollups, launched in February 2021. It allows users to list any ERC20 token pair, exchange tokens,and offers four fee token options, a 100% buyback and burn program for ZKS,and unique liquidity mining activities. In 2021, ZKSwap achieved tens of billions of dollars in trading volume, with a peak TVL of $2.50 billion.Today, ZKSwap is expanding to ZKSync Era and joining ZKSync's mission to accelerate the mass adoption of crypto for personal sovereignty.

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	ZKSpace
Platform & Language	Solidity
Codebase	• https://github.com/l2labs/zksync-v1-periphery • audit commit - 111f808f0188fdb396e0538b2976539a0329feae • final commit - d2a7041ce00fc0b44f965d8b0dd4a6f22bb266d0 • https://github.com/l2labs/zksync-v1-core • audit commit - 25b537c48cd6b451103a7cef7dc3d48ea308090b • final commit - e06f30ee69431daf7a7133238923b00455c18eb8
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Code Vulnerability Review Summary

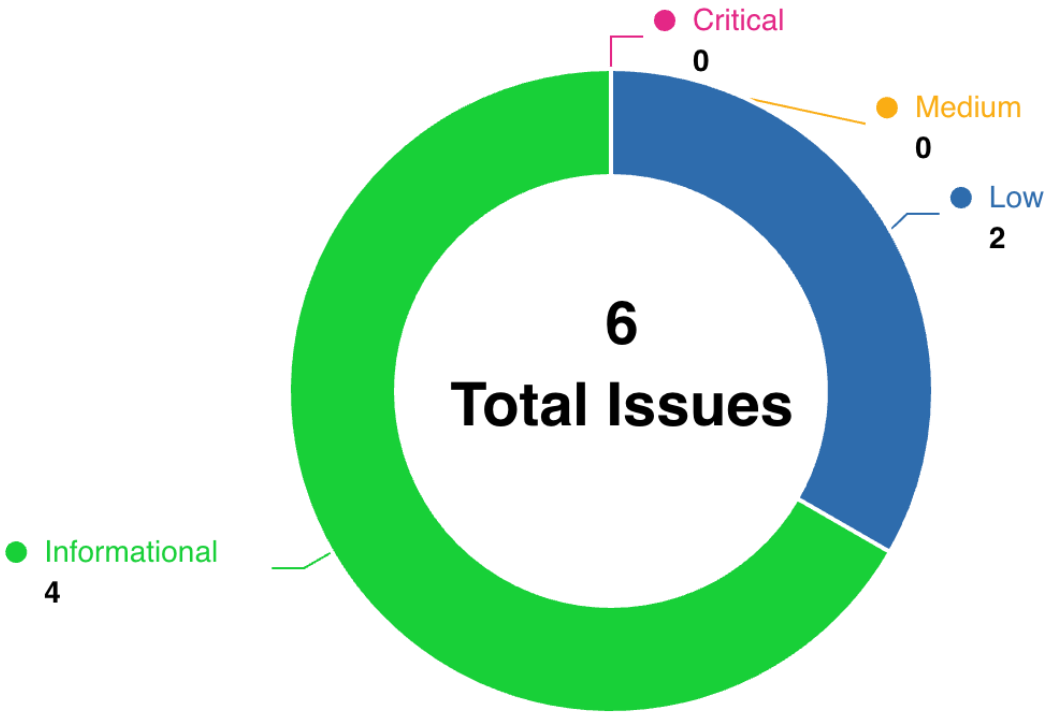
Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	0	0	0	0	0	0
Medium	0	0	0	0	0	0
Low	2	0	1	1	0	0
Informational	4	0	1	3	0	0

Audit Scope

File	Commit Hash
zksync-v1-periphery/contracts/ZksRouter.sol	111f808f0188fdb396e0538b2976539a0329feae
zksync-v1-periphery/contracts/interfaces/IZksRouter.sol	111f808f0188fdb396e0538b2976539a0329feae
zksync-v1-periphery/contracts/libraries/ZksLiquidityMathLibrary.sol	111f808f0188fdb396e0538b2976539a0329feae
zksync-v1-periphery/contracts/libraries/FixedPoint.sol	111f808f0188fdb396e0538b2976539a0329feae
zksync-v1-periphery/contracts/libraries/BitMath.sol	111f808f0188fdb396e0538b2976539a0329feae
zksync-v1-periphery/contracts/libraries/ZksLibrary.sol	111f808f0188fdb396e0538b2976539a0329feae
zksync-v1-periphery/contracts/interfaces/IZksPair.sol	111f808f0188fdb396e0538b2976539a0329feae
zksync-v1-periphery/contracts/libraries/ZksOracleLibrary.sol	111f808f0188fdb396e0538b2976539a0329feae
zksync-v1-periphery/contracts/interfaces/IERC20.sol	111f808f0188fdb396e0538b2976539a0329feae
zksync-v1-periphery/contracts/interfaces/IZksFactory.sol	111f808f0188fdb396e0538b2976539a0329feae
zksync-v1-periphery/contracts/interfaces/IWETH.sol	111f808f0188fdb396e0538b2976539a0329feae
zksync-v1-core/contracts/ZksPair.sol	25b537c48cd6b451103a7cef7dc3d48ea308090b
zksync-v1-core/contracts/ZksERC20.sol	25b537c48cd6b451103a7cef7dc3d48ea308090b
zksync-v1-core/contracts/WETH.sol	25b537c48cd6b451103a7cef7dc3d48ea308090b
zksync-v1-core/contracts/ZksFactory.sol	25b537c48cd6b451103a7cef7dc3d48ea308090b
zksync-v1-core/contracts/interfaces/IZksPair.sol	25b537c48cd6b451103a7cef7dc3d48ea308090b
zksync-v1-core/contracts/interfaces/IZksERC20.sol	25b537c48cd6b451103a7cef7dc3d48ea308090b
zksync-v1-core/contracts/interfaces/IERC20.sol	25b537c48cd6b451103a7cef7dc3d48ea308090b
zksync-v1-core/contracts/interfaces/IZksFactory.sol	25b537c48cd6b451103a7cef7dc3d48ea308090b
zksync-v1-core/contracts/libraries/UQ112x112.sol	25b537c48cd6b451103a7cef7dc3d48ea308090b
zksync-v1-core/contracts/Token.sol	25b537c48cd6b451103a7cef7dc3d48ea308090b
zksync-v1-core/contracts/interfaces/IWETH.sol	25b537c48cd6b451103a7cef7dc3d48ea308090b

zksync-v1-core/contracts/interfaces/IZksCallee.sol	**25b537c48cd6b451103a7cef7dc3d48ea308090b**

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
ZKS-1	EIP-2612 permit is not encouraged to be used in zksync era	Logical	Low	Acknowledged	comcat
ZKS-2	Keep the `sqrt` method of the Math Library the same	Logical	Low	Fixed	comcat
ZKS-3	Gas Optimization:No need to call abi.encodePacked when there's only a single `bytes` argument	Gas Optimization	Informational	Fixed	alansh
ZKS-4	Gas Optimization:Unused variable	Gas Optimization	Informational	Fixed	hunya

ZKS-5	**Missing error message in require statements**	**Code Style**	**Informational**	**Fixed**	**SAir**
ZKS-6	**Unlocked Pragma Version**	**Language Specific**	**Informational**	**Acknowledged**	**hunya**

ZKS-1:EIP-2612 permit is not encouraged to be used in zksync era

Category	Severity	Status	Contributor
Logical	Low	Acknowledged	comcat

Code Reference

- code/zksync-v1-core/contracts/ZksERC20.sol#L73

```
73:    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32
r, bytes32 s) external {
```

Description

comcat : the zksPair contracts implement the `Permit` function, as the EIP-2612 requires. However, according to the zksync era docs, (https://era.zksync.io/docs/dev/building-on-zksync/contracts/contracts.html#evm-compatibility), its said like the follows:

ecrecover usage: If you are using 'ecrecover' to validate a signature of a user account, note that zkSync Era comes with native account abstraction support. It is highly recommended not to rely on the fact that the account has an ECDSA private key attached to it, since they may be ruled by a multisig and use another signature scheme. Read more about zkSync Account Abstraction support

recommend to adopt the AA account feature in zksync era. since AA account doesnt has the ECDSA as the default signing algo.

Recommendation

comcat : recommend to adopt the AA account feature in zksync era.

Client Response

Acknowledged.We did not support AA at this version, we will include this in the future release.

ZKS-2:Keep the `sqrt` method of the Math Library the same

Category	Severity	Status	Contributor
Logical	Low	Fixed	comcat

Code Reference

- code/zksync-v1-core/contracts/ZksPair.sol#L10
- code/zksync-v1-periphery/contracts/libraries/ZksLiquidityMathLibrary.sol#L23
- code/zksync-v1-core/contracts/ZksPair.sol#L96

```
10:import '@openzeppelin/contracts/utils/math/Math.sol';

23:        uint256 leftSide = Babylonian.sqrt(

96:                uint rootKLast = Math.sqrt(_kLast);
```

Description

comcat : in side the Pair contract, it use the openzeppelin's sqrt method, which is "Hacker's Delight", however in the Routers 'computeProfitMaximizingTrade' function, it use the Babylonian method to calculate the sqrt

in the openzeppelin, the way it implements sqrt is : (https://github.com/OpenZeppelin/openzeppelin-contracts/blob/ffceb3cd98887436980 6139ae9e59d2e2a93daec/contracts/utils/math/Math.sol#LL218C4-L249C6)

```
 function sqrt(uint256 a) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }

        // For our first guess, we get the biggest power of 2 which is smaller than the square root
of the target.
        //
        // We know that the "msb" (most significant bit) of our target number `a` is a power of 2 su
ch that we have
        // `msb(a) <= a < 2*msb(a)`. This value can be written `msb(a)=2**k` with `k=log2(a)`.
        //
        // This can be rewritten `2**log2(a) <= a < 2**(log2(a) + 1)`
        // → `sqrt(2**k) <= sqrt(a) < sqrt(2**(k+1))`
        // → `2**(k/2) <= sqrt(a) < 2**((k+1)/2) <= 2**(k/2 + 1)`
        //
        // Consequently, `2**(log2(a) / 2)` is a good first approximation of `sqrt(a)` with at least
1 correct bit.
        uint256 result = 1 << (log2(a) >> 1);

        // At this point `result` is an estimation with one bit of precision. We know the true value
is a uint128,
        // since it is the square root of a uint256. Newton's method converges quadratically (precis
ion doubles at
        // every iteration). We thus need at most 7 iteration to turn our partial result with one bi
t of precision
        // into the expected uint128 result.
        unchecked {
            result = (result + a / result) >> 1;
            result = (result + a / result) >> 1;
            result = (result + a / result) >> 1;
            result = (result + a / result) >> 1;
            result = (result + a / result) >> 1;
            result = (result + a / result) >> 1;
            result = (result + a / result) >> 1;
            return min(result, a / result);
        }
    }
```

however for the Babylonian, the way it implement sqrt is (https://github.com/Uniswap/v2-core/blob/ee547b17853e71ed4e0101ccfd52e70d5acded58/contracts/libraries/Math.sol#LL10C5-L22C6)

```
// babylonian method (https://en.wikipedia.org/wiki/Methods_of_computing_square_roots#Babylonian_method)
    function sqrt(uint y) internal pure returns (uint z) {
        if (y > 3) {
            z = y;
            uint x = y / 2 + 1;
            while (x < z) {
                z = x;
                x = (y / x + x) / 2;
            }
        } else if (y != 0) {
            z = 1;
        }
    }
```

Recommendation

comcat : recommend to keep the way to calculate SQRT the same, try to use babylon method as uniswap does.

Client Response

Fixed.Use uniswap version sqrt function

ZKS-3:Gas Optimization:No need to call abi.encodePacked when there's only a single `bytes` argument

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	alansh

Code Reference

- code/zksync-v1-core/contracts/ZksFactory.sol#L8

```
8:    bytes32 public constant INIT_CODE_PAIR_HASH = keccak256(abi.encodePacked(type(ZksPair).creatio
nCode));
```

Description

alansh : `keccak256(abi.encodePacked(type(ZksPair).creationCode))` is the same as `keccak256(type(ZksPair).creationCode)`.

Recommendation

alansh : Replace `keccak256(abi.encodePacked(type(ZksPair).creationCode))` with `keccak256(type(ZksPair).creationCode)` to save some gas.

Client Response

Fixed.Changed as suggested.

ZKS-4:Gas Optimization:Unused variable

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	hunya

Code Reference

- code/zksync-v1-core/contracts/ZksFactory.sol#L8

```
8:    bytes32 public constant INIT_CODE_PAIR_HASH = keccak256(abi.encodePacked(type(ZksPair).creatio
nCode));
```

Description

hunya : Variables not used could be deleted to save gas. Variable `INIT_CODE_PAIR_HASH` not used in `ZksFactory` contract, but used in deploy scripts to do some test or debug.It could be deleted to save gas in mainnet launch.

Recommendation

hunya : Delete constant variable `INIT_CODE_PAIR_HASH` to save gas in mainnet launch.

Client Response

Fixed.Comment the variable

ZKS-5:Missing error message in require statements

Category	Severity	Status	Contributor
Code Style	Informational	Fixed	SAir

Code Reference

- code/zksync-v1-core/contracts/WETH.sol#L62-L76

```
62:     function transferFrom(address src, address dst, uint wad) public returns (bool) {
63:         require(balanceOf[src] >= wad, '');
64:
65:         if (src != msg.sender && allowance[src][msg.sender] != type(uint).max) {
66:             require(allowance[src][msg.sender] >= wad, '');
67:             allowance[src][msg.sender] -= wad;
68:         }
69:
70:         balanceOf[src] -= wad;
71:         balanceOf[dst] += wad;
72:
73:         emit Transfer(src, dst, wad);
74:
75:         return true;
76:     }
```

Description

SAir : In the two require statements of the transferFrom function, an appropriate error message should be provided to better understand the reason for triggering the require.

Recommendation

SAir : Add 'insufficient balance' error message to the first require() statement and 'insufficient number of allowances' error message to the second require() statement.

Consider below fix in the `WETH9.transferFrom()` function

```
function transferFrom(address src, address dst, uint wad) public returns (bool) {
        require(balanceOf[src] >= wad, 'Insufficient balance');

        if (src != msg.sender && allowance[src][msg.sender] != type(uint).max) {
            require(allowance[src][msg.sender] >= wad, 'Insufficient allowance');
            allowance[src][msg.sender] -= wad;
        }

        balanceOf[src] -= wad;
        balanceOf[dst] += wad;

        emit Transfer(src, dst, wad);

        return true;
    }
```

Client Response

Fixed.Add error message.

ZKS-6:Unlocked Pragma Version

Category	Severity	Status	Contributor
Language Specific	Informational	Acknowledged	hunya

Code Reference

- code/zksync-v1-core/contracts/Token.sol#L2
- code/zksync-v1-core/contracts/ZksERC20.sol#L2
- code/zksync-v1-core/contracts/ZksFactory.sol#L2
- code/zksync-v1-core/contracts/ZksPair.sol#L2
- code/zksync-v1-periphery/contracts/ZksRouter.sol#L2
- code/zksync-v1-core/contracts/WETH.sol#L16

```
2:pragma solidity ^0.8.0;

2:pragma solidity ^0.8.0;

2:pragma solidity ^0.8.0;

2:pragma solidity ^0.8.0;

2:pragma solidity ^0.8.0;

16:pragma solidity ^0.8.0;
```

Description

hunya : Most solidity files have a pragma solidity version number with ^0.8.0 . The caret(^) points to unlocked pragma, meaning the compiler will use the specified version or above.

Recommendation

hunya : It's good practice to use specific solidity versions to know compiler bug fixes and optimisations were enabled at the time of compiling the contract.

Client Response

Acknowledged.Not necessary to change.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.