



December 2nd 2020 — Quantstamp Verified

Wrapped Filecoin

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

Executive Summary

Type	ERC20 token				
Auditors	Kacper Bąk, Senior Research Engineer Leonardo Passos, Senior Research Engineer Luís Fernando Schultz Xavier da Silveira, Security Consultant				
Timeline	2020-11-12 through 2020-12-02				
EVM	Muir Glacier				
Languages	Solidity, Javascript				
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review				
Specification	None				
Documentation Quality	<div><div></div>High</div>				
Test Quality	<div><div></div>High</div>				
Source Code	<table><tr><th>Repository</th><th>Commit</th></tr><tr><td>wfil-contracts</td><td>964c30b</td></tr></table>	Repository	Commit	wfil-contracts	964c30b
Repository	Commit				
wfil-contracts	964c30b				
Goals	<ul style="list-style-type: none">Does wrapping and unwrapping work correctly?Does role setting work correctly?				



⬆️ 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.

Total Issues	7 (6 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	0 (0 Resolved)
Low Risk Issues	5 (4 Resolved)
Informational Risk Issues	1 (1 Resolved)
Undetermined Risk Issues	1 (1 Resolved)



🔴 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

We have found few low-severity and informational issues in the reviewed code. We also provide ideas for further improvements in the code and documentation.
Update: the reaudit is based on commit [2bcacf3](#) in <https://github.com/wfil/wfil-token>. The team has addressed most of the issues.
Disclaimer: the contract relies on an external DAO contract. Such a contract and any external interaction with the DAO is outside of the scope of this audit.

ID	Description	Severity	Status
QSP-1	Privileged Roles and Ownership	Low	Acknowledged
QSP-2	No restrictions on the fees that can be set	Low	Fixed
QSP-3	The contract may receive <code>WFIL</code> tokens through minting	Low	Fixed
QSP-4	Race condition between <code>wrap/unwrap</code> and <code>setFee</code>	Low	Fixed
QSP-5	Function parameters lack sanity checks	Low	Fixed
QSP-6	Allowance Double-Spend Exploit	Informational	Mitigated
QSP-7	<code>false</code> returned from functions <code>addMinter()</code> and <code>removeMinter()</code> upon successful execution	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 Privileged Roles and Ownership

Severity: Low Risk

Status: Acknowledged

File(s) affected: [WFIL.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.

Update: the team informed us that they plan on governing the contract via a multisig wallet.

QSP-2 No restrictions on the fees that can be set

Severity: *Low Risk*

Status: Fixed

File(s) affected: [WFIL.sol](#)

Description: Fee setters may set arbitrarily low or high fees through the function `_setFee()`. If set to zero, no fee will be charged; if set to be high, wrapping/unwrapping tokens will end up costing a lot.

Recommendation: We recommend either adding a hard-coded limit for the fees and/or informing users about this (lack of) limit.

QSP-3 The contract may receive [WFIL](#) tokens through minting

Severity: *Low Risk*

Status: Fixed

File(s) affected: [WFIL.sol](#)

Description: The contract may receive [WFIL](#) tokens accidentally through the `_mint` function in `wrap`

Recommendation: To avoid this issue as well as accidental transfers to the contract, move the check in L180 to the `_beforeTokenTransfer()` function.

QSP-4 Race condition between `wrap/unwrap` and `setFee`

Severity: *Low Risk*

Status: Fixed

File(s) affected: [WFIL.sol](#)

Description: An actor with privilege to change the fee could set a new value (t2) while another transaction had already been sent to (un-)wrap tokens (t1); if t2 is mined before t1, the (un-)wrap function will collect a different fee than expected.

Recommendation: Since `setFee()` and `wrap()/unwrap()` are protected operations, coordinate their execution off-chain amongst `FEE_SETTER_ROLE` actors with `MINTER_ROLE` actors.

QSP-5 Function parameters lack sanity checks

Severity: *Low Risk*

Status: Fixed

File(s) affected: [WFIL.sol](#)

Description: Arguments passed to the contract functions do not have any sanity checks. For example, addresses could be set to `0x0`.

Recommendation: Add checks verifying that the given input addresses are not `0x0`. If there is any expectation that any of those addresses should also be a contract, add a corresponding check. Additionally, consider checking that:

- `minter_ != address(0)` in the `constructor()`;
- `wfilFee` is at most 1000 in `_setFee()`;
- `amount > 0` in `wrap()` and `unwrap()`;
- `filaddress` is valid (or an approximation of that) in `unwrap()`;
- `account != address(0)` in `addMinter()` and `addFeeSetter()`.

QSP-6 Allowance Double-Spend Exploit

Severity: *Informational*

Status: Mitigated

File(s) affected: [WFIL.sol](#)

Description: As it presently is constructed, the contract is vulnerable to the [allowance double-spend exploit](#), as with other ERC20 tokens.

Exploit Scenario:

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.

Recommendation: 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.

QSP-7 `false` returned from functions `addMinter()` and `removeMinter()` upon successful execution

Severity: *Undetermined*

Status: Fixed

File(s) affected: `WFIL.sol`

Description: The functions `addMinter()` and `removeMinter()` always return `false` when the execution succeeds.

Recommendation: We recommend either changing function signatures to return nothing, or return `true` upon successful execution.

Automated Analyses

Slither

Slither reported no issues.

Adherence to Specification

- 1. It is unclear what the `_beforeTokenTransfer()` function override is attempting to achieve. It appears unnecessary since the contract inherits from `ERC20Pausable`. **Update:** fixed, the contract no longer inherits from `ERC20Pausable`.
- 2. The fee granularity is set to 1000. Is this intentional? The fee granularity can be increased without risking overflow problems. This may be important as currently the minimum fee is 0.1%, which may render the contract unable to compete with alternative solutions or forks. **Update:** fixed.
- 3. From the contract’s event set, there is no way for an event listener to figure out who unwrapped the funds from an account, but that is important information as this is who should receive the FIL on the filecoin blockchain. Recommendation: add `msg.sender` to the `UnwrappedFrom` event.

Code Documentation

- 1. L160,167: not all the functions are impacted.
- 2. L106: typo “uwnrapOut”. **Update:** fixed.
- 3. L108: typo “uwrap”. **Update:** fixed.
- 4. There are some typos in `README.md`, e.g., `fist` (instead of `first`), `custodial` (instead of `custodian`), `they'll get automatically` (instead of `they will automatically get`), etc. **Update:** fixed.
- 5. 1. In `WFIL.sol` (L98, L112), `1000` is a magic constant. Document it properly. **Update:** fixed.
- 6. The `WFIL` token has the default 18 decimal places but, according to a web search, the filecoin token `FIL` has 8. Although not likely to become an issue, this is better documented.
- 7. In the updated repository, L44: remove “which send value”.
- 8. In the updated repository, L49: replace “WFIL” by “FIL” or “filecoin”.
- 9. In the updated repository, L64: replace “WFil” by “WFIL”.
- 10. In the updated repository, L101,108: replace “all functions” by “transfers”.

Adherence to Best Practices

- 1. There is no need to inherit from both `ERC20` and `ERC20Pausable` as `ERC20Pausable` itself inherits from `ERC20`. **Update:** fixed.
- 2. Unless there is a strong reason not to, consider making the functions `setFee()`, `setFeeTo()`, `wrap()`, `unwrap()`, `addMinter()`, `removeMinter()`, `addFeeSetter()` and `removeFeeSetter()` return nothing. They always return `true` unless they revert.
- 3. Consider providing an error message in L60. **Update:** fixed.
- 4. The constructor uses a different argument naming convention than the rest of the code.
- 5. Consider indexing event parameters. **Update:** fixed.
- 6. Use unsigned integer type explicitly, i.e., replace all uses of `uint` by `uint256`. **Update:** fixed.
- 7. In `WFIL.sol`, the `filaddress` is not in lower camel case as in other parts of the code. The parameter is also misspelled. Follow the name conventions already in place. Typo: `filaddress` => `fileAddress`. **Update:** fixed.
- 8. In the updated repository, consider checking that `account` is a minter in `removeMinter()`.

Test Results

Test Suite Results

All tests executed successfully.

```
WFIL
  Setup
    ✓ dao has the default admin role (63ms)
    ✓ dao has the pauser role (51ms)
    ✓ pauser is the default admin
  fallback()
    ✓ should revert when sending ether to contract address
WFIL metadata
  ✓ has a name
  ✓ has a symbol
wrap()
```

```

    ✓ factory can mint tokens (58ms)
    ✓ should emit the appropriate event when wfil is wrapped (38ms)
    ✓ should revert when amount is less than zero (72ms)
    ✓ other accounts cannot wrap tokens
  unwrap()
    ✓ wfil merchant should be able to burn wfil (50ms)
    ✓ should emit the appropriate event when wfil is unwrapped
    ✓ should revert when amount is less than zero
    ✓ other accounts cannot unwrap tokens
  unwrapFrom()
    ✓ wfil burner should be able to burn wfil (53ms)
    ✓ should emit the appropriate event when wfil is unwrappedFrom
    ✓ should revert when amount is less than zero
    ✓ other accounts cannot unwrapFrom tokens
  addMinter()
    ✓ default admin should be able to add a new minter (48ms)
    ✓ should emit the appropriate event when a new minter is added
    ✓ should revert when account is set to zero address
    ✓ other address should not be able to add a new minter
  removeMinter()
    ✓ default admin should be able to remove a minter (49ms)
    ✓ should emit the appropriate event when a minter is removed
    ✓ other address should not be able to remove a minter
  pausing
    ✓ owner can pause (56ms)
    ✓ owner can unpause (71ms)
    ✓ cannot wrap while paused (102ms)
    ✓ cannot transfer while paused (119ms)
    ✓ cannot unwrap while paused (112ms)
    ✓ other accounts cannot pause
  ERC20 _beforeTokenTransfer hook
    ✓ check wrap() for revert when trying to mint to the token contract
    ✓ check transfer() for revert when trying to transfer to the token contract (41ms)
    ✓ check transferFrom() for revert when trying to transfer to the token contract (60ms)

34 passing (7s)
```

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

4828e656ac83c0ae86223e4350ddffb6947804d32ec8b19f59950641f551e02e ./contracts/WFIL.sol
8a6b38936c738a0e612391ee231f39352cc8878f4a5b41c05f0895fb662b3fd6 ./contracts/Migrations.sol

Tests

2ca7e444a187a86f92347b3370c2f6ee7d1c37699279ded5b752f16ab69c6198 ./test/WFIL.test.js

Changelog

- 2020-11-18 - Initial report
- 2020-12-02 - reaudit based on commit [2bcacf3](#) in <https://github.com/wfil/wfil-token>

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.

