XSEC FINANCE

Smart Contract Audit

Date	April 2021

GITCOIN TOKEN DISTRIBUTION

This report contains the results of our engagements with Gitcoin in reviewing GTC Token Distribution and Governance.

1 | EXECUTIVE SUMMARY

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

Shayan Eskandari, Daniel Luca and all other participants conducted the review over two weeks from April 19, 2021 toApril 30, 2021. Total of 20 person-days were used.

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. You can find the Appendix with a list of Solidity files within scope.

3 | SECURITY SPECIFICATION

This section describes the security implications of the system being audited. This section is not intended to replace documentation. This section identifies security properties that have been validated by the audit team.

3.1 | Actors

Below are the relevant actors and their abilities:

Signer: Signer is used in conjunction with `gitcoin.co` to verify the token distribution and the merkleproofs.
- | The signer can approve (Sign), or reject token claims requests

GTC Minter: Specified during the deployment
- | Can change the Minter Address
- | Can modify and set TokenDistribution Address `GTCDist`
- | After the specified `mintingAllowedAfter` =365 days, Minter can mint `mintCap` = 2 percent of the totalSupply

Users: Users who have a valid user-id at `gitcoin.co` are eligible to be included in the initial distribution
- | They can claim their tokens if they have a valid signature from the Signer
- | They can set the delegator on tokens that allow them to vote in governance
- | You can invoke any ERC20 functionality using GTC token
- | Participate in governance of GTC token

Governance
- | Governance fork of Uniswap
- | The governance mechanism could not be used to change any properties on the chain system at the time the audit was performed.

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

TimeLock Contract:
- | TimeLock TimeLock contract with uniswap
- | All tokens not claimed within 24 weeks (6 month) of launch will be transferred into the TimeLock contract.
- | The assigned admin will control this contract.

TreasuryVester:
- | To vest tokens, fork of Uniswap TreasuryVester

3.2 | Security concerns

Here is a list of security concerns and properties that you might be interested in:

Github login via Gitcoin.co is the entry point to this system. While this is not covered in the audit, it should still be noted that an attacker could gain access to any Github account and claim tokens if the user hasn't claimed them first. It is strongly recommended to conduct a security audit and perform penetration testing on gitcoin.co.

The Ethereum Signed Message Services (ESMS) is a micro-service which will be used to verify token claims. This service is crucial as it holds the signer's private key and all merkle proofs. To be able reconstruct the tokens of each user, anyone with access to these data needs to brute force an integer called user_amount.

4 | FINDINGS

Each issue is assigned a severity:

- **Minor** problems are subjective. These are usually suggestions about best practices or readability. These issues should be addressed by code maintainers.
- **Medium** issues are objective, but they are not security vulnerabilities. These issues should be addressed, unless there are compelling reasons not to.
- Security vulnerabilities are critical issues that can't be exploited directly or require special conditions to be exploited. All of these **Major** problems should be addressed.
- Security vulnerabilities that could be exploited to cause **Critical** issues need to be addressed.

4.1 | ESMS use of sanitized user_amount & user_id values Medium Fixed

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

Description

Signer service values are correctly checked. However, the values are not saved and user input is passed on to the function.

Here, the values are cleaned up:

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')
```

However, the original inputs of the user are 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')
```

Example

if `user_amount` is given a floating amount, all checks will pass. However, the final amount may be slightly lower than what was intended.

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

Recommendation

For the remainder of the code flow, you can use the sanitized result after the sanity test.

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

Fixed in gitcoinco/governance#7

Description

When a user claims an airdrop, the `method _hashLeaf` will be used.

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 accepts the `user_id as arguments` and the `user_amount`.

code/governance-main-ee5e45a008d65021831de9f3e83053026f2a4dd2/contracts/TokenDistributor.sol:L253-L257.sol:L128-L129

```
/**
* @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 can be abi encoded and then hashed together to create 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 compared to the third argument for equality.

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

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

If the hash matches the first argument, it returns true. It also considers that the `user_id` (and `user_amount`) are correct.

But, it is possible to cause collisions by packing different-sized arguments.

Solidity documentation says that collisions can be caused by packing dynamic types, but it is also true for packing `uint32` or `uint256`.

Examples

Here's an example that shows how packing `uint32` and `uint256` both orders can result in collisions with carefully selected 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
        // 0x9e46e582607c5c6e85587dacf66d311c4ced8819378a41d4b4c5adf99d72408e
        return (
            keccak256(packed1),
            keccak256(packed2)
        );
    }
}
```

The transaction will fail if `abi.encodePacked` is changed to `abi.encodePacked` in the library.

Recommendation

Unless you have a special use case for `abi.encodePacked` you should always use it. Although you might require a few extra bytes for transaction data, it prevents collisions. You can use `unit256` to pack both values in order to avoid collisions.

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

Minor **Fixed**

Fixed in gitcoinco/governance#4
Structure Claim can still be removed for further optimization.

Description

`TokenDistributor`'s method `claimTokens` must pass a few tests before it can distribute tokens.

Several of these checks can easily be simplified or optimized.

Because it is only used once, the method `hashMatch` may be deleted and its contents can be moved 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_hex), 'TokenDistributor: Hash Mismatch.');
```

This method also makes use of a few internal calls so they must be transferred to 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:L18

```
hashClaim(claim)
```

Users will save gas by moving the code into the parent method and then removing them.

Because `Claim` isn't used elsewhere in the code, it can be also removed.

Recommendation

Consider simplifying `claimTokens` and remove unused methods.

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

Minor Fixed

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

Description

Variables that create domain separators are hardcoded into the code. It requires the modification code for 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='0xBD2525B5F0B2a663439a78A99A066055549D25cE5')
```

Recommendation

These values can be stored in the environment variable. This allows you to avoid having to modify the source code for different deployments. It can also be scripted to prevent possible errors in the code base.

4.5 | Rename method _hashLeaf to something that represents the validity of the leaf

Minor Fixed

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

Description

The method `_hashLeaf` can accept 3 arguments.

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

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

To create a Keccak256 hash, the arguments `user_id` (and `user_amount`) are required.

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

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

The hash is then compared to the third argument.

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

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

The method returns the result of the equality.

It is confusing to call the method "Return True" if it is valid.

Recommendation

You might consider changing the name of the method to `isValidLeafHash`.

4.6 | Method returns bool but result is never used in TokenDistributor.claimTokens

Minor Fixed

Removed in gitcoinco/governance#4

Description

When a user claims their tokens, the method `_delegateTokens` will be called. It allows them to delegate their tokens automatically to another address.

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

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

This method accepts addresses from the delegator as well as the delegate, and returns a binary.

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;
}
```

This boolean, however, is not used.

Recommendation

The transaction will be slightly cheaper if you remove the returned boolean. It's always `true` anyway.

4.7 | Use a unified compiler version for all contracts

Minor Fixed

Compiler versions updated to 0.6.12 in gitcoinco/governance#2

Description

The smart contracts used for governance and the Gitcoin token use different versions Solidity compilers (`^0.5.16`, `0.6.12` , `0.5.17`).

Recommendation

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

It is strongly recommended that you use the most recent Solidity compiler with security updates (currently 0.8.3). However, these contracts are forks from the battle-tested Uniswap governance agreements, so the Gitcoin team prefers to keep modifications to the code to a minimum.

4.8 | Improve efficiency by using immutable in TreasuryVester

Minor **Fixed**

Fixed in gitcoinco/governance#5

Description

When the `TreasuryVester` contract is deployed, it has some 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 will be 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;
```

They are not changing.

Recommendation

For significant gas improvement, consider setting storage variables to `immutable` type.

APPENDIX 1 - FILES IN SCOPE

The audit included the ESMS components as well as 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 Dialigence ("CD") receives compensation from clients (the Clients) for the analysis performed in these reports (the Reports). Reports can be distributed via ConsenSys publications or other distributions.

Reports are not intended to endorse or indict any project or team. They also do not guarantee security for any project. This Report doesn't consider or have any bearing on the economics of token sales, token tokens, or any other product, services, or assets. Cryptographic tokens, which are emerging technologies, carry high technical risks and uncertainties. Any Report does not provide any representation or warranty to Third-Parties in any way. This includes regarding the bug-free nature of code, any business model or proprietors, or the legal compliance of such businesses. The Reports should not be relied upon by any third party, even if it is used to make decisions about buying or selling tokens, products, services, or assets. This Report is not intended as investment advice and should not be relied on as such. It is also not intended to be used as such as an endorsement of the project or its team. Furthermore, it does not guarantee absolute security. CD is not obligated to any Third-Party for publishing these Reports.

PURPOSE OF THE REPORTS Reports and analysis contained therein are only for Clients. They can be published with their permission. Our review will only cover Solidity code. We are limited to reviewing the Solidity codes we have identified as being included in this report. Solidity language is still under development. It may have flaws and risks. The review does NOT cover the compiler layer or any other areas that could pose security risks beyond Solidity. Cryptographic tokens, which are emerging technologies, carry high technical risk and uncertainty.

CD makes the Reports accessible to clients and other parties (i.e. "third parties") via its website. CD hopes that the public availability of these analyses will help the blockchain ecosystem to develop best practices in this rapidly changing area of innovation.

LINKS TO OTHER WEBSITES FROM THIS WEB site You can, via hypertext or other computer hyperlinks, gain access web sites owned by people other than ConsenSys. These hyperlinks are provided only for your convenience and are not intended to replace the owners of these web sites. ConsenSysys or CD are not responsible or liable for any content or operation of these Web sites. You also agree that ConsenSysys or CD will not be liable for the use of third-party Web sites. Except as stated below, linking from this Web Site to another site does not mean or imply that ConsenSysys or CD endorses that Web site's content or its operator. It is up to you to decide whether or not you can use content from any other websites to which the Reports link. ConsenSys or CD will not be responsible for third-party software used on the Web Site. They also assume no liability for any errors or inaccuracies of any output generated by such software.

TIMELINESS CONTENT. The Reports are current as of the Report's date. However, they can be modified at any time. ConsenSys or CD are the only sources of information, unless otherwise indicated.