DeHacker

Code Security Assessment

Self Crypto

Sep 7th, 2023

Contents

CONTENTS 1
SUMMARY 2
ISSUE CATEGORIES 3
OVERVIEW 4
PROJECT SUMMARY 4
VULNERABILITY SUMMARY 4
AUDIT SCOPE 5
FINDINGS 6
INFORMATIONAL....7
SEL-02|UNCHECKED ERC-20 transfer() / transferFrom() CALL....7
DESCRIPTION 7
RECOMMENDATION 7
INFORMATIONAL....8
SEL-03|UNLOCKED COMPILER VERSION....8
DESCRIPTION 8
RECOMMENDATION 8
INFORMATIONAL....9
SEL-04|MISSING ERROR MESSAGES....9
DESCRIPTION 9
RECOMMENDATION 9
INFORMATIONAL....10
SEL-05|UNUSED CONTRACT....10
DESCRIPTION 10
RECOMMENDATION 10
DISCLAIMER....11
APPENDIX....12
ABOUT....13

Summary

DeHacker's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.
Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow/underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service/logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Issue Categories

Every issue in this report was assigned a severity level from the following:

Critical severity issues

A vulnerability that can disrupt the contract functioning in a number of scenarios or creates a risk that the contract may be broken.

Major severity issues

A vulnerability that affects the desired outcome when using a contract or provides the opportunity to use a contract in an unintended way.

Medium severity issues

A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.

Minor severity issues

A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.

Informational

A vulnerability that has informational character but is not affecting any of the code.

Overview

Project Summary

Project Name	Self Crypto
Platform	BSC
Website	https://selfcrypto.io/
Type	Defi
Language	Solidity

Vulnerability Summary

Vulnerability Level	Total	Mitigated	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
Major	0	0	0	0	0	0
Medium	0	0	0	0	0	0
Minor	0	0	0	0	0	0
Informational	4	0	0	4	0	0
Discussion	0	0	0	0	0	0

Audit scope

ID	File	SHA256 Checksum
SEL	SELF_Token.sol	d483d2cfe1be07416b09ecfeb071063184 66352c1d2be2aba6f6c00a5490b880

Findings

ID	Category	Severity	Status
SEL-02	Volatile Code	Informational	Acknowledged
SEL-03	LanguageSpecific	Informational	Acknowledged
SEL-04	Coding Style	Informational	Acknowledged
SEL-05	Coding Style	Informational	Acknowledged

INFORMATIONAL

SEL-02|UNCHECKED ERC-20 transfer() / transferFrom() CALL

Category	Severity	Location	Status
Volatile Code	Informational	SELF_Token.sol: 471	Acknowledged

Description

The return value of the transfer()/transferFrom() call is not checked.

```
471        IERC20(tokenAddress).transfer(owner(), tokenAmount);
```

Recommendation

Since some ERC-20 tokens return no values and others return a bool value, they should be handled with care. We adviseusing the OpenZeppelin's SafeERC20.sol implementation to interact with the transfer() and transferFrom()functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value andreverts if false is returned, making it compatible with all ERC-20 token implementations.

INFORMATIONAL

SEL-03|UNLOCKED COMPILER VERSION

Category	Severity	Location	Status
Volatile Code	Informational	ARTY.sol (f3a46e0): 138	Acknowledged

Description

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

Recommendation

We recommend the compiler version is instead locked at the lowest version possible that the contract can be compiled at.For example, for version v0.8.7 the contract should contain the following line:

```
pragma solidity 0.8.7;
```

INFORMATIONAL

SEL-04|MISSING ERROR MESSAGES

Category	Severity	Location	Status
Coding Style	Informational	SELF_Token.sol: 591	Acknowledged

Description

The require can be used to check for conditions and throw an exception if the condition is not met. It is better to provide astring message containing details about the error that will be passed back to the caller.

Recommendation

We advise adding error messages to the linked require statements.

INFORMATIONAL

SEL-05|UNUSED CONTRACT

Category	Severity	Location	Status
Coding Style	Informational	SELF_Token.sol: 313	Acknowledged

Description

```
313  library Address {
```

The library Address is declared but never used.

Recommendation

We advise removing the unused contracts or libraries.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Coding Style

Coding Style findings usually do not affect the generated bytecode but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Volatile Code

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

Logical Issue

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

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux “sha256sum" command against the target file.

About

DeHacker is a team of auditors and white hat hackers who perform security audits and assessments. With decades of experience in security and distributed systems, our experts focus on the ins and outs of system security. Our services follow clear and prudent industry standards. Whether it's reviewing the smallest modifications or a new platform, we'll provide an in-depth security survey at every stage of your company's project. We provide comprehensive vulnerability reports and identify structural inefficiencies in smart contract code, combining high-end security research with a real-world attacker mindset to reduce risk and harden code.

BLOCKCHAIINS

Ethereum

Cosmos

Eos

Substrate

TECH STACK

Python

Solidity

Rust

c++

CONTACTS

https://dehacker.io

https://twitter.com/dehackerio

https://github.com/dehacker/audits_public

https://t.me/dehackerio

https://blog.dehacker.io/

DeHacker

Sep 2023