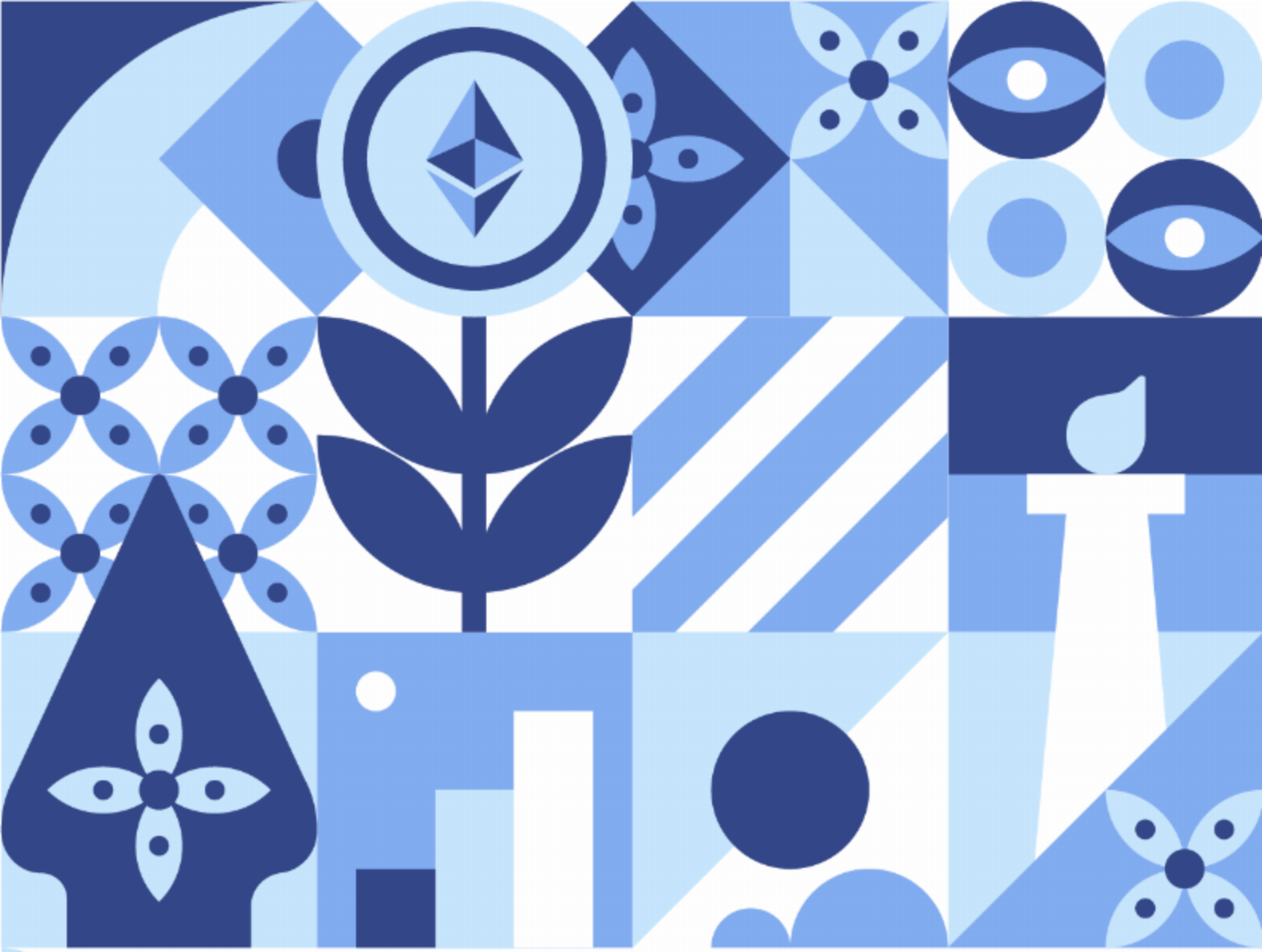

Eggs

Smart Contract Audit Report

SYSFIXED

29 Jan 2023

TABLE OF CONTENTS

AUDITED DETAILS

Audited Project

Project name	Token ticker	Blockchain
Eggs	EGGS	Ethereum

Addresses

Contract address	0x2e516ba5bf3b7ee47fb99b09eadb60bde80a82e0
Contract deployer address	0xF5bcd6Bef04a6d6c5643b16D8E00D2EA42956f81

Project Website

https://eggs.care/

Codebase

https://etherscan.io/address/0x2e516ba5bf3b7ee47fb99b09eadb60bde80a82e0#code

SUMMARY

EGGS is an experiment in decentralized finance, the way it works is that each block, EGGS can debase by 0.001% so you will have less EGGS. However if you place your EGGS in a Protec Single or LP staking Vault then you will get more EGGS as reward.

Contract Summary

Documentation Quality

Eggs provides a very good documentation with standard of solidity base code.

- The technical description is provided clearly and structured and also dont have any high risk issue.

Code Quality

The Overall quality of the basecode is standard.

- Standard solidity basecode and rules are already followed by Eggs with the discovery of several low issues.

Test Coverage

Test coverage of the project is 100% (Through Codebase)

Audit Findings Summary

- SWC-103 | Pragma statements can be allowed to float when a contract is intended on lines 7, 445, 472, 507, 878, 952, 1057, 1091, 1117, 1398, 1491, 1580, 1890, 1952, 2043, 2071, 2497, 2533, 2555, 2704, 2951, 3029 and 3055.
- SWC-110 SWC-123 | It is recommended to use of revert(), assert(), and require() in Solidity, and the new REVERT opcode in the EVM on lines 1878 and 1735.
- SWC-116 | It is recommended to use oracles for block values as a proxy for time on lines 3418.

CONCLUSION

We have audited the Eggs project released on January 2023 to discover issues and identify potential security vulnerabilities in Eggs Project. This process is used to find technical issues and security loopholes which might be found in the smart contract.

The security audit report provides satisfactory results with low-risk issues.

The Eggs smart contract code issues do not pose a considerable risk. The writing of the contract is close to the standard of writing contracts in general. The low-risk issues found are some arithmetic operation issues, a floating pragma is set, a user-provided assertion failure, a control flow decision is made based on The block.timestamp environment variable, and requirement violation. A floating pragma is set, the current pragma Solidity directive is ""^0.8.0"". Specifying a fixed compiler version is recommended to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code. A user-provided assertion failed, and a user-provided assertion failed with the message 'Panic(0x11)'. A control flow decision is made based on The block.timestamp environment variable. The block.timestamp environment variable determines a control flow decision. Note that the values of variables like coinbase, gas limit, block number, and timestamp are predictable and can be manipulated by a malicious miner. Also, keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness; be aware that using these variables introduces a certain level of trust into miners.

AUDIT RESULT

Article	Category	Description	Result
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	PASS
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	PASS
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	PASS
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	ISSUE FOUND
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	PASS
Unprotected Ether Withdrawal	SWC-105	Due to missing or insufficient access controls, malicious parties can withdraw from the contract.	PASS
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	PASS
Reentrancy	SWC-107	Check effect interaction pattern should be followed if the code performs recursive call.	PASS
Uninitialized Storage Pointer	SWC-109	Uninitialized local storage variables can point to unexpected storage locations in the contract.	PASS
Assert Violation	SWC-110 SWC-123	Properly functioning code should never reach a failing assert statement.	ISSUE FOUND
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	PASS
Delegate call to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	PASS

DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	PASS
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	PASS
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	PASS
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	ISSUE FOUND
Signature Unique ID	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	PASS
Incorrect Constructor Name	SWC-118	Constructors are special functions that are called only once during the contract creation.	PASS
Shadowing State Variable	SWC-119	State variables should not be shadowed.	PASS
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	PASS
Write to Arbitrary Storage Location	SWC-124	The contract is responsible for ensuring that only authorized user or contract accounts may write to sensitive storage locations.	PASS
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order. The rule of thumb is to inherit contracts from more /general/ to more /specific/.	PASS
Insufficient Gas Griefing	SWC-126	Insufficient gas griefing attacks can be performed on contracts which accept data and use it in a sub-call on another contract.	PASS
Arbitrary Jump Function	SWC-127	As Solidity doesnt support pointer arithmetics, it is impossible to change such variable to an arbitrary value.	PASS

Typographical Error	SWC-129	A typographical error can occur for example when the intent of a defined operation is to sum a number to a variable.	PASS
Override control character	SWC-130	Malicious actors can use the Right-To-Left-Override unicode character to force RTL text rendering and confuse users as to the real intent of a contract.	PASS
Unused variables	SWC-131 SWC-135	Unused variables are allowed in Solidity and they do not pose a direct security issue.	PASS
Unexpected Ether balance	SWC-132	Contracts can behave erroneously when they strictly assume a specific Ether balance.	PASS
Hash Collisions Variable	SWC-133	Using abi.encodePacked() with multiple variable length arguments can, in certain situations, lead to a hash collision.	PASS
Hardcoded gas amount	SWC-134	The transfer() and send() functions forward a fixed amount of 2300 gas.	PASS
Unencrypted Private Data	SWC-136	It is a common misconception that private type variables cannot be read.	PASS

SMART CONTRACT ANALYSIS

Started	Saturday Jan 28 2023 01:35:56 GMT+0000 (Coordinated Universal Time)
Finished	Sunday Jan 29 2023 19:18:39 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Main Source File	Eggs.sol

Detected Issues

ID	Title	Severity	Status
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged

SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-110	A USER-PROVIDED ASSERTION FAILED.	low	acknowledged
SWC-116	A CONTROL FLOW DECISION IS MADE BASED ON THE BLOCK.TIMESTAMP ENVIRONMENT VARIABLE.	low	acknowledged
SWC-123	REQUIREMENT VIOLATION.	low	acknowledged

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 7

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
6
7   pragma solidity ^0.8.0;
8
9   /**
10   * @dev Library for managing
11
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 445

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
444
445   pragma solidity ^0.8.0;
446
447   /**
448   * @dev Interface of the ERC165 standard, as defined in the
449
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 472

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
471
472   pragma solidity ^0.8.0;
473
474   /**
475   * @dev Implementation of the {IERC165} interface.
476
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 507

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
506
507   pragma solidity ^0.8.0;
508
509   /**
510   * @dev Standard math utilities missing in the Solidity language.
511
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 878

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
877
878   pragma solidity ^0.8.0;
879
880   /**
881   * @dev String operations.
882
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 952

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
951
952   pragma solidity ^0.8.0;
953
954   /**
955   * @dev External interface of AccessControl declared to support ERC165 detection.
956
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 1057

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
1056
1057   pragma solidity ^0.8.0;
1058
1059   /**
1060   * @dev External interface of AccessControlEnumerable declared to support ERC165
detection.
1061
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 1091

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
1090
1091   pragma solidity ^0.8.0;
1092
1093   /**
1094   * @dev Provides information about the current execution context, including the
1095
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 1117

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
1116
1117   pragma solidity ^0.8.0;
1118
1119   /**
1120   * @dev Contract module that allows children to implement role-based access
1121
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 1398

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
1397
1398   pragma solidity ^0.8.0;
1399
1400   /**
1401   * @dev Extension of {AccessControl} that allows enumerating the members of each
role.
1402
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 1491

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
1490
1491   pragma solidity ^0.8.0;
1492
1493   /**
1494   * @dev Contract module which provides a basic access control mechanism, where
1495
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 1580

low SEVERITY

The current pragma Solidity directive is ""^0.8.1"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
1579
1580   pragma solidity ^0.8.1;
1581
1582   /**
1583   * @dev Collection of functions related to the address type
1584
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 1890

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
1889
1890   pragma solidity ^0.8.0;
1891
1892   /**
1893   * @dev Interface of the ERC20 Permit extension allowing approvals to be made via
signatures, as defined in
1894
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 1952

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
1951
1952   pragma solidity ^0.8.0;
1953
1954   /**
1955   * @dev Interface of the ERC20 standard as defined in the EIP.
1956
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 2043

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
2042
2043   pragma solidity ^0.8.0;
2044
2045   /**
2046   * @dev Interface for the optional metadata functions from the ERC20 standard.
2047
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 2071

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
2070
2071   pragma solidity ^0.8.0;
2072
2073   /**
2074   * @dev Implementation of the {IERC20} interface.
2075
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 2497

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
2496
2497   pragma solidity ^0.8.0;
2498
2499   /**
2500   * @dev Extension of {ERC20} that allows token holders to destroy both their own
2501
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 2533

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
2532
2533   pragma solidity ^0.8.0;
2534
2535   contract ERC20PresetMinterRebaser is
2536   Context,
2537
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 2555

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
2554
2555   pragma solidity ^0.8.0;
2556
2557   /**
2558   * @title SafeERC20
2559
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 2704

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
2703
2704   pragma solidity ^0.8.0;
2705
2706   // CAUTION
2707   // This version of SafeMath should only be used with Solidity 0.8 or later,
2708
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 2951

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
2950
2951   pragma solidity ^0.8.0;
2952
2953   // Storage for a EGGS token
2954   contract EGGS {
2955
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 3029

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
3028
3029   pragma solidity ^0.8.0;
3030
3031   abstract contract IEGGS {
3032   /**
3033
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 3055

low SEVERITY

The current pragma Solidity directive is ""^0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- Eggs.sol

Locations

```
3054
3055   pragma solidity ^0.8.0;
3056
3057   contract Eggs is ERC20PresetMinterRebaser, Ownable, IEGGS {
3058   using SafeMath for uint256;
3059
```

SWC-110 | A USER-PROVIDED ASSERTION FAILED.

LINE 1878

low SEVERITY

A user-provided assertion failed with the message 'Panic(0x11)'.

Source File

- Eggs.sol

Locations

```
1877   let returndata_size := mload(returndata)
1878   revert(add(32, returndata), returndata_size)
1879   }
1880   } else {
1881   revert(errorMessage);
1882
```

SWC-116 | A CONTROL FLOW DECISION IS MADE BASED ON THE BLOCK.TIMESTAMP ENVIRONMENT VARIABLE.

LINE 3418

low SEVERITY

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source File

- Eggs.sol

Locations

```
3417   ) public {
3418   require(block.timestamp <= deadline, "EGGS/permit-expired");
3419
3420   bytes32 digest = keccak256(
3421   abi.encodePacked(
3422
```

SWC-123 | REQUIREMENT VIOLATION.

LINE 1735

low SEVERITY

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source File

- Eggs.sol

Locations

```
1734   );
1735   (bool success, bytes memory returndata) = target.call{value: value}(
1736   data
1737   );
1738   return
1739
```

DISCLAIMER

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

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is 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 Sysfixed to perform a security assessment. This report does 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.

This is a limited report on our findings based on our analysis, in accordance with good industry practice as of the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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.

This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Sysfixed and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (Sysfixed) owe no duty of care.

ABOUT US

Sysfixed is a blockchain security certification organization established in 2021 with the objective to provide smart contract security services and verify their correctness in blockchain-based protocols. Sysfixed automatically scans for security vulnerabilities in Ethereum and other EVM-based blockchain smart contracts. Sysfixed a comprehensive range of analysis techniques—including static analysis, dynamic analysis, and symbolic execution—can accurately detect security vulnerabilities to provide an in-depth analysis report. With a vibrant ecosystem of world-class integration partners that amplify developer productivity, Sysfixed can be utilized in all phases of your project's lifecycle. Our team of security experts is dedicated to the research and improvement of our tools and techniques used to fortify your code.