Quantstamp Security Assessment Certificate

QUANTSTAMP VERIFIED · SECURITY CERTIFICATE

December 19th 2020 — Quantstamp Verified

Wrapped Filecoin Factory

This security assessment was prepared by Quantstamp, the leader in blockchain security

Executive Summary

Type	WFIL management contract
Auditors	Kacper Bąk, Senior Research Engineer Fayçal Lalidji, Security Auditor Luís Fernando Schultz Xavier da Silveira, Security Consultant
Timeline	2020-11-30 through 2020-12-04
EVM	Muir Glacier
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Documentation Quality	Low
Test Quality	Medium

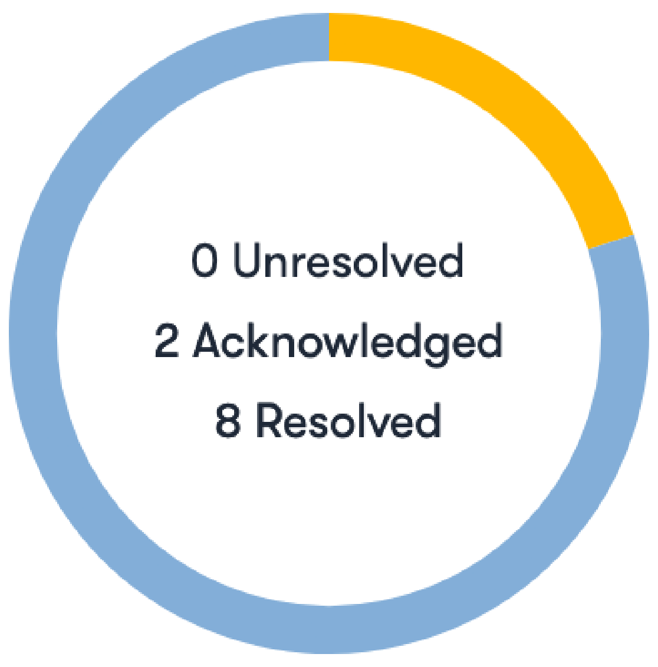
Source Code

Repository	Commit
wfil-factory	c6e5bc2
None	d791b24
None	4b44e1b

Goals

- Can funds get locked up in the contract?
- Can an unauthorized user abused the contract?

Total Issues	10	(8 Resolved)
High Risk Issues	3	(3 Resolved)
Medium Risk Issues	2	(1 Resolved)
Low Risk Issues	3	(2 Resolved)
Informational Risk Issues	1	(1 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

The scope of this review as limited to the file `WFILFactory.sol`. We have found a few issues. Notably, three high- and two medium-severity issues among others. The project lacks documentation so the intent is not always clear. We highly recommend addressing all the issues before deploying the code.

ID	Description	Severity	Status
QSP-1	Not Returned Fund When Rejecting Burn Requests	︽ High	Fixed
QSP-2	Burn Request Rejection	︽ High	Fixed
QSP-3	Unused modifier `whenNotPaused`	︽ High	Fixed
QSP-4	Untrusted Bridge	⌃ Medium	Mitigated
QSP-5	Custodian fees	⌃ Medium	Acknowledged
QSP-6	Privileged Roles and Ownership	⌄ Low	Acknowledged
QSP-7	Functions return `false` upon success	⌄ Low	Fixed
QSP-8	Default admin removal	⌄ Low	Fixed
QSP-9	Unupdated Pauser Role	○ Informational	Fixed
QSP-10	Deposits can be set multiple times	? 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.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

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:

- Slither v0.6.12

Steps taken to run the tools:

1. Installed the Slither tool: `pip install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 Not Returned Fund When Rejecting Burn Requests

Severity: *High Risk*

Status: Fixed

File(s) affected: `WFILFactory.sol`

Description: When a merchant create a burn request, the amount of `wfil` tokens are burned using `unwrap`. However, when `rejectBurnRequest` is called by the custodian address, the initially burned amount is not given back to the merchant.

Recommendation: The initially burned value should be minted to the merchant when rejecting a bun request.

QSP-2 Burn Request Rejection

Severity: *High Risk*

Status: Fixed

File(s) affected: `WFILFactory.sol`

Description: A burn request set by a merchant and triggered by calling `burn()` can only be confirmed by a custodian. If the token transfer on the other side of the bridge fails for any reason, the burnt amount will not be returned to the merchant since no logic is implemented to handle such situation.

Recommendation: We recommend adding a function that rejects burn requests and returns the funds.

QSP-3 Unused modifier `whenNotPaused`

Severity: *High Risk*

Status: Fixed

File(s) affected: `WFILFactory.sol`

Description: No functions are actually paused by `pause()` in the contract as opposed to the description in L422. The modifier `whenNotPaused` appears unused.

Recommendation: We recommend either removing the pause functionality altogether or using the modifier `whenNotPaused` where appropriate.

QSP-4 Untrusted Bridge

Severity: *Medium Risk*

Status: Mitigated

File(s) affected: `WFILFactory.sol`

Description: Wrapped filecoin is a bridge that allows cross chain token transfer. No on-chain consensus or incentives/penalties are implemented to guarantee the trustless nature of the assets creation or transfers. Any listed custodian address will be allowed to approve merchants mint/burn requests.

Recommendation: We recommend informing users about these privileged roles. Furthermore, we recommend setting up a multisig wallet with multiple custodian voting.

QSP-5 Custodian fees

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `WFILFactory.sol`

Description: Custodians have to pay for gas themselves to approve/reject requests and also run risks related to value storage. They are not compensated anywhere in the contract logic (e.g., by fees).

Recommendation: We recommend documenting the fees and incentives structure. If there are none, this could be a major design flaw.
Update: Following the project team, the architecture implies that every custodian will negotiate the fee with its respective merchant

QSP-6 Privileged Roles and Ownership

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `WFILFactory.sol`

Description: Smart contracts will often have an owner or other roles to designate entities with special privileges to govern the smart contract.

Recommendation: 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.

QSP-7 Functions return `false` upon success

Severity: *Low Risk*

Status: Fixed

File(s) affected: `WFILFactory.sol`

Description: The following functions return `false` (by default) upon success: `addCustodian()`, `removeCustodian()`, `addMerchant()`, `removeMerchant()`.

Recommendation: We recommend either returning `true` or changing function signatures to return nothing.

QSP-8 Default admin removal

Severity: *Low Risk*

Status: Fixed

File(s) affected: `WFILFactory.sol`

Description: The function `setOwner()` does not check if `newOwner != _owner`. Consequently, if they are equal, the contract will be left without a default admin.

Recommendation: We recommend adding a relevant check.

QSP-9 Unupdated Pauser Role

Severity: *Informational*

Status: Fixed

File(s) affected: `WFILFactory.sol`

Description: Pauser role and admin role is assigned to the same address inside the constructor. However, when updating the admin address using `setOwner()` the pauser role is not handled, meaning that the previous admin address will still have the pauser role assigned. However, the role can still be removed using `revokeRole()`.

Recommendation: We recommend checking whether this is the intended behavior of the contract.

QSP-10 Deposits can be set multiple times

Severity: *Undetermined*

Status: Fixed

File(s) affected: `WFILFactory.sol`

Description: The functions `setCustodianDeposit()` and `setMerchantDeposit()` do not check whether a deposit already exists, and therefore allow to overwrite the previous deposit information.

Recommendation: We recommend checking whether this design is intentional. Furthermore, we recommend explaining how and when the deposits are set and cleared.

Automated Analyses

Slither

Slither reported reentrancy in the function `reclaimToken()`. We classified it as a false positive.

Adherence to Specification

1. Why is burnNonce updated in L300 but mintNonce not updated similarly?
2. The WFIL market is invite-only, which greatly limits its usefulness.

Code Documentation

1. The project lacks documentation and lacks natspec descriptions of functions.
2. L160: not the caller (in general).
3. L120: replace “wfil token” by “owner”.
4. L137: did you mean “@notice” rather than “@dev”?

Adherence to Best Practices

1. In the function `addMintRequest()` deposit parameter is not needed since for each merchant deposit address is set by the custodian in the custodian mapping.
2. The revert message in L160 is incorrect since the caller is not the merchant but the custodian.
3. When adding and removing a merchant using `addMerchant()` and `removeMerchant()`, it should be checked that the address is not already listed.
4. Inconsistent indentation in L449-485.
5. Some boolean functions always return `true`. Why is it needed?
6. Consider making the following checks:
    1. `addMintRequest()`: `amount > 0` and `txId` is not empty.
    2. `burn()`: `amount > 0`.
    3. `confirmBurnRequest()`: `txId` is not empty.
    4. `getMintRequest()`, `getBurnRequest()`: that `nonce` is valid (less than the corresponding counter’s value).
7. L484: better to revert.
8. L359: did you mean `memory` rather than `storage` (as in L328)?
9. Function `_isEmptyString()`: why not just `a.length == 0`?
10. Choose a convention and keep it throughout: either `uint` or `uint256`. We recommend `uint256`.

11. Inconsistent function parameter naming convention.

12. Consider moving L54 to after L23 to keep library declarations together.

Test Results

Test Suite Results

All tests passed successfully.

```
WFILFactory
  Setup
     ✓ dao has the default admin role (69ms)
     ✓ dao has the pauser role (82ms)
     ✓ pauser is the default admin
  fallback()
     ✓ should revert when sending ether to contract address
  setCustodianDeposit()
     ✓ custodian can set custodian deposit address (100ms)
     ✓ should emit the appropriate event when new custodian deposit address is set (53ms)
     ✓ should revert when merchant is set to zero address (68ms)
     ✓ other accounts cannot set a new custodian deposit address (48ms)
  setMerchantDeposit()
     ✓ merchant can set merchant deposit address (88ms)
     ✓ should emit the appropriate event when new merchant deposit address is set (59ms)
     ✓ other accounts cannot set a new merchant deposit address (45ms)
  addMintRequest()
     ✓ merchant can add a mint request (141ms)
     ✓ should emit the appropriate event when a merchant add a mint request (99ms)
     ✓ other accounts cannot add a mint request (38ms)
  cancelMintRequest()
     ✓ merchant can cancel a mint request (173ms)
     ✓ should emit the appropriate event when a merchant cancel a mint request (160ms)
     ✓ other accounts cannot cancel a mint request (143ms)
  confirmMintRequest()
     ✓ custodian can confirm a mint request (270ms)
     ✓ should emit the appropriate event when a custodian confirm a mint request (180ms)
     ✓ other accounts cannot confirm a mint request (228ms)
  rejectMintRequest()
     ✓ custodian can reject a mint request (283ms)
     ✓ should emit the appropriate event when a custodian reject a mint request (230ms)
     ✓ other accounts cannot reject a mint request (259ms)
  addBurnRequest()
     ✓ merchant can burn wrapped filecoin (766ms)
     ✓ should emit the appropriate event when a merchant burn wrapped filecoin (744ms)
     ✓ other accounts cannot burn wrapped filecoin (757ms)
  confirmBurnRequest()
     ✓ merchant can confirm a burn request (1607ms)
     ✓ should emit the appropriate event when a merchant confirm a burn request (1229ms)
     ✓ other accounts cannot confirm a burn request (808ms)
  rejectBurnRequest()
     ✓ custodian can reject a burn request (1110ms)
     ✓ should emit the appropriate event when a custodian reject a burn request (624ms)
     ✓ other accounts cannot reject a burn request (617ms)
  reclaimToken()
     ✓ dao can reclaim erc20 tokens sent to factory contract (147ms)
     ✓ should emit the appropriate event when an erc20 token is claimed (98ms)
     ✓ other accounts cannot reclaim erc20 tokens (77ms)
  addCustodian()
     ✓ default admin should be able to add a new custodian (137ms)
     ✓ should emit the appropriate event when a new custodian is added (60ms)
     ✓ should revert when account is set to zero address (60ms)
     ✓ other address should not be able to add a new custodian (75ms)
  removeCustodian()
     ✓ default admin should be able to remove a custodian (102ms)
     ✓ should emit the appropriate event when a custodian is removed (80ms)
     ✓ other address should not be able to remove a custodian (82ms)
  addMerchant()
     ✓ default admin should be able to add a new merchant (134ms)
     ✓ should emit the appropriate event when a new merchant is added (64ms)
     ✓ should revert when account is set to zero address (82ms)
     ✓ other address should not be able to add a new merchant (99ms)
  removeMerchant()
     ✓ default admin should be able to remove a merchant (151ms)
     ✓ should emit the appropriate event when a merchant is removed (79ms)
     ✓ other address should not be able to remove a merchant (78ms)
  pausing
     ✓ dao can pause (149ms)
     ✓ dao can unpause (281ms)
     ✓ custodian cannot set custodian deposit while paused (303ms)
     ✓ merchant cannot set merchant deposit while paused (203ms)
     ✓ merchants cannot add mint requests while paused (339ms)
     ✓ merchants cannot cancel mint requests while paused (587ms)
     ✓ custodians cannot confirm/reject mint requests while paused (695ms)
     ✓ merchants cannot add burn requests while paused (1265ms)
     ✓ custodians cannot confirm/reject burn requests while paused (1232ms)
     ✓ other accounts cannot pause (63ms)
```

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

`df5d1befcff1a9fcffb1e48c68be4600ac9ac8d35dd9fc1b5de55d5744baac0f ./WFILFactory.sol`

Tests

`a1b9160f52d415980655d9ad33b8d3932532ae72f5296472e8dfbfa6a0735bd7 ./WFILFactory.test.js`

Changelog

- 2020-12-04 - Initial report
- 2020-12-18 - Re-audit report update
- 2020-12-19 - Re-audit, commit 4b44e1b

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.