CONSENSYS Diligence

Gitcoin Token Distribution

Date	April 2021
Lead Auditor	Shayan Eskandari
Co-auditors	Daniel Luca

1 Executive Summary

This report presents the results of our engagement with **Gitcoin** to review **GTC Token Distribution and Governance**.

The review was conducted over two weeks, from **April 19, 2021** to **April 30, 2021** by **Shayan Eskandari** and **Daniel Luca**. A total of 20 person-days were spent.

2 Scope

Our review focused the smart contract files for governance on the commit hash `ee5e45a008d65021831de9f3e83053026f2a4dd2` and the Ethereum Signed Message Service (ESMS) repository on the commit hash `5eb22e882e28e6f3192b80f237f7a3bcd15b1ee9`. The list of Solidity files in scope can be found in the Appendix.

3 Security Specification

This section describes, **from a security perspective**, the expected behavior of the system under audit. It is not a substitute for documentation. The purpose of this section is to identify specific security properties that were validated by the audit team.

3.1 Actors

The relevant actors are listed below with their respective abilities:

- **Signer**: Given that not all Gitcoin users have an Ethereum address on their profile, Signer, in combination with `gitcoin.co`, is used to validate the token distribution and the merkle proofs
  - Signer has the ability to approve (Sign) or reject requests for token claims
- **GTC Minter**: Specified at the deployment
  - Can change the *Minter* address

  - Can change and set TokenDistribution address `GTCDist`
  - Minter has the ability to mint `mintCap = 2` percentage of the *totalSupply*, after the specified `mintingAllowedAfter = 365 days`

- **Users**: Users with a valid user-id on `gitcoin.co` who are included in the initial distribution
  - Can claim their tokens (Given that they have valid signature from *Signer*)
  - Can set delegator on their tokens for governance voting
  - Can invoke any ERC20 functionality on GTC token
  - Can participate in the governance of GTC token

- **Governance**
  - Fork of Uniswap governance
  - No on-chain system properties could be changed through the governance mechanism at the time of the audit.

- **TokenDistribution** `GTCDist` address:
  - Can delegate votes from `delegator` to `delegatee`

- **TimeLock Contract**:
  - Fork of uniswap TimeLock contract
  - All tokens that have not been claimed in 24 weeks (6 months) after launch will be transferred to the TimeLock contract.
  - This contract will be in control of the assigned `admin`.

- **TreasuryVester**:
  - Fork of Uniswap TreasuryVester to vest tokens

3.2 Security Concerns

The following is a non-exhaustive list of security properties and concerns:

- The entry point to the system is through Github login on Gitcoin.co website. This is outside the scope of this audit, however, it should be noted that if an attacker can gain access to any user’ s Github account, they can claim the user’ s tokens (given that the user has not claimed the tokens first). It is highly advised to perform a full security audit and penetration testing on `gitcoin.co`.

- The Ethereum Signed Message Service (ESMS), is a micro-service that will be used as the verification method for token claims. It should be noted that this service plays a **critical** role as it contains the *private key of the signer* and the *database of all the merkle proofs*. Anyone with access to all these data, just needs to brute force an integer (`user_amount`) for each user to be able to reconstruct all data needed to claim the user’ s tokens.

4 Findings

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

4.1 ESMS use of sanitized `user_amount` & `user_id` values Medium

Fixed

Resolution

Fixed in https://github.com/nopslip/gtc-request-signer/pull/4/ , by using the sanitized integer value in the code flow.

Description

In the Signer service, values are properly checked, however the checked values are not preserved and the user input is passed down in the function.

The values are sanitized here:

code/gtc-request-signer-main-5eb22e882e28e6f3192b80f237f7a3bcd15b1ee9/app.py:L98-L108

```
try:
    int(user_id)
except ValueError:
    gtc_sig_app.logger.error('Invalid user_id received!')
    return Response('{"message":"ESMS error"}', status=400, mimetype='application/json')
# make sure it's an int
try:
    int(user_amount)
except ValueError:
    gtc_sig_app.logger.error('Invalid user_amount received!')
    return Response('{"message":"ESMS error"}', status=400, mimetype='application/json')
```

But the original user inputs are being used here:

code/gtc-request-signer-main-5eb22e882e28e6f3192b80f237f7a3bcd15b1ee9/app.py:L110-L113

```
try:
    leaf = proofs[str(user_id)]['leaf']
    proof = proofs[str(user_id)]['proof']
    leaf_bytes = Web3.toBytes(hexstr=leaf)
```

code/gtc-request-signer-main-5eb22e882e28e6f3192b80f237f7a3bcd15b1ee9/app.py:L128-L131

```
# this is a bit of hack to avoid bug in old web3 on frontend
# this means that user_amount is not converted back to wei before tx is broadcast
user_amount_in_eth = Web3.fromWei(user_amount, 'ether')
```

Examples

if a float amount is passed for `user_amount`, all checks will pass, however the final amount will be slightly different that what it is intended:

```
>>> print(str(Web3.fromWei(123456789012345, 'ether')))
0.000123456789012345
>>> print(str(Web3.fromWei(123456789012345.123, 'ether')))
0.000123456789012345125
```

Recommendation

After the sanity check, use the sanitized value for the rest of the code flow.

4.2 Prefer using `abi.encode` in `TokenDistributor` Medium Fixed

Resolution

Fixed in gitcoinco/governance#7

Description

The method `_hashLeaf` is called when a user claims their airdrop.

code/governance-main-ee5e45a008d65021831de9f3e83053026f2a4dd2/contracts/TokenDistributor.sol:L128-L129

```
// can we repoduce leaf hash included in the claim?
require(_hashLeaf(user_id, user_amount, leaf), 'TokenDistributor: Leaf Hash Mismatch.');
```

This method receives the `user_id` and the `user_amount` as arguments.

code/governance-main-ee5e45a008d65021831de9f3e83053026f2a4dd2/contracts/TokenDistributor.sol:L253-L257

```
/**
* @notice hash user_id + claim amount together & compare results to leaf hash
* @return boolean true on match
*/
function _hashLeaf(uint32 user_id, uint256 user_amount, bytes32 leaf) private returns (bool) {
```

These arguments are abi encoded and hashed together to produce a unique hash.

code/governance-main-ee5e45a008d65021831de9f3e83053026f2a4dd2/contracts/TokenDistributor.sol:L258

```
bytes32 leaf_hash = keccak256(abi.encodePacked(keccak256(abi.encodePacked(user_id, user_amount))));
```

This hash is checked against the third argument for equality.

code/governance-main-ee5e45a008d65021831de9f3e83053026f2a4dd2/contracts/TokenDistributor.sol:L259

```
return leaf == leaf_hash;
```

If the hash matches the third argument, it returns true and considers the provided `user_id` and `user_amount` are correct.

However, packing differently sized arguments may produce collisions.

The Solidity documentation states that packing dynamic types will produce collisions, but this is also the case if packing `uint32` and `uint256`.

Examples

Below there’ s an example showing that packing `uint32` and `uint256` in both orders can produce collisions with carefully picked values.

```
library Encode {
    function encode32Plus256(uint32 _a, uint256 _b) public pure returns (bytes memory) {
        return abi.encodePacked(_a, _b);
    }

    function encode256Plus32(uint256 _a, uint32 _b) public pure returns (bytes memory) {
        return abi.encodePacked(_a, _b);
    }
}

contract Hash {
    function checkEqual() public pure returns (bytes32, bytes32) {
        // Pack 1
        uint32  a1 = 0x12345678;
        uint256 b1 = 0x99999999999999999999999999999999999999999999999999999999FFFFFFFF;

        // Pack 2
        uint256 a2 = 0x1234567899999999999999999999999999999999999999999999999999999999;
        uint32  b2 = 0xFFFFFFFF;

        // Encode these 2 different values
        bytes memory packed1 = Encode.encode32Plus256(a1, b1);
        bytes memory packed2 = Encode.encode256Plus32(a2, b2);

        // Check if the packed encodings match
        require(keccak256(packed1) == keccak256(packed2), "Hash of representation should match");

        // The hashes are the same
        // 0x9e46e582607c5c6e05587dacf66d311c4ced0819378a41d4b4c5adf99d72408e
        return (
            keccak256(packed1),
            keccak256(packed2)
        );
    }
}
```

Changing `abi.encodePacked` to `abi.encode` in the library will make the transaction fail with error message `Hash of representation should match`.

Recommendation

Unless there's a specific use case to use `abi.encodePacked`, you should always use `abi.encode`. You might need a few more bytes in the transaction data, but it prevents collisions. Similar fix can be achieved by using `unit256` for both values to be packed to prevent any possible collisions.

4.3 Simplify claim tokens for a gas discount and less code Minor

Fixed

Resolution

Fixed in gitcoinco/governance#4

Structure `Claim` can still be removed for further optimization.

Description

The method `claimTokens` in `TokenDistributor` needs to do a few checks before it can distribute the tokens.

A few of these checks can be simplified and optimized.

The method `hashMatch` can be removed because it’ s only used once and the contents can be moved directly into the parent method.

code/governance-main-ee5e45a008d65021831de9f3e83053026f2a4dd2/contracts/TokenDistributor.sol:L125-L126

```
// can we reproduce the same hash from the raw claim metadata?
require(hashMatch(user_id, user_address, user_amount, delegate_address, leaf, eth_signed_message_hash_
```

Because this method also uses a few other internal calls, they also need to be moved into the parent method.

code/governance-main-ee5e45a008d65021831de9f3e83053026f2a4dd2/contracts/TokenDistributor.sol:L211

```
return getDigest(claim) == eth_signed_message_hash_hex;
```

code/governance-main-ee5e45a008d65021831de9f3e83053026f2a4dd2/contracts/TokenDistributor.sol:L184

```
hashClaim(claim)
```

Moving the code directly in the parent method and removing them will improve gas costs for users.

The structure `Claim` can also be removed because it’ s not used anywhere else in the code.

Recommendation

Consider simplifying `claimTokens` and remove unused methods.

4.4 ESMS use of environment variable for chain info [Optimization] Minor Fixed

Resolution

Fixed in nopslip/gtc-request-signer#5 by moving the variables to the environment variable.

Description

Variables to create domain separator are hardcoded in the code, and it requires the modify code on different deployments (e.g. testnet, mainnet, etc).

Examples

code/gtc-request-signer-main-5eb22e882e28e6f3192b80f237f7a3bcd15b1ee9/app.py:L203-L208

```
domain = make_domain(
    name='GTA',
    version='1.0.0',
    chainId=4,
    verifyingContract='0xBD2525B5F0B2a663439a78A99A06605549D25cE5')
```

Recommendation

Use environment variable for these values. This way there is no need to change the source code on different deployments and it can be scripted to prevent any possible errors on the code base.

4.5 Rename method `_hashLeaf` to something that represents the validity of the leaf Minor Fixed

Resolution

Closed because the method was removed in gitcoinco/governance#4

Description

The method `_hashLeaf` accepts 3 arguments.

code/governance-main-ee5e45a008d65021831de9f3e83053026f2a4dd2/contracts/TokenDistributor.sol:L257

```
function _hashLeaf(uint32 user_id, uint256 user_amount, bytes32 leaf) private returns (bool) {
```

The arguments `user_id` and `user_amount` are used to create a keccak256 hash.

code/governance-main-ee5e45a008d65021831de9f3e83053026f2a4dd2/contracts/TokenDistributor.sol:L258

```
bytes32 leaf_hash = keccak256(abi.encodePacked(keccak256(abi.encodePacked(user_id, user_amount))));
```

This hash is then checked if it matches the third argument.

code/governance-main-

ee5e45a008d65021831de9f3e83053026f2a4dd2/contracts/TokenDistributor.sol:L259

```
return leaf == leaf_hash;
```

The result of the equality is returned by the method.

The name of the method is confusing because it should say that it returns true if the leaf is considered valid.

Recommendation

Consider renaming the method to something like `isValidLeafHash`.

4.6 Method returns bool but result is never used in `TokenDistributor.claimTokens` Minor Fixed

Resolution

Removed in gitcoinco/governance#4

Description

The method `_delegateTokens` is called when a user claims their tokens to automatically delegate the claimed tokens to their own address or to a different one.

code/governance-main-ee5e45a008d65021831de9f3e83053026f2a4dd2/contracts/TokenDistributor.sol:L135

```
_delegateTokens(user_address, delegate_address);
```

The method accepts the addresses of the delegator and the delegate and returns a boolean.

code/governance-main-ee5e45a008d65021831de9f3e83053026f2a4dd2/contracts/TokenDistributor.sol:L262-L270

```
/**
* @notice execute call on token contract to delegate tokens
* @return boolean true on success
*/
function _delegateTokens(address delegator, address delegatee) private returns (bool) {
     GTCErc20  GTCToken = GTCErc20(token);
     GTCToken.delegateOnDist(delegator, delegatee);
     return true;
}
```

But this boolean is never used.

Recommendation

Remove the returned boolean because it's always returned as `true` anyway and the transaction will be a bit cheaper.

4.7 Use a unified compiler version for all contracts Minor Fixed

Resolution

Compiler versions updated to `0.6.12` in gitcoinco/governance#2

Description

Currently the smart contracts for the Gitcoin token and governance use different versions of Solidity compiler (`^0.5.16`, `0.6.12` , `0.5.17`).

Recommendation

It is suggested to use a unified compiler version for all contracts (e.g. `0.6.12`).

Note that it is recommended to use the latest version of Solidity compiler with security patches (currently 0.8.3), although given that these contracts are forks of the battle tested Uniswap governance contracts, the Gitcoin team prefer to keep the modifications to the code at minimum.

4.8 Improve efficiency by using immutable in TreasuryVester Minor Fixed

Resolution

Fixed in gitcoinco/governance#5

Description

The `TreasuryVester` contract when deployed has a few fixed storage variables.

code/governance-main-ee5e45a008d65021831de9f3e83053026f2a4dd2/contracts/TreasuryVester.sol:L30

```
gtc = gtc_;
```

code/governance-main-ee5e45a008d65021831de9f3e83053026f2a4dd2/contracts/TreasuryVester.sol:L33-L36

```
vestingAmount = vestingAmount_;
vestingBegin = vestingBegin_;
vestingCliff = vestingCliff_;
vestingEnd = vestingEnd_;
```

These storage variables are defined in the contract.

code/governance-main-ee5e45a008d65021831de9f3e83053026f2a4dd2/contracts/TreasuryVester.sol:L8

```
address public gtc;
```

code/governance-main-ee5e45a008d65021831de9f3e83053026f2a4dd2/contracts/TreasuryVester.sol:L11-L14

```
uint public vestingAmount;
uint public vestingBegin;
uint public vestingCliff;
uint public vestingEnd;
```

But they are never changed.

Recommendation

Consider setting storage variables as `immutable` type for a considerable gas improvement.

Appendix 1 - Files in Scope

This audit, in addition to the ESMS components, covered the following Solidity files:

File Name	SHA-1 Hash
governance/contracts/GTC.sol	a909f97b7a200d9cf148bc275e48b8e9f800e5e3
governance/contracts/Timelock.sol	501bca9e092f6119425423fbf113dc67537a7872
governance/contracts/GovernorAlpha.sol	b52f893c6d6aa0162e0c3c5e9c0ca698217a456f
governance/contracts/SafeMath.sol	5a3e130059a4672bd4defa577c6ce292a9ef76d6
governance/contracts/TokenDistributor.sol	3015d9659f613d8b262bc8f35ec5f482797af5c4
governance/contracts/TreasuryVester.sol	344c5a1ea9932b9da3ac2433caa8b40c9b7ebad8

Appendix 2 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself 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 Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD 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 ConsenSys and CD 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 Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.