Quantstamp Security Assessment Certificate

QUANTSTAMP VERIFIED SECURITY CERTIFICATE

August 29th 2020 — Quantstamp Verified

PerlinXerc20emission

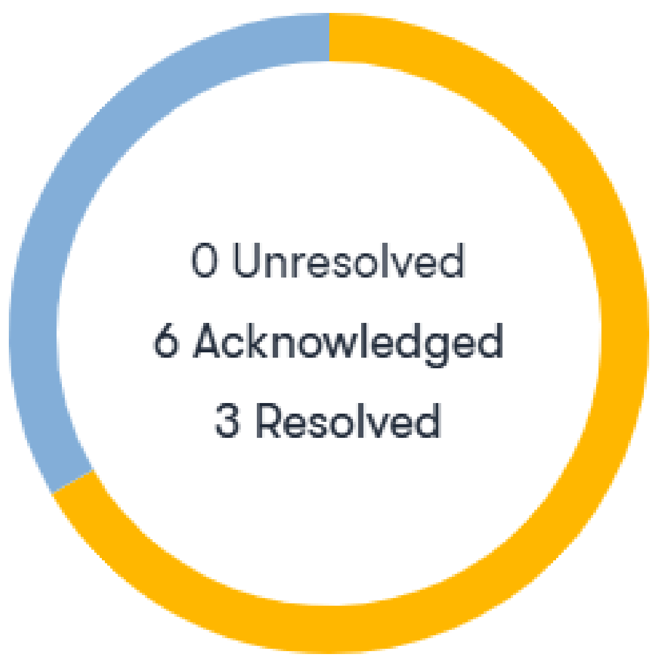
This smart contract audit was prepared by Quantstamp, the protocol for securing smart contracts.

Executive Summary

Type	ERC20
Auditors	Poming Lee, Research Engineer Jan Gorzny, Blockchain Researcher Leonardo Passos, Senior Research Engineer
Timeline	2020-08-04 through 2020-08-19
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	README
Documentation Quality	Low
Test Quality	Low
Source Code	

Repository	Commit
erc20-emissions	1eb828e
perlin2-token-contract	bc94675

Total Issues	9 (3 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	0 (0 Resolved)
Low Risk Issues	3 (0 Resolved)
Informational Risk Issues	5 (2 Resolved)
Undetermined Risk Issues	1 (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 nine potential issues of various levels of severity: three low-severity issues, five informational-level findings, and one undetermined issue. Overall, we recommend better documenting the code and checking function input parameter(s) whenever possible.
Disclaimer: Quantstamp was requested to and had audited a single file: `Perlin.sol`; the whole system was not audited.
** 2020-08-18 udpate **: Perlin team has either fixed or acknowledged all of the issues.

ID	Description	Severity	Status
QSP-1	Constructor does not perform any validation check on input parameter	⌄ Low	Acknowledged
QSP-2	`changeIncentiveAddress` does not perform any validation check on input parameter	⌄ Low	Acknowledged
QSP-3	`changeEraDuration` does not perform any validation check on input parameter	⌄ Low	Acknowledged
QSP-4	Clone-and-Own	○ Informational	Acknowledged
QSP-5	Business logic contradicts the code	○ Informational	Fixed
QSP-6	Privileged Roles and Ownership	○ Informational	Acknowledged
QSP-7	Integer Overflow / Underflow	○ Informational	Fixed
QSP-8	Allowance Double-Spend Exploit	○ Informational	Acknowledged
QSP-9	Contract is subject to race conditions	? 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 Constructor does not perform any validation check on input parameter

Severity: *Low Risk*

Status: Acknowledged

Description: The constructor does not check if `_perlin1` is different from `0x0`, nor if it refers to a contract. If the given address is `0x0` or refers to an externally owned account (EOA), the `Perlin2` contract will not work as expected.
** 2020-08-19 update **: Perlin team states that it is for testing only and will be removed before the deployment of the contract.

Recommendation: Add two requirement statements right at the beginning of the constructor:

- Check that `_perlin1` is not `0x0`

- Check that `_perlin1` refers to a contract (e.g., by using `isContract` from the [`Address`] library)

QSP-2 `changeIncentiveAddress` does not perform any validation check on input parameter

Severity: Low Risk

Status: Acknowledged

Description: The `changeIncentiveAddress` function does not check if `newIncentiveAddress` is different from `0x0`.
** 2020-08-18 update **: Perlin team states that the design permits 0x0 as a valid input parameter.

Recommendation: Add a `require` statement to check that `newIncentiveAddress` is different from `0x0`.

QSP-3 `changeEraDuration` does not perform any validation check on input parameter

Severity: Low Risk

Status: Acknowledged

Description: There is no check on the input value given to `changeEraDuration`, which allows setting the era duration to zero.
** 2020-08-18 update **: Perlin team states that the design permits 0x0 as a valid input parameter.

Recommendation: Add a `require` statement to check that `newDuration` is greater than zero.

QSP-4 Clone-and-Own

Severity: Informational

Status: Acknowledged

Description: The `SafeMath` library is cloned in the file. A subset of the ERC20 has been cloned, with some modifications.
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.

Recommendation: Rather than the clone-and-own approach, a good industry practice is to use the Truffle framework for managing library dependencies (or any other dependency management system of your choice). This eliminates the clone-and-own risks yet allows for following best practices, such as, using libraries.

QSP-5 Business logic contradicts the code

Severity: Informational

Status: Fixed

Description: The `.div(baseline)` on `L234` contradicts the comment about `1bn` on `L233`, since `ERC20(perlin1).totalSupply()` is a variable instead of a constant.

QSP-6 Privileged Roles and Ownership

Severity: Informational

Status: Acknowledged

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. However, this centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner. Specifically, `Perlinx2` gives the `DAO` address a privileged role, as only the `DAO` address can call functions marked as `onlyDAO` (e.g., `changeEmissionCurve`, `changeEraDuration`, etc).
** 2020-08-18 update **: Perlin team has added related explanation to the README file.

Recommendation:

- Invest in public facing documentation stating the specific situations that trigger privileged operations to occur
- Document what security practices are employed by the `DAO` address owner to protect the key that allows him to sign transactions involving privileged operations

QSP-7 Integer Overflow / Underflow

Severity: Informational

Status: Fixed

Description: `SafeMath` is not used on `L223`, `L224`.

Recommendation: Theoretically, lines 223 and 224 may overflow. As such, consider using `SafeMath` instead of Solidity's builtin arithmetic operators.

QSP-8 Allowance Double-Spend Exploit

Severity: Informational

Status: Acknowledged

Description: As it presently is constructed, the contract is vulnerable to the [allowance double-spend exploit], as with other ERC20 tokens. An example of an exploit goes as follows:

1. Alice allows Bob to transfer `N` amount of Alice's tokens (`N>0`) by calling the `approve()` method on `Token` smart contract (passing Bob's address and `N` as method arguments)
2. After some time, Alice decides to change from `N` to `M` (`M>0`) the number of Alice's tokens Bob is allowed to transfer, so she calls the `approve()` method again, this time passing Bob's address and `M` as method arguments
3. Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls the `transferFrom()` method to transfer `N` Alice's tokens

somewhere

4. If Bob's transaction will be executed before Alice's transaction, then Bob will successfully transfer `N` Alice's tokens and will gain an ability to transfer another `M` tokens
5. Before Alice notices any irregularities, Bob calls `transferFrom()` method again, this time to transfer `M` Alice's tokens. The exploit (as described above) is mitigated through use of functions that increase/decrease the allowance relative to its current value, such as `increaseAllowance` and `decreaseAllowance`.

Pending community agreement on an ERC standard that would protect against this exploit, we recommend that developers of applications dependent on `approve()` / `transferFrom()` should keep in mind that they have to set allowance to 0 first and verify if it was used before setting the new value. Teams who decide to wait for such a standard should make these recommendations to app developers who work with their token contract.

Recommendation: Nothing to be done, as all ERC20 contracts are subject to double-spend exploit. At best, at the contract level one can can partially mitigate the issue by exposing functions to increase and decrease allowance, which is already implemented in the current contract.

QSP-9 Contract is subject to race conditions

Severity: *Undetermined*

Status: Fixed

Description: Due to mining non-deterministic order, `changeEmissionCurve`, `changeEraDuration`, `changeIncentiveAddress` may lead to transaction order dependency situations. ** 2020-08-18 update **: Perlin team added a safety check to the function `changeEraTime()`. For the two other functions, Perlin team stated that the likelihood of this attack is rather small due to: "1) Emissions are only paid (intended) once a day, the likelihood that DAO will adjust params on the same block as the emissions is extremely low and 2) The worse case is the DAO transaction reverts, or the emission transaction reverts" and hence suggested not to modify their code.

Recommendation: Consider adding a requirement statement to the referred operations s.t. their execution requires `emitting` to be false.

Automated Analyses

Mythril

The analysis was completed successfully. No issues were detected.

Slither

Slither identified possible Reentrancys in function `upgrade`, after examination it's considered to be a false positive.

Adherence to Best Practices

- (Fixed)`L204` should be split into two lines for readability.
- (Fixed)The literal `totalCap` on `L94` has too many zeros; use exponentiation for readability.
- At the very least, document all external and public functions using a natspec format.
- (Fixed)Make `one` a constant.
- Make `totalCap` a constant.
- (Fixed)Changing a token name is very unusual, and may cause issues to exchanges and others tracking the target token. Unless there is a real justification behind this, we recommend disabling token name modification.

Test Results

Test Suite Results

All 6 tests were passed.

```
Compiled 8 contracts successfully


  Deploy
    ✓ Should deploy (83ms)

  Upgrade
    ✓ Should upgrade (59ms)
    ✓ Should upgrade to next drop (55ms)
    ✓ Should upgrade to full (52ms)

  Be a valid ERC-20
    ✓ Should transfer From
    ✓ Should burn
    ✓ Should burn from

  DAO Functions
    ✓ Non-DAO fails
    ✓ DAO daoChangeEmissionCurve
    ✓ DAO daoChangeIncentiveAddress
    ✓ DAO daoChangeDAO
    ✓ DAO start emitting
    ✓ DAO daoChangeDAO fails
    ✓ DAO daoChangeDAO pass
    ✓ Old DAO fails

  Emissions
    ✓ Should emit properly (2078ms)


  16 passing (3s)
```

Code Coverage

While many statements in the code are tested, the branching coverage is poor. The test coverage could be improved.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	82.96	48.21	78.26	83.21	
Perlin.sol	88.54	58.33	84.38	88.78	… 200,224,225
Perlin1.sol	68.42	30	61.54	68.42	… 50,54,58,59
PerlinDAO.sol	100	100	100	100	
All files	**82.96**	**48.21**	**78.26**	**83.21**	

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

```
ba25a329669666e66ffa44a010003208f0515f637f4c1e4642179537c72bfa66 ./contracts/Perlin.sol
```

Tests

```
0db8f0d3b959cda02a55f8d58cf384633e98bc830141595868364f388265428f ./test/1_perl.js
```

Changelog

- 2020-08-10 - Initial report
- 2020-08-19 - reaudit report

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp, Inc. (Quantstamp). Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on the website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.