BalanceAI Baseline Security Assurance

Hacking assessment report

v1.1, 11 July 2024

Audited By: Gabriel Arnautu gabriel@srlabs.de

Marc Heuse marc@srlabs.de

Abstract. This work describes the result of the thorough and independent security assurance audit of the BalanceAI blockchain performed by Security Research Labs. Security Research Labs is a consulting firm that has been providing specialized audit services in the Polkadot ecosystem since 2019, including for the Substrate and Polkadot projects.

During this study, the BalanceAI team provided access to relevant documentation and supported the research team effectively. The code of the blockchain [1] was verified to assure that the business logic of the product is resilient to hacking and abuse.

The research team identified several issues ranging from high to info-level severity, many of which concerned the runtime configuration, weights benchmarking and logic implementation for the *evm-bridge-pallet*.

In addition to mitigating the open issues, Security Research Labs recommends further enhancing the existing documentation and tests around the BalanceAI system.

Content

1 Disclaimer

This report describes the findings and core conclusions derived from the audit carried out by Security Research Labs within the agreed-on timeframe and scope as detailed in Table 1. 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 6 may not ensure all future code to be bug free.

2 Motivation and scope

Blockchains evolve in a trustless and decentralized environment, which by its own nature could lead to security issues. Ensuring availability and integrity is a priority for BalanceAI as it aims to create a marketplace for AI (Artificial Intelligence) Agents, including a bridge with the Ethereum ecosystem. As such, a security review of the project should not only highlight the security issues uncovered during the audit, but also bring additional insights from an attacker's perspective, which the BalanceAI team can then integrate into their own threat modeling and development process to enhance the security of the product.

BalanceAI is a blockchain network built on top of Substrate. Like other Substrate-based blockchain networks, the BalanceAI code is written in Rust, a memory safe programming language. Mainly, Substrate-based chains utilize three technologies: a WebAssembly (WASM) based runtime, decentralized communication via libp2p, and a block production engine.

The BalanceAI runtime consists of multiple modules compiled into a WASM Binary Large Object (blob) that is stored on-chain. Nodes execute the runtime code either natively or will execute the on-chain WASM blob.

The in-scope components of the BalanceAI project are reflected in Table 1. The audit covered up to commit **7b86ca0** from BalanceAI's GitHub repository [1].

Repository	Component(s)
balanceai	runtime configuration
	evm-bridge-pallet

Table 1. In-scope BalanceAI components

3 Methodology

We carried out a hybrid strategy utilizing a combination of manual code review, static analysis and dynamic tests (e.g., fuzz testing) to assess the security of the BalanceAI codebase.

While fuzz testing and dynamic tests establish a baseline assurance, the focus of this audit was a manual code review of the BalanceAI codebase to identify logic bugs, design flaws, and best practice deviations. The approach of the review was to trace the intended functionality of the runtime modules in scope and to assess whether an attacker can bypass/misuse/abuse these components or trigger unexpected behavior on the blockchain due to logic bugs or missing checks.

Fuzz testing is a technique to identify issues in code that handles untrusted input, which in BalanceAI's case are extrinsics in the runtime. (Note that the network part is handled by Substrate, which was not in scope for this review, but is built with a strong emphasis on security and where fuzz testing is also used). Fuzz testing works by taking some valid input for a method under test, applying a semi-random mutation to it, and then invoking the method under test again with this semi-valid input. Through repeating this process, fuzz testing can unearth inputs that would cause a crash or other undefined behavior (e.g., integer overflows) in the method under test. The fuzz testing methods written for this assessment utilized the test runtime genesis configuration as well as mocked externalities to execute the fuzz test effectively against the extrinsics in scope.

4 Findings summary

We identified 6 issues - summarized in Table 2 - during our analysis of the runtime modules in scope in the BalanceAI codebase. In summary, 1 high severity, 3 medium severity and 2 info severity issues were found.

As BalanceAI is in early development stages and there is no deployment of the system, 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.

Issue	Severity	Status
Multiple security concerns about the *evm-bridge-pallet*	High	Closed [2]
The fee multiplier can go down to extremely low values	Medium	Closed [3]
The value of *OnSlash* parameter for *pallet_treasury* is empty	Medium	Closed [4]
Incorrect configuration of runtime weights	Medium	Closed [5] [6] [7]
Import of an insecure randomness algorithm	Info	Closed [8]
The Substrate dependencies are not bound to a Substrate version	Info	Closed [9]

Table 2 Issue summary

5 Detailed findings

5.1 Multiple security concerns about *evm-bridge-pallet*

Attack scenario	**Attacker tricks the user into submitting a transaction twice**
Location	pallets/evm-bridge-pallet
Attack impact	User loses funds while trying to send tokens via the bridge.
Status	Closed [2]
Severity	High

The *evm-bridge-pallet* allows users to transfer tokens between Ethereum network and BalanceAI, by burning the tokens on one chain and minting them on the other chain, creating a bridge.

As of now, the pallet is under active development, the current state defining a boilerplate and a proof-of-concept, rather than a fully functional product.

There are multiple concerns about this pallet that should be addressed by BalanceAI team. As the project is in early development stage and critical components are still missing (e.g.: Ethereum smart contracts, guardians' implementation), it is difficult to create an end-to-end attack path.

The following list describes our concerns about the *evm-bridge-pallet*.

1. When a user calls the *teleport* extrinsic, the specified amount is going to be transferred to the pallet's account. If the user does not get enough vouches on the Ethereum side in order to get their tokens minted, there is no mechanism for the user to claim back the tokens on BalanceAI, the user being in the position of losing their tokens.

2. The *teleport_id* is defined by the user while calling the *teleport* extrinsic. There is no mechanism in place in order to protect the user from submitting the same *teleport_id* twice. This could lead to an unknown behaviour, possibly making the user lose some of their tokens. The *teleport_id* should not be user controllable, and ideally it should be a hash.

3. The values of *VouchThreshold* and *DisavowThreshold* should have a minimum cap, defined in the runtime. This prevents a mistaken extrinsic call to set the values too low, for example *VouchThreshold = 0*, allowing an attacker to mint tokens without a guardians vouch.

4. There are no liquidity checks between the BalanceAI and Ethereum side. This means that a user could call the *teleport* function in the Ethereum smart contract, using an amount higher than what *evm-bridge-pallet* has available, making the *withdraw* extrinsic fail on the BalanceAI side. This could lead to the user losing their tokens or force them to wait a long period of time until their requested amount will be available on BalanceAI.

5. The extrinsics *vouch* and *disavow* should check that a guardian has not submitted a vote using the contrary extrinsic already. Precisely, once a guardian submits a *vouch*, they should not be able to submit a *disavow*. The other way around is also applicable, once a guardian submits a *disavow*, they should not be able to submit a *vouch*. The current implementation allows a guardian to submit both *vouch* and *disavow* for a teleport.

6. Consider changing the name of the *Timestamp* storage. Substrate already provides a *Timestamp* type, and this could be confusing for developers as the project gets bigger.

7. The incentives for the guardians are currently unknown. Basically, the guardians have to pay the fees to call the *vouch* and *disavow* extrinsics. There is no mechanism in place that would incentivize them to call the extrinsics and pay the fees.

8. There are no benchmarks available for the pallet. Once the maturity of the code increases and more logic is added, benchmarks should be created accordingly.

Overall, the bridge implementation does not make use of any cryptographic verifiable proof between BalanceAI and Ethereum, making the entire system more prone to attacks.

Creating bridges between different blockchain ecosystems represents a very complex topic which can be approached in different ways, most of the time by leveraging cryptographic proofs. The *evm-bridge-pallet* proposes a basic bridge implementation which can leave opportunities for different attacks, increasing the chances of tokens getting lost or stolen.

Moreover, the adoption of BalanceAI is greatly reduced, as the bridge implementation might not offer enough confidence to potential users.

As the bridges topic is well researched, it is recommended to take a deeper understanding of the available architecture implementations and concepts, and to consider a system architecture redesign, including more advance cryptographic proofs. The economic model is also a critical part of a bridge, extra attention being required for the current implementation of the *evm-bridge-pallet*.

The BalanceAI team addressed point number 2 by using a hash function to generate a teleport identifier [2]. All the other points described in the issue were acknowledged as intended features of the system.

5.2 The fee multiplier can go down to extremely low values

Attack scenario	**An attacker spams the network with transactions**
Location	runtime
Attack impact	Service availability or network slows down.
Status	Closed [3]
Severity	Medium

The *pallet_transaction_payment* defines the parameter *FeeMultiplierUpdate* which controls how the fees are going to change based on network load. The current runtime implementation sets the *FeeMultiplierUpdate* as follow:

```
parameter_types! {
    pub const TargetBlockFullness: Perquintill =
Perquintill::from_percent(25);
    pub AdjustmentVariable: Multiplier =
Multiplier::saturating_from_rational(1, 100_000);
    pub MinimumMultiplier: Multiplier =
Multiplier::saturating_from_rational(1, 1_000_000_000u128);
    pub MaximumMultiplier: Multiplier = Bounded::max_value();
}

impl pallet_transaction_payment::Config for Runtime {
    ...
    type FeeMultiplierUpdate = TargetedFeeAdjustment<
        Self,
        TargetBlockFullness,
        AdjustmentVariable,
        MinimumMultiplier,
        MaximumMultiplier,
    >;
}
```

The value of *MinimumMultiplier* specifies that the fee multiplier can go down to extremely low values (i.e. 1e-9). When the blockchain is launched and starts with a very low transaction volume, the multiplier will exponentially go down and reach that value after some time (in about 6-7 months). Once the multiplier is at that minimum, it will take several months of completely full blocks to get the weight fees back to anything close to the original values.

During most of the time where the fee multiplier is low, transaction weight is basically free and it is therefore possible to DoS all no-tip transactions with relatively low investment, making the blockchain harder to operate.

We suggest setting *MinimumMultiplier* to *SlowAdjustingFeeUpdate*, which is defined in Polkadot and uses a 1/10 value, reducing the recovery time from the minimum value to a somewhat more significant fee value.

The values defined for *SlowAdjustingFeeUpdate* are:

```
pub const TargetBlockFullness: Perquintill =
Perquintill::from_percent(25);
pub AdjustmentVariable: Multiplier =
Multiplier::saturating_from_rational(75, 1000_000);
pub MinimumMultiplier: Multiplier =
Multiplier::saturating_from_rational(1, 10u128);
pub MaximumMultiplier: Multiplier = Bounded::max_value();
```

The remediation was done by setting the parameters of *AdjustmentVariable* and *MinimumMultiplier* to the same values that are being used in the Polkadot runtime [3].

5.3 The value of *OnSlash* parameter for *pallet_treasury* is empty

Attack scenario	**A delegator gets slashes**
Location	runtime
Attack impact	Tokens that are slashed from delegators are not being assigned to any account.
Status	Closed [4]
Severity	Medium

pallet_treasury has a parameter *OnSlash* which handles the unbalanced reduction when slashing a delegator. The current BalanceAI runtime configuration sets this parameter to *()*, the slashing funds not being redirected to any account.

```
impl pallet_treasury::Config for Runtime {
    ...
    type OnSlash = ();
    ...
}
```

Funds that could come from slashing a delegator are not handled, which means they will not be returned to any account.

To mitigate this issue, set the *OnSlash* value to an account. Usually, the *Treasury* account is being set.

This issue was mitigated by setting the *OnSlash* value to *BalanceTreasury* [4].

5.4 Incorrect configuration of runtime weights

Attack scenario	**An attacker spams the network with transactions**
Location	runtime
Attack impact	Service availability or network slows down.
Status	Closed [5] [6] [7]
Severity	Medium

Every extrinsic available in a Substrate project needs to have a weight assigned which is calculated using a benchmarking system. Depending on the runtime configuration for every pallet, the benchmarking result might be different.

BalanceAI relies on weights for their FRAME pallet dependencies that are benchmarked with some external runtimes instead of the actual runtime of the project.

The WeightInfo value is incorrectly set for all the pallets used in the runtime, as it points to the SubstrateWeight defined in the pallet itself.

For example, the weights for pallet-indices are set as:

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

The benchmarks for these pallets were done using other runtimes, such as Polkadot or Kusama.

Furthermore, there are instances of *WeightInfo* being configured to *()* which results in zero-value, for *pallet-fast-unstake*, *pallet-nomination-pools* and *pallet_message_queue*.

Since these weights are not dependent on the runtime configuration, this could lead to underweight extrinsics, allowing for malicious spamming and bloating network storage.

All extrinsics, including Substrate ones, should be benchmarked with the actual runtime configuration by including them in the *define_benchmarks!* [10] block and setting the corresponding *WeightInfo* value.

The BalanceAI team has started to implement weights for all the pallets used by the runtime [5] [6] [7].

5.5 Import of an insecure randomness algorithm

Attack scenario	**An attacker can take advantage of knowing the next randomness**
Location	runtime
Attack impact	If the randomness will be used in the future, depending on the use-case, an incentive advantage could be created for specific users.

Status	Closed [8]
Severity	Info

The source of randomness in the runtime is configured to use the *pallet_insecure_randomness_collective_flip*, implemented in Substrate.

The output of collective flip is highly predictable as it is based on the last 81 blocks and should not be used as a true source of randomness.

The import is defined as:

```
construct_runtime!(
    pub struct Runtime
    {
        ...
        RandomnessCollectiveFlip:
pallet_insecure_randomness_collective_flip,
        ...
    }
);
```

As of now, the randomness is not being used by any pallet. However, maintaining that insecure import could lead to unsafe usage in the future if any need for randomness arises, giving the possibility to a malicious collator to influence the randomness.

As a mitigation, remove the usage of *pallet_insecure_randomness_collective_flip* and consider using the BABE pallet to generate randomness, as described in the Substrate documentation [11]. Check out the Kusama runtime configuration for an example [12].

The *pallet_insecure_randomness_collective_flip* was removed from the runtime, thus the issue is mitigated [8].

5.6 The Substrate dependencies are not bound to a Substrate version

Attack scenario	**Dependencies follow different Substrate versions, an attacker being able to spot the usage of deprecated and vulnerable ones**
Location	runtime
Attack impact	Depending on the dependency and vulnerability, attacker might gain financial incentives or make the blockchain harder to operate.
Status	Closed [9]
Severity	Info

BalanceAI imports the Substrate pallets by specifying their version available on *crates.io*, rather than specifying a git repository and version tag / branch.

For example, *frame-support* and *pallet-im-online* are imported as:

```
frame-support = { version = "29.0.0", default-features = false }
pallet-im-online = { version = "28.0.0", default-features = false }
```

Substrate is a fast-evolving framework, containing many dependencies, new versions are being released once in a few weeks. While using the framework, it is a best practice to be specific about the version of dependencies that your project imports, and not rely on the versioning from *crates.io*.

For example, a proper way to import *frame-support* or *pallet-im-online* is:

```
frame-support = { git = "https://github.com/paritytech/polkadot-
sdk.git", tag = "polkadot-v1.7.2", default-features = false }
pallet-im-online = { git = "https://github.com/paritytech/polkadot-
sdk.git", tag = "polkadot-v1.7.2", default-features = false }
```

Using this methodology, you make sure that all the dependencies are pointing to the same version, ensuring their compatibility. Moreover, this helps the developers to know what versions they should use for any new dependencies that they might want to add to the project.

The mitigation for this issue is to specify a git repository (usually https://github.com/paritytech/polkadot-sdk.git) and a *tag* or *branch* while importing Substrate dependencies.

All the dependencies were set to *polkadot-v1.7.2*, mitigating this issue [9].

6 Evolution suggestions

The overall impression of the auditors was that BalanceAI as a product is still in the very early stages of development. The developers are aware of the security implications and are proactively looking to minimize the attack surface of BalanceAI.

To ensure that BalanceAI is secure against unknown or yet undiscovered threats, we recommend considering the evolution suggestions and best practices described in this section.

6.1 Further recommended best practices

Improve documentation clarity. There is currently no documentation available for developers or users. We suggest diligently starting to create the documentation, including extensive details about the *evm-bridge-pallet*, how the architecture will work on both BalanceAI and Ethereum and how the guardian nodes are developed. Furthermore, you can request multiple audit reviews for your bridge architecture to minimize any potential critical change at a later stage.

The economic model of BalanceAI is also more unique, having a halving based emissions system, defined via the *HalvingBasedRewardPayout.* There are also two additional account addresses defined via *BalanceTreasury* which will get a specific amount of rewards over the time. As these characteristics influence the economics of the system, we recommend creating a transparent documentation aiming to describe the mechanics and reasons behind this model for BalanceAI's users.

Engage in an extensive bridge audit. As bridges are one complex topic in the blockchain ecosystem, BalanceAI should request a full bridge audit once the module is entirely implemented. The current implementation of the bridge misses critical parts, like guardian nodes, Ethereum smart contracts, which are going to be developed in the future.

Create extensive unit and integration tests. As one of the BalanceAI's goal is to create a bridge to Ethereum, the developers should make sure that unit and integration tests are covering as many edge cases as possible. The project should also include tests against the bridge, while having both BalanceAI and Ethereum nodes running.

Engage in an economic audit. Although SRLabs has some knowledge of economic attacks, our primary goal during engagements is to find logic vulnerabilities through code assurance. Therefore, an economic audit for the *evm-bridge-pallet* is recommended to ensure the safety of BalanceAI's platform and users.

Regular code review and continuous fuzz testing. Regular code reviews are recommended to avoid introducing new logic or arithmetic bugs, while continuous fuzz testing can identify potential vulnerabilities early in the development process. Ideally, BalanceAI should continuously fuzz their code on each release.

Regular updates. New releases of Substrate may contain fixes for critical security issues. Since BalanceAI is reliant on Substrate, safely updating to the latest version as soon as possible whenever a new release is available is recommended.

7 Bibliography

[1] [Online]. Available: https://github.com/balanceainetwork/balanceai.

[2] [Online]. Available: https://github.com/balanceainetwork/balanceai/commit/9f1d8ff1a498c92e0498de2b325294eb0662d019.

[3] [Online]. Available: https://github.com/balanceainetwork/balanceai/commit/a4a6bb7ecdc58a94e0fce3b62b47e200e3833c24.

[4] [Online]. Available: https://github.com/balanceainetwork/balanceai/commit/23a924ce5c28fe1e70e9e635d6f911db5acd02d4.

[5] [Online]. Available: https://github.com/balanceainetwork/balanceai/commit/eb27165b2c02945dc323c2df333bda620198cc76.

[6] [Online]. Available: https://github.com/balanceainetwork/balanceai/commit/b6460a6554759ff4a32eecf31cd589f821d10b31.

[7] [Online]. Available: https://github.com/balanceainetwork/balanceai/commit/e26c4ee65af41e2bf09b01ec642117b566a810c8.

[8] [Online]. Available: https://github.com/balanceainetwork/balanceai/commit/db0875add262976de3eccc13ff5b0744a302dfe5.

[9] [Online]. Available: https://github.com/balanceainetwork/balanceai/commit/9dcc84ecad23fc980c2ef4f1b04c327e1daeb0c8.

[10] [Online]. Available: https://docs.substrate.io/test/benchmark/#adding-benchmarks.

[11] [Online]. Available: https://docs.substrate.io/build/randomness/#generate-and-consume-randomness.

[12] [Online]. Available: https://github.com/polkadot-fellows/runtimes/blob/30e0dbfdcb78722ed61325c0ebf1efdcdb6033ba/relay/kusama/src/lib.rs#L996.