VIDMA

SMART CONTRACT AUDIT

Project: ARTT Network
Date: April 5th, 2023

TABLE OF CONTENTS

SUMMARY

Vidma is pleased to present this audit report outlining our assessment of code, smart contracts, and other important audit insights and suggestions for management, developers, and users.

The audited scope of work includes the ArtToken smart contract. ArtToken is a reflective token. It has 4 types of fees: taxFee, liquidityFee, burnFee, foundationFee. Depending on the percentage of commissions and types of transfers, a small part of the amount is deducted from the transaction, which goes to cover general needs. In addition, the ArtToken through the router interacts with uniswap's functions: adding liquidity, swapping tokens, etc. There are also several categories of users in the contract: those who are exempted or added to the lists regarding the collection of commissions, as well as those who are added or removed from the lists of rewards. Depending on which category a certain user is in, a different type of token transfer takes place.

As the contract is deployed, the project team hasn't fixed the issues presented in the report. However, the unfixed issues are not bearing the security or operational risks for the users of the contract.

During the audit process, the Vidma team found several issues. A detailed summary and the current state are displayed in the table below.

Severity of the issue	Total found	Resolved	Invalid	Unresolved
Critical	0 issues	0 issues	0 issues	0 issues
High	0 issues	0 issues	0 issues	0 issues
Medium	4 issues	0 issues	2 issues	2 issues
Low	21 issues	0 issues	0 issues	21 issues
Informational	7 issues	0 issues	0 issues	7 issues
Total	**32 issues**	**0 issues**	**2 issues**	**30 issues**

After evaluating the findings in this report and the final state after fixes, the Vidma auditors can state that the contracts are secure. However, we still recommend fixing the issues mentioned in this report to enhance the performance of the smart contracts. Under the given circumstances, we set the following risk level:

To set the codebase quality mark, our auditors are evaluating the initial commit given for the scope of the audit and the last commit with the fixes. This approach helps us adequately and sequentially evaluate the quality of the code. Code style, optimization of the contracts, the number of issues, and risk level of the issues are all taken into consideration. The Vidma team has developed a transparent evaluation codebase quality system presented below.

Severity of the issue	Resolved	Unresolved
Critical	1	10
High	0.8	7
Medium	0.5	5
Low	0.2	0.5
Informational	0	0.1

Please note that the points are deducted out of 100 for each and every issue on the list of findings (according to the current status of the issue). Issues marked as "not valid" are not subject to point deduction.

Codebase quality: 79.20

Evaluating the **initial commit** and the **last commit with the fixes**, Vidma audit team set the following **codebase quality** mark.

Score

Based on the **overall result of the audit** and the state of the final reviewed commit, the Vidma audit team grants the following **score**:

In addition to manual check and static analysis, the auditing team has conducted a number of integrated autotests to ensure the given codebase has an adequate performance and security level.

The test results and coverage can be found in the accompanying section of this audit report.

Please be aware that this audit does not certify the definitive reliability and security level of the contract. This document describes all vulnerabilities, typos, performance issues, and security issues found by the Vidma audit team. If the code is still under development, we highly recommend running one more audit once the code is finalized.

SCOPE OF WORK

We strive to establish a strong presence as a foundation dedicated to supporting creators on an unprecedented scale. This commitment has led to the creation of the first crypto-funded scholarship platform.

Within the scope of this audit, two independent auditors thoroughly investigated the given codebase and analyzed the overall security and performance of the smart contracts.

The audit was conducted from March 27, 2023 to April 3, 2023. The outcome is disclosed in this document.

The scope of work for the given audit consists of the following contract:

- ArtToken.

The source code was taken from the following **source**:
https://bscscan.com/token/0x9158e70119d661ba0caeb2b392edd9565cac82b7?a=0x9158e70119d661ba0caeb2b392edd9565cac82b7#code

Initial commit submitted for the audit:
https://bscscan.com/token/0x9158e70119d661ba0caeb2b392edd9565cac82b7?a=0x9158e70119d661ba0caeb2b392edd9565cac82b7#code

As a reference to the contracts logic, business concept, and the expected behavior of the codebase, the ARTT Network team has provided the following documentation:

Whitepaper:
https://artt.network/wp-content/uploads/2023/03/ARTT-Whitepaper-21.03.2023.pdf

WORKFLOW OF THE AUDITING PROCESS

Vidma audit team uses the most sophisticated and contemporary methods and well-developed techniques to ensure contracts are free of vulnerabilities and security risks. The overall workflow consists of the following phases:

Phase 1: The research phase

Research

After the Audit kick-off, our security team conducts research on the contract's logic and expected behavior of the audited contract.

Documentation reading

Vidma auditors do a deep dive into your tech documentation with the aim of discovering all the behavior patterns of your codebase and analyzing the potential audit and testing scenarios.

The outcome

At this point, the Vidma auditors are ready to kick off the process. We set the auditing strategies and methods and are prepared to conduct the first audit part.

Phase 2: Manual part of the audit

Manual check

During the manual phase of the audit, the Vidma team manually looks through the code in order to find any security issues, typos, or discrepancies with the logic of the contract. The initial commit as stated in the agreement is taken into consideration.

Static analysis check

Static analysis tools are used to find any other vulnerabilities in smart contracts that were missed after a manual check.

The outcome

An interim report with the list of issues.

Phase 3: Testing part of the audit

Integration tests

Within the testing part, Vidma auditors run integration tests using the Truffle or Hardhat testing framework. The test coverage and the test results are inserted in the accompanying section of this audit report.

The outcome

Second interim report with the list of new issues found during the testing part of the audit process.

STRUCTURE AND ORGANIZATION OF THE FINDINGS

For simplicity in reviewing the findings in this report, Vidma auditors classify the findings in accordance with the severity level of the issues. (from most critical to least critical).

All issues are marked as “Resolved” or “Unresolved”, depending on if they have been fixed by ARTT Network or not. The issues with “Not Relevant” status are left on the list of findings but are not eligible for the score points deduction.

The latest commit with the fixes reviewed by the auditors is indicated in the “Scope of Work” section of the report.

The Vidma team always provides a detailed description of the issues and recommendations on how to fix them.

Classification of found issues is graded according to 6 levels of severity described below:

Critical

The issue affects the contract in such a way that funds may be lost or allocated incorrectly, or the issue could result in a significant loss.

Example: Underflow/overflow, precisions, locked funds.

High

The issue significantly affects the ability of the contract to compile or operate. These are potential security or operational issues.

Example: Compilation errors, pausing/unpausing of some functionality, a random value, recursion, the logic that can use all gas from block (too many iterations in the loop), no limitations for locking period, cooldown, arithmetic errors which can cause underflow, etc.

Medium

The issue slightly impacts the contract's ability to operate by slightly hindering its intended behavior.

Example: Absence of emergency withdrawal of funds, using assert for parameter sanitization.

Low

The issue doesn't contain operational or security risks, but are more related to optimization of the codebase.

Example: Unused variables, inappropriate function visibility (public instead of external), useless importing of SCs, misuse or disuse of constant and immutable, absent indexing of parameters in events, absent events to track important state changes, absence of getters for important variables, usage of string as a key instead of a hash, etc.

Informational

Are classified as every point that increases onboarding time and code reading, as well as the issues which have no impact on the contract's ability to operate.

Example: Code style, NatSpec, typos, license, refactoring, naming convention (or unclear naming), layout order, functions order, lack of any type of documentation.

MANUAL REPORT

The hardcoded value of variable *numTokensSellToAddToLiquidity*

Medium | TM – 01 | Invalid

There are cases that cause negative price action by accumulating a native token and then a 50% sell-off occurs when a hard-coded limit (value of the *numTokensSellToAddToLiquidity* variable) is reached. To avoid this, it is important that the owner has the ability to change the threshold value.

Recommendation:

Please, create an additional new function through which the owner can set a new value for the *numTokensSellToAddToLiquidity* variable.

Comment from the project team:

I agree with this, it's hardcoded, but that's what we need, we don't want to change this and even if it's not hardcoded when we renounce ownership it will be the same as harcoded, nobody could be able to change it. No impact on users.

```
uint256 private numTokensSellToAddToLiquidity = 500000 * 10 ** 6 *
10 ** 9;
```

Reason to mark the issue as "Invalid":
This is the intended behavior of the contract accordingly to the business logic.

Unchecked returned value by *addLiquidityETH()*

Medium | TM – 02 | Unresolved

The return values of function *addLiquidityETH()* are not properly handled.

Recommendation:

Use variables to receive the return value of the functions mentioned above and handle both success and failure cases if needed by the business logic.

Comment from the project team:

Yes, the returned values are not handled, but we don't care about the business logic about the returned values, if we have them they won't be used and we just make the smart contract heavier.

```
function addLiquidity (uint256 tokenAmount, uint256 ethAmount)
private {
  // approve token transfer to cover all possible scenarios
  _approve (address (this), address (uniswapV2Router), tokenAmount);
  // add the Liquidity
  uniswapV2Router.addLiquidityETH{value: ethAmount}(
    address(this),
    tokenAmount,
    0, // slippage is unavoidable
    0, // slippage 1s unavoidable
    owner(),
    block.timestamp
  );
}
```

Possible *out of gas* error

Medium | TM – 03 | Invalid

Use of multiple for loops in the contract, this has the danger of running into *out of gas* errors if they are not kept under control.

Recommendation:

This can be avoided by adding a *gasleft() < 20000* type of condition that if it returns true it will break the execution so the *out of gas* error message will be avoided.

Comment from the project team:

In theory, this is possible, but in our case, it's not, because the for loop has O(n) complexity where the n is the number of excluded addresses which are only few. No impact on users.

Incorrect logic of the ArtToken contract re accumulated funds

Medium | TM – 04 | Unresolved

As a result of interaction with the uniswap's functions, part of the native currency is accumulated on the contract. The ArtToken's logic does not provide for the correct processing of funds or the possibility of their withdrawal.

Recommendation:

Add logic for plugs or implement the new function for withdrawing native currency that can accumulate on the contract when performing functions from the router (applies to liquidity and swaps).

Comment from the project team:

I don't really see a critical issue here, since the point is that the contract should accumulate funds and then use the swap and liquify method, it will sell half of the artt for bnb and then will add the whole amount to the pancakeswap liquidity pool.

Also when the token raises we have a plan to renounce ownership. If we create the method for removing the tokens, that means that the owner could remove them instead of putting more liquidity to the pool which is a good thing for users. I even think that this is much better that we don't have this method. If we want to not have funds accumulating in the contract we can change the fee to 0 and disable swap and liquify.

```
uint256 contractiokenBalance = balanceOf(address(this))
if (contractTokenBalance >= _maxTxAmount)
{
     contractTokenBalance = _maxTxAmount;
}
bool overMinTokenBalance = contractTokenBalance >=
numTokensSellToAddToLiquidity;
if (
     overMinTokenBalance &&
     !inSwapAndLiquify &&
     from != uniswapV2Pair &&
     swapAndLiquifyEnabled
) {
     contractTokenBalance = numTokensSellToAddToLiquidity;
     //add liquidity
     swapAndLiquify(contractTokenBalance);
}
```

State variable visibility is not set

Low | ML – 01 | Unresolved

It is best practice to set the visibility of state variables explicitly. The default visibility for *inSwapAndLiquify* is internal. Other possible visibility settings are public and private.

Recommendation:

Consider to set specify the visibility of the state variable.

Comment from the project team:

The default visibility is internal and we don't have an explicit modifier since we need it to be internal. No impact on users.

Floating pragma

Low | ML – 02 | Unresolved

The current version of solc in contract ArtToken is >=0.6.0 <0.8.0 and it is better to lock the pragma to a specific version.

Contracts should be deployed with the same compiler version and flags they have tested thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

As important additional recommendation: change the 0.6 solidity version to 0.8 The new version of the contract opens up much more possibilities, helps to better optimize the code, and saves gas usage, which also saves money on commission. In addition, you can use many new technologies that are only available in newer versions.

Recommendation:

Lock pragma to a specific version, also change the solidity version to a newer one.

Comment from the project team:

Yes, it's better to lock the version but since we deployed the token we are not having a lot of benefits if we lock the version and when the token was developed we weren't using any features newer than 0.8.0 which were relatively new. No impact on users.

Useless event declaration

Low | ML – 03 | Unresolved

The event *MinTokensBeforeSwapUpdated* is never used.

Recommendation:

Consider removing the useless event.

Comment from the project team:

Yes, it's useless, we were not using it and we forgot to remove it. One event is more on the contract, no real impact here. No impact on users.

Lack of event emits while some important data is set

Low | ML – 04 | Unresolved

In the ArtToken contract should be event emit in functions where the state variables are overriding. The next functions functions do not satisfy this requirement: *deliver()*, *excludeFromReward()*, *includeInReward()*, *excludeFromFee()*, *includeInFee()*, *setTaxFeePercent()*, *setLiquidityFeePercent()*, *setFoundationFee()*, *setFoundationWallet()*, *setFeeManager()*, *setMaxTxPercent()*, *_reflectFee()*, *_takeLiquidity()*, *_removeAllFee()*, *_restoreAllFee()*, *_addLiquidity()*, *swapTokensForEth()*.

Recommendation:

Consider the emitting events in these specific cases to be able to track critical data change.

Comment from the project team:

I agree with this, not that we needed in our case but this is minus for us. I would have fix this immediately if the contract was not deployed. It does not have real impact but it really good feature to have. No impact on users.

Usage of *msg.sender*

Low | ML – 05 | Unresolved

In ArtToken smart contract could be used *_msgSender()* function instead of *msg.sender* because it is inherited from Context contract. In addition, the *_msgSender()* function call requires less gas than *msg.sender*.

Recommendation:

Use *_msgSender()* function instead of msg.sender.

Comment from the project team:

Yes, this is also correct, but we are using only in the modifier for the *feeManager* and only the fee manager will pay that small amount of more gas. No real impact for the end user.

Lack of zero-value check

Low | ML – 06 | Unresolved

There is no zero address check for the variables *_routerAddress*, *_foundationAddress*, *_feeManager*, *tokenOwner* in the ArtToken's constructor.

Recommendation:

Consider requiring non-zero address initialization for the state variables.

Comment from the project team:

Also this is valid but since the contract is deployed and addresses are set we don't need this. Constructor is called only once on deployment and there is no impact on the users.

Lack of require statements

Low | ML – 07 | Unresolved

The next functions can change the state variables, but doesn't provided with input parameters validation: *deliver()*, *excludeFromReward()*, *includeInReward()*, *excludeFromFee()*, *includeInFee()*, *setTaxFeePercent()*, *setLiquidityFeePercent()*, *setFoundationFee()*, *setFoundationWallet()*, *setFeeManager()*, *setMaxTxPercent()*, *swapAndLiquify()*, *swapTokensForEth()*, *addLiquidity()*.

Recommendation:

Consider adding the require statements.

Comment from the project team:

Yes, we don't have require statements on these few functions, but some are just addresses and some percentages which more or less we don't have some specific requirements. Also, these methods could be called by the owner and again there is no real impact on the users.

Useless interface

Low | ML – 08 | Unresolved

The *IUniswapV2Pair* interface is never used.

Recommendation:

Consider removing needless interface.

Comment from the project team:

Correct, it's useless, we should have deleted it, there is no impact on the users or token, just made the contract size a little bit bigger. No impact on users.

Useless inherit

Low | ML – 09 | Unresolved

The ArtToken contract inherits Context and Ownable smart contracts. But Ownable is also inherits Context contract. The same contract used twice.

Recommendation:

Consider removing the Context inheritance from the ArtToken smart contract.

Comment from the project team:

Correct, it's useless, we should have deleted, there is no impact to the users or token, just made the contract size a little bit bigger. No impact on users.

Incorrect error in require statement

Low | ML – 10 | Unresolved

There is a situation that function *includeInReward()* is provided with check that account must be included in reward and *_isExcluded[account]* must be true in next require statement:
require(_isExcluded[account], "Account is already excluded");
But the message doesn't describe correctly error message for this require.

Recommendation:

Consider changing the message to "Account is already included".

Comment from the project team:

Correct, this is *onlyOwner* function and we will have a not about this typo, it should have been "account is already included". No impact on users.

The code optimization of locking functions

Low | TL – 01 | Unresolved

To avoid data redundancy and redundant use of the same code in the *lock()* and *unlock()* functions, you can use ready-made functions with the same actions from the Ownable contract and its inherited Context SC.

Recommendation:

Use ready-made functions *_transferOwnership()*, *owner()*, *_msgSender()*.

Comment from the project team:

It's correct, but no real difference here if we change the value directly or through pre-made function in inherited contracts. No impact on users.

Deprecated record of the maximum value of the *uint256* variable

Low | TL – 02 | Unresolved

Starting with version 0.6.8, a new entry for the maximum value of the *uint256* variable has been introduced. Instead of *~uint256(0)*, it is customary to write *type(uint256).max*.

Recommendation:

Please update the entry for the constant variable *MAX*.

Comment from the project team:

Correct, but no real impact here.

The variables that must be declared as constants

Low | TL – 03 | Unresolved

The ArtToken contract uses variables whose values are immutable, but they are not declared as constants.

Recommendation:

Declare the following variables as constants: *_tTotal*, *_name*, *_symbol*, *_decimals*.

Comment from the project team:

The variables should be declared as constants, but they don't have to be. No impact on users.

Redundant initialization of variables during declaration

Low | TL – 04 | Unresolved

The initialization of variables with zero values when they are declared is the default, so it does not need to be explicitly specified.

Recommendation:

Remove explicit zero initialization for the next variables: *_burnFee*, *_previousBurnFee*, *_foundationFee*, *_previousFoundationFee*.

Comment from the project team:

It's redundant, but it's better for reading the code, that's why they stayed. In my opinion, we should have done the initialization in the constructor. No impact on users.

Functions visibility optimization

Low | TL – 05 | Unresolved

There are some functions that are never called in the contract itself with visibility public so they can be marked as external:

- *name()*, *symbol()*, *decimals()*, *totalSupply()*, *transfer()*, *allowance()*, *approve()*, *transferFrom()*, *increaseAllowance()*, *decreaseAllowance()* – if don't use from ERC20 SC;
- *isExcludedFromReward()*, *totalFees()*, *deliver()*, *reflectionFromToken()*, *excludeFromReward()*, *excludeFromFee()*, *includeInFee()*, *setSwapAndLiquifyEnabled()*, *isExcludedFromFee()*.

Recommendation:

Consider changing visibility from public to external to safe gas usage on calling in the functions.

Comment from the project team:

I agree with this. This functions will be called but Token operator and he will pay more gas. No real impact on the users.

The redundant condition

Low | TL – 06 | Unresolved

The function *_tokenTransfer()* has an extra condition, which is additionally simply included in the else branch:

```
else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
_transferStandard(sender, recipient,
(amount.sub(burnAmt).sub(foundationAmount)));}.
```

Recommendation:

Remove the redundant code.

Comment from the project team:

Correct. No real impact on the user.

Local variable shadowing

Low | TL – 07 | Unresolved

In the contract ArtToken there are several shadowing of the variables. Shadowing appears in the next functions: *_approve()* and *allowance()*.

Recommendation:

Consider renaming the local variables that shadow another component.

Comment from the project team:

Correct, but we are overriding the approve method and using the *_approve()* as a internal which in my opinion is much more understandable for reading the code. No impact on users.

Unable to set value for the *_burnFee* variable

Low | TL – 08 | Unresolved

The contract stores multiple values for different commissions. All of them can be set via setters. Instead, the burning commission is initialized to zero at the time of announcement and cannot be changed later by the fee manager or owner. If your business logic dictates that *_burnFee* will always be 0, then this variable is redundant and so are all calculations with it.

Recommendation:

Add a function to be able to set the value of the *_burnFee* or remove this variable with all unneeded calculations with this.

Comment from the project team:

This is correct, we can't change the burn fee. The fee is set to 0 so when users are burning tokens they won't be taken any fee. No impact on users.

Duplicate saved data

Low | TL – 09 | Unresolved

The contract ArtToken declares 2 variables that store the address of the router: *uniswapV2Router* and *routerAddress*. At the moment, the value of router can be set only once when the contract is deployed in its constructor. The router address is stored twice in the constructor. This is meaningless, even when there is an additional option to install a new router, since the result can be stored in one variable. In this case, the redundancy of stored data, as well as wasted reserved memory, is implemented in the contract.

Recommendation:

Remove the *routerAddress* variable, instead store the router address only in the *uniswapV2Router* variable.

Comment from the project team:

Correct, we have duplicate value of one address. Contract storage is wasted for 1 address which in reality is not a problem. No impact on users.

The same calculations are repeated

Low | TL – 10 | Unresolved

In the *_tokenTransfer()* function, the value of the same calculation is passed as the token transfer amount parameter:

```
(amount.sub(burnAmt).sub(foundationAmount))
```

Recommendation:

Save the result of the calculation in a separate variable, and then use it in conditions.

Comment from the project team:

Yes, the calculations are repeated in different cases but that's just code styling, the calculation is always done only once so there is no real difference if the calculation is written only once before the if statements. It could just save a really small amount of memory when deploying the contract. No impact on users.

Possible limits for the percentage value are not specified

Low | TL – 11 | Unresolved

The value of the input parameter of the function, on the basis of which the value of the variable *_maxTxAmount* is calculated, must be within the range of possible percentages, that is, from 0 to 100. In addition, when setting values for *_taxFee*, *_liquidityFee*, *_foundationFee* such a check for interest must also be added.

Recommendation:

Add the requires to functions that set percentage values. For example, *require(maxTxPercent <= 100, "maxTxPercent can't exceed 100%")*;

Comment from the project team:

I agree with this one, it's not a must but it's good to have it. In theory, this method could be updated only by the owner and when we renounce the ownership nobody will be able to set them to a bigger percent. No impact on users.

Naming convention

Informational | MI – 01 | Unresolved

According to the solidity style guide non-external functions and variables should be named using the single leading underscore prefix. Examples: *_addLiquidity*, *_swapTokensForEth*.

This convention is suggested for non-external functions and state variables (private or internal).

When designing a smart contract, the public-facing API (functions that can be called by any account) is an important consideration. Leading underscores allow you to immediately recognize the intent of such functions, but more importantly, if you change a function from non-external to external (including public) and rename it accordingly, this forces you to review every call site while renaming. This can be an important manual check against unintended external functions and a common source of security vulnerabilities (avoid find-replace-all tooling for this change).

There are several generally accepted rules that are used when specifying a variable name:

1) Constants should be named with all capital letters with underscores separating words;
2) Names of variables with private and internal visibility must begin with _;
3) Names of variables with public visibility should simply begin with a letter, not _.

Recommendation:

Consider changing the naming style for non-external functions and state variables (private or internal) according to the style guide recommendation.

Comment from the project team:

Correct, we usually follow the "_" pattern for private functions but here we missed it. No impact on users.

Functions order at contracts

Informational | MI – 02 | Unresolved

The functions in contracts ArtToken are not grouped according to their visibility and order.

Functions should be grouped according to their visibility and ordered in the following way:

- *constructor*;
- *receive function (if exists)*;
- *fallback function (if exists)*;
- *external*;
- *public*;
- *internal*;
- *private*.

Recommendation:

Consider changing functions order according to solidity documentation and style guide. Documentation: Order of Functions.

Comment from the project team:

This is more or less out of the convention and I partially agree with this. No impact on users.

Lack of NatSpec annotations

Informational | MI – 03 | Unresolved

Smart contracts are not covered by the NatSpec annotations.

Recommendation:

Consider to cover SCs with NatSpec annotation.

Comment from the project team:

I agree with this, we can provide the comments separately since the contract is already deployed. No impact on users.

The need for refactoring

Informational | TI – 01 | Unresolved

Most additional contracts can be imported from the @openzeppelin library. This will optimize the code and reduce its processing time. In addition, according to the basic style rules for Solidity code, each contract or interface should be separated and stored in separate files.

Recommendation:

1) divide the contracts into separate files;
2) import ready-made contracts from the @openzeppelin library: Ownable, IERC20, SafeMath, Address;
3) move the *geUnlockTime()* (also rename to *getUnlockTime()*), *lock()*, *unlock()* functions directly to the ArtToken contract with external visibility (to save gas usage);
4) create a separate interfaces folder and transfer there the interfaces that are not in the @openzeppelin library;
5) import ERC20 SC instead of the IERC20 interface, then remove functions from ArtToken SC, which are in the ERC20.

Comment from the project team:

I agree, but it will be cleaner for developer, in reality it's the same. No impact on users.

Incorrect the order of layout

Informational | TI – 02 | Unresolved

The layout contract elements in the ArtToken contract are not logically grouped. The contract elements should be grouped and ordered in the following way:

- Pragma statements;
- Import statements;
- Interfaces;
- Libraries;
- Contract.

Inside each contract, library or interface, use the following order:

- Library declarations (using statements);
- Constant variables;
- Type declarations;
- State variables;
- Events;
- Modifiers;
- Functions.

Ordering helps readers to navigate the code and find the elements more quickly.

Recommendation:

Consider changing the order of layout according to solidity documentation: Order of Layout.

Comment from the project team:

I agree, it will be cleaner and the readability would be better if this order is changed. No impact on users.

Incorrect modifier's call

Informational | TI – 03 | Unresolved

The functions in contract ArtToken call the modifier as a function, although this is not exactly the right approach, even from a stylization point of view.

Recommendation:

Call the modifier differently than the function. Example: should be *onlyFeeManager*, but not *onlyFeeManager()*.

Comment from the project team:

I agree, it should be called as modifier rather than as a function, but there is no real difference. No impact on users.

The commented code

Informational | TI – 04 | Unresolved

The *excludeFromReward()* function contains a commented out line of code that is not used anywhere and is redundant.

Recommendation:

Remove the unused commented line.

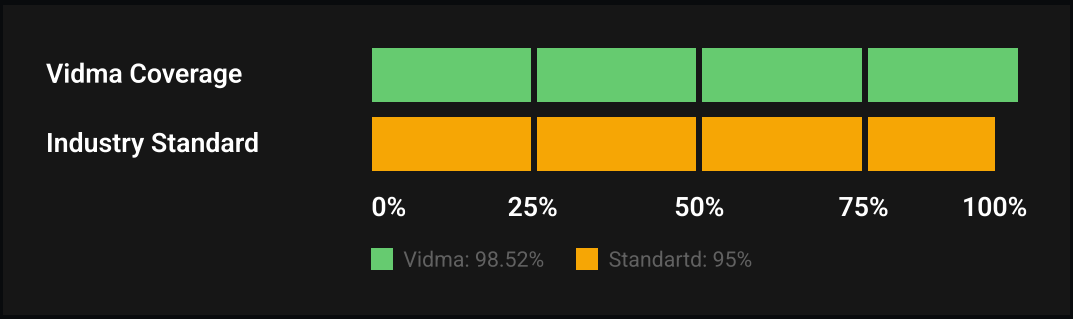
Comment from the project team:

Redundant comment, should have been deleted. No impact on users.

TEST RESULTS

To verify the security of contracts and the performance, a number of integration tests were carried out using the Hardhat testing framework.

In this section, we provide tests written by Vidma auditors.



It is important to note that Vidma auditors do not modify, edit or add tests to the existing tests provided in the ARTT Network repository. We write totally separate tests with code coverage of a minimum of 95% to meet the industry standards.

Tests written by Vidma auditors

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
contracts\	98.52	90.91	100.00	99.49
ArtToken.sol	98.52	90.91	100.00	99.49
contracts\interfaces\	100.00	100.00	100.00	100.00
IUniswapV2Factory.sol	100.00	100.00	100.00	100.00
IUniswapV2Router01.sol	100.00	100.00	100.00	100.00
IUniswapV2Router02.sol	100.00	100.00	100.00	100.00
All Files	**98.52**	**90.91**	**100.00**	**99.49**

Test Results

```
Contract: ArtToken
  ArtToken Constructor Phase Test Cases
    ✓ should set the _rOwned value for the owner correctly (90ms)
    ✓ should set the router correctly (66ms)
    ✓ should create the uniswap's pair correctly (156ms)
    ✓ should set the foundation wallet correctly
    ✓ should set the fee manager correctly
    ✓ should exclude the owner and this contract from fee
      correctly
    ✓ should emit the event while deploying SC correctly
  ArtToken Set/Get Functions Phase Test Cases
    ✓ should get name of the token correctly
```

- ✓ should get symbol of the token correctly
- ✓ should get decimals of the token correctly
- ✓ should get the total supply correctly
- ✓ should approve the tokens correctly (58ms)
- ✓ should not approve the tokens correctly if the recipient is zero's address (52ms)
- ✓ should increase allowance correctly (46ms)
- ✓ should decrease allowance correctly (58ms)
- ✓ should exclude the account from fee correctly by the fee manager (85ms)
- ✓ should not exclude the account from fee correctly if caller is not the fee manager
- ✓ should include the account in fee correctly by the fee manager (48ms)
- ✓ should not include the account in fee correctly if caller is not the fee manager (38ms)
- ✓ should set percent of the tax fee correctly by the fee manager (45ms)
- ✓ should not set percent of the tax fee correctly if caller is not the fee manager (38ms)
- ✓ should set percent of the liquidity fee correctly by the fee manager
- ✓ should not set percent of the liquidity fee correctly if the caller is not the fee manager
- ✓ should set foundation fee correctly by the fee manager (58ms)
- ✓ should not set foundation fee correctly if the caller is not the fee manager
- ✓ should set foundation wallet correctly by the owner
- ✓ should not set foundation wallet correctly if caller is not the owner
- ✓ should set fee manager correctly by the owner (54ms)
- ✓ should not set fee manager correctly if caller is not the owner
- ✓ should set percent of the max tx correctly (39ms)
- ✓ should not set percent of the max tx correctly if caller is not the owner (38ms)
- ✓ should set value of the 'swapAndLiquifyEnabled' correctly by the owner (48ms)
- ✓ should not set value of the 'swapAndLiquifyEnabled' correctly if caller is not the owner
- ✓ should exclude the account from rewards without _rOwned by the owner (53ms)

- ✓ should not exclude the account from rewards if caller is not the owner
- ✓ should not exclude the account from rewards if the account is already excluded (67ms)
- ✓ should exclude the account from rewards with _rOwned correctly (78ms)
- ✓ should include account in rewards correctly (43ms)
- ✓ should not include the account in rewards if caller is not the owner
- ✓ should not include the account in rewards if the account is already included (38ms)
- ✓ should deliver correctly (58ms)
- ✓ should not deliver if the account isn't excluded (64ms)
- ✓ should return reflection from the tokens with deduct transfer fee correctly
- ✓ should return reflection from the tokens without deduct transfer fee correctly (47ms)
- ✓ should not return reflection from the tokens if amount more than supply
- ✓ should return the tokens from reflection correctly
- ✓ should not return the tokens from reflection if amount more than total reflections

ArtToken Transfer Functions Phase Test Cases

- ✓ should not transfer the tokens correctly if the sender is zero's address (51ms)
- ✓ should not transfer the tokens correctly if the recipient is zero's address (39ms)
- ✓ should not transfer the tokens correctly if amount is zero (47ms)
- ✓ should not transfer amount of tokens if amount exceeds the maxTxAmount (65ms)
- ✓ should transfer the tokens from the account that is excluded from fee correctly (125ms)
- ✓ should transfer the tokens from the account that is excluded from fee correctly when fee already is zero (134ms)
- ✓ should transfer the tokens from the account that is excluded from rewards correctly (153ms)
- ✓ should transfer the tokens to the account that is excluded from rewards correctly (146ms)
- ✓ should transfer the tokens if the both accounts are excluded correctly (188ms)

```
    ✓ should standard transfer amount of the tokens correctly if
      not satisfying the threshold and with fee (149ms)
    ✓ should standard transfer amount of the tokens correctly if
      satisfying the threshold (2703ms)
    ✓ should not transfer amount of the tokens correctly if the
      amount exceeds maxTxAmount (1684ms)
  ArtToken Fee Functions Phase Test Cases
    ✓ should transfer the tokens depending on transfer fee
      correctly when taxFee = 0, liquidityFee = 2 (173ms)
    ✓ should transfer tokens depending on fee correctly
      taxFee = 50, liquidityFee = 0 (175ms)
    ✓ should transfer the tokens correctly depending on the fee
      while add liquidity (2598ms)

62 passing (2m)
```

We are delighted to have a chance to work with the ARTT Network team and contribute to your company's success by reviewing and certifying the security of your smart contracts.

The statements made in this document should be interpreted neither as investment or legal advice, nor should its authors be held accountable for decisions made based on this document.

Vidma is a security audit company helping crypto companies ensure their code and products operate safely and as intended, enabling founders to sleep soundly at night. We specialize in auditing DeFi protocols, layer one protocols, and marketplace solutions. Our team consists of experienced and internationally trained specialists. Our company is based in Ukraine, known for its strong engineering, cryptography, and cybersecurity culture.

Website: **vidma.io**
Email: **security@vidma.io**

VIDMA