Quantstamp Security Assessment Certificate

July 6th 2020 — Quantstamp Verified

Cryptograph

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

Executive Summary

Type	Crypto Collectible and Auction Contracts
Auditors	Poming Lee, Research Engineer Ed Zulkoski, Senior Security Engineer Kevin Feng, Software Engineer
Timeline	2019-09-30 through 2020-07-06
EVM	Muir Glacier
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	README Cryptograph White Paper v4.1

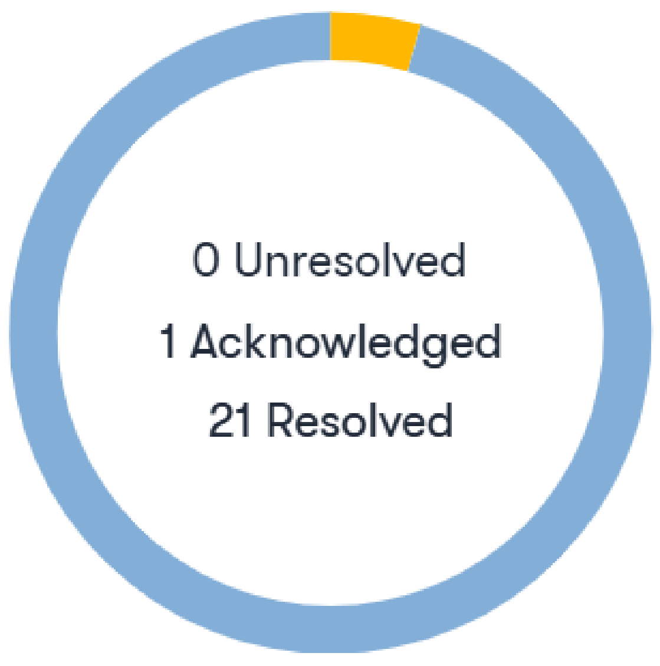
Source Code

Repository	Commit
Cryptograph_Audit	96acd40
Cryptograph_Audit	813b1a9
Cryptograph_Audit	fbb4013
Cryptograph_Audit	daafe66
Cryptograph_Audit	d61ecfb
Cryptograph_Audit	05650e3
Cryptograph_Audit	f000b29

Changelog

- 2019-10-14 - Initial report (96acd40)
- 2019-10-18 - Second Report (813b1a9)
- 2019-10-24 - Third Report (fbb4013)
- 2019-10-29 - Fourth Report (daafe66)
- 2019-10-31 - Final Report (d61ecfb)
- 2020-07-03 - Initial report for the diff audit (05650e3)

Total Issues	22	(21 Resolved)
High Risk Issues	5	(5 Resolved)
Medium Risk Issues	3	(3 Resolved)
Low Risk Issues	6	(5 Resolved)
Informational Risk Issues	8	(8 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

All of the high and medium severity issues that were discovered in the first four rounds of audits were addressed. A significant amount of additional time should be put in to manually testing the logic and enhancing the current unit tests to verify that all general and edge cases have been covered.
2020-07-03 update: we have audited the code diff between Cryptographv3 and v4.1. Six informational findings along with multiple best practice suggestions have been added to this report.
2020-07-06 update: all issues have been resolved.

ID	Description	Severity	Status
QSP-1	Incorrect Linked List logic	︽ High	Resolved
QSP-2	Unprotected `initCry()` function	︽ High	Resolved
QSP-3	Unprotected `reInitAuction()` function	︽ High	Resolved
QSP-4	Bypass Senate Vote	︽ High	Resolved
QSP-5	Manually Change `pendingWithdrawals`	︽ High	Resolved
QSP-6	Circumventing fees	︿ Medium	Resolved
QSP-7	Gas Usage / `for` Loop Concerns	︿ Medium	Resolved
QSP-8	Incorrect Cryptograph lookup	︿ Medium	Resolved
QSP-9	Incorrect event emitted in `SenateLogicV1`	﹀ Low	Resolved
QSP-10	Write to Arbitrary Storage Location	﹀ Low	Resolved
QSP-11	Storage corruption due to proxy calls	﹀ Low	Resolved
QSP-12	Use time units instead of integers	﹀ Low	Resolved
QSP-13	Integer Overflow / Underflow	﹀ Low	Acknowledged
QSP-14	Unlocked Pragma	﹀ Low	Resolved
QSP-15	`centimani()` not implemented	○ Informational	Resolved
QSP-16	Unimplemented White Paper logic	○ Informational	Resolved
QSP-17	Other Best Practices	○ Informational	Resolved
QSP-18	Missing functionality in `getApproved()`	○ Informational	Fixed
QSP-19	Missing functionality in `transferFromInternal()`	○ Informational	Fixed
QSP-20	Missing functionality in `safeTransferFrom()`	○ Informational	Fixed
QSP-21	Incorrect while loop condition	○ Informational	Fixed
QSP-22	Outdated implementation of `isContract ()`	○ Informational	Fixed

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Maian
- Truffle
- Ganache
- Mythril
- Securify
- Slither

Steps taken to run the tools:

1. Installed Truffle: `npm install -g truffle`
2. Installed Ganache: `npm install -g ganache-cli`
3. Installed the Mythril tool from Pypi: `pip3 install mythril`
4. Ran the Mythril tool on each contract: `myth -x path/to/contract`
5. Ran the Securify tool: `java -Xmx6048m -jar securify-0.1.jar -fs contract.sol`
6. Cloned the MAIAN tool: `git clone --depth 1 https://github.com/MAIAN-tool/MAIAN.git maian`
7. Ran the MAIAN tool on each contract: `cd maian/tool/ && python3 maian.py -s path/to/contract contract.sol`
8. Installed the Slither tool: `pip install slither-analyzer`
9. Run Slither from the project directory `slither .`

Assessment

Findings

QSP-1 Incorrect Linked List logic

Severity: High Risk

Status: Resolved

File(s) affected: `MintingAuctionLogicV1.sol`

Description:
There seems to be multiple flaws with the Linked List logic found in `MintingAuctionLogicV1`.

1. L186-L192 and L228-L235 have duplicated logic with the one difference being the last line where the tail is linked to the new head. L234 should be using the `setAbove()`function instead of `setBelow()`. This results in an incorrectly linked list and faulty logic for maintaining eligible minters.
2. L208 The tail bid is being reset to itself, it should be set to `BidLink(bidLinks[tailBidder]).above()).bidder()`
3. As mentioned in the comment on L130, if the number of bids is already `maxSupply`, there should be a check to make sure that the new bid is greater than the tail bid. Otherwise, the tail bid is incorrectly removed on L197-208. This allows an exploit where a significantly low bid can replace the tail bid right before the auction ends.
4. If the `_newBidAmount` is the second highest bid, the `if` clause on L260 causes it to be incorrectly inserted as the highest bid.
5. When the new bid is the new tail bid, `tailBidder` is not being set in L260-274

Recommendation:

1. Carefully review the linked list logic again. This is an essential foundational piece of logic for the minting auction to work properly.
2. Think through edge cases and spend enough time on testing and adding appropriate unit tests. The bugs mentioned above should have been caught by unit tests if unit tests were properly added.
3. Consider using other implementations of linked lists that are already used of have been audited. Eg. Modular's linked list
4. L249-257 can be significantly simplified. Pseudocode:

```
while (
  BidLink(currentLink).below() != address(0) &&
  BidLink(BidLink(currentLink.below()).bidAmount >= _newBidAmmount)
) {
  currentLink = BidLink(currentLink).below()
}
```

QSP-2 Unprotected `initCry()` function

Severity: High Risk

Status: Resolved

File(s) affected: `TheCryptographLogicV1.sol`

Description:
Any user can re-initialized an official cryptograph as long as `hasCurrentOwnerMarked == false`. In `TheCryptographLogicV1 initCry()`, L45, 46 are calling a user supplied contract `_myAuction` that can return any arbitrary value. This allows the auction to be changed even when it is ongoing and may lead to compromised Cryptographs.

Exploit Scenario: User uses a fake auction contract deployed at the address passed in as `_myAuction` to bypass the `require()`statements that check if the caller is the factory and that the auction hasn't started.

Recommendation:

1. L45, 46 should be replaced with `myAuction` instead of the user supplied parameter `_myAuction`.
2. Think about edge cases and conduct more rigorous testing

QSP-3 Unprotected `reInitAuction()` function

Severity: High Risk

Status: Resolved

File(s) affected: `CryptographFactoryLogicV1.sol`

Description: Anyone can call the `CryptographFactoryLogicV1 reInitAuction()` function as long as the auction isn't locked. The attacker can change multiple parameters of the auction including, the starting price, fee percentage and fee receiver addresses, start and end times, and whether to lock the auction.
This can lead to the attacker stealing fees and potentially compromising the Cryptograph through an unfair auction.

Recommendation: Limit the address that can call this function to the `officialPublisher`.

QSP-4 Bypass Senate Vote

Severity: High Risk

Status: Resolved

File(s) affected: `SenateLogicV1.sol`

Description: An integer overflow can be exploited to bypass the `enactionTime` requirement, allowing the `lawmaker` to bypass senate approval.

Exploit Scenario:

1. Lawmaker sets an artificially high `_duration`
2. `require` condition on L152 passes
3. L161 `enactionTime = now + _duration;` overflow, setting the `enactionTime` to a number smaller than `now`
4. Lawmaker can immediately call `EnactLaw()`
5. L189, 190 `enactable()` passes because `now > enactionTime` and `noCount == yesCount == 0`
6. Lawmaker can forcibly change any logic contract
7. This is especially problematic for `AuctionHouseLogicV1`where the contract can be upgraded to allow the lawmaker to withdraw all the ether

Recommendation: Use SafeMath

QSP-5 Manually Change `pendingWithdrawals`

Severity: High Risk

Status: Resolved

File(s) affected: `AuctionHouseLogicV1.sol`

Description: A user can exploit the unprotected `initCry()` function mentioned above to set a fake auction contract and exploit integer underflows to manipulate the `pendingWithdrawals` balance. This compromises the integrity of the AuctionHouse contract, resulting in the user having an unlimited balance to bid on Cryptographs and potentially steal ether that he does not own.

Exploit Scenario:

1. User exploits the unprotected `initCry()` function mentioned in the previous vulnerability to set a fake auction contract.
2. User calls `bid()` in `AuctionHouseLogicV1`
3. L87 `SingleAuctionLogicV1 _auc` is set to be the fake contract
4. User exploits the same vulnerability described in detail in the `Circumventing fees` section to return different values when `_auc.currentBids(msg.sender)` is called (L101, 103).
5. L106 underflows resulting in an artificially high `pendingWithdrawals` balance
6. User can siphon off ether from the `AuctionHouse` contract to the fake auction contract where a back door can be implemented to withdraw the stolen ether

Recommendation:

1. Address recommendations in the section `Unprotected initCry() function`
2. Address recommendations in the section `Circumventing fees`
3. Use SafeMath

QSP-6 Circumventing fees

Severity: Medium Risk

Status: Resolved

File(s) affected: `CryptographFactoryLogicV1`

Description:
Users may bypass the require-logic in `instanceCryptograph()` and `instanceCryptographGGBMA()` that checks correct bounds on the "cut" variables (specifically on L244, L247-250, L336, L339-343). `CryptographInitiator(_cryInitiator).perpetualAltruismCut()` is called multiple times, and because `_cryInitiator` is a user supplied parameter, the contract may return different values each time it is called. This allows the user to bypass the requirements that the PA's cut is at least `25000` and that all the cuts add up to `100000`.

Exploit Scenario:
One example of exploiting this is using a fake initiator contract that returns different values based on `gasleft`.

```
pragma solidity 0.5.1;

contract CryptographInitiator{
    uint public perpetualAltruismCut = 25000;
}

contract FakeCryptographInitiator{
    function perpetualAltruismCut() public view returns(uint256) {
        uint256 remaining = gasleft();

        // gasleft threshold derived from setting the gas limit as 3000000 on Remix using the Javascript VM
        if (remaining <= 2927147) {
            return 1;
        }
        return remaining;
    }
}

contract LogicContract {
    uint256 public perpetualAltruismCut;
    function setPerpetualAltruismCut(address _address) public {

        require(CryptographInitiator(_address).perpetualAltruismCut() >= 25000);
        perpetualAltruismCut = CryptographInitiator(_address).perpetualAltruismCut();
    }
}
```

It is apparent that the `perpetualAltruismCut() >= 25000` requirement is circumvented and the user is able to set a significantly lower fee.

Recommendation:

1. Only call `CryptographInitiator(_cryInitiator).perpetualAltruismCut()` once and save the return value in a temporary variable. Use that variable in the subsequent logic.
2. The same issue applies to all the getter functions in `CryptographInitiator` and any contract which is user supplied.

QSP-7 Gas Usage / `for` Loop Concerns

Severity: Medium Risk

Status: Resolved

File(s) affected: `SingleAuctionLogicV1.sol`

Description: Gas usage is a main concern for smart contract developers and users, since high gas costs may prevent users from wanting to use the smart contract. Even worse, some gas usage issues may prevent the contract from providing services entirely. For example, if a `for` loop requires too much gas to exit, then it may prevent the contract from functioning correctly entirely. It is best to break such loops into individual functions as possible.

Exploit Scenario: The `win()` function may be susceptible to exceeding gas limits due to the loop on L368 if an auction is very popular. As a crude estimate, there appear to be three storage writes in each iteration of the loop, totaling at minimum 15000 gas. If the block gas limit remains at 10 million, then an auction could be locked by approximately 667 bids or less.
An attacker can purposely submit multiple bids and execute a Denial-of-service attack where the Cryptograph cannot be claimed.

Recommendation:

1. Break up the `win()` function into multiple functions where bids can be cancelled in chunks.
2. Conduct further stress testing and decide whether to place a maximum limit on the number of bids
3. The same issue applies to all functions that rely on potentially large while or for loops

QSP-8 Incorrect Cryptograph lookup

Severity: Medium Risk

Status: Resolved

File(s) affected: `CryptographIndexLogicV1.sol`

Description: In `CryptographIndexLogicV1.sol`, `getCryptograph() L218` looks up the `cryptographs` array when it should be using `communityCryptographs`. This will result in treating the `edition/GGBMA` cryptographs as `unique` and result in unintended consequences.

Recommendation:

1. Replace L218 `cryptographs[_cryptographIssue]` with `communityCryptographs[_cryptographIssue]`. Furthermore this array lookup should be saved in a temporary variable and reused to prevent similar issues.
2. Conduct thorough testing and add appropriate unit tests

QSP-9 Incorrect event emitted in `SenateLogicV1`

Severity: Low Risk

Status: Resolved

File(s) affected: `SenateLogicV1.sol`

Description: The function `quanranteNeufTrois()` only emits the `AddedLogicCodeToVC` event even when a candidate address is removed. This may lead to unintended consequences if off-chain services rely on these events.

Recommendation: Emit the `RemovedLogicCodeToVC` event instead of `AddedLogicCodeToVC` when an address is removed.

QSP-10 Write to Arbitrary Storage Location

Severity: Low Risk

Status: Resolved

File(s) affected: `EditionIndexerLogicV1.sol`

Related Issue(s): SWC-124

Description:
[Update] CryptoGraph is aware of the risks associated with the ability to write to arbitrary storage locations. This vulnerability is mitigated by the function invocation being limited to the `CryptographIndex` *contract through the use of the* `restrictedToIndex` *modifier.*

A smart contract's data (e.g., storing the owner of the contract) is persistently stored at some storage location (i.e., a key or address) on the EVM level. The contract is responsible for ensuring that only authorized user or contract accounts may write to sensitive storage locations. If an attacker is able to write to arbitrary storage locations of a contract, the authorization checks may easily be circumvented. This can allow an attacker to corrupt the storage; for instance, by overwriting a field that stores the address of the contract owner.

`insertACryptographAt()` in `EditionIndexerLogicV1` allows writes to arbitrary storage locations. While this risk is mitigated by restricting who can call this function, it can still lead to unintended overriding of storage.

Exploit Scenario:
As an example, if `insertACryptographAt()` is called with the following parameter:

- `_cryptograph = 0x0000000000000000000000000000000000000000`
- `_index = 114245411204874937970903528273105092893277201882823832116766311725579567940179`

`index` will be overwritten by address(0x0), hence freezing all access to any functions with the `restrictedToIndex()` modifier.

Recommendation: Avoid using a function that enables arbitrary access to a large array like in `insertACryptographAt()`. Instead, use a mapping to implement the same functionality. Alternatively, `insertACryptographAt()` can also limit the maximum index to be less than `editionSize`.

QSP-11 Storage corruption due to proxy calls

Severity: Low Risk

Status: Resolved

File(s) affected: `TheCryptographV1.sol`

Description: `TheCryptographStorageInternalV1` is missing the storage variable `string public history;` that is included in `TheCryptographStoragePublicV1`. This may cause unintended storage corruption when making proxy calls.

Recommendation: Make sure the storage variable ordering perfectly match up for all proxy and logic contracts.

QSP-12 Use time units instead of integers

Severity: Low Risk

Status: Resolved

Description:
[Update] CryptoGraph has opted not to use time units but to break up timestamps into numbers that are easier to comprehend.
eg. `1209600` => `60*60*24*14`

Throughout the code, integers are used to express time intervals and cutoff dates. Solidity has time units that can be used to express time and make the code more readable.

As an example, L136 in `TheCryptographLogicV1` has the following line:

```
require(now >= lastOwnerInteraction + 31622400, "Two years have not yet elapsed since last owner interaction");
```

There is a code/comment mismatch here where the code `31622400` specifies 366 days while the comment says `Two years`. Such mistakes can easily be prevented if `31622400` is replaced with `732 days` or `731 days`, (depending on how leap years are counted).

Recommendation: Replace all places in the code where integers are used to express time with Solidity's time units. If the same number is used multiple times in a contract, the value should also be declared and used as a constant.

QSP-13 Integer Overflow / Underflow

Severity: Low Risk

Status: Acknowledged

Description:
[Update] The CryptoGraph team is aware that adding SafeMath leads to defensive programming but has opted not to add it due to gas concerns and the lack of flexibility in future updates. "Safemath is better than no Safemath. Defensive programming have less bugs and glitch than "risky" programming. However, the auctions smart contracts (where actual math are happening) are already big, and our updatable proxy solution already make each interaction with a smart contract quite expensive. Very little defensive programming is used in our smart contracts in general, as it would simply prevent them from being deployed. Hence defensive programming is only used where users could actually cheat the system without it. Splitting the smart contracts in several parts in order to deploy them would also set in stone the ABI for the "split" features, and we never know if we want to add more parameters to functions/change features."

Integer overflow/underflow occur when an integer hits its bit-size limit. Every integer has a set range; when that range is passed, the value loops back around. A clock is a good analogy: at 11:59, the minute hand goes to 0, not 60, because 59 is the largest possible minute. Integer overflow and underflow may cause many unexpected kinds of behavior and was the core reason for the `batchOverflow` attack. Here's an example with `uint8` variables, meaning unsigned integers with a range of `0..255`.

```
function under_over_flow() public {
    uint8 num_players = 0;
    num_players = num_players - 1;  // 0 - 1 now equals 255!
    if (num_players == 255) {
        emit LogUnderflow();        // underflow occurred
    }
    uint8 jackpot = 255;
    jackpot = jackpot + 1;          // 255 + 1 now equals 0!
    if (jackpot == 0) {
        emit LogOverflow();         // overflow occurred
    }
}
```

Recommendation: There is already a high severity vulnerability that exploits an integer overflow. All math operations throughout all the contracts should also use SafeMath to prevent unintended consequences.

QSP-14 Unlocked Pragma

Severity: Low Risk

Status: Resolved

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.4.*`. 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." 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.

Recommendation: Lock the solidity version on `pragma solidity 0.5.1;`

QSP-15 `centimani()` not implemented

Severity: Informational

Status: Resolved

Description:
[Update] CryptoGraph team's response: "While earlier version of the code had those features, our current decision is to implement it later after engaging with the community to see how they would like those long term features materialized -along with other changes to the senate-, as opposed to fully designed and imposed by us."

The `centimani()`function is mentioned in the Whitepaper but not implemented in the code.

Recommendation: Implement the feature, update the White Paper, or update the documentation to include a note on this feature.

QSP-16 Unimplemented White Paper logic

Severity: Informational

Status: Resolved

File(s) affected: `SingleAuctionLogicV1.sol, CryptographFactoryLogicV1.sol`

Description:
[Update] Same as the issue above, this will be implemented in the future when necessary.

The white paper states the following, "If ever Perpetual Altruism were to disappear, each Cryptograph owner will become responsible for their Cryptograph and will receive Perpetual Altruism's share of the proceeds to maintain the Cryptograph ecosystem and continue carrying out its purpose." This would technically require PA to upload a new version of the smart contracts, since otherwise L433-438 of `distributeStakeholdersPayouts()` will pay PA at least 25% of the revenue.

Recommendation: Publish a version of the contract that doesn't require a minimum `perpetualAltruismCut` or outline plans to do so in the future in the README.

QSP-17 Other Best Practices

Severity: Informational

Status: Resolved

Description:

- Explicitly use `uint256` instead of `uint`
- Fix multiple spelling errors that appear in variable names, README, and code comments. Some examples include `UserCanceldBid, herit, aligment, versionning, abandonned, iniator, cryptgoraph, perpetial, Unitiliazed, _newBidAmmount, communiy, trigerring, trigered, Reseting, adress, quanrante`
- Use `constants` for values that do not change. Eg. `SingleAuctionLogicV1 L60 bid_Decimals`
- `CryptographIndexV1.sol` Consider using enums instead of integers for `cryptographType`
- `condition == false` should be replaced with a simple negation `!condition`. This happens in multiple places throughout the code base. Eg. `SenateLogicV1 L48 require(democracy == false)`
- Add a return value to `CryptographFactoryLogicV1 mintGGBMA()`
- In `CryptographFactoryLogicV1` `msg.sender == officialPublisher` is used multiple times. If used multiple times in the same function, save the value to a temporary variable and reuse it in the subsequent logic
- `SenateProxiedV1 L14` is meaningless because it is later overwritten when `SenateLogicV1 init()` is called
- `CryptographFactoryLogicV1 L6` has an unnecessary `"/"` at the end of the import. This should be removed. Also verify that the code compiles and tests can be executed.
- `CryptographFactoryV1 L10` remove commented out code
- `SingleAuctionLogicV1 L91, 92` remove commented out code
- `TheCryptographLogicV1 L140` An event should be fired to indicate that `renatus()` has started
- Double check the comment on `SenateLogicV1.sol L36`. It does not match the code

QSP-18 Missing functionality in `getApproved()`

Severity: Informational

Status: Fixed

File(s) affected: `contracts/ERC2665LogicV1.sol`

Description: `getApproved()` never returns an address other than `address(0)`. It is not clear from the inline documentation if this is intended, or if there is missing functionality.

QSP-19 Missing functionality in `transferFromInternal()`

Severity: Informational

Status: Fixed

File(s) affected: `contracts/ERC2665LogicV1.sol`

Description: In `transferFromInternal()`, on `L439` we have the comment "//Check that there is no auction going on" that does not appear to have any associated code. It should be confirmed that there are no missing checks in this function.

QSP-20 Missing functionality in `safeTransferFrom()`

Severity: Informational

Status: Fixed

File(s) affected: `contracts/ERC2665LogicV1.sol`

Description: For function `safeTransferFrom()`, the check described on `L95` stating "Throws if `_to` is the zero address." is not enforced.

QSP-21 Incorrect while loop condition

Severity: Informational

Status: Fixed

File(s) affected: `contracts/EditionIndexerLogicV1.sol`

Description: The `while()` condition on `L75` in `contracts/EditionIndexerLogicV1.sol` should use "<=" instead of "<"

QSP-22 Outdated implementation of `isContract ()`

Severity: Informational

Status: Fixed

File(s) affected: `contracts/ERC2665LogicV1.sol`

Description: The function `isContract` will return `false` when a contract invokes this function in its constructor (see https://consensys.github.io/smart-contract-best-practices/recommendations/#avoid-using-extcodesize-to-check-for-externally-owned-accounts for detailed explanation).

Recommendation: It is recommended to use the slightly more robust `isContract()` implementation provided by OpenZeppelin in its `Address.sol`: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/b991fca3419696342604a86626c5dd56d635e0aa/contracts/utils/Address.sol#L26.

Automated Analyses

Maian

Maian did not report any vulnerabilities.

Mythril

Mythril did not report any vulnerabilities.

Securify

Securify did not report any vulnerabilities.

Slither

- Slither reported several potential reentrancy issues where state variables and events were emitted after external calls, however since these calls were to trusted Cryptograph contracts, we labelled these as false positives.
- The function CryptographFactoryLogicV1.mintGGBMA() does not have an explicit return-statement that returns a bool.

Adherence to Specification

The contracts adhere to the specifications in the White Paper and README.

Code Documentation

The code is well documented with a high level README and specific comments at the contract level.

Adherence to Best Practices

The contract does not use any templated libraries or contracts like OpenZeppelin's. Best practices such as using SafeMath should be adopted in order to prevent simple but large impact issues. The contract also relies on an upgradable design that separates logic and state. While upgradability can be beneficial, the use of proxy contracts and delegate calls adds another layer of risk that needs to be carefully examined.

2020-07-03 update: We have audited the code diff between Cryptographv3 (d61ecfb) and v4.1 (05650e3):

It is recommended to use `SafeMath` library for all arithmetic operations, to prevent unexpected results. E.g., `L412` and `L413` in `contracts\ERC2665LogicV1.sol`.

`L6` in `contracts\CryptographKYCLogicV1.sol` the comment is incorrect.

A typo "adress" on `L114` in `contracts\SingleAuctionBidLogicV1.sol`.

On `L439` of `contracts\ERC2665LogicV1.sol` the code described in the comment is not implemented.

For `ERC2665LogicV1.sol` in both `safeTransferFrom()` functions, the constant value of `bytes4(keccak256("onERC2665Received(address,address,uint256,bytes)"))` could be stored in a variable to avoid gas costs of the `keccak256` computation on each safe transfer. Similarly, `keccak256(bytes("ETH"))` on `L242` could be pre-computed.

For `ERC2665LogicV1.sol` in `addressToString()`, it is not clear why on `L476`: `bytes memory str = new bytes(51);` has length 51. A length of 42 seems sufficient.

In `AuctionHouseLogicV1.sol`, the `init()` function should ensure that the address arguments are non-zero.

`L508` in `contracts/CryptographFactoryLogicV1.sol` “communityc ryptographs” should be “community cryptographs”

`L36` in `contracts/ERC2665LogicV1.sol` there is the typo `hapenning` -> `happening`

In `contracts/ERC2665LogicV1.sol` In the function `approve` it seems that the transfer fee is automatically being paid if the extra ETH is added into the payment by accident. This should be documented.

`L321` in `contracts/CryptographFactoryLogicV1.sol` there doesn’t exist documentation of why this function would be disabled when it is in production. It is recommended for this to be added into the specs

The overall structure of the project is hard to read, it may be a good idea to separate the proxy, interface, and implementation files

Test Results

Test Suite Results

All tests provided have been ran and have successfully passed.

```
Contract: Initial Auction Test
   ✓ Grabbing the deployed ecosystem (4542ms)
   ✓ Can put and withdraw funds from the auction house (306ms)
   ✓ Bidding: Placing a bid in an initial auction and winning it (2496ms)
   ✓ Bidding: Can not win before the end of the initial auction (2327ms)
   ✓ Bidding: Can not place an initial bid of 0 (1600ms)
   ✓ Bidding: Getting outbid in initial auction cancel your previous bid (2471ms)
   ✓ Bidding: Can partially fund a new bid using a previous one (2103ms)
   ✓ Bidding: Can not place an out-offer too low (2070ms)
   ✓ Bidding: Can not cancel your bid in initial auction (1870ms)
   ✓ Bidding: Incentive value testing (3692ms)
   ✓ Bidding: Single bid proceeds to seller (2129ms)
   ✓ Bidding: Multiple bid proceeds to seller (2964ms)

Contract: ERC2665 Test
   ✓ Grabbing the deployed ecosystem (4097ms)
   ✓ ERC2665: ERC-165 Signatures (149ms)
   ✓ ERC2665: ERC-721 Symbol, Name (71ms)
   ✓ ERC2665: Can transfer a Cryptograph by paying a transfer fee (2325ms)
   ✓ ERC2665: Cannot transfer without paying a transfer fee (2192ms)
   ✓ ERC2665: Transfer fee calculations and fetching (4014ms)
   ✓ ERC2665: Approve can be used to set the transfer fee to 0 (2632ms)
   ✓ ERC2665: An approved recipient can Transfer (2542ms)
   ✓ ERC2665: A non approved can't Transfer (2533ms)
   ✓ ERC2665: An Operator can approve/transfer (2933ms)
   ✓ ERC2665: A non operator can't approve/transfer (2425ms)
   ✓ ERC2665: Multiple operators per owner (3733ms)
   ✓ ERC2665: Approve and Transfer reset Sale price (3599ms)
   ✓ ERC2665: Cannot transfer during secondary auction (3331ms)
   ✓ ERC2665: Transfer do not impact existing offers (3614ms)
   ✓ ERC2665: Can prepay next transfer fee using approve (2324ms)
   ✓ ERC2665: Can prepay next transfer fee using transfer (2334ms)
   ✓ ERC2665: Enumerate all cryptographs minted so far (1690ms)
   ✓ ERC2665: Enumerate all Cryptographs owned by a user (518ms)
   ✓ ERC2665: Initial auction emit proper transfer event (1976ms)
   ✓ ERC2665: Secondary auction emit proper transfer event (2706ms)
   ✓ ERC2665: xTransferFrom emit proper Transfer event (2311ms)
   ✓ ERC2665: Approve emit Approval event (2387ms)
   ✓ ERC2665: setApprovalForAll emit ApprovalForAll event (76ms)

Contract: Secondary Market Offers Test
   ✓ Grabbing the deployed ecosystem (4567ms)
   ✓ Bidding: Placing an offer in the secondary market (2577ms)
   ✓ Bidding: Can not place a secondary offer of 0 (2140ms)
   ✓ Bidding: First offer can be cancelled for free (3104ms)
   ✓ Bidding: Subsequent offers must meet minimum bid (3998ms)
   ✓ Bidding: Getting out-offered do not replace your bid (2814ms)
   ✓ Bidding: Getting out-offered gives you returns (3592ms)
   ✓ Bidding: Cancelling the highest bid do not net full returns (4730ms)
   ✓ Bidding: Getting out-offered allow to cancel offers for free (5113ms)
   ✓ Bidding: Cancelling the highest bid recalculate the next highest bid (6404ms)
   ✓ Bidding: Outoffering yourself replace your previous bid (5114ms)
   ✓ Bidding: Outoffering yourself and cancelling calculate new highest bidder (3983ms)

Contract: VC and Senate Governance Test
   ✓ Grabbing the deployed ecosystem (8384ms)
   ✓ VC: Checking VC contract logic matching (1136ms)
   ✓ VC: Perpetual Altruism can change features of deployed contracts (1760ms)
   ✓ VC: Only Perpetual Altruism can change features of deployed contracts (751ms)
   ✓ VC: Only Valid addresses can be applied to the VC (284ms)
   ✓ Senate: Perpetual Altruism can 49.3 when democracy is not enabled (340ms)
   ✓ Senate: Perpetual Altruism can enact a return to democracy (450ms)
   ✓ Senate: Perpetual Altruism can't enact a 49.3 if democracy is enabled (580ms)
   ✓ Senate: Token holders can vote to allow a law being passed (2880ms)
   ✓ Senate: Token holders can vote to prevent a law being passed (4709ms)

Contract: Renatus Test
   ✓ Grabbing the deployed ecosystem (5366ms)
   ✓ Renatus: Can Call Renatus on a Cryptograph auction that had no bidders (4104ms)
   ✓ Renatus: Can not call Renatus too early (3027ms)
   ✓ Renatus: Renatus preserve bidders (4407ms)
   ✓ Renatus: Renatus bids behave like initial auction (5467ms)
   ✓ Renatus: ERC2665 operators can reset renatus timer (3267ms)

Contract: Ecosystem Setup Test
   ✓ Control Test
   ✓ Grabbing the deployed ecosystem (5262ms)
   ✓ Setup: Ecosystem init twice protection (795ms)

Contract: Minting Test
   ✓ Grabbing the deployed ecosystem (5692ms)
   ✓ Minting: First Single Cryptograph (1084ms)
   ✓ Minting: Second single Cryptograph (1150ms)
   ✓ Minting: Single Cryptograph event emissions (1595ms)
   ✓ Minting: Can modify a Cryptograph before auction start (2557ms)
   ✓ Minting: Third parties can not modify a cryptograph (1113ms)


73 passing (4m)
```

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

```
4b40a76a56d691b7e157d88e95f7049fc4aed17483796f2f821ad0e550f862cd ./contracts/AuctionHouseEmptyV1.sol
77ebb171be34695e13e2ec6f482c0a61b7b4a1c315de90baff9991bd7b05356c ./contracts/AuctionHouseLogicV1.sol
28b3c1dae529463ff4bb68ade0c4161a6cafa685ebdb1fcf8edcac5fe3cb8a6b ./contracts/AuctionHouseProxiedV1.sol
13713186cb6b73b4c672f2bfd1963cb9deb041272c1268bce80897e42a2275cf ./contracts/AuctionHouseV1.sol
46e291fb3f776e35450061f5c4a83cba5a059097d9e6ba3903ac6cc1a2b83264 ./contracts/BidLink.sol
afad1debe9cdd52ac6696a9dcf764e1f619c08bce6cc9771712900fcb0509831 ./contracts/BidLinkSimple.sol
9e4fb553a65d0097d38f8b29301ec1b2ce9ec6e6ab5a3da27816bb8512d60aa1 ./contracts/CryptographFactoryLogicV1.sol
571cdf9b9a5bbb171b0d32e854705dd6422b55af9a45a1c7fae1fdc345208d41 ./contracts/CryptographFactoryProxiedV1.sol
41e5ee1f0a00735200b63d2f362e4e93c25cce7aa5d9b0f500db27b8a8afa44b ./contracts/CryptographFactoryV1.sol
cd65322b470d4c132a68f700f77f6cdb16c4fa348eb337b72776c2d0e96b514a ./contracts/CryptographIndexLogicV1.sol
1a086030093d5fe5693bd0433460ba781a8ad1419fc0519151b2d08aebac7452 ./contracts/CryptographIndexProxiedV1.sol
ecee93a327ee56f7d6ff689ba9ab7ad3cd64ea1685be78ed875a154edc46a431 ./contracts/CryptographIndexV1.sol
a34a824b0ceded31f0b2f1786850887d6ac915c29aa099ff49b51fde23e63764 ./contracts/CryptographInitiator.sol
eb5babb44aa95d841bab6eb91101dbc7a99587d9045fc8f7ab3eaaae2e5cabf8 ./contracts/CryptographKYCLogicV1.sol
64183d367057a4a55bab7dc5b0c0b9697f2be8d28568a773ca5f2f832aed6d66 ./contracts/CryptographKYCProxiedV1.sol
8f68587036916a0730c2a312974fdc12eb29a00466c2e119fbb50bd1eefefa72 ./contracts/CryptographKYCV1.sol
c19b0f587abb400a3ae7ad9f9dbf785bd752c93d0a17e3fdef2d3efd2488bca0 ./contracts/EditionIndexerLogicV1.sol
65f0e2b6d6545f8736fc3064971c5ebddac4d364d95c7a74a29330e4fa5292b2 ./contracts/EditionIndexerProxiedV1.sol
83aea10db7dcf73a98c23edad0a55ed7d4b0a5db8ed04a39e434f30fcced5672 ./contracts/EditionIndexerV1.sol
2bb968f650601a59bbf0be89ccc87de0b8a21a3eedccb56b71ea1f134adf26d2 ./contracts/ERC2665LogicV1.sol
95e75b490fa2bc5165613f199806f25a2e114a51bd0ee953c3c4f0b26ac30865 ./contracts/ERC2665ProxiedV1.sol
df94c8a436f7d71a7c483ccc258789c5ffebd29a869484883fba6ea028459f4a ./contracts/ERC2665V1.sol
58d4754dfbd4703d49d878b226b28b6aa636899b5626eeeb122332e7d79e78d3 ./contracts/Migrations.sol
26ee5b05f672f21236d9d96c824d3d02e9b21c9ab40dffffbd73f8ddc66e4a1e ./contracts/MintingAuctionLogicV1.sol
65830151f2bf509d302798084a574129436b5f48d9fc292c16968f4b32706202 ./contracts/MintingAuctionProxiedV1.sol
7a32af379f9cec02197e1764becd076e658b24d9ef48d88199fc6718a96505da ./contracts/MintingAuctionV1.sol
b9f77278a85bf83f25987f3966cb1a5ccab707b32be2bdd99f5ebd5b6a6109ee ./contracts/SenateLogicV1.sol
78d0d1d3f16529e2f7e6c0af48b8f30b157b13ff52b1905dd2c2be180f40d0ae ./contracts/SenateProxiedV1.sol
f2d8fdd3ffc9e1d21c32d99c7af032a2583ae4c6c562e8198cd57a74cdf3ce96 ./contracts/SenateV1.sol
0115a4bf9fe8c6f4c8ef1ee2d1ed04abfcf716fc90d26901f4cea4ec957c9ab0 ./contracts/SingleAuctionBidLogicV1.sol
16213d17116b2968e7e4f698a308380eb90fd204f7fe6ed93ee8b27e218fcdd7 ./contracts/SingleAuctionLogicV1.sol
4be18216518b1025370911433a214f4826c72f78759b20934475ebc87710dfa6 ./contracts/SingleAuctionProxiedV1.sol
13577656a31e4a72c5f781b619c4be9d7f2f23957fd00b2032f37971e300aa8d ./contracts/SingleAuctionV1.sol
7c868e6db17413f44ce22cb9d36ccb611cf1f5601ebc03e1e5edbc91dd5969a3 ./contracts/TheCryptographLogicV1.sol
105a34cff775d07ba415b01470c84ec1cc05d8134049a6044841b37286999756 ./contracts/TheCryptographProxiedV1.sol
5bd2c7a53e22f4590e8671fdbaf41af8194488d75697b251c84f374a38b2be06 ./contracts/TheCryptographV1.sol
37f813e5d6039ec06fd493b17ba3f3e122cafb17e7f07990f005a4ef7636ecac ./contracts/VCProxy.sol
b291e688079a5019582e4f00b634fdcdc4210760ef35f7b1f2537eed267729c2 ./contracts/VersionControlLogicV1.sol
6294b591aa2eb698c44e6d0c85a656039b4a56fed235975a5397bff3a34b5aa4 ./contracts/VersionControlProxiedV1.sol
894ad4ad044f579d03227eb243e64cb0500016c08a897e612c571f017c071952 ./contracts/VersionControlV1.sol
```

Tests

```
8d3796735d408242ddef70287e2585444a715090054ea142312fd00fddc77c4e ./test/0003CryptographAuction.js
64b8fc821553dd10210b6c8f071fc67b08420c6ef4d311368f4bb0fc58476cc0 ./test/0004ERC2665.js
ba7a76db608d97af9f47330351fe7f806b19ac5b45d9bf8a4293212c4b162081 ./test/0005CryptographSecondaryOffers.js
0359d8b2857585b5f6330c6099ac3785c6671858f7cb3205c1c100cc4936d22d ./test/0006CryptographSenate.js
ec9e981e92c5b5fc87d40540a1a24d2bfe9d6680994c838f1db22920e65a4b8c ./test/0007CryptographRenatus.js
c9670c2363e5506f37552316dd17a7a6b4deab67e0b59e6fd4fd47b79de3be75 ./test/001EcosystemSetupTest.js
175f2a8b6bb839cb0bf01d1b4a0b7577138d61675ae5c5518363a646656da272 ./test/002CryptographMinting.js
```

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 $1B 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.