CredShields

Smart Contract Audit

30 January 2026 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Blockwill between 21 January 2026, and 23 January 2026. A retest was performed on 28 January 2026.

Author

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

Reviewers

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

Prepared for

Blockwill

Table of Contents

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

Blockwill engaged CredShields to perform a smart contract audit from 21 January 2026, to 23 January 2026. During this timeframe, 15 vulnerabilities were identified. **A retest was performed on 28 January 2026, and all the bugs have been addressed.**

During the audit, 1 vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Blockwill" 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
Smart Contracts	0	1	4	5	2	3	**15**

Table: Vulnerabilities Per Asset in Scope

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

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

Blockwill engaged CredShields to perform a 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 21 January 2026, to 23 January 2026, 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
Audited Scope: UserEventRegistry and BlocWillFactory **Retested Scope: UserEventRegistry and BlocWillFactory**

2.1.2 Documentation

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

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

Blockwill 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, 15 security vulnerabilities were identified in the asset.

Vulnerability Title	Severity	Vulnerability Type	Status
Platform Can Create Dead Man Switch Events Arbitrarily	High	Business Logic (SC03-LogicErrors)	Fixed
Platform Can Record Missing Event Without Mandatory Waiting Period	Medium	Business Logic (SC03-LogicErrors)	Fixed
Admin Can Change Platform Without Updating User Contracts	Medium	Configuration Inconsistency	Fixed
Platform Can Create Multiple Dead Man Switch Events	Medium	Business Logic (SC03-LogicErrors)	Fixed
Platform Can Record Conflicting Life Status Events	Medium	Business Logic (SC03-LogicErrors)	Fixed
Platform Can Set Invalid Dead Man Switch Date	Low	Lack of Input Validation (SC04-Lack Of Input Validation)	Fixed
Platform Can Deploy Contract With Invalid Dead Man Switch Date	Low	Lack of Input Validation (SC04-Lack Of Input Validation)	Fixed

Platform Can Record Event With Empty Executor Identifier	Low	Lack of Input Validation (SC04-Lack Of Input Validation)	Fixed
Missing events in important functions	Low	Missing Event Emission (SCWE-063)	Fixed
Floating and Outdated Pragma	Low	Floating Pragma (SCWE-060)	Fixed
Caller Can Waste Gas By Rewriting Unchanged Configuration	Informational	Missing functionality (SCWE-006)	Fixed
Dead Code	Informational	Code With No Effects (SCWE-062)	Fixed
Gas Optimization in Increments	Gas	Gas optimization (SCWE-082)	Fixed
Cheaper conditional operators	Gas	Gas Optimization (SCWE-082)	Fixed
Custom error to save gas	Gas	Gas Optimization (SCWE-082)	Fixed

Table: Findings and Remediations

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

Bug ID #H001 [Fixed]

Platform Can Create Dead Man Switch Events Arbitrarily

Vulnerability Type
Business Logic (SC03-LogicErrors)

Severity
High

Description
The UserEventRegistry.recordEvent function allows the platform to record inheritance-related events by specifying an EventType value, and it is intended to be used for standard user-triggered events that follow normal dispute and confirmation flows.
However, the function does not restrict the DEAD_MAN_SWITCH event type, allowing the platform to create DEAD_MAN_SWITCH events directly via recordEvent instead of exclusively through UserEventRegistry.triggerDeadManSwitch. The root cause is the absence of an explicit check preventing EventType.DEAD_MAN_SWITCH from being passed to recordEvent, bypassing the dedicated trigger logic and its enforced conditions.

Affected Code

- UserEventRegistry#L204

Impacts
Dead Man Switch events can be created without satisfying the required trigger conditions, undermining the intended execution flow and weakening protocol guarantees.

Remediation
Explicitly disallow recording DEAD_MAN_SWITCH events through recordEvent function.

Retest
This issue has been fixed by adding a require check to restrict creating DMS events through the recordEvent function.

Bug ID #M001 [Fixed]

Platform Can Record Missing Event Without Mandatory Waiting Period

Vulnerability Type
Business Logic (SC03-LogicErrors)

Severity
Medium

Description
The system records inheritance trigger events through UserEventRegistry.recordEvent, including events of type MISSING. A missing-person scenario is expected to follow a strict two-phase process where an initial report starts an on-chain waiting period, and only after that period elapses can the missing status be confirmed.
However, the current design allows recordEvent to create a MISSING event immediately without any prior on-chain timestamp or enforced waiting period. There is no intermediate state such as a recorded "missing since" timestamp, nor any validation that a required multi-year delay has elapsed before the event is created. The root cause is the absence of a dedicated two-phase missing-person flow and the lack of temporal enforcement before recording MISSING events.

Affected Code

- UserEventRegistry

Impacts
Missing-person events can be recorded instantly without satisfying the intended long-term waiting requirement, bypassing expected safeguards and undermining the correctness of the missing-person process.

Remediation
Introduce a two-phase missing-person flow by recording a missing start timestamp on-chain and disallow recording MISSING events until the required waiting period has fully elapsed.

Retest
This issue has been fixed by adding a two-phase flow implemented with reportMissingPerson() enforcing mandatory waiting period and resets missingPersonReportedAt = 0 after recording, preventing multiple MISSING events per report cycle.

Bug ID #M002 [Fixed]

Admin Can Change Platform Without Updating User Contracts

Vulnerability Type
Configuration Inconsistency

Severity
Medium

Description
The BlockWillFactory.updatePlatform function allows the platform address stored in the factory to be updated and is intended to control which address is authorized to perform privileged actions across the system. Newly deployed user registry contracts rely on the factory-provided platform address during initialization.
However, updating the platform address in BlockWillFactory.updatePlatform does not propagate the new address to already deployed UserEventRegistry contracts, which each store their own platform value set at initialization. The root cause is that the factory maintains no mechanism to update the platform address across existing user registry instances.

Affected Code

- BlocWillFactory#L228

Impacts
Previously deployed user registry contracts remain bound to the old platform address, leading to inconsistent authorization behavior and requiring the old platform to remain active or manual remediation on each deployed contract.

Remediation
Add a mechanism to update the platform address in all deployed user registry contracts when the factory platform is changed.

Retest
This issue has been fixed by adding batched propagation to update the platform address on all the user deployed addresses.

Bug ID #M003 [Fixed]

Platform Can Create Multiple Dead Man Switch Events

Vulnerability Type
Business Logic (SC03-LogicErrors)

Severity
Medium

Description
The UserEventRegistry.triggerDeadManSwitch function allows the platform to create a Dead Man Switch event once the configured trigger date has been reached. This function is intended to represent a one-time transition when the Dead Man Switch condition is satisfied.
However, the function does not enforce that it can only be executed once. After the trigger conditions are met, triggerDeadManSwitch can be invoked repeatedly, each time creating a new DEAD_MAN_SWITCH event. The root cause is the absence of a state flag or guard that marks the Dead Man Switch as already triggered.

Affected Code

- UserEventRegistry#L371

Impacts
Multiple Dead Man Switch events can be created for the same user, leading to duplicated records and inconsistent event history.

Remediation
Add a state check to ensure the Dead Man Switch can only be triggered once.

Retest
This issue has been successfully resolved by introducing a deadManSwitchTriggered boolean state variable that prevents duplicate Dead Man Switch events.

Bug ID #M004 [Fixed]

Platform Can Record Conflicting Life Status Events

Vulnerability Type
Business Logic (SC03-LogicErrors)

Severity
Medium

Description
The UserEventRegistry.recordEvent function allows the platform to independently record lifecycle-related events such as DEATH and INCAPACITATION. These events represent mutually exclusive real-world states, and a terminal state such as death is expected to finalize the user's status.
However, the function does not enforce any mutual exclusivity between event types. As a result, it is possible to record an INCAPACITATION event even when a DEATH event is already confirmed, or to record additional non-terminal events after death. The root cause is the absence of a state check in recordEvent to detect whether a terminal DEATH event already exists and block incompatible future events.

Affected Code

- UserEventRegistry#204

Impacts
The contract can enter an invalid logical state where contradictory life status events coexist, leading to inconsistent event history and ambiguous downstream behavior.

Remediation
Prevent recording of any new events if a DEATH event has already been confirmed, while still allowing DEATH to be recorded even if non-terminal events exist.

Retest
This issue has been fixed by restricting creation of events once the DEATH event is confirmed.

Bug ID #L001 [Fixed]

Platform Can Set Invalid Dead Man Switch Date

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

Severity
Low

Description
The UserEventRegistry.updateDeadManSwitchDate function allows the platform to update the stored Dead Man_Switch_trigge_ date, which is later relied upon by other contract logic.
However, the function explicitly permits newDate to be set to zero, while UserEventRegistry.triggerDeadManSwitch enforces deadManSwitchDate > 0 before allowing execution. The root cause is that updateDeadManSwitchDate accepts a zero value that violates the non-zero invariant enforced by downstream logic.

Affected Code

- UserEventRegistry.sol#L339

Impacts
The Dead Man Switch can be configured to a value that prevents it from being triggered without further updates.

Remediation
Disallow zero as a valid Dead Man Switch date to align configuration with trigger requirements.

Retest
This issue has been fixed by allowing only updating DMS date in future.

Bug ID #L002 [Fixed]

Platform Can Deploy Contract With Invalid Dead Man Switch Date

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

Severity
Low

Description
The BlockWillFactory.deployUserContract function deploys a new user registry contract and forwards initialization parameters, including the Dead Man Switch trigger date, to UserEventRegistry.initialize. This parameter is expected to represent an enabled trigger time and therefore must be a valid future timestamp.
However, the function does not validate that initialDeadManSwitchDate is strictly greater than zero and greater than the current block timestamp before deployment. The root cause is a missing input validation check in BlockWillFactory.deployUserContract, allowing invalid values to be passed during initialization.

Affected Code

- BlockWillFactory#L69
- BlockWillFactory#L113

Impacts
Contracts can be deployed with an invalid Dead Man Switch date, resulting in an incorrect initial configuration that requires later administrative correction.

Remediation
Validate that the provided Dead Man Switch date is strictly greater than zero and greater than the current block timestamp before deploying the contract.

Retest
This issue has been fixed by adding required checks.

Bug ID #L003 [Fixed]

Platform Can Record Event With Empty Executor Identifier

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

Severity
Low

Description
The UserEventRegistry.recordEvent function accepts an executorId parameter and records it as part of the event data. This value is expected to be non-empty when provided.
However, the function does not validate that executorId is not empty before storing it. The root cause is a missing empty-value check in UserEventRegistry.recordEvent.

Affected Code

- UserEventRegistry.sol#L204

Impacts
Events can be recorded with an empty executor identifier, resulting in incomplete on-chain data

Remediation
Add a non-empty check for the executor identifier before recording the event.

Retest
This issue is fixed by adding require check for an empty executorId.

Bug ID #L004 [Fixed]

Missing events in important functions

Vulnerability Type
Missing Event Emission (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

- UserEventRegistry.sol#L360

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 issue has been fixed by emitting event.

Bug ID #L005 [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., >= 0.8.19. 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

- BlockWillFactory#L2
- UserEventRegistry.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://scs.owasp.org/SCWE/SCSVS-CODE/SCWE-060/

Retest
This issue has been fixed.

Bug ID #I001 [Fixed]

Caller Can Waste Gas By Rewriting Unchanged Configuration

Vulnerability Type
Missing functionality (SCWE-006)

Severity
Informational

Description
Administrative and configuration functions are designed to update stored parameters only when a change is required, allowing the platform to manage protocol settings efficiently. These functions are typically called by a trusted actor and are expected to mutate state only when the new value differs from the existing one.
However, multiple setter-style functions update storage and emit events without first checking whether the new value is equal to the current value. For example, UserEventRegistry.updateDeadManSwitchDate assigns deadManSwitchDate = newDate and emits an event even when newDate is identical to the existing value. The root cause is the absence of a guard condition such as require(oldValue != newValue) before performing the write and emitting events.

Affected Code

- UserEventRegistry.sol#L338
- UserEventRegistry.sol#L360
- BlockWillFactory#L228

Impacts
Repeated calls with unchanged values unnecessarily consume gas, increase operational costs for the platform, and generate redundant on-chain events that add noise for indexers, auditors, and off-chain monitoring systems.

Remediation
Add an explicit equality check in all configuration setters to revert when the new value matches the existing value, preventing redundant state writes and event emissions.

Retest

This issue has been.

Bug ID #I002 [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

- UserEventRegistry.sol#L146

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 issue has been fixed by using the modifier in an function.

Bug ID #G001 [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.

Affected Code

- UserEventRegistry.sol#L343
- UserEventRegistry.sol#L498
- UserEventRegistry.sol#L509
- BlockWillFactory#L127

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 issue has been fixed.

Bug ID #G002 [Fixed]

Cheaper conditional operators

Vulnerability Type
Gas Optimization (SCWE-082)

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

- UserEventRegistry.sol#L372
- UserEventRegistry.sol#L666

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 fixed.

Bug ID #G003 [Fixed]

Custom error to save gas

Vulnerability Type
Gas Optimization (SCWE-082)

Severity
Gas

Description
During code analysis, it was observed that the smart contract is using the revert() statements for error handling. However, since Solidity version 0.8.4, custom errors have been introduced, providing a better alternative to the traditional revert(). Custom errors allow developers to pass dynamic data along with the revert, making error handling more informative and efficient. Furthermore, using custom errors can result in lower gas costs compared to the revert() statements.

Affected Code

- UserEventRegistry.sol#L676

Impacts
Custom errors allow developers to provide more descriptive error messages with dynamic data. This provides better insights into the cause of the error, making it easier for users and developers to understand and address issues.

Remediation
It is recommended to replace all the instances of revert() statements with error() to save gas..

Retest
This issue has been fixed.

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

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

20A TANJONG PAGAR ROAD, SINGAPORE (088443)

info@credshields.com