ADASWAP

REPORT ON SECURITY ASSESSMENT OF MILKOMEDA SMART CONTRACTS FOR ADASWAP

zokyo

September 12th, 2022 | v. 1.1

Table of Contents

1. INTRODUCTION

By request of Adaswap (Customer, Company), and according to Purchase Order dated 2 Aug 2022, Zokyo has delivered the professional information security services, namely, security assessment of the Customer's smart contracts (target object).

After reviewing the implementation of Milkomeda-AdaSwap's smart contracts, this audit report has been prepared to discover potential issues and vulnerabilities of their source code. We have outlined our approach to evaluate the potential security risks. Advice on how to improve the security and performance has also been given in the report.

2. WHAT IS A SMART CONTRACT AUDIT

Smart Contracts are the crux of all DApps and Token Sales. Smart Contracts are essentially programs designed to execute automatically and enforce a set of rules autonomously.

Smart Contracts are completely unchangeable once deployed on the blockchain — a quality that makes smart contracts uniquely reliable and trustworthy, but also dangerous objects.

Coding for the blockchain is a relatively new field, without many security standards, documentation, or best practices. It is also the ultimate test of defensive software engineering. Smart contracts can end up controlling tens of millions of dollars, making them a target for attackers.

Audit of Smart Contracts is focused on finding logic flaws and security vulnerabilities, especially, which could let an attacker to misuse the Smart Contract, to violate Customer's business requirements or cause any other harm to the Customer or its clients or partners. The goal of the audit is to model and verify the target object compromise, sensitive information theft, weak conditions or other ways or prerequisites for realization of fraud or security incidents. To achieve this goal, tools and techniques very similar to those that an attacker would use are typically required.

The audit was carried out externally from the auditors' premises. The best practices Solidity Style Guide and Smart Contract Security Best Practices were used. Automated and manual techniques were used to verify and evaluate the security of the target object.

3. AUDIT SUMMARY

According to the assessment, the Customer's Solidity smart contracts are well secured . In general, the code is well commented. Commenting provides extensive documentation of functions, return variables, and so on, and helps auditors quickly understand the flow of code logic.

According to the findings, we recommend eliminating all low-risk findings, additionally reviewing information-level findings as well. This will help to ensure confidentiality, integrity, and availability.

Based on the number of findings, low-risk vulnerabilities could be a potential security threat for the system. Based on the number of information-level findings, this indicates the violation of code best practices.

4. RECOMMENDATIONS

Auditors recommend mitigating all the vulnerabilities described in this report.

The nature of information threats involves the uncertainty of penetration paths that may be used by an attacker. In addition, the set of known technical vulnerabilities in libraries, components, and hosting environments is constantly increasing. Therefore, the results of this audit cannot guarantee uncovering all possible compromise or penetration ways and security problems, and only show the weakest points in the security of the target object.

Besides smart contract audits, to enhance Customer's security effectively and to reduce Customer's business risks, other appropriate security management processes and security solutions should be designed and implemented. These security measures include but are not limited to: secure development lifecycle, regular security audits by an independent party, security event monitoring, incident response.

Using internationally recognized standards and best practices such as ISO 27000, PCI DSS, NIST is recommended. We can help in the implementation of these processes and solutions.

5. METHODOLOGY

To evaluate the potential vulnerabilities or issues, we go through a checklist of well-known smart contract-related security issues, using automatic verification tools and manual review. We test some discovered issues on our private network to reproduce the issue and prove our findings.

In this audit, we considered the following important features of the code.

- Compliance of the code with the requirements of the SWC registry.
- Implementation of protocol standards.
- Whether the code conforms to coding best practices.

Manual audit:

- Manual code analysis for security vulnerabilities.
- Evaluation of the overall structure, complexity, and quality of the project.
- Checking SWC registry errors in the code.
- Analysis of data security in the network.
- Failover analysis to check the operation of the smart contract in case of errors and vulnerabilities.

Automated analysis:

- Scanning the project's codebase with Mythx, Slither , Echidna, Manticore, and others.
- Manual check of all problems found by the tools.
- Manual security testing according to SWC-Registry.

6. PROJECT SCOPE

Adaswap is an automated liquidity protocol based on a constant product formula and implemented on a non-updatable smart contract system on the **Milkomeda** blockchain. The project is based on the **Uniswap V2 Protocol** and consists of core and peripheral smart contracts. **AdaswapFactory, AdaswapPair** and **AdaswapERC20** core contracts provide fundamental security guarantees for all parties interacting with Adaswap.
AdaswapFactory contains the generic bytecode responsible for pairing. Its main task is to create one and only one smart contract for each unique pair of tokens.
AdaswapPair serves as an automated market maker and keeps track of the pool's token balance. They also provide data for creating decentralized price oracles and more.
AdaswapERC20 (Adaswap LP Token) is an implementation of an ERC20 token received by a liquidity provider (LP) for a pool that contributes the equivalent value of each underlying token in exchange for pool tokens. These tokens track LP's proportional shares in the total reserves and can be exchanged for underlying assets at any time. Peripheral contracts interact with one or more base contracts.
AdaswapRouter02 provides many ways to exchange securely between different tokens.
WETH (wrapped ETH) is an ERC20 emulator for creating pools of ERC20 tokens.
The project infrastructure was assessed based on three types of users:

- Liquidity providers (LP) - interested in contributing ERC-20 tokens to common liquidity pools.
- Traders - exchanging ERC-20 tokens between them.
- Developers - using Adaswap smart contracts to create new interactions with tokens, trading interfaces and more.

Project Scope
The scope of the project are the following smart contracts and their dependencies:
- AdaSwapERC20,
- AdaSwapFactory,
- AdaSwapPair,
- Math
- UQ112×112;
And the serverless infrastructure:
MILKOMEDA_TESTNET_PROVIDER=https://rpc-devnet-cardano-evm.c1.milkomeda.com
MILKOMEDA_MAINNET_PROVIDER=https://rpc-mainnet-cardano-evm.c1.milkomeda.com
We have checked this smart contract for well-known and more specific vulnerabilities, for example:

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call
- Unchecked math

We also checked if the changes made by the developers impacted the infrastructure of the project. The list of changes was formed on the basis of the documentation provided by the Customer.
List of changes:

- changed the Solidity version from 0.5.16 to 0.8.13;
- moved events to interfaces;
- removed public constructor visibility;
- max uint number changed from uint(-1) to type(uint).max;
- added a new function pairCodeHash() that just returns the hash of the pair contract creation code;
- replaced IERC20 interface to use imported IERC20 from Openzeppelin version 4.7.0;
- removed unused contracts/libraries;
- removed SafeMath library.

During the audit, 5 files were analyzed. The following files have been scanned using the tools, checked and tested manually.

FILE	keccak256
AdaswapERC20.sol	a6dfddb7b1c5d8337661ce56141ea063766660cc5c767bd14f5e8506c9c0db2d
AdaswapFactory.sol	f0fdc7600268314adf9ff272d95f7d647a7e168a8148bb370d8782d12d98bbc5
AdaswapPair.sol	f04aa490f47b9622ea10d8537d1663f128bf692da03896848afde4200669e480
Math.sol	c59e50c307b9ff60cd3301d461b73079aaacea0c24b241a7fb0a03e88c732b1f
UQ112×112.sol	0fe1efe45c79702742aa04051812c14ae96620e366c13a6453d96cc3bb1103ae

Used dependencies

Solidity - version 0.8.13 and OpenZeppelin's smart contract version 4.7.0

- @openzeppelin/contracts/token/ERC20/IERC20.sol

Also, during the testing process, the following smart contracts were used, which form the infrastructure of the project.

FILE	keccak256
IAdaswapCallee.sol	c6e4e9a4006c4a70c1b55d923927a403e69afd361f28b3e75dd94e33bd15ba20
IAdaswapERC20.sol	b8a908d3416e2b092b50a1fab14cad9d7e07ce248808872166246c2e754feedd
IAdaswapFactory.sol	367c9fc753ad737c365600eda7ab6f94ba3d2387ffd8749a5779e74bdb725469
IAdaswapPair.sol	c2bc96ef99355cb6b8fff6dda07092454be6cf599458db8d0fa0a59e2d1ac8e5
TestAdaswapERC20.sol	7984aca630fb1721b3cd86d7a9532213f5b14c8ddefb38affc1abdf0a608c4af
TestERC20.sol	f7b9554d2a8cceeceefd68a5d48750f9193b5f27a18c3cef5b2704a1bb864204
TransferHelper.sol.sol	641bed5c630cfe38fa8d46779621b37da77f7d2342629124b0a83b569abf8ce4

FILE	keccak256
AdaswapRouter02.sol	ff8a43218edcd2e442556e8215742d4c94ab97f5da6123834bc958f4c1153154
WETH.sol	22f43b87c272b7dd6907a3b46ae7df918f4c1538cb37de9b20553557fdaca2ed
IAdaswapRouter01.sol	d7d52a2603be7fa1c9c68468b5b72324ab9ae57ca88d89e06399aa51a221a73f
IAdaswapRouter02.sol	0103572ffc5dc3c60b77a09a850142a59ce8e9dea1309c612a0de9c483530c9f
IWETH.sol	958e98b4b3b49f714de89733791d857208f02ad328b33ffb0470c98a4efa3896
AdaswapLibrary.sol	4781ab32c1ba9a0e72240611ec286dbc34e42711e2deb0351c29fb9bb152a222
DeflatingERC20.sol	e3a613ddcc96ca514f2164a69c56e8095148798f435e491ded3e9b1c5827e5ca
TestERC20.sol	3ccfb408940c41a27e5f07eadf0198b139b08d2fcadb05b5e683454cee82eba5
TestWETH.sol	0d9301ea438d06d21c3d7d1789479eaaeff68f3aae828a1735c4d4c7ee19d4dd

7. THE SEVERITY LEVEL OF THE ISSUES

Severity	Description
Critical	The problem will result in loss of assets or manipulation of data.
High	The problem will seriously affect the correctness of the business model.
Medium	The problem is still important to solve, but impractical to use.
Low	The problem is mainly related to outdated, unused code snippets.
Informational	This issue is mainly related to code style, informational statements, and is not required to be fixed.

8. FINDINGS AND RISK LEVELS

In the table below you can find a list of vulnerabilities found during manual code analysis.

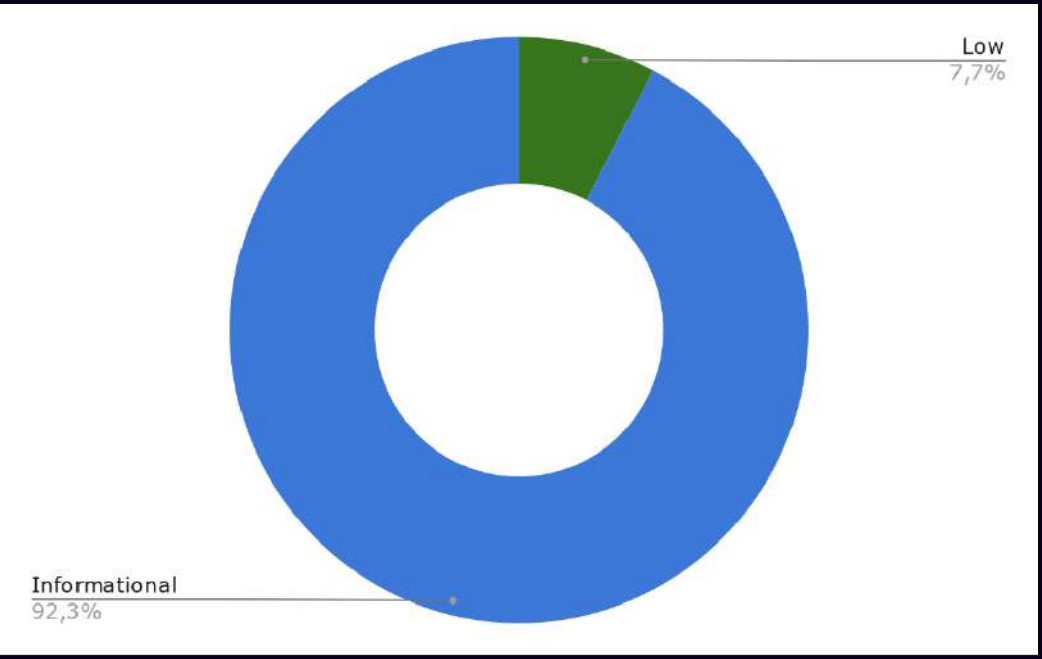
Findings	keccak256	Status
Extra gas consumption	Low	Fixed in commit @642630b
Replace array with counter	Low	Resolved
Assembly usage is not required	Informational	Fixed in commit @c6fcdbb
Using approve function of the ERC-20 token standard	Informational	Fixed in commit @1ae3ce6
Unnecessary keyword	Informational	Fixed in commit @ebafb3c
Missing zero address validation	Informational	Fixed in commit @84cbf03

List of results determined by automated analysis.

Findings	Risk level	Status
Weak PRNG	Informational	Noted
Dangerous strict equalities	Informational	Noted
Dangerous strict equalities	Informational	Noted
Reentrancy bug (no impact)	Informational	Noted
Missing zero address validation	Informational	Noted
Missing zero address validation	Informational	Noted
Missing zero address validation	Informational	Noted
Timestamp usage	Informational	Noted
Timestamp usage	Informational	Noted
Assembly usage	Informational	Noted
Low-level calls	Informational	Noted
Conformance to Solidity naming conventions	Informational	Noted
Conformance to Solidity naming conventions	Informational	Noted
Conformance to Solidity naming conventions	Informational	Noted
Conformance to Solidity naming conventions	Informational	Noted
Conformance to Solidity naming conventions	Informational	Noted
Conformance to Solidity naming conventions	Informational	Noted
Conformance to Solidity naming conventions	Informational	Noted
Conformance to Solidity naming conventions	Informational	Noted
Conformance to Solidity naming conventions	Informational	Noted

9. DIAGRAM OF THE FINDINGS

Critical	High	Medium	Low	Informational
0	0	0	2	24



10. FINDINGS OVERVIEW

SWC ID	Title	Pass
SWC-100	Function Default Visibility	OK
SWC-101	Integer Overflow and Underflow	OK
SWC-102	Outdated Compiler Version	OK
SWC-103	Floating Pragma	OK
SWC-104	Unchecked Call Return Value	OK
SWC-105	Unprotected Ether Withdrawal	OK
SWC-106	Unprotected SELFDESTRUCT Instruction	OK
SWC-107	Reentrancy	OK
SWC-108	State Variable Default Visibility	OK
SWC-109	Uninitialized Storage Pointer	OK
SWC-110	Assert Violation	OK
SWC-111	Use of Deprecated Solidity Functions	OK
SWC-112	Delegatecall to Untrusted Callee	OK
SWC-113	DoS with Failed Call	OK
SWC-114	Transaction Order Dependence	OK
SWC-115	Authorization through tx.origin	OK
SWC-116	Timestamp Dependence	OK
SWC-118	Incorrect Constructor Name	OK
SWC-119	Shadowing State Variables	OK
SWC-120	Weak Sources of Randomness	OK

SWC ID	Title	Pass
SWC-123	Requirement Violation	OK
SWC-124	Write to Arbitrary Storage Location	OK
SWC-127	Arbitrary Jump	OK
SWC-128	Gas Exhaustion	OK
SWC-129	Typographical Error	OK
SWC-130	Right-To-Left-Override control character	OK
SWC-131	Presence of unused variables	OK
SWC-132	Unexpected Ether balance	OK
SWC-133	Hash Collisions With Multiple Variable Length Arguments	OK
SWC-134	Message call with hardcoded gas amount	OK
SWC-135	Code With No Effects	OK
SWC-136	Unencrypted Private Data On-Chain	OK

11. RESULTS FROM MANUAL ANALYSIS

Below are a few results that may negatively affect the infrastructure of the project as a whole.

11.1 EXTRA GAS CONSUMPTION

Description:

The **sqrt** function from the **Math** library increases gas consumption as the value passed increases. Optimizing the smart contract's consumption of gas, and therefore the associated transaction costs, can reduce the cost of a transaction. It also has the potential to prevent malicious use of smart contracts.

Risk: Low

Location:

- ./core/contracts/AdaswapFactory.sol **#12**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;

// a library for performing various math operations

library Math {
	function min(uint x, uint y) internal pure returns (uint z) {
		z = x < y ? x : y;
	}

	// 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;
		}
	}
}
```

11.1.1 RECOMMENDATION ON IMPROVEMENT:

The **sqrt** function from the **Math** library can be replaced by the square root implementation below.

```
function sqrt(uint256 a) public 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 such that we have
	// `msb(a) <= a < 2*msb(a)`.
	// We also know that `k`, the position of the most significant bit, is such that
`msb(a) = 2**k`.
	// This gives `2**k < a <= 2**(k+1)` → `2**(k/2) <= sqrt(a) < 2 ** (k/2+1)`.
	// Using an algorithm similar to the msb computation, we are able to compute `result
= 2**(k/2)` which is a
	// good first approximation of `sqrt(a)` with at least 1 correct bit.
	uint256 result = 1;
	uint256 x = a;
	if (x >> 128 > 0) {
		x >>= 128;
		result <<= 64;
	}
	if (x >> 64 > 0) {
		x >>= 64;
		result <<= 32;
	}
	if (x >> 32 > 0) {
		x >>= 32;
		result <<= 16;
	}
	if (x >> 16 > 0) {
		x >>= 16;
		result <<= 8;
	}
	if (x >> 8 > 0) {
		x >>= 8;
		result <<= 4;
	}
	if (x >> 4 > 0) {
		x >>= 4;
		result <<= 2;
	}
	if (x >> 2 > 0) {
		result <<= 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
(precision doubles at
        // every iteration). We thus need at most 7 iteration to turn our partial result with
one bit 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);
        }
}
```

Proof of work:
Remix IDE v0.25.2 is used. Contracts were compiled with the optimizer enabled with a value of 999 999.

Created 4 smart contracts:

- **MathOld** - a smart contract containing the current implementation of the sqrt function
- **MathNew** - a smart contract containing an alternative implementation of the sqrt function taken from openzeppelin-contracts. (https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/math/Math.sol)
- **A** - a smart contract containing the callSqrt function where the sqrt function is called from the **MathOld** library
- **B** - a smart contract containing the callSqrt function where the sqrt function is called from the **MathNew** library

It should also be noted that when deploying smart contract B, where an alternative implementation of the sqrt function is used, the transaction price (gas) was less than when deploying smart contract A. The current implementation of the sqrt function in the Math library consumes less gas compared to the alternative with values less than 12,005,000.

Value	sqrt (old) gas consumption	sqrt (new) gas consumption
12000000	25152	25302
49000000	25668	25309
5700000000	26406	25285

Value	sqrt (old) gas consumption	sqrt (new) gas consumption
3354434332321	27660	25333
7777733544343323721	30390	25412

11.1.2 DECISION:

Replaced and tests fixed in commit @642630b.

11.2 REPLACE ARRAY WITH COUNTER

Description:

In the presented smart contract infrastructure, the allPairs array in AdaswapFactory is used for counting all created pairs. Using an array to store and retrieve the number of created pairs is not an optimal solution. In the case of iterating over the allPairs array, it would need to be split into chunks with constant gas, and furthermore, iterating the array from the outer contract would be very expensive, requiring one call to read each element.

Risk: Low

Location:

- ./core/contracts/AdaswapFactory.sol **#12**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...
contract AdaswapFactory is IAdaswapFactory {
        ...
        address[] public override allPairs;
        ...
}
```

11.2.1 RECOMMENDATION ON IMPROVEMENT:

The **allPairs** array in **AdaswapFactory** can be replaced with a simple counter. Also change the **IAdaswapFactory** interface and the **allPairsLength** function.

11.2.2 DECISION:

The developers decided to leave the array implementation in our current infrastructure, because it can be useful to get a paired address by index.

Below are the results that do not affect the infrastructure of the project.

11.3 ASSEMBLY USAGE IS NOT REQUIRED

Description:

Chain ID is available in the original Solidity 0.8.0 release. This way you can get the chain ID, such as a block timestamp. (https://docs.soliditylang.org/en/v0.8.0/units-and-global-variables.html)

Risk: Informational

Location:

- ./core/contracts/AdaswapERC20.sol **#22-31**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

contract AdaswapFactory is IAdaswapFactory {
        ...
        constructor() {
                uint chainId;
                assembly {
                        chainId := chainid()
                }
                DOMAIN_SEPARATOR = keccak256(
                abi.encode(
                        keccak256('EIP712Domain(string name,string version,uint256 chainId,address
verifyingContract)'),
                        keccak256(bytes(name)),
                        keccak256(bytes('1')),
                        chainId,
                        address(this)
                )
                );
        }
        ...
}
```

11.3.1 RECOMMENDATION ON IMPROVEMENT:

The assembly statement can be replaced with the native variable **block.chainid.**

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

contract AdaswapFactory is IAdaswapFactory {
        ...
        constructor() {
                DOMAIN_SEPARATOR = keccak256(
                abi.encode(
                        keccak256('EIP712Domain(string name,string version,uint256
chainId,address verifyingContract)'),
                        keccak256(bytes(name)),
                        keccak256(bytes('1')),
                        block.chainid,
                        address(this)
                )
                );
        }
        ...
}
```

11.3.2 DECISION:

Replaced and tests fixed in commit **@c6fcdbb.**

Below are the results that have no impact.

11.4 USING APPROVE FUNCTION OF THE ERC-20 TOKEN STANDARD

Description:
Ethereum mempool is the place where all pending transactions sit until the miner decides to include them into the block. For most cases, transactions with the highest gas prices are included first as the miners get the gas price for including these transactions into the block. Using a front-running attack one can spend approved tokens before the allowance value is updated.

Risk: Informational

11.4.1 RECOMMENDATION ON IMPROVEMENT:

Set to `0` the setting for one transaction allowance. The second transaction is approved for the new desired amount. Or use non-standard functions increaseAllowance() and decreaseAllowance().

11.4.2 DECISION:

Added functions **increaseAllowance** and **decreaseAllowance** from Openzeppelin ERC20 standard in commit @1ae3ce6.

11.5 UNNECESSARY KEYWORD

Description:
Since Solidity 0.8.0 release, we can remove the **override** keyword to implement interface functions.

Risk: Informational

Location:
- ./core/contracts/AdaswapERC20.sol
- ./core/contracts/AdaswapFactory.sol
- ./core/contracts/AdaswapPair.sol

11.5.1 RECOMMENDATION ON IMPROVEMENT:

Remove the unnecessary **override** keyword.

11.5.2 DECISION:

Removed all unnecessary keywords in commit @ebafb3c.

11.6 MISSING ZERO ADDRESS VALIDATION

Description:
Missing zero address validation of **feeToSetter.**

Risk: Informational

Location:
- ./core/contracts/AdaswapFactory.sol **#9**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

contract AdaswapFactory is IAdaswapFactory {
        ...
        address public override feeToSetter;
        ...
        constructor(address _feeToSetter) public {
                feeToSetter = _feeToSetter;
        }
        ...
}
```

Location:
- ./core/contracts/AdaswapFactory.sol **#50**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

contract AdaswapFactory is IAdaswapFactory {
        ...
        address public override feeToSetter;
        …
        function setFeeToSetter(address _feeToSetter) external override {
                require(msg.sender == feeToSetter, 'Adaswap: FORBIDDEN');
                feeToSetter = _feeToSetter;
        }
}
```

11.6.1 RECOMMENDATION ON IMPROVEMENT:

It is recommended to validate the variable **_feeToSetter** to contain a non-zero value before the assignment. Or implement the logic to delete the variable when it is set to zero address, which will allow you to get a gas refund for freeing up storage space. If for some reason it is necessary to permanently block access to changing _feeToSetter, that is, set a zero address in _feeToSetter, then instead of setting a zero value in _feeToSetter directly, you can delete it from the storage. That would provide a refund for this transaction (no more than half from the cost of the transaction or 15,000). In the original version, the zero address is set directly, and in the version with the deletion, the zero address will also be set to zero plus we will get a refund, but the deletion must be written in the code.

11.6.2 DECISION:

Added validation in commit @84cbf03.

12. RESULTS FROM SEMI-AUTOMATIC SCANS

We also used a Solidity static analysis framework of our own development which runs a suite of vulnerability detectors, shows visual information about the contract details, and provides an API to write custom analyses quickly. Slither enables developers to find vulnerabilities, enhance their code comprehension, and promptly prototype custom analyses. Each Solidity file and the whole project have been analysed. We got a report with a few results. **None of the outcomes pose a risk to the project infrastructure.** Below are some of them.

12.1 WEAK PRNG

Description:
Weak PRNG due to a modulo on block.timestamp, now or blockhash. These can be influenced by miners to some extent, so they should be avoided.

Risk: Informational

Location:

- ./core/contracts/AdaswapPair.sol **#60-73**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

contract AdaswapPair is IAdaswapPair, AdaswapERC20 {
	...
	function _update(uint balance0, uint balance1, uint112 _reserve0, uint112 _reserve1) private {
		...
		uint32 blockTimestamp = uint32(block.timestamp % 2**32);
		...
	}
	...
}
```

12.2 DANGEROUS STRICT EQUALITIES

Description:

Use of strict equalities that can be easily manipulated by an attacker.

Risk: Informational

Location:

- ./core/contracts/AdaswapPair.sol **#97-118**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

contract AdaswapPair is IAdaswapPair, AdaswapERC20 {
	...
	function mint(address to) external override lock returns (uint liquidity) {
		...
		if (_totalSupply == 0) {
			liquidity = Math.sqrt(amount0 * amount1) - MINIMUM_LIQUIDITY;
			_mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first
MINIMUM_LIQUIDITY tokens
		} else {
			liquidity = Math.min(amount0 * _totalSupply / _reserve0, amount1 *
_totalSupply / _reserve1);
		}
		...
	}
	...
}
```

12.3 DANGEROUS STRICT EQUALITIES

Description:
Use of strict equalities that can be easily manipulated by an attacker.

Risk: Informational

Location:
- ./core/contracts/AdaswapPair.sol **#43-46**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

contract AdaswapPair is IAdaswapPair, AdaswapERC20 {
        ...
        function _safeTransfer(address token, address to, uint value) private {
                (bool success, bytes memory data) =
token.call(abi.encodeWithSelector(SELECTOR, to, value));
                require(success && (data.length == 0 || abi.decode(data, (bool))), 'Adaswap:
TRANSFER_FAILED');
        }
        ...
}
```

12.4 REENTRANCY BUG (NO IMPACT)

Description:
Detection of the reentrancy bug. State variables written after the **initialize** call.

Risk: Informational

Location:
- ./core/contracts/AdaswapFactory.sol #**26-41**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

contract AdaswapFactory is IAdaswapFactory {
        ...
        function createPair(address tokenA, address tokenB) external override returns
(address pair) {
                ...
                IAdaswapPair(pair).initialize(token0, token1);
                getPair[token0][token1] = pair;
                getPair[token1][token0] = pair; // populate mapping in the reverse direction
                allPairs.push(pair);
                emit PairCreated(token0, token1, pair, allPairs.length);
        }
        ...
}
```

Decision: Noted. This does not affect core functionality.

12.5 MISSING ZERO ADDRESS VALIDATION

Description:
Missing zero address validation of **_feeTo.**

Risk: Informational

Location:
- ./core/contracts/AdaswapFactory.sol **#43**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

contract AdaswapFactory is IAdaswapFactory {
            ...
            address public override feeTo;
            ...
            function setFeeTo(address _feeTo) external override {
                        require(msg.sender == feeToSetter, 'Adaswap: FORBIDDEN');
                        feeTo = _feeTo;
            }
            ...
}
```

12.6 MISSING ZERO ADDRESS VALIDATION

Description:
Missing zero address validation of **_token0.**

Risk: Informational

Location:
- ./core/contracts/AdaswapPair.sol **#55**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

contract AdaswapPair is IAdaswapPair, AdaswapERC20 {
        ...
        function initialize(address _token0, address _token1) external override {
                require(msg.sender == factory, 'Adaswap: FORBIDDEN'); // sufficient check
                token0 = _token0;
                token1 = _token1;
        }
        ...
}
```

12.7 MISSING ZERO ADDRESS VALIDATION

Description:
Missing zero address validation of **_token1.**

Risk: Informational

Location:
- ./core/contracts/AdaswapPair.sol **#56**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

contract AdaswapPair is IAdaswapPair, AdaswapERC20 {
	...
	function initialize(address _token0, address _token1) external override {
		require(msg.sender == factory, 'Adaswap: FORBIDDEN'); // sufficient check
		token0 = _token0;
		token1 = _token1;
	}
	...
}
```

12.8 TIMESTAMP USAGE

Description:
A **block.timestamp** value can be manipulated by miners.

Risk: Informational

Location:
- ./core/contracts/AdaswapERC20.sol **#74-86**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

contract AdaswapERC20 is IAdaswapERC20 {
	...
	function permit(address owner, address spender, uint value, uint deadline, uint8 v,
bytes32 r, bytes32 s) external override {
		require(deadline >= block.timestamp, 'Adaswap: EXPIRED');
	...
	}
	...
}
```

Decision: Noted. This does not affect core functionality.

12.9 TIMESTAMP USAGE

Description:
A **block.timestamp** value can be manipulated by miners.

Risk: Informational

Location:
- ./core/contracts/AdaswapPair.sol **#60-73**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

contract AdaswapPair is IAdaswapPair, AdaswapERC20 {
    ...
    function _update(uint balance0, uint balance1, uint112 _reserve0, uint112 _reserve1)
private {
        require(balance0 <= type(uint112).max && balance1 <= type(uint112).max,
'Adaswap: OVERFLOW');
        uint32 blockTimestamp = uint32(block.timestamp % 2**32);
        ...
    }
    ...
}
```

Decision: Noted. This does not affect core functionality.

12.10 ASSEMBLY USAGE

Description:
The use of assembly is error-prone and should be avoided.

Risk: Informational

Location:
- ./core/contracts/AdaswapFactory.sol **#26-41**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

contract AdaswapFactory is IAdaswapFactory {
        ...
        assembly {
                pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        ...
}
```

Decision: Noted. This does not affect core functionality.

12.11 LOW-LEVEL CALLS

Description:
The use of low-level calls is error-prone. Low-level calls do not check for code existence or call success.

Risk: Informational

Location:
- ./core/contracts/AdaswapPair.sol **#43-46**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

contract AdaswapFactory is IAdaswapFactory {
        ...
        function _safeTransfer(address token, address to, uint value) private {
                (bool success, bytes memory data) = token.call(abi.encodeWithSelector(SELECTOR, to,
value));
                require(success && (data.length == 0 || abi.decode(data, (bool))), 'Adaswap:
TRANSFER_FAILED');
        }
        ...
}
```

Decision: Noted. This does not affect core functionality.

12.12 CONFORMANCE TO SOLIDITY NAMING CONVENTIONS

Description:
Naming conventions are powerful when adopted and used broadly. The use of different conventions can convey significant meta information that would otherwise not be immediately available.
Parameter **_feeTo** is not in mixedCase.

Risk: Informational

Location:
- ./core/contracts/AdaswapFactory.sol **#43**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

contract AdaswapFactory is IAdaswapFactory {
        ...
        function setFeeTo(address _feeTo) external override {
                require(msg.sender == feeToSetter, 'Adaswap: FORBIDDEN');
                feeTo = _feeTo;
        }
        ...
}
```

12.13 CONFORMANCE TO SOLIDITY NAMING CONVENTIONS

Description:
Naming conventions are powerful when adopted and used broadly. The use of different conventions can convey significant meta information that would otherwise not be immediately available.
Parameter **_feeToSetter** is not in mixedCase.

Risk: Informational

Location:

- ./core/contracts/AdaswapFactory.sol **#48**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

contract AdaswapFactory is IAdaswapFactory {
        ...
        function setFeeToSetter(address _feeToSetter) external override {
                require(msg.sender == feeToSetter, 'Adaswap: FORBIDDEN');
                feeToSetter = _feeToSetter;
        }
        ...
}
```

12.14 CONFORMANCE TO SOLIDITY NAMING CONVENTIONS

Description:

Naming conventions are powerful when adopted and used broadly. The use of different conventions can convey significant meta information that would otherwise not be immediately available.

Function **AdaswapCall** is not in mixedCase.

Risk: Informational

Location:

- ./core/contracts/interfaces/IAdaswapCallee.sol **#5**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

interface IAdaswapCallee {
      function AdaswapCall(address sender, uint amount0, uint amount1, bytes calldata data)
external;
}
```

12.15 CONFORMANCE TO SOLIDITY NAMING CONVENTIONS

Description:
Naming conventions are powerful when adopted and used broadly. The use of different conventions can convey significant meta information that would otherwise not be immediately available.
Parameter **_token0** is not in mixedCase.

Risk: Informational

Location:
- ./core/contracts/AdaswapPair.sol #**53**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

contract AdaswapPair is IAdaswapPair, AdaswapERC20 {
	...
	function initialize(address _token0, address _token1) external override {
		require(msg.sender == factory, 'Adaswap: FORBIDDEN'); // sufficient check
		token0 = _token0;
		token1 = _token1;
	}
	...
}
```

12.16 CONFORMANCE TO SOLIDITY NAMING CONVENTIONS

Description:
Naming conventions are powerful when adopted and used broadly. The use of different conventions can convey significant meta information that would otherwise not be immediately available.
Parameter **_token1** is not in mixedCase.

Risk: Informational

Location:

- ./core/contracts/AdaswapPair.sol **#53**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

contract AdaswapPair is IAdaswapPair, AdaswapERC20 {
	...
	function initialize(address _token0, address _token1) external override {
		require(msg.sender == factory, 'Adaswap: FORBIDDEN'); // sufficient check
		token0 = _token0;
		token1 = _token1;
	}
	...
}
```

12.17 CONFORMANCE TO SOLIDITY NAMING CONVENTIONS

Description:

Naming conventions are powerful when adopted and used broadly. The use of different conventions can convey significant meta information that would otherwise not be immediately available.

Function **DOMAIN_SEPARATOR** is not in mixedCase.

Risk: Informational

Location:

- ./core/contracts/interfaces/IAdaswapERC20.sol **#8**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

interface IAdaswapERC20 is IERC20 {
	...
	function DOMAIN_SEPARATOR() external view returns (bytes32);
	...
}
```

12.18 CONFORMANCE TO SOLIDITY NAMING CONVENTIONS

Description:
Naming conventions are powerful when adopted and used broadly. The use of different conventions can convey significant meta information that would otherwise not be immediately available.
Function **PERMIT_TYPEHASH** is not in mixedCase.

Risk: Informational

Location:
- ./core/contracts/interfaces/IAdaswapERC20.sol **#9**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

interface IAdaswapERC20 is IERC20 {
        ...
        function PERMIT_TYPEHASH() external pure returns (bytes32);
        ...
}
```

12.19 CONFORMANCE TO SOLIDITY NAMING CONVENTIONS

Description:
Naming conventions are powerful when adopted and used broadly. The use of different conventions can convey significant meta information that would otherwise not be immediately available.
Variable **DOMAIN_SEPARATOR** is not in mixedCase.

Risk: Informational

Location:

- ./core/contracts/AdaswapERC20.sol **#15**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

contract AdaswapERC20 is IAdaswapERC20 {
        ...
        constructor() {
                DOMAIN_SEPARATOR = keccak256(
                        abi.encode(
                        keccak256('EIP712Domain(string name,string version,uint256
chainId,address verifyingContract)'),
                        keccak256(bytes(name)),
                        keccak256(bytes('1')),
                        block.chainid,
                        address(this)
                        )
                );
        }
}
```

12.20 CONFORMANCE TO SOLIDITY NAMING CONVENTIONS

Description:

Naming conventions are powerful when adopted and used broadly. The use of different conventions can convey significant meta information that would otherwise not be immediately available.

Function **MINIMUM_LIQUIDITY** is not in mixedCase.

Risk: Informational

Location:

- ./core/contracts/interfaces/IAdaswapPair.sol **#20**

Code section:

```
// SPDX-License-Identifier: MIT
pragma solidity =0.8.13;
...

interface IAdaswapPair is IAdaswapERC20 {
        ...
        function MINIMUM_LIQUIDITY() external pure returns (uint);
        ...
}
```

13. CONCLUSION

Auditors conducted the smart contract security audit. The specific goal of the security audit was to identify if an attacker could compromise Customer’s smart contract protection.

The goal of the audit was met. It was determined that it was not possible to misuse the smart contract or to violate the Customer's business requirements directly. The Customer has promptly mitigated all the findings that were discovered during the audit.

The changes made to the basic protocol did not violate the integrity and security of the project.