DeHacker

Code Security Assessment

Kiwigo

Aug 16th, 2023

Contents

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	Kiwigo
Platform	BSC
Website	https://www.kiwigo.app/
Type	Defi
Language	Solidity

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	0	0	0	0	0	0
Minor	0	0	0	0	0	0
Informational	3	3	0	0	0	0
Discussion	1	1	0	0	0	0

Audit scope

ID	File	SHA256 Checksum
VCK	kiwiVault.sol	85bbd230dbf79cf7e134142f3c853eb437945da64b94ee3e531b4d40b1323a3c
VVR	kiwigo.sol	5eb9dc25b53d1ca8177c89af7094272f11f31587de1b3da66546adf91c9eaf89

Findings

ID	Category	Severity	Status
CKP-01	Gas Optimization	Informational	Pending
CKP-02	Language Specific	Informational	Pending
VCK-01	Language Specific	Informational	Pending
VCK-02	Logical Issue	Discussion	Pending

INFORMATIONAL

CKP-01 | Proper Usage of public and external type

Category	Severity	Location	Status
Gas Optimization	Informational	kiwigo.sol: 320, 329, 469, 488, 501	Pending

Description

Public functions that are never called by the contract could be declared external . When the inputs arearrays external functions are more efficient than public functions.

For example,
renounceOwnership() ;
transferOwnership() ;
increaseAllowance() ;
decreaseAllowance() ;
mint() .

Recommendation

We recommend using the external attribute for functions never called within the contract.

INFORMATIONAL

CKP-02 | Unused Internal Function

Category	Severity	Location	Status
Language Specific	Informational	kiwigo.sol: 592	Pending

Description

Internal function _burnFrom() is never called within the contract BEP20Token . Considering internalfunctions cannot be called externally, this function will never be used.

Recommendation

We recommend removing function _burnFrom() or declaring it external so that it could be calledexternally.

INFORMATIONAL

VCK-01 | Inappropriate Usage of assert

Category	Severity	Location	Status
Language Specific	Informational	kiwiVault.sol: 12, 24, 30	Pending

Description

assert() should only be used to make sure the condition never happens according to the docs.

Recommendation

We recommend using require() instead of assert() at the aforementioned lines.

DISCUSSION

VCK-02 | Incorrect Calculation

Category	Severity	Location	Status
Logical Issue	Discussion	kiwiVault.sol: 97~101	Pending

Description

The comment at line #73 indicates 50 million token will be unlocked in every 6 months. However, thecalculation in function getPendingUnlocked() is different from the description: let timeDiff = 180days , function getPendingUnlocked() returns 475000000000000000 .

The comment at line #74 indicates it is impossible to unlock before 180 days (lockDuration).However, the calculation in function getPendingUnlocked() implies lockDuration is the timeduration to unlock a certain amount of token

Recommendation

We advise the client using the require() function, along with a custom error message when the conditionfails, instead of the assert() function

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

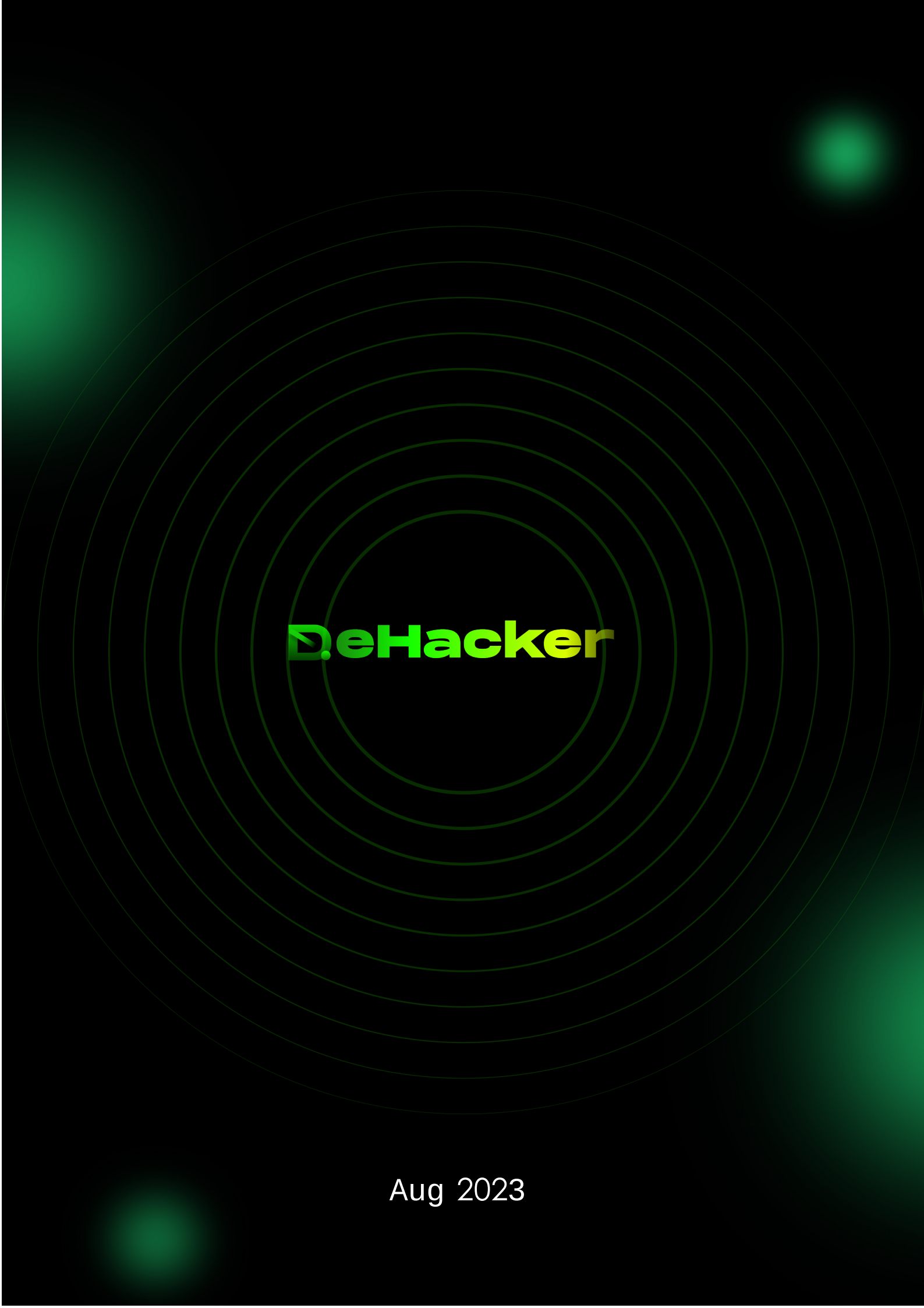
- Ethereum
- Cosmos
- Eos
- Substrate

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/



Aug 2023