Quantstamp Security Assessment Certificate

November 8th 2020 — Quantstamp Verified

StormX - Token Swap

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

Executive Summary

Type	Token and Token Swap
Auditors	Kacper Bąk, Senior Research Engineer Leonardo Passos, Senior Research Engineer Ed Zulkoski, Senior Security Engineer
Timeline	2020-03-18 through 2020-04-24
EVM	Constantinople
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Computer-Aided Verification, Manual Review
Specification	StormX ERC20 Token Swap
Source Code	

Repository	Commit
stormx-token	0d1a63b

Goals	• Can users' tokens be locked up forever? • Can users be charged correctly after the token upgrade? • Can user lose any tokens in the process of token upgrade?

Total Issues	**3** (0 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	0 (0 Resolved)
Low Risk Issues	**3** (0 Resolved)
Informational Risk Issues	0 (0 Resolved)
Undetermined Risk Issues	0 (0 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

Generally, the code is well written, well documented, and well tested. We communicated several issue and recommendations to the development team, provided ideas for improvements.
Update: as of commit `813acda` and `8177534` our main concerns are addressed.

ID	Description	Severity	Status
QSP-1	Centralization of Power	Low	Acknowledged
QSP-2	Transaction order dependencies between `setChargeFee()` and functions that read `chargeFee`	Low	Acknowledged
QSP-3	Allowance Double-Spend Exploit	Low	Acknowledged

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:

- SolidityCoverage v0.5.8
- Mythril v0.2.7
- Slither v0.6.6

Steps taken to run the tools:

1. Installed the solidity-coverage tool (within the project's root directory): `npm install --save-dev solidity-coverage`
2. Ran the coverage tool from the project's root directory: `./node_modules/.bin/solidity-coverage`
3. Installed the Mythril tool from Pypi: `pip3 install mythril`
4. Ran the Mythril tool on each contract: `myth -x path/to/contract`
5. Installed the Slither tool: `pip install slither-analyzer`
6. Run Slither from the project directory: `slither .s`

Findings

QSP-1 Centralization of Power

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `StormXGSNRecipient.sol`

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.
In this project, centralization of power has the form of the owner being able to set fees associated with the GSN network, and to close the migration after the time window enforced by the smart contract elapses. The owner does not have any privileges to confiscate users' tokens.

Recommendation: The centralization of power is documented and explained to users.

QSP-2 Transaction order dependencies between `setChargeFee()` and functions that read `chargeFee`

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `StormXGSNRecipient.sol`

Description: There exists transaction order dependencies between `setChargeFee()` and functions that read `chargeFee`. If `chargeFee` increases, it may result in rejecting a user's meta-transaction.

Exploit Scenario: For example, assume that `chargeFee` = 1. If a user knows that and they have exactly enough tokens to cover the `chargeFee` , they can submit the meta-transaction to the relayer and it will get executed. If after submitting the transaction to the relayer (but before it gets submitted to the contract), the owner changes `chargeFee` to 2 and then the transaction gets executed, it will fail.

Recommendation: We do not have a recommendation. TOD issues are typically benign yet difficult to fix.

QSP-3 Allowance Double-Spend Exploit

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `StormXToken.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. 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`.

Recommendation: 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.

Adherence to Specification

- It is unclear whether `unlockedBalanceOf()` should return 0 instead of reverting if `lockedBalanceOf[account]` > `balanceOf(account)`.
- in `Swap.sol`, is there a reason that `disableMigration()` needs the `reserve` parameter?

Code Documentation

- In `StormXToken.sol`, what is the point of enabling/disabling `transfers()`?
- The following requirement in the `README.md` may be unclear to some readers:

If the user does not have enough unlocked token balance and is calling the function convert(uint256 amount), this contract accepts the GSN relayed call and charges users only if they will have enough unlocked new token balance after convert(uint256 amount) is executed, i.e. amount >= chargeFee.

For example, it could be parsed like this:

[If the user does not have enough unlocked token balance and is calling the function convert(uint256 amount), this contract accepts the GSN relayed call] **and** *[charges users only if they will have enough unlocked new token balance after convert(uint256 amount) is executed, i.e. amount >= chargeFee].*

although the following is implemented:

[If the user does not have enough unlocked token balance and is calling the function convert(uint256 amount), this contract accepts the GSN relayed call and charges users] **only if** *[they will have enough unlocked new token balance after convert(uint256 amount) is executed, i.e. amount >= chargeFee]*. **Update:** resolved.

- ` README.md states that:

The token contract includes features for privileged access that allow StormX to mint new tokens for and remove tokens from arbitrary accounts. The StormX team sought to develop a new ERC20 token smart contract that will not include the aforementioned functions.

We recommend to explicitly state the names of those functions that you refer to when when stating *aforementioned functions*. As the new contract (the one developed by Quanstamp) does have a mint function, such a clarification is required to give a clear understanding.

- `README.md`: GSN acronym is used, but not defined. When first using the GSN acronym, write what it means => “Gas Station Network”. **Update:** resolved.

- In `StormXGSNRecipient.sol`: the `constructor` and `acceptRelayedCall` lack documentation. We recommend adding documentation. At the very least, all external and public functions should be documented. **Update:** resolved.

- In `StormXToken.sol#40` the following comment is confusing: *@param reserve address of the StormX's reserve that receives*”. We recommend improving the comment. **Update:** resolved.

- In `StormXToken.sol#L75` the following this comment is confusing: *@param account address of the user this function queries unlocked balance for*. We recommend improving the comment.

- In `StormXToken.sol#L150` the following comment is confusing: *@param recipients an array of address of the recipient*. Probably meant *an array of recipient addresses*. We recommend improving the comment. **Update:** resolved.

Adherence to Best Practices

- in `StormXGSNRecipient.sol#48`, we recommend defining a constant for `selector` to save gas

- in `StormXGSNRecipient.sol`, `convert()` is only relevant to the `Swap` contract. `StormXToken`, however, inherits from `StormXGSNRecipient`. If you want both contracts to reuse the code, please document the use case.

- in `StormXGSNRecipient.sol#58`, instead of defining `INSUFFICIENT_BALANCE`, we recommend reusing `GSNRecipient.RELAYED_CALL_REJECTED`, particularly since the else clause is not related to the balance.

- `Swap.sol` still references `mock/OldStormXToken`. We recommend declaring an interface to import instead.

- in `Swap.sol#17`, "supporing" should be "supporting". **Update:** resolved.

- in `StormXToken` the constructor could re-use `addGSNRecipient()`. **Update:** resolved (function removed).

- in `StormXToken`, in `transfers()`, the parameter `recipients` shadows the state variable of the same name. **Update:** resolved.

- in `StormXToken.sol` there is a lot of trailing space throughout the file. We recommend remove any trailing space. **Update:** resolved.

- in `StormXToken.sol` and `StormXGSNRecipient.sol` some functions always return `true`. Consequently, the return value has no semantics, as no exception is ever going to happen (otherwise, a false value could be returned). `enableTransfers()` is an example of such a function. We recommend for functions that only return true and are not part of the ERC-20 standard to return nothing.

- in `StormXToken.sol` the functions `addGSNRecipient()` and `deleteGSNRecipient()` do not check if `recipient` is different from 0x0. We recommend adding `require()` statements to check if `recipient` is different from 0x0. **Update:** resolved.

- in `StormXToken.sol`, the `mint()` function uses the `onlyMinter` modifier, but later restricts that `sender` must be equal to the `swap` address. Two scenarios are possible here: (a) if the sender must be equal to the swap address, then the `onlyMinter` modifier is unnecessary; (b) on the other hand, if only added minters can call such a function, then the require statement checking equality against the swap address is not needed. Choose what scenario applies and change the code accordingly. **Update:** resolved.

Test Results

Test Suite Results

```
Contract: StormX token GSN rewarding feature test
   ✓ owner and only owner can assign reward role via GSN test (675ms)
   ✓ owner and rewardRole can reward users via GSN success test (880ms)
   ✓ users cannot invoke reward() test (86ms)
   ✓ revert when invalid parameter provided in reward() test (109ms)
   ✓ revert when not enough tokens to reward users test (136ms)
   ✓ revert when input lenghts do not match in rewards() via GSN test (134ms)
   ✓ owner and rewardRole can reward users in batch via GSN success test (989ms)
   ✓ rewarding in batch via GSN fails if not enough balance of caller test (183ms)
   ✓ setAutoStaking via GSN success test (555ms)
   ✓ rewarded tokens will not be staked if auto-staking feature is disabled test (889ms)

Contract: StormX token staking feature GSN test
   ✓ GSN lock fails if not enough unlocked balance of user test (252ms)
   ✓ GSN lock success test (346ms)
   ✓ GSN unlock fails if not enough locked balance of user test (347ms)
   ✓ GSN unlock succeeds if enough unlocked balance of user after transaction test (1088ms)
   ✓ GSN unlock fails if not enough unlocked balance of user after transaction test (592ms)
   ✓ GSN unlock success test (573ms)

Contract: StormX token GSN test
   ✓ GSN call fails if not enough balance of user test (209ms)
   ✓ GSN transfer fails if not enough balance after being charged test -- case1 (298ms)
   ✓ GSN transfer fails if not enough balance after being charged test -- case2 (126ms)
   ✓ GSN transfer success test (283ms)
   ✓ GSN transferFrom fails if not enough balance to be transferred test (308ms)
   ✓ GSN transferFrom fails if not enough balance to be charged test (425ms)
   ✓ GSN transferFrom success only with enough allowance test (634ms)
   ✓ GSN transfers success test (423ms)
```

```
    ✓ GSN transfers fails if input lengths do not match in transfers test (94ms)
    ✓ GSN transfers fails if any transfer fails test (125ms)
    ✓ owner and only owner can enable/disable GSN transfers via GSN test (1147ms)
    ✓ GSN transferFrom success if enough token balance after transaction test (1100ms)
    ✓ owner and only owner can set GSN charge test (451ms)
    ✓ reverts if invalid parameter provided in set stormx reserve address test (98ms)
    ✓ owner and only owner can set stormx reserve address test (1021ms)

  Contract: StormX token rewarding test
    ✓ owner and only owner can assign reward role test (88ms)
    ✓ owner and rewardRole can reward users success test (285ms)
    ✓ revert when invalid parameter provided in reward() test (60ms)
    ✓ revert when not enough tokens to reward users test (65ms)
    ✓ users cannot invoke reward() test
    ✓ revert when input lenghts do not match in rewards() test
    ✓ owner and rewardRole can reward users in batch success test (393ms)
    ✓ rewarding in batch fails if not enough balance of caller test (158ms)
    ✓ setAutoStaking success test (121ms)
    ✓ rewarded tokens will not be staked if auto-staking feature is disabled test (352ms)

  Contract: StormX token test
    ✓ name test
    ✓ symbol test
    ✓ decimals test
    ✓ initialize success test (106ms)
    ✓ revert if initialize not called by owner test (82ms)
    ✓ revert if initialize twice test (120ms)
    ✓ revert if invalid parameters provided in initialize test (101ms)
    ✓ revert if invalid parameters provided in constructor test (47ms)
    ✓ revert if not authorized to mint test
    ✓ revert if calling mint before initialization test (83ms)
    ✓ owner and only owner can set stormX reserve (94ms)
    ✓ revert when invalid address provided in set stormX reserve
    ✓ read locked balance of user success test (77ms)
    ✓ read unlocked balance of user success test (70ms)
    ✓ transfer reverts if not enough unlocked token test (102ms)
    ✓ transfer success test (186ms)
    ✓ transferFrom reverts if not enough unlocked token test (122ms)
    ✓ transferFrom success test (249ms)
    ✓ lock reverts if no enough unlocked token test (102ms)
    ✓ lock success test (176ms)
    ✓ unlock reverts if no enough locked token test (103ms)
    ✓ unlock success test (218ms)
    ✓ revert if input lengths do not match in transfers test
    ✓ revert if transfers not available test (60ms)
    ✓ revert if any transfer fails test
    ✓ transfers success test (84ms)
    ✓ owner and only owner can enable/disable transfers test (456ms)

  Contract: StormX token swap test
    ✓ revert if invalid parameters provided in constructor test (125ms)
    ✓ revert if initialize twice test
    ✓ revert if ownership is not transferred before initialize test (74ms)
    ✓ initialize success test (224ms)
    ✓ revert if transferring ownership without holding the ownership test (90ms)
    ✓ revert if transferring ownership not called by owner test (184ms)
    ✓ swap contract owner transfers ownership success test (244ms)
    ✓ revert if not enough balance in token swap test
    ✓ token swap reverts when it is not available test (74ms)
    ✓ token swap success test (95ms)
    ✓ owner and only owner can close token migration after specified time period test (307ms)
    ✓ revert if invalid reserve address is provided in disableMigration test
    ✓ revert if closing token migration when token swap is not open test (535ms)

  Contract: Token swap GSN test
    ✓ revert if initialize is called twice (83ms)
    ✓ revert if transferring old token ownership without holding the ownership test (375ms)
    ✓ transfer old token ownership fails if not enough token balance test (174ms)
    ✓ owner and only owner can transfer old token ownership test (377ms)
    ✓ convert via GSN call fails if not enough old token balance of user test (93ms)
    ✓ convert via GSN call fails if not enough unlocked new token balance of user even after conversion test (174ms)
    ✓ convert via GSN call succeeds with charging if have enough unlocked new token balance after conversion test (1283ms)
    ✓ convert via GSN call success test (668ms)
    ✓ revert if disabling token swap too early via GSN call test (82ms)
    ✓ revert if invalid parameters provided in disabling token swap via GSN call test (81ms)
    ✓ owner and only owner can disable token swap via GSN call success test (446ms)
    ✓ owner and only owner can set GSN charge test (491ms)
    ✓ reverts if invalid parameter provided in set stormx reserve address test (76ms)
    ✓ owner and only owner can set stormx reserve address test (1191ms)


  95 passing (1m)
```

Code Coverage

The contracts feature excellent test coverage.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	94.59	100	100	
StormXGSNRecipient.sol	100	80	100	100	
StormXToken.sol	100	100	100	100	
Swap.sol	100	100	100	100	
All files	**100**	**94.59**	**100**	**100**	

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

```
d87b00c4d1e25e004e3134a39901952d93771572a7d84f32bc1f8ffd5e3763c4 ./contracts/Migrations.sol
4702c6256cc84d644e5fec90da1fe0ed8a0ce19af660c2c4ce6f801989a9cb40 ./contracts/StormXToken.sol
5654854ee811cfcb1e5dcefd9b890f9b4ae1266fa0a8b8b0cc89cb604128cb3d ./contracts/Swap.sol
7029552776a09d30eb61f4d43ad438132856c97bdfd7010fb826912afd01f69d ./contracts/StormXGSNRecipient.sol
```

Tests

```
7f2e9842aa84dfb0ca2281ece03aa7e2d790a0bcd998dd5c86fa65d5d70886e2 ./test/Constants.js
bfd4054b1dd343f44d2ff81b0518d2aa89538b7fefccd7cea86ef2836495d535 ./test/StormXGSNRewarding.test.js
aa33128cdddcf6f5e9bd37fc66de07291dc8f14fd7e5f8538bbe842b6e3d8079 ./test/SwapGSNTest.test.js
01f294402df68eb4b92a88abb8135af0cb6d975ba7b2530157ecc4fa46bce5b8 ./test/StormXToken.test.js
ea82665a191090df0175fa2f101ecad7b5d1eb47a45d8cd1325b8dc80e629e8e ./test/Swap.test.js
00b7da24c0eafcfbcf69fb151d89e59bfe8ae25477d0490928aa211ee3cc1529 ./test/StormXRewarding.test.js
5bce96cdbe7aae1a069e81491df59c0d0320e7b09aeee8a939fc76affe297131 ./test/Utils.js
1565921b89d01ce27757ec34fca42808d764f5710c57c97c79eca8fbe4b6db9d ./test/StormXGSNStaking.test.js
026468731ac66312447e2c63843e77da9929646c85d1cc78fa1b5f4766c370c3 ./test/StormXGSNTest.test.js
```

Changelog

- 2020-03-24 - Initial report
- 2020-03-25 - Revised report based on commit `eb3be3c`
- 2020-03-26 - Revised report based on commit `813acda`
- 2020-04-24 - Revised report based on commit `8177534`

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.