SECURITY AUDIT REPORT

Ta-da user-funds (4)
smart contract

by ARDA

on January 15, 2025

Table of Content

Disclaimer

The report makes no statements or warranties, either expressed or implied, regarding the security of the code, the information herein or its usage. It also cannot be considered as a sufficient assessment regarding the utility, safety and bugfree status of the code, or any other statements.

This report does not constitute legal or investment advice. It is for informational purposes only and is provided on an "as-is" basis. You acknowledge that any use of this report and the information contained herein is at your own risk. The authors of this report shall not be liable to you or any third parties for any acts or omissions undertaken by you or any third parties based on the information contained herein.

Terminology

Inherent risk: A risk for users that comes from a behavior inherent to the smart contract design.

Inherent risks only represent the risks inherent to the smart contract design, which are a subset of all the possible risks. **No inherent risk doesn't mean no risk.** It only means that no risk inherent to the smart contract design has been identified. Other kind of risks could still be present. For example, the issues not fixed incur risks for the users, or the smart contracts deployed as upgradeable also incur risks for the users.

Issue: A behavior unexpected by the users or by the project, or a practice that increases the chances of unexpected behaviors to appear.

Critical issue: An issue intolerable for the users or the project, that must be addressed.

Major issue: An issue undesirable for the users or the project, that we strongly recommend to address.

Medium issue: An issue uncomfortable for the users or the project, that we recommend to address.

Minor issue: An issue imperceptible for the users or the project, that we advise to address for the overall project security.

Audit Summary

Scope of initial audit

- **Repository:** https://gitlab.com/ta-da2/smart-contracts/sc-tada
- **Commit:** `b1566255d701cb4b629c4bfd8699818485db78ec`
- **Path to Smart contract:** `./user_funds/`

Scope of final audit

- **Repository:** https://gitlab.com/ta-da2/smart-contracts/sc-tada
- **Commit:** `c7e1355a21f066f09279b873a9f31e24507b384b`
- **Path to Smart contract:** `./user_funds/`

Report objectives

1. Reporting all **inherent risks** of the smart contract.
2. Reporting all **issues** in the smart contract **code**.
3. Reporting all **issues** in the smart contract **test**.
4. Reporting all **issues** in the **other** parts of the smart contract.
5. Proposing **recommendations** to address all issues reported.

3 inherent risks in the final commit

2 issues in the final commit

10 issues reported from the initial commit and 2 remaining in the final commit:

Severity	Reported			Remaining		
	Code	Test	Other	Code	Test	Other
Critical	0	0	0	0	0	0
Major	3	0	0	1	0	0
Medium	5	0	0	1	0	0
Minor	2	0	0	0	0	0

Inherent Risks

R1: Users can be slashed although they correctly completed their tasks.

This is because when a user gives his consent to be slashed in case he incorrectly complete a given task, even if he then correctly completes the task, the consent would still allow addresses whitelisted by Ta-da to slash him. Thus the user might be erroneously slashed e.g. if the whitelisted address is corrupted or has made a mistake.

R2: Users might be debited for purchases but not get the expected benefits in return.

This is because when a user is debited in the smart contract for purchases, he must trust that the associated benefit, which is handled outside of the smart contract, will be granted to him in return.

Example: if a user purchases a potion on the TADA application, which should help him getting a higher ranking among users and thereby increase his rewards, then there is neither guarantee that he would receive the potion, neither that he would get a higher ranking thanks to the potion, nor that he would earn more rewards thanks to the higher ranking.

R3: Users requesting to withdraw using signatures in the format of the TON Telegram wallet might never receive their TADA.

This is because when a user requests to withdraw with a signature in the format of the TON Telegram wallet, the user's TADA are withdrawn to a Ta-Da whitelisted address, and there is trust that this address will then bridge the TADA to the TON blockchain and forward them to the user. In particular, the user might not receive his TADA if the whitelisted address is inactive, compromised or unable to bridge the TADA from MultiversX to TON.

Code Issues & Recommendations

Since the smart contract code is not open-source, only the remaining issues are published.

C3: User can be debited any amount of TADA without agreement by the slashing mechanism

Severity: Major **Status:** Won't fix

Location

`user_funds/src/lib.rs`

Description

Current behavior: A user can be debited any amount of TADA at any time even if he did not agree with it, through the slashing mechanism. Indeed, at any time, the addresses whitelisted by Ta-da can call `slash` to debit the TADA of a user which are held in the smart contract.

In particular, a user could be slashed even if he did not misbehave while performing tasks. Consequently, if a whitelisted address is corrupted or makes a mistake, it could make some users lose funds.

Expected behavior: It should be possible to slash a user only if the user gave his consent for it, which he would do if he performed tasks knowing that misbehaving would entail slashing.

In the recommendation, we describe how to verify that a user gave his consent using signatures.

Worst consequence: A whitelisted address is corrupted and steals the funds of all users.

Example: A whitelisted address is corrupted and slashes all users. The slashed TADA are spread between 3 addresses as follows: 34% go to `prize_pool_address`, 33% go to `burn_address`, and 33% go to `stake_address`.

However, the attacker also got access to the private keys of `prize_pool_address`, thus he can further withdraw all the funds from it, and has effectively stolen 34% of all users' funds.

Recommendation

In the `slash` endpoint, we recommend adding some arguments proving that the user previously agreed to be debited, exactly as recommended for the `purchase` endpoint in another issue of a previous audit report.

Alternatively, we describe an alleviation that strongly mitigates the issue, and whose advantage over the solution above is to not ask the user to sign messages each time he performs a task. At a high-level, the idea is that when a user is slashed, his funds are not withdrawn immediately, rather they enter a cooldown period `COOLDOWN_BEFORE_WITHDRAW_SLASHED_FUNDS` of 10 days after which they can be withdrawn, allowing the owner to intervene and cancel the slashing if he realizes that users are mistakenly slashed by the whitelisted addresses.

More precisely, in `slash`, instead of withdrawing the slashed funds immediately, we insert the tuple `(user, amount, current_epoch + COOLDOWN_BEFORE_WITHDRAW_SLASHED_FUNDS)` at the end of a `LinkedListMapper` `user_slashed_funds`.

Furthermore, there is a new endpoint `withdraw_slashed_funds` that iterates over `user_slashed_funds` from the beginning (which correspond to the oldest epochs), and for each tuple `(user, amount, epoch)`:

- If there is less than 30M gas left, the endpoint stops and returns.
- If `epoch > current_epoch`, the endpoint stops and returns.
- Else, when `epoch <= current_epoch`, then funds are withdrawn and the tuple is taken out of `user_slashed_funds`.

Note that, if one day a problem is detected with slashing, the owner can pause the call to `withdraw_slashed_funds` by removing the backend addresses making errors from the whitelist. It would then be relatively simple to determine how to return the slashed funds to the users, as they are still in the smart contract and the exact amounts to send back are in `user_slashed_funds`.

Moreover, note that this approach is a strong mitigation of the issue, but does not fully resolve it since there remains the possibility that users are slashed although they should not, and that the owner does not take action or is not available to prevent the slashed funds from being withdrawn.

Finally, in case the approaches presented above do not fit the needs of the project, we suggest reaching out to the auditor in order to discuss alternative solutions.

Resolution notes

The issue has not been fixed.

C7: The authorization for debiting a user through the slashing mechanism never expires

Severity: Medium **Status:** Won't fix

Description

Current behavior: The addresses whitelisted by Ta-da are able to debit a user account in the `slash` endpoint without any time constraint. Therefore, a user could be slashed for a task he performed very long ago.

However, users would likely not expect to be debited for some tasks they performed a long time ago and that they may have completely forgotten about. This could in particular disturb their financial plans.

Another consequence is that, if a whitelisted address is corrupted, then the attacker would be able to slash all users who completed tasks in the past, even if they did not misbehave.

Expected behavior: A user should be debited within a reasonable delay, e.g. 10 days, after he misbehaved while completing a task. After this delay, the authorization for debiting the user should expire, protecting him from being debited for outdated tasks. This would moreover let the user better anticipate when debits could happen, and to plan his expenses and financial plans accordingly.

Worst consequence: A whitelisted address is corrupted and slashes all users, including all users who were only active a long time ago and behaved correctly.

Recommendation

We suggest following the recommendation of User can be debited any amount of TADA without agreement by the slashing mechanism, which resolves this issue by letting users specify the maximum period during which they accept to be debited, and further enforcing that this period is no bigger than 10 days.

Resolution notes

The issue has not been fixed.