SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Venus
Date: 28 June, 2023

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Venus
Approved By	Noah Jelich | Lead Solidity SC Auditor at Hacken OU
Tags	Lending/Borrowing
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://venus.io/
Changelog	24.05.2023 - Initial Review 28.06.2023 - Second Review

Table of contents

Introduction **4**
System Overview **4**
Executive Summary **6**
Risks **7**
Checked Items **8**
Findings **11**
Critical 11
High 11
H01. Missing Swap Path Validation 11
Medium 11
M01. Mishandled Edge Case 11
M02. Unverifiable Logic 12
Low 12
L01. Missing Zero Address Validation 12
L02. Boolean Equality 12
L03. Redundant Check 13
Informational 13
I01. SPDX License Identifier Not Provided 13
I02. Floating Pragma 14
I03. Public Functions That Should Be External 14
I04. Copy of Well-Known Contract 14
Disclaimers **16**
Appendix 1. Severity Definitions **17**
Risk Levels 17
Impact Levels 18
Likelihood Levels 18
Informational 18
Appendix 2. Scope **19**

Introduction

Hacken OÜ (Consultant) was contracted by Venus (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

System Overview

Venus Protocol (“Venus”) is an algorithmic-based money market system designed to bring a complete decentralized finance-based lending and credit system.
Venus enables users to utilize their cryptocurrencies by supplying collateral to the network that may be borrowed by pledging over-collateralized cryptocurrencies. This creates a secure lending environment where the lender receives a compounded interest rate annually (APY) paid per block, while the borrower pays interest on the borrowed cryptocurrency.
The Venus Protocol has been designed to give platform users a decentralized and secure marketplace to take out loans, earn an interest, and mint synthetic stablecoins.

The files in the scope:

- **RouterHelper.sol** - The Abstract contract facilitates various token swaps, either for tokens supporting fees or not, and provides library functions to determine transaction specifics.
- **IRouterHelper.sol** - The interface of RouterHelper.
- **SwapRouter.sol** - The contract contains various functions for token swaps, including supporting multiple swap routes. Users can swap tokens and, in the same transaction, supply to any market of Venus or repay their debt in any market of Venus.
- **CustomErrors.sol** - The error handling contract.
- **PancakeLibrary.sol** - The library contract provides the necessary functionality for the swap operation. It manages token sorting, pair address calculation, and token amount conversions.
- **TransferHelper.sol** - The library contract provides methods for interacting with ERC20 and ETH tokens, as well as for safely sending them.
- **InterfaceComptroller.sol** - The interface of Comptroller contract.
- **IPancakePair.sol** - The interface of PancakePair contract.
- **IPancakeSwapV2Factory.sol** - The interface of PancakeSwapV2Factory contract.
- **IPancakeSwapV2Router.sol** - The interface of IPancakeSwapV2Router contract.
- **IVBNB.sol** - The interface of VBep20 contract.
- **IVtoken.sol** - The interface of VToken contract.
- **IWBNB.sol** - The interface of VBep20 contract.

Privileged roles

- SwapRouter.sol:
    - onlyOwner privilege roles:
        - sweepToken() method caller: can withdraw ERC20 token from the contract.
        - setVBNBAddress() : set the VBNB token address.

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are provided.
- Technical description is provided.
- NatSpecs are very good.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.
- The order of the function does not follow the style guide perfectly, but the exceptions make sense.

Test coverage

Code coverage of the project is **67.02%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is missed.
- Some functions are not tested.
- Interactions by several users are not tested thoroughly.

Security score

As a result of the audit, the code contains **no** issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **8.7**. The system users should acknowledge all the risks summed up in the risks section of the report.

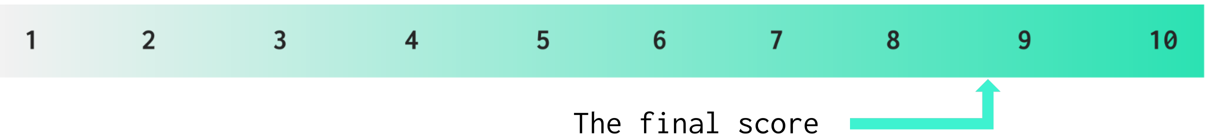

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
24 May 2023	3	1	1	0
28 June 2023	0	0	0	0

Risks

- The protocol provides the possibility to repay a loan from the Venus market and to supply tokens to the Venus market. By doing that, the system interacts with other contracts that are **out of the scope of this audit**.

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Description	Status	Related Issues
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed	
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.	Passed	
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed	
Unchecked Call Return Value	The return value of a message call should be checked.	Passed	
Access Control & Authorization	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed	
SELFDESTRUCT Instruction	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant	
Check-Effect-Interaction	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed	
Deprecated Solidity Functions	Deprecated built-in functions should never be used.	Passed	
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.	Not Relevant	
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed	
Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed	

Authorization through tx.origin	tx.origin should not be used for authorization.	Not Relevant	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Passed	
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant	
Shadowing State Variable	State variables should not be shadowed.	Passed	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
Incorrect Inheritance Order	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed	
Calls Only to Trusted Addresses	All external calls should be performed only to trusted addresses.	Passed	
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Passed	
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed	
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed	
Flashloan Attack	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used. Contracts shouldn’t rely on values that can be changed in the same transaction.	Passed	
Token Supply Manipulation	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed	

Gas Limit and Loops	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed	
Style Guide Violation	Style guides and best practices should be followed.	Passed	
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed	
Environment Consistency	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed	
Secure Oracles Usage	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant	
Tests Coverage	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed	
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed	

Findings

🟥🟥🟥🟥 Critical

No critical severity issues were found.

🟧🟧🟧 High

H01. Missing Swap Path Validation

Impact	High
Likelihood	Medium

All the functions that would execute a swap take as argument the path that will be used to swap between the two tokens concerned. However, no check is done to verify that the path matches with the two tokens that are supposed to be swapped.

It is possible to perform a swap with a path if the user has the relative tokens, even if the vTokenAddress used as parameter is the address of another token.

Paths: ./contracts/Swap/SwapRouter.sol

./contracts/Swap/RouterHelper.sol

Recommendation: Check that the tokens at the beginning and the end of the path are the ones that are supposed to be swapped.

Found in: e75d0ac

Status: Fixed (cf6b8cb)

🟨🟨 Medium

M01. Mishandled Edge Case

Impact	Medium
Likelihood	Medium

In the presented code, if-else control statements are structured in a manner that can lead to unintended execution under certain conditions. Specifically, when the contract has insufficient liquidity, statement 'A' or 'B' could become zero. Despite this, due to the use of the logical "AND" operator (&&), the function will still execute successfully.

Path: ./contracts/Swap/lib/PancakeLibrary.sol : quote(), getAmountOut(), getAmountIn()

Recommendation: Change the “&&” operator with “||” operator.

Found in: e75d0ac

Status: Fixed (cf6b8cb)

M02. Unverifiable Logic

Impact	Low
Likelihood	Medium

The SwapRouter contract externally calls *VBep20.sol* inside the *_supply()* function. The contract uses or interacts with code that is out of audit scope.

Path: ./contracts/Swap/SwapRouter.sol

Recommendation: Add the code that cannot be verified in the scope or document it properly if it cannot be added to the scope.

Found in: e75d0ac

Status: Mitigated (The used contract, which is a wrapper contract, calls the main contract without modifying the function.)

Low

L01. Missing Zero Address Validation

Impact	Medium
Likelihood	Low

Address parameters are being used without checking against the possibility of 0x0.

This can lead to unwanted external calls to 0x0.

Path: ./contracts/Swap/SwapRouter.sol: constructor(), sweepToken()

Recommendation: Implement zero address checks.

Found in: e75d0ac

Status: Fixed (cf6b8cb)

L02. Boolean Equality

Impact	Low
Likelihood	Medium

Boolean constants can be used directly and do not need to be compared to true or false.

Path: ./contracts/Swap/SwapRouter.sol : ensureVTokenListed()

Recommendation: Remove boolean equality.

Found in: e75d0ac

Status: Fixed (cf6b8cb)

L03. Redundant Check

Impact	Low
Likelihood	Medium

In the PancakeLibrary.quote() function, the following checks are done:

```
if (reserveA == 0 && reserveB == 0) {revert InsufficientLiquidity();}
```

```
require(reserveA > 0 && reserveB > 0,
```

The first check is useless as it checks something that is also checked in the second one.

Path: ./contracts/Swap/lib/PancakeLibrary.sol : quote()

Recommendation: Remove redundant code.

Found in: e75d0ac

Status: Fixed (cf6b8cb)

Informational

I01. SPDX License Identifier Not Provided

Impact	Low
Likelihood	Medium

"SPDX-License-Identifier” is not provided in the source files.

Paths: ./contracts/Swap/interfaces/CustomErrors.sol,

./contracts/Swap/interfaces/IPancakePair.sol,

./contracts/Swap/interfaces/IVBNB.sol,

./contracts/Swap/interfaces/IVtoken.sol,

./contracts/Swap/interfaces/IPancakeSwapV2Router.sol,

./contracts/Swap/interfaces/IPancakeSwapV2Factory.sol,

./contracts/Swap/interfaces/InterfaceComptroller.sol,

./contracts/Swap/interfaces/PancakeLibrary.sol,

./contracts/Swap/interfaces/IRouterHelper.sol,

./contracts/Swap/interfaces/SwapRouter.sol,

Recommendation: "SPDX-License-Identifier” should be added to each source file.

Found in: e75d0ac

Status: Fixed (cf6b8cb)

I02. Floating Pragma

Impact	Low
Likelihood	Medium

The project uses floating pragmas *^0.8.13*.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Path: ./contracts/Swap/interfaces/CustomErrors.sol :

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Found in: e75d0ac

Status: Fixed (cf6b8cb)

I03. Public Functions That Should Be External

Impact	Low
Likelihood	Medium

Functions that are only called from outside the contract should be defined as external.

Path: ./contracts/Swap/RouterHelper.sol : quote(), getAmountOut(), getAmountIn(), getAmountsOut(), getAmountsIn()

Recommendation: Make these functions external.

Found in: e75d0ac

Status: Fixed (cf6b8cb)

I04. Copy of Well-Known Contract

The system uses a copy of a well-known contract instead of reusing it.

Path: ./contracts/Swap/SwapRouter.sol : nonReentrant()

Recommendation: Use the OpenZeppelin non reentrancy guard instead of copying it.

Found in: cf6b8cb

Status: New

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	Critical	High	Medium
Medium Likelihood	High	Medium	Low
Low Likelihood	Medium	Low	Low

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/VenusProtocol/venus-protocol
Commit	e75d0ac9df768263dbc830546280b4d0f9385c4e
Whitepaper	Link
Requirements	-
Technical Requirements	Link
Contracts	File: contracts/Swap/IRouterHelper.sol SHA3: f7ab1d6fb7f1afaa1684bfc95ef56d58aa5ad30d37a581250b207b0eb04b338d File: contracts/Swap/RouterHelper.sol SHA3: 3e5613f214d78956d481d2c0cb60c207287f503d8db0261f2ef86f4bf1e94db3 File: contracts/Swap/SwapRouter.sol SHA3: 0f32e109c87f996066f16868842a027d3125a86d44b531ae64cf8471d17039b9 File: contracts/Swap/interfaces/CustomErrors.sol SHA3: 4225b3b6377f70b868fe872d8dbc7216572ebc1c50576ccde3ffc5fdf22e6c0f File: contracts/Swap/interfaces/InterfaceComptroller.sol SHA3: a5a831e6e068c43a5a5225be20eed03240f22a648e78da2f4fc476abcda0a7cf File: contracts/Swap/interfaces/IPancakePair.sol SHA3: 6074813057a1995f74f490a0c0b856ea43fad25f3618524637849740f02e8735 File: contracts/Swap/interfaces/IPancakeSwapV2Factory.sol SHA3: 908ce34715e7e318e5b7386b5041e4eebb75522eb470f4fae76043f05b3a8154 File: contracts/Swap/interfaces/IPancakeSwapV2Router.sol SHA3: 9116fd180e5f7411ae7563d3649f80a87004495b8a69320c82aa8b7585513585 File: contracts/Swap/interfaces/IVBNB.sol SHA3: 0d7704d96da6d31a0742c85caf79672223882199bbb0e9857b981cf711dfab6e File: contracts/Swap/interfaces/IVtoken.sol SHA3: 88b6a0c0770789dca08bea9cef03957fad2cbc31dfa31ae0df6b526f8b91e400 File: contracts/Swap/interfaces/IWBNB.sol SHA3: 218eefa8e720cb40c4e17a13e5273ce426a2b72cf610535f91367e01f8d9bf89 File: contracts/Swap/lib/PancakeLibrary.sol SHA3: 55c636c9f3f052f2d778d131b3c75f2f7d667d979dfd6cc92803fcfd88cc18a4 File: contracts/Swap/lib/TransferHelper.sol SHA3: 0cb8db1603b9af73d02dda580b438939f66e650baddc03467e4dd4340e10221b

Second review scope

Repository	https://github.com/VenusProtocol/venus-protocol
Commit	cf6b8cb0735cf0ded3435161c6ea2e2d6c4b48e4
Whitepaper	Link
Requirements	-
Technical Requirements	Link
Contracts	File: Swap/IRouterHelper.sol SHA3: f2a7576b2c39e2cacd81a1d144a4f10bf24932dfbce174ec27bdb48b84c14ba7 File: Swap/RouterHelper.sol SHA3: 8c99cb80352bd4449eb39d8707ec11a9183a6ca6ac725e2db4508aa6df66211d File: Swap/SwapRouter.sol SHA3: b4f6a2f44600d7b4d6c3b1cdf129f0d63516d776b94498dd5f38ad6207c9c909 File: Swap/interfaces/CustomErrors.sol SHA3: d43aa739c641ff068d0c3c3d7957ad5a7d10ea5ed92ce0ad6fe59391243a369e File: Swap/interfaces/InterfaceComptroller.sol SHA3: b3206e373b1ba94e645345b1e327422c8b52472e1f7d7f90ccbf4f3b9376011f File: Swap/interfaces/IPancakePair.sol SHA3: dbe3ba78586c5e73dd1a76d9d351271ec050577b8e43705495556476cfb62905 File: Swap/interfaces/IPancakeSwapV2Factory.sol SHA3: cfa2589a6868650193532ae76f17c081564bcde00f4c838066cc8fe8cff72607 File: Swap/interfaces/IPancakeSwapV2Router.sol SHA3: db2076da49487773ed7b0fef2ec3aeb6aaae61f07dc6a4a3d34b6689bab80fbe File: Swap/interfaces/IVBNB.sol SHA3: 53c99ab847aa492d8ffe52574fb3a9a9f613afd8450c8f9c133c1bd7155665d8 File: Swap/interfaces/IVtoken.sol SHA3: 2dbbb26f294a1f66c64283d2b2cfe531ba061aceb069beefc3ba1312d991349f File: Swap/interfaces/IWBNB.sol SHA3: 218eefa8e720cb40c4e17a13e5273ce426a2b72cf610535f91367e01f8d9bf89 File: Swap/lib/PancakeLibrary.sol SHA3: 06b77bedfd04a27cba17c1e3a27f58b95a66c60f7fef6db2f9b98974d05e3aeb File: Swap/lib/TransferHelper.sol SHA3: d9e84a2af99af80e4b2b77e6209f03f2f6efd9b0d41eeb7a15e00799381b8349