HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Venus
Date: 26 June, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Venus
Approved By	Oleksii Zaiats | SC Audits Head at Hacken OU
Type	DEX
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://venus.io/
Changelog	10.05.2023 - Initial Review 26.06.2023 - Second Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Introduction **4**
System Overview **4**
Executive Summary **6**
Checked Items **8**
Findings **11**
Critical 11
C01. Incorrect Mathematical Operation 11
High 11
H01. Unverifiable Logic 11
Medium 12
M01. Check-Effect-Interaction 12
Low 12
L01. Undocumented Logic 12
Informational 13
I01. Style Guide Violation - Order of Layout 13
I02. Environmental Consistency 13
I03. Test Coverage 14
Disclaimers **15**
Appendix 1. Severity Definitions **16**
Risk Levels 16
Impact Levels 17
Likelihood Levels 17
Informational 17
Appendix 2. Scope **18**

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

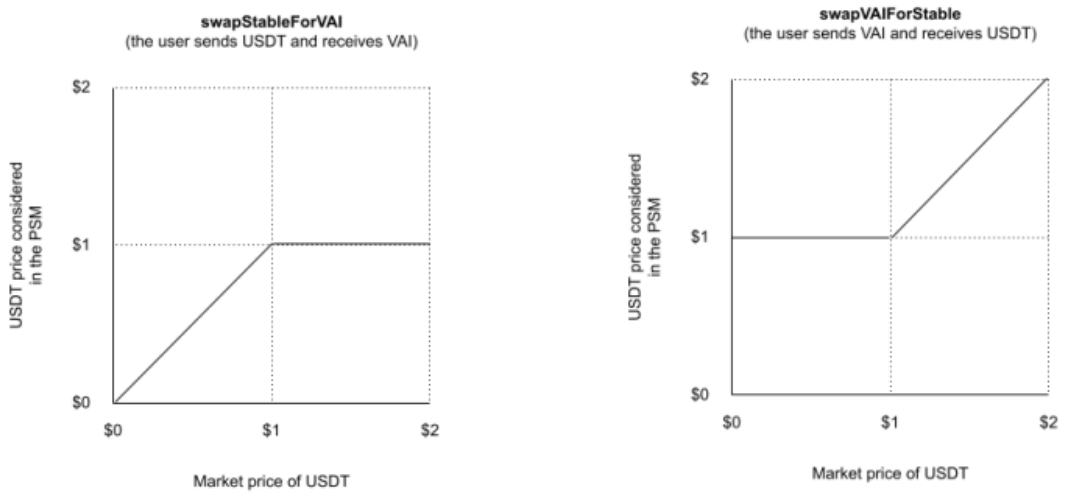
Introduction

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

The audit consists of a pegged token stability contract that is used to help peg the price of VAI to 1$ which is the ERC20 token for the Venus ecosystem.

The contract aims to achieve this by providing a swapping functionality of the VAI token with USDT and USDC tokens by also validating their current prices using oracles. The intended swapping logic can be visualized by the following figure:



This way, even if the pegged token used to ensure VAI is pegged to 1$ gets corrupted, VAI will not be affected as much.

The platform also takes a fee from all swaps in the form of VAI. The VAI outside of the fee is burned when swapping for pegged tokens, and it's minted when pegged tokens are being swapped for VAI.

The files in the scope:

- **PegStability.sol:** The contract that has the swapping mechanism between VAI and the chosen stable token. This contract interacts with price oracles to ensure the intended logic is implemented. The swapped pegged tokens are stored in this contract.
- **AccessControlledV8.sol:** An access control mechanism for the Venus ecosystem that allows certain addresses to perform certain actions, all controlled by the owner of this contract.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Privileged roles

- Owner: Can give certain permissions to addresses such as; pause contract, resume contract, set fees, set VAI token mint cap, set treasury for fee collection, and set comptroller(contract which stores the priceOracle address).
- User: Can swap pegged tokens for VAI tokens and vice versa.
- Access Allowed Addresses: Addresses given access by the owner of the contract can: pause contract, resume contract, set fees, set VAI token mint cap, set treasury for fee collection, and set comptroller(contract which stores the priceOracle address) according to the access they are given.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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 **9** out of **10**.

- Functional requirements are present.
- Technical description is partially missing. There is undocumented logic and interfaces.
- NatSpec is sufficient.
- Development environment description is given.

Code quality

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

- Style guides are followed.
- Code is well-written and well-designed.

Test coverage

Code coverage of the project is **90.48%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative cases are covered.
- Interactions with several users are not tested thoroughly.

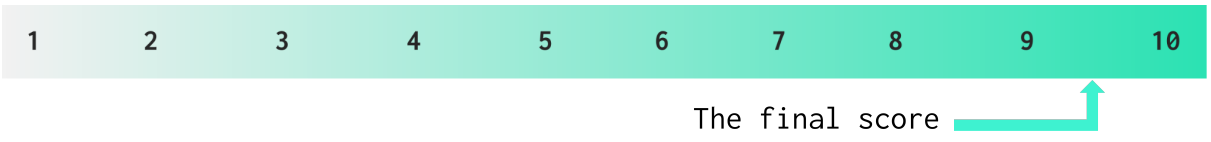
Security score

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: **9.5**.The system users should acknowledge all the risks summed up in the risks section of the report.



Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
10 May 2023	1	1	1	2
26 June 2023	0	0	0	0

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.	Not Relevant	
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	

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.	Not Relevant	
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.	Not Relevant	
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	

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.	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.	Not Relevant	
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.	Passed	
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	I03
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed	

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

■■■■ Critical

C01. Incorrect Mathematical Operation

Impact	High
Likelihood	High

When doing mathematical operations on the stable token amounts retrieved from the oracle, it is assumed that the decimals for the token are 18. This may not be the case for USDT and USDC in some networks.

Comparing values assuming decimals are 18 may lead to incorrect results when swapping.

Path: ./contracts/PegStability/PegStability.sol : previewTokenUSDAmount(), getPriceInUSD()

Recommendation: Normalize the price obtained from the oracle so that VAI token and pegged token decimals are the same when doing calculations.

Found in: 7f394e087828001b0c5b5e1022701b05c0150488

Status: Fixed (Revised commit: be743ca)

■■■ High

H01. Unverifiable Logic

Impact	High
Likelihood	Medium

There are external function calls made that have implementations out of scope.

In the AccessControlledV8.sol contracts' `_checkAccessAllowed()` function, `_accessControlManager.isAllowedToCall()` call is made, which has implementation out of scope.

In the PegStability.sol contract, the interfaces OracleProviderInterface, IVTokenUnderlying, and IPriceOracle have critical functionality, but their implementation is out-of-scope. This leads to the logic of swapping to be unverifiable.

The safety of contracts that use interfaces with out of scope implementations cannot be verified.

Paths:
@venusprotocol\governance-contracts\contracts\Governance\AccessControlledV8.sol : _checkAccessAllowed()

./contracts/PegStability/PegStability.sol : IPriceOracle.getUnderlyingPrice(), IVTokenUnderlying.underlying(), OracleProviderInterface.oracle()

Recommendation: All contracts that are used in the system should be added to the scope.

Found in: 7f394e087828001b0c5b5e1022701b05c0150488

Status: Mitigated (It is confirmed that the out-of-scope contracts are audited by Hacken in another audit.)

Medium

M01. Check-Effect-Interaction

Impact	Medium
Likelihood	Medium

In the PegStability.sol contracts' *swapVAIForStable()* and *swapStableForVAI()* functions, the state variable *vaiMinted* is updated after external calls.

Even though the functions are protected with ReentrancyGuard, it is best practice to follow CEI when possible.

Path:

./contracts/PegStability/PegStability.sol : swapVAIForStable(), swapStableForVAI()

Recommendation: Make all transfers after updating the *vaiMinted* variable.

Found in: 7f394e087828001b0c5b5e1022701b05c0150488

Status: Mitigated (The CEI violation cannot be followed because of the support for deflationary tokens; however, the nonReentrant modifier is used for protection.)

Low

L01. Undocumented Logic

Impact	Low
Likelihood	Low

In the PegStability.sol contracts' *swapStableForVAI()* function, the pegged token balance is checked before and after the transfer, and the amount is taken as the difference between these balances instead of using the *stableTknAmount* variable.

This logic is not documented, uses more Gas, and results in the violation of the Check-Effects-Interaction Pattern (M01).

Path: ./contracts/PegStability/PegStability.sol : swapStableForVAI()

Recommendation: Use the *stableTknAmount* variable directly or document why the currency logic is used.

Found in: 7f394e087828001b0c5b5e1022701b05c0150488

Status: Fixed (Revised commit: be743ca)

Informational

I01. Style Guide Violation - Order of Layout

The project should follow the official code style guidelines.
Inside each contract, library, or interface, use the following order:

- Type declarations
- State variables
- Events
- Modifiers
- Functions

Functions should be grouped according to their visibility and ordered:

- constructor
- receive function (if exists)
- fallback function (if exists)
- external
- public
- internal
- private

Within a grouping, place the view and pure functions at the end.

Path: ./contracts/PegStability/PegStability.sol : *

Recommendation: The official Solidity style guidelines should be followed.

Found in: 7f394e087828001b0c5b5e1022701b05c0150488

Status: Fixed (Revised commit: be743ca)

I02. Environmental Consistency

The development environment is not configured for the contracts in the scope.

Not having clear instruction on how to compile, test, and deploy the code is against best practices.

Path: ./contracts/PegStability/PegStability.sol

Recommendation: Provide documentation on how to compile, test, and deploy the contracts.

Found in: 7f394e087828001b0c5b5e1022701b05c0150488

Status: Fixed (Revised commit: be743ca)

I03. Test Coverage

It is important to write full coverage tests in order to catch possible logical issues.

Paths: ./contracts/PegStability/PegStability.sol

@venusprotocol\governance-contracts\contracts\Governance\AccessControlledV8.sol

Recommendation: Obtain full coverage from tests.

Found in: 7f394e087828001b0c5b5e1022701b05c0150488

Status: Reported (Test coverage increased 90.48)

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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	7f394e087828001b0c5b5e1022701b05c0150488
Whitepaper	Link
Requirements	Link
Technical Requirements	Link
Contracts	File: contracts/PegStability/PegStability.sol SHA3: 50079c57130d57847c78cc8709df2923cfd610c33832194b190724fba0ee9fe4 File:@venusprotocol/governance-contracts/contracts/Governance/AccessControlledV8.sol SHA3: 92b607a9b30f25134e53df2f96a83dba2feadfc8996d80e9821c23e2d067097a

Second review scope

Repository	https://github.com/VenusProtocol/venus-protocol
Commit	be743caf07c8b15496524db5bac594754d85cc2a
Whitepaper	Link
Requirements	Link
Technical Requirements	Link
Contracts	File: contracts/PegStability/PegStability.sol SHA3: abe75dc806284a3aa1515858cac254a6852f5023ede45b0c7aede8561aba1857 File:@venusprotocol/governance-contracts/contracts/Governance/AccessControlledV8.sol SHA3: 92b607a9b30f25134e53df2f96a83dba2feadfc8996d80e9821c23e2d067097a