Audited by CRED SHIELDS

CredShields

Smart Contract Audit

August 21, 2025 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Safle Network Pvt. Ltd. between May 28th, 2025, and May 30th, 2025. A retest was performed on June 30th, 2025.

Author
Shashank (Co-founder, CredShields) shashank@CredShields.com

Reviewers
Aditya Dixit (Research Team Lead), Shreyas Koli(Auditor), Naman Jain (Auditor), Sanket Salavi (Auditor), Yash Shah (Auditor), Prasad Kuri (Auditor)

Prepared for
Safle Network Pvt. Ltd.

Table of Contents

1. Executive Summary ----------

Safle Network Pvt. Ltd. engaged CredShields to perform a smart contract audit from May 28th, 2025, to May 30th, 2025. During this timeframe, 19 vulnerabilities were identified. **A retest was performed on June 30th, 2025, and all the bugs have been addressed.**

During the audit, 3 vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Safle Network Pvt. Ltd." and should be prioritized for remediation.

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
Unified ID Contracts	2	1	3	5	1	7	**19**
	2	**1**	**3**	**5**	**1**	**7**	**19**

Table: Vulnerabilities Per Asset in Scope

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

2. The Methodology ----------

Safle Network Pvt. Ltd. engaged CredShields to perform a Unified ID 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 May 28th, 2025, to May 30th, 2025, 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/getsafle/unified-id-contracts/tree/d27d2faf85570be4300db97f1c85527a820fbf7b

2.1.2 Documentation

The Safle team provided us with the following Documentation to help us with the audit - https://getsafle.notion.site/Unified-ID-System-Documentation-1e3a5bff0d5f8054985bcfd6f21e431b

2.1.3 Audit Goals

CredShields employs a combination of in-house tools and thorough manual review processes to deliver comprehensive smart contract security audits. The majority of the audit involves manual inspection of the contract's source code, guided by OWASP's Smart Contract Security Weakness Enumeration (SCWE) framework and an extended, self-developed checklist built from industry best practices. The team focuses on deeply understanding the contract's core logic, designing targeted test cases, and assessing business logic for potential vulnerabilities across OWASP's identified weakness classes.

CredShields aligns its auditing methodology with the OWASP Smart Contract Security projects, including the Smart Contract Security Verification Standard (SCSVS), the Smart Contract Weakness Enumeration (SCWE), and the Smart Contract Secure Testing Guide (SCSTG). These frameworks, actively contributed to and co-developed by the CredShields team, aim to bring consistency, clarity, and depth to smart contract security assessments. By adhering to these OWASP standards, we ensure that each audit is performed against a transparent, community-driven, and technically robust baseline. This approach enables us to deliver structured, high-quality audits that address both common and complex smart contract vulnerabilities systematically.

2.2 Retesting Phase

Safle Network Pvt. Ltd. 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, and 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	None	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 about the engagement or this document.

3. Findings Summary ----------

This chapter contains the results of the security assessment. Findings are sorted by their severity and grouped by asset and OWASP SCWE classification. Each asset section includes a summary highlighting the key risks and observations. The table in the executive summary presents the total number of identified security vulnerabilities per asset, categorized by risk severity based on the OWASP Smart Contract Security Weakness Enumeration framework.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, 19 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SCWE | Vulnerability Type
Ether can be permanently stuck in a contract	Critical	Fund Stuck
Incorrect Signature Verification	Critical	Insecure Signature Verification (SCWE-019)
Inconsistent price feed scaling can overcharge users	High	Incorrect decimal handling (SCWE-002)
Relayer Gas Waste on Nonexistent Secondary Address	Medium	Gas Griefing (SC04-Lack Of Input Validation)
Chainlink Oracle Min/Max price validation	Medium	Input Validation (SC04-Lack Of Input Validation)
Missing Price Feed Validation	Medium	Input Validation (SC04-Lack Of Input Validation)
Missing _disableInitialzers() call in constructor	Low	Insecure Upgradeable Proxy Design (SCWE-005)

Missing zero address validations	Low	Missing Input Validation (SC04-Lack Of Input Validation)
Missing events in important functions	Low	Missing Best Practices (SCWE-063)
Use Ownable2Step	Low	Missing Best Practices
Floating and Outdated Pragma	Low	Floating Pragma (SCWE-060)
Dead Code	Low	Code With No Effects (SCWE-062)
Cheaper conditional operators	Gas	Gas Optimization
Splitting Require Statements	Gas	Gas Optimization
Unused Imports	Gas	Gas Optimization
Cheaper Inequalities in require()	Gas	Gas & Missing Best Practices
Cheaper Inequalities in if()	Gas	Gas & Missing Best Practices
Gas Optimization in Increments	Gas	Gas optimization (SCWE-082)
Unnecessary Bytes-to-String Conversion in registrarChecks Modifier	Gas	Gas Optimization

Table: Findings in Smart Contracts

4. Remediation Status ----------

Safle Network Pvt. Ltd. is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on June 30th, 2025, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Ether can be permanently stuck in a contract	Critical	**Fixed** [June 30, 2025]
Incorrect Signature Verification	Critical	**Fixed** [June 30, 2025]
Inconsistent price feed scaling can overcharge users	High	**Fixed** [June 30, 2025]
Relayer Gas Waste on Nonexistent Secondary Address	Medium	**Fixed** [June 30, 2025]
Chainlink Oracle Min/Max price validation	Medium	**Acknowledged** [June 30, 2025]
Missing Price Feed Validation	Medium	**Acknowledged** [June 30, 2025]
Missing _disableInitialzers() call in constructor	Low	**Fixed** [June 30, 2025]
Missing zero address validations	Low	**Fixed** [June 30, 2025]
Missing events in important functions	Low	**Fixed** [June 30, 2025]
Use Ownable2Step	Low	**Fixed** [June 30, 2025]
Floating and Outdated Pragma	Low	**Fixed** [June 30, 2025]

Dead Code	Low	**Fixed** [June 30, 2025]
Cheaper conditional operators	Gas	**Fixed** [June 30, 2025]
Splitting Require Statements	Gas	**Fixed** [June 30, 2025]
Unused Imports	Gas	**Fixed** [June 30, 2025]
Cheaper Inequalities in require()	Gas	**Fixed** [June 30, 2025]
Cheaper Inequalities in if()	Gas	**Fixed** [June 30, 2025]
Gas Optimization in Increments	Gas	**Fixed** [June 30, 2025]
Unnecessary Bytes-to-String Conversion in registrarChecks Modifier	Gas	**Fixed** [June 30, 2025]

Table: Summary of findings and status of remediation

5. Bug Reports ----------

Bug ID #1 [Fixed]

Ether can be permanently stuck in a contract

Vulnerability Type
Fund stuck

Severity
Critical

Description
The RegistrarStorageChildEvents contract includes several functions such as registerRegistrar, initiateRegisterUnifiedId, initiateUpdateUnifiedId, initiateUpdateUnifiedIdPrimaryAddress, initiateAddSecondaryAddress, and initiateRemoveSecondaryAddress that are marked payable, allowing users to send ETH when calling them. However, the contract does not implement any function to withdraw or manage the received ETH. As a result, if a user accidentally or intentionally sends ETH to these functions, the funds will be permanently locked in the contract with no way to recover them.

Affected Code

- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.sol#L131-L148
- https://github.https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.sol#L131-L148com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.sol#L178-L192
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.sol#L210-L227
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.sol#L262-L277
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.sol#L293-L310
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.sol#L323-L331

Impacts

Any ETH sent to these payable functions will become irretrievable, leading to potential financial loss for users. This can also harm user trust and damage the credibility of the project if users perceive it as insecure or negligent.

Remediation

Add a secure withdrawal function restricted to an authorized address (such as the contract owner or an admin) that allows recovery of stuck ETH from the contract.

Retest

- This bug has been partially fixed as withdrawEth() has been implemented properly but withdrawERC20() has been implemented wrongly as there are some tokens which do not return bool during transfers so it is better to use safeTransfer() during transfer of tokens as it handles all the edge cases.
- This issue has been fixed now as per the recommendation.

Bug ID #2 [Fixed]

Incorrect Signature Verification

Vulnerability Type
Insecure Signature Verification (SCWE-019)

Severity
Critical

Description
The verifySignatureSecure() function in the RegistrarStorageUtil contract contains a critical cryptographic implementation flaw that renders signature verification completely ineffective. The vulnerability stems from incorrect handling of EIP-712 signature verification where the function applies double hashing to the digest before signature recovery.

The flawed implementation occurs in the signature recovery process where the function first constructs a proper EIP-712 digest using the standard \x19\x01 prefix and domain separator, but then incorrectly applies additional hashing operations before calling ecrecover. Specifically, the function calls recoverSigner(getEthSignedMessageHash(keccak256(abi.encode(digest))), signature) instead of directly using the EIP-712 digest for signature recovery.

The getEthSignedMessageHash() function adds the Ethereum signed message prefix \x19Ethereum Signed Message:\n32 to the already properly formatted EIP-712 digest, creating a double-prefixed hash that does not match what the signer originally signed. Additionally, the unnecessary keccak256(abi.encode(digest)) operation further corrupts the hash by encoding and re-hashing the digest bytes.

This implementation error means that any signature created using proper EIP-712 signing methods will fail verification, while the function may incorrectly accept malformed signatures that happen to match the corrupted hash structure. The vulnerability affects all operations that rely on the verifySignatureSecure() function for authentication and authorization.

Affected Code

- https://github.com/getsafle/unified-id-contracts/blob/34a21ef35c8e026d3a05ebc52de378db51101e12/contracts/RegistrarStorageUtil.sol#L434

Impacts

Legitimate users following proper EIP-712 signing procedures will find their valid signatures consistently rejected, leading to complete denial of service for any functionality requiring signature verification. This effectively breaks the intended access control mechanisms and prevents authorized users from performing critical operations.

Remediation

The vulnerability requires immediate remediation by eliminating the double hashing issue where two different prefixes are being applied to the same digest. The current implementation incorrectly applies both the EIP-712 prefix (`\x19\x01`) and the Ethereum signed message prefix (`\x19Ethereum Signed Message:\n32`) to the same hash, creating a corrupted digest that cannot be verified properly.

Retest

This issue has been fixed.

Bug ID #3 [Fixed]

Inconsistent price feed scaling can overcharge users

Vulnerability Type
Incorrect decimal handling (SCWE-002)

Severity
High

Description
The RegistrarStorageUtil smart contract contains a critical flaw in the getRequiredTokenAmount() function that leads to overcharging users when converting ETH-based fees into token amounts. The core issue arises due to inconsistent handling and normalization of decimals between registrarFees (18 decimals) and Chainlink price feed values, which are typically 8-decimal USD prices. When both ETH and token price feeds return the same number of decimals (most commonly 8), the calculation path enters the else block that does not apply any decimal adjustment.

To illustrate, consider a case where the registrar fee is 0.01 ETH, represented in wei as 1e16. Let's assume ETH is priced at $3,000 and the user chooses to pay in USDC, a 6-decimal stablecoin priced at $1.00. The Chainlink price feeds return:

- ethPriceInUSD = 3000 * 1e8 = 300000000000
- tokenPriceInUSD = 1 * 1e8 = 100000000
- tokenDecimal[USDC] = 6

Given both price feeds use 8 decimals, the function uses the direct formula:
result[4] = (1e16 * 300000000000) / 100000000 = 3e27 / 1e8 = 3e19

The returned amount is then normalized based on the USDC decimal precision:
return result[4] / 1e6 = 3e19 / 1e6 = 3e13

This outputs 30,000,000,000,000 USDC units — or 30 trillion base units, equivalent to 30 million USDC. However, a manual sanity check reveals that 0.01 ETH at $3,000 should only equate to $30, which corresponds to 30 * 1e6 = 30,000,000 USDC units.

Affected Code

- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageUtil.sol#L40-L66

Impacts

This miscalculation leads to users being severely overcharged when paying fees in tokens, especially stablecoins like USDC. The discrepancy can result in users losing large amounts of tokens unnecessarily, even for relatively small ETH-denominated fees.

Remediation

To address this vulnerability, decimal handling should be correctly implemented throughout the price conversion and token amount calculation process. This includes ensuring consistent scaling between registrarFees, token price feeds, and token base units, taking into account the decimals of both the price oracles and the target token.

Retest

This issue has been fixed by handling decimals during conversion.

Bug ID #4 [Fixed]

Relayer Gas Waste on Nonexistent Secondary Address

Vulnerability Type
Gas Griefing (SC04-Lack Of Input Validation)

Severity
Medium

Description
In the initiateRemoveSecondaryAddress function, there is no validation to check whether the _secondaryAddress exists for the specified _unifiedId. This contrasts with initiateAddSecondaryAddress, which verifies that a secondary address is not already associated before proceeding. As a result, a registrar can initiate removal for an address that was never added. The event emitted then triggers a relayer operation on a different chain, which consumes gas without effect because the address does not exist in the target's secondary list.

Affected Code

- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.sol#L323-L331

Impacts
An invalid removal request causes relayers to perform unnecessary operations, consuming gas and increasing system inefficiency without achieving any state change.

Remediation
Add a validation in initiateRemoveSecondaryAddress to ensure _secondaryAddress is currently associated with _unifiedId before emitting the event.

```
function initiateRemoveSecondaryAddress(
      string calldata _unifiedId,
      address _secondaryAddress,
      bytes calldata _signature,
      bytes calldata _options
    ) external payable WhenNotPaused unifiedIdExists(_unifiedId) onlyRegistrar returns (bool) {
+     UserData storage userData = userAddresses[_unifiedId];
+     require(!userData.isSecondary[_secondaryAddress], "Secondary address does not exists");
      emit RemoveSecondaryAddressInitiated(_unifiedId, _secondaryAddress, _signature,
```

```
_options);
    return true;
}
```

Retest

This issue has been fixed by adding a validation if _secondaryAddress exists or not.

Bug ID #5 [Acknowledged]

Chainlink Oracle Min/Max price validation

Vulnerability Type
Input Validation (SC04-Lack Of Input Validation)

Severity
Medium

Description
Chainlink has a library AggregatorV3Interface with a function called latestRoundData(). This function returns the price feed among other details for the latest round.
Chainlink aggregators have a built-in circuit breaker if the price of an asset goes outside of a predetermined price band. The result is that if an asset experiences a huge drop in value, the price of the oracle will continue to return the minPrice instead of the actual price of the asset.

Affected Code

- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageUtil.sol#L70-L76
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageUtil.sol#L81-L87

Impacts
This would allow users to store their allocations with the asset but at the wrong price.

Remediation
The contract should check the returned answer/price against the minPrice/maxPrice and revert if the answer is outside of the bounds.

```
if (price >= maxPrice or price <= minPrice) revert(); // eg
```

Retest
The price mechanism is yet to be decided. The smart contracts are upgradeable and will be updated later.

Bug ID #6 [Acknowledged]

Missing Price Feed Validation

Vulnerability Type
Input Validation (SC04-Lack Of Input Validation)

Severity
Medium

Description
Chainlink has a library AggregatorV3Interface with a function called latestRoundData(). This function returns the price feed among other details for the latest round.
The contract was found to be using latestRoundData() without proper input validations on the returned parameters which might result in a stale and outdated price.

Affected Code

- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageUtil.sol#L70-L76
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageUtil.sol#L81-L87

Impacts
Having oracles with functions to fetch price feed without any validation might introduce erroneous or invalid price values that could result in an invalid price calculation further in the contract.

Remediation

It is recommended to have input validations for all the parameters obtained from the Chainlink price feed. Here's a sample implementation:

```
(uint80 roundID ,int256 answer, uint256 timestamp, uint80 answeredInRound) = AggregatorV3Interface(chainLinkAggregatorMap[underlying]).latestRoundData();

require(answer > 0, "Chainlink price <= 0");
require(answeredInRound >= roundID, "Stale price");
require(timestamp != 0, "Round not complete");
```

Retest

The price mechanism is yet to be decided. The smart contracts are upgradeable and will be updated later.

Bug ID #7 [Fixed]

Missing _disableInitialzers() call in constructor

Vulnerability Type
Insecure Upgradeable Proxy Design (SCWE-005)

Severity
Low

Description
When using UUPSUpgradeable, the best practice is to call _disableInitizliers() in the constructor since the initializer function in the implementation contract can be called by anyone. The proxy's storage is separate from the implementation contract's storage. Directly initializing the implementation means you're not affecting the state that will be used when interacting via the proxy. This can lead to confusion or unexpected behavior if someone accidentally initializes the implementation contract.

Affected Code
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.sol#L10
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/MotherContract.sol#L9

Impacts
An uninitialized implementation contract can be taken over by an attacker.

Remediation
It is recommended to call _disableInitializers() in the constructor.

Eg:

```
constructor(){
  _disableInitializers();
}
```

Retest
This issue has been resolved by applying the recommended remediation. Fixes: fix1 & fix2

Bug ID #8 [Fixed]

Missing zero address validations

Vulnerability Type
Missing Input Validation (SC04-Lack Of Input Validation)

Severity
Low

Description:
The contracts were found to be setting new addresses without proper validations for zero addresses.
Address type parameters should include a zero-address check otherwise contract functionality may become inaccessible or tokens burned forever.
Depending on the logic of the contract, this could prove fatal and the users or the contracts could lose their funds, or the ownership of the contract could be lost forever.

Affected Code

- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageUtil.sol#L27-L31
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageUtil.sol#L34-L37
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.sol#L124
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/MotherContract.sol#L71

Impacts
If address type parameters do not include a zero-address check, contract functionality may become unavailable or tokens may be burned permanently.

Remediation
Add a zero address validation to all the functions where addresses are being set.

Retest
This bug has been fixed by adding zero address validations.

Bug ID #9 [Fixed]

Missing events in important functions

Vulnerability Type
Missing Best Practices (SCWE-063)

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/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.sol#L127-L129

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 important functions to keep track of them.

Retest
This bug has been resolved as the affected function has been removed from the code.

Bug ID #10 [Fixed]

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/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/MotherContract.sol
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.sol

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:
This bug has been fixed by implementing Ownable2Step.

Bug ID #11 [Fixed]

Floating and Outdated Pragma

Vulnerability Type
Floating Pragma (SCWE-060)

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., ^0.8.20,^0.8.22. 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/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageUtil.sol#L2
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/MotherContract.sol#L2
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.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 low.

Remediation
Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.29 pragma version
Reference: https://swcregistry.io/docs/SWC-103

Retest

This bug is resolved as adding strict compiler version across all the contracts.

Bug ID #12 [Fixed]

Dead Code

Vulnerability Type
Code With No Effects (SCWE-062)

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/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/MotherContract.sol#L28
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.sol#L24

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 library functions are not supposed to be used anywhere, consider removing them from the contract.

Retest
This bug has been fixed.

Bug ID #13 [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

- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageUtil.sol#L77
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageUtil.sol#L88

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 issue has been resolved by applying the recommended remediation — replacing x > 0 with x != 0 for the unsigned integer condition.

Bug ID #14 [Fixed]

Splitting Require Statements

Vulnerability Type
Gas Optimization

Severity
Gas

Description
Require statements when combined using operators in a single statement usually lead to a larger deployment gas cost but with each runtime calls, the whole thing ends up being cheaper by some gas units.

Affected Code

- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageUtil.sol#L213

Impacts
The multiple conditions in one **require** statement combine require statements in a single line, increasing deployment costs and hindering code readability.

Remediation
It is recommended to separate the **require** statements with one statement/validation per line.

Retest
This bug has been resolved as the affected function has been removed from the code.

Bug ID #15 [Fixed]

Unused Imports

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The contract RegistrarStorageChildEvents.sol was importing contracts IERC20.sol & IERC20Metadata.sol which was not used anywhere in the code. This increases the gas cost and overall contract's complexity.

Affected Code

- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.sol#L7
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.sol#L8

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:
This bug has been fixed by removing unused imports.

Bug ID #16 [Fixed]

Cheaper Inequalities in require()

Vulnerability Type
Gas & Missing Best Practices

Severity
Gas

Description
The contract was found to be performing comparisons using inequalities inside the require statement. When inside the require statements, non-strict inequalities (>=, <=) are usually costlier than strict equalities (>, <).

Affected Code

- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageUtil.sol#L213
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.sol#L155

Impacts
Using non-strict inequalities inside "require" statements costs more gas.

Remediation
It is recommended to go through the code logic, and, **if possible**, modify the non-strict inequalities with the strict ones to save gas as long as the logic of the code is not affected.

Retest:
This bug has been resolved as the affected function has been removed from the code.

Bug ID #17 [Fixed]

Cheaper Inequalities in if()

Vulnerability Type
Gas & Missing Best Practices

Severity
Gas

Description
The contract was found to be doing comparisons using inequalities inside the "if" statement. When inside the "if" statements, non-strict inequalities (>=, <=) are usually cheaper than the strict equalities (>, <).

Affected Code

- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageUtil.sol#L77
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageUtil.sol#L88

Impacts
Using strict inequalities inside "if" statements costs more gas.

Remediation
It is recommended to go through the code logic, and, **if possible**, modify the strict inequalities with the non-strict ones to save gas as long as the logic of the code is not affected.

Retest:
This bug has been fixed by using non-strict inequalities.

Bug ID #18 [Fixed]

Gas Optimization in Increments

Vulnerability Type
Gas optimization (SCWE-082)

Severity
Gas

Description
The contract uses two for loops**,** which 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/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageUtil.sol#L162-L162
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageUtil.sol#L192-L192
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/MotherContract.sol#L169-L169
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/MotherContract.sol#L198-L198
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/MotherContract.sol#L204-L204
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.sol#L242-L242
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.sol#L248-L248
- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.sol#L339-L339
-

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 meanwhile saves some gas.

Retest
This bug has been fixed by using ++i instead of i++.

Bug ID #19 [Fixed]

Unnecessary Bytes-to-String Conversion in `registrarChecks` Modifier

Vulnerability Type
Gas Optimization / Inefficient Code Pattern

Severity
Gas

Description

The `registrarChecks` modifier contains an unnecessary conversion from `string` to `bytes` and back to `string`, which introduces redundant operations and slightly increases gas usage. Solidity's `string` is internally stored as dynamic `bytes`, so directly using the string variable is sufficient for the comparison.

This conversion `string(regNameBytes)` is functionally equivalent to the original `_registrarName` string, making it redundant.

Affected Code

- https://github.com/getsafle/unified-id-contracts/blob/d27d2faf85570be4300db97f1c85527a820fbf7b/contracts/RegistrarStorageChildEvents.sol#L97

Impacts
Slightly increased gas usage per invocation
Reduced code readability due to unnecessary casting

Remediation
Remove the conversion and use the `_registrarName` string directly in the `require` condition

Retest
This bug has been fixed by removing the unnecessary conversion from bytes to string.

6. The 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.

YOUR SECURE FUTURE STARTS HERE

CRED SHIELDS

At CredShields, we're more than just auditors. We're your strategic partner in ensuring a secure Web3 future. Our commitment to your success extends beyond the report, offering ongoing support and guidance to protect your digital assets

Audited by CRED SHIELDS

20A TANJONG PAGAR ROAD, SINGAPORE (088443)

info@credshields.com