CredShields

Smart Contract Audit

Aug 30th, 2023 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Solace Protocol between Aug 18th, 2023, and Aug 24th, 2023. A retest was performed on Aug 28th, 2023.

Author

Shashank (Co-founder, CredShields)

shashank@CredShields.com

Reviewers

Aditya Dixit (Research Team Lead)

aditya@CredShields.com

Prepared for

Solace Protocol

Table of Contents

1. Executive Summary

Solace Protocol engaged CredShields to perform a smart contract audit from Aug 18th, 2023, to Aug 24th, 2023. During this timeframe, Nine (9) vulnerabilities were identified. **A retest was performed on Aug 28th, 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 "Solace Protocol" 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	1	2	3	3	**9**
	0	**0**	**1**	**2**	**3**	**3**	**9**

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 Solace Protocol'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 Solace Protocol can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, Solace Protocol can future-proof its security posture and protect its assets.

2. Methodology

Solace Protocol engaged CredShields to perform a Solace Protocol 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 Aug 18th, 2023, to Aug 24th, 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/solace-labs/contracts/tree/3e3929effbbcce9b2f63b25baa36b82754a41eaa

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

Solace Protocol 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, Nine (9) security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Misconfigured ERC-4337 Implementation	Informational	Misconfiguration
Floating and Outdated Pragma	Low	Floating Pragma (SWC-103)
Missing Events in Important Functions	Low	Missing Events in Important Functions
Unnecessary Checked Arithmetic In Loop	Gas	Unnecessary Checked Arithmetic In Loop
Functions should be declared External	Gas	Gas Optimization
Gas Optimization in Increments	Gas	Gas optimization
Missing NatSpec Comments	Informational	Missing best practices

Unused Variables	Informational	Dead Code
Enhancing Value Passing In Batch Function	Medium	Business Logic

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	Not 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

Solace Protocol is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on Aug 28th, 2023, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Misconfigured ERC-4337 Implementation	Informational	**Won't Fix**
Floating and Outdated Pragma	Low	**Fixed [28/08/2023]**
Missing Events in Important Functions	Low	**Fixed [28/08/2023]**
Unnecessary Checked Arithmetic In Loop	Gas	**Fixed [28/08/2023]**
Functions should be declared External	Gas	**Fixed [28/08/2023]**
Gas Optimization in Increments	Gas	**Fixed [28/08/2023]**
Missing NatSpec Comments	Informational	**Fixed [28/08/2023]**
Unused Variables	Informational	**Won't Fix**

Enhancing Value Passing In Batch Function	Medium	**Fixed [28/08/2023]**

Table: Summary of findings and status of remediation

5. Bug Reports

Bug ID #1 [Won't Fix]

Misconfigured ERC-4337 Implementation

Vulnerability Type
Misconfiguration

Severity
Informational

Description
The ERC-4337 standard defines certain structure and return values for the function validateUserOp(). They are:
The return value MUST be packed of authorizer, validUntil and validAfter timestamps.

- authorizer - 0 for valid signature, 1 to mark signature failure. Otherwise, an address of an authorizer contract. This ERC defines "signature aggregator" as authorizer.
- validUntil is 6-byte timestamp value, or zero for "infinite". The UserOp is valid only up to this time.
- validAfter is 6-byte timestamp. The UserOp is valid only after this time.

Even though the contract SolaceAccount.sol was successfully returning 0 and 1 values for signature validation, it was not returning the timestamp values.

Affected Code

- https://github.com/solace-labs/contracts/blob/3e3929effbbcce9b2f63b25baa36b82754a41eaa/contracts/SolaceAccount.sol#L127

Impacts
Right now, there seems to be no security impact, thus, the informational severity.

CRED SHIELDS

Remediation

It is recommended to adhere to the ERC-4337 format when implementing account abstraction.

Retest:

This won't be fixed since there's no impact right now.

Bug ID #2 [Fixed]

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 was allowing floating or unlocked pragma to be used, i.e., **^0.8.12.**
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/solace-labs/contracts/blob/3e3929effbbcce9b2f63b25baa36b82754a41eaa/contracts/SolaceAccount.sol#L2
- https://github.com/solace-labs/contracts/blob/3e3929effbbcce9b2f63b25baa36b82754a41eaa/contracts/SolaceAccountFactory.sol#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 and will depend on the actual pragma version used to compile and deploy the contracts.

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
The pragma version has been fixed to 0.8.19.

Bug ID #3 [Fixed]

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
The following functions were affected -

- https://github.com/solace-labs/contracts/blob/3e3929effbbcce9b2f63b25baa36b82754a41eaa/contracts/SolaceAccount.sol#L150-L152
- https://github.com/solace-labs/contracts/blob/3e3929effbbcce9b2f63b25baa36b82754a41eaa/contracts/SolaceAccount.sol#L159-L164

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

Events have been added.

Bug ID #4 [Fixed]

Unnecessary Checked Arithmetic In Loop

Vulnerability Type
Gas Optimization

Severity
Gas

Description
Upon reviewing the code, it has been identified that the contract uses checked arithmetic operations inside loops where increments occur. However, it's important to note that increments inside loops are unlikely to cause overflow since the transaction will run out of gas before the variable reaches its limits. As a result, using checked arithmetic for increments within loops may be unnecessary and can lead to additional gas consumption.

Affected Code

- https://github.com/solace-labs/contracts/blob/3e3929effbbcce9b2f63b25baa36b82754a41eaa/contracts/SolaceAccount.sol#L93

Impacts
Unnecessary checked arithmetic operations can lead to higher gas consumption, as each arithmetic operation comes with its own gas cost. This can contribute to increased transaction fees and operational costs.

Remediation
Consider having the increment value inside the unchecked block to save some gas.

Retest
The loops have been made unchecked to save gas.

Bug ID#5 [Fixed]

Functions should be declared External

Vulnerability Type
Gas Optimization

Severity
Gas

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 the public visibility useless.

Affected Code
The following functions were affected -

- https://github.com/solace-labs/contracts/blob/3e3929effbbcce9b2f63b25baa36b82754a41eaa/contracts/SolaceAccountFactory.sol#L28-L45
- https://github.com/solace-labs/contracts/blob/3e3929effbbcce9b2f63b25baa36b82754a41eaa/contracts/SolaceAccount.sol#L150-L152
- https://github.com/solace-labs/contracts/blob/3e3929effbbcce9b2f63b25baa36b82754a41eaa/contracts/SolaceAccount.sol#L159-L164

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

Public functions that are not called anywhere have been updated as external.

Bug ID #6 [Fixed]

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/solace-labs/contracts/blob/3e3929effbbcce9b2f63b25baa36b82754a41eaa/contracts/SolaceAccount.sol#L93

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
This has been fixed. ++i is being used to save gas.

Bug ID#7 [Fixed]

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.
The contracts in the scope were missing these comments.

Impacts:
Without Natspec comments, it can be challenging for other developers to understand the code's intended behavior and purpose. This can lead to errors or bugs in the code, making it difficult to maintain and update the codebase. Additionally, it can make it harder for auditors to evaluate the code for security vulnerabilities, increasing the risk of potential exploits.

Remediation:
Developers should review their codebase and add Natspec comments to all relevant functions, variables, and events. Natspec comments should include a description of the function or event, its parameters, and its return values.

Retest
Descriptive comments have been added.

Bug ID#8 [Won't Fix]

Unused Variables

Vulnerability Type
Dead Code

Severity
Informational

Description
It is recommended to keep the production repository clean to prevent confusion and the introduction of vulnerabilities.
The functions and parameters, contracts, and interfaces that are never used or called externally or from inside the contracts should be removed when the contract is deployed on the mainnet.

Affected Code

- https://github.com/solace-labs/contracts/blob/3e3929effbbcce9b2f63b25baa36b82754a41eaa/contracts/SolaceAccount.sol#L20

Impacts
This does not impact the security aspect of the Smart contract but prevents confusion when the code is sent to other developers or auditors to understand and implement.
This reduces the overall size of the contracts and also helps in saving gas.

Remediation
If the variable is not supposed to be used, consider deleting it from the production code.

Retest
This variable is being used for versioning to track future updates.

Bug ID#9 [Fixed]

Enhancing Value Passing In Batch Function

Vulnerability Type
Business Logic

Severity
Medium

Description
Upon reviewing the code, it has been identified that the executeBatch() function is used for batch execution of external calls to other contracts. However, during these external calls, the msg.value is always set to 0, which means no Ether value is transferred with the calls. This may limit the functionality and use cases of the executeBatch() function. To enhance the flexibility and utility of this function, it's recommended to allow the passing of an array to specify the msg.value to be sent along with each call.

Affected Code

- https://github.com/solace-labs/contracts/blob/3e3929effbbcce9b2f63b25baa36b82754a41eaa/contracts/SolaceAccount.sol#L87-L96

Impacts
Without the ability to specify msg.value for each individual call, the contract is unable to perform batch execution of functions that require Ether transfers as part of their execution.

Remediation
Modify the executeBatch() function to accept an additional parameter, an array of uint256 values, representing the msg.value to be sent along with each external call.

Retest
The function has been updated to accept multiple values.

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.