Security Assessment

Wixpool

Aug 21st, 2022

Table of Contents

Summary

Overview

- Project Summary
- Audit Summary
- Vulnerability Summary
- Audit Scope

Findings

- ATC-01 : Unlocked Compiler Version
- ATC-02 : Function & Variable Visibility
- ATC-03 : Inefficient Greater-Than Comparison w/ Zero
- ATC-04 : Unconventional Naming of `public` Variables
- ATC-05 : EIP712 Adjustment
- ATC-06 : Redundant Assignments & `_setupDecimals` Invocation
- ATC-07 : Inconsistent EIP2612 Implementation
- LTA-01 : Unlocked Compiler Version
- LTA-02 : Function & Variable Visibility
- LTA-03 : Inefficient Greater-Than Comparison w/ Zero
- LTA-04 : Unconventional Naming of `public` Variables
- LTA-05 : Truncation of LEND Migration Amount
- LTA-06 : Redundant `SafeMath` Utilization
- LTA-07 : Purpose of `initializer` Modifier
- VIC-01 : Function & Variable Visibility

Appendix

Disclaimer

About

Summary

This report has been prepared for Wixpool to discover issues and vulnerabilities in the source code of the Wixpool project as well as any contract dependencies that were not part of an officially recognized library . A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

-
- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client. Cross referencing contract structure.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices.

We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Wixpool
Platform	Ethereum
Language	Solidity
Codebase	https://wixpool.gitbook.com/wixpool
Commit	

Audit Summary

Delivery Date	Aug 21, 2022
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
Major	0	0	0	0	0	0
Medium	1	0	0	0	0	1
Minor	1	0	0	1	0	0
Informational	13	0	0	7	2	7
Discussion	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
ATC	Wixpool.sol	59e910fd59514a379bc0dfd9eb030821b7624411
ERC	ERC20.sol	c543bae32c32b503212dc0186e792b3e7e64e5fd
LTW	LendToWixpoolMigrator.sol	e316261f318659c6af36bed651ae52bb026f0c49
VIC	VersionedInitializable.sol	e5a8b87b8f89b6c5f28b25be9f7499d14b5b6ff3

Introduction

CertiK team was contracted by the Wixpool team to audit the design and implementation of their Wixpool token smart contract , its compliance with the multiple EIPs it is meant to implement as well as its LEND to Wixpool migrator contract.

The audited source code link is:

- **Wixpool Rate Stabilizer Code:** https://wixpool.gitbook.com/wixpool

The goal of this audit was to review the Solidity implementation for its business model, study potential security vulnerabilities, its general design, and architecture, and uncover bugs that could compromise the software in production.

The findings of the initial audit have been conveyed to the team behind the contract implementations and the source code is expected to be re-evaluated before another round of auditing has been carried out.

The Wixpool team swiftly dealt with and responded to all of the Exhibits we laid out in the first round of audit, providing us with a new commit to continue the audit from as well as a dedicated branch for resolving the vulnerability detailed in Exhibit 9.

A final revision of the codebase was conducted on another commit hash to make sure the branch was merged properly and to ensure that all Exhibits have been uniformly dealt with or responded to. The SHA-1 digests of each file in scope can be observed in the paragraph below. 5

These digests were generated by running the “git ls-files -s” command for each file and copying the generated SHA-1 digest. The reason behind choosing this method of generation is the OS agnostic nature of the generated digest as a conventional SHA-1 digest would differ between OSes due to the way the newline character is represented.

Documentation

The sources of truth regarding the operation of the contracts in scope were comprehensive and **greatly aided our efforts to audit the project as well as generally increase the legibility of the codebase**. To help aid our understanding of each contract’s functionality we referred to in-line comments and naming conventions.

These were considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the Wixpool team or reported an issue.

Summary

The codebase of the project is a typical EIP20 implementation with additional support for EIP712 and EIP2612. Additionally, as the contract is meant to be deployed via a proxy mechanism, the EIP being used is EIP1967.

Certain optimization steps that we pinpointed in the source code mostly referred to coding standards and inefficiencies, however, **1 minor and 1 medium severity vulnerability were identified during our audit that solely concerns the specification**. The codebase of the project strictly adheres to the standards and interfaces imposed by the OpenZeppelin open-source libraries and **can be deemed to be of high security and quality**.

Certain discrepancies between the expected specification and its implementation were identified and were relayed to the team, however, the exploitation surface is minimal and relates to a type of phishing attack against the variable sequence a user/application expects.

Recommendations

Overall, the codebase of the contracts should be refactored to assimilate the findings of this the report, enforce linters and/or coding styles as well as correct any spelling errors and mistakes that appear throughout the code **to achieve a high standard of code quality and security**.

Conclusion

Our findings were either fully assimilated in the codebase or were omitted justifiably by the Wixpool team in a very short timeframe, indicating **a dedicated and well-versed team** in the Solidity space **capable of maintaining a secure and exemplary codebase**.

Findings

Critical	0 (0.00%)
Major	0 (0.00%)
Medium	1 (5.56%)
Minor	1 (5.56%)
Informational	16 (88.89%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
ATC-01	Unlocked Compiler Version	Language Specific	Informational	Resolved
ATC-02	Function & Variable Visibility	Coding Style	Informational	Resolved
ATC-03	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	Partially Resolved
ATC-04	Unconventional Naming of `public` Variables	Coding Style	Informational	Acknowledged
ATC-05	EIP712 Adjustment	Coding Style	Informational	Acknowledged
ATC-06	Redundant Assignments & `_setupDecimals` Invocation	Gas Optimization	Informational	Resolved
ATC-07	Inconsistent EIP2612 Implementation	Logical Issue	Medium	Resolved
ATC-08	Redundant `SafeMath` Utilization	Gas Optimization	Informational	Acknowledged
ATC-09	Declaration Optimization	Gas Optimization	Informational	Resolved
DTH-01	Inexistent Access Control	Logical Issue	Informational	Acknowledged
LTA-01	Unlocked Compiler Version	Language Specific	Informational	Resolved
LTA-02	Function & Variable Visibility	Coding Style	Informational	Resolved
LTA-03	Inefficient Greater-Than Comparison w/ Zero	Gas Optimization	Informational	Partially Resolved
LTA-04	Unconventional Naming of `public` Variables	Coding Style	Informational	Acknowledged

ID	Title	Category	Severity	Status
LTA-05	Truncation of LEND Migration Amount	Mathematical Operations	Minor	Acknowledged
LTA-06	Redundant `SafeMath` Utilization	Gas Optimization	Informational	Acknowledged
LTA-07	Purpose of `initializer` Modifier	Gas Optimization	Informational	Acknowledged
VIC-01	Function & Variable Visibility	Coding Style	Informational	Resolved

ATC-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	Wixpool.sol: 1	Resolved

Description

The smart contract "pragma" statements regarding the compiler version indicate that version 0.6.10 or higher should be utilized

Recommendation

We advise that the compiler version is locked at version 0.6.10 or whichever Solidity version higher than that satisfies the requirements of the codebase as an unlocked compiler version can lead to discrepancies between compilations of the same source code due to compiler bugs and differences.

Alleviation

As per our recommendation, the Wixpool team locked both contracts at version 0.6.10 aiding in pinpointing compiler bugs should they occur.

ATC-02 | Function & Variable Visibility

Category	Severity	Location	Status
Coding Style	Informational	Wixpool.sol: 31, 121~123	Resolved

Description

The Wix team has applied an adjusted version of the Initializable trait defined in the OpenZeppelin libraries whereby a `revision` number is utilized for discerning between initialize deployments.

To achieve this, a `getRevision` function is meant to be implemented as an `internal` function within derivative contracts of `VersionedInitializable`. For this purpose, Wix has defined these functions as well as declared a `REVISION` constant that is publicly accessible.

Recommendation

We advise that the function signature of `getRevision` is instead converted to `public`. As constant variables are meant to conform to the UPPER_CASE_FORMAT, a getter function in the form of `getRevision` is more legible and sensible than invoking “REVISION” from off-chain applications.

Alleviation

The Wixpool team responded to this Exhibit by stating that the internal styling guideline they conform to utilizes auto-generated getters instead of user-defined ones as they are less error prone and less verbose and as such, this Exhibit is inapplicable.

ATC-03 | Inefficient Greater-Than Comparison w/ Zero

Category	Severity	Location	Status
Gas Optimization	Informational	Wixpool.sol: 135	Partially Resolved

Description

The lines above conduct a greater-than `>` comparison between unsigned integers and the value literal `0`.

Recommendation

As unsigned integers are restricted to the positive range, it is possible to convert this check to an inequality `!=` reducing the gas cost of the functions. Additionally, L62 of `migrateFromLEND` in `LendToWixpMigrator` could instead internally call the function `migrationStarted` which would have to be converted to `public`. This would ensure consistency in the checks and enable additional checks to be imposed on `migrationStarted` in the future if necessary. If no subsequent development is expected, the last point can be safely ignored as it would lead to a miniscule increase in the gas cost of `migrateFromLEND`.

Alleviation

The Wixpool team evaluated this Exhibit and proceeded with applying the greater-than to inequality optimization on the aforementioned lines and chose to avoid declaring `migrationStarted` as `public` and retaining L62 as is.

ATC-04 | Unconventional Naming of `public` Variables

Category	Severity	Location	Status
Coding Style	Informational	Wixpool.sol: 34, 36, 38, 43, 45, 47	Acknowledged

Description

The variables of L34, L36, L38 and L43 are `public` yet prefixed with an underscore. The `DOMAIN_SEPARATOR` defined in L45 is listed as `public` yet follows the naming convention of `constant` and `immutable` variables and the `PERMIT_TYPEHASH` is declared as `public` correctly following the UPPER_CASE_FORMAT but being illegible for off-chain applications via its compiler-generated getter.

Recommendation

We advise that the first four variable declarations omit the underscore, the `DOMAIN_SEPARATOR` variable is converted to the camelCase format and that a dedicated getter is defined for the `PERMIT_TYPEHASH` variable which should be set to either `internal` or `private`.

If Exhibit 7 is followed, the point about `DOMAIN_SEPARATOR` should be ignored. Additionally, the variable of L38 could be renamed to `snapshotsLength` instead of `countsSnapshots` to aid in understanding its purpose.

Alleviation

As per the Wixpool's response to Exhibit 2 and Exhibit 4, they chose to retain the naming convention as it is compliant with the team's internal styling guidelines.

ATC-05 | EIP712 Adjustment

Category	Severity	Location	Status
Coding Style	Informational	Wixpool.sol: 45, 65~71	Acknowledged

Description

The `DOMAIN_SEPARATOR` variable of EIP712 can be converted to `immutable` and set within the `constructor` by utilizing the EIP1344 which was partially created for EIP712.

Recommendation

The assignment of L65 - L71 should instead be moved to the `constructor` of the contract. As at its current state it relies on the `chainId` parameter, this can be derived using EIP1344 via assembly by using the `chainid()` opcode. A simple example of utilization would be to declare a `uint256` variable titled `chainId`, declare an assembly block and assign the result of `chainid()` via the `:=` operator to the variable `chainId` and subsequently use it as per the original assignment. Additionally, the statement of L68 should instead use the current `REVISION` of the contract rather than the string literal `1` to ensure that updates in the codebase are reflected in the EIP712 domain separator.

Alleviation

The Wix team partially acknowledged this Exhibit by following our recommendation regarding retrieving the `chainId` variable of EIP1344 via assembly. Our recommendation with regards to setting the `DOMAIN_SEPARATOR` as immutable was avoided as the assignment of the variable relies on the statement `address(this)` which would differ when executed in the constructor, resulting in the address of the logic contract, and when executed in the `initialize` function, resulting in the address of the proxy contract. It is still feasible to set the variable as `immutable` by passing in the address of the proxy to the constructor of the `Wixpool` logic contract, however we leave this optimization up to the discretion of the Wix team as they may wish to avoid linking the logic contract with the proxy contract directly. The rationale behind this optimization is that gas cost will be greatly reduced for `permit` invocations as they would not require to read from state and would instead read the resulting literal on the code itself as that is what the `immutable` trait does. Additionally, our point with regards to utilizing the `REVISION` variable instead of the string literal `1` on L75 still stands.

Similarly to Exhibit 5, the time constraints that the Wixp team had to comply with did not permit major changes in the codebase meaning that the team decided not to alter the deployment procedure of the contracts after weighing the benefit.

ATC-06 | Redundant Assignments & `_setupDecimals` Invocation

Category	Severity	Location	Status
Gas Optimization	Informational	Wixpool.sol: 72~74	Resolved

Description

The function `_setupDecimals` is called to set the `decimals` of the ERC20 interface to 18. Internally, the ERC20 interface assigns the literal `18` by default to the value of `decimals`. Additionally, the values of `NAME` and `SYMBOL` are set to `_name` and `_symbol` respectively whereas the constructor of `ERC20` is called on L51 which assigns those values as well.

Recommendation

All aforementioned statements can be safely omitted as they are duplicate assignments.

Alleviation

The Wix team articulated how both contracts are meant to be utilized via a proxy and as such, the points with regards to the redundant assignments are void. The team addressed the `_setupDecimals` invocation by utilizing a constant `DECIMALS` variable instead of the value literal `18`.

ATC-07 | Inconsistent EIP2612 Implementation

Category	Severity	Location	Status
Logical Issue	Medium	Wixpool.sol: 47, 91~116	Resolved

Description

The `typehash` of the permit function as well as its internal statements do not conform to the EIP2612 specification as the `expiration` and `value` variables are swapped in between implementations.

Recommendation

We advise that EIP2612 is conformed to the letter, as the current implementation is misleading. The reason it is misleading is because the function signature matches the specification's ABI (`address, address, uint256, uint256, uint8, bytes32, bytes32`) yet the `typehash` utilized is different as well as the way the values are utilized. This could lead to a generic EIP2612 implementation misleading users to input the `value` to be transacted and the `deadline` whilst those two will be utilized in place of one another within the codebase. As EIPs are meant to streamline the way smart contracts interface with off chain applications on the Solidity ecosystem, we advise that the lines of this Exhibit are amended accordingly to conform to EIP2612.

Alleviation

The Wixpool team fully addressed this Exhibit in a dedicated merge-request as it necessitated changes throughout the codebase as well as the test suites.

LTA-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	LendToWixpoolMigrator.sol: 1	Resolved

Description

The smart contract "pragma" statements regarding the compiler version indicate that version 0.6.10 or higher should be utilized

Recommendation

We advise that the compiler version is locked at version 0.6.10 or whichever Solidity version higher than that satisfies the requirements of the codebase as an unlocked compiler version can lead to discrepancies between compilations of the same source code due to compiler bugs and differences.

Alleviation

As per our recommendation, the Wixpool team locked both contracts at version 0.6.10 aiding in pinpointing compiler bugs should they occur.

LTA-02 | Function & Variable Visibility

Category	Severity	Location	Status
Coding Style	Informational	LendToWixpoolMigrator.sol: 20, 74~76	Resolved

Description

The Wix team has applied an adjusted version of the Initializable trait defined in the OpenZeppelin libraries whereby a `revision` number is utilized for discerning between initialize deployments.

To achieve this, a `getRevision` function is meant to be implemented as an `internal` function within derivative contracts of `VersionedInitializable`. For this purpose, Wix has defined these functions as well as declared a `REVISION` constant that is publicly accessible.

Recommendation

We advise that the function signature of `getRevision` is instead converted to `public`. As constant variables are meant to conform to the UPPER_CASE_FORMAT, a getter function in the form of `getRevision` is more legible and sensible than invoking “REVISION” from off-chain applications.

Alleviation

The Wix team responded to this Exhibit by stating that the internal styling guideline they conform to utilizes auto-generated getters instead of user-defined ones as they are less error prone and less verbose and as such, this Exhibit is inapplicable.

LTA-03 | Inefficient Greater-Than Comparison w/ Zero

Category	Severity	Location	Status
Gas Optimization	Informational	LendToWixMigrator.sol: 52, 62	Partially Resolved

Description

The lines above conduct a greater-than `>` comparison between unsigned integers and the value literal `0`.

Recommendation

As unsigned integers are restricted to the positive range, it is possible to convert this check to an inequality `!=` reducing the gas cost of the functions. Additionally, L62 of `migrateFromLEND` in `LendToWixpMigrator` could instead internally call the function `migrationStarted` which would have to be converted to `public`. This would ensure consistency in the checks and enable additional checks to be imposed on `migrationStarted` in the future if necessary. If no subsequent development is expected, the last point can be safely ignored as it would lead to a miniscule increase in the gas cost of `migrateFromLEND`.

Alleviation

The Wix team evaluated this Exhibit and proceeded with applying the greater-than to inequality optimization on the aforementioned lines and chose to avoid declaring `migrationStarted` as `public` and retaining L62 as is.

LTA-04 | Unconventional Naming of `public` Variables

Category	Severity	Location	Status
Coding Style	● Informational	LendToWixpoolMigrator.sol: 19, 22	ⓘ Acknowledged

Description

The variable of L19 (`LEND_Wixp_RATIO`) properly conforms to the naming convention of `immutable` and `constant` variables yet is declared `public`. The variable of L22 does not conform to the naming convention of publicly accessible variables as it is prefixed with an underscore.

Recommendation

For the former, we advise that it is instead set to `private` or `internal` and a dedicated getter function is set that allows off-chain applications to retrieve the ratio. The Solidity compiler automatically generates getter functions for `public` variables and as such, the above statements are equivalent to the current code in terms of gas impact. For the latter, we advise that the underscore is simply removed from the variable declaration.

Alleviation

The Wix team proceeded with not changing the aforementioned naming conventions of the variables as the former, L19, is answered by the `Alleviation` chapter of Exhibit 2, and the latter once again is a result of Wix's internal styling guide mandating that the `_` prefix is set on all state variables regardless of visibility

LTA-05 | Truncation of LEND Migration Amount

Category	Severity	Location	Status
Mathematical Operations	Minor 67	LendToWixpMigrator.sol: 64~	Acknowledged

Description

The `migrateFromLEND` function accepts an `amount` input in the form of a `uint256`, transfers the full amount of LEND to the contract and subsequently transfers the Wixp equivalent to the sender by dividing the amount with the `LEND_Wixp_RATIO` variable.

Recommendation

The division with `LEND_WIXP_RATIO` will always truncate if `LEND_WIXP_RATIO` is different than 1 and would lead to the truncated amount being permanently locked within `LendToWixpMigrator`, thus causing the fi nal conversion ratio to be different than the imposed one. We advise that the modulo of `amount` with `LEND_WIXP_RATIO` is subtracted from itself to calculate the exact amount of LEND that will be migrated and prevent trailing LEND from being locked up in the contract.

Alleviation

The documentation material provided to us by Wix indicated that the team was aware of this truncation mechanism and that the solutions they have investigated are not ideal for this type of issue due to their complexity. The important thing here to note is that trailing LEND as well as trailing Wix will be locked up in the contract as the LEND token is expected to be fully converted to the Wix token, meaning truncations will lead to units of both LEND and Wix remaining out of circulation forever. As this is undesirable behavior, a novel solution we propose would be to instead store the surplus `LEND` within a variable of the contract that is carried over to the next conversion.