DeHacker

Code Security Assessment

Vulcan Forged II

July 2 5 th, 2023

Contents

CONTENTS 1
SUMMARY 2
ISSUE CATEGORIES 3
OVERVIEW 4
PROJECT SUMMARY 4
VULNERABILITY SUMMARY 4
AUDIT SCOPE 5
FINDINGS 6
MINOR 7
VVF-01|Missing Input Validation 7
DESCRIPTION 7
RECOMMENDATION 7
MINOR 8
VVF-02|Divide by Zero 8
DESCRIPTION 8
RECOMMENDATION 8
MINOR 9
VVR-01 : Missing Input Validation 9
DESCRIPTION 9
RECOMMENDATION 9
INFORMATIONAL 10
VVR-02 : Proper Usage of `require()` And `assert()` 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	Vulcan Forged II
Platform	Ethereum
Website	https://www.vulcanforged.com/
Type	Gamefi
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	3	0	0	0	0	3
Informational	1	0	0	0	0	1
Discussion	0	0	0	0	0	0

Audit scope

ID	File	SHA256 Checksum
VVF	VulcanV2Factory.sol	80b4d593a6c2762d2c2266330a01af671d1f963fea40eb91e8c4be6e36dafc66
VVR	VulcanV2Router02.sol	e991a9e597c0334a71fc000d26b2b509eaf9d73cc6232c3c245fddb6db359912

Findings

ID	Category	Severity	Status
VVF-01	Volatile Code	Minor	Resolved
VVF-02	Logical Issue	Minor	Resolved
VVR-01	Volatile Code	Minor	Resolved
VVR-02	Coding Style	Informational	Resolved

MINOR

VVF-01 | Missing Input Validation

Category	Severity	Location	Status
Volatile Code	Minor	VulcanV2Factory.sol: 461, 468, 185, 180, 174	Resolved

Description

Addresses should be checked before assigning to make sure they are not zero addresses.

Recommendation

We recommended adding validation to check this like bellow:

```
174 function _burn(address from, uint value) internal {
175     require(from != address(0), "burn from zero address!");
176     balanceOf[from] = balanceOf[from].sub(value);
177     totalSupply = totalSupply.sub(value);
178     emit Transfer(from, address(0), value);
179 }
```

```
180 function _approve(address owner, address spender, uint value) private {
181     require(owner != address(0), "owner is zero address!");
182     require(spender != address(0), "spender is zero address!");
183     allowance[owner][spender] = value;
184     emit Approval(owner, spender, value);
185 }
```

```
185 function _transfer(address from, address to, uint value) private {
186     require(from != address(0), "from is zero address!");
187     require(to != address(0), "to is zero address!");
188     balanceOf[from] = balanceOf[from].sub(value);
189     balanceOf[to] = balanceOf[to].add(value);
190     emit Transfer(from, to, value);
191 }
```

MINOR

VVF-02 | Divide By Zero

Category	Severity	Location	Status
Logical Issue	Minor	VulcanV2Factory.sol : 364~366	Resolved

Description

If the value of totalSupply is 0, the following two division operations will fail due to the divide by 0 error,which ultimately make the invocation to burn() function fail.

```
364  amount0 = liquidity.mul(balance0) / _totalSupply; // using balances ensures pro-rata distribution
365  amount1 = liquidity.mul(balance1) / _totalSupply; // using balances ensures pro-rata distribution
```

Recommendation

We advise the client to add the following validation in the function burn()

```
364  function burn(address to) external lock returns (uint amount0, uint amount1) {
365    require(totalSupply != 0, "The value of totalSupply must not be 0");
366      ...
367  }
```

MINOR

VVR-01 | Missing Input Validation

Category	Severity	Location	Status
Volatile Code	Minor	VulcanV2Router0 2.sol: 255~256	Resolved

Description

Addresses should be checked before assigning to make sure they are not zero addresses
.

Recommendation

We recommended adding validation to check this like below:

```
254 constructor(address _factory, address _WETH) public {
255    require(_factory != address(0),"_factory should not be address(0)");
256    require(_WETH != address(0),"_WETH should not be address(0)");
257      factory = _factory;
258      WETH = _WETH;
259 }
```

INFROMATIONAL

VVR-02 | Proper Usage Of require() And assert()

Category	Severity	Location	Status
Coding Style	Informational	VulcanV2Router02.sol: 602, 544, 495, 327, 286, 260	Resolved

Description

The assert() function should only be used to test for internal errors, and to check invariants. Therequire() function should be used to ensure valid conditions, such as inputs, or contract state variablesare met, or to validate return values from calls to external contracts.

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/

DeHacker

July 2023