Quantstamp Security Assessment Certificate

August 29th 2020 — Quantstamp Verified

PerlinXRewards.sol

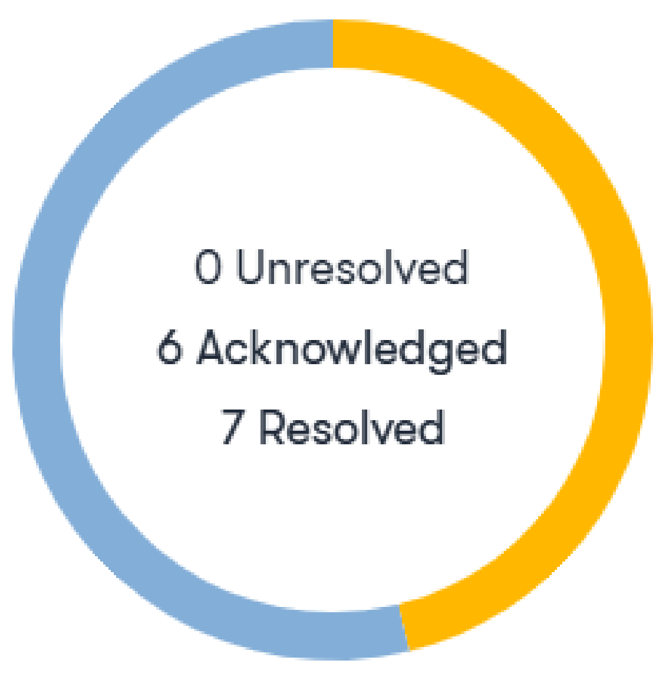
This smart contract audit was prepared by Quantstamp, the protocol for securing smart contracts.

Executive Summary

Type	Audit
Auditors	Poming Lee, Research Engineer Jan Gorzny, Blockchain Researcher Leonardo Passos, Senior Research Engineer
Timeline	2020-07-31 through 2020-08-07
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	PerlinX Rewards Contract Spec - documentation
Source Code	

Repository	Commit
perlinx-contracts	9c62c6c
perlinx-contracts	3c73f50

Total Issues	13	(7 Resolved)
High Risk Issues	2	(2 Resolved)
Medium Risk Issues	1	(0 Resolved)
Low Risk Issues	4	(3 Resolved)
Informational Risk Issues	3	(1 Resolved)
Undetermined Risk Issues	3	(1 Resolved)



High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
? Undetermined	The impact of the issue is uncertain.

Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

During auditing, we found 13 potential issues of various levels of severity: 2 high-severity, 1 medium-severity, 4 low-severity issues, 3 undetermined-severity issues, as well as 3 informational-level findings. The code looks well-structured and concise. We made 21 best practices recommendations.
We highly recommend addressing the findings before going live.
Disclaimer: Please be aware that Quantstamp was requested and had audited a single file: `PerlinXRewards.sol`; the whole system was not audited.
** 2020-08-07 update ** The Perlin team has received and taken care of all the findings and the best practice suggestions.

ID	Description	Severity	Status
QSP-1	A user can increase their shares arbitrarily	High	Fixed
QSP-2	Missing balance check after transfer of ERC20 token	High	Fixed
QSP-3	No minimum quorum for privilege operations	Medium	Acknowledged
QSP-4	Contract allows incorrect combinations of pool/synth/emp addresses	Low	Acknowledged
QSP-5	Modifier `flashSafe()` cannot avoid flash loan and reentrancy	Low	Fixed
QSP-6	Incorrect weight precision	Low	Fixed
QSP-7	`updateRewards` may run out of gas	Low	Fixed
QSP-8	Unlocked Pragma	Informational	Fixed
QSP-9	Clone-and-Own	Informational	Acknowledged
QSP-10	Missing input checks	Informational	Acknowledged
QSP-11	Should zero out a member's rewards for an era after it is claimed	Undetermined	Acknowledged
QSP-12	Potential division by zero	Undetermined	Acknowledged
QSP-13	`snapshotPoolsInEra` can be called on past eras	Undetermined	Fixed

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Mythril 0.22.8
- Slither v0.6.6

Steps taken to run the tools:

1. Installed the Mythril tool from Pypi: `pip3 install mythril`
2. Ran the Mythril tool on each contract: `myth analyze FlattenedContract.sol`
3. Installed the Slither tool: `pip install slither-analyzer`
4. Run Slither from the project directory: `slither .`

Findings

QSP-1 A user can increase their shares arbitrarily

Severity: High Risk

Status: Fixed

Description: L244 accumulates based on the input amount. In addition, the function `registerAllClaims` can be called by anyone at any time. So one can call the function registerAllClaims multiple times to increase its ratio of memberClaimInEra/totalClaimsInEra.

Recommendation: Assign the passed in amount to mapMemberEraPool_Claim[member][currentEra][pool] instead of accumulating it.

QSP-2 Missing balance check after transfer of ERC20 token

Severity: *High Risk*

Status: Fixed

Description: The transfer of ERC20 will not always be successful. Hence, the success of transferFrom/transfer on L200, 225, L234, L265 should be checked.

Recommendation: It is recommended to check the balance before and after the transfer. Could also put the transfer/transferFrom into a require statement

QSP-3 No minimum quorum for privilege operations

Severity: *Medium Risk*

Status: Acknowledged

Description: The contract has multiple admins. However, one admin can change the contract parameters without the consent of others. In this case, it is preferred to introduce a minimum quorum between admins instead of having a single admin doing a change.
** 2020-08-07 update ** The Perlin team stated that they have added treasury address for fund-sensitive actions.

Recommendation: Change admin settings such that any change to the contract requires a minimum quorum amongst registered admins.

QSP-4 Contract allows incorrect combinations of pool/synth/emp addresses

Severity: *Low Risk*

Status: Acknowledged

Description: Currently, listSynth does not check if the input triple <pool, synth, emp> is valid; hence, it is possible to invoke listSynth with a pool address that has no relation to the given synth address, which in turn, has no relation to the given emp address. A similar issue occurs for listPool.
** 2020-08-05 update ** The Perlin team stated that: “A balance of flexibility required, since these are just lookups and future AMMs might be used where it is unknown how to verify. It is not critical if they are listed wrong. The WEB-UI will show errors and alert admin if they incorrectly added. It is then possible to overwrite by just listing again”.

Recommendation: If it is possible to assert whether all three addresses are related, please do so; if not (e.g., information is off-chain), perhaps rely on an oracle that automatically extracts this relation somewhere and feed it to the contract.

QSP-5 Modifier `flashSafe()` cannot avoid flash loan and reentrancy

Severity: *Low Risk*

Status: Fixed

Description: An attacker's TX could pass this modifier by using multiple contracts to change their `msg.sender`.

Recommendation: Use a single global flag to stop all the potential callees from re-entering these functions within a single TX instead of using user-based global flags to stop each one of them.
For instance:
Remove `mapping(address => uint) public memberLock` and change it into a `bool public has_entered`.

QSP-6 Incorrect weight precision

Severity: *Low Risk*

Status: Fixed

Description: On L124, the requirement statement checks that the weight is greater than or equal to `1` and less than or equal to `1000`. The corresponding message says: "Must be greater than 0.1, less than 10". Taking `1000` to be the baseline, the contract is representing the weight with two decimal precision; hence 1 shall denote `0.01`, instead of `0.1` as the message suggests. An incorrect requirement statement message may inadvertently convey the incorrect precision, which could cause callees to incorrectly adjust the weight as a consequence.

Recommendation: Change the requirement statement message to "Weight must be greater than 0.01, less than 10". In `listPool`, document the expected weight precision in the function.

QSP-7 `updateRewards` may run out of gas

Severity: *Low Risk*

Status: Fixed

Description: The loop in `updateRewards` expects the pool count to be less than `100`; however, the code does not bound `arraySynths` in any manner. If not controlled for, the function `updateRewards` may reach a point of never working again due to high gas costs. P.S. The Perlin team said that they're aware of this and the loop count is planned to be less than 100, so it might not be a problem as long as this maximum loop count is being carefully maintained.

Recommendation: Either bound `arraySynths` to always be less than `100`, or change `snapshotPoolsInEra` such as it receives a given offset (start & end positions) to take a snapshot of; such offset is then passed on to `updateRewards`, which uses it to bound the loop iterations.

QSP-8 Unlocked Pragma

Severity: *Informational*

Status: Fixed

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.4.*`. The caret (^) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked." For consistency and to prevent unexpected behavior in the future, it is recommended to remove the caret to lock the file onto a specific Solidity version.

QSP-9 Clone-and-Own

Severity: *Informational*

Status: Acknowledged

Description: The clone-and-own approach involves copying and adjusting open source code at one's own discretion. From the development perspective, it is initially beneficial as it reduces the amount of effort. However, from the security perspective, it involves some risks as the code may not follow the best practices, may contain a security vulnerability, or may include intentionally or unintentionally modified upstream libraries. Rather than the clone-and-own approach, a good industry practice is to use the Truffle framework for managing library dependencies. This eliminates the clone-and-own risks yet allows for following best practices, such as, using libraries.
** 2020-08-25 update **: The Perlin team considers that this is at low risk and all the problems related to it are reversible.

QSP-10 Missing input checks

Severity: *Informational*

Status: Acknowledged

Description: Functions taking an input address (`constructor`, `listSynth`, `listPool`, `addAdmin`, etc) do not check if the given input is different from `0x0`, or if in the case of an expected contract address, that the input is indeed a contract. Passing in an incorrect input address may cause temporary downtime and/or unexpected behavior.
** 2020-08-25 update **: The Perlin team considers that this is at low risk and all the problems related to it are reversible.

Recommendation: 1) Add require statements to check if the input address is different from `0x0`, 2) In the case where the address is expected to be a contract, add a require statement checking if the given address refers to a contract (see https://docs.openzeppelin.com/contracts/2.x/api/utils). For instance, could check if:

- function `listPool` should check if `pool` is not `0x0`
- function `snapshotPools` should check if `rewardAsset` is not `0x0`
- function `sweep` should check if `asset` is not `0x0`

QSP-11 Should zero out a member's rewards for an era after it is claimed

Severity: *Undetermined*

Status: Acknowledged

Description: The function `claim` does not zero out a member's rewards for an era. The specification stated that this should happen, and while it decreases the rewards, there is nothing that decreases the rewards mapping to a specific member. Specifically, the `mapMemberEraPool_Claim` is not zeroed out.
** 2020-08-05 update ** The Perlin team stated that: "Multiple reward assets can be claimed, eg PERL, or BAL, or in future UNI (Uniswap token). These assets are likely to be airdropped to the rewards contract and need to be claimed. Thus, must not zero out claims, since these claims need to be re-used. Instead, a new mapping mapMemberEraAsset_hasClaimed[msg.sender][era][rewardAsset] = true; Tracks whether a member has claimed for a particular reward asset or not".

QSP-12 Potential division by zero

Severity: *Undetermined*

Status: Acknowledged

Description: On `L188`, `total` could be zero; if so, the transaction will revert and taking the snapshot will fail for all pools.

Recommendation: Add an if-statement checking that `total` is not zero; if not zero, then proceed to the assignment `mapEraPool_Share[era][pool] = getShare(part, total, rewardForEra).`
** 2020-08-25 update **: The Perlin team considers that this is at low risk and all the problems related to it are reversible.

QSP-13 `snapshotPoolsInEra` can be called on past eras

Severity: *Undetermined*

Status: Fixed

Description: Currently, it is possible to call `snapshotPoolsInEra` on past eras, that is, on eras whose value is less than `currentEra`.

Recommendation: Change the function `snapshotPoolsInEra` to `internal`, or clarify if this behavior is indeed intended; if not, could also consider adding a requirement statement that `era == currentEra` if this fits the designed user scenario.

Automated Analyses

Mythril

The analysis was completed successfully. No issues were detected.

Slither

Slither identified possible Reentrancys in functions `claim`, `lock`, and `unlock`. These issues, combined with our findings, were added to the finding section.

Adherence to Specification

The functions `listPool` and `removeReward` have more arguments in the code than in the specification table. The functions `snapshotPoolsInEra` and `registerAllClaims` are not in the specification table.

Code Documentation

Current specification does not clarify on the rationale of the system; rather, it seems to verbalize the code as written, which provides auditors with little insights into the design of the system, which hinders any feedback on the design itself. Consider to improve existing spec with further insights into the system's rationale and design choices.

Adherence to Best Practices

- If the intent of `updateRewards` is to be called after `snapshotPools`, perhaps `updateRewards` should require `eraIsOpen[era]=false`.
- Function `snapshotPoolsInEra` could be made internal, which is more in line with the specification.
- `L188`: `mapEra_Total[era]` could be passed in to the call, instead of `total` as that variable is not used elsewhere.
- The `requires` comments of the function in L280 should be enforced by the code.
- It is recommended to replace `L225` and insert `L223` to avoid reentrancy.
- The `transferAdmin` should check that the new admin is not the old admin. Also, separating the case where the new admin is `0x0` into a function `renounceOwnership` or similar would be helpful, unless that case is not desired (in which case the target address should be checked to be non-zero).
- Pool addresses are never checked to be non-zero; they should be.
- Document all `public` and `external` functions following a Natspec format.
- Don't use `uint`; rather, specify which specific integer flavour to use. In this case, it seems `uint256` is the desired one. If so, replace all `uint`s to `uint256`.
- Variable names seem inconsistent. Most of the time developers use camelCase (e.g., `poolIsListed`), but on some occasions they mix it with _ (e.g., `mapEraPool_Balance`). Make sure all variable names are consistent.
- Index events to allow others to search logs efficiently.
- `L93`, typo: `modify in method` -> `modify method`.
- On `L123`, consider changing the `require` statement message to "Asset must not be PERL".
- On `L159`, consider changing the requirement statement message to "Amount must be non-zero".
- `L152`, `L160`, `L173`, `170`, `176`, `180`, `185`, `L219`, `L223`, `L245`, `L246`, `L276`, etc can theoretically overflow. Consider using `SafeMath` for ALL arithmetic operations.
- Add comments to the line immediately prior to what they are documenting. Currently, many comments are to the right of the code.
- Comments in `lock` & `unlock` are not properly indented. Indent them.
- Modifiers are generally used to check whether certain conditions hold. Updating state using modifiers is rather unusual. Hence, consider removing the `flashProof` modifier and adding its content right at the beginning of `lock`.
- On `L259`, consider changing the requirement statement message to "Reward asset must not have been claimed"
- On `L260`, consider changing the requirement statement message to "Era must be opened"
- Function `claim` only returns `true` or reverts. Hence, having a return value is useless and can be removed.

Test Results

Test Suite Results

All 32 tests were passed.

```
Compiled 16 contracts successfully


  PerlinXRewards
    ✓ Should deploy
    ✓ add admin
    ✓ change admin
    ✓ listPool
    ✓ delistPool
    ✓ listPool again
    ✓ listSynth (58ms)
    ✓ User 1 Locks LP1 (45ms)
    ✓ Admin fails to transfer rewards before Snapshots
    ✓ Admin Snapshots (52ms)
    ✓ Users 1 checks Era 1 (88ms)
    ✓ Users 1 claims Era 1 (54ms)
    ✓ Admin Snapshots Era 1, Reward2 (60ms)
    ✓ Users 1 checks Era 1, Reward2 (42ms)
    ✓ Users 1 claims Era 1, Reward2 (43ms)
    ✓ User 2 Locks LP2
    ✓ Admin Snapshots Era 2 (63ms)
    ✓ Users 1&2 checks Era 2 (80ms)
    ✓ Users 1&2 claims Era 2 (80ms)
    ✓ User 2 Locks LP1, User1 Locks up more LP1 (45ms)
    ✓ Admin Snapshots Era 3 (45ms)
    ✓ Users  1&2 checks Era 3 (44ms)
    ✓ Users  1&2 claims Era 3 (97ms)
    ✓ User 1 unlocks
    ✓ Admin Snapshots Era 4 (45ms)
    ✓ Users 2 checks Era 4 (57ms)
    ✓ Users 2 claims Era 4 (52ms)
    ✓ Admin Snapshots Era 5 (45ms)
    ✓ Users checks Era 5 (45ms)
    ✓ Admin Disables Era 5
    ✓ Users fails Era 5
    ✓ User 2 unlocks 2 assets


  32 passing (4s)
```

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

```
82f0289fdeebf6c61dad95201b505d037422ebd9bad5538016eed5a05088bfa4 ./contracts/PerlinXRewards.sol
```

Tests

```
6f51fd19856488a306a3b6e1a2433a8305342a4d0b686b1eae819d76bb426e64 ./test/1_px.js
```

Changelog

- 2020-08-04 - Initial report
- 2020-08-07 - reaudit report

About Quantstamp

Timeliness of content

Notice of confidentiality

Links to other websites

Disclaimer