G₀ Group

Security Review of

Gnosis Conditional Tokens

November 22, 2019

Overview

G0 Group was engaged to perform a security review of Gnosis Conditional Tokens. G0 Group was contracted for a six person-week effort to that end. The primary subjects of this review were the smart contracts which implement Gnosis Conditional Tokens: tokens which represent positions in conditional markets and retain the fungibility of logically equivalent positions. This review was initially performed on https://github.com/gnosis/conditional-tokens-contracts/tree/a050b6c16aba8e3bfd6697e9a68bd23aeba307b4.

Files in Scope

```
contracts/
    ERC1155/
        ERC1155.sol
        ERC1155TokenReceiver.sol
        IERC1155.sol
        IERC1155TokenReceiver.sol
    CTHelpers.sol
    ConditionalTokens.sol
```

Result Summary

During the course of this review, 3 issues were discovered, reported, and addressed.

No further issues were discovered in https://github.com/gnosis/conditional-tokens-contracts/commit/4afa2fed1dfa62d8f413e126f238811f1d40bbfc

Issues

1. By splitting non-existent collections, it's possible to forge other collections and ultimately steal all collateral tokens from the contract

Type: *security* / ***Severity:*** *critical*

It's possible to split non-existent position tokens so that some of the resulting tokens will share the same `collectionId` as a different position, this is possible for three reasons:

- **A.** When splitting tokens, the split tokens are destroyed after the output tokens resulting from the split have been created
- **B.** When new tokens are created and transferred to the recipient, the `onERC1155Received` function on the recipient's address is called, enabling re-entracny of the `ConditionalTokens` contract
- **C.** Complex `collectionId`s are derived in a predictable way from the `collectionId`s of included positions. The `collectionId` of a complex position is a simple sum of all contained positions. This allows an attacker to craft a position that upon splitting will result in tokens that have a `collectionId` that collides with a different position.

For example, a collection with collectionId:
`bytes32(uint(keccak256(abi.encodePacked(conditionId, 0b01))) - uint(keccak256(abi.encodePacked(conditionId, 0b10))))`

when split will result in collections with ids:

`bytes32(uint(keccak256(abi.encodePacked(conditionId, 0b01)))`

(if `0b01` is the winning outcome, this position can be directly redeemed as collateral)

and

`bytes32(uint(keccak256(abi.encodePacked(conditionId, 0b01))) + uint(keccak256(abi.encodePacked(conditionId, 0b01))) - uint(keccak256(abi.encodePacked(conditionId, 0b10))))`

which if `0b01` is the winning outcome can be be redeemed back into

`bytes32(uint(keccak256(abi.encodePacked(conditionId, 0b01))) - uint(keccak256(abi.encodePacked(conditionId, 0b10))))`

in full, ensuring the original split terminates correctly.

Replication:

```
splitPosition(
    collateralToken,
    bytes32(uint(keccak256(abi.encodePacked(conditionId, 0b01))) -
uint(keccak256(abi.encodePacked(conditionId, 0b10)))),
    conditionId,
    [0b10, 0b01],
    amount
)

    ConditionalTokens._mint(..) -> msg.sender.onERC1155Received(..) ->

            redeemPositions(
                collateralToken,
                bytes32(uint(keccak256(abi.encodePacked(conditionId,
0b01))) - uint(keccak256(abi.encodePacked(conditionId, 0b10)))),
                conditionId,
                [0b01]
            ) //redeems `amount` of position of collection with id
`bytes32(uint(keccak256(abi.encodePacked(conditionId, 0b01))) -
uint(keccak256(abi.encodePacked(conditionId, 0b10))))` so that
splitPosition can burn it and successfully terminate

            redeemPositions(
                collateralToken,
                bytes32(0),
                conditionId,
                [0b01]
            ) //redeems collateral from the winning position that has been
forged
```

Fix Description:

The issue was addressed by burning the split tokens, before minting new ones and is no longer present in:
https://github.com/gnosis/conditional-tokens-contracts/tree/4afa2fed1dfa62d8f413e126f238811f1d40bbfc

2. The multihash algorithm employed is vulnerable to generalised birthday attack

Type: *security* / ***Severity:*** *critical*

IDs of complex collections are sums of hashes of data describing simple collections (an algorithm known as AdHash[1]). Unfortunately there are known practical techniques that allow finding sets of different hashes that sum to the same number, opening the indexing system to fatal collision attacks[2].

Fix Description:

The issue was addressed by replacing the AdHash algorithm with Elliptic Curve Multiset Hash[3], this seems to be a promising solution, though G0 Group has recommended further analysis by cryptography specialists. Gnosis contracted a 3rd party to that end, and their analysis can be found here.

3. Possible efficiency gains by adding a batch mint method to the ERC1155 contract

Type: *efficiency* / ***Severity:*** *minor*

Adding a batch mint in ERC1155 that invokes `_doSafeBatchTransferAcceptanceCheck` instead of `_doSafeTransferAcceptanceCheck` might be useful because `ConditionalTokens.sol:L126` can be executed many times and generate a lot of external calls through `_doSafeTransferAcceptanceCheck`.

Fix Description:

Batch mint is present in https://github.com/gnosis/conditional-tokens-contracts/tree/4afa2fed1dfa62d8f413e126f238811f1d40bbfc

References

[1] M. Bellare, D. Micciancio, (1996). A New Paradigm for Collision-free Hashing: Incrementality at Reduced Cost.

[2] D. Wagner, (2002). A Generalized Birthday Problem.

[3] J. Maitin-Shepard, M. Tibouchi, D. Aranha, (2016). Elliptic Curve Multiset Hash.