SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: WoW Max
Date: 29 Aug, 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 WoW Max
Approved By	Oleksii Zaiats | SC Audits Head at Hacken OÜ
Tags	DEX
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://wowmax.exchange/
Changelog	07.08.2023 - Initial Review 29.08.2023 - Second Review

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

Table of contents

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

Introduction

Hacken OÜ (Consultant) was contracted by WoW Max(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 project "WoW Max" is a decentralized exchange (DEX) aggregator designed with the objective of optimizing exchange amounts. Its primary function involves analyzing token prices across multiple DEX platforms and identifying the most efficient exchange path among various exchange protocols.
By doing so, WOWMAX aims to provide users with the best possible returns on their token exchanges, maximizing their gains while reducing potential losses.
The files in the scope:

- **WowmaxRouter.sol - ** The contract that the users interact with. Responsible for the swapping logic of WoW Max which interacts with several verified DEXes.
- **WowmaxSwapReentrancyGuard.sol - ** The reentrancy guard implementation for the WowmaxRouter.sol which does not allow for reentry during any swapping operations.
- **IWETH.sol -** Interface for the native token wrapper.
- **IWowmaxRouter.sol -** Interface for WowmaxRouter.sol
- **Curve.sol -** Swapping library for Curve like swapping protocols.
- **DODOV1.sol -** Swapping library for DODOV1 swapping protocol.
- **DODOV2.sol -** Swapping library for DODOV2 swapping protocol.
- **Fulcrom.sol -** Swapping library for Fulcron swapping protocol.
- **Hashflow.sol -** Swapping library for Hashflow swapping protocol.
- **Level.sol -** Swapping library for Level swapping protocol.
- **PancakeSwapStable.sol -** Swapping library for PancakeSwap like protocols.
- **Saddle.sol -** Swapping library for Saddle swapping protocol.
- **UniswapV2.sol -** Swapping library for UniswapV2 swapping protocol.
- **UniswapV3.sol -** Swapping library for UniswapV3 swapping protocol.
- **Wombat.sol -** Swapping library for Wombat swapping protocol.
- **WooFi.sol -** Swapping library for WooFi swapping protocol.

Privileged roles

- Owner: Can withdraw excess funds from the contract, in case of leftovers after a swap, or invalid swap requests.
- User: Can interact with the WowmaxRouter.swap to swap tokens.

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.
- Description of the development environment is present.

Code quality

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

- The code follows the Solidity style guides.
- The development environment is configured.

Test coverage

Code coverage of the project is **100%** (branch coverage)

- Deployment and user interactions are covered with tests.

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

1	2	3	4	5	6	7	8	9	10

The final score

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
7 August 2023	2	3	2	3
29 August 2023	0	0	0	0

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

Risks

- The implementations of the swapping logic used in the system are **out of scope** of this contract, and therefore their safety cannot be verified.
- The *amountOutExpected* parameter during swaps is user-provided. A check for it to be non-zero is implemented; however, the value cannot be fully monitored and the risk for it to be invalid is present.

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.	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	

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.	Passed	
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	

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. 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.	Passed	
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. Access Control Violation

Impact	High
Likelihood	High

In the contract, any user can call functions and grant approval to any specified address without any validation. An attacker could create a malicious pool contract, obtain approvals for their contract, and if there are any funds in the *WowmaxRouter* contract, transfer those tokens to their own wallet. Inside the *swap()* functions. A malicious attacker can deploy a malicious contract and pass it as *swapData.add* parameter to these functions. In this scenario, the *WowmaxRouter* contract will grant token approval to the given contract address. Later, malicious user can withdraw ERC20 tokens using the *transferFrom()* function. Since they already obtained the approval, he can successfully execute the *transferFrom()* from *WowmaxRouter* to a malicious contract.

Paths: ./contracts/WowmaxRouter.sol,

./contracts/WowmaxSwapReentrancyGuard.sol,

./contracts/libraries/Curve.sol,

./contracts/libraries/DODOV1.sol,

./contracts/libraries/Hashflow.sol,

./contracts/libraries/PancakeSwapStable.sol,

./contracts/libraries/Saddle.sol,

./contracts/libraries/UniswapV2.sol,

./contracts/libraries/Wombat.sol,

Recommendation: Implement validation for the *swapData.addr* for each contract, such as a whitelist. The contract should be designed to only accept addresses that are approved by the system. If the address is not recognized, the contract should revert the transaction.

Found in: 7c053df

Status: Mitigated (The edge case of this exploit is if a user sends accidental funds to the contract, or there are dust values left in the contract. Since there should not be funds for an attacker in the contract and even if there are funds, the exploit only affects the individual user in fault and not the system itself. The issue is reported as a risk.)

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

C02. Balance Manipulation

Impact	High
Likelihood	High

Within the *swap()* function, if *request.amountIn* is set to zero during a swap operation, the contract fails to execute the swap, instead transferring the contract's current balance to the function's executor.

Within the *swap()* function, if the contract already has a non-zero token balance during a swap operation, it will send the excess amount of tokens to the executor of the function, due to *balanceOf()* usage and user-given *amountsOutExpected* parameter. This could enable users to illicitly extract tokens from the contract's balance.

Also, *exchangeRoutes* could be set as empty list, which results in skipping *exchange* function calls.

Path: ./contracts/WowmaxRouter.sol : swap()

Recommendation: Introduce *beforeBalance* and *afterBalance* checks before and after the *exchange()* function, respectively. Initially, capture the contract's balance using *balanceOf()* and, once all operations are complete, call *balanceOf()* again. Subtracting these values *(balanceAfter - balanceBefore)* will yield the accurate amount of tokens exchanged during the operation. Use this amount on transfer instead of using the *balanceOf()* amount directly. Introduce validation of input parameters to revert transaction with incorrect input data.

Found in: 7c053df

Status: Mitigated (The edge case of this exploit is if a user sends accidental funds to the contract, or there are dust values left in the contract. Since there should not be funds for an attacker in the contract and even if there are funds, the exploit only affects the individual user in fault and not the system itself. The issue is reported as a risk.)

C03. Missing Validation

Impact	High
Likelihood	High

The *uniswapV3SwapCallback()* function is intended for use only during an active swap. But when handling native token transfers, if a contract is deployed to accept these swapped tokens, its *receive()* function gets triggered. This allows a malicious actor to potentially exploit the process by invoking the *uniswapV3SwapCallback()* within the *receive()* function, redirecting tokens to an unauthorized contract.

Also, the same scenario is possible by deploying and passing malicious contracts to the swap function.

Path: ./contracts/WowmaxRouter.sol : swap()

Recommendation: Ensure that the msg.sender is associated with the uniswapV3 pool Factory. Incorporate an address computation function that accepts the swapped token addresses (token0 and token1) and the init code hash, subsequently computing the pool address. At the onset of the function, validate that *msg.sender* equals the computed address.

Found in: 7c053df

Status: Mitigated (The edge case of this exploit is if a user sends accidental funds to the contract, or there are dust values left in the contract. Since there should not be funds for an attacker in the contract and even if there are funds, the exploit only affects the individual user in fault and not the system itself. The issue is reported as a risk.)

High

H01. Sandwich Attack

Impact	High
Likelihood	Medium

The contract performs swaps based on user-provided slippage values, but it lacks a proper mechanism for slippage calculation. The absence of a proper mechanism for slippage calculation can make the system vulnerable to sandwich attacks initiated by a malicious actor.

Path: ./contracts/WowmaxRouter.sol : swap(), sendTokens()

Recommendation: Implement proper minimum slippage calculation and validate the given input.

Found in: 7c053df

Status: Fixed (Revised commit: e34a1be)

H02. Possible Funds Loss

Impact	High
Likelihood	Medium

Inside the *swap()* function, during an ETH to Token swap operation, if a user sends an empty *request.exchangeRoutes* and specifies zero for *amountOutExpected*, the router contract still retains the user's Ether even though the swap is not complete due to the absence of *request.exchangeRoutes*.

In the *swap()* function, during a Token to Token swap operation, if the *amountOutExpected* is set too low (e.g., 1 or 2), the majority of the swapped tokens will be transferred to the treasury. Users will receive an extremely small amount of tokens due to the logic inside the sendTokens if-statement:

amountExtra = amountOut - request.amountOutExpected[i];

amountsOut[i] = request.amountOutExpected[i];

This can result in a loss of funds for the user.

Path: ./contracts/WowmaxRouter.sol : swap(), receiveTokens(), sendTokens()

Recommendation: Implement a proper validation for *request.amountOutExpected[i]*

Found in: 7c053df

Status: Fixed (Revised commit: e34a1be)

Medium

M01. Usage Of Built-in Transfer

Impact	Medium
Likelihood	Medium

The built-in transfer and send functions process hard-coded amount of Gas. In case of receiver is a contract with receive or fallback function, the transfer may fail due to the “out of Gas” exception.

Path: ./contracts/WowmaxRouter.sol : transfer(), withdrawETH()

Recommendation: Replace transfer and send functions with call or provide special mechanism for interacting with a smart contract.

Found in: 7c053df

Status: Fixed (Revised commit: e34a1be)

M02. Missing Validation

Impact	Medium
Likelihood	Medium

The fee variable in the swapData function is sent as an input to the contract without any validation checks. Notably, while UniswapV2's fee is hardcoded at *%0.3*, the fee input parameter can be specified to be greater than this value. This discrepancy could lead to inconsistencies in the contract's operation.

Path: ./contracts/libraries/UniswapV2.sol: swap()

Recommendation: Either explain clearly in the public documentation if the system takes additional fee or define fee as a constant variable.

Found in: 7c053df

Status: Fixed (Revised commit: e34a1be)

M03. Unsafe Usage of Third Party Protocol

Impact	Medium
Likelihood	Medium

The contract uses third party protocols in its swapping logic; however, there is no proper way to disconnect these protocols from the system in case they get corrupted.

This may result in unexpected behavior and fund losses if the used swapping protocols get hacked.

Paths:

./contracts/libraries/Curve.sol
./contracts/libraries/DODOV1.sol
./contracts/libraries/DODOV2.sol
./contracts/libraries/Fulcrom.sol
./contracts/libraries/Hashflow.sol
./contracts/libraries/Level.sol
./contracts/libraries/PancakeSwapStable.sol
./contracts/libraries/Saddle.sol
./contracts/libraries/UniswapV2.sol
./contracts/libraries/UniswapV3.sol
./contracts/libraries/Wombat.sol
./contracts/libraries/WooFi.sol

Recommendation: Implement a pausing mechanism for every third party swapping protocol that is used in the system so that proper actions can be taken in case of a hack in the corresponding third party protocol.

Found in: 7c053df

Status: Mitigated (The slippage protection implemented is enough to ensure safety of swaps.)

Low

L01. Out-Of-Bounds Array Access

Impact	Low
Likelihood	Medium

Out-of-bounds array access issues in Solidity arise when reading from or writing to an array position that exceeds the current array length. This leads to unexpected outcomes, such as default value returns for read operations or exceptions for write operations.

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

Improper handling of these cases can potentially expose the contract to unexpected behavior.

Path: ./contracts/WowmaxRouter.sol : sendTokens()

Recommendation: All array accesses should be within bounds. Enforcement of array bounds can be ensured by implementing 'require' statements. Implement Check *(request.to.length == request.amountOutExpected.length)*

Found in: 7c053df

Status: Fixed (Revised commit: e34a1be)

L02. Redundant Import

Impact	Low
Likelihood	Medium

The contract inherits OpenZeppelin’s ReentrancyGuard, but it does not use its functionality.

The redundancy in inheritance and import can lead to unnecessary Gas consumption during deployment and potentially impact code quality.

Path: ./contracts/WowmaxRouter.sol : ReentrancyGuard

Recommendation: Remove redundant import and inheritance to save Gas on deployment and increase the code quality.

Found in: 7c053df

Status: Fixed (Revised commit: e34a1be)

Informational

I01. Floating Pragma

The project uses floating pragmas *^0.8.7*.

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/*.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: 7c053df

Status: Fixed (Revised commit: e34a1be)

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

I02. Solidity Style Guides Violation

Contract readability and code quality are influenced significantly by adherence to established style guidelines. In Solidity programming, there exist certain norms for code arrangement and ordering. These guidelines help to maintain a consistent structure across different contracts, libraries, or interfaces, making it easier for developers and auditors to understand and interact with the code.

The suggested order of elements within each contract, library, or interface is as follows:

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

Functions should be ordered and grouped by their visibility as follows:

- Constructor
- Receive function (if exists)
- Fallback function (if exists)
- External functions
- Public functions
- Internal functions
- Private functions

Within each grouping, view and pure functions should be placed at the end.

Furthermore, following the Solidity naming convention and adding NatSpec annotations for all functions are strongly recommended. These measures aid in the comprehension of code and enhance overall code quality.

Path: ./contracts/WowmaxRouter.sol

Recommendation: Consistent adherence to the official Solidity style guide is recommended. This enhances readability and maintainability of the code, facilitating seamless interaction with the contracts. Providing comprehensive NatSpec annotations for functions and following Solidity's naming conventions further enrich the quality of the code.

Found in: 7c053df

Status: Fixed (Revised commit: e34a1be)

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/wowswap-io/wowmax-contracts
Commit	7c053dfd12460e6dd9c351ba9f1bc6e28a80e103
Whitepaper	Link
Requirements	Link
Technical Requirements	Link
Contracts	File: contracts/WowmaxRouter.sol SHA3: aae55d441e19246e05007ed73b6a52f444c8bac33f83b64e5cf67d10ccbddfb4 File: contracts/WowmaxSwapReentrancyGuard.sol SHA3: 72f4fc233037fb4ae6c3c8bbb59638e00329066e820ca3b6923628b27a6a58a9 File: contracts/interfaces/IWETH.sol SHA3: 90ab897ec8f6b350bed91d01d3b980ce196eb315aaa7b775394d6c5f58c2c5b5 File: contracts/interfaces/IWowmaxRouter.sol SHA3: 584500427482c53bb1b240c3d06dd6a95b85a9986a5bce56b129cffe2860ed42 File: contracts/libraries/Curve.sol SHA3: feb0b9b91f6c546ddea09d224362c5f66ec03f2028f1d87a1157477400f0d066 File: contracts/libraries/DODOV1.sol SHA3: 18c9a2b6c64ab08198e5299b848cd7ab4d22cc9dd9d6461179dbac84a476cdbf File: contracts/libraries/DODOV2.sol SHA3: 14b6ff7ac7f64d317f718203323fa4a44e89887a7591f738469037bcbe19b6e5 File: contracts/libraries/Fulcrom.sol SHA3: 75f2f4de7cac4f6e1b101cbd440af724d8cb466bfceda7baa55fd67153507c0f File: contracts/libraries/Hashflow.sol SHA3: 307a326fedef3017f293baa4087d5690e2e8f823bee12a6868bf8f8bb69c87f5 File: contracts/libraries/Level.sol SHA3: 5946a80d0c66be0f83c3eaf6c0d4f84b70b7638f0346a614e87edfcace9a20a3 File: contracts/libraries/PancakeSwapStable.sol SHA3: 3667bc3a47d75d31a9280c12a6e1e34581b9d88effe7b9498fa0abd95056c619 File: contracts/libraries/Saddle.sol SHA3: a0bdd201fe01810ebaeaaad563831a61bd3bfefe6d5a7f04a300dd7ca2ac2b12 File: contracts/libraries/UniswapV2.sol SHA3: 2d4aecfbf5f0594b69181346c6f72528575c9d0c93cf02233dd93281d3c41367 File: contracts/libraries/UniswapV3.sol SHA3: 77f133dbe17355c295f0991ecf92b84896d570525a4b5e6d236264651a6a5e8c

	File: contracts/libraries/Wombat.sol SHA3: 61bd2fa4168f5d96a97f1b36a1e01f0253f71dd4fd7792b4ce244087a5816bdc File: contracts/libraries/WooFi.sol SHA3: e8516905ab709f989fe348e19b6f6666041bed566fdb91002d221f8cfbab5f78

Second review scope

Repository	https://github.com/wowswap-io/wowmax-contracts
Commit	e34a1be3c45996ba52861a1fa4ec843071a20b37
Whitepaper	Link
Requirements	Link
Technical Requirements	Link
Contracts	File: contracts/WowmaxRouter.sol SHA3: 3a2369275bbd5689ff13b2edff3ddeaaa60bc16a00cf18942e7a73433c3ff766 File: contracts/WowmaxSwapReentrancyGuard.sol SHA3: 33746522269301071b49b107f3967b950d3a688fc6faf34d28a9cdd6609af0b9 File: contracts/interfaces/IWETH.sol SHA3: cbbf604794519641fde2247a0263def580d225dfd8ef3acef5f0e9dd9576ed77 File: contracts/interfaces/IWowmaxRouter.sol SHA3: 45b47da1fb93c9e3393805c87fde2037be003ab5b2d15a923073eb0441198b97 File: contracts/libraries/Curve.sol SHA3: 7183a160685b927e598a5dd08e7d4e8a5a31a18c0eb7735aee2dc13d088f9c14 File: contracts/libraries/DODOV1.sol SHA3: ace71a581d8cc833286344a6efc66404f5865fad60bbffc5b08cc0a6d02f3c76 File: contracts/libraries/DODOV2.sol SHA3: 5d386f137ec37fa5a271e27a848a42a4dff489bbd5b1b33c8079428c608bf3bd File: contracts/libraries/Fulcrom.sol SHA3: 89c37345ff0f265c538e30eb5567379869ee274d4936ed1f7b5ecdc59c0b21b0 File: contracts/libraries/Hashflow.sol SHA3: bb561ec996c807a59a74a92e63364778b39b8cb65f44fb9a5984bec4f4a831cc File: contracts/libraries/Level.sol SHA3: 41042c4dd7ff21c13145792e2ecea7cc4a4d35759cedded3469732c89196eb49 File: contracts/libraries/PancakeSwapStable.sol SHA3: c850463a378f8c8cf61752b08b37e179bd87f6f56182a557756657efe50605dd File: contracts/libraries/Saddle.sol SHA3: d8865ff72d1af012497ccf849375d640eae896276254a50e4cc2305e2342eb33 File: contracts/libraries/UniswapV2.sol SHA3: 024f067cd50dd6f71cd44c9747c17f289b7840af3517548e53ea5dad04b19893 File: contracts/libraries/UniswapV3.sol

	SHA3: fb4e25430a774906434a84f97bb0e761b8211e75ba79618675cfbedd45bd7261 File: contracts/libraries/Wombat.sol SHA3: 9a50f2651560ae1fb53fa78bfebaf915eaa0126e043133e042a7237562d430c0 File: contracts/libraries/WooFi.sol SHA3: ec579cfeb5f69c46a49e8389d48c4aea49ee710f8d0b2aadfdff9b83ebff318e