Quantstamp Security Assessment Certificate

June 25th 2021 — Quantstamp Verified

SeleCT x StormX NFT

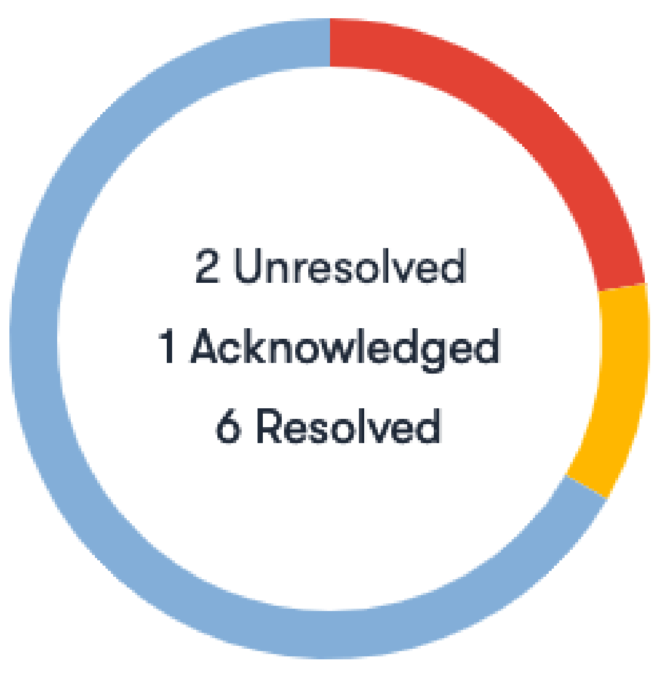
This smart contract audit was prepared by Quantstamp, the leader in blockchain security.

Executive Summary

Type	NFT Token and Initial NFT Offering
Auditors	Poming Lee, Research Engineer Joseph Xu, Technical R&D Advisor
Timeline	2021-06-16 through 2021-06-25
EVM	Muir Glacier
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Computer-Aided Verification, Manual Review
Specification	None
Documentation Quality	Low
Test Quality	Medium
Source Code	

Repository	Commit
nft-shrug	1d5d0ec
nft-shrug	03d4bc3

Total Issues	**9** (6 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	**5** (4 Resolved)
Low Risk Issues	0 (0 Resolved)
Informational Risk Issues	**4** (2 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

During auditing, we found 9 potential issues of various levels of severity: 5 medium-severity, and 4 informational-level findings. We made 3 best practices recommendations.
We highly recommend addressing the findings before going live.

2021-06-25 update: During this reaudit, StormX team has brought some of the status of findings either into fixed or acknowledged. For the others, StormX team decided not to fix them because in their opinion they are not necessary.

ID	Description	Severity	Status
QSP-1	Max supply of 500 is not enforced at the token level	^ Medium	Fixed
QSP-2	Minter can burn token on any address	^ Medium	Fixed
QSP-3	Possibly stale price feeds due to deprecated Chainlink API	^ Medium	Fixed
QSP-4	There is no backup oracle nor protection from erroneous price data	^ Medium	Unresolved
QSP-5	Dangerous external calls from `ShrugSale.sol` to arbitrary contact that are added as `recipients`	^ Medium	Fixed
QSP-6	Privileged roles and ownership	○ Informational	Unresolved
QSP-7	`TokenBought` event does not accurately reflect the NFT transactions	○ Informational	Fixed
QSP-8	Unlocked pragma	○ Informational	Fixed
QSP-9	Possible difficulty in properly managing admin privileges	○ Informational	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 v0.7.16
- Slither v0.8.0

Steps taken to run the tools:

1. `npm install solidity-coverage`
2. add `plugins: ["solidity-coverage"]` to `truffle-config.js`
3. change `truffle-config.js` to add a development network

```
development: {
            host: "127.0.0.1",     // Localhost (default: none)
            gasPrice: 100000000000,
            port: 8545,            // Standard Ethereum port (default: none)
            network_id: "5555",       // Any network (default: none)
           },
```

5. modifying the `secret.localnet.json` to have recipient[1,2]_address from the new network

```
    {
      "mnemonic": "",
      "infura_api_key": "",
      "recipient1_address": "0xb56ec59083bca56e374f25677108cb4534a474d7",
      "recipient2_address": "0xb538d7a6d7495689e2219b26c3e189e2ad3c92e7",
      "usdt_token_address": "",
      "stmx_token_address": "",
      "eth_usd_aggregator_address": "",
      "stmx_usd_aggregator_address": "",
      "usdt_usd_aggregator_address": ""
    }
```

7. comment out `L2-L3` in `truffle-config.js`
8. `ganache-cli --networkId 5555`
9. `truffle migrate --network development`
10. `truffle run coverage --network development`
11. Installed the Slither tool: `pip install slither-analyzer`
12. Run Slither from the project directory: `slither .`

Findings

QSP-1 Max supply of 500 is not enforced at the token level

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `./token/ShrugToken.sol`

Description: The max supply of 500 is enforced in the sale contract instead of at the token level. The minter address has the privilege to mint more than 500 tokens, which can be a problem if the address with the `DEFAULT_ADMIN_ROLE` or the minter address somehow becomes compromised.

Recommendation: Enforce the max supply within the token contract by tracking the token supply and max token supply, instead of within the token sale contract.

QSP-2 Minter can burn token on any address

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `./token/ShrugToken.sol`

Description: The minter address can burn any token ID that has been issued. This can be a problem if the address with the `DEFAULT_ADMIN_ROLE` or the minter address somehow becomes compromised. While a minter address can re-mint the same token ID, this may lead to a temporary loss of existing NFTs.

Recommendation: Remove the function `burn` from the token if it is not part of the intended functionality. Alternatively, only allow the owner of a specific token ID to burn it, instead of the minter address.

QSP-3 Possibly stale price feeds due to deprecated Chainlink API

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `./curves/Exponential.sol`

Description: The current conversion between ETH to USDT/STMX obtains the Chainlink price feed via `IAggregator.latestAnswer`, which is part of the deprecated API. This could lead to incorrect pricing due to stale prices.

Recommendation: Use the latest `AggregatorV3Interface` and check for liveness of the feeds using function `latestRoundData`. Develop fall-back plans (e.g., pause minting) if the price feed is stale.

QSP-4 There is no backup oracle nor protection from erroneous price data

Severity: *Medium Risk*

Status: Unresolved

File(s) affected: `contracts/curves/Exponential.sol`

Description: According to `contracts/curves/Exponential.sol`: the oracle that the current system uses does not have any backup currently. The system only collects price data from Chainlink. The system could fail to work correctly when any of the oracle aggregators is operating abnormally or being manipulated.

Recommendation: Please add more than one oracle for each pair in order to increase the security level. Also, consider adding some sanity checks to the collected price data.

Update: StormX team decided not to fix it because in their opinion they are not necessary.

QSP-5 Dangerous external calls from `ShrugSale.sol` to arbitrary contact that are added as `recipients`

Severity: Medium Risk

Status: Fixed

File(s) affected: `contracts/sale/ShrugSale.sol`

Description: Function `buyInETH`: when sending ether to the `recipients`, the `recipients` could perform re-entrancy attack to `contracts/sale/ShrugSale.sol` and this pattern could be used as a tool to conduct a complex attack.

Recommendation: Consider excluding the contract type of address as a recipient, or make sure all the critical functions such as function `buyInETH`, `buyInUSDT`, `buyInSTMX` cannot be re-entered.

QSP-6 Privileged roles and ownership

Severity: Informational

Status: Unresolved

File(s) affected: `contracts/role/MinterRole.sol`, `contracts/curves/Exponential.sol`

Description: There are some actions that could have important consequences for end-users:

1. [Fixed] The owner of `contracts/role/MinterRole.sol` can arbitrarily mint/burn any tokens at will at any point in time.
2. The owner of `contracts/role/MinterRole.sol` can arbitrarily add any role (e.g., `DEFAULT_ADMIN_ROLE` and `MINTER_ROLE`) to any address at will.
3. The owner of `contracts/curves/Exponential.sol` can change price aggregators and manipulate the price at will.

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: StormX team decided not to fix it because in their opinion they are not necessary.

QSP-7 `TokenBought` event does not accurately reflect the NFT transactions

Severity: Informational

Status: Fixed

File(s) affected: `./sale/ShrugSale.sol`

Description: In functions `buyInETH`, `buyInUSDT`, and `buyInSTMX`: buying multiple tokens would only result in one event being emitted containing only one `uint256 tokenId` representing the last token ID minted and the `uint256 value` representing the price paid for all of the tokens minted in the purchase. It's also possible to supply `_count = 0` as the argument when buying NFTs. This transaction does not revert but emits an event.

Recommendation: Expand the `TokenBought` event to include the last tokenId minted before transaction and the last tokenId minted after the transaction.

QSP-8 Unlocked pragma

Severity: Informational

Status: Fixed

File(s) affected: `All contracts`

Description: Every Solidity file specifies in the header a version number of the format pragma solidity (^)0.8.*. The caret (^) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version and above, hence the term "unlocked".

Recommendation: For consistency and to prevent unexpected behavior in the future, it is recommended to remove the caret to lock the file onto a specific Solidity version.

QSP-9 Possible difficulty in properly managing admin privileges

Severity: Informational

Status: Acknowledged

File(s) affected: `./role/MinterRole.sol`, `./libs/AccessControl.sol`

Description: The `./libs/AccessControl.sol` contract manages administrative privileges in a complicated manner with an additional admin `RoleData.adminRole` on top of each role. While the current `DEFAULT_ADMIN_ROLE = 0x00` serves as the central admin for all other roles, modifying this (or the `adminRole` for `DEFAULT_ADMIN_ROLE`) can lead to infinitely nested or cyclic admins.
It is also possible to burn the admin privileges of `DEFAULT_ADMIN_ROLE` using function `revokeRole` and `renounceRole`. It is unclear whether or not this is part of the intended specification.

Recommendation: Consider simplifying the implementation using only `mapping (bytes32 => mapping (address => bool)) private _roles;` instead of the `RoleData` struct. In addition, clarify what would happen to the contracts when `DEFAULT_ADMIN_ROLE` is lost through function `revokeRole` or `renounceRole`.

Update: StormX team decided not to fix it because in their opinion they are not necessary.

Automated Analyses

Slither

Slither has output 120 results, the majority of which have been filtered out because they were false positives. The remaining issues have been included in this report.

Adherence to Best Practices

1. `contracts/curves/Exponential.sol`: function `calculatePrice`: consider using parentheses to explicitly order the precedence of the operation.
2. `./libs/AccessControl`: function `_setRoleAdmin` is an unused internal function.
3. Duplicate lines in `./sale/ShrugSale.sol L5-L6`.

Test Results

Test Suite Results

All tests have passed.

```
Contract: ShrugSale
  Shrug Sale Setting
     ✓ setting USDT token contract is not working if the caller is not the owner (40ms)
     ✓ setting USDT token contract is working if the caller is the owner
     ✓ setting STMX token contract is not working if the caller is not the owner
     ✓ setting STMX token contract is working if the caller is the owner
     ✓ setting ETH / USD aggregator contract is not working if the caller is not the owner
     ✓ setting ETH / USD aggregator contract is working if the caller is the owner
     ✓ setting STMX / USD aggregator contract is not working if the caller is not the owner (52ms)
     ✓ setting STMX / USD aggregator contract is working if the caller is the owner
     ✓ setting USDT / USD aggregator contract is not working if the caller is not the owner
     ✓ setting USDT / USD aggregator contract is working if the caller is the owner
     ✓ setting recipients is not working if the caller is not the owner (39ms)
     ✓ setting recipients is working if the caller is the owner
     ✓ adding minter is not working if the caller is not the owner
     ✓ adding minter is working if the caller is the owner
  Price List
     ✓ ETH (7624ms)
     ✓ USDT (12712ms)
     ✓ STMX (16157ms)
  Shrug Token
     ✓ mint is not working if caller is not the sale contract (40ms)
  Sale
     ✓ buy is not working with insuffient balance (45ms)
     ✓ buy is working with correct balance (118ms)
     ✓ buy is working with correct balance (86ms)
     ✓ buy is working with insuffient USDT (148ms)
     ✓ buy is working without approval (60ms)
     ✓ buy is working with USDT (190ms)
     ✓ buy is working with insuffient USDT (124ms)
     ✓ buy is working without approval (58ms)
     ✓ buy is working with STMX (329ms)
     ✓ buy is working with STMX (3179ms)
     ✓ buy is working with STMX (3240ms)
     ✓ buy is working with USDT (2709ms)
     ✓ buy is working with USDT (2743ms)
     ✓ buy is not working if there isn't enough token (1001ms)
     ✓ buy is working with USDT (2877ms)


33 passing (55s)
```

Code Coverage

While there are already plenty of tests in the test suite, the coverage data shows that the security could improve from adding more tests, especially to the important contracts such as `ShrugSale.sol`.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
curves/	100	100	100	100	
Exponential.sol	100	100	100	100	
fake/	55.56	100	55.56	55.56	
ETHUSDAggregator.sol	50	100	50	50	9
STMX.sol	100	100	100	100	
STMXUSDAggregator.sol	50	100	50	50	9
USDT.sol	50	100	50	50	13
USDTUSDAggregator.sol	50	100	50	50	9
interfaces/	100	100	100	100	
IAggregator.sol	100	100	100	100	
IShrugToken.sol	100	100	100	100	
libs/	27.78	12.5	27.27	26.32	
AccessControl.sol	27.78	12.5	27.27	26.32	… 67,78,79,80
role/	66.67	100	66.67	72.73	
MinterRole.sol	66.67	100	66.67	72.73	29,33,37
sale/	100	66.67	100	100	
ShrugSale.sol	100	66.67	100	100	
token/	71.43	50	60	71.43	
ShrugToken.sol	71.43	50	60	71.43	41,48
All files	**78.22**	**62.96**	**63.04**	**77.88**	

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

```
4fd6092bdfa8b42f19d535c5ac69c4323b0b894717c699e58d5552eeabd04cd4 ./contracts/Migrations.sol
b7d236862a8e1f46232408822e006a06beb0879b5b8f9afbb52fd75cb52aeaa5 ./contracts/token/ShrugToken.sol
98e44654401f517ca085f9acc8fcdd4f043d2ba7f4b2138a1fe50d72fd70c6b3 ./contracts/sale/ShrugSale.sol
d239e835610ad4e5f5bc8578be78bb324a1be4e0216a2712386405b0cf8f8b89 ./contracts/role/MinterRole.sol
ee2a33a1ff422aaf8cd871e80311bc23ac99dafa8121aa82778806bbb526dfa9 ./contracts/libs/AccessControl.sol
e8ecf0414ef654a61a4aca57e7ee45007749d0c3713e62340586a9c3324bd064 ./contracts/interfaces/IAggregator.sol
bd66bd4660be8d666867b5152729978a8f9439c44ae935904fc04c46489d7c0d ./contracts/interfaces/IShrugToken.sol
7be191028d8ae5fb49630b4e89be57021471289acbfc5f844594149024c5910e ./contracts/fake/ETHUSDAggregator.sol
7a8169da4317cdf451a738ae9bf464e71f9293317144ce33a016241d6df6c4e3 ./contracts/fake/STMX.sol
adc036f585d012250078a80b8d77be5d3c67037d0a722b72295b147daa95775e ./contracts/fake/STMXUSDAggregator.sol
43dd1ec1f92d47e27f7dc77be1dedaf7d5b20512063b70f6e0972c8019f93e58 ./contracts/fake/USDT.sol
310bb99a40a3cb6cd8a8abb60bf6301f49a68c073d9b6285a4c5df7d97787916 ./contracts/fake/USDTUSDAggregator.sol
0721c14938ecee6e5d29a442ad2d8f0cf46106c57b6b909779aa3aeeeda977eb ./contracts/curves/Exponential.sol
```

Tests

```
2e1e53ca849802e7a4394ec4552c228ad3d4188491e9b2a07d266a55bcf63ee1 ./test/ShrugSale.test.js
```

Changelog

- 2021-06-18 - Initial report
- 2021-06-25 - 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.