CredShields

Smart Contract Audit

Oct 15th, 2023 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of 9mm between Oct 2nd, 2023, and Oct 7th, 2023. And a retest was performed on Oct 10th, 2023.

Author

Shashank (Co-founder, CredShields)

shashank@CredShields.com

Reviewers

Aditya Dixit (Research Team Lead)

aditya@CredShields.com

Prepared for

9mm

Table of Contents

CRED SHIELDS

1. Executive Summary

9mm engaged CredShields to perform a smart contract audit from Oct 2nd, 2023, to Oct 7th, 2023. During this timeframe, Ten (10) vulnerabilities were identified. **A retest was performed on Oct 10th, 2023, and all the bugs have been addressed.**

During the audit, Zero (0) vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "9mm" and should be prioritized for remediation, and fortunately, none were found.

The table below shows the in-scope assets and a breakdown of findings by severity per asset. Section 2.3 contains more information on how severity is calculated.

Assets in Scope	Critical	High	Medium	Low	info	Gas	Σ
Smart Contract	0	0	0	2	2	6	**10**
	0	**0**	**0**	**2**	**2**	**6**	**10**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in Smart Contract's scope during the testing window while abiding by the policies set forth by Smart Contract's team.

State of Security

To maintain a robust security posture, it is essential to continuously review and improve upon current security processes. Utilizing CredShields' continuous audit feature allows both 9mm's internal security and development teams to not only identify specific vulnerabilities, but also gain a deeper understanding of the current security threat landscape.

To ensure that vulnerabilities are not introduced when new features are added, or code is refactored, we recommend conducting regular security assessments. Additionally, by analyzing the root cause of resolved vulnerabilities, the internal teams at 9mm can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, 9mm can future-proof its security posture and protect its assets.

2. Methodology

9mm engaged CredShields to perform a 9mm Smart Contract audit. The following sections cover how the engagement was put together and executed.

2.1 Preparation phase

The CredShields team meticulously reviewed all provided documents and comments in the smart-contract code to gain a thorough understanding of the contract's features and functionalities. They meticulously examined all functions and created a mind map to systematically identify potential security vulnerabilities, prioritizing those that were more critical and business-sensitive for the refactored code. To confirm their findings, the team deployed a self-hosted version of the smart contract and performed verifications and validations during the audit phase.

A testing window from Oct 2nd, 2023, to Oct 7th, 2023, was agreed upon during the preparation phase.

2.1.1 Scope

During the preparation phase, the following scope for the engagement was agreed-upon:

IN SCOPE ASSETS
https://github.com/9mmPro/9mm-v3-contracts/tree/e1b6591dca9ac918c4e05a19afa0434024a0c438

Table: List of Files in Scope

2.1.2 Documentation

Documentation was not required as the code was self-sufficient for understanding the project.

2.1.3 Audit Goals

CredShields uses both in-house tools and manual methods for comprehensive smart contract security auditing. The majority of the audit is done by manually reviewing the contract source code, following SWC registry standards, and an extended industry standard self-developed checklist. The team places emphasis on understanding core concepts, preparing test cases, and evaluating business logic for potential vulnerabilities.

2.2 Retesting phase

9mm is actively partnering with CredShields to validate the remediations implemented towards the discovered vulnerabilities.

2.3 Vulnerability classification and severity

CredShields follows OWASP's Risk Rating Methodology to determine the risk associated with discovered vulnerabilities. This approach considers two factors - Likelihood and Impact - which are evaluated with three possible values - **Low**, **Medium**, and **High**, based on factors such as Threat agents, Vulnerability factors, Technical and Business Impacts. The overall severity of the risk is calculated by combining the likelihood and impact estimates.

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Overall, the categories can be defined as described below -

1. **Informational**

   We prioritize technical excellence and pay attention to detail in our coding practices. Our guidelines, standards, and best practices help ensure software stability and reliability. Informational vulnerabilities are opportunities for improvement and do

not pose a direct risk to the contract. Code maintainers should use their own judgment on whether to address them.

2. Low

Low-risk vulnerabilities are those that either have a small impact or can't be exploited repeatedly or those the client considers insignificant based on their specific business circumstances.

3. Medium

Medium-severity vulnerabilities are those caused by weak or flawed logic in the code and can lead to exfiltration or modification of private user information. These vulnerabilities can harm the client's reputation under certain conditions and should be fixed within a specified timeframe.

4. High

High-severity vulnerabilities pose a significant risk to the Smart Contract and the organization. They can result in the loss of funds for some users, may or may not require specific conditions, and are more complex to exploit. These vulnerabilities can harm the client's reputation and should be fixed immediately.

5. Critical

Critical issues are directly exploitable bugs or security vulnerabilities that do not require specific conditions. They often result in the loss of funds and Ether from Smart Contracts or users and put sensitive user information at risk of compromise

or modification. The client's reputation and financial stability will be severely impacted if these issues are not addressed immediately.

6. **Gas**

   To address the risk and volatility of smart contracts and the use of gas as a method of payment, CredShields has introduced a "Gas" severity category. This category deals with optimizing code and refactoring to conserve gas.

2.4 CredShields staff

The following individual at CredShields managed this engagement and produced this report:

- **Shashank, Co-founder CredShields**
    - shashank@CredShields.com

Please feel free to contact this individual with any questions or concerns you have around the engagement or this document.

3. Findings

This chapter contains the results of the security assessment. Findings are sorted by their severity and grouped by the asset and SWC classification. Each asset section will include a summary. The table in the executive summary contains the total number of identified security vulnerabilities per asset per risk indication.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, Ten (10)security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Use Ownable2Step	Low	Missing Best Practices
Floating and Outdated Pragma	Low	Floating Pragma (SWC-103)
Require With Empty Message	Informational	Missing Best Practices
Hardcoded Static Address	Informational	Missing Best Practises
Large Number Literals	Gas	Gas & Missing Best Practices
Use of SafeMath	Gas	Gas Optimization
Cheaper Conditional Operators	Gas	Gas Optimization

Array Length Caching	Gas	Gas Optimization
Gas Optimization in Require Statements	Gas	Gas Optimization
Gas Optimization in Increments	Gas	Gas optimization

Table: Findings in Smart Contracts

3.1.2 Findings Summary

SWC ID	SWC Checklist	Test Result	Notes
SWC-100	Function Default Visibility	Not Vulnerable	Not applicable after v0.5.X (Currently using solidity v >= 0.8.6)
SWC-101	Integer Overflow and Underflow	Not Vulnerable	The issue persists in versions before v0.8.X.
SWC-102	Outdated Compiler Version	Not Vulnerable	Version 0^.8.0 and above is used
SWC-103	Floating Pragma	Vulnerable	Contract uses floating pragma
SWC-104	Unchecked Call Return Value	Not Vulnerable	call() is not used
SWC-105	Unprotected Ether Withdrawal	Not Vulnerable	Appropriate function modifiers and require validations are used on sensitive functions that allow token or ether withdrawal.
SWC-106	Unprotected SELFDESTRUCT Instruction	Not Vulnerable	selfdestruct() is not used anywhere
SWC-107	Reentrancy	Not Vulnerable	No notable functions were vulnerable to it.
SWC-108	State Variable Default Visibility	Not Vulnerable	Not Vulnerable
SWC-109	Uninitialized Storage Pointer	Not Vulnerable	Not vulnerable after compiler version, v0.5.0

SWC-110	Assert Violation	Not Vulnerable	Asserts are not in use.
SWC-111	Use of Deprecated Solidity Functions	Not Vulnerable	None of the deprecated functions like block.blockhash(), msg.gas, throw, sha3(), callcode(), suicide() are in use
SWC-112	Delegatecall to Untrusted Callee	Not Vulnerable	Not Vulnerable.
SWC-113	DoS with Failed Call	Not Vulnerable	No such function was found.
SWC-114	Transaction Order Dependence	Not Vulnerable	Not Vulnerable.
SWC-115	Authorization through tx.origin	Not Vulnerable	tx.origin is not used anywhere in the code
SWC-116	Block values as a proxy for time	Not Vulnerable	Block.timestamp is not used
SWC-117	Signature Malleability	Not Vulnerable	Not used anywhere
SWC-118	Incorrect Constructor Name	Not Vulnerable	All the constructors are created using the constructor keyword rather than functions.
SWC-119	Shadowing State Variables	Not Vulnerable	Not applicable as this won't work during compile time after version 0.6.0
SWC-120	Weak Sources of Randomness from Chain Attributes	Not Vulnerable	Random generators are not used.
SWC-121	Missing Protection against Signature Replay Attacks	Not Vulnerable	No such scenario was found

SWC-122	Lack of Proper Signature Verification	Not Vulnerable	Not used anywhere
SWC-123	Requirement Violation	Not Vulnerable	Not vulnerable
SWC-124	Write to Arbitrary Storage Location	Not Vulnerable	No such scenario was found
SWC-125	Incorrect Inheritance Order	Not Vulnerable	No such scenario was found
SWC-126	Insufficient Gas Griefing	Not Vulnerable	No such scenario was found
SWC-127	Arbitrary Jump with Function Type Variable	Not Vulnerable	Jump is not used.
SWC-128	DoS With Block Gas Limit	Not Vulnerable	Not Vulnerable.
SWC-129	Typographical Error	Not Vulnerable	No such scenario was found
SWC-130	Right-To-Left-Override control character (U+202E)	Not Vulnerable	No such scenario was found
SWC-131	Presence of unused variables	Not Vulnerable	No such scenario was found
SWC-132	Unexpected Ether balance	Not Vulnerable	No such scenario was found
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Not Vulnerable	abi.encodePacked() or other functions are not used.
SWC-134	Message call with hardcoded gas amount	Not Vulnerable	Not used anywhere in the code
SWC-135	Code With No Effects	Not Vulnerable	No such scenario was found
SWC-136	Unencrypted Private Data On-Chain	Not Vulnerable	No such scenario was found

4. Remediation Status

9mm is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on Oct 10th, 2023, and all the issues have been addressed.**

Also, the table shows the remediation status of each finding.

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Use Ownable2Step	Low	**Won't Fix**
Floating and Outdated Pragma	Low	**Won't Fix**
Require With Empty Message	Informational	**Won't Fix**
Hardcoded Static Address	Informational	**Won't Fix**
Large Number Literals	Gas	**Won't Fix**
Use of SafeMath	Gas	**Won't Fix**
Cheaper Conditional Operators	Gas	**Won't Fix**
Array Length Caching	Gas	**Won't Fix**
Gas Optimization in Require Statements	Gas	**Won't Fix**
Gas Optimization in Increments	Gas	**Won't Fix**

Table: Summary of findings and status of remediation

5. Bug Reports

Bug ID #1 [Won't Fix]

Use Ownable2Step

Vulnerability Type

Missing Best Practices

Severity

Low

Description

The "Ownable2Step" pattern is an improvement over the traditional "Ownable" pattern, designed to enhance the security of ownership transfer functionality in a smart contract. Unlike the original "Ownable" pattern, where ownership can be transferred directly to a specified address, the "Ownable2Step" pattern introduces an additional step in the ownership transfer process. Ownership transfer only completes when the proposed new owner explicitly accepts the ownership, mitigating the risk of accidental or unintended ownership transfers to mistyped addresses.

Affected Code

- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/StableSwapRouter.sol#L17-L122

Impacts

Without the "Ownable2Step" pattern, the contract owner might inadvertently transfer ownership to an unintended or mistyped address, potentially leading to a loss of control

over the contract. By adopting the "Ownable2Step" pattern, the smart contract becomes more resilient against external attacks aimed at seizing ownership or manipulating the contract's behavior.

Remediation

It is recommended to use either Ownable2Step or Ownable2StepUpgradeable depending on the smart contract.

Retest:

It is a best pratice to implement and absence of it will not cause any active exploitation. Hence this will remain as it is.

Bug ID #2 [Won't Fix]

Floating and Outdated Pragma

Vulnerability Type

Floating Pragma (SWC-103)

Severity

Low

Description

Locking the pragma helps ensure that the contracts do not accidentally get deployed using an older version of the Solidity compiler affected by vulnerabilities.

The contract allowed floating or unlocked pragma to be used.

This allows the contracts to be compiled with all the solidity compiler versions above the limit specified.

Affected Code

- All the contract files

Impacts

If the smart contract gets compiled and deployed with an older or too recent version of the solidity compiler, there's a chance that it may get compromised due to the bugs present in the older versions or unidentified exploits in the new versions.

Incompatibility issues may also arise if the contract code does not support features in other compiler versions, therefore, breaking the logic.

The likelihood of exploitation is really low therefore this is only informational.

Remediation

Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.20 pragma version.

Since most of the code is developed for older versions, it is highly recommended to carefully consider this remediation because it may break the code or introduce other inconsistencies.

Reference: https://swcregistry.io/docs/SWC-103

Retest

It is a best pratice to implement and absence of it will not cause any active exploitation.
Hence this will remain as it is.

Bug ID #3 [Won't Fix]

Require With Empty Message

Vulnerability Type

Missing Best Practices

Severity

informational

Description

During code analysis, it has been observed that some require statements lack descriptive messages, which provide crucial information to users when conditions are not met. These messages, limited to 32 bytes, improve user understanding of why a transaction was reverted.

Vulnerable Code

- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-core/PancakeV3Factory.sol#L65-L65
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-core/PancakeV3Factory.sol#L67-L67
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-core/PancakeV3Factory.sol#L74-L74
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-core/PancakeV3Factory.sol#L90-L90
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-core/PancakeV3Factory.sol#L94-L94

- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-core/PancakeV3Factory.sol#L95-L95
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-core/PancakeV3Factory.sol#L117-L117
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/V3SwapRouter.sol#L41-L41
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/V3SwapRouter.sol#L120-L120
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/V3SwapRouter.sol#L159-L159
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/V3SwapRouter.sol#L195-L195
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/V3SwapRouter.sol#L214-L214
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/V3SwapRouter.sol#L229-L229
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/V2SwapRouter.sol#L78-L78
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/V2SwapRouter.sol#L91-L91
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/base/PoolInitializer.sol#L19-L19
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/SwapRouter.sol#L62-L62
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/SwapRouter.sol#L199-L199
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/base/PeripheryPaymentsWithFee.sol#L23-L23

- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/base/PeripheryPaymentsWithFee.sol#L44-L44
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/NonfungiblePositionManager.sol#L191-L191
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/NonfungiblePositionManager.sol#L266-L266
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/NonfungiblePositionManager.sol#L270-L270
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/NonfungiblePositionManager.sol#L317-L317
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/base/ApproveAndCall.sol#L31-L31
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/base/ApproveAndCall.sol#L42-L42
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/base/ApproveAndCall.sol#L47-L47
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/base/ApproveAndCall.sol#L52-L52
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/base/ApproveAndCall.sol#L53-L53
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/base/ApproveAndCall.sol#L58-L58
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/base/ApproveAndCall.sol#L59-L59
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/base/OracleSlippage.sol#L28-L28
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/base/OracleSlippage.sol#L41-L41

- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/base/OracleSlippage.sol#L118-L118
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/base/OracleSlippage.sol#L131-L131
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/base/OracleSlippage.sol#L157-L157
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/base/OracleSlippage.sol#L169-L169
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-core/PancakeV3Pool.sol#L154-L154
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-core/PancakeV3Pool.sol#L165-L165
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-core/PancakeV3Pool.sol#L199-L199
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-core/PancakeV3Pool.sol#L208-L208
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-core/PancakeV3Pool.sol#L487-L487
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-core/PancakeV3Pool.sol#L869-L872
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/StableSwapRouter.sol#L39-L39
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/StableSwapRouter.sol#L52-L52
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/StableSwapRouter.sol#L90-L90
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/StableSwapRouter.sol#L110-L110

Impacts

Users may be left without clear context when a transaction is reverted due to unmet conditions, leading to confusion and frustration

Remediation

Add concise, informative messages to require statements, explaining why the condition failed. Ensure messages are clear and within the 32-byte limit.

Retest

Gas optimizations are not exploiatble it more of a code optimization and hence the team decided not to fix it.

Bug ID #4 [Won't Fix]

Hardcoded Static Address

Vulnerability Type
Missing Best Practises

Severity
Informational

Description
The contracts were found to be using hardcoded addresses.
This could have been optimized using dynamic address update techniques along with proper access control to aid in address upgrade at a later stage.

Affected Code

- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/NonfungibleTokenPositionDescriptor.sol#L20-L20
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/NonfungibleTokenPositionDescriptor.sol#L21-L21
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/NonfungibleTokenPositionDescriptor.sol#L22-L22
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/NonfungibleTokenPositionDescriptor.sol#L23-L23
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/NonfungibleTokenPositionDescriptor.sol#L24-L24

Impacts
Hardcoding address variables in the contract make it difficult for it to be modified at a later stage in the contract as everything will need to be deployed again at a different address if there's a code upgrade.

Remediation

It is recommended to create dynamic functions to address upgrades so that it becomes easier for developers to make changes at a later stage if necessary.
The said function should have proper access controls to make sure only administrators can call that function using access control modifiers.
There should also be a zero address validation in the function to make sure the tokens are not lost.
If the address is supposed to be hardcoded, it is advisable to make it a constant if its value is not getting updated.

Retest
Gas optimizations are not exploiatble it more of a code optimization and hence the team decided not to fix it.

Bug ID #5 [Won't Fix]

Large Number Literals

Vulnerability Type
Gas & Missing Best Practices

Severity
Gas

Description
Solidity supports multiple rational and integer literals, including decimal fractions and scientific notations. The use of very large numbers with too many digits was detected in the code that could have been optimized using a different notation also supported by Solidity.

Affected Code

- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-core/PancakeV3Factory.sol#L90-L90
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/NFTDescriptorEx.sol#L288-L288

Impacts
Having a large number literals in the code increases the gas usage of the contract during its deployment and when the functions are used or called from the contract.
It also makes the code harder to read and audit and increases the chances of introducing code errors.

Remediation
Scientific notation in the form of 2e10 is also supported, where the mantissa can be fractional, but the exponent has to be an integer. The literal MeE is equivalent to M * 10**E. Examples include 2e10, 2e10, 2e-10, 2.5e1, as suggested in official solidity documentation.
https://docs.soliditylang.org/en/latest/types.html#rational-and-integer-literals
It is recommended to use numbers in the form "35 * 1e7 * 1e18" or "35 * 1e25".

The numbers can also be represented by using underscores between them to make them more readable such as “35_00_00_000”

Retest
Gas optimizations are not exploiatble it more of a code optimization and hence the team decided not to fix it.

Bug ID #6 [Won't Fix]

Use of SafeMath

Vulnerability Type
Gas Optimization

Severity
Gas

Description:
SafeMath library is found to be used in the contract. This increases gas consumption more than traditional methods and validations if done manually.
Also, Solidity **0.8.0** and above includes checked arithmetic operations by default, and this renders SafeMath unnecessary.

Affected Code:

- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/NFTDescriptorEx.sol#L19-L19

Impacts:
This increases the gas usage of the contract.

Remediation:
We do not recommend using the SafeMath library for all arithmetic operations. It is good practice to use explicit checks where it is really needed and to avoid extra checks where overflow/underflow is impossible.
It is recommended to upgrade to the latest compiler because versions above 0.8.0+ automatically check for overflows and underflows.

Retest:
Gas optimizations are not exploiatble it more of a code optimization and hence the team decided not to fix it.

Bug ID #7 [Won't Fix]

Cheaper Conditional Operators

Vulnerability Type

Gas Optimization

Severity

Gas

Description

Upon reviewing the code, it has been observed that the contract uses conditional statements involving comparisons with unsigned integer variables. Specifically, the contract employs the conditional operator x > 0 interchangeably. However, it's important to note that during compilation, x != 0 is generally more cost-effective than x > 0 for unsigned integers within conditional statements.

Affected Code

- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/base/PeripheryPayments.sol#L23-L23
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/base/PeripheryPayments.sol#L38-L38
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/base/PeripheryPayments.sol#L45-L45
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/V3Migrator.sol#L39-L39
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/lens/Quoter.sol#L34-L34

- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/SwapRouter.sol#L62-L62
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/NonfungiblePositionManager.sol#L317-L317
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/NonfungiblePositionManager.sol#L330-L330
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/NFTDescriptorEx.sol#L117-L117
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/NFTDescriptorEx.sol#L301-L301
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/lens/MixedRouteQuoterV1.sol#L64-L64
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/lens/QuoterV2.sol#L37-L37
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/lens/QuoterV2.sol#L46-L46
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-core/PancakeV3Factory.sol#L94-L94
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/V3SwapRouter.sol#L41-L41
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/base/LiquidityManagement.sol#L33-L33
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/base/LiquidityManagement.sol#L34-L34
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/base/PeripheryPaymentsWithFee.sol#L23-L23
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/base/PeripheryPaymentsWithFee.sol#L28-L28

- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/base/PeripheryPaymentsWithFee.sol#L31-L31
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/base/PeripheryPaymentsWithFee.sol#L49-L49
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/base/PeripheryPaymentsWithFee.sol#L51-L51
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/lens/Quoter.sol#L43-L43
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-core/PancakeV3Pool.sol#L487-L487
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-core/PancakeV3Pool.sol#L487-L487
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-core/PancakeV3Pool.sol#L502-L502
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-core/PancakeV3Pool.sol#L506-L506

Impacts

Employing x != 0 in conditional statements can result in reduced gas consumption compared to using x > 0. This optimization contributes to cost-effectiveness in contract and user interactions.

Remediation

Whenever possible, use the x != 0 conditional operator instead of x > 0 for unsigned integer variables in conditional statements.

Retest

Gas optimizations are not exploiatble it more of a code optimization and hence the team decided not to fix it.

Bug ID #8 [Won't Fix]

Array Length Caching

Vulnerability Type

Gas Optimization

Severity

Gas

Description

During each iteration of the loop, reading the length of the array uses more gas than is necessary. In the most favorable scenario, in which the length is read from a memory variable, storing the array length in the stack can save about 3 gas per iteration. In the least favorable scenario, in which external calls are made during each iteration, the amount of gas wasted can be significant.

Affected Code

- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/base/Multicall.sol#L13-L26
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/NFTDescriptorEx.sol#L112-L116
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/V2SwapRouter.sol#L24-L42
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/lens/PancakeInterfaceMulticall.sol#L30-L37
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/base/OracleSlippage.sol#L138-L144
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/StableSwapRouter.sol#L54-L60

- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/lens/TokenValidator.sol#L40-L42
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/lens/TokenValidator.sol#L50-L55

Impacts

Reading the length of an array multiple times in a loop by calling .length costs more gas.

Remediation

Consider storing the array length of the variable before the loop and use the stored length instead of fetching it in each iteration.

Retest

Gas optimizations are not exploiatble it more of a code optimization and hence the team decided not to fix it.

Bug ID #9 [Won't Fix]

Gas Optimization in Require Statements

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The **require()** statement takes an input string to show errors if the validation fails.
The strings inside these functions that are longer than **32 bytes** require at least one additional MSTORE, along with additional overhead for computing memory offset, and other parameters. For this purpose, having strings lesser than 32 bytes saves a significant amount of gas.

Vulnerable Code

- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/base/ERC721Permit.sol#L74-L74
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/NonfungiblePositionManager.sol#L391-L391
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-core/PancakeV3Factory.sol#L72-L72

Impacts
Having longer require strings than 32 bytes costs a significant amount of gas.

Remediation
It is recommended to shorten the strings passed inside **require()** statements to fit under **32 bytes**. This will decrease the gas usage at the time of deployment and at runtime when the validation condition is met.

Retest

Gas optimizations are not exploiatble it more of a code optimization and hence the team decided not to fix it.

Bug ID #10 [Won't Fix]

Gas Optimization in Increments

Vulnerability Type
Gas optimization

Severity
Gas

Description
The contract uses **for** loops that use post increments for the variable "**i**". The contract can save some gas by changing this to **++i**.
++i costs less gas compared to **i++** or **i += 1** for unsigned integers. In **i++**, the compiler has to create a temporary variable to store the initial value. This is not the case with **++i** in which the value is directly incremented and returned, thus, making it a cheaper alternative.

Vulnerable Code

- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/lens/TickLens.sol#L23-L23
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/lens/TickLens.sol#L30-L30
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/base/Multicall.sol#L13-L13
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/NFTDescriptorEx.sol#L112-L112
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/NFTDescriptorEx.sol#L243-L243
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/V2SwapRouter.sol#L24-L24
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/NonfungibleTokenPositionDescriptor.sol#L46-L46
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/v3-periphery/lens/PancakeInterfaceMulticall.sol#L30-L30

- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/base/OracleSlippage.sol#L69-L69
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/base/OracleSlippage.sol#L138-L138
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/StableSwapRouter.sol#L54-L54
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/lens/TokenValidator.sol#L40-L40
- https://github.com/9mmPro/9mm-v3-contracts/blob/e1b6591dca9ac918c4e05a19afa0434024a0c438/router/lens/TokenValidator.sol#L50-L50

Impacts

Using **i++** instead of **++i** costs the contract deployment around 600 more gas units.

Remediation

It is recommended to switch to **++i** and change the code accordingly so the function logic remains the same and saves some gas.

Retest

Gas optimizations are not exploiatble it more of a code optimization and hence the team decided not to fix it.

6. Disclosure

The Reports provided by CredShields is not an endorsement or condemnation of any specific project or team and do not guarantee the security of any specific project. The contents of this report are not intended to be used to make decisions about buying or selling tokens, products, services, or any other assets and should not be interpreted as such.

Emerging technologies such as Smart Contracts and Solidity carry a high level of technical risk and uncertainty. CredShields does not provide any warranty or representation about the quality of code, the business model or the proprietors of any such business model, or the legal compliance of any business. The report is not intended to be used as investment advice and should not be relied upon as such.

CredShields Audit team is not responsible for any decisions or actions taken by any third party based on the report.