DeHacker

Code Security Assessment

DMTR Token

May13th, 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	DMTR Token
Platform	Ethereum
Website	https://dimitra.io/
Type	Token
Language	Solidity

Vulnerability Summary

Vulnerability Level	**Total**	**Pending**	**Declined**	**Acknowledged**	**Partially Resolved**	**Resolved**
Critical	1	0	0	0	0	1
Major	0	0	0	0	0	0
Medium	1	0	0	0	0	1
Minor	0	0	0	0	0	0
Informational	5	0	0	0	0	5
Discussion	0	0	0	0	0	0

Audit scope

ID	File	SHA256 Checksum
DTD	DimitraToken.sol	e96b43ea57f78531b9f5827708180c650369577e113eced39710cef64e9e836f

Findings

ID	Category	Severity	Status
DTD-01	Language Specific	Informational	Resolved
DTD-02	Gas Optimization	Informational	Resolved
DTD-03	Logical Issue	Critical	Resolved
DTD-04	Logical Issue	Medium	Resolved
DTD-05	Gas Optimization	Informational	Resolved
DTD-06	Volatile Code	Informational	Resolved
DTD-07	Gas Optimization	Informational	Resolved

Informational

DTD-01 | Unlocked Compiler Version

Category	Severity	Location	Status
Language Specific	Informational	DimitraToken.sol: 2	Resolved

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of thecontract permits the user to compile it at or above a particular version. This, in turn, leads to differences inthe generated bytecode between compilations due to differing compiler version numbers. This can lead toan ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard toidentify 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 canbe compiled at. For example, for ^0.8.3 , the version can be locked at 0.8.3

Informational

DTD-02 | User-Defined Getters

Category	Severity	Location	Status
Gas Optimization	Informational	DimitraToken.sol: 7 , 22~23	Resolved

Description

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

Recommendation

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

Critical

DTD-03 | Incorrect implementation of approve function

Category	Severity	Location	Status
Logical Issue	Critical	DimitraToken.sol: 53~58	Resolved

Description

The approve function on the aforementioned line is implemented incorrectly where it only allows the senderto set allowance limited to its balance which is not accordingly with the ERC20 standard. Thisimplementation will fail for the decentralized applications that opt to set the maximum uint value forallowance. The current approve implementation intends to limit allowance to current balance of sender, sothe spender cannot transfer out the locked balance of sender. However, this implementation is ineffectualas any spender can increase their balance temporarily up-to the locked balance, set approval to theirsecond address and use transferFrom to take out the locked balance.

Recommendation

We advise to revisit the approve function's implementation and remove the amount limit from setting theallowance to comply with the ERC20 transfer. To safeguard the transferFrom function, we advise tooverride the internal _transfer function and add statement unlockTokens(sender,amount); inside itsbody before calling super._transfer . With this change , the overriding transfer function on L46-51 canbe removed.

Medium

DTD-04 | Same release time can be added multiple times

Category	Severity	Location	Status
Logical Issue	Medium	DimitraToken.sol: 38	Resolved

Description

The aforementioned line will add the locked release time for recipient if there are already locked tokenswith the same release time in userReleaseTimes array. The current implementation does not affect theunlocking balances as the locked amount against the release time is already deleted when the samerelease time appears second time in the userReleaseTimes array on L69 . However, the functionsgetLockedBalance and getReleasedBalance will return incorrect amounts as they take into account thelocked amounts against a release time as many times as it appears in the array.

Recommendation

We advise to revise the logic of issueLockedTokens function such that the same release time is not addedthe second time in userReleaseTimes array against the same recipient.

Informational

DTD-05 | Inefficient function implementation

Category	Severity	Location	Status
Gas Optimization	Informational	DimitraToken.sol: 61~80	Resolved

Description

The function unlockTokens on the aforementioned lines is implemented efficiently where it excessivelymakes use contract's storage for its operation.

Recommendation

We advise to delete the user's release times within the first for loop and avoid second for loop to savegas cost. A more optimized approach of releasing locked balances would be the following.

```
uint lockedAmount;
uint256 len = userReleaseTimes[sender].length;
uint256 j;
for (uint i = 0; i < len; i++) {  // Release all expired locks
    uint256 releaseTime = userReleaseTimes[sender][j];
    if(block.timestamp <= releaseTime) {
        lockedAmount += lockBoxMap[sender][releaseTime];
        j++;
    } else {
        totalLockBoxBalance -= lockBoxMap[sender][releaseTime];
        delete lockBoxMap[sender][releaseTime];
        userReleaseTimes[sender][j] = userReleaseTimes[
            sender
        ][
            userReleaseTimes[sender].length - 1
        ];
        userReleaseTimes[sender].pop();
    }
}
```

Informational

DTD-06 | Ineffectual restriction of view function

Category	Severity	Location	Status
Volatile Code	Informational	DimitraToken.sol: 86, 91, 107	Resolved

Description

The view functions on the aforementioned lines are restricted to be only called by the address withISSUER role to read the balances. This restriction is ineffectual as any data on the blockchain is publiclyavailable for everyone to read.

Recommendation

We advise to remove the ineffectual call restrictions from the aforementioned functions.

Informational

DTD-07 | Explicitly returning local variable

Category	Severity	Location	Status
Gas Optimization	Informational	DimitraToken.sol: 91	Resolved

Description

The function on the aforementioned line explicitly returns local variable which increases overall cost of gas.

Recommendation

Since named return variables can be declared in the signature of a function, consider refactoring to removethe local variable declaration and explicit return statement in order to reduce the overall cost of gas.

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

May 2023