



# CredShields

# Smart Contract Audit

---

**Sept 21st, 2023 • CONFIDENTIAL**

## **Description**

This document details the process and result of the SWYP Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of SWYP Foundation between Sept 16th, 2023, and Sept 19th, 2023. A retest was performed on September 20th, 2023.

## **Author**

Shashank (Co-founder, CredShields)

[shashank@CredShields.com](mailto:shashank@CredShields.com)

## **Reviewers**

Aditya Dixit (Research Team Lead)

[aditya@CredShields.com](mailto:aditya@CredShields.com)

## **Prepared for**

SWYP Foundation

# Table of Contents

<b>1. Executive Summary</b>	<b>2</b>
State of Security	4
<b>2. Methodology</b>	<b>5</b>
2.1 Preparation phase	5
2.1.1 Scope	6
2.1.2 Documentation	6
2.1.3 Audit Goals	6
2.2 Retesting phase	7
2.3 Vulnerability classification and severity	7
2.4 CredShields staff	10
<b>3. Findings</b>	<b>11</b>
3.1 Findings Overview	11
3.1.1 Vulnerability Summary	11
3.1.2 Findings Summary	12
<b>4. Remediation Status</b>	<b>16</b>
<b>5. Bug Reports</b>	<b>17</b>
Bug ID #1	17
Use Ownable2Step	17
Bug ID #2	19
Floating and Outdated Pragma	19
Bug ID#3	21
Large Number Literals	21
Bug ID#4	22
Missing NatSpec Comments	22
Bug ID#5	24
Functions should be declared External	24
Bug ID#6	26
Missing Events in Important Functions	26
Bug ID#7	28
Bypassing Max Transaction Limit	28
<b>6. Disclosure</b>	<b>31</b>

# 1. Executive Summary

SWYP Foundation engaged CredShields to perform a smart contract audit from Sept 16th, 2023, to Sept 19th, 2023. During this timeframe, Seven (7) vulnerabilities were identified. **A retest was performed on Sept 20th, 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 "SWYP Foundation" 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
SWYP Smart Contract	0	0	0	3	3	1	7
	0	0	0	3	3	1	7

*Table: Vulnerabilities Per Asset in Scope*

The CredShields team conducted the security audit to focus on identifying vulnerabilities in SWYP Smart Contract's scope during the testing window while abiding by the policies set forth by SWYP 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 SWYP Foundation'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 SWYP Foundation can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, SWYP Foundation can future-proof its security posture and protect its assets.

## 2. Methodology

---

SWYP Foundation engaged CredShields to perform a SWYP Foundation 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 Sept 16th, 2023, to Sept 19th, 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
<a href="https://github.com/vineethmk121/swyp.foundation/blob/main/swyp.foundation">https://github.com/vineethmk121/swyp.foundation/blob/main/swyp.foundation</a>

*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

SWYP Foundation 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](mailto: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, Seven (7) security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC   Vulnerability Type
Use Ownable2Step	Informational	Missing Best Practices
Floating and Outdated Pragma	Low	Floating Pragma (SWC-103)
Large Number Literals	Gas	Gas & Missing Best Practices
Missing NatSpec Comments	Informational	Missing best practices
Functions should be declared External	Informational	Best Practices
Missing Events in Important Functions	Low	Missing Best Practices

Bypassing Max Transaction Limit	Low	Business Logic
---------------------------------	-----	----------------

*Table: Findings in Smart Contracts*

### 3.1.2 Findings Summary

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

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

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





## 4. Remediation Status

SWYP Foundation is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on Sept 20th, 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	Informational	Won't Fix 20/09/2023
Floating and Outdated Pragma	Low	Won't Fix 20/09/2023
Large Number Literals	Gas	Won't Fix 20/09/2023
Missing NatSpec Comments	Informational	Won't Fix 20/09/2023
Functions should be declared External	Informational	Won't Fix 20/09/2023
Missing Events in Important Functions	Low	Won't Fix 20/09/2023
Bypassing Max Transaction Limit	Low	Won't Fix 20/09/2023

*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

Informational

### 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/vineethmk121/swyp.foundation/blob/d836d041e8921b4e2cc4bf6e2d18e7023b8fff5/swyp.foundation#L641>

### 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**

**The team mentioned that they are okay with the current implementation and the CredShields team agrees to the decision.**

## 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, i.e., `>= 0.8.9`. This allows the contracts to be compiled with all the solidity compiler versions above the limit specified.

The following contracts were found to be affected -

#### Affected Code

- <https://github.com/vineethmk121/swyp.foundation/blob/d836d041e8921b4e2cc4fbf6e2d18e7023b8fff5/swyp.foundation#L2>

#### 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 0.8.18 pragma version

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

## **Retest**

**Since this is not exploitable hence the team decided not to fix it.**

## Bug ID#3 [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/vineethmk121/swyp.foundation/blob/d836d041e8921b4e2cc4fbf6e2d18e7023b8fff5/swyp.foundation#L645>
- <https://github.com/vineethmk121/swyp.foundation/blob/d836d041e8921b4e2cc4fbf6e2d18e7023b8fff5/swyp.foundation#L652>

#### Impacts

Having 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**

**The team is okay with a significant increase in gas cost and hence this was not fixed.**

Bug ID#4 [Won't Fix]

## Missing NatSpec Comments

### Vulnerability Type

Missing best practices

### Severity

Informational

### Description

Solidity contracts use a special form of comments to document code. This special form is named the Ethereum Natural Language Specification Format (NatSpec).

The document is divided into descriptions for developers and end-users along with the title and the author.

### Affected Code

- contract SWYP

### Impacts

Missing NatSpec comments and documentation about a library or a contract affect the audit and future development of the smart contracts.

### Remediation

Add necessary NatSpec comments inside the library along with documentation specifying what it's for and how it's implemented.

### Retest

This is just additional information for the contract users and hence the team decided not to add comments in the contract.



## Bug ID#5 [Won't Fix]

### Functions should be declared External

#### Vulnerability Type

Best Practices

#### Severity

Informational

#### Description

Public functions that are never called by a contract should be declared external in order to conserve gas.

The following functions were declared as public but were not called anywhere in the contract, making public visibility useless.

#### Affected Code

- <https://github.com/vineethmk121/swyp.foundation/blob/d836d041e8921b4e2cc4fbf6e2d18e7023b8fff5/swyp.foundation#L677-L679>
- <https://github.com/vineethmk121/swyp.foundation/blob/d836d041e8921b4e2cc4fbf6e2d18e7023b8fff5/swyp.foundation#L681-L683>
- <https://github.com/vineethmk121/swyp.foundation/blob/d836d041e8921b4e2cc4fbf6e2d18e7023b8fff5/swyp.foundation#L685-L687>
- <https://github.com/vineethmk121/swyp.foundation/blob/d836d041e8921b4e2cc4fbf6e2d18e7023b8fff5/swyp.foundation#L689-L691>
- <https://github.com/vineethmk121/swyp.foundation/blob/d836d041e8921b4e2cc4fbf6e2d18e7023b8fff5/swyp.foundation#L693-L695>
- <https://github.com/vineethmk121/swyp.foundation/blob/d836d041e8921b4e2cc4fbf6e2d18e7023b8fff5/swyp.foundation#L697-L699>
- <https://github.com/vineethmk121/swyp.foundation/blob/d836d041e8921b4e2cc4fbf6e2d18e7023b8fff5/swyp.foundation#L701-L703>
- <https://github.com/vineethmk121/swyp.foundation/blob/d836d041e8921b4e2cc4fbf6e2d18e7023b8fff5/swyp.foundation#L705-L707>

## Impacts

Smart Contracts are required to have effective Gas usage as they cost real money and each function should be monitored for the amount of gas it costs to make it gas efficient.

**"public"** functions cost more Gas than **"external"** functions.

## Remediation

Use the **"external"** state visibility for functions that are never called from inside the contract.

## Retest

**The above report just helps the contract to be more optimized and is not a security concern. Hence this remains unfixed.**

## Bug ID#6 [Won't Fix]

### Missing Events in Important Functions

#### Vulnerability Type

Missing Best Practices

#### Severity

Low

#### Description

Events are inheritable members of contracts. When you call them, they cause the arguments to be stored in the transaction's log—a special data structure in the blockchain. These logs are associated with the address of the contract which can then be used by developers and auditors to keep track of the transactions.

The contract was found to be missing these events on certain critical functions which would make it difficult or impossible to track these transactions off-chain.

#### Affected Code

- <https://github.com/vineethmk121/swyp.foundation/blob/d836d041e8921b4e2cc4fbf6e2d18e7023b8fff5/swyp.foundation#L685-L687>
- <https://github.com/vineethmk121/swyp.foundation/blob/d836d041e8921b4e2cc4fbf6e2d18e7023b8fff5/swyp.foundation#L689-L691>
- <https://github.com/vineethmk121/swyp.foundation/blob/d836d041e8921b4e2cc4fbf6e2d18e7023b8fff5/swyp.foundation#L693-L695>
- <https://github.com/vineethmk121/swyp.foundation/blob/d836d041e8921b4e2cc4fbf6e2d18e7023b8fff5/swyp.foundation#L697-L699>
- <https://github.com/vineethmk121/swyp.foundation/blob/d836d041e8921b4e2cc4fbf6e2d18e7023b8fff5/swyp.foundation#L701-L703>
- <https://github.com/vineethmk121/swyp.foundation/blob/d836d041e8921b4e2cc4fbf6e2d18e7023b8fff5/swyp.foundation#L705-L707>

#### Impacts

Events are used to track the transactions off-chain and missing these events on critical functions makes it difficult to audit these logs if they're needed at a later stage.

### **Remediation**

Consider emitting events for the functions mentioned above. It is also recommended to have the addresses indexed.

### **Retest**

**Event makes tracking onchain records easy. The team decided not to fix it as it didn't cause any security implications.**

## Bug ID#7 [Won't Fix]

### Bypassing Max Transaction Limit

#### Vulnerability Type

Business Logic

#### Severity

Low

#### Description

The contract employs a "\_maxTransactionLimit" variable as a safeguard against transactions involving exceedingly large token amounts. However, due to the absence of a transaction mapping for individual users, it remains possible to circumvent this transaction limit by simply executing multiple consecutive transactions, each with the maximum allowable amount.

#### Affected Code

- <https://github.com/vineethmk121/swyp.foundation/blob/d836d041e8921b4e2cc4fbf6e2d18e7023b8fff5/swyp.foundation#L669-L671>

#### Impacts

Since there's no mapping on the number of tokens transacted by the user, it could be exploited to bypass the anti-whale mechanisms by doing a maximum token transaction multiple times.

#### Remediation

If the intention of the contract is to prevent transactions of very large amounts in every transfer, the current code is sufficient. But if the logic behind the transaction limit is to limit the user from transferring large amounts throughout the lifetime of the contract, it is recommended to have a mapping that enforces a limit on the total number of tokens the user can transfer.

#### Retest

**The team mentioned that this is the only limit they want to have on tracking and hence this won't be fixed.**

## 6. Disclosure

---

The Reports provided by CredShields are 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.