Security Research Labs

BalanceAI Baseline Security Assurance

Hacking assessment report

v0.1, November 08, 2024

Prepared for:
BalanceAI

- Confidential, 46 pages -

Content

Disclaimer

This report describes the findings and core conclusions derived from the audit carried out by Security Research Labs within the timeframe and scope detailed in Chapter 3.

Please note that this report does not guarantee that all existing security vulnerabilities were discovered in the codebase exhaustively and that following all evolution suggestions described in Chapter 5 may not ensure all future code to be bug free.

Version:	v0.1
The Client:	BalanceAI
Date:	November 08, 2024
The Auditor:	Cayo Fletcher-Smith cayo@srlabs.de Rachna Shriwas rachna@srlabs.de Marc Heuse marc@srlabs.de

Timeline

Security Research Labs performed the BalanceAI Substrate-EVM Bridge's code security assessment. The analysis took 2 weeks, starting from October 7, 2024. See Table 1.

Date	Event
October 7, 2024	Commencement of the code audit
October 22, 2024	Report for the baseline security check delivered
November 08, 2024	Final report delivered after issue mitigations

Table 1: Audit timeline

Confidentiality Notice

1 Executive summary

1.1 Engagement overview

Security Research Labs has been providing specialized audit services for Polkadot and Polkadot SDK projects since 2019. This report documents the results of a security assurance audit of BalanceAI's Substrate-EVM bridge implementation, that Security Research Labs performed. During this study, BalanceAI provided access to the code and effectively supported the research team.

This audit focused on assessing BalanceAI's Substrate-EVM bridge codebase for resilience against hacking and abuse scenarios. Key areas of scrutiny included inter-chain operations, such as teleport and withdrawal, between the Ethereum and Substrate chains via the bridge.

The testing approach combined static analysis and manual source code analysis, leveraging both automated tools and manual inspection. We prioritized reviewing critical functionalities and conducting thorough security tests to ensure the robustness of BalanceAI's platform.

1.2 Observations and Risk

Security Research Labs identified several issues ranging from high to informational level severity, many of which concerned the loss of user funds, costly gas operations and underweight extrinsics leading to resource exhaustion via transaction spamming. It was also identified that there is too much dependency on the governance to flag and resolve misbehavior in the network, instead of proactively restricting the misbehavior. For example, aborted teleport operations can only be recovered via governance. Furthermore, guardians are regarded as trusted entities that are expected to act honestly, but do not face slashing or penalties for misbehavior. Their role involves flagging any invalid operations within the network and having the authority to endorse a fraudulent teleport operation. Resolution of any misbehavior by the guardians can only occur through governance intervention.

There were also discrepancies identified between the implementation on the Substrate side and the EVM side, for example the data type for thresholds had different maximum limit, a nonce was used in the teleport operation in the EVM but not in Substrate, which leads to replayed teleport attack resulting in loss of user funds, as outline in S3-19. Moreover, there is no verification that the values of the storage parameters are consistent between the chains, which can result in teleport operations to succeed on one chain but fail in another.

Finally, mechanisms to safeguard token teleport authenticity were identified to be incomplete, which could result in privileged participants performing targeted exploitation on user liquidity, as outlined in issue S3-22.

BalanceAI acknowledged and mitigated some of these severe issues, in coordination with Security Research Labs, and plans further remediation for less-severe issues to occur in the later stages of development.

1.3 Recommendations

In addition to mitigating the remaining open issues, Security Research Labs recommends focusing on ensuring the protocol can remain robust without full reliance on governance intervention. We strongly recommend considering the business logic suggestions as mentioned in 2.1, in future releases such as transitioning to a more decentralized model and implementing appropriate incentivization for honest participation. Since, our recommendations include significant design changes, we encourage conducting extensive testing alongside an additional audit pre-launch once these changes are in place to ensure that no further vulnerabilities have been introduced.

2 Evolution suggestions

We acknowledge that BalanceAI has improved some security metrics and partially or fully implemented some security recommendations that were outlined in the previous audit for the evm-bridge-pallet. During this audit we identified a multitude of issues, many of which suggest immaturity in the system's architecture and implementation. We view this early-stage review as a crucial step in uncovering these issues, enabling the team to address them proactively.

In this section we recommend considering some key evolution suggestions and best practices derived from our findings and analysis during this audit.

2.1 Business logic improvement suggestions

Maintaining safe thresholds. As outlined in issue S2-6 threshold parameters are entirely arbitrary and not derived from sound security assumptions. We recommend directly deriving these thresholds from the current guardian network state. In this model regular manual intervention would not be required for bridge maintenance, and instead threshold updates would be triggered once per epoch.

These updates would re-establish thresholds as a percentage of active guardians in the network. This approach ensures that the system dynamically adjusts to the changes in the guardian participation and ensures consistency in the threshold values across chains.

Transition towards decentralization. Certain aspects of the system should be decentralized over a period. Aspects such as the ownership over the Ethereum smart contracts should be backed by governance, or at least some multi-signature implementation, to effectively distribute decision-making authority and reduce centralization risks, such as rogue owner. Furthermore, the guardian network could eventually be derestricted to a semi-permissioned or permissionless network, where increased security can be derived from the quantity of participants. This evolution would also reduce the likelihood and impact of exploitation of core off-chain critical infrastructure.

Guardians staking, rewards & slashing. Require guardians to stake tokens to assume their roles and implement slashing penalties for dishonest or malicious behavior. Currently, there are no rewards for guardians to vouch and disavow for teleport operations, but they still need to pay for submitting the associated operations. We recommend implementing rewards and slashing mechanisms that ensure guardians have financial incentives to act responsibly and deters potential misbehavior that could harm the network.

Cross-chain liquidity checks. Before executing teleport operation, specifically from the Ethereum module, we recommend ensuring there is enough liquidity locked in the Substrate module to satisfy the operation. Furthermore, we recommend integrating checks on the Ethereum side that ensure the minted amount can never exceed the total locked liquidity on the source chain.

Rate limiting. Cross-chain rate limiting should be implemented to bound the amount of value that may exit the bridge within a given period. Since the bridge is currently heavily reliant on governance intervention it is critical that governance should be given adequate time to respond to exploitation. Rate limiting coupled with sufficient off-chain monitoring would allow governance to reduce the impact of exploits targeting user liquidity.

Upgradeable logic. Consideration should be given to integrating upgradable logic within the Solidity smart contracts to allow smooth migrations to a new logic layer. This would allow for usability issues or security issues to be patched without requiring a full state storage migration. Consider *EIP-1822: Universal upgradable proxy standard* [1] or *openzeppelin-contracts-upgradeable* [2]

Set-up event timeout. Implement a timeout mechanism for stale events (such as a teleport operation not yet withdrawn, and aborted teleports) to ensure that user funds are not lost, and the integrity of the transfer is maintained.

2.2 Secure development improvement suggestions

We recommend to further strengthen the security of the blockchain system by implementing the following recommendations:

Event logging for error handling. Replace print statements in the codebase with proper logging and event emission to ensure that errors and unexpected behaviors are captured reliably. This approach enables better tracking, monitoring and resolution of issues.

Chain genesis verification. Verify the BalanceAI chain genesis hash when establishing connection to the chain to ensure authenticity and security. This prevents unauthorized or tampered chains from interacting with the network, safeguarding against potential attacks.

Adopt pair programming. Pair programming practices help identify security vulnerabilities early in the development cycle. This is beneficial for better code security because it involves immediate code review, allowing vulnerabilities to be caught in real time. Pair programming fosters continuous knowledge transfer reduces the risk of oversights and encourages adherence to coding standards. This collaborative approach also boosts problem solving, morale, and accountability, ensuring early detection of security issues and fostering continuous learning and improvement.

Implement Pull Requests with strict peer review requirements. The audit made evident that the code quality suffered considerably from a severe lack of peer review that likely contributed to the quantity of vulnerabilities. Using pull requests (PRs) instead of direct commits, establishes a review process that prevents bypassing essential feedback. Each PR should require 2-3 approvals before merging, ensuring code quality and security with input from multiple reviewers. This collaborative review structure helps identify issues early, encourages best practices, and leads to higher overall code security.

Perform threat modeling. Threat modeling for all new features and major updates before coding promotes better code security. This practice lets developers identify potential security threats and vulnerabilities early in the design phase, enabling them to implement appropriate mitigations from the outset. Including the threat model in the pull request description ensures that the entire team is aware of the identified risks and the measures taken to address them, promoting a proactive security culture and enhancing the overall robustness of the codebase. Additionally, it helps the audit team to identify gaps in the threat model and focus their assessment.

Use static analysis. Static analysis tools help detect security flaws in the codebase, thus improving code security. These tools, such as Dylint or Semgrep for Rust and Slither for Solidity, analyze code without executing it. They can identify vulnerabilities, coding errors, and compliance issues early in the development process. This proactive approach helps developers address potential security issues before they reach production, ensuring a more secure and reliable codebase.

Perform dynamic analysis. We recommend developing more tests for the bridge service and bridge pallet. Creating fuzzing harnesses for critical components is also essential for identifying security vulnerabilities and business logic issues. By employing invariants, these fuzzing tests can effectively uncover subtle flaws that might otherwise go unnoticed. The Polkadot codebase exemplifies this approach by utilizing multiple fuzzing harnesses based on the substrate-runtime-fuzzer. This demonstrates how comprehensive and targeted fuzz testing can significantly enhance the security and reliability of complex systems. For details on substrate based fuzz testing, see the substrate-runtime-fuzzer [3], for details on smart contract invariant based fuzz testing see Foundry [4] or Echidna [5].

Create an incident response plan. Developing a comprehensive incident response plan to address potential security breaches is vital for maintaining code security and organizational resilience. This plan should include detailed procedures for responding to various scenarios, such as compromised developers or exploited blockchain vulnerabilities, to ensure quick and effective mitigation of threats. By having a well-defined response strategy, organizations can minimize the impact of security incidents and maintain trust with users and stakeholders.

Launch a bounty program. Establishing a bounty program to encourage the external discovery and reporting of security vulnerabilities is crucial for enhancing code security. By offering incentives, such programs motivate individuals to responsibly report vulnerabilities to BalanceAI rather than exploit them. This approach not only broadens the pool of people actively searching for security flaws but also fosters a collaborative security environment, helping to identify and address issues more quickly and effectively.

2.3 Address currently open security issues

Even if an issue has a perceived limited impact, it may be integrated into a more complex, and unforeseen, exploitation chain. Furthermore, the remaining security issues associated with the EVM module smart contract implementation must be fixed pre-launch. These vulnerabilities may not be remediated post-launch consequential to the aforementioned lack of upgrade logic.

For this reason, we strongly recommend addressing all known security issues before launching on mainnet to prevent unnecessary risk or exploitation.

2.4 Further recommended best practices

Extensive test implementation. Currently, the solidity contracts have unit tests, but they are neither comprehensive nor exhaustive. For example, the thresholds can be set to any arbitrary value, the teleport operation fails to verify and update the nonce, and there exists various factors that affect the overall cost of the operation. Although the essential tests are already implemented, they do not cover all scenarios. We recommend extending unit tests to provide a stronger focus on security. This includes testing for edge cases, potential vulnerabilities, and common attack vectors. Additionally, we strongly encourage integrating end-to-end tests that focus on catching errors in the message passing layer in evm-bridge-service, before considering a mainnet launch.

Phased deployment for production readiness. Launching initially on a test network allows real-world simulation without risking the stability of the main network, providing an opportunity to identify and address potential issues in a controlled environment before a full release. After careful deliberation we believe the current state of code maturity would greatly benefit from further testing in a live environment.

Documentation improvement. Currently there exists incredibly limited technical documentation available for developers or users. We recommend dedicating time to the documentation of system architecture and design decisions. This will increase the observable correlation between technical decisions and the projects road map alongside ultimately fostering transparency between the development team and stakeholders. To achieve this, establish a practice of updating the documentation concurrently with any code changes. Incorporating documentation verification into the code review process can help detect discrepancies early.

Regular updates. New releases of cargo crates may contain fixes for critical security issues. Since BalanceAI is a product that heavily relies on these dependencies, updating to the latest stable version as soon as possible whenever a new release is available, is recommended.

3 Motivation and scope

On a technical level, the core business logic of BalanceAI aims to create a marketplace for AI (Artificial Intelligence) agents, including a bridge between Substrate and the Ethereum ecosystem.

BalanceAI is a blockchain network built on top of Polkadot SDK. Like other blockchain networks based on this SDK, the BalanceAI's code is written in Rust, a memory safe programming language. The bridge to Ethereum and Substrate is also written in Rust, along with Solidity contracts that are deployed and executed on the Ethereum blockchain.

Security Research Labs collaborated with the BalanceAI team to create an overview containing the runtime configuration and bridge implementation in scope and their audit priority. The in-scope components are reflected in Table 3.1.1.

Repository	Component(s)	Commit ID	Reference
balanceai-core	runtime configuration	970ad0a	[6]
evm-bridge-service	Bridge implementation	80e6502	[7]
evm-bridge-solidity	EVM bridge solidity contracts	347bc95	[8]

Table 3.1.1: In-scope BalanceAI components

4 Methodology

We carried out a hybrid strategy utilizing a combination of static analysis and manual code review, to assess the security of the BalanceAI codebase. While static analysis establishes a baseline assurance, the manual code review aimed to identify logic bugs, design flaws, and best practice deviations. The approach of the review was to trace the intended functionality of the modules in scope and to assess whether an attacker can bypass/misuse/abuse these components or trigger unexpected behavior on the chain due to logic bugs or missing checks.

The final step is supporting BalanceAI's remediation process to fix or remediate the identified issues. Each finding was documented and published with mitigation recommendations. Once the mitigation solution is implemented, the auditors verify the fix to ensure that it mitigates the issue and does not introduce other bugs.

During the audit, findings were shared via the GitHub repositories [6] [7] [8]. We also used private channel for asynchronous communication and status updates.

5 Findings summary

We identified 22 issues during our analysis of the BalanceAI codebase. In summary, we found 2 high-severity, 5 medium-severity, 6 low-severity, 3 information-level and 6 gas-level issues were found.

As BalanceAI is not deployed yet, the severities of the issues were also lowered. For example, an issue could have a critical severity if it is immediately exploitable in a deployed network, but its severity can be lowered to high in case of early development stages.

For Ethereum smart-contracts we report logic-efficiency optimization to reduce transaction gas fees outside of our traditional severity model. Gas optimizations can be considered non-critical recommendations that should, in all cases, be actionable.

Find an overview of all findings in Table 5.1.1.

Critical	0
High	2
Medium	5
Low	6
Informational	3
Gas	6
Total Issues	**22**

5.1 Issue summary

ID	Issue	Severity	Status
S3-19 [9]	Replayed teleport operation can disrupt internal accounting	High	Mitigated [10] [11] [12] [13] [14]
S3-22 [15]	Missing `disavow` logic enables malicious guardians to drain liquidity	High	Mitigated [16] [17] [18] [19] [20]
S2-8 [21]	Aborted teleport permanently locks user funds	Medium	Open
S2-1 [22]	Missing and incorrect runtime weights for balanceai-core-pallets	Medium	Mitigated [23]
S2-5 [24]	Nothing at stake could enable dishonest guardians	Medium	Open
S2-6 [25]	Thresholds are not derived from meaningful safety assumptions	Medium	Open
S2-10 [26]	Unchecked `transfer()` could result in unexpected behavior	Medium	Open
S1-7 [27]	Missing access control for `enableBridgeBack()`	Low	Mitigated [28]

S1-9 [29]	Single-step update could render `admin` and `owner` irrecoverable	Low	Mitigated [30]
S1-11 [31]	Unsafe use of `transfer()`	Low	Open
S1-15 [32]	Unsafe use of `abi.encodeWithSelector()`	Low	Mitigated [33]
S1-16 [34]	Missing `address(0)` checks for `admin` update	Low	Mitigated [35]
S1-20 [36]	Users can permanently burn tokens via `bridgeBack()`	Low	Open
S0-2 [37]	Outdated and unmaintained project dependencies in balanceai-core	Info	Open
S0-3 [38]	Unmaintained project dependency proc-macro-error in evm-bridge-service	Info	Open
S0-4 [39]	RPC URL and bridge contract address is hardcoded within the codebase	Info	Open
S0-12 [40]	Use of post-increment in loop iterators increases gas costs	Gas	Partially mitigated [41]
S0-13 [42]	Reverting via failed `require` statements consumes more gas	Gas	Open
S0-14 [43]	Explicitly initializing a variable to default values costs unnecessary gas	Gas	Open
S0-17 [44]	The uncached use of `array.length` can increase gas	Gas	Mitigated [45]
S0-18 [46]	Deprecated dependencies used for smart contracts	Gas	Open
S0-21 [47]	Use of `public` instead of `external` increases gas	Gas	Open

Table 5.1.1: Findings overview

6 Detailed findings

6.1 S3-19: Replayed teleport operation can disrupt internal accounting

Attack scenario	An attacker tricks the user to submit same teleport operation
Component	balanceai-core, evm-bridge-solidity, evm-bridge-service
Tracking	[9]
Attack impact	Users can lose funds while trying to transfer tokens via the bridge
Severity	High
Status	Mitigated [10] [11] [12] [13] [14]

Background
BalanceAI provides a bridge to transfer tokens between Ethereum and Substrate chains, by burning or locking tokens on one chain and minting or unlocking them on the other chain. This is referred to as teleport operation in the codebase.

Issue description
An attacker can trick a user to call multiple duplicate teleport operations within the same block. In this case, the multiple calls are treated as a single teleport operation on the destination chain resulting in the amount getting transferred only once.

In the scenario when a user wants to transfer funds from Ethereum to Substrate. The user will call teleport [48] on the EVM side, providing the `subAccount` to transfer the funds to, the `amount` to transfer and the `nonce`. This `nonce` is user defined and there is no validation on the `nonce`.

```
function teleport(uint value, bytes32 subAccount, bytes32 nonce) external isNotPaused
{
        ...
        emit Teleport(msg.sender, value, subAccount, nonce);
}
```

Once the teleport operation reaches the destination module, the `nonce` is discarded and only the `evm_chain_id`, `evm_address`, `evm_block_number`, `subAccount` and `amount` is passed to the substrate pallet using the `Vouch` [49] operation by the guardians. A `teleport_id` is then generated using these parameters.

```
pub fn vouch(
       ...
       let teleport_id = Self::get_teleport_id(evm_chain_id, &evm_address, evm_block,
&account, value);
       ...
}
```

If a user submits multiple duplicate teleport operations within one block, all operations will generate the same `teleport_id`. Therefore, on the EVM chain the tokens will be burned multiple times, but on the substrate side they can be withdrawn only once [50].

The same issue is also applicable for the scenario of transferring funds from Substrate chains to Ethereum [51] [52] [53].

Risk
Users could accidentally lose funds or maliciously disrupt the total available supply by invoking the teleport function multiple times.

Mitigation
The `nonce` value in the Teleport function on the EVM side should not be a user defined parameter, instead should be incremented with every operation. This `nonce` should be used on the substrate side to generate the `teleport_id` to distinguish among multiple operations.

On the substrate pallet, the extrinsic `teleport` should have a nonce which should be used to generate the `teleportId` on the EVM side.

Remediation
The BalanceAI team remediated this issue on the EVM side by introducing an incremental `nonce` to the `teleport()` function in `Bridge.sol` and included nonce in the calculation of `teleport_id` [10] [11]. The same was done on the substrate side as well [12] [13] [14].

6.2 S3-22: Missing `disavow` logic enables malicious guardians to drain liquidity

Attack scenario	**A malicious guardian vouches for a fake teleport**
Component	evm-bridge-service
Tracking	[15]
Attack impact	Attackers can inflate the total token supply as well as drain the liquidity
Severity	High
Status	Mitigated [16] [54] [17] [18] [19] [20]

Background
Guardians run the `evm-bridge-service` which monitors teleport operations from either side of the bridge. When a `Teleport` event is detected, the guardians dispatch a `Vouch` message to the destination module. When guardians submit a `Vouch` message, peer-guardians are expected to verify the authenticity of the vote and, if illegitimate, submit a `Disavow` message.

Issue description
The `evm-bridge-service` monitors for the `Teleport` event from substrate [55] as well as EVM [56] side and automatically dispatches the `Vouch` message. There is not automated functionality to verify the authenticity of peer-guardian `Vouch` messages. This means that to flag a malicious `Vouch` message, peer-guardians must both:

- manually inspect the source chain-state
- manually submit a call to `disavow()` on the destination module (either `evm-bridge-pallet` [57] or `Bridge.sol` [58])

The user is allowed to withdraw the funds on the destination chain, once a teleport operation has reached the `vouchThreshold` and the `cooldownPeriod` is over. In case, the `cooldownPeriod` is set too low, a malicious guardian can submit an illegitimate `Vouch` message, allowing the funds to be withdrawn on the destination chain before an honest guardian can `Disavow`.

Risk
Relying on guardians to manually intervene and disavow malicious operations, without incorporating automated handling in the guardian node logic, is impractical. Even if manual intervention were feasible, a short `cooldownPeriod` would not provide sufficient time for such actions to be effectively executed.

In this model, a group of malicious guardians (>= `VouchThreshold`) could arbitrarily mint tokens on the destination chain without being challenged by honest guardians. If exploited, this vulnerability could:

- Inflate the token supply, leading to significant value loss for all users
- Disrupt the balance of liquidity between the EVM (`Bridge.sol`) and substrate (`evm-bridge-pallet`) modules, causing systemic instability

In a scenario where an attacker exploits the system to mint [59] tokens on the EVM module via the `bridgedTo()` function, they could subsequently execute a legitimate `bridgeBack()` [60] operation using the illicitly minted tokens to unlock liquidity from the `evm-bridge-pallet`. Since liquidity is pooled within the `evm-bridge-pallet`, the attacker could drain funds belonging to other users. Alternatively, the attacker could target the Substrate chain directly to unlock user tokens without first disrupting liquidity.

This attack could be executed within a few blocks, depending on the coordination of the attacker and the duration of the `cooldownPeriod`, potentially leaving insufficient time for governance to intervene by invoking `pause()`. If the attack results in tokens being minted in the `wBAI.sol` contract without

bridging them back to Substrate, governance intervention would be impossible due to the absence of remediation mechanisms.

While guardians are trusted entities selected by governance, the current implementation relies too heavily on this trust, with no consequences for abuse. This is particularly evident given the absence of slashing for dishonest actions and the inability of honest guardians to effectively disavow incorrect teleport operations.

Mitigation
We recommend automating the process of verifying peer-guardian `Vouch` messages and submitting `Disavow` messages, instead of relying on manual intervention which would likely not occur in time.

In this approach peer-guardians must inspect the chain state of the source chain for every single new `Vouch` message submitted, and call `Vouch` or `Disavow` accordingly.

Remediation
The issue was mitigated by verifying each teleport operation after a `Vouch` event is emitted and emitting a `Disavow` if the teleport is invalid [16] [54] [17] [18] [19] [20].

6.3 S2-8: Aborted teleport permanently locks user funds

Attack scenario	Multiple guardians maliciously disavow a teleport operation
Component	balanceai-core, evm-bridge-solidity
Tracking	[21]
Attack impact	User funds will be locked
Severity	Medium
Status	Open

Background
Guardians monitor teleport operations and submit either `vouch` or `disavow` messages for them. After the cooldown period if there are enough vouches for an operation then the funds can be withdrawn on the destination chain.

Issue description
Guardians have the ability to cast `vouch()` or `disavow()` votes for each teleport operation. If the disavow votes pass the `disavowThreshold` [61] the teleport operation is aborted, and the `Aborted()` [62] event is emitted.

Here is an example from the Ethereum component `Bridge.sol`:

```
if (disavows.length + 1 < disavowThreshold) {
        disavows.push(msg.sender);
} else {
        . . .
        emit Aborted(subAccount, subBlock, account, value);
}
```

These `Aborted` event emissions are not monitored or handled appropriately to unlock user funds that were either burned by `bridgeBack()` or locked in `evm-bridge-pallet`.

Tokens may also be locked if not enough guardians `vouch()` for a teleport operation within a reasonable time. In this scenario the operation has not actually been aborted but remains in a state of infinite processing.

Risk
In cases where `teleport` operations fail user funds will be completely locked due to the insufficient handling of `Aborted()`events emitted by the receiving component. This may impact BalanceAI's community sentiment and increase the attack impact of potential rogue Guardians (in-line with finding).

Furthermore, if the guardian network, at any point, becomes insufficient: tokens may become locked in infinite teleport operations that do not receive enough vouches.

Mitigation
We recommend integrating logic to safely handle cases where teleport operations fail and unlock user tokens on the source chain.

Furthermore, we recommend implementing some time-out mechanism where if an operation is not either explicitly successful or explicitly unsuccessful after some extended period, the operation can be aborted, and the token unlock can be handled by the abort functionality (as described previously).

6.4 S2-1: Missing and incorrect runtime weights for balanceai-core pallets

Attack scenario	**An attacker spams the network with cheap transactions**
Component	balanceai-core
Tracking	[22]
Attack impact	Attackers may spam the network and cause storage bloating.
Severity	Medium
Status	Mitigated [23]

Background
Every substrate extrinsic needs to have a weight assigned which is calculated using a benchmarking system.

Issue description
The `WeightInfo` in the runtime configuration determines the weight of the extrinsics. The runtime weights for several pallets are configured to be zero i.e. `type WeightInfo = ()`.

```
impl pallet_balances::Config for Runtime {
    ...
    type WeightInfo = ();
    ...
}
```

Some of the affected pallets include `pallet_balances` [63], `pallet_babe` and `pallet_fast_unstake`.

Additionally, several FRAME pallets have weights configured to `SubstrateWeight`, instead of the actual runtime. Some of these affected pallets include `pallet_scheduler` [64], `pallet_staking` and `pallet_election_provider_multi_phase`.

```
impl pallet_scheduler::Config for Runtime {
     ...
     type WeightInfo = pallet_scheduler::weights::SubstrateWeight<Runtime>;
     ...
}
```

`SubstrateWeight` is benchmarked with external runtimes such as Polkadot or Kusama and it doesn't reflect the actual runtime weight of BalanceAI pallets.

Risk
As these pallet extrinsic weights are not dependent on the actual runtime configuration, this could lead to underweight extrinsics. Setting the weights to `()` effectively makes it a zero-cost execution for the extrinsic. Both the scenarios potentially lead to low-effort attacks such as spamming, storage bloating, and block stalling when invoking the extrinsics.

Mitigation
Configure the `WeightInfo` of all pallets, including the Substrate ones, using the actual runtime benchmarks.

Remediation
The issue was remediated by benchmarking all the pallets and adding correct weight information in the runtime [23].

6.5 S2-5: Nothing at stake could enable dishonest guardians

Attack scenario	**A malicious guardian vouches for an illegitimate operation or disavows a legitimate operation**
Component	balanceai-core, evm-bridge-solidity
Tracking	[24]
Attack impact	Users lose funds due to insufficient vouches or aborted teleport operation
Severity	Medium
Status	Open

Background
Blockchain networks fundamentally rely on both:

- enough nodes to remain byzantine fault tolerant
- mechanisms to disincentivize dishonest nodes (provided the majority remain honest)

In BalanceAI, guardians monitor the network to detect the transfer event and either vouch or disavow the transfer. It is expressed in the documentation [65] that guardians participating in illegitimate processes will be removed, although malicious behavior is not explicitly disincentivized.

Issue description
In the current implementation, guardians are treated as trusted entities and are not required to provide any stake to assume their role, and there are no slashing mechanisms in place to penalize misbehavior.

The reactionary approach of governance intervention to remove the guardians, aims to limit the continued impact of malicious guardians but fails to introduce economic feasibility for dishonest participation in the first place.

Risk
The absence of appropriate stake may enable malicious behavior and result in `vouch` for illegitimate operations and `disavow` for legitimate operations.

Furthermore, governance intervention is entirely dependent on the quality of off-chain monitoring and may be too slow to effectively limit the impact of ongoing abuse.

Mitigation
In addition to the reactionary governance mechanism in place, we recommend implementing mechanisms to disincentivize dishonest guardians. This can be achieved by introducing slashing mechanisms.

6.6 S2-6: Thresholds are not derived from meaningful safety assumptions

Attack scenario	**The admin misconfigures the `VouchThreshold` to zero**
Component	balanceai-core, evm-bridge-solidity
Tracking	[25]
Attack impact	Attackers may transfer funds with too few or no vouches
Severity	Medium
Status	Open

Background
For a successful bridge transfer, the following three thresholds are defined:

- `vouchThreshold` – The minimum vouches required for a transfer to succeed
- `disavowThreshold` – The limit of disavows before a transfer is aborted
- `cooldownPeriod` – The minimum time that must elapse before a transfer can succeed

Issue description
The admin of `Bridge.sol` contract can set the values of the thresholds and there is no bound check on the values [66]. An admin may accidentally or maliciously misconfigure these thresholds with null or zero values, causing the transfers to succeed or fail without sufficient guardian input. The value of these thresholds may also become inadequate over time if there is increase or decrease in the number of guardians in the network.

For example: In a network of 10 guardians, if the threshold for success is set to 6 vouches, then a vouch from more than 50% of the total guardians is required for a successful transfer. With time, if the number of guardians increase to 30 and the threshold is not updated, then the number of vouches required is only 20% of the total guardians.

Moreover, the threshold values may contain inconsistencies between the substrate and EVM modules, since there is verification that the threshold value is same on both sides. This could result in a scenario where a teleport operation could succeed on the substrate side, but the bridge-back operation could fail.

There exists also a threshold type-mismatch between the two modules. In Substrate the `VouchThreshold` is of type `u8` (maximum 255 vouches) [67], while the `vouchThreshold` in the EVM is `u256` (maximum 2^256 -1 vouches) [68].

Risk
The lack of a bound check on the thresholds can allow a transfer to succeed with no vouches, causing the bridging service to lose integrity in the event of misconfiguration. It also makes the service vulnerable to Sybil attacks if the `vouchThreshold` is too low compared to number of guardians in the network.

Additionally, mismatch in the threshold value between substrate and EVM modules may allow teleport operations to be inconsistent.

Mitigation
We recommend deriving the `vouchThreshold` and `disavowThreshold` values from the quantity of active guardians in the network. This needs to be maintained as guardians join and leave the network. The threshold data types should also be consistent between modules.

Remediation
The BalanceAI team acknowledged the issue and will introduce algorithmic thresholds based on configured guardians in future releases.

6.7 S2-10: Unchecked `transfer()` could result in unexpected behavior

Attack scenario	**A user reclaims tokens on an invalid address**
Component	evm-bridge-solidity
Tracking	[26]
Attack impact	The transfer fails, but may not result in a failure
Severity	Medium
Status	Open

Background
In Solidity, the ERC-20 `transfer()` function returns a boolean value to indicate success or failure. If the return value is not properly checked, it could result in undetected failures or unexpected behavior.

Issue description
The function `reclaimToken()` performs an ERC-20 `transfer()` [69] call to an arbitrary address `_token`. The success of this operation is not ensured. Depending on ERC-20 implementation, `transfer()` may not revert on failure, but instead return a boolean value. These return values should be checked and handled accordingly.

```
function reclaimToken(address _token) public onlyOwner {
    ...
    token.transfer(owner(), balance);
    ...
}
```

Risk
Unchecked return values can lead to unexpected behavior, such as execution continuing under the assumption a transfer has succeeded.

Mitigation
We recommend consistently checking the return values of external calls and appropriately handling subsequent executions.

As a best practice, consider implementing the safeERC20 [70] library which wraps these calls and handles associated values by default.

6.8 S1-7: Missing access control for `enableBridgeBack()`

Attack scenario	A non-privileged user activates the bridge back functionality
Component	evm-bridge-solidity
Tracking	[27]
Attack impact	Attackers may exploit this feature to drain assets or disrupt liquidity
Severity	Low
Status	Mitigated [28]

Background
The `enableBridgeBack()` function is responsible for managing the state of the `bridge_back_active` boolean. This variable is used to deny access to the `bridgeBack()` function.

Issue description
The `enableBridgeBack()` [71] is publicly callable by anyone and implements no access control restrictions. This functionality is either incomplete or entirely redundant since any non-privileged user may arbitrarily activate bridge back by calling `enableBridgeBack()` then calling `bridgeBack()` [72].

Risk
Bridging tokens back to Substrate may be enabled and performed at any time, which given the context, seems to violate the intended use. Furthermore, once enabled, the `bridgeBack()` function cannot be deactivated, making this irrecoverable.

Mitigation
We recommend implementing appropriate access control checks (for example `onlyOwner`) on the `enableBridgeBack()` function.

Remediation
This has since been mitigated by the team by implementing the `onlyOwner` modifier [28].

6.9 S1-9: Single-step update could render `admin` and `owner` irrecoverable

Attack scenario	Accidental misconfiguration results in loss of privilege
Component	evm-bridge-solidity
Tracking	[29]
Attack impact	Access loss to critical functionality
Severity	Low
Status	Mitigated [30]

Background
When updating the configuration of admin in `Bridge.sol` or owner in wBAI.sol, 2-step verification should be implemented to ensure the new admin or owner address is correct, and able to sufficiently interact with the system.

Issue description
When the function `setAdmin()` is called the admin role is immediately updated to the input `_admin` [73].

A similar implementation is found in the dependency `Ownable.sol` used by `wBAI.sol`: where `transferOwnership()` is performed in a single step [74].

Risk
Lack of 2-step transferal increases the vulnerable surface area to include human error. If the `admin` or `owner` is set to an address that is uncontrollable or does not have sufficient capabilities to interact with the system, critical functionality will be unreachable.

Mitigation
We recommended implementing a mechanism for the current admin to set a `pendingAdmin` that must be verified by the newly proposed `pendingAdmin` via some `accept()` function. The usage of OpenZeppelin's `Ownable.sol` dependency should be replaced with `Ownable2Step.sol` [75].

Remediation
This issue was remediated by migrating both contracts to `Ownable2Step.sol` in-line with our recommendations [30].

6.10 S1-11: Unsafe use of `transfer()`

Attack scenario	The call fails due to insufficient gas on `transfer()`
Component	evm-bridge-solidity
Tracking	[31]
Attack impact	Localized denial of service and unexpected behavior
Severity	Low
Status	Open

Background

The `reclaimToken()` [76] function performs a transfer() operation to the owner address.

When performing low-level calls in Solidity there are three main options:

- `send()` - forwards 2300 gas and returns boolean for success
- `transfer()` - forwards 2300 gas and reverts on failure
- `call()` - forwards all remaining gas and returns boolean on failure

Issue description

If the recipient (`owner`) is a smart contract, `transfer()` will not send sufficient gas to perform meaningful operations and may always revert. To contextualize this an `SSTORE` operation costs 2100 units of gas [77].

Risk

If owner is, or becomes, a smart contract there will be insufficient gas forwarding to execute reception logic, such as internal accounting. This may become more severe with Ethereum forks that alter the gas cost of operations. See EIP-1884 as an example of when such assumptions around `transfer()` compatibility were broken due to increased gas costs [78].

Mitigation

It is standard best practice in Solidity that low-level external calls are performed using `call()`, alongside some re-entrant checks, instead of `send()` or `transfer()`. This ensures future compatibility with recipients. See an example of the `sendValue()` function from OpenZeppelin [79].

When using `call()` the return values should be checked, and custom reversion should be performed by the caller. Furthermore, callbacks will no longer be prevented by gas limitations and could thus require some reentrancy guard.

6.11 S1-15: Unsafe use of `abi.encodeWithSelector()`

Attack scenario	Low levels call results in unexpected behavior due to mis-matched types
Component	evm-bridge-solidity
Tracking	[32]
Attack impact	External calls may unexpectedly fail resulting in undefined behavior.
Severity	Low
Status	Mitigated [33]

Background
Performing low-level calls in Solidity should be performed with caution. Appropriate sanity checks should be implemented to avoid unforeseen behavior.

Issue description
There are multiple instances of `abi.encodeWithSelector` throughout the codebase. This does not implement checks on input arguments, and as such mismatched types may not be caught at compilation. See the following instances [80] and [81].

Risk
Incorrect argument types in low-level calls may introduce unexpected behavior or compatibility issues.

Mitigation
Since version 0.8.11 of Solidity `abi.encodeCall` now provides type-safe encoding and is the recommended best practice alternative to `abi.encodeWithSelector` [82]. We recommended migrating all instance to use `abi.encodeCall`.

Remediation
The issue was mitigated by the team, by migrating to `abi.encodeCall` [33].

6.12 S1-16: Missing `address(0)` checks for `admin` update

Attack scenario	Accidental misconfiguration results in loss of privilege
Component	evm-bridge-solidity
Tracking	[34]
Attack impact	Loss of certain critical functionality
Severity	Low
Status	Mitigated [35]

Background
The privileged role `admin` does not have appropriate input checks to avoid misconfiguration when updated by `setAdmin()` or initialized at construction.

Issue description
When updating the value of `admin` via `setAdmin() [83]` the new `_admin` is not checked for potential configuration to `address(0)`. If `setAdmin()` is accidentally called with null input (e.g zero), the admin address will become irrecoverable.

Risk
Accidental misconfiguration of access control roles would lock critical functionality to manage the bridge, such as pause/unpause functionality and governance over thresholds.

Mitigation
We recommended best practice to filter updates to critical data such as `admin` for `address(0)`.

Remediation
The development team implemented appropriate address zero checks [35] in-line with this recommendation.

6.13 S1-20: Users can permanently burn tokens via `bridgeBack()`

Attack scenario	**User accidentally or maliciously calls bridgeBack()**
Component	evm-bridge-solidity
Tracking	[36]
Attack impact	System token accounting may be manipulated and users may experience financial loss.
Severity	Low
Status	Open

Background
To bridge tokens back from the EVM to Substrate, a user will call `teleport()` which then executes a sub-call to `bridgeBack()` in `wBAI.sol` then emits the `Teleport` event.

In `wBAI.sol` the `bridgeBack()` function may be arbitrarily called without directly calling it via the `Bridge.sol` `teleport()` function.

Issue description
There are no caller checks performed within the `bridgeBack()` function [84]. Since the `BridgeBack` event emitted here is not monitored off-chain, calling this function will not trigger a valid teleport operation, unless explicitly called through `teleport()` [85].

Risk
Users are needlessly exposed to the risk of accidental system misuse which would result in their tokens being burned, without remediation. Furthermore, this essentially exposes the `_burn()` function publicly, which may be abused to manipulate internal system accounting.

Mitigation
We recommend introducing some check to ensure that this function may only be called by the `Bridge.sol` contract address, so users do not accidentally `_burn()` tokens through an unintended operation.

6.14 S0-2: Outdated and unmaintained project dependencies in balanceai-core

Attack scenario	**An attacker exploits the known vulnerabilities in the dependencies**
Component	balanceai-core
Tracking	[37]
Attack impact	Attackers may cause resource exhaustion
Severity	Info
Status	Open

Background

In a rust project, dependencies are external crates that the project relies on for additional functionalities. Crates are managed through the cargo package manager, which handles downloading, compiling, and managing these dependencies.

Issue description

Several dependencies in the project are outdated or unmaintained. These crates no longer receive security patches or updates, leaving the project vulnerable to potential security risks. The affected dependencies are mentioned below:

```
Crate:     rustls
Version:   0.20.9
Title:     `rustls::ConnectionCommon::complete_io` could fall into an infinite loop
based on network input
Date:      2024-04-19
ID:        RUSTSEC-2024-0336
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0336
Severity:  7.5 (high)
Solution:  Upgrade to >=0.23.5 OR >=0.22.4, <0.23.0 OR >=0.21.11, <0.22.0

Crate:     ansi_term
Version:   0.12.1
Warning:   unmaintained
Title:     ansi_term is Unmaintained
Date:      2021-08-18
ID:        RUSTSEC-2021-0139
URL:       https://rustsec.org/advisories/RUSTSEC-2021-0139

Crate:     atty
Version:   0.2.14
Warning:   unmaintained
Title:     `atty` is unmaintained
Date:      2024-09-25
ID:        RUSTSEC-2024-0375
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0375

Crate:     mach
Version:   0.3.2
Warning:   unmaintained
Title:     mach is unmaintained
Date:      2020-07-14
ID:        RUSTSEC-2020-0168
URL:       https://rustsec.org/advisories/RUSTSEC-2020-0168

Crate:     parity-util-mem
Version:   0.12.0
Warning:   unmaintained
Title:     parity-util-mem Unmaintained
Date:      2022-11-30
ID:        RUSTSEC-2022-0080
```

```
URL:       https://rustsec.org/advisories/RUSTSEC-2022-0080

Crate:     parity-wasm
Version:   0.45.0
Warning:   unmaintained
Title:     Crate `parity-wasm` deprecated by the author
Date:      2022-10-01
ID:        RUSTSEC-2022-0061
URL:       https://rustsec.org/advisories/RUSTSEC-2022-0061

Crate:     proc-macro-error
Version:   1.0.4
Warning:   unmaintained
Title:     proc-macro-error is unmaintained
Date:      2024-09-01
ID:        RUSTSEC-2024-0370
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0370

Crate:     atty
Version:   0.2.14
Warning:   unsound
Title:     Potential unaligned read
Date:      2021-07-04
ID:        RUSTSEC-2021-0145
URL:       https://rustsec.org/advisories/RUSTSEC-2021-0145
```

Risk

Unmaintained dependencies may have unresolved vulnerabilities and/or over time might become incompatible with other components of the project.

Mitigation

Refer to the official RustSec Advisory Database [86] and other security advisories to identify and fix the vulnerable crates. If the project dependencies have known vulnerabilities, prioritize updating or replacing them. If a crate is no longer maintained or updated, consider replacing it with an alternative library that provides similar functionality that is actively maintained and secure.

6.15 S0-3: Unmaintained project dependency proc-macro-error in evm-bridge-service

Attack scenario	**An attacker exploits the known vulnerabilities in the dependencies**
Component	balanceai-core
Tracking	[38]
Attack impact	Attackers may cause resource exhaustion
Severity	Info
Status	Open

Background
In a rust project, dependencies are external crates that the project relies on for additional functionalities. Crates are managed through the cargo package manager, which handles downloading, compiling, and managing these dependencies.

Issue description
The dependency `proc-macro-error` in the project is marked as unmaintained. This crate will no longer receive security patches or updates, leaving the project vulnerable to potential security risks.

```
Crate:     proc-macro-error
Version:   1.0.4
Warning:   unmaintained
Title:     proc-macro-error is unmaintained
Date:      2024-09-01
ID:        RUSTSEC-2024-0370
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0370
```

Risk
Unmaintained dependencies may have unresolved vulnerabilities and/or over time might become incompatible with other components of the project.

Mitigation
Refer to the official RustSec Advisory Database [86] and other security advisories to identify and fix the vulnerable crates. If the project dependencies have known vulnerabilities, prioritize updating or replacing them. If a crate is no longer maintained or updated, consider replacing it with an alternative library that provides similar functionality that is actively maintained and secure.

6.16 S0-4: RPC URL and bridge contract address is hardcoded within the codebase

Attack scenario	**An update modifies the URL or the bridge contract address**
Component	evm-bridge-service
Tracking	[39]
Attack impact	Hardcoding information makes the code harder to be maintained
Severity	Info
Status	Open

Issue description

The RPC URL to connect to the EVM chain and the bridge contract address is hardcoded in the codebase. These are not sensitive information and will be publicly available for the users to interact with.

```
let rpc_url = "wss://sepolia.infura.io/ws/v3/7920389bb*****";
let ws = WsConnect::new(rpc_url);
...
// Subscribe to bridge contract.
let contract = Bridge::new(
        "0x36d15697e39cf*****".parse()?,
        provider.clone(),
);
```

Risk

Although, there is no security risk since the information is publicly available, it is still a best practice deviation to have them hardcoded in the code. In case the URL changes over time, it makes the code harder to maintain, since the information is scattered throughout the codebase instead of being managed from a central location.

Mitigation

We recommend moving the RPC URL and contract addresses to external configuration files such as '.env', '.json', or '.toml'.

6.17 S0-12: Use of post-increment in loop iterators increases gas costs

Component	evm-bridge-solidity
Tracking	[40]
Attack impact	User pays more money for executing the transactions
Severity	Gas
Status	Partially mitigated [41]

Background
Using post-increments (`i++`) in the loop iteration consumes more gas.

Issue description
For post-increments, the compiler creates another temporary variable to return the un-incremented value of '`i`'. While using pre-increments (`++i`), the compiler doesn't use a temporary variable and instead returns the incremented value of '`i`'.

A few instances exist, namely: `vouch()` [87], `disavow()` [88] and `bridgeTo()` [89]. Similarly, this applies to incrementing the nonce using `_nonce++` [90].

Risk
Failing to optimize gas costs could result in high transaction costs for users and may cause transaction failures if a single transaction consumes too much gas. This may impact the community sentiment towards the project, as users may be reluctant to interact with contracts that are expensive to use.

Mitigation
Post-increment consumes an estimated 5-gas more per iterations than pre-increment, resulting in significant unnecessary gas consumption in more complex operations. We recommend changing iterators to increment with `++i`.

Remediation
This has been partially mitigated by the BalanceAI development team [41].

6.18 S0-13: Reverting via failed `require` statements consumes more gas

Component	evm-bridge-solidity
Tracking	[42]
Attack impact	User pays more money for executing the transactions
Severity	Gas
Status	Open

Background
Reverting via failed require statements consumes unnecessary gas due to the, sometimes lengthy, strings detailing the reason for the revert. Revert logic should be handled using custom errors and not strings.

Issue description
Multiple instances of reverting via `require()` were found in `wBAI.sol [91] [92] [93]`, though similar cases exist throughout the codebase. For example:

```
require(_amount <= balanceOf(_msgSender()), "Not enough balance"); // Un-optimized
```

Risk
Failing to optimize gas costs could result in high transaction costs for users and may cause transaction failures if a single transaction consumes too much gas. This may impact the community sentiment towards the project, as users may be reluctant to interact with contracts that are expensive to use.

Mitigation
The use of `require()` should be replaced by conditional logic with a nested `revert`. These reverts should avoid hard-coded string descriptions and instead opt for reusable custom errors that (if applicable) ship necessary data off-chain. An example of the fix could be:

```
// Potential Fix
error InsufficientBalance()
if (_amount > balanceOf(_msgSender())) {
        revert InsufficientBalance() // Optimized better practice
}
```

Using the `revert` will improve gas efficiency by around `227-gas` (dependent on prior complexity) and improve uniformity of error codes.

6.19 S0-14: Explicitly initializing a variable to default values costs unnecessary gas

Component	evm-bridge-solidity
Tracking	[43]
Attack impact	User pays more money for executing the transactions
Severity	Gas
Status	Open

Background
When state variables are declared their value is assigned at contract construction. If no value is provided, they are initialized to default values. For Boolean this is `false` and for unsigned integers this is `0`.

Issue description
Explicitly assigning default values to state variables costs additional gas at deployment. For example: uint256 value; is cheaper than `uint256 value = 0;`

Multiple instances were found in the `wBAI.sol` [94] [95] [96] [97].

Risk
Failing to optimize gas costs could result in high transaction costs for users and may cause transaction failures if a single transaction consumes too much gas. This may impact the community sentiment towards the project, as users may be reluctant to interact with contracts that are expensive to use.

Mitigation
Do not explicitly assign default values to state variables, instead allow that to be handled by the initialization process. Note that this only saves gas at deployment.

6.20 S0-17: The uncached use of `array.length` can increase gas

Component	evm-bridge-solidity
Tracking	[44]
Attack impact	User pays more money for executing the transactions
Severity	Gas
Status	Mitigated [45]

Background
In Solidity `storage` reads are exponentially more computationally intensive than `memory` reads. Due to this the gas cost users pay for execution is far greater.

Issue description
When iterating over an array (for example disavows [98] or vouches [99]) the array length should be cached in memory to avoid performing storage access on each iteration. In the current implementation an `SLOAD` operation will be performed in each iteration when the `array.length` lookup is performed.

Risk
Failing to optimize gas costs could result in high transaction costs for users and may cause transaction failures if a single transaction consumes too much gas. This may impact the community sentiment towards the project, as users may be reluctant to interact with contracts that are expensive to use.

Mitigation
We recommend caching the array length in `memory` outside of the iteration.

Remediation
The development team fixed this issue by caching the length in memory [45].

6.21 S0-18: Deprecated dependencies used for smart contracts

Component	evm-bridge-solidity
Tracking	[46]
Attack impact	User pays more money for executing the transactions
Severity	Gas
Status	Open

Background
Solidity dependencies, like OpenZeppelin, provide pre-built, secure libraries for common smart contract functionalities such as token standards (ERC-20, ERC-721), access control, and security features. These libraries are rigorously audited and widely used to minimize vulnerabilities. Keeping these dependencies updated is thus crucial.

Issue description
The wBAI.sol contract is reliant on various OpenZeppelin dependencies, many of which are deprecated, and can introduce avoidable bugs. Another issue that arises is more gas costs, for example `ERC20.sol` in OpenZeppelin version 4.9.3 [100] consists of a lot of require statements with lengthy string descriptions and is hence, more costly to execute than version >5.0.0 [101].

Risk
Using the current outdated dependencies will cost significantly more in gas alongside potentially introducing unsafe logic.

Mitigation
We recommend upgrading to 5.0.0 with the important stipulation that: the `increaseAllowance()` and `decreaseAllowance()` functions should be manually implemented within the `wBAI.sol` contract. These two functions have been depreciated from `ERC20.sol` and refactored exclusively into the `safeERC20.sol` library since they technically did not comply with ERC-20: Token Standard [102] and increased the gas cost of deployment. This decision places responsibility on developers to decide if integration is necessary.

The rationale for including these functions is to prevent `approve()` calls updating `allowances` to be front-run. Front-running the `approve()` function is rare in the wild, but we still recommend mitigation against it as a best-practice.

6.22 S0-21: Use of `public` instead of `external` increases gas

Component	evm-bridge-solidity
Tracking	[46]
Attack impact	User pays more money for executing the transactions
Severity	Gas
Status	Open

Background
Solidity offers `external` and `public` function types with the only difference being `public` may be called both externally and internally from the within the contract.

Issue description
Setting functions that are never called internally, to `public` results in additional unnecessary gas costs as `public` functions copy their arguments from `calldata` to `memory`. In contrast, `external` functions read arguments directly from `calldata`, making them more gas-efficient for functions that do not need to be accessed internally.

Some examples include (although more cases exists): the functions `bridgedTo()` [103] and `bridgeBack()` [72].

Risk
Failing to optimize gas costs could result in high transaction costs for users and may cause transaction failures if a single transaction consumes too much gas. This may impact the community sentiment towards the project, as users may be reluctant to interact with contracts that are expensive to use.

Mitigation
We recommend changing all functions listed as `public` and that are not called internally to `external` to save gas costs when called.

7 Bibliography

[1] [Online]. Available: https://eips.ethereum.org/EIPS/eip-1822.

[2] [Online]. Available: https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable.

[3] [Online]. Available: https://github.com/srlabs/substrate-runtime-fuzzer.

[4] [Online]. Available: https://book.getfoundry.sh/forge/fuzz-testing.

[5] [Online]. Available: https://github.com/crytic/echidna.

[6] [Online]. Available: https://github.com/balanceainetwork/balanceai-core/.

[7] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-service/.

[8] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity.

[9] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/issues/12.

[10] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/commit/bd57883a5bc972d406cef523965c2ac673211754.

[11] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/commit/6c5ad2290d1d3b383ab256cf3dd6e3936b92dc68.

[12] [Online]. Available: https://github.com/balanceainetwork/balanceai-core/commit/5831f612fe115aa586584730dd6d79e06993316a.

[13] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-service/commit/7b6535abf6a0b4bcaf671f3bd2fbefae4f2d75ee.

[14] [Online]. Available: https://github.com/balanceainetwork/balanceai-core/commit/eef171c7686c7fd80a5a19dfec3f191588f4105f#diff-9c64616a8ab79a9a49b5bcd272e4140b6fecf5a3dd99f0aff0d199e783db5817R518.

[15] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-service/issues/3.

[16] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-service/commit/c6934280b30bb1dd3d2c177ce4083fde29d9b902.

[17] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-service/commit/6d0cdf026f5b5d7f90b6b325761a4791c5b43935.

[18] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-service/commit/26053f1a8e1d21033c86e43aa3e889c91428997c.

[19] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-service/commit/e73dbef9cf6f2e177592a981139a19eda5191ad2.

[20] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-service/commit/f0ff2ec3cd6225be4dec69130fe0416fcda4775e.

[21] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/issues/13.

[22] [Online]. Available: https://github.com/balanceainetwork/balanceai-core/issues/1.

[23] [Online]. Available: https://github.com/balanceainetwork/balanceai-core/commit/bd42861b5afbcdf6eb43c9aea7c970f73650ccec.

[24] [Online]. Available: https://github.com/balanceainetwork/balanceai-core/issues/14.

[25] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/issues/10.

[26] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/issues/8.

[27] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/issues/5.

[28] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/commit/517775be87680487b747d43279ba66f81203fbe9.

[29] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/issues/9.

[30] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/commit/59753259a0e1257cbc931901dc86811f271a5e2a.

[31] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/issues/6.

[32] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/issues/7.

[33] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/commit/1c4e4a2cc4b75c641dda4a6e2458adfc39cce20b.

[34] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/issues/11.

[35] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/commit/c4f512d02b06dc0fb71c747f90abf567b09a57f2.

[36] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/issues/14.

[37] [Online]. Available: https://github.com/balanceainetwork/balanceai-core/issues/2.

[38] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-service/issues/1.

[39] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-service/issues/2.

[40] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/issues/1.

[41] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/commit/9ce9e1ffaf82ab0069a49c78696395555b436bbe.

[42] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/issues/3.

[43] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/issues/2.

[44] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/issues/4.

[45] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/commit/bc21691beb5d549d17f18345d2ce2057db27de0e.

[46] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/issues/16.

[47] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/issues/15.

[48] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/Bridge.sol#L171.

[49] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-service/blob/80e650260817a62b808ac48ec7cb503985f5bb26/src/evm.rs#L66-L67.

[50] [Online]. Available: https://github.com/balanceainetwork/balanceai-core/blob/970ad0a2c65457784be447f972afbc18f610e0b8/pallets/evm-bridge-pallet/src/lib.rs#L353-L355.

[51] [Online]. Available: https://github.com/balanceainetwork/balanceai-core/blob/970ad0a2c65457784be447f972afbc18f610e0b8/pallets/evm-bridge-pallet/src/lib.rs#L221-L227.

[52] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-service/blob/80e650260817a62b808ac48ec7cb503985f5bb26/src/substrate.rs#L57-L63.

[53] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/Bridge.sol#L180.

[54] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-service/commit/8a2aed2082527e7c41a9e171ca9ca4e16cce67f3.

[55] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-service/blob/80e650260817a62b808ac48ec7cb503985f5bb26/src/substrate.rs#L50-L73.

[56] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-service/blob/80e650260817a62b808ac48ec7cb503985f5bb26/src/evm.rs#L66-L77.

[57] [Online]. Available: https://github.com/balanceainetwork/balanceai-core/blob/970ad0a2c65457784be447f972afbc18f610e0b8/pallets/evm-bridge-pallet/src/lib.rs#L284-L287.

[58] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/Bridge.sol#L198-L204.

[59] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/wBAI.sol#L2255.

[60] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/wBAI.sol#L2263.

[61] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/Bridge.sol#L215.

[62] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/Bridge.sol#L224.

[63] [Online]. Available: https://github.com/balanceainetwork/balanceai-core/blob/970ad0a2c65457784be447f972afbc18f610e0b8/runtime/src/lib.rs#L438.

[64] [Online]. Available: https://github.com/balanceainetwork/balanceai-core/blob/970ad0a2c65457784be447f972afbc18f610e0b8/runtime/src/lib.rs#L358.

[65] [Online]. Available: https://github.com/balanceainetwork/balanceai-core/blob/970ad0a2c65457784be447f972afbc18f610e0b8/pallets/evm-bridge-pallet/README.md#rouge-guardians-vouch-for-teleports-that-did-not-happen.

[66] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/Bridge.sol#L153.

[67] [Online]. Available: https://github.com/balanceainetwork/balanceai-core/blob/970ad0a2c65457784be447f972afbc18f610e0b8/pallets/evm-bridge-pallet/src/lib.rs#L463-L464.

[68] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/Bridge.sol#L12.

[69] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/wBAI.sol#L2283.

[70] [Online]. Available: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/utils/SafeERC20.sol.

[71] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/wBAI.sol#L2237.

[72] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/wBAI.sol#L2263.

[73] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/Bridge.sol#L114.

[74] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/wBAI.sol#L1007.

[75] [Online]. Available: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol.

[76] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/wBAI.sol#L2284.

[77] [Online]. Available: https://github.com/wolflo/evm-opcodes/blob/main/gas.md#a7-sstore.

[78] [Online]. Available: https://eips.ethereum.org/EIPS/eip-1884.

[79] [Online]. Available: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/72c152dc1c41f23d7c504e175f5b417fccc89426/contracts/utils/Address.sol#L33.

[80] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/Bridge.sol#L167.

[81] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/Bridge.sol#L245.

[82] [Online]. Available: https://soliditylang.org/blog/2021/12/20/solidity-0.8.11-release-announcement/.

[83] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/Bridge.sol#L114.

[84] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/wBAI.sol#L2263.

[85] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/Bridge.sol#L170.

[86] [Online]. Available: https://rustsec.org/advisories/.

[87] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/Bridge.sol#L191.

[88] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/Bridge.sol#L209.

[89] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/wBAI.sol#L2254.

[90] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/wBAI.sol#L2258.

[91] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/wBAI.sol#L2252.

[92] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/wBAI.sol#L2244.

[93] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/wBAI.sol#L2264.

[94] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/wBAI.sol#L2209.

[95] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/wBAI.sol#L2211.

[96] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/wBAI.sol#L2212.

[97] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/wBAI.sol#L2214.

[98] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/Bridge.sol#L209.

[99] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/Bridge.sol#L191.

[100] [Online]. Available: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.9/contracts/token/ERC20/ERC20.sol.

[101] [Online]. Available: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/ERC20.sol.

[102] [Online]. Available: https://eips.ethereum.org/EIPS/eip-20.

[103] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/blob/347bc95424315928601b5095cb6cea02a6d2c824/src/wBAI.sol#L2249.

[104] [Online]. Available: https://github.com/srlabs/ziggy.

[105] [Online]. Available: https://github.com/balanceainetwork/evm-bridge-solidity/commit/59753259a0e1257cbc931901dc86811f271a5e2a.

Appendix A: Technical services

Security Research Labs delivers extensive technical expertise to meet your security needs. Our comprehensive services include software and hardware evaluation, penetration testing, red team testing, incident response, and reverse engineering. We aim to equip your organization with the security knowledge essential for achieving your objectives.

SOFTWARE EVALUATION We provide assessments of application, system, and mobile code, drawing on our employees' decades of experience in developing and securing a wide variety of applications. Our work includes design and architecture reviews, data flow and threat modelling, and code analysis with targeted fuzzing to find exploitable issues.

BLOCKCHAIN SECURITY ASSESSMENTS We offer specialized security assessments for blockchain technologies, focusing on the unique challenges posed by decentralized systems. Our services include smart contract audits, consensus mechanism evaluations, and vulnerability assessments specific to blockchain infrastructure. Leveraging our deep understanding of blockchain technology, we ensure your decentralized applications and networks are secure and robust.

POLKADOT ECOSYSTEM SECURITY We provide comprehensive security services tailored to the Polkadot ecosystem, including parachains, relay chains, and cross-chain communication protocols. Our expertise covers runtime misconfiguration detection, benchmarking validation, cryptographic implementation reviews, and XCM exploitation prevention. Our goal is to help you maintain a secure and resilient Polkadot environment, safeguarding your network against potential threats.

TELCO SECURITY We deliver specialized security assessments for telecommunications networks, addressing the unique challenges of securing large-scale and critical communication infrastructures. Our services encompass vulnerability assessments, secure network architecture reviews, and protocol analysis. With a deep understanding of telco environments, we ensure robust protection against cyberthreats, helping maintain the integrity and availability of your telecommunications services.

DEVICE TESTING Our comprehensive device testing services cover a wide range of hardware, from IoT devices and embedded systems to consumer electronics and industrial controls. We perform rigorous security evaluations, including firmware analysis, penetration testing, and hardware-level assessments, to identify vulnerabilities and ensure your devices meet the highest security standards. Our goal is to safeguard your hardware against potential attacks and operational failures.

CODE AUDITING We provide in-depth code auditing services to identify and mitigate security vulnerabilities within your software. Our approach includes thorough manual reviews, automated static analysis, and targeted fuzzing to uncover critical issues such as logic flaws, insecure coding practices, and exploitable vulnerabilities. By leveraging our expertise in secure software development, we help you enhance the security and reliability of your codebase, ensuring robust protection against potential threats.

PENETRATION & RED TEAM TESTING We perform high-end penetration tests that mimic the work of sophisticated adversaries. We follow a formal penetration testing methodology that emphasizes repeatable, actionable results that give your team a sense of the overall security posture of your organization.

SOURCE CODE-ASSISTED SECURITY EVALUATIONS We conduct security evaluations and penetration tests based on our code-assisted methodology that lets us find deeper vulnerabilities, logic flaws, and fuzzing targets than a black-box test would reveal. This gives your team a stronger assurance that the significant security-impacting flaws have been found and corrected.

SECURITY DEVELOPMENT LIFECYCLE CONSULTING We guide organizations through the Security Development Lifecycle to integrate security at every phase of software development. Our services include secure coding training, threat modelling, security design reviews, and automated security testing implementation. By embedding security practices into your development processes, we help you proactively identify and mitigate vulnerabilities, ensuring robust and secure software delivery from inception to deployment.

REVERSE ENGINEERING We assist clients with reverse engineering efforts that are not associated with malware or incident response. We also provide expertise in investigations and litigation by acting as experts in cases of suspected intellectual property theft.

HARDWARE EVALUATION We evaluate new hardware devices ranging from novel microprocessor designs, embedded systems, mobile devices, and consumer-facing end products to core networking equipment that powers Internet backbones.

VULNERABILITY PRIORITIZATION We streamline vulnerability information processing by consolidating data from compliance checks, audit findings, penetration tests, and red team insights. Our prioritization and automation strategies ensure that the most critical vulnerabilities are addressed promptly, enhancing your organization's security posture. By systematically categorizing and prioritizing risks, we help you focus on the most impactful threats, ensuring efficient and effective remediation efforts.

SECURITY MATURITY REVIEW We conduct comprehensive security maturity reviews to evaluate your organization's current security practices and identify areas for improvement. Our assessments cover a wide range of criteria, including policy development, risk management, incident response, and security awareness. By benchmarking against industry standards and best practices, we provide actionable insights and recommendations to enhance your overall security posture and guide your organization toward achieving higher levels of security maturity.

SECURITY TEAM INCUBATION We provide comprehensive support for building security teams for new, large-scale IT ventures. From Day 1, our ramp-up program offers essential security advisory and assurance, helping you establish a robust security foundation. With our proven track record in securing billion-dollar investments and launching secure telco networks globally, we ensure your new enterprise is protected against cyberthreats from the start.

HACKING INCIDENT SUPPORT We offer immediate and comprehensive support in the event of a hacking incident, providing expert analysis, containment, and remediation. Our services include detailed forensics, malware analysis, and root cause determination, along with actionable recommendations to prevent future incidents. With our rapid response and deep expertise, we help you mitigate damage, recover swiftly, and strengthen your defenses against potential threats.

Appendix B: Risk and advisory services

Security Research Labs enhances an organization's security and risk management capabilities. We offer a practical approach to information security that aligns with our clients' tolerance for security processes and programs. Our team of industry-leading experts brings a diverse range of security and risk management skills, providing clients with deep technical expertise, strategic security leadership, and effective incident response capabilities whenever needed.