CFX Token, Swap & Crowdsale

Security Assessment

April 22nd, 2021

For :
Chain Partners

Audited By:
Angelos Apostolidis @ CertiK
angelos.apostolidis@certik.org

Reviewed By:
Camden Smallwood @ CertiK
camden.smallwood@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	CFX token, swap & crowdsale
Description	Typical ERC20 token, swap and crowdsale implementations with enhanced features.
Platform	Ethereum; Solidity, Yul
Codebase	1. CFX Swap 2. CFX Crowdsale
Commits	Pre-audit: 1. e22058367e17cdcea7c0551647796c60ef729575 2. 9bbe2eb711dfd2918a1c34ea2ca6ad141b66d571 Post-audit: 1. 2128613fa5f59c06b180603f962b598bc7ec52a7 2. 067b74ca38a566cd571db908f3769c1cbaed2c1a

Audit Summary

Delivery Date	April 22nd, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	February 8th, 2021 - April 22nd, 2021

Vulnerability Summary

Total Issues	29
Total Critical	0
Total Major	0
Total Medium	0
Total Minor	6
Total Informational	23

Executive Summary

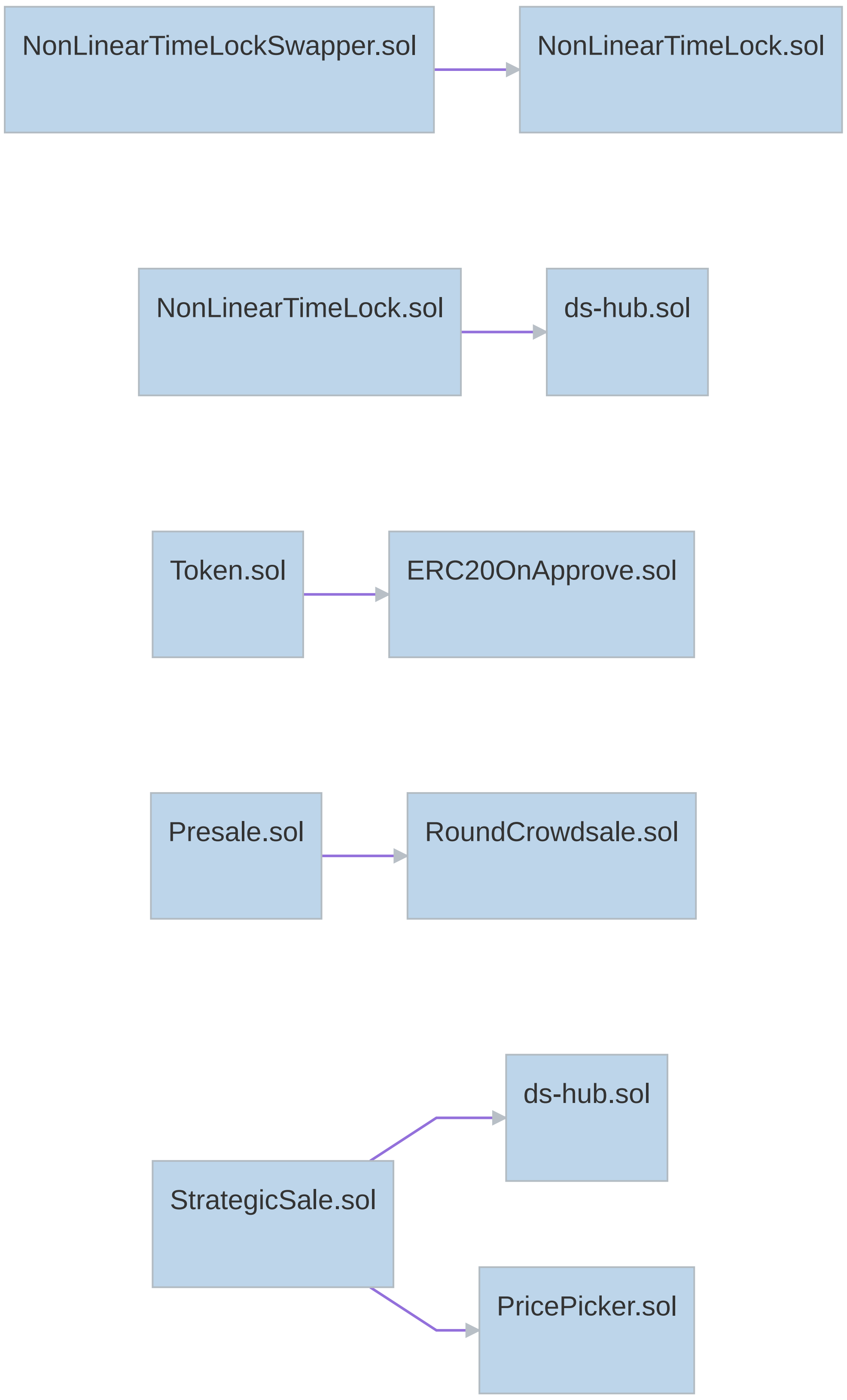
This report represents the results of CertiK's engagement with Chain Partners on their implementation of the token, swap and crowdsale smart contracts.

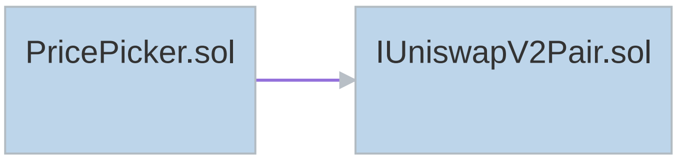
No notable vulnerabilities were identified in the codebase and it makes use of the latest security principles and style guidelines. There were certain optimizations observed as well as security principles that can optionally be applied to the codebase to fortify the codebase to a greater extent.

Files In Scope

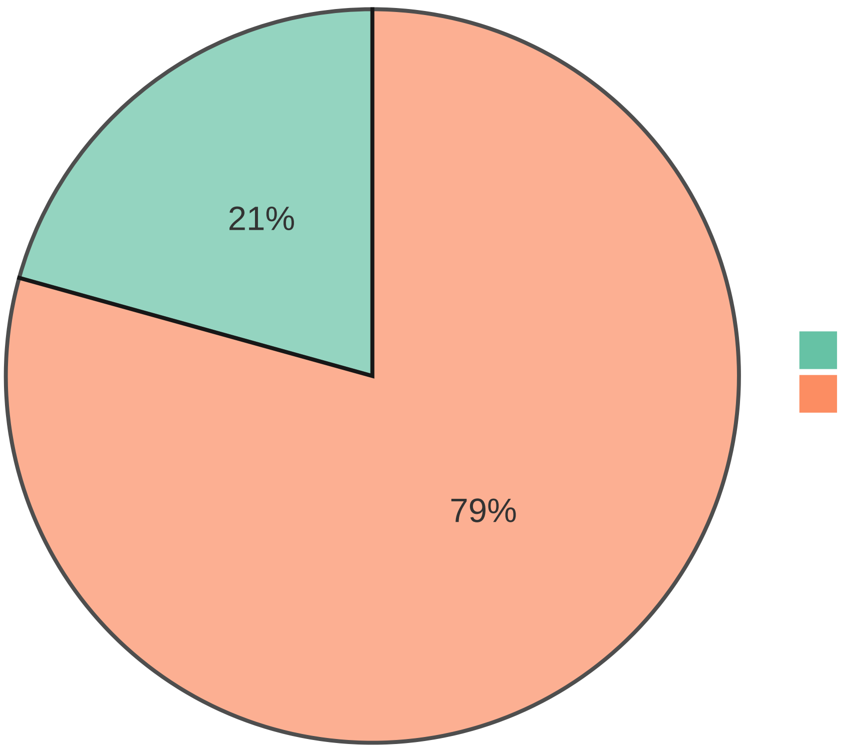
ID	Contract	Location
ERC	ERC20OnApprove.sol	contracts/token/ERC20OnApprove.sol
NLT	NonLinearTimeLock.sol	contracts/timelock/NonLinearTimeLock.sol
NLL	NonLinearTimeLockSwapper.sol	contracts/swapper/NonLinearTimeLockSwapper.sol
TOK	Token.sol	contracts/token/Token.sol
PRE	Presale.sol	contracts/sale/Presale.sol
PPR	PricePicker.sol	contracts/sale/PricePicker.sol
RCE	RoundCrowdsale.sol	contracts/sale/RoundCrowdsale.sol
SSE	StrategicSale.sol	contracts/sale/StrategicSale.sol

File Dependency Graph




PricePicker.sol
IUniswapV2Pair.sol

Findings



ID	Title	Type	Severity	Resolved
TOK-01	Potentially `admin`-less Contract	Logical Issue	Minor	✓
TOK-02	Unlocked Compiler Version	Language Specific	Informational	❗
ERC-01	Function Visibility Optimization	Gas Optimization	Informational	✓
ERC-02	Unlocked Compiler Version	Language Specific	Informational	❗
NLT-01	Redundant `array` Look-Up	Gas Optimization	Informational	✓
NLT-02	Typo in the NatSpec Comments	Coding Style	Informational	✓
NLT-03	Potential Re-Entrancy	Volatile Code	Minor	❗
NLT-04	Unlocked Compiler Version	Language Specific	Informational	❗
NLT-05	Inexistant Input Sanitization	Volatile Code	Minor	✓
NLL-01	Redundant `array` Look-Up	Gas Optimization	Informational	✓
NLL-02	Potential Re-Entrancy	Volatile Code	Minor	✓
NLL-03	Unused Parameter	Coding Style	Informational	✓
NLL-04	Typo in the Error Message	Coding Style	Informational	✓
NLL-05	Ambiguous Parameter Type	Gas Optimization	Informational	✓
NLL-06	Redundant Statement	Coding Style	Informational	✓
NLL-07	Change to a `modifier`	Coding Style	Informational	✓
NLL-08	Unlocked Compiler Version	Language Specific	Informational	❗
NLL-09	Inexistant Input Sanitization	Volatile Code	Minor	✓

ID	Title	Type	Severity	Resolved
NLL-10	Inexistant Input Sanitization	Volatile Code	Minor	✓
PRE-01	Unlocked Compiler Version	Language Specific	Informational	❗
PPR-01	Unlocked Compiler Version	Language Specific	Informational	❗
PPR-02	Change to a `pure` Function	Gas Optimization	Informational	✓
PPR-03	Redundant Variable	Gas Optimization	Informational	✓
RCE-01	Unlocked Compiler Version	Language Specific	Informational	❗
RCE-02	User-Defined Getters	Gas Optimization	Informational	❗
RCE-03	Redundant `array` Look-Up	Gas Optimization	Informational	✓
RCE-04	Ambiguous Function Visibility	Gas Optimization	Informational	❗
SSE-01	Unlocked Compiler Version	Language Specific	Informational	❗
SSE-02	Return Variable Utilization	Gas Optimization	Informational	✓

TOK-01: Potentially `admin`-less Contract

Type	Severity	Location
Logical Issue	Minor	Token.sol General

Description:

The `admin` of the system can revoke his role, leading to a static list of pausers and minters.

Recommendation:

We advise to consider introducing a temporary admin in case the admin uses the `revokeRole()` function instead of the `changeAdminRole()` one.

Alleviation:

The development team opted to consider our references and introduced admin roles for both the pausers and minters of the system.

TOK-02: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	Token.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

ERC-01: Function Visibility Optimization

Type	Severity	Location
Gas Optimization	Informational	ERC20OnApprove.sol L25-L33

Description:

The linked function is declared as `public`, contains array function arguments and is not invoked in any of the contract's contained within the project's scope.

Recommendation:

We advise that the functions' visibility specifiers are set to `external` and the array-based arguments change their data location from `memory` to `calldata`, optimizing the gas cost of the function.

Alleviation:

The development team opted to consider our references and changed the data location of the arguments to `calldata`.

ERC-02: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	ERC20OnApprove.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

NLT-01: Redundant `array` Look-Up

Type	Severity	Location
Gas Optimization	Informational	NonLinearTimeLock.sol L58, L63, L135

Description:

The linked `for` conditionals redundantly look-up respective arrays on each iteration.

Recommendation:

We advise to introduce a local variable before the `for` loop block, initialize it with the `length` member of the specific array and use this variable in the conditional.

Alleviation:

The development team opted to consider our references and introduced local variables before the linked `for` loops, as proposed.

NLT-02: Typo in the NatSpec Comments

Type	Severity	Location
Coding Style	Informational	NonLinearTimeLock.sol L79-L82

Description:

There is a typo in the NatSpec comment section.

Recommendation:

We advise to update the linked comments.

Alleviation:

The development team opted to consider our references and updated the NatSpec comments of the `claim()` function.

NLT-03: Potential Re-Entrancy

Type	Severity	Location
Volatile Code	Minor	NonLinearTimeLock.sol L89-L90

Description:

The re-entrancy in the `claim()` function can modify the `claimed` state variable due to the transfer taking place after it, which will affect the values returned from the `claimable()` and `claimableAt()` functions (L109, L118).

Recommendation:

We advise to update the `claimed` state variable after the transfer is executed.

Alleviation:

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase, as the `claim()` function implementation follows the `checks-effects-interactions pattern`.

NLT-04: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	NonLinearTimeLock.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

NLT-05: Inexistant Input Sanitization

Type	Severity	Location
Volatile Code	Minor	NonLinearTimeLock.sol L39-L44, L83-L93

Description:

The `constructor()` function lacks a check on the `beneficiary_` parameter before assigning it to the `beneficiary` state variable. This is also unchecked in the `claim` function before transferring to the `beneficiary` address.

Recommendation:

We advise to add a `require` statement checking against the zero address.

Alleviation:

The development team opted to consider our references and added a `require` statement in the `constructor`, checking `beneficiary_` against the zero address .

NLL-01: Redundant `array` Look-Up

Type	Severity	Location
Gas Optimization	Informational	NonLinearTimeLockSwapper.sol L77, L82

Description:

The linked `for` conditionals redundantly look-up respective arrays on each iteration.

Recommendation:

We advise to introduce a local variable before the `for` loop block, initialize it with the `length` member of the specific array and use this variable in the conditional.

Alleviation:

The development team opted to consider our references and introduced local variables before the linked `for` loops, as proposed.

NLL-02: Potential Re-Entrancy

Type	Severity	Location
Volatile Code	Minor	NonLinearTimeLockSwapper.sol L143-L147

Description:

The `tokenAmount` is calculated after an external call.

Recommendation:

We advise to apply the Checks-Effects-Interactions pattern.

Alleviation:

The development team opted to consider our references and declared the value of `tokenAmount` before the following external calls.

NLL-03: Unused Parameter

Type	Severity	Location
Coding Style	Informational	NonLinearTimeLockSwapper.sol L104-L115

Description:

The `data` parameter of the linked function remains unused throughout the `onApprove()` function.

Recommendation:

We advise to directly call the said parameter to silence the compiler warning. Removing it will not allow the `onApprove()` function to override the parent function.

Alleviation:

The development team opted to consider our references and directly called the said parameter to silence the compiler warning.

NLL-04: Typo in the Error Message

Type	Severity	Location
Coding Style	Informational	NonLinearTimeLockSwapper.sol L128

Description:

There is a typo in the linked error message.

Recommendation:

We advise to update the linked error message.

Alleviation:

The development team opted to consider our references and updated the linked error message.

NLL-05: Ambiguous Parameter Type

Type	Severity	Location
Gas Optimization	Informational	NonLinearTimeLockSwapper.sol L62, L63

Description:

The linked parameters as ambiguously declared with the `uint256` data type, as their purpose is to create a `Data` struct instance, where the `rate` and `startTime` struct members are declared as `uint128`.

Recommendation:

We advise to declare the linked parameters with the `uint128` data type, which allow the removal of the type convertions.

Alleviation:

The development team opted to consider our references, changed the data type of the linked arguments to `uint128` and removed the type convertions.

NLL-06: Redundant Statement

Type	Severity	Location
Coding Style	Informational	NonLinearTimeLockSwapper.sol L174

Description:

The linked statement redundantly uses `return`, as the function is not implemented to return any value.

Recommendation:

We advise to remove the `return` keyword and directly call the `claim()` function of the `NonLinearTimeLock` instance.

Alleviation:

The development team opted to consider our references, removed the `return` statement and directly invoked the `claim()` function.

NLL-07: Change to a `modifier`

Type	Severity	Location
Coding Style	Informational	NonLinearTimeLockSwapper.sol L170-L173, L182-L185, L194-L197, L206-L209, L218-L221

Description:

The linked `require` statement is repeatedly used throughout the codebase.

Recommendation:

We advise to replace the linked `require` statements with a `modifier`, to enable better code readability.

Alleviation:

The development team opted to consider our references , implemented the `onlyDeposit` modifier and utilized that in the linked functions.

NLL-08: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	NonLinearTimeLockSwapper.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

NLL-09: Inexistant Input Sanitization

Type	Severity	Location
Volatile Code	Minor	NonLinearTimeLockSwapper.sol L49-L52, L118-L161

Description:

The `constructor()` function lacks a check on the `tokenWallet_` parameter before assigning it to the `tokenWallet` state variable. This is also unchecked in the `deposit()` function before transferring to the `tokenWallet` address.

Recommendation:

We advise to add a `require` statement checking against the zero address.

Alleviation:

The development team opted to consider our references , implemented the `onlyValidAddress` modifier and utilized that for the `token_` and `tokenWallet_` addresses.

NLL-10: Inexistant Input Sanitization

Type	Severity	Location
Volatile Code	Minor	NonLinearTimeLockSwapper.sol L118-L161

Description:

The `deposit()` function lacks a check on the `beneficiary` parameter before using it to construct a `NonLinearTimeLock` on L138 and attempting to transfer from it on L143, but this is only possible if `msg.sender` is the supplied `sourceToken` (which must be non-zero) address due to the requirement on L131.

Recommendation:

We advise to add a `require` statement checking against the zero address.

Alleviation:

The development team opted to consider our references and utilized the `onlyValidAddress` modifier for the `beneficiary` address.

PRE-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	Presale.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

PPR-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	PricePicker.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

PPR-02: Change to a `pure` Function

Type	Severity	Location
Gas Optimization	Informational	PricePicker.sol L11-L13

Description:

The `src()` function does not read or write to the state of the contract.

Recommendation:

We advise to change the `view` attribute of the linked function to `pure`.

Alleviation:

The development team opted to consider our references and changed the attribute of the linked function to `pure`.

PPR-03: Redundant Variable

Type	Severity	Location
Gas Optimization	Informational	PricePicker.sol L16

Description:

The `blockTimestampLast` local variable remains unused throughout the function it was declared.

Recommendation:

We advise to directly omit the third value returned by the `getReserves()` of the `IUniswapV2Pair` library.

Alleviation:

The development team opted to consider our references and removed the redundant variable.

RCE-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	RoundCrowdsale.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

RCE-02: User-Defined Getters

Type	Severity	Location
Gas Optimization	Informational	RoundCrowdsale.sol L51-L53, L55-L57, L67-L69

Description:

The linked variables contain user-defined getter functions that are equivalent to their name barring for an underscore (`_`) prefix / suffix.

Recommendation:

We advise that the linked variables are instead declared as `public` and that they are renamed to their respective getter's name as compiler-generated getter functions are less prone to error and much more maintainable than manually written ones.

Alleviation:

The development team has acknowledged this exhibit, commenting that the linked getter functions are manually implemented to guarantee encapsulation.

RCE-03: Redundant `array` Look-Up

Type	Severity	Location
Gas Optimization	Informational	RoundCrowdsale.sol L37

Description:

The linked `for` conditional redundantly looks-up `roundEndTime` array on each iteration.

Recommendation:

We advise to introduce a local variable before the `for` loop block, initialize it with the `length` member of the `roundEndTime` array and use this variable in the conditional.

Alleviation:

The development team opted to consider our references and introduced a local variable `n`, as proposed.

RCE-04: Ambiguous Function Visibility

Type	Severity	Location
Gas Optimization	Informational	RoundCrowdsale.sol L117-L124, L131-L134, L141-L145

Description:

The linked function are declared as `internal`, yet they remain unused throughout the contract.

Recommendation:

We advise to revise the vibility specifiers of the linked functions.

Alleviation:

The development team has acknowledged this exhibit, commenting that the linked functions are implemented in such way to override the respective functions in OpenZeppelin's Crowdsale contract.

SSE-01: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	StrategicSale.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation:

The development team has acknowledged this exhibit but decided to not apply its remediation in the current version of the codebase.

SSE-02: Return Variable Utilization

Type	Severity	Location
Gas Optimization	Informational	StrategicSale.sol L67

Description:

The linked function declaration contains an explicitly named `return` variable that is not utilized within the function's code block.

Recommendation:

We advise that the linked variable is either utilized or omitted from the declaration.

Alleviation:

The development team opted to consider our references, removed the `return` statement and utilized the return variable.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.