Security Assessment for xBacked

Report by Ulam Labs

Staking Smart Contract

Findings and Recommendations Report Presented to:

xBacked Team

May 13, 2022 Version: 0.1

Presented by:

Ulam Labs

Grabiszynska 163/502,

53-332 Wroclaw, POLAND

Executive Summary

Overview

xBacked engaged Ulam Labs to perform a Security Assessment for xBacked smart contracts.

The assessment was conducted remotely by the Ulam Labs Security Team. Testing took place on May 16 - June 20, 2022, and focused on the following objectives:

- Provide the customer with an assessment of their overall security posture and any risks discovered within the environment during the engagement.
- Provide a professional opinion on the maturity, adequacy, and efficiency of the security measures.
- Identify potential issues and include improvement recommendations based on the result of our tests.
- Confirmation of remediation for all reported issues.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the Ulam Labs Security Teams took to identify and validate each issue, and any applicable recommendations for remediation.

Scope

The audit has been conducted on the commit **0a610a0111d66c91026bd07966f5185b5e095a51** of xBacked private GitHub Repository. All the required fixes have been delivered (see Table 1 for details). However, commits are introducing a lot of new code, so Ulam Labs Security Team cannot guarantee that any new problem has not been introduced.

commit	issue ID
a5471c0	XBS-1, XBS-3, XBS-5, XBS-7, XBS-8, XBS-11
46be0a3	XBS-2, XBS-4
a911a21	XBS-12
37d9ec6	XBS-6

Table 1: Fixes delivery.

Files included in the audit

```
xbacked-contracts
  └──── src
     ├──── master_staking.rsh
```

Key findings

During the Security Assessment, following findings have been discovered:

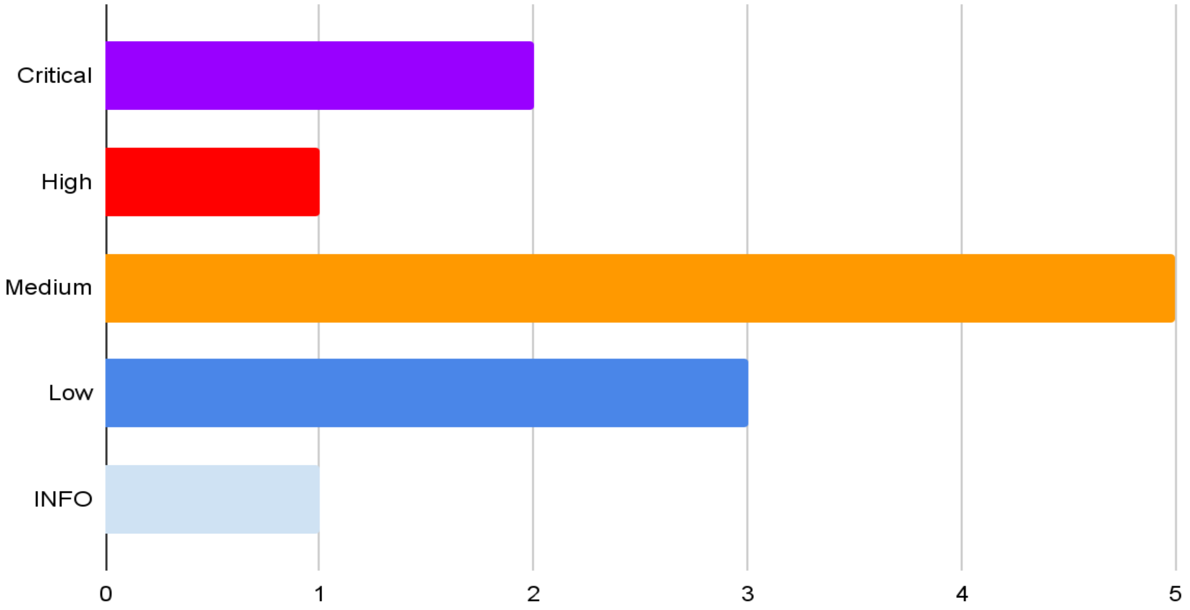

Chart 1: Findings by severity.

- 2 findings with a CRITICAL severity rating,
- 1 findings with a HIGH severity rating,
- 5 findings with a MEDIUM severity rating,
- 3 findings with a LOW severity rating.
- 1 findings with a INFO severity rating

All findings have been acknowledged and fixed by the xBacked team.

Disclaimer

This report does not constitute legal or investment advice. The preparers of this report present it as an informational exercise documenting the due diligence involved in the secure development of the target contract only, and make no material claims or guarantees concerning the contract's operation post-deployment. The preparers of this report assume no liability for any and all potential consequences of the deployment or use of the contract.

Smart contracts are still a nascent software arena, and their deployment and public offering carries substantial risk. This report makes no claims that its analysis is fully comprehensive, and recommends always seeking multiple opinions and audits.

This report is also not comprehensive in scope, excluding a number of components critical to the correct operation of this system.

The possibility of human error in the manual review process is very real, and we recommend seeking multiple independent opinions on any claims which impact a large quantity of funds.

Technical analysis & findings

Unacceptable precision loss while calculating rewards

Finding ID: **XBS-1**
Contract: master_staking@0a610a0
Severity: Critical
Status: Fixed

Description

Let's assume that **rewardPerToken** is updated in each block and the user deposits funds as first and withdraws them as last. This is the worst case for the user, who wants to minimize precision loss. Rewards for users per token are calculated from formula:

$$rewardPerToken(t) \;=\; \sum_{i=1}^{t} \frac{calculatedRewardRate_i * SCALE\_FACTOR}{totalDeposit_i}$$

Integer division is causing the error to be up to one.

$$x \;=\; a * y + b$$

$$b \;<\; y$$

$$\frac{x}{y} = a + \frac{b}{y}$$

$$e \;=\frac{b}{y} \approx 1$$

If division introduces an error, the sum of divisions accumulates those errors.

$$rewardPerToken(t) \;=\; \sum_{i=1}^{t} rewardPerTokenDelta_i \;+ e_i$$

$$rewardPerTokenLost \;=\; e_i * t \approx t$$

At the end, rewards are calculated from formula:

$$t_u \leq t$$

$$rewards(t, t_u) = \frac{rewardPerToken(t) - rewardPerToken(t_u)}{SCALE\_FACTOR} * amountDeposited$$

As proved above, division is introducing an error. In this case, error is multiplied by the amount deposited, which makes it significant.

$$rewardsLost = e_i * amountDeposited \approx amountDeposited$$

If “multiply first” rule is applied:

$$rewards(t, t_u) = \frac{(rewardsPerToken(t) - rewardPerToken(t_{u)}) * amountDeposited}{SCALE\_FACTOR}$$

$$rewardsLost = \frac{t * amountDeposited}{SCALE\_FACTOR} + e_i \approx \frac{t * amountDeposited}{SCALE\_FACTOR} + 1$$

The error is divided by scale factor. This is very important to choose the biggest possible scale factor to get the best precision. However choosing too big value can lead to integer overflow. The worst case, when total deposit is one, then following condition must be met:

$$maxTimeElapsed * maxRewardRate * SCALE\_FACTOR < 2^{64}$$

If contract is not deprecated for one year, with

$$SCALE\_FACTOR = 10^{6}$$

$$maxRewardRate = 10^{6}$$

$$maxTimeElapsed = \frac{365*24*60*60}{blockTime}$$

$$\frac{365*24*60*60*10^{6}*10^{6}}{blockTime} \approx 2^{63} < 2^{64}$$

Integer overflow is not possible.

Impact

Precision loss caused by division in line **473** is unacceptable.

Solution

Every time multiplication and division is required within the same consensus step, function **muldiv** should be used.

Status

Addressed by the xBacked team. The fix was applied to the source code with commit **a5471c0** and reviewed by Ulam Labs Security Team.

Unexpectedly low scale factor

Finding ID: **XBS-2**
Contract: master_staking@0a610a0
Severity: Critical
Status: Fixed

Description

As described in **XBS-1**, it is very important to use as big a scale factor as possible. In line **16**, scale factor is defined as **1e6**, but Reach evaluates it as **246**.

Reach is using **Text.ParserCombinators.Parsec.Token.numberValue** function to calculate decimal number value from string. Each character is transformed to integer value using **Char.digitToInt**. This function is checking if a character is a valid hex digit, that's why **e** is evaluated as **14.** Final scale factor is calculated from expression:

$$SCALE\_FACTOR \ = \ 1 * 100 + 0xe * 10 + 6 = 100 \ + \ 140 \ + \ 6 \ = 246$$

Impact

Scale factor is expected to be **1000000**, but it is **246**. The calculation precision is broken.

Solution

Problem was reported to the Reach team. In the meantime as a workaround **1e6** should be replaced with **1000000**.

Status

Addressed by the xBacked team. The fix was applied to the source code with commit **46be0a3** and reviewed by Ulam Labs Security Team.

Integer overflow while withdrawing more rewards than possible

Finding ID: **XBS-3**
Contract: master_staking@0a610a0
Severity: Low
Status: Fixed

Description

The stacking contract has three balances. One for network token, the other two for staking and reward ASA. Balance for staking ASA is shared between total deposit and remaining rewards. It is extremely important to update a proper global variable, when staking funds or rewards are deposited or withdrawn and never use balance to check if some API calls are possible.

Impact

In the line **930** asset withdrawal is validated. Validation is correct for most of the assets, but not for staking ASA, as described above. User funds could be used as rewards if remaining rewards are not updated in line **949**. Any try to withdraw more than allowed will cause integer overflow and panic. Because of panic, staked funds are safe, but such an approach is not recommended since the end user will get a cryptic error message if a problem occurs.

Solution

Use the remaining rewards array instead of reward balances.

Status

Addressed by the xBacked team. The fix was applied to the source code with commit **a5471c0** and reviewed by Ulam Labs Security Team.

User rewards are lost if contract has no remaining rewards

Finding ID: **XBS-4**
Contract: master_staking@a5471c0
Severity: High
Status: Fixed

Description

When user wants to withdraw the rewards, rewards amount stored in the local state is updated using formula:

$$userRewards = userRewards - rewardsToWithdraw$$

It is very important to make sure that the difference between user reward balance before and after local state update is transferred to the user or contract fails.

Initially, in version **0a610a0**, some rewards were lost, because of division error. Total number of rewards sent to user is calculated using formula:

$$actualRewards = \sum_{i=0}^{2} \frac{rewardRati_i * rewardsToWithDraw}{TOTAL\_PERCENTAGE}$$

$$rewardsLost = rewardsToWithraw - totalRewards \approx 3$$

As shown on **XBS-1**, each division is introducing an error equal to one, so up to three tokens could be lost. It is not too much, especially when rewards are using six decimal places.

However in version **a5471c0**, while correcting **XBS-3**, a new problem has been introduced:

$$actualRewards = \sum_{i=0}^{2} min(\frac{rewardRati_i * rewardsToWithDraw}{TOTAL\_PERCENTAGE}, remainingRewards_i)$$

$$(\sum_{i=0}^{2} remainingRewards_i) == 0 \Rightarrow actualRewards = 0$$

$$rewardsLost \approx rewardsToWithdraw$$

All the rewards can be lost.

Impact

If a user decides to withdraw the rewards, when there are no remaining rewards in the contract, all the earned rewards are lost.

Solution

To mitigate all the problems with lost rewards, the way how reward balance is updated should be changed in the code at line **967** using formula:

$$userRewards = userRewards - actualRewards$$

Status

Addressed by the xBacked team. The fix was applied to the source code with commit **46be0a3** and reviewed by Ulam Labs Security Team.

Saturation arithmetics is not working

Finding ID: **XBS-5**
Contract: master_staking@0a610a0
Severity: Medium
Status: Fixed

Description

There are some calculations, which require saturation addition and multiplication. Intuitive version of saturation addition is:

$$sadd_0(a, b) = a + b \geq UInt.max\, ?\, UInt.max : a + b$$

There are a few problems here. If integer overflow occurs, **AVM** will panic. Let’s improve the function above.

$$a + b \leq UInt.max$$

$a \leq UInt.max - b$

$sadd_1(a,\ b) \ = a \leq UInt.max - b\,?\,a + b : UInt.max$

It looks good, but **Reach** will produce the **Teal** code, which evaluates all the expressions, so this version is not going to work either.

$sadd(a,\ b) \ = \ a + b\ if\ a \leq UInt.max - b\ else\ UInt.max$

Saturated multiplication can be implemented similarly.

$a\ *\ b \leq UInt.max$

$a \leq UInt.max\ /\ b$

$smul(a,\ b) \ = \ 0\ if\ !\,b\ else\ a\ *\ b\ if\ a \leq UInt.max\ /\ b\ else\ UInt.max$

Impact

Calculations from lines **477** and **504** are using invalid saturation additions and will not work as expected.

Solution

Implement presented formulas in the contract code.

Status

Addressed by the xBacked team. The fix was applied to the source code with commit **a5471c0** and reviewed by Ulam Labs Security Team.

Lack of saturation arithmetic can cause permanent fund freeze

Finding ID: **XBS-6**
Contract: master_staking@a5471c0
Severity: Medium
Status: Fixed

Description

Algorand virtual machine panics, when integer overflow occurs. In most of the cases it is recommended to check arguments before calling the arithmetic function. One of such exceptions, when checking is not necessary, is summing tokens. No more tokens can be minted than **64bit** integers can support, so it is impossible to cause integer overflow while adding valid balances.

$$a < 2^{64} \land a = b + c \Rightarrow b + c < 2^{64}$$

It is also common to see that elapsed time in seconds is multiplied by some variables.

$$timeElapsed * X < 2^{64} \land timeElapsed = 1\,year \Rightarrow X < 2^{40} \approx 10^{12}$$

It is a problematic case, because if the contract fails, because of integer overflow caused by time elapsed, the problem is permanent as elapsed time will never be smaller.

The integer overflow described in **XBS-1** while calculating reward per token also is not yet addressed.

Impact

Operations from lines **761, 807, 823** are safe, because those are operations on tokens.

However, it is recommended to use saturation arithmetic at line **508** (multiplying by time elapsed and reward per token calculation). The risk that the problem occurs is not so high, but if it happens, all the funds stored in contract are lost forever.

Lack of saturation arithmetic at line **508** is very dangerous from the user point of view. As proved in **XBS-1**, if the reward rate is too high, calculating reward per token will always panic.

Leaving the possibility to lock all the user funds forever is dangerous, because the admin can accidentally set too big reward rate, the admin private key can be stolen or the admin can just become malicious.

Solution

Implement formulas described in **XBS-5** in the contract code at line **508**.

Status

Addressed by the xBacked team. The fix was applied to the source code with commit **37d9ec6** and reviewed by Ulam Labs Security Team.

It is impossible to set reward rate as zero after deployment

Finding ID: **XBS-7**
Contract: master_staking@0a610a0
Severity: Medium
Status: Fixed

Description

Reward rate is a parameter, which determines how many rewards are available to claim per block. If reward rate is zero, remaining rewards are distributed evenly amongst remaining blocks using formula:

$$deprecatedAt > lastRewardBlock \Rightarrow$$

$$rewardRate = \frac{sum(remainingRewards)}{deprecatedAt - lastRewardBlock}$$

If the contract is fully deprecated, the calculated reward rate is zero, which is a bit problematic, because if reward per token is not updated during the depreciation period, no rewards are distributed for this period.

However it is not a problem, because validations from line **739**, there is no way to set the reward rate as zero.

Impact

Specified functionality is unreachable.

Solution

Reward rate provided by admin should be used as provided without validation.

Check from line **104** is not necessary.

Status

Addressed by the xBacked team. The fix was applied to the source code with commit **a5471c0** and reviewed by Ulam Labs Security Team.

Rewards not distributed correctly during deprecation period

Finding ID: **XBS-8**
Contract: master_staking@0a610a0
Severity: Medium
Status: Fixed

Description

As described in **XBS-7**, if the reward rate is zero, the final reward rate is calculated from the current state. It is important not to use raw reward rate anywhere, because zero value has a special meaning. The other problem is the calculated reward rate after the depreciation period. It is always zero, but if reward per token is not updated at the last block, there are some undistributed rewards.

Impact

In the line 916, raw reward rate is used causing problems described above.

Solution

Always use calculated reward rate.

Status

Addressed by the xBacked team. The fix was applied to the source code with commit **a5471c0** and reviewed by Ulam Labs Security Team.

Contract cannot be closed if user clears local state

Finding ID: **XBS-9**

Contract: master_staking@0a610a0
Severity: Medium
Status: Acknowledged

Description

In Algorand it is possible to clear the user's local state, even if the clear program fails. **Reach** currently is not handling clear state calls, so total deposit may never reach zero, keeping the contract alive forever.

Impact

All the untracked funds and remaining rewards are locked forever. There is no easy way to solve that problem, so it’s better just to assume that once paid, all the untracked funds, fees and remaining rewards are unreachable for the admin.

Solution

Wait for reach to introduce clear state handling. Creating a custom handler in teal is dangerous and not recommended.

Status

The xBacked team is aware of the problem, but as users' funds are not in danger, correction is not planned.

Reward ratios precision is too low

Finding ID: **XBS-10**
Contract: master_staking@0a610a0
Severity: Info
Status: Acknowledged

Description

Reward ratios are used to distribute rewards by ratios provided by admin. Sum of all array elements should be the total percentage.

$$TOTAL\_PERCENTAGE = 100$$

Pending rewards are calculated using formula

$$pendingReward_i = muldiv(rewardRatio_i, rewards, TOTAL\_PERCENTAGE)$$

Impact

What if staking ASA is **1000** times more worthy than ALGOS and we want to distribute them with **1:1** ratio? It is impossible with a total percentage as **100.** Ratios are used just once, so the total percentage can be any **64** bit integer.

Solution

Increase total percentage at line **13** and add support in frontend for bigger ratios.

Status

The xBacked team is aware of the problem, but total percentage as **100** is enough to handle planned business use cases.

Updating reward ratios possible if contract is deprecated

Finding ID: **XBS-11**
Contract: master_staking@0a610a0
Severity: Low
Status: Fixed

Description

Updating reward ratios is forbidden using generated frontend, but contract allows it.

Impact

Validation from line **705**, not present at line **711**.

Solution

Adjust contract checks with frontend checks.

Status

Addressed by the xBacked team. The fix was applied to the source code with commit **a5471c0** and reviewed by Ulam Labs Security Team.

Initial parameters not validated during deployment

Finding ID: **XBS-12**
Contract: master_staking@0a610a0
Severity: Low
Status: Fixed??

Description

Checks present in the frontend code during deployments steps are not in generated contract code. The parameters needing validation on backed:

- Reward ratios.
- Deprecated time.
- Reward rate.

Impact

Admin is now allowed to set mentioned parameters to any value, breaking contract logic.

Solution

Adjust contract checks with frontend checks. Duplicate validations from line **242**, **250**, **253** and put them before making a commitment from line **294**.

Status

Addressed by the xBacked team. The fix was applied to the source code with commit **a911a21** and reviewed by Ulam Labs Security Team.

Additional Recommendation

Use multisig wallet for storing admin key

The account managing contract should be a multisig wallet controlled by xBacked DAO. This can derisk scenarios when a centralized wallet is hacked, private key is leaked or key holder is coerced to hand over the key.

Status

Acknowledged by xBacked team.

Other

Severity classification

We have adopted a severity classification inspired by the **Immunefi Vulnerability Severity Classification System - v2**. It can be found here.