CredShields

Smart Contract Audit

Oct 13th, 2023 • CONFIDENTIAL

Description

This document details the process and result of the Sendit Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Arcana between Sept 28th, 2023, and Oct 4th, 2023. And a retest was performed on Oct 9th, 2023.

Author

Shashank (Co-founder, CredShields)

shashank@CredShields.com

Reviewers

Aditya Dixit (Research Team Lead)

aditya@CredShields.com

Prepared for

Arcana

Table of Contents

1. Executive Summary

Arcana engaged CredShields to perform a smart contract audit from Sept 28th, 2023, to Oct 4th, 2023. During this timeframe, eight (8) vulnerabilities were identified. **A retest was performed on Oct 9th, 2023, and all the bugs have been addressed.**

During the audit, two (2) vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Arcana" and should be prioritized for remediation, and they all have been fixed.

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
Sendit Smart Contract	0	2	0	2	1	3	**8**
	0	**2**	**0**	**2**	**1**	**3**	**8**

Table: Vulnerabilities Per Asset in Scope

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

2. Methodology

Arcana engaged CredShields to perform a Arcana 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 28th, 2023, to Oct 4th, 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/arcana-network/sendit-sc/tree/4892df1c34186f291ed1de9a8a1a85a6b972782b/contracts

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

Arcana 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, eight (8) security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Signature Malleability in Ecrecover	High	Signature Malleability
Cross-Chain Signature Replay Attack	High	Cross-Chain Signature Replay
Outdated Pragma version	Low	Outdated Pragma
Use safeTransfer/safeTransferFrom instead of transfer/transferFrom	Low	Missing best practices
Wrong NatSpec Comments	Informational	Missing best practices
Cheaper Conditional Operators	Gas	Gas Optimization

Unused Imports	Gas	Gas Optimization
Boolean Equality	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	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

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

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Signature Malleability in Ecrecover	High	**Fixed [09/10/2023]**
Cross-Chain Signature Replay Attack	High	**Fixed [09/10/2023]**
Outdated Pragma version	Low	**Fixed [09/10/2023]**
Use safeTransfer/safeTransferFrom instead of transfer/transferFrom	Low	**Fixed [09/10/2023]**
Wrong NatSpec Comments	Informational	**Fixed [09/10/2023]**
Cheaper Conditional Operators	Gas	**Fixed [09/10/2023]**
Unused Imports	Gas	**Fixed [09/10/2023]**
Boolean Equality	Gas	**Fixed**

		[09/10/2023]

Table: Summary of findings and status of remediation

5. Bug Reports

Bug ID #1 [Fixed]

Signature Malleability in Ecrecover

Vulnerability Type
Signature Malleability

Severity
High

Description
Signature Malleability is a vulnerability that can occur when improperly utlizing ECDSA (Elliptic Curve Digital Signature Algorithm) signatures. The vulnerability allows an attacker to change the signature slightly without invalidating the signature itself. This often happens when a smart contract doesn't validate signatures properly, enabling attackers to modify them and potentially bypass security measures.

Affected Code

- https://github.com/arcana-network/sendit-sc/blob/4892df1c34186f291ed1de9a8a1a85a6b972782b/contracts/Sendit.sol#L52-L54

Impacts
Using a vulnerable version of ecrecover could allow users to use the signature signed by the same user twice by manipulating the signature such that it still stays valid. This could lead to a loss of funds.

Remediation
To fix the signature malleability vulnerability, follow these steps:

- Use a well-tested library like OpenZeppelin's ECDSA.sol that already implements a secure signature validation process.
- Make sure to enforce a specific condition on the s value during the signature validation process, ensuring it has a low value. This prevents attackers from manipulating the signature.

Retest

The contract is now using Openzeppelin's ECDSA.sol to remediate this issue.
https://github.com/arcana-network/sendit-sc/blob/1441339f8271e1e0ea29e44dbbcfc23017adcdfb/contracts/Sendit.sol#L100

Bug ID # 2 [Fixed]

Cross-Chain Signature Replay Attack

Vulnerability Type

Cross-Chain Signature Replay

Severity

High

Description

The send() function in the contract appears to be vulnerable to a cross-chain signature replay attack. This type of attack occurs when a signature from one chain is used on another chain, effectively replaying the action in a different context. In this function, a signature is used to validate the request, but there is no differentiation between chains, allowing attackers to potentially use a valid signature from one chain on another.

Affected Code

- https://github.com/arcana-network/sendit-sc/blob/4892df1c34186f291ed1de9a8a1a85a6b972782b/contracts/Sendit.sol#L52-L54

Impacts

If this vulnerability is exploited, it could lead to unintended transfers of assets. An attacker could replay a legitimate request signature from one chain on another chain, causing assets to be transferred to the recipient unintentionally. This could result in financial losses and unexpected behavior in the contract.

Remediation

Add logic to ensure that the request and signature are valid only within the intended chain. This can be achieved by including the chain's identifier or network ID in the data that is signed. When verifying the signature, check that the chain ID matches the expected value.

Retest

Cross-Chain Signature Replay Attac has been fixed using ChainId in in hash .

https://github.com/arcana-network/sendit-sc/blob/1441339f8271e1e0ea29e44dbbcfc23017adcdfb/contracts/Sendit.sol#L87-L99

Bug ID #3 [Fixed]

Outdated Pragma version

Vulnerability Type
Outdated Pragma

Severity
Low

Description
Using an outdated compiler version can be problematic, especially if there are publicly disclosed bugs and issues that affect the current compiler version.
The contracts found in the repository were allowing an old compiler version to be used, i.e., 0.8.8.

Affected Code

- https://github.com/arcana-network/sendit-sc/blob/4892df1c34186f291ed1de9a8a1a85a6b972782b/contracts/Sendit.sol#L2-L3

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 Low severity.

Remediation
Keep the compiler versions updated in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.20 pragma version which is stable and not too recent.
Reference: https://swcregistry.io/docs/SWC-103

Retest

Pragma has been updated to a recent version.

https://github.com/arcana-network/sendit-sc/blob/1441339f8271e1e0ea29e44dbbcfc23017adcdfb/contracts/Sendit.sol#L2

Bug ID #4 [Fixed]

Use safeTransfer/safeTransferFrom instead of transfer/transferFrom

Vulnerability Type

Missing best practices

Severity

Low

Description

The transfer() and transferFrom() method is used instead of safeTransfer() and safeTransferFrom(), presumably to save gas however OpenZeppelin's documentation discourages the use of transferFrom(), use safeTransferFrom() whenever possible because safeTransferFrom auto-handles boolean return values whenever there's an error.

Affected Code

- https://github.com/arcana-network/sendit-sc/blob/4892df1c34186f291ed1de9a8a1a85a6b972782b/contracts/Sendit.sol#L69-L70

Impacts

Using safeTransferFrom has the following benefits -

- It checks the boolean return values of ERC20 operations and reverts the transaction if they fail,
- at the same time allowing you to support some non-standard ERC20 tokens that don't have boolean return values.
- It additionally provides helpers to increase or decrease an allowance, to mitigate an attack possible with vanilla approve.

Remediation

Consider using safeTransfer() and safeTransferFrom() instead of transfer() and transferFrom().

Retest

safeTransferFrom is now being used.

https://github.com/arcana-network/sendit-sc/blob/1441339f8271e1e0ea29e44dbbcfc23017adcdfb/contracts/Sendit.sol#L123

Bug ID #5 [Fixed]

Wrong 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 Sendit contracts are using the wrong NatSpec comment format in the code which won't be parsed by the compiler and will throw an error.

Affected Code

- https://github.com/arcana-network/sendit-sc/blob/4892df1c34186f291ed1de9a8a1a85a6b972782b/contracts/Sendit.sol#L30-L36

Impacts
Due to the incorrect format, these NatSpec comments won't be parsed by the compiler and will throw an error.

Remediation
You may choose "///" for single or multi-line comments, or "/**" and ending with "*/".

Retest
NatSpec comments have been updated.
https://github.com/arcana-network/sendit-sc/blob/1441339f8271e1e0ea29e44dbbcfc23017adcdfb/contracts/Sendit.sol#L44-L54

Bug ID #6 [Fixed]

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 operators x != 0 and 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

The following functions were affected -

- https://github.com/arcana-network/sendit-sc/blob/4892df1c34186f291ed1de9a8a1a85a6b972782b/contracts/Sendit.sol#L42

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 interactions.

Remediation

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

Retest

This has been fixed to save gas. “!=” is now being used.
https://github.com/arcana-network/sendit-sc/blob/1441339f8271e1e0ea29e44dbbcfc23017adcdfb/contracts/Sendit.sol#L68

Bug ID #7 [Fixed]

Unused Imports

Vulnerability Type

Gas Optimization

Severity

Gas

Description

The contract was importing some contracts or libraries that were not used anywhere in the code. This increases the gas cost and the overall contract's complexity.

Affected Code

- https://github.com/arcana-network/sendit-sc/blob/4892df1c34186f291ed1de9a8a1a85a6b972782b/contracts/Sendit.sol#L4

Impacts

Unused imports in smart contracts can lead to an increase in the size of the code, making it more difficult to verify and potentially slowing down its execution. Moreover, having unused code in a smart contract can also increase the attack surface by potentially introducing vulnerabilities that can be exploited by malicious actors. This can lead to security issues and compromise the integrity of the contract.

Additionally, including unused imports in smart contracts can also increase deployment and gas costs, making it more expensive to deploy and run the contract on the Ethereum network.

Remediation

It is recommended to remove the import statement if the external contracts or libraries are not used anywhere in the contract.

Retest

Unused library has been removed from the code.

Bug ID #8 [Fixed]

Boolean Equality

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The contract was found to be equating variables with a boolean constant inside a "require()" statement which is not recommended and is unnecessary. Boolean constants can be used directly in conditionals.

Affected Code

- https://github.com/arcana-network/sendit-sc/blob/4892df1c34186f291ed1de9a8a1a85a6b972782b/contracts/Sendit.sol#L40

Impacts
Equating the values to boolean constants in conditions cost gas and can be used directly.

Remediation
It is recommended to use boolean constants directly. It is not required to equate them to true or false.

Retest:
This has been updated to save gas.
https://github.com/arcana-network/sendit-sc/blob/1441339f8271e1e0ea29e44dbbcfc23017adcdfb/contracts/Sendit.sol#L66

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.