



c@inspect
You build, we defend.

JX Labs

Smart Contract Audit

BΔLT

July, 2025



BALT Smart Contract Audit

Version: v250730

Prepared for: JxLabs

July 2025

Security Assessment

1. Executive Summary
2. Summary of Findings
 - 2.3 Solved issues & recommendations
3. Scope
4. Assessment
 - 4.1 Security assumptions
 - 4.2 Decentralization
 - 4.3 Testing
 - 4.4 Code quality
5. Detailed Findings
 - BALT-001 - No test coverage for contracts

BΔLT-002 - Missing lastCheckIn update in registerInheritance may lead to premature inheritance claims

BΔLT-003 - Fee calculation rounds to zero for small deposit amounts

BΔLT-004 - Reentrancy risk when registering an inheritance

BΔLT-005 - Commission wallet cannot be updated

BΔLT-006 - Missing zero address validation for heir parameter




BΔLT-007 - Unbounded arrays can lead to gas exhaustion

6. Disclaimer

1. Executive Summary

In **July, 2025**, **JXLabs** engaged **Coinspect** to perform a Smart Contract Audit of **BΔLT**. The objective of the engagement was to evaluate the security of the smart contracts.

The **BΔLT** protocol is a decentralized application on the **RSK** blockchain that allows users to create time-locked digital inheritance vaults, enabling the automated transfer of **RBTC** to a designated heir after a defined period of user inactivity.

 Solved	 Caution Advised	 Resolution Pending
High 1	High 0	High 0
Medium 1	Medium 0	Medium 0
Low 3	Low 0	Low 0
No Risk 2	No Risk 0	No Risk 0
Total 7	Total 0	Total 0

2. Summary of Findings

This section provides a concise overview of all the findings in the report grouped by remediation status and sorted by estimated total risk.

2.3 Solved issues & recommendations

These issues have been fully fixed or represent recommendations that could improve the long-term security posture of the project.

Id	Title	Risk
BALT-001	No test coverage for contracts	High
BALT-002	Missing lastCheckIn update in registerInheritance may lead to premature inheritance claims	Medium
BALT-003	Fee calculation rounds to zero for small deposit amounts	Low
BALT-004	Reentrancy risk when registering an inheritance	Low
BALT-005	Commission wallet cannot be updated	Low
BALT-006	Missing zero address validation for heir parameter	None
BALT-007	Unbounded arrays can lead to gas exhaustion	None

3. Scope

The scope was set to be the repository:

- https://github.com/JXLabsOK/BALT_SmartContracts at commit 39ae582e885734f900db02c2f924712804509a20.

4. Assessment

This report presents the security assessment of the BALT protocol, a set of Solidity smart contracts designed to facilitate the transfer of digital assets post-mortem or upon prolonged user inactivity. The system is intended for deployment on the RSK blockchain. It comprises two main smart contracts: `InheritanceVault` and `InheritanceFactory`.

The core component, `InheritanceVault`, acts as an individual time-locked escrow. A user, designated as the testator, can create a vault, deposit RBTC, and name an heir. The funds are released to the heir only after a pre-defined `inactivityPeriod` has elapsed since the testator's last interaction (`performCheckIn`). The testator retains the ability to cancel the arrangement and withdraw their funds at any time.

The `InheritanceFactory` contract serves as a discovery and deployment mechanism, allowing users to create their own `InheritanceVault` instances and providing a public registry of all created vaults. A protocol fee is collected upon the initial deposit and transferred to a designated `commissionWallet`.

The protocol enables a "dead man's switch" functionality for on-chain assets. The primary workflow is initiated when a user interacts with the `InheritanceFactory` to deploy a personal `InheritanceVault` contract, specifying an `inactivityPeriod`.

The contract does not enforce a strict time window during which the testator must check in. The testator may perform a check-in at any time, even long after the `inactivityPeriod` has passed, as long as the heir has not yet claimed the inheritance. Additionally, the testator retains the unilateral ability to cancel the inheritance and reclaim the escrowed funds at any point prior to the release, regardless of the elapsed inactivity period. This design places full control in the hands of the testator until the exact moment the inheritance is claimed.

The operational flow proceeds as follows:

1. **Vault Creation:** A testator calls `createInheritanceVault` on the factory, which deploys a new `InheritanceVault` contract. The testator's address and the chosen inactivity period are permanently stored in the new vault.
2. **Inheritance Registration:** The testator then calls the `registerInheritance` function on their newly created vault. This transaction must include the RBTC to be placed in escrow and specifies the heir's address. A 0.5% commission on the deposited amount is immediately transferred to the `commissionWallet`.
3. **Activity Proof:** To prevent the premature release of funds, the testator must periodically call `performCheckIn`. This action resets the inactivity timer.

4. Claim by Heir: If the `inactivityPeriod` passes without a check-in from the testator, the `claimInheritance` function becomes callable by the designated heir, who can then withdraw the entire balance of the contract.
5. Cancellation by Testator: At any point before the inheritance is claimed, the testator can execute `cancelInheritance` to nullify the agreement and reclaim all funds held within the vault.

The system relies on `block.timestamp` as the source of time for tracking inactivity.

4.1 Security assumptions

For this assessment, Coinspect made the following assumptions:

1. The addresses for the testator and the heir are correct and controlled by the intended individuals.
2. The integrity and availability of the underlying RSK blockchain are trusted.
3. The `block.timestamp` is considered a sufficiently reliable source of time for the defined `inactivityPeriod`. The protocol assumes that minor timestamp manipulation by RSK miners will not be significant enough to compromise the intended logic.
4. The entity controlling the `commissionWallet` is trusted to manage the collected fees appropriately.

4.2 Decentralization

The protocol's design involves specific roles with distinct, centralized privileges within the scope of each `InheritanceVault`.

- Testator: This role holds significant authority over an individual vault. The testator is the only party who can register the inheritance, perform check-ins, and unilaterally cancel the vault to reclaim funds. These privileges are secured by `msg.sender` checks.
- Heir: The heir is a passive recipient with a single privilege: the ability to claim the inheritance, conditional upon the testator's inactivity.
- Protocol Owner: There is an implicit protocol owner role embodied by the controller of the `commissionWallet`. This address is set at the time of the `InheritanceFactory` deployment and is immutable for all subsequently created vaults. This entity's sole privilege is the receipt of protocol fees.

- Ownership: The `InheritanceFactory` contract itself has no administrative owner with special privileges after deployment. Each `InheritanceVault` is effectively co-owned by the testator and the heir, with the testator holding dominant control until the inactivity condition is met.
- Multisig: The protocol does not implement any multisignature scheme. The `commissionWallet` could be a multisig address, which would decentralize control over protocol fees. Coinspect considers that for a production system, employing a multisig for the `commissionWallet` would be a prudent measure to mitigate single-point-of-failure risks associated with a single private key. Users should be aware that control of this wallet resides with an external entity.

4.3 Testing

At the time of review, no tests were provided for the smart contracts. The absence of a test suite limits confidence in the correctness of the system, especially under edge cases or adversarial conditions. It is strongly recommended to implement tests to validate the expected behavior of the system before advancing to the production phase.

4.4 Code quality

- Documentation: Functions, contracts and events lack comprehensive `natspec` documentation. Key functions like `registerInheritance` and `claimInheritance` would benefit from `@param`, `@return`, and `@dev` tags to explicitly describe their parameters, return values, and intended behavior.
- Readability: The code is straightforward and easy to follow. The use of an `enum` for `Status` (`Active`, `Released`, `Cancelled`) significantly improves readability and makes the state machine logic explicit. Variable names are clear and descriptive.
- Flows: The logical flows are well-defined. State transitions are managed through the `inheritanceStatus` variable and validated using `require` statements, which helps prevent improper function execution. For instance, an inheritance cannot be claimed if it has already been `Cancelled` or `Released`.

5. Detailed Findings

BALT-001

No test coverage for contracts

Status

Solved

Risk

High

Resolution


Fixed

Impact

High

Likelihood

High



Location

BALT_SmartContracts/contracts/InheritanceVault.sol
BALT_SmartContracts/contracts/InheritanceFactory.sol

Description

Neither the `InheritanceFactory` nor `InheritanceVault` contracts have any test coverage, leaving critical functionality untested and increasing the risk of undiscovered bugs that could lead to fund loss or unintended behavior.

Coinspect considers the absence of tests to pose a significant operational and security risk, particularly for protocols handling real user funds. While issues arising from missing tests may not occur immediately, the likelihood of

introducing critical bugs during future development is high, and the impact of such bugs could be severe.

Recommendation

Implement a comprehensive test suit that validates all critical paths, access control logic, failure scenarios and edge cases.

Status

Fixed on commit `18564f9fcee5fdef7c8fc02d517dabdc9a083b`.

Tests were added to the project.

However, Coinspect identified that the following test description is misleading:

```
it("should allow the testator to cancel before inactivity", async () =>
{
    await
    expect(vault.connect(testator).cancelInheritance()).to.not.be.reverted;
});
```

The test states "testator can cancel before inactivity" but the smart contract allows testator to cancel at any time as long as the heir has not claimed the inheritance.

BALT-002

Missing lastCheckIn update in registerInheritance may lead to premature inheritance claims

Status

Solved

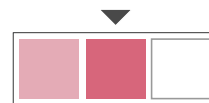


Resolution

Fixed

Risk

Medium



Impact

Medium

Likelihood

Medium

Location

BALT_SmartContracts/contracts/InheritanceVault.sol

Description

The `lastCheckIn` timestamp is not updated when the `registerInheritance` function is called, allowing the heir to prematurely claim the inheritance if the inactivity period has already elapsed since contract deployment.

The `lastCheckIn` timestamp is initialized in the `constructor` and later updated via `performCheckIn`, but it remains unchanged when `registerInheritance` is called.

If there is a significant time gap between vault creation and inheritance registration, the heir might be able to claim the inheritance immediately after registration if the inactivity period has already elapsed from the constructor timestamp.

Coinspect considers this issue to have a medium likelihood, especially for users who pre-deploy vaults and fund them later. The impact is medium, as it undermines the intended time-lock mechanism and may lead to unexpected fund claims.

Recommendation

Update the `lastCheckIn` timestamp when the `registerInheritance` function is called.

Status

Fixed on commit `d938c1e827961d7cafdec1b56df70fd8647a49`.

The `lastCheckIn` variable is now updated when registering an inheritance.

BALT-003

Fee calculation rounds to zero for small deposit amounts

Status

Solved



Resolution

Fixed

Risk

Low



Impact

Low

Likelihood

Low

Location

BALT_SmartContracts/contracts/InheritanceVault.sol

Description

The commission fee calculation in the `registerInheritance` function rounds down to zero when the deposit amount is less than 200, bypassing the fee payment to the commission wallet.

```
function registerInheritance(address _heir) public payable {  
    //...  
    uint fee = (msg.value * 5) / 1000; // 0.5% commission  
    uint netAmount = msg.value - fee;  
    //...  
}
```

Coinspect considers this to be a low likelihood and low impact issue. While the financial implications are minimal due to the extremely small amounts involved, it still represents a deviation from expected behavior.

Recommendation

Enforce a minimum fee or set a minimum deposit amount to ensure the commission logic is always executed as intended.

Status

Fixed on commit `d4adb5b2329a809ab4a4a3e933cb3a0eba018969`.

The JX Labs team added two checks to enforce minimum deposit amounts and non-zero fees.

Coinspect identified that the first require statement checking `fee > 0` is redundant. The subsequent check `netAmount >= MIN_DEPOSIT` already ensures that the deposit is sufficient to generate a non-zero fee, making the explicit fee check unnecessary.

BALT-004

Reentrancy risk when registering an inheritance



Location

BALT_SmartContracts/contracts/InheritanceVault.sol

Description

The `registerInheritance` function performs an external call to the commission wallet before updating storage variables, leading to a reentrancy risk.

The external call occurs before setting `heir` and `inheritanceAmount` storage values, violating the checks-effects-interactions pattern.

```
function registerInheritance(address _heir) public payable {  
    //...  
    (bool sent, ) = commissionWallet.call{value: fee}("");  
    require(sent, "Commission transfer failed");  
  
    heir = _heir;  
    inheritanceAmount = netAmount;  
  
    emit InheritanceRegistered(testator, heir, inheritanceAmount,
```



```
inactivityPeriod);  
}
```

In this contract the threat is minimal. Only the designated testator can invoke `registerInheritance`, and the `commissionWallet` is assumed to be a trusted address.

Coinspect considers both the likelihood and impact of this issue to be low. However, adhering to the checks-effects-interactions pattern remains a best practice for secure and maintainable code.

Recommendation

Follow the checks-effects-interactions pattern by updating all storage variables before making external calls. Move the commission wallet transfer to the end of the function after setting the `heir` and `inheritance` amount.

Status

Fixed on commit `22b06f02e4817178693c733273bf44eed3fbd838`.

The function now performs the external call after updating the relevant variables.

Commission wallet cannot be updated

Status

Solved



Resolution

Fixed

Risk

Low



Impact

Low

Likelihood

Low

Location

BALT_SmartContracts/contracts/InheritanceFactory.sol

Description

The `commissionWallet` address is set in the deployment of the `InheritanceFactory` contract and cannot be updated, creating operational inflexibility if the commission wallet needs to be changed.

Additionally, the constructor does not validate that the `_commissionWallet` parameter is a non-zero address. If a zero address is mistakenly provided, all commission payments will be irreversibly lost upon vault creation.

Once deployed, the factory contract will permanently route all commission payments to the initially configured address. If the commission wallet is compromised, the private keys are lost, or the organization needs to update the receiving address for operational reasons, there is no way to update it. This could lead to a permanent loss of protocol revenue or the need to redeploy the factory contract entirely.

Coinspect considers both the likelihood and impact of this issue to be low. It is unlikely that the commission wallet will require frequent changes, and no funds are directly at risk as long as the wallet remains controlled by the intended party.

Recommendation

Use a multisig wallet with enough signers and a secure threshold as the `commissionWallet` to enable controlled key management and minimize the risk of permanent fund loss. Include a zero address check in the constructor to prevent misconfiguration.

Status

Fixed on commit `dc241d09ab1fd0796351d503e11eb0a2813b023e`

The JXLabs team added a check to prevent the misconfiguration of the `commissionWallet`.

However, deploy script currently contains a hardcoded EOA address for the commission wallet. JXLabs has confirmed they will replace this hardcoded address with a multisig address before production deployment

BALT-006

Missing zero address validation for heir parameter

Status

Solved



Resolution

Fixed

Risk

None



Impact

Recommendation

Likelihood

—

Location

BALT_SmartContracts/contracts/InheritanceVault.sol

Description

The `registerInheritance` function does not validate that `_heir` is not the zero address, which could result in inheritance funds being permanently lost. Since inheritance can only be registered once and the `heir` cannot be changed, this would make the funds unrecoverable.

Recommendation

Add a check to validate that `_heir` is not the zero address.

Status

Fixed on commit `764492280e1e609b6c9c9206a1217539c17d3510`.

BALT-007

Unbounded arrays can lead to gas exhaustion

Status

Solved



Resolution

Acknowledged

Risk

None



Impact

Recommendation

Likelihood

—

Location

BALT_SmartContracts/contracts/InheritanceFactory.sol

Description

Both the `allVaults` array and the arrays within the `vaultsByTestator` mapping grow unbounded, making the `getAllVaults` and `getVaultsByTestator` functions consume more and more gas until they exceed block gas limits and become unusable.

Recommendation

Remove both the `allVaults` array and the arrays within `vaultsByTestator` mapping. Consider using a double mapping structure for `vaultsByTestator`. Use event indexing to retrieve all created vaults when needed, as the `VaultCreated` event provides the necessary information.

Status

Acknowledged.

The team acknowledged the gas exhaustion concern but considers the practical risk negligible given their expected usage patterns. They stated the arrays serve a useful tracking purpose and remain open to migrating to event-based tracking if future growth requires it.

6. Disclaimer

The contents of this report are provided "as is" without warranty of any kind. Coinspect is not responsible for any consequences of using the information contained herein.

This report represents a point-in-time and time-boxed evaluation conducted within a specific timeframe and scope agreed upon with the client. The assessment's findings and recommendations are based on the information, source code, and systems access provided by the client during the review period.

The assessment's findings should not be considered an exhaustive list of all potential security issues. This report does not cover out-of-scope components that may interact with the analyzed system, nor does it assess the operational security of the organization that developed and deployed the system.

This report does not imply ongoing security monitoring or guaranteeing the current security status of the assessed system. Due to the dynamic nature of information security threats, new vulnerabilities may emerge after the assessment period.

This report should not be considered an endorsement or disapproval of any project or team. It does not provide investment advice and should not be used to make investment decisions.