Quantstamp Security Assessment Certificate

August 29th 2022 — Quantstamp Verified

CapsuleNFT - Dollar Store Kids and Protocol Updates

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

Executive Summary

Type	NFT
Auditors	Zeeshan Meghji, Auditing Engineer Fatemeh Heidari, Security Auditor Roman Rohleder, Research Engineer
Timeline	2022-08-10 through 2022-08-19
EVM	Gray Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Capsule Protocol Documentation
Documentation Quality	Medium
Test Quality	Medium

Source Code

Repository	Commit
capsule-contracts (initial audit)	7f00c53
capsule-add-ons (initial audit)	6bb890f
capsule-contracts (fix verification)	080902c
dollar-store-kids (fix verification)	b7b0059
dollar-store-kids (fix verification - squashed)	a842f87

Total Issues	**16** (6 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	**1** (0 Resolved)
Low Risk Issues	**6** (3 Resolved)
Informational Risk Issues	**5** (2 Resolved)
Undetermined Risk Issues	**4** (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).
Fixed	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 Capsule Protocol gives users the ability to create ERC-721 `Capsule` tokens which can contain many kinds of ERC-20, ERC-721 and ERC-1155 tokens. The CapsuleNFT team has made key updates to the protocol since the previous audit including the ability to mint `Capsule` tokens which contain ERC-1155 tokens. The CapsuleNFT team has also written a new smart contract called `DollarStoreKids` which allows users to mint tokens in a special collection of `Capsule` tokens.

During the audit, we found several important security issues which can be resolved by the CapsuleNFT team. In particular, QSP-1 could result in a user not being able to redeem their `Capsule` tokens. QSP-2 highlights incorrectly implemented validation when minting `Capsule` tokens. The vulnerability described in QSP-3 could result in an attacker taking control of the `CapsuleMinter` implementation contract.

The specification document provided for the audit was out of date and contained many discrepancies with the source code. We have highlighted these within the **Adherence to Specifications** section of the report. The code coverage of the tests can also be improved, as not all contracts have `100%` branch coverage. For instance, the branch coverage of the `DollarStoreKids` contract is `83.33%`. We also noted that there are no meaningful integration tests using a mainnet fork. We recommend testing the `Capsule` and `CapsuleMinter` contracts using real examples of ERC-20, ERC-721 and ERC-1155 tokens rather than just testing with mock token contracts.

We would like to note that the CapsuleNFT team has been cooperative during the initial phase of the audit by answering the auditing team's questions and providing additional specs as requested. We strongly recommend that the CapsuleNFT team fixes all issues mentioned in the report. In addition, the CapsuleNFT team should update their specification documents to match the current state of the source code.

Update: Following the fix verification, we determined that most of the issues have been either fixed or sufficiently acknowledged. However, QSP-1 remains unresolved, and we still recommend fixing it. QSP-11 was introduced as part of the fixes submitted by the CapsuleNFT team and has been acknowledged. We also note that none of the discrepancies mentioned in the **Adherence to Specifications** section of the report have been fixed. We recommend updating the specifications to avoid confusion among users. We further noted that no significant integration tests have been implemented as per our recommendations. We also discovered that a `README.md` file has been added to the `dollar-store-kids` repository which provides a helpful and detailed overview of the `DollarStoreKids` contract.

ID	Description	Severity	Status
QSP-1	Capsule Redemption Can Fail	^ Medium	Unresolved
QSP-2	Broken Validation for ERC-1155 Minting	⌄ Low	Fixed
QSP-3	Attacker Can Take Control of Implementation Contracts	⌄ Low	Acknowledged
QSP-4	Privileged Roles and Ownership	⌄ Low	Acknowledged
QSP-5	Can Create Capsules Which Cannot Be Redeemed	⌄ Low	Fixed
QSP-6	Tokens Can Be Permanently Locked in `CapsuleMinter`	⌄ Low	Acknowledged
QSP-7	Checks-Effects-Interactions Pattern Violations	⌄ Low	Fixed
QSP-8	`safeApprove` Is Deprecated	○ Informational	Fixed
QSP-9	Missing Event Emissions During Key State Transitions	○ Informational	Fixed
QSP-10	Events Emitted After External Calls	○ Informational	Acknowledged
QSP-11	Can Mint to Contract Not Supporting ERC-721	○ Informational	Acknowledged
QSP-12	Clone-and-Own	○ Informational	Acknowledged
QSP-13	Cannot Change Token URI of `D$K`	? Undetermined	Acknowledged
QSP-14	Token URI Not Set when Minting	? Undetermined	Fixed
QSP-15	`D$K` Can Be Burned Directly	? Undetermined	Acknowledged
QSP-16	Lack of Pause/Emergency Stop Mechanism	? Undetermined	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.

DISCLAIMER:
If the final commit hash provided by the client contains features that are not in the scope of the audit or a fix verification, those features are excluded from the consideration in this report. We also note that this audit focused exclusively on the following files:

- contracts/Capsule.sol (capsule-contracts)
- contracts/CapsuleMinter.sol (capsule-contracts)
- contracts/DollarStoreKids.sol (capsule-add-ons)

Update: Before the fix verification, the code in the `capsule-add-ons` repository was moved to the `dollar-store-kids` repository. Thus the fix verification involved reviewing the `dollar-store-kids` repository and not the `capsule-add-ons` repository. Furthermore, commit `a842f87` referenced in the source code section on the first page refers to a squashed commit for which the `DollarStoreKids` contract remains identical to the version in the fix verification commit `b7b0059`.

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 v0.8.3

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 Capsule Redemption Can Fail

Severity: *Medium Risk*

Status: Unresolved

File(s) affected: `capsule-contracts/contracts/CapsuleMinter.sol`

Description: If the transfer for any of a `Capsule` token's components fails, it will be impossible for the owner to redeem any of the tokens within the `Capsule`. The `CapsuleMinter` contract provides the following functions for redeeming `Capsule` tokens which contain multiple tokens: `burnMultiERC20Capsule`, `burnMultiERC721Capsule`, and `burnMultiERC1155Capsule`. Each of these functions will burn the `Capsule` token and transfer each of the `Capsule` token's components back to the caller. However, if the transfer for any of the `Capsule` token's components fails, the entire transaction will revert.
The `CapsuleMinter` contract allows any ERC-20, ERC-721 and ERC-1155 token to be included in a `Capsule`. Thus, there are many possible reasons why the token transfers could fail. For example, a malicious token which fails on subsequent transfers could be minted as part of a `Capsule` token. Token transfers can also fail if the token contract has been paused.
We've included the relevant parts of each function below:

- `burnMultiERC20Capsule` (`L371` to `L374`)

```
for (uint256 i; i < tokens.length; i++) {
    // send tokens back to the user
    IERC20(tokens[i]).safeTransfer(_msgSender(), amounts[i]);
}
```

- `burnMultiERC721Capsule` (`L438` to `L441`)

```
for (uint256 i; i < tokens.length; i++) {
    // send tokens back to the user
    IERC721(tokens[i]).safeTransferFrom(address(this), _msgSender(), ids[i]);
}
```

- `burnMultiERC1155Capsule` (`L507` to `L510`)

```
for (uint256 i; i < _tokens.length; i++) {
    // send tokens back to the user
    IERC1155(_tokens[i]).safeTransferFrom(address(this), _msgSender(), _ids[i], _amounts[i], "");
}
```

Recommendation:

1. Add a version of the `burnMultiERC20Capsule`, `burnMultiERC721Capsule`, and `burnMultiERC1155Capsule` functions which will allow a user to redeem the `Capsule` token even if one of the token transfers for the `Capsule` fails.
2. Add a `force` parameter to the `burnMultiERC20Capsule`, `burnMultiERC721Capsule`, and `burnMultiERC1155Capsule` functions. If the `force` parameter is set to `true`, the `Capsule` token redemption should succeed even if one of the component token transfers fail.
3. Add functionality which allows for redeeming some but not all tokens within a `Capsule` token.

Update: The CapsuleNFT team has decided not to fix this issue. Their reasoning is that a problematic token would have to be included in the `Capsule` token for this issue to occur. While this is true, we still recommend fixing the issue as any token can be included in a capsule, and token transfers could fail in unpredictable ways. If the CapsuleNFT team still chooses not to fix the issue, we recommend emphasizing to users in the documentation to be very cautious about which tokens to include in a `Capsule` token.

QSP-2 Broken Validation for ERC-1155 Minting

Severity: *Low Risk*

Status: Fixed

File(s) affected: `capsule-contracts/contracts/CapsuleMinter.sol`

Description: The `mintMultiERC1155Capsule` function of the `CapsuleMinter` contract lets a user mint a `Capsule` token containing ERC-1155 tokens. The function transfers the ERC-1155 tokens from the caller to the contract and attempts to verify the transfer from `L467` to `L471`:

```
// transfer token to contract, safeTransferFrom does check from is the owner of id
IERC1155(_token).safeTransferFrom(_msgSender(), address(this), _id, _amounts[i], "");

// check the contract owns that NFT
require(IERC1155(_token).balanceOf(address(this), _id) > 0, Errors.NOT_NFT_OWNER);
```

The function only validates that the contract has at least one token. It does not validate that the balance of the contract has increased by `_amount[i]`. If the token transfer was unsuccessful and the contract already possessed one token, then the validation check will incorrectly pass.

Recommendation: Modify the check on `L471: require(IERC1155(_token).balanceOf(address(this), _id) > 0, Errors.NOT_NFT_OWNER);` so that the function verifies that the balance of the contract has increased by `_amounts[i]` after the transfer.

Update: The CapsuleNFT team has fixed the issue by correcting the validation logic so that it ensures the expected amount of ERC-1155 tokens has been added to the contract.

QSP-3 Attacker Can Take Control of Implementation Contracts

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `capsule-contracts/contracts/CapsuleMinter.sol`

Description: Upgradeability often involves two different kinds of contracts: implementation contracts and proxies. `CapsuleMinter` is an implementation contract and it is currently not automatically initialized. If an implementation contract is not initialized, a malicious user could call the `initialize` function on the implementation contract to take ownership of it. If the implementation contract uses delegate calls, it may be possible for an attacker to self-destruct the contract.

Recommendation: Add a constructor to the `CapsuleMinter` contract which calls `_disableInitializers()` to automatically initialize the contract and block a malicious user from taking control of it.

Update: The CapsuleNFT team will manually initialize the implementation contracts immediately after deployment as a mitigation strategy.

QSP-4 Privileged Roles and Ownership

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `capsule-contracts/contracts/Capsule.sol`, `capsule-contracts/contracts/CapsuleMinter.sol`, `capsule-add-ons/contracts/DollarStoreKids.sol`

Description: Smart contracts will often have roles to designate a user with special privileges to make modifications to the smart contract. We have listed all instances of privileged roles and their permissible actions within each contract:

- `Capsule`
  - `owner`
    - `lockCollectionCount`: Set the maximum number of tokens for the collection.
    - `renounceOwnership`: Renounce ownership of the collection. The owner cannot be reset. Once this function is called, `lockCollectionCount`, `updateRoyaltyConfig` and `transferOwnership` will no longer be callable.
    - `updateRoyaltyConfig`: Set the royalty recipient and any royalty rate up to `100%` for the collection.
    - `transferOwnership`: Change the `owner` to a different address.
  - `tokenURIOwner`
    - `setBaseURI`: Set the base URI for the collection.
    - `setTokenURI`: Set the token URI for any token in the collection.
    - `updateTokenURIOwner`: Change the `tokenURIOwner` to a different address or renounce it by setting it to the zero address.
- `CapulesMinter`
  - `governor`
    - `addToWhitelist`: Add an address to the whitelist. This will allow that address to mint `Capsule` tokens without paying any fees.
    - `flushTaxAmount`: Transfer the ETH balance of the contract to the `taxCollector`.
    - `removeFromWhitelist`: Removes an address from the whitelist. The address will then have to pay fees for minting `Capsule` tokens.
    - `updateCapsuleMintTax`: Changes the fee for minting a `Capsule` token.
  - `taxCollector`
    - `flushTaxAmount`: Transfer the ETH balance of the contract to the `taxCollector`.
- `DollarStoreKids`
  - `governor`
    - `sweep`: Transfer any ERC-20 token from the contract to the `governor`.
    - `toggleMint`: Enables or disables minting of `D$K` tokens.
    - `transferCollectionOwnership`: Transfer ownership of the `D$K` collection to another address.
    - `updateMetamaster`: Updates the `tokenURIOwner` of the `D$K` collection to another address.
    - `updateBaseURI`: Update the base URI of the `D$K` collection.
    - `updateRoyaltyConfig`: Set the royalty recipient and any royalty rate up to `100%` for the `D$K` collection.

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. Ideally, all privileged actions should be done through a timelock contract to allow users to opt out of the protocol if the changes are undesirable. Privileged roles should also be secured by large multi-signature wallets or by a decentralized governance process.

Update: The CapsuleNFT team will explain all privileged roles in user-facing documentation. The team also indicated that many of the privileged roles are owned by users who have created a `Capsule` token collection.

QSP-5 Can Create Capsules Which Cannot Be Redeemed

Severity: *Low Risk*

Status: Fixed

File(s) affected: `capsule-contracts/contracts/CapsuleMinter.sol`

Description: In order to redeem a `singleERC20Capsule` token, the stored token amount must be greater than `0` as seen on `L248:require(_capsuleData.tokenAmount > 0, Errors.NOT_ERC20_CAPSULE_ID);` of the `mintSingleERC20Capsule` function in the `CapsuleMinter` contract. However, it is possible to mint a `singleERC20Capsule` with a stored amount of `0`. As seen above, this token would be impossible to redeem.
The `mintSingleERC20Capsule` function validates that the `_amount` parameter is greater than `0` on `L221:require(_amount > 0, Errors.INVALID_TOKEN_AMOUNT);`. However, it does not validate that the `_actualAmount` transferred to the contract is greater than 0. It is the `_actualAmount` that is stored on the contract and not the `_amount` as seen on `L234:singleERC20Capsule[_capsule][_capsuleId].tokenAmount = _actualAmount;`.

Recommendation: Within the `mintSingleERC20Capsule` function, validate that the `_actualAmount` of tokens transferred to the contract is greater than `0`.

Update: The CapsuleNFT team has fixed the issue by validating that the amount of tokens transferred exceeds `0`.

QSP-6 Tokens Can Be Permanently Locked in `CapsuleMinter`

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `capsule-contracts/contracts/CapsuleMinter.sol`

Description: Any `ERC-721` and `ERC-1155` tokens can be directly transferred to the `CapsuleMinter` contract. Once these tokens are directly transferred to the contract, there will be no way to retrieve them. We can see that the `onERC721Received` function from `L175` to `L183` below accepts all ERC-721 tokens without exception:

```
function onERC721Received(
    address,
    address,
    uint256,
    bytes calldata
) external pure override returns (bytes4) {
    // `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`
    return 0x150b7a02;
}
```

Since `CapsuleMinter` inherits from `ERC1155Holder`, it similarly accepts any ERC-1155 token without exception. We can see this from the source code of `ERC1155Holder` below:

```
function onERC1155Received(
    address,
    address,
    uint256,
    uint256,
    bytes memory
) public virtual override returns (bytes4) {
    return this.onERC1155Received.selector;
}

function onERC1155BatchReceived(
    address,
    address,
    uint256[] memory,
    uint256[] memory,
    bytes memory
) public virtual override returns (bytes4) {
    return this.onERC1155BatchReceived.selector;
}
```

Recommendation: Implement the `onERC721Received`, `onERC1155Received` and `onERC1155BatchReceived` functions so that they revert outside of calls to any of the burn functions. This can be achieved via the addition of a flag which is toggled to a specific value for the duration of the burn functions and reversed when they end. The flag just needs to be checked within the `onERC721Recevied`, `onERC1155Received` and `onERC1155BatchReceived` functions.

Update: The CapsuleNFT team will update their user-facing documentation to inform users that they should not send tokens directly to the contract.

QSP-7 Checks-Effects-Interactions Pattern Violations

Severity: *Low Risk*

Status: Fixed

File(s) affected: `capsule-contracts/contracts/Capsule.sol`

Description: The checks-effects-interactions coding pattern is meant to mitigate any chance of other contracts manipulating the state of the blockchain in unexpected and possibly malicious ways before control is returned to the original contract. As the name implied, only after checking whether appropriate conditions are met and acting internally on those conditions should any external calls to, or interactions with, other contracts be done. We have listed all violations of the check-effects-interactions pattern below:

- `Capsule`
  - `mint`: An external call can be made by `_safeMint` on `L124`. State changes are made after that on `L126` and `L128`.
  - `renounceOwnership`: State changes are made on `L182` after external calls.
  - `transferOwnership`: State changes are made on `L188` after external calls.

Recommendation: Follow the checks-effects-interactions pattern by making all state changes before external calls.

Update: The CapsuleNFT team has fixed the issue by modifying the mentioned functions so that the state is always updated before making external calls. However, we note that they replaced the `_safeMint` call with `_mint`. This could result in a `Capsule` token being minted to a contract which is not designed to handle ERC-721 tokens.

QSP-8 `safeApprove` Is Deprecated

Severity: *Informational*

Status: Fixed

File(s) affected: `capsule-add-ons/contracts/DollarStoreKids.sol`

Description: `safeApprove` is a deprecated function within OpenZeppelin's `SafeERC20Upgradeable` library. The deprecation notice can be seen in the source code. `safeApprove` is still vulnerable to the ERC-20 double-spend exploit. There is one instance of a `safeApprove` call in the `DollarStoreKids` contract on `L47:IERC20(USDC).safeApprove(address(CAPSULE_MINTER), MAX_DSK * ONE_DOLLAR);`.

Recommendation: Replace calls to `safeApprove` with `safeIncreaseAllowance`.

Update: The CapsuleNFT team has fixed the issue by replacing the call to `safeApprove` with `approve`. While `approve` is generally vulnerable to the ERC-20 double-spend exploit, it is unlikely to occur within the current context.

QSP-9 Missing Event Emissions During Key State Transitions

Severity: *Informational*

Status: Fixed

File(s) affected: `capsule-contracts/contracts/Capsule.sol`, `capsule-add-ons/contracts/DollarStoreKids.sol`

Description: In order to validate the proper deployment and initialization of the contracts, it is a good practice to emit events. Also, any important state transitions can be logged, which is beneficial for monitoring the contract and also tracking eventual bugs or hacks. Emitting events also allows off-chain applications to dynamically respond to important state transitions. Below we present a non-exhaustive list of events that could be emitted to improve application management:

- `Capsule`
  - `lockCollectionCount`: Emit an event which represents the locking of the collection.
- `DollarStoreKids`
  - `toggleMint`: Emit an event which represents whether minting has been enabled or disabled

Recommendation: Emit the suggested events during important state transitions.

Update: The CapsuleNFT team has fixed the issue by emitting all the recommended events.

QSP-10 Events Emitted After External Calls

Severity: *Informational*

Status: Acknowledged

File(s) affected: `capsule-add-ons/contracts/DollarStoreKids.sol`

Description: There are multiple events which are emitted after external calls. If these external calls lead to the same function being reentered, then the events will be emitted in an incorrect order. We have listed all such event emissions which occur after external calls below:

- `DollarStoreKids`
  - `L69:emit DollarStoreKidsMinted(_caller, _counter);`
  - `L84:emit DollarStoreKidsBurnt(_caller, id_);`

Recommendation: Emit the listed events before all external calls are made.

Update: The CapsuleNFT team has decided not to fix the issue since they believe it will not cause any problems for the `DollarStoreKids` contract

QSP-11 Can Mint to Contract Not Supporting ERC-721

Severity: *Informational*

Status: Acknowledged

File(s) affected: `capsule-contracts/contracts/CapsuleMinter.sol`

Description: The `mint` function of the `Capsule` contracts mints a token to any address `_account` passed by the caller. It is possible for `_account` to be the address of a contract which cannot correctly handle ERC-721 tokens. This may result in the token being locked within that contract forever. Furthermore, the `mint` function does not call the `onERC721Recevied` function of the contract receiving the minted token. This issue was introduced as part of the fix verification changes.

Recommendation: If the `_account` being minted to is a contract, call the `onERC721Received` function on the contract and verify that the correct function signature is returned.

Update: The CapsuleNFT team has opted not to call `onERC721Received` to reduce gas costs. Users are expected to carefully implement any receiving contracts to handle `Capsule` tokens.

QSP-12 Clone-and-Own

Severity: *Informational*

Status: Acknowledged

Description: The clone-and-own approach involves copying and adjusting open source code at one's own discretion. From the development perspective, it is initially beneficial as it reduces the amount of effort. However, from the security perspective, it involves some risks as the code may not follow the best practices, may contain a security vulnerability or may include intentionally or unintentionally modified upstream libraries. We note that all contracts within the `contracts/openzeppelin` folder of the `capsule-contracts` have been cloned locally.

Recommendation: Rather than the clone-and-own approach, a good industry practice is to use a package manager (e.g., npm) for handling library dependencies. This eliminates the clone-and-own risks yet allows for following best practices, such as using libraries. If the file is cloned anyway, a comment including the repository, commit hash of the version cloned, and the summary of modifications (if any) should be added. This helps to improve the traceability of the file.

If changes to storage layout are a concern as voiced in a previous audit, consider locking the desired dependencies in `packages.json`.

Update: The CapsuleNFT team uses the clone-and-own approach to avoid any potential updates to the storage layout of dependent contracts.

QSP-13 Cannot Change Token URI of `D$K`

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `capsule-add-ons/contracts/DollarStoreKids.sol`

Description: While the `DollarStoreKids` contract provides the function `updateBaseURI` to update the base URI of the `D$K` collection, there is no function provided to change the token URI of an individual token. As a result, there is no way to change the token URI of any individual token within the `D$K` collection unless the `updateMetamaster` function is called to transfer the `tokenURIOwner` role to another address.

Recommendation: Add a function to the `DollarStoreKids` contract to allow the `governor` to change the token URI of individual tokens within the `D$K` collection.

Update: The CapsuleNFT team has indicated that it is unnecessary to set the token URL of each individual `D$K` token. The collection will be managed by setting the base URL.

QSP-14 Token URI Not Set when Minting

Severity: *Undetermined*

Status: Fixed

File(s) affected: `capsule-contracts/contracts/CapsuleMinter.sol`

Description: The `mint` function on the `Capsule` contract allows the minting of a new `Capsule` token within the collection. One of the parameters of function is `_uri`. However, if the `baseURI` of the collection has already been set, the `_uri` parameter is ignored and the token URI for the minted token is not set as seen from `L125` to `L127` below:

```
if (bytes(baseURI).length == 0) {
    _setTokenURI(counter, _uri);
}
```

Recommendation: Either the `mint` function should be modified to set the token URI properly or the code should be documented with why the token URI is not set.

Update: The CapsuleNFT team has fixed the issue by adding a code comment to explicitly indicate that the token URI should not be set if the base URI has already been set.

QSP-15 `D$K` Can Be Burned Directly

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `capsule-contracts/contracts/CapsuleMinter.sol`, `capsule-add-ons/contracts/DollarStoreKids.sol`

Description: The `DollarStoreKids` contract provides a `burn` function which allows a `D$K` owner to burn their `D$K` token and redeem it for one USDC token. However, it is also possible for the `D$K` owner to burn their token by calling the `burnSingleERC20Capsule` function directly on the `CapsuleMinter` contract.

Recommendation: If it is desirable to allow a user to redeem a `D$K` token through the `CapsuleMinter` contract directly, consider removing the `burn` function from the `DollarStoreKids` contract. Otherwise, implement a mechanism to prevent users from redeeming their `D$K` by directly calling `burnSingleERC20Capsule` on the `CapsuleMinter` contract.

Update: The CapsuleNFT team indicated that it is acceptable to burn `D$K` tokens through both methods.

QSP-16 Lack of Pause/Emergency Stop Mechanism

Severity: *Undetermined*

Status: Acknowledged

Description: Many protocols implement a pause or emergency stop mechanism to mitigate the impact when a protocol is compromised. These mechanisms may prevent attackers from withdrawing tokens from the protocol or prevent them from taking certain critical actions. Currently, the Capsule Protocol does not implement such a mechanism.

Recommendation: Consider implementing a pause mechanism which can be activated during emergencies. This mechanism could block attackers from exploiting the protocol by blocking critical actions such as redeeming and minting tokens.

Update: The CapsuleNFT team indicated that they do not want to add additional centralization to the protocol by adding a pause or emergency stop mechanism. They recommend that users implement their own pause mechanisms in contracts which integrate with the Capsule Protocol.

Automated Analyses

Slither

Slither was used to analyze both repositories. Most issues found by the Slither analysis were false positives and the rest have been included in this report.

Adherence to Specification

1. `Capsule.lockCollectionCount()`:
    1. There is an incorrect parameter name `_name` for parameter `_nftCount` in the specification.
    2. Consider adding to the specification that the function may only be called once and only if `_nftCount` does not exceed the already minted `counter` value.
    3. Consider adding that only the `owner` role can call this function.

2. `Capsule.transferOwnership()`:
    1. There is an incomplete description of the function which trails with `The new owner of the Capsule will be able to` in the specification.
    2. There is an incorrect function keyword `virtual` which should be `override` in the specification.
    3. There is an incorrect function parameter name `_account` which should be `_newOwner` in the specification.
    4. The specification states that the parameter `can also be the zero address`. However, this is wrong as the called inherited function performs the following check `require(newOwner != address(0), "Ownable: new owner is the zero address");`.
    5. Consider adding that only the `owner` role can call this function.

3. `Capsule.tokenURI()`: There is a typo in the parameter name `_tokenId` which should be `tokenId` in the specification.
4. `Capsule.setTokenURI()`: The specification states that the parameter `_tokenId` is `the id of the NFT to burn`. However, the NFT for the given id is not burnt. Instead, its token URI is changed.
5. `Capsule.updateTokenURIOwner()`: There is an incorrect function parameter name `_account`, which should be `_newTokenURIOwner` in the specification.
6. `Capsule.mint()`: Consider adding to the specification that only up to `maxId` NFTs can be minted.
7. `Capsule.burn()`: The specification states that parameter `_account` is `the account that is burning the NFT`. However, more precisely it is the address of the NFT holder, as the account performing the burn is `capsuleMinter`.
8. Consider renaming `Key Constants` in the specification to `Key Constants and Variables`, as it also lists mutable variables.
9. The specification for the `Capsule.sol` contract is missing documentation for the following functions, variables, events and modifiers:
    1. `royaltyInfo()`
    2. `setBaseURI()`
    3. `updateRoyaltyConfig()`
    4. `renounceOwnership()`
    5. `royaltyInfo()`
    6. `royaltyReceiver`
    7. `royaltyRate`
    8. `event RoyaltyConfigUpdated()`
    9. `modifier onlyTokenURIOwner()`

10. The specification for `CapsuleMinter.sol` states that deflationary tokens are prohibited because `the Capsule Protocol performs a check that the exact amount of token deposited is found at the contract post-transfer`.
    1. However, while the contract does take only into account the actually deposited amount instead of the stated one, it does not cross-check them. Consider adjusting the specification or adding a corresponding check which fails when encountering a mismatch.
    2. Further, the documentation contradicts itself a few sentences later: `NOTE: The Capsule Protocol supports deflationary tokens`. Consider adjusting one of the sentences (and the code) accordingly.

11. The specification for `CapsuleMinter.mintSingleERC20Capsule()` is missing parameter `_receiver` and its description in the `Parameters` listing.
12. The specification for `CapsuleMinter.mintSingleERC721Capsule()` is missing parameter `_receiver` and its description in the `Parameters` listing.
13. The specification for `CapsuleMinter.mintMultiERC20Capsule()` is missing parameter `_receiver` and its description in the `Parameters` listing and in the listed example.
14. The specification for `CapsuleMinter.mintMultiERC721Capsule()` is missing parameter `_receiver` and its description in the `Parameters` listing and in the listed example.
15. There is a mismatched parameter in the specification for parameter `uint256 capsuleId` in the following events:
    1. `CapsuleMinter.SimpleCapsuleMinted()`
    2. `CapsuleMinter.SimpleCapsuleBurnt()`
    3. `CapsuleMinter.SingleERC20CapsuleMinted()`
    4. `CapsuleMinter.SingleERC20CapsuleBurnt()`
    5. `CapsuleMinter.SingleERC721CapsuleMinted()`
    6. `CapsuleMinter.SingleERC721CapsuleBurnt()`
    7. `CapsuleMinter.MultiERC20CapsuleMinted()`
    8. `CapsuleMinter.MultiERC20CapsuleBurnt()`
    9. `CapsuleMinter.MultiERC721CapsuleMinted()`
    10. `CapsuleMinter.MultiERC721CapsuleBurnt()`
16. The specification for the `CapsuleMinter.sol` contract is missing documentation for the following:
    1. `mintMultiERC1155Capsule()`
    2. `burnMultiERC1155Capsule()`
    3. `event AddedToWhitelist()`
    4. `event RemovedFromWhitelist()`
    5. `event FlushedTaxAmount()`
    6. `event MultiERC1155CapsuleMinted()`
    7. `event MultiERC1155CapsuleBurnt()`
17. Consider adding the following to the specification for the `CapsuleMinter.sol`:
    1. Modifiers `onlyCollectionMinter` and `checkStatus` are further constraining:
        1. `mintSimpleCapsule()`
        2. `mintSingleERC20Capsule()`
        3. `mintSingleERC721Capsule()`
        4. `mintMultiERC20Capsule()`
        5. `mintMultiERC721Capsule()`
        6. `mintMultiERC1155Capsule()`
    2. The modifier `onlyValidCapsuleCollections` further constrains the following functions:
        1. `burnSimpleCapsule()`
        2. `burnSingleERC20Capsule()`
        3. `burnSingleERC721Capsule()`
        4. `burnMultiERC20Capsule()`
        5. `burnMultiERC721Capsule()`
        6. `burnMultiERC1155Capsule()`

Code Documentation

1. Many functions and events are not documented using code comments. We recommend documenting all events as well as all `public` and `external` functions using the NatSpec standard.
2. `L303` of `CapsuleMinter.sol` states `get the amount of tokens held by the Capsule NFT id`. However, the subsequent line `L304` is getting the underlying NFT token id. (**Update:** Fixed)
3. `L394` of `CapsuleMinter.sol` states `that the token address and amount is mapped to the same index`. However, it should state `that the token address and id are mapped to the same index`. (**Update:** Fixed)
4. `L395` of `CapsuleMinter.sol` states that `the user is sending _amounts[0] of _tokens[0]`. However, it should state that `the user is sending _ids[0] of _tokens[0]`. (**Update:** Fixed)
5. The comment on `L470` of `CapsuleMinter.sol` states `// check the contract owns that NFT`. However, since the subsequent line is checking ownership of an ERC1155 token, the comment should be corrected to state `// check that this contract owns the ERC-1155 token`. (**Update:** Fixed)

Adherence to Best Practices

1. `VERSION` on `L13` of `contracts/Capsule.sol` should be in mixed case since it is not a constant.
2. `VERSION` on `L13` of `contracts/Capsule.sol` can be marked as `immutable`.

3. `CapsuleMinter` is responsible for both minting and redeeming `Capsule` tokens. Consider giving it a more general name such as `CapsuleManager`.
4. The `checkStatus` modifier in the `CapsuleMinter` contract checks whether the caller has provided the minting fee. Consider renaming the modifier to `feeProvided` or `checkTaxRequirement`. (**Update:** Fixed)
5. The `mintSingleERC20Capsule` of the `CapsuleMinter` contract currently emits a `SingleERC20CapsuleMinted` event which has an `amount` parameter. Currently, the `amount` function parameter the user provided is being passed as the parameter for the event. Consider passing to the event the `_actualAmount` transferred to the contract instead. (**Update:** Fixed)
6. The `mintMultiERC20Capsule` of the `CapsuleMinter` contract currently emits a `MultiERC20CapsuleMinted` event which has an `amounts` parameter. Currently, the `_amounts` function parameter the user provided is being passed as the parameter for the event. Consider passing to the event the `_actualAmounts` transferred to the contract instead. (**Update:** Fixed)
7. Consider using an `unchecked` block within for loops just to increment the counter to save gas.
8. To improve readability and lower the risk of introducing errors when making code changes, it is advised to not use magic constants throughout code, but instead to declare them once as named constants and use these named constants instead. The following instances should therefore be changed accordingly:
    - `L99` of `Capsule.sol`: `10000` (Consider reusing `MAX_BPS`). (**Update:** Fixed)
9. Do not hardcode `USDC`, `CAPSULE_FACTORY`, and `CAPSULE_MINTER` in the `DollarStoreKids` contract. Instead, the addresses of these contracts should be passed to the constructor.

Test Results

Test Suite Results

The tests were executed by running `npx hardhat test`. We noted that only mock token contracts were used when testing `Capsule` and `CapsuleMinter`. We suggest writing integration tests using a mainnet fork to test with real examples of ERC-20, ERC-721 and ERC-1155 tokens.

Update: No additional tests were added during the fix verification phase.

```
Capsule NFT tests
   ✓ Should revert on mint() if caller is minter
   ✓ Should lock collection at 4 (153ms)
   ✓ Should revert if tries to lock at count less than counter (69ms)
   ✓ Should revert if tries to lock at 0 NFT count
   ✓ Should revert if tries to lock twice
   ✓ Should renounce ownership (67ms)
  TokenURIOwner
     ✓ Should allow current owner to update tokenURIOwner
     ✓ Should revert on updateTokenURIOwner if caller is not tokenURIOwner
     ✓ Should allow current owner to set tokenURI (54ms)
     ✓ Should revert setTokenURI call if caller is not tokenURIOwner (42ms)
    set baseURI
       ✓ Should allow tokenURIOwner to set baseURI
       ✓ Should revert if non tokenURIOwner try to set baseURI
       ✓ Should ignore tokenURI input if baseURI is set (45ms)
       ✓ Should use tokenURI input if baseURI is not set
       ✓ Should return incorrect tokenURI if baseURI is set after minting (44ms)
       ✓ Should return proper tokenURI if corrected by tokenURIOwner (67ms)
  Royalty
     ✓ Should allow owner to update royalty config
     ✓ Should revert if rate is too high
     ✓ Should revert if receiver is null
     ✓ Should be able to get royalty info

Capsule Factory tests
   ✓ Should verify factory initialization
  Whitelist
    Add to whitelist
       ✓ Should add address to whitelist
       ✓ Should revert on invalid add calls (94ms)
    Remove from whitelist
       ✓ Should remove address to whitelist (44ms)
       ✓ Should revert on invalid remove calls (55ms)
  Blacklist
    Add to blacklist
       ✓ Should add address to blacklist
       ✓ Should revert on invalid updates (89ms)
    Remove from blacklist
       ✓ Should remove address to blacklist (47ms)
       ✓ Should revert on invalid remove calls (53ms)
  Update tax collector by governor
     ✓ Should update tax collector
     ✓ Should revert on invalid update (51ms)
  Update capsule creation tax
     ✓ Should update capsule creation tax
     ✓ Should revert on invalid update (56ms)
  Update Capsule collection owner
     ✓ Should update capsule owner at ownership transfer (83ms)
     ✓ Should not allow direct calls to updateCapsuleCollectionOwner (136ms)
  Flush tax amount
     ✓ Should allow governor to flush tax amount (64ms)
     ✓ Should allow tax collector to flush tax amount (66ms)
     ✓ Should allow only authorized to call flushTaxAmount
  Update Capsule minter
     ✓ Should verify that CapsuleMinter can be updated only once (109ms)
     ✓ Should revert it setting address zero as minter
  Capsule creation
     ✓ Should create capsule (39ms)
     ✓ Should fail capsule creation if incorrect tax sent
     ✓ Should verify whitelisted can create capsule without tax (50ms)
     ✓ Should verify Capsule configuration

CapsuleFactory proxy tests
   ✓ Should upgrade Capsule factory proxy (88ms)

Capsule Minter tests
   ✓ Should verify Capsule Minter is created properly
  Whitelist
    Add to whitelist
       ✓ Should add address to whitelist
       ✓ Should revert on invalid add calls (55ms)
    Remove from whitelist
       ✓ Should remove address to whitelist (45ms)
       ✓ Should revert on invalid remove calls (56ms)
  Flush tax amount
     ✓ Should allow governor to flush tax amount (61ms)
     ✓ Should allow tax collector to flush tax amount (64ms)
     ✓ Should allow only authorized to call flushTaxAmount
  Update Capsule mint tax
     ✓ Should update Capsule mint tax
     ✓ Should revert on invalid update (54ms)
  Simple Capsule
    Mint simple Capsule
```

```
      ✓ Should fail when try to mint non capsule NFT
      ✓ Should be able to mint simple capsule (44ms)
      ✓ Should verify whitelisted can create simple capsule without tax (47ms)
      ✓ Should show correct nft counter after 2 capsules are minted (57ms)
      ✓ Should set correct token uri (58ms)
    Mint simple capsule for private collection
      ✓ Should verify that collection is private
      ✓ Should allow owner to mint capsule
      ✓ Should fail when non owner try to mint capsule
    Burn simple Capsule
      ✓ Should revert when burning non capsule NFT
      ✓ Should revert when burning NFT of other user
      ✓ Should revert when burning other type of Capsule (125ms)
      ✓ Should burn simple capsule
  ERC20 Capsule
    Mint ERC20 Capsule
      ✓ Should revert if passing invalid params (38ms)
      ✓ Should mint ERC20 Capsule NFT (47ms)
    Burn ERC20 Capsule
      ✓ Should burn ERC20 Capsule NFT (39ms)
      ✓ Should revert if invalid id is being burnt
      ✓ Should revert if burning some other user's id (91ms)
  Multi ERC20 Capsule
    Mint multi ERC20 Capsule
      ✓ Should mint multi ERC20 Capsule NFT (63ms)
      ✓ Should revert when minting with empty token and amount array
      ✓ Should revert when minting with empty token array
      ✓ Should revert when minting with empty amount array
      ✓ Should revert when minting with 101 addresses in token array
      ✓ Should revert when minting with 101 amounts in amount array
      ✓ Should revert when token array and amount array length mismatch
      ✓ Should revert when token amount is zero
      ✓ Should revert when token address is zero
    Burn multi ERC20 Capsule
      ✓ Should burn multi ERC20 Capsule NFT (48ms)
      ✓ Should revert if burning invalid id
      ✓ Should revert when burning other user's id (136ms)
  ERC721 Capsule
    Mint ERC721 Capsule
      ✓ Should mint ERC721 Capsule NFT (41ms)
      ✓ Should revert if caller is not owner of ERC721 (61ms)
    Burn ERC721 Capsule
      ✓ Should burn ERC721 Capsule NFT (38ms)
      ✓ Should revert if burning other user's id (90ms)
      ✓ Should revert if burning invalid id
  Multi ERC721 Capsule
    Mint multi ERC721 Capsule
      ✓ Should mint multi ERC721 Capsule NFT (53ms)
      ✓ Should revert when minting with empty token array
      ✓ Should revert when minting with empty ids array
      ✓ Should revert when minting with 101 addresses in token array
      ✓ Should revert when minting with 101 ids in id array
      ✓ Should revert when token array and amount array length mismatch
      ✓ Should revert when token address is zero
    Burn multi ERC721 Capsule
      ✓ Should burn multi ERC721 Capsule NFT (45ms)
      ✓ Should revert if burning invalid id
      ✓ Should revert when burning other user's id (151ms)
  Multi ERC1155 Capsule
    Mint multi ERC1155 Capsule
      ✓ Should mint multi ERC1155 Capsule NFT (53ms)
      ✓ Should revert when minting with empty token array
      ✓ Should revert when minting with array with different length
      ✓ Should revert when minting with 101 addresses in token array
      ✓ Should revert when input array length doesn't match
      ✓ Should revert when token address is zero
    Burn multi ERC1155 Capsule
      ✓ Should burn multi ERC1155 Capsule NFT (45ms)
      ✓ Should revert if burning invalid id
      ✓ Should revert when burning other user's id (146ms)

CapsuleMinter proxy tests
   ✓ Should upgrade Capsule Minter proxy (104ms)
  ERC20 Capsule
     ✓ Mint ERC20 Capsule, Upgrade Minter and Burn ERC20 Capsule (141ms)


110 passing (28s)

Dollar Store Kids tests
  Verify deployment
     ✓ Should verify DSK deployed correctly
  Mint status
     ✓ Should revert if non governor toggle mint status
     ✓ Should toggle mint status (48ms)
  Mint DSK
     ✓ Should revert if minting is not allowed (41ms)
     ✓ Should revert when mint tax is not sent (143ms)
     ✓ Should revert when there are no USDC in contract (239ms)
     ✓ Should mint DSK (221ms)
     ✓ Should verify capsule data after DSK minting (174ms)
     ✓ Should revert when same address minting again (196ms)
  Burn DSK
     ✓ should burn DSK (247ms)
     ✓ should verify USDC balance after burning DSK  (71ms)
  Transfer collection ownership
     ✓ Should revert if non governor user call transfer ownership
     ✓ Should transfer collection ownership of DSK collection (754ms)
  Update MetaMaster
     ✓ Should revert if non governor user call update meta master
     ✓ Should update meta master of DSK collection
  Update baseURI
     ✓ Should revert if non governor user call updateBaseURI
     ✓ Should update baseURI of DSK collection
  Royalty
     ✓ Should allow governor to update royalty config
     ✓ Should revert if non governor calls update
     ✓ Should be able to get royalty info
  Sweep tokens
     ✓ Should sweep DAI from DSK (356ms)


21 passing (29s)
```

Code Coverage

The code coverage results were obtained by running `npx hardhat coverage`. Both the `DollarStoreKids` and `CapsuleMinter` contracts have a branch coverage which is lower than `100%`. We recommend raising the branch coverage of both contracts to `100%`.

Update: No additional tests were added during the fix verification phase.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	99.15	94.2	96.97	99.19	
Capsule.sol	95.45	100	90.48	95.74	71,109
CapsuleFactory.sol	100	95.45	100	100	
CapsuleFactoryStorage.sol	100	100	100	100	
CapsuleMinter.sol	100	91.43	100	100	
CapsuleMinterStorage.sol	100	100	100	100	
Errors.sol	100	100	100	100	
contracts/access/	53.85	33.33	60	57.14	
Governable.sol	53.85	33.33	60	57.14	... 64,65,66,67
All files	**96.76**	**91.67**	**94.37**	**96.92**	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	83.33	100	100	
DollarStoreKids.sol	100	83.33	100	100	
All files	**100**	**83.33**	**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

```
e1fd224666946547d8cd11462428573d16ff48b22467c5376b4ca516344fcb7a ./DollarStoreKids.sol

016b3d99cf80491fbe0ed519ef4f91a8f847a73ea350156c5d09b12c814755f0 ./capsule_nft-capsule-contracts-080902c39aec3619052156620c761ad31c4c7cd7-github 2/contracts/CapsuleMinter.sol

16289919f54284cfd5b1f07d1a4bbe0a6c42a2e622fccce655a3a34a3ad8f825 ./capsule_nft-capsule-contracts-080902c39aec3619052156620c761ad31c4c7cd7-github 2/contracts/Capsule.sol
```

Tests

```
108a91cc630d52e201d80960a30b8d96825f120bfe6ffa942a4f3fcfce3424aa ./Capsule.test.ts

dffcac57c691212199e89140464798969f56c93d86d3f25acf6e701dd4f0b6cd ./CapsuleFactory.test.ts

0b6ffd406dc84e7ce6a023b79ae4849cba633f31e9c5ac8b3e1bb603b7a14f4f ./CapsuleFactoryProxy.test.ts

1c119ce9311da2605d5aeac35ca844735196ea8f63fb21181c643ad2f04a96d5 ./CapsuleMinter.test.ts

49efb2fdf69160d341723647cb8751768b1b645f65fcadc6ae313175e341e407 ./CapsuleMinterProxy.test.ts

18079cc9112bdb8a72f38465b2cfd5fb7974b12fd2326d77385f3a3c4440c7d3 ./test/DollarStoreKids.spec.ts
```

Changelog

- 2022-08-19 - Initial report
- 2022-08-29 - Fix verification

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.