Tokenize It

Date	December 2023
Auditors	Sergii Kravchenko, François Legué

1 Executive Summary

This report presents the results of our engagement with **Tokenize It** to review the **Tokenize It smart contracts**.

The review was conducted over three weeks, from **November 20th, 2023** to **December 8th, 2023**, by **Sergii Kravchenko** and **François Legué**. A total of **5 person-weeks** were spent on this audit.

The Tokenize It platform helps companies create and issue or sell tokens that grant economic rights to the token holder. This platform can be used to:

- create an ERC20 token that grants participation rights in the company to the token holders;
- raise funds through public or private offers;
- give employees participation (directly or through a vesting plan).

The platform serves as an intermediate and facilitates the deployment of contracts and the verification of the requirements for the token sender and the token receiver.

2 Scope

Our review focused on the commit hash 9973fc31043e8500bc187d852fc50494f1007f96. The list of files in scope can be found in the Appendix.

During the audit, some changes and fixes were made, and the final codebase has the following commit hash: 19bd32b835ff24ff2c7decf70adbd8ac5968a931.

2.1 Objectives

Together with the **Tokenize It** team, we identified the following priorities for our review:

1. Correctness of the implementation, consistent with the intended functionality and without unintended edge cases.
2. Identify known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry.
3. Cloning and initialization functionality works as expected.
4. Permission checks while transferring tokens work as intended.
5. Vesting commitments work as expected.

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:

- Tokenize It team. Manages the infrastructure required to run correctly, including the frontend enabling users to deploy contracts. The team manages the `AllowList` contracts that help attest address attributes (KYCed, nationality, age, etc.). Tokenize It team controls the `FeeSettings` contract which manages the fees paid to the Tokenize It platform.
- Company or founder. Creates a token that represents participation rights in the company. Determines how the tokens can be emitted (through public fundraising, private investing or giving tokens to employees)
- Investor. Buys tokens of a company through a public or private fundraising / offer.
- Employee. Works for a company and receives tokens as part of their compensation.

3.2 Trust Model

In any system, it's important to identify what trust is expected/required between various actors. For this audit, we established the following trust model:

- Tokenize It team. The team is trusted in attesting and inserting attributes associated with addresses in the `AllowList` contract. The team is also in charge of legally validating the companies and founders that want to create a token.
- Company or founder. The company owner or founder who wants to emit tokens is trusted to operate correctly. The central component of the Tokenize It project is the `Token` contract. It is owned by the company owner or founder. It is worth noting that this contract can be upgraded and its logic modified.

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 Limiting the Price in the `buy` and `onTokenTransfer` Functions Medium ✓ Fixed

Resolution

Fixed here for the `buy` function by adding a `_maxCurrencyAmount` variable check. It is also mitigated in the `onTokenTransfer` function. But since you cannot add an extra argument to this function, the minimal token amount is optionally added to the `_data` parameter. This parameter became a bit complicated and remains optional. So, the result of the direct token transfer with an empty `_data` can theoretically be manipulated by the owner.

Description

When an investor tries to buy the tokens in the `Crowdinvesting` contract, the `buy` function does not allow to limit the amount of tokens that can be spent during this particular transaction:

contracts/Crowdinvesting.sol:L277-L279

```
function buy(uint256 _amount, address _tokenReceiver) public whenNotPaused nonReentrant {
    // rounding up to the next whole number. Investor is charged up to one currency bit more in case of a fractional currency bit
    uint256 currencyAmount = Math.ceilDiv(_amount * getPrice(), 10 ** token.decimals());
```

The owner of the price oracle can front-run the transaction and twist the price.

Of course, the buyer can try to regulate that limit with the token allowance, but there may be some exceptions. Sometimes, users want to give more allowance and buy in multiple transactions over time. Or even give an infinite allowance (not recommended) out of convenience.

The same issue can be found in the `onTokenTransfer` function. This function works differently because the amount of currency is fixed, and the amount of tokens minted is undefined. Because of that, limiting the allowance won't help, so the user doesn't know how many tokens can be bought.

Recommendation

It's recommended to explicitly limit the amount of tokens that can be transferred from the buyer for the `buy` function. And allow users to define a minimal amount of tokens bought in the `onTokenTransfer` function.

4.2 Potential Re-Entrancy Attack in the `Crowdinvesting` Contract Minor ✓ Fixed

Resolution

Fixed by storing the currency early in the function to the memory and reusing that value.

Description

The attack requires a set of pre-requisites:

1. The currency token should have a re-entrancy opportunity inside the token transfer.
2. The re-entrancy can be done on a token transfer from the `_msgSender()` to the `feeCollector`, so there are not a lot of attackers who can potentially execute it.
3. The owner should be involved in the attack, so it's most likely an attack by the owner.

contracts/Crowdinvesting.sol:L277-L290

```
function buy(uint256 _amount, address _tokenReceiver) public whenNotPaused nonReentrant {
    // rounding up to the next whole number. Investor is charged up to one currency bit more in case of a fractional currency bit
    uint256 currencyAmount = Math.ceilDiv(_amount * getPrice(), 10 ** token.decimals());

    (uint256 fee, address feeCollector) = _getFeeAndFeeReceiver(currencyAmount);
    if (fee != 0) {
        currency.safeTransferFrom(_msgSender(), feeCollector, fee);
    }

    currency.safeTransferFrom(_msgSender(), currencyReceiver, currencyAmount - fee);
    _checkAndDeliver(_amount, _tokenReceiver);

    emit TokensBought(_msgSender(), _amount, currencyAmount);
}
```

So on the token transfer to the `feeCollector` above, the `currency` parameter can be changed by the `owner`. And the following token transfer (`currency.safeTransferFrom(_msgSender(), currencyReceiver, currencyAmount - fee);`) will be made in a different currency.

A possible scenario of the attack could look as follows:

1. Malicious owner sells tokens for a valuable currency. People are placing allowance for the tokens.
2. The owner changes the currency to a new one with a much lower price and re-entrancy during transfer.
3. When a victim wants to buy tokens, the owner reenters on fee transfer and returns the old currency.
4. The victim transfers the updated currency that is more expensive.

Recommendation

Save the currency in memory at the beginning of the function and use it further.

4.3 Lack of Validation of `PrivateOffer` Initialization Parameters Minor ✓ Fixed

Resolution
Addressed by adding extra validation of the parameters.

Description

The `PrivateOffer` contract allows to create a customised deal for a specific investor. The `initialize()` function receives parameters to set up the `PrivateOffer` accordingly.

The following parameters lack of validation during initialization:

- `tokenAmount`
- `token`
- `currency`

`tokenAmount`

contracts/PrivateOffer.sol:L81-L84

```
uint256 currencyAmount = Math.ceilDiv(
    _arguments.tokenAmount * _arguments.tokenPrice,
    10 ** _arguments.token.decimals()
);
```

`tokenAmount` is not validated at all. It should be verified to be greater than zero.

`token`

`token` is not validated at all. It should be verified to be different than zero address.

`currency`

`currency` is not validated at all. The documentation mentions a restricted list of supported currencies. It should be enforced by checking this parameter against a whitelist of currency addresses.

Recommendation

Enhance the validation of the following parameters: `tokenAmount`, `token`, `currency`.

4.4 Lack of Validation of `Crowdinvesting` Initialization Parameters Minor ✓ Fixed

Resolution
Mitigated by adding extra validation.

Description

The `Crowdinvesting` contract allows everyone who meets the requirements to buy tokens at a fixed price. The `initialize()` function receives parameters to set up the `Crowdinvesting` accordingly.

The following parameters lack of validation during initialization:

- `tokenPrice`
- `minAmountPerBuyer`
- `lastBuyDate`
- `currency`

`tokenPrice`

contracts/Crowdinvesting.sol:L160

```
require(_arguments.tokenPrice != 0, "_tokenPrice needs to be a non-zero amount");
```

`tokenPrice` is checked to be different to zero. It should be verified to be in between `priceMin` and `priceMax` when these parameters are provided.

`minAmountPerBuyer`

contracts/Crowdinvesting.sol:L156-L159

```
require(
    _arguments.minAmountPerBuyer <= _arguments.maxAmountPerBuyer,
    "_minAmountPerBuyer needs to be smaller or equal to _maxAmountPerBuyer"
);
```

`minAmountPerBuyer` is checked to be below or equal to `maxAmountPerBuyer`. It should be verified to not be zero.

`lastBuyDate`

contracts/Crowdinvesting.sol:L172

```
lastBuyDate = _arguments.lastBuyDate;
```

`lastBuyDate` is not validated at all. It should be verified to be greater than the current `block.timestamp`. Currently, a `Crowdinvesting` contract with `lastBuyDate` parameter set to a value (different than zero) below `block.timestamp` will not be able to sell any token.

contracts/Crowdinvesting.sol:L249-L265

```
function _checkAndDeliver(uint256 _amount, address _tokenReceiver) internal {
    require(tokensSold + _amount <= maxAmountOfTokenToBeSold, "Not enough tokens to sell left");
    require(tokensBought[_tokenReceiver] + _amount >= minAmountPerBuyer, "Buyer needs to buy at least minAmount");
    require(
        tokensBought[_tokenReceiver] + _amount <= maxAmountPerBuyer,
        "Total amount of bought tokens needs to be lower than or equal to maxAmount"
    );

    if (lastBuyDate != 0 && block.timestamp > lastBuyDate) {
        revert("Last buy date has passed: not selling tokens anymore.");
    }

    tokensSold += _amount;
    tokensBought[_tokenReceiver] += _amount;

    token.mint(_tokenReceiver, _amount);
}
```

`currency`

contracts/Crowdinvesting.sol:L154

```
require(address(_arguments.currency) != address(0), "currency can not be zero address");
```

`currency` is checked to be different than zero. The documentation mentions a restricted list of supported currencies. It should be enforced by checking this parameter against a whitelist of currency addresses.

Recommendation

Enhance the validation of the following parameters: `tokenPrice`, `tokenPrice`, `lastBuyDate`, `currency`.

4.5 Keeping Denominators of Fees Is Redundant ✓ Fixed

Resolution
Addressed by major code refactoring of the `FeeSettings` contract.

Description

All the fees are stored as numerators and denominators. That requires storing two numbers instead of just one. That solution increases code size and makes a comparison of two different values more complicated:

../code-a3/contracts/FeeSettings.sol:L264-L269

```
defaultTokenFeeNumerator = proposedDefaultFees.tokenFeeNumerator;
defaultTokenFeeDenominator = proposedDefaultFees.tokenFeeDenominator;
defaultCrowdinvestingFeeNumerator = proposedDefaultFees.crowdinvestingFeeNumerator;
defaultCrowdinvestingFeeDenominator = proposedDefaultFees.crowdinvestingFeeDenominator;
defaultPrivateOfferFeeNumerator = proposedDefaultFees.privateOfferFeeNumerator;
defaultPrivateOfferFeeDenominator = proposedDefaultFees.privateOfferFeeDenominator;
```

../code-a3/contracts/FeeSettings.sol:L306-L313

```
function _isFractionAGreater(
    uint32 aNumerator,
    uint32 aDenominator,
    uint32 bNumerator,
    uint32 bDenominator
) internal pure returns (bool) {
    return uint256(aNumerator) * bDenominator > uint256(bNumerator) * aDenominator;
}
```

Recommendation

Instead, the common practice is to fix the denominator for all values as a constant, for example, 10e5. That would keep the code simpler, shorter, and cheaper.

4.6 Non-Normalized Salt Computation ✓ Fixed

Resolution
Addressed by using `abi.encode`.

Description

The deployment of new contracts (`Crowdinvesting`, `PriceLinear`, `PrivateOffer` and `Vesting`) relies on the cloning and `create2` features. The `create2` opcode gives the ability to predict the address of a contract given its bytecode, the address of the deployer and a salt.

Currently, the salt is computed doing a `keccak256` hash of encoded parameters that will define the characteristics of the contract to be deployed. However, the encoding method is not consistent across different contracts.

`CrowdinvestingCloneFactory` salt computation is based on `abi.encode` encoding of parameters

contracts/factories/CrowdinvestingCloneFactory.sol:L62-L69

```
    function _getSalt(
        bytes32 _rawSalt,
        address _trustedForwarder,
        CrowdinvestingInitializerArguments memory _arguments
    ) internal pure returns (bytes32) {
        return keccak256(abi.encode(_rawSalt, _trustedForwarder, _arguments));
    }
}
```

`PriceLinearCloneFactory` salt computation is based on `abi.encodePacked` encoding of parameters

contracts/factories/PriceLinearCloneFactory.sol:L122-L147

```
function _generateSalt(
    bytes32 _rawSalt,
    address _trustedForwarder,
    address _owner,
    uint64 _slopeEnumerator,
    uint64 _slopeDenominator,
    uint64 _startTimeOrBlockNumber,
    uint32 _stepDuration,
    bool _isBlockBased,
    bool _isRising
) internal pure returns (bytes32) {
    return
        keccak256(
            abi.encodePacked(
                _rawSalt,
                _trustedForwarder,
                _owner,
                _slopeEnumerator,
                _slopeDenominator,
                _startTimeOrBlockNumber,
                _stepDuration,
                _isBlockBased,
                _isRising
            )
        );
}
```

`PrivateOfferFactory` salt computation is based on `abi.encode` encoding of parameters

contracts/factories/PrivateOfferFactory.sol:L170-L182

```
function _getSalt(
    bytes32 _rawSalt,
    PrivateOfferArguments calldata _arguments,
    uint64 _vestingStart,
    uint64 _vestingCliff,
    uint64 _vestingDuration,
    address _vestingContractOwner
) private pure returns (bytes32) {
    return
        keccak256(
            abi.encode(_rawSalt, _arguments, _vestingStart, _vestingCliff, _vestingDuration, _vestingContractOwner)
        );
}
```

`TokenFactory` salt computation is based on `abi.encodePacked` encoding of parameters

contracts/factories/TokenProxyFactory.sol:L125-L148

```
function _getSalt(
    bytes32 _rawSalt,
    address _trustedForwarder,
    IFeeSettingsV2 _feeSettings,
    address _admin,
    AllowList _allowList,
    uint256 _requirements,
    string memory _name,
    string memory _symbol
) private pure returns (bytes32) {
    return
        keccak256(
            abi.encodePacked(
                _rawSalt,
                _trustedForwarder,
                _feeSettings,
                _admin,
                _allowList,
                _requirements,
                _name,
                _symbol
            )
        );
}
```

`VestingCloneFactory` salt computation is based on `abi.encodePacked` encoding of parameters

contracts/factories/VestingCloneFactory.sol:L31

```
bytes32 salt = keccak256(abi.encodePacked(_rawSalt, _trustedForwarder, _owner, _token));
```

Recommendation

Normalize the computation of the salt that will be used along with `create2` feature. Note, as a reminder, that it is preferable to use `abi.encode` in order to prevent any potential hash collisions when encoding dynamic types variables.

4.7 Unused or Redundant Imports in Multiple Contracts ✓ Fixed

Resolution
Fixed.

Description

Multiple contracts import unused or redundant libraries.

`PrivateOffer`

contracts/PrivateOffer.sol:L4

```
import "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol";
```

`PriceLinear`

contracts/PriceLinear.sol:L6

```
import "@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol";
```

`VestingCloneFactory`

contracts/factories/VestingCloneFactory.sol:L7

```
import "@openzeppelin/contracts/proxy/Clones.sol";
```

`PrivateOfferFactory`

contracts/factories/PrivateOfferFactory.sol:L4-L5

```
import "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol";
import "@openzeppelin/contracts/utils/math/SafeCast.sol";
```

`PriceLinearCloneFactory`

contracts/factories/PriceLinearCloneFactory.sol:L7

```
import "@openzeppelin/contracts/proxy/Clones.sol";
```

`CrowdinvestingCloneFactory.sol`

contracts/factories/CrowdinvestingCloneFactory.sol:L7

```
import "@openzeppelin/contracts/proxy/Clones.sol";
```

Recommendation

Remove unused or redundant imports.

4.8 Missing Events on Important State Changes ✓ Fixed

Resolution
Events added.

Description

The `setLastBuyDate()` function from `Crowdinvesting` contract updates the `lastBuyDate` state variable without emitting an event.

`setLastBuyDate()`

contracts/Crowdinvesting.sol:L390-L393

```
function setLastBuyDate(uint256 _lastBuyDate) external onlyOwner whenPaused {
    lastBuyDate = _lastBuyDate;
    coolDownStart = block.timestamp;
}
```

Recommendation

Emit an event on important state change.

4.9 `DynamicPricingActivated` Event Is Emitted Twice ✓ Fixed

Resolution
Fixed.

Description

The `DynamicPricingActivated(address,uint256,uint256)` event is emitted twice when activating the dynamic price feature.

`DynamicPricingActivated`

contracts/Crowdinvesting.sol:L186-L212

```
function activateDynamicPricing(
    IPriceDynamic _priceOracle,
    uint256 _priceMin,
    uint256 _priceMax
) external onlyOwner whenPaused {
    _activateDynamicPricing(_priceOracle, _priceMin, _priceMax);
    coolDownStart = block.timestamp;
    emit DynamicPricingActivated(address(_priceOracle), _priceMin, _priceMax);
}

/**
 * Activates dynamic pricing and sets the price oracle, as well as the minimum and maximum price.
 * @param _priceOracle this address is queried for the current price of a token
 * @param _priceMin price will never be less that this
 * @param _priceMax price will never be more than this
 */
function _activateDynamicPricing(IPriceDynamic _priceOracle, uint256 _priceMin, uint256 _priceMax) internal {
    require(address(_priceOracle) != address(0), "_priceOracle can not be zero address");
    priceOracle = _priceOracle;
    require(_priceMin <= priceBase, "priceMin needs to be smaller or equal to priceBase");
    priceMin = _priceMin;
    require(priceBase <= _priceMax, "priceMax needs to be larger or equal to priceBase");
    priceMax = _priceMax;
    coolDownStart = block.timestamp;
    emit DynamicPricingActivated(address(_priceOracle), _priceMin, _priceMax);
}
```

Recommendation

Remove `DynamicPricingActivated` event emitted in the `activateDynamicPricing()` function.

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
contracts/AllowList.sol	`766f10f387c4c2ada81e4bed6eaa3214a14bd0b7`
contracts/Crowdinvesting.sol	`d21eb07948cf5d391050200b9ced55a05d759c78`
contracts/FeeSettings.sol	`6977c2542990472b6dc4ba8e61c08f7200e4cf03`
contracts/PriceLinear.sol	`bfbb2c3864bf99f0eb0b56a066f10ed262dd289f`
contracts/PrivateOffer.sol	`cd046028a956313e01a3f70cc806b871ee08a8b2`
contracts/Token.sol	`01c8969df1f4c3fa71d4e3e33b6e0c22aa8753e4`
contracts/Vesting.sol	`049afb346463c1270c3d9de65ef288d73c891327`
contracts/factories/CloneFactory.sol	`94ee64e1052eb9caa0e6a3ddb9755f202520de01`
contracts/factories/CrowdinvestingCloneFactory.sol	`38465e3ea22dcc400c4d70b28d64dd9078af4797`
contracts/factories/Factory.sol	`67dd11155cb1bcf3a7f68ba57a15db13a8269775`
contracts/factories/PriceLinearCloneFactory.sol	`599549e59e206ea61ec55cc7f1a7b21ea9a49829`
contracts/factories/PrivateOfferFactory.sol	`d84fadcc440b920d15b06db1c0ad85875b235715`
contracts/factories/TokenProxyFactory.sol	`b5f968ba31c5de5c50edf63c22884e39da07c4e3`
contracts/factories/VestingCloneFactory.sol	`5413e9e1057c0304a8aa3755b8577549930cf082`
contracts/interfaces/IFeeSettings.sol	`d4268e7e9dbc233f75041c14c5c18862b778930f`
contracts/interfaces/IPriceDynamic.sol	`243a304b240a2944481770427977c0a15d68813e`

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 bug-free 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.

A.2.1 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 code and only the code we note as being within the scope of our review within this report. Any Solidity code itself presents unique and unquantifiable risks as 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 specified code

that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. In some instances, we may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

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.

A.2.2 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.

A.2.3 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.