Open Zeppelin v2.0

Security Audit

Level K
146 Magazine Street,
Apt. 2
Cambridge, MA 02139
www.levelk.io

Version
1.1

Date
October 21st, 2018

openzeppelin

Prepared by
Chris Whinfrey
Paul Cowgill
Shane Fontaine

audits@levelk.io

LEVELK

Introduction

The Zeppelin team asked us to review and audit all of the smart contracts contained in their widely used OpenZeppelin library in order to prepare it for the OpenZeppelin v2.0 release.

The audited code is located in the OpenZeppelin/openzeppelin-solidity repository. The version used for this report is commit **`dac5bccf803696d9d98d269b8c27c7aac5fa1c5c`**.

The Zeppelin team did a great job of being consistent with their style, even though there are over 150 contributors to the repository. This consistency of the contracts combined with the well-written code allowed us to focus on the critical pieces of the codebase during the audit. We commend the team for their ability to write modular code while keeping it both simple and usable. We are very pleased with the team's communication with us throughout the entire process, as it allowed for a more continuous audit flow and quick clarification whenever it was needed.

We found one critical issue in BreakInvariantBounty.sol that was susceptible to frontrunning by a malicious party that may lead to a loss of funds. After independently coming up with the issue, we worked with the team and an existing PR to take the best course of action given the situation. The Zeppelin team removed the contract until a better solution has been found. Our goal with this audit was twofold: remove any vulnerabilities that may be found in the existing framework and help future developers easily deploy these contracts as they are intended to be used. Our suggestions that follow range from gas optimizations and comment clarifications to recommended fixes of potentially exploitable code.

Update: *The Zeppelin team has followed most of our recommendations and updated their contracts appropriately.*

Issues Overview

Open Zeppelin v2.0

Number of Issues per severity level

	LOW	MEDIUM	HIGH	CRITICAL
Open	3	0	0	0
Closed	16	4	2	1

Issues by severity level

Directory	Issue Title	Status	Severity
drafts	Avoid frontrunning by a malicious actor or the contract owner	Resolved	CRITICAL
crowdsale	Stop crowdsale manipulation via reentrancy by adding the nonReentrant modifier to the buyTokens() function	Resolved	HIGH
token	Allow for safe changes to allowances through the SafeERC20 interface	Resolved	HIGH
crowdsale	Require closingTime to be strictly greater than openingTime	Resolved	MEDIUM
crowdsale	Return the true value of remainingTokens()	Resolved	MEDIUM
crowdsale	Prevent reentrancy in finalize()	Resolved	MEDIUM

token	Make _clearApproval() private	Resolved	MEDIUM
access	Role contracts emit events when a role has already been assigned or unassigned	Resolved	LOW
crowdsale	Use the internal keyword to force correct contract usage	Resolved	LOW
crowdsale	Mark validation functions as view to ensure they don’t change state	Resolved	LOW
crowdsale	Require initialRate is strictly greater than finalRate	Resolved	LOW
crowdsale	Consider overriding the rate() function to avoid any confusion	Resolved	LOW
drafts	Checks-Effects-Interactions	Resolved	LOW
drafts	Consider being more explicit upon contract creation to force correct usage of the contract	Resolved	LOW
drafts	Consider being explicit about the number of tokens that are meant to exist within the contract	Resolved	LOW
introspection	_supportedInterfaces should be private	Resolved	LOW
math	Additional test cases are needed for SafeMath to have full coverage	Resolved	LOW
ownership	Allow subclasses to renounce ownership	Open	LOW

ownership	Secondary's constructor should be internal because it is meant to be extended	Resolved	LOW
payment	Explicitly prevent adding payees late	Resolved	LOW
payment	PullPayment's constructor should be internal because it is meant to be extended.	Resolved	LOW
token	Cast 0 to an address type on lines 169 and 182	Open	LOW
token	Expose an internal _transfer() function	Resolved	LOW
token	Unused function _burn()	Resolved	LOW
token	ERC20Capped should override _mint() instead of mint()	Resolved	LOW
utils	Consider changing the name of isContract()	Open	LOW

Issues
Open Zeppelin v2.0

access/

LOW

1. Role contracts emit events when a role has already been assigned or unassigned

Roles.sol#L16-L27 - In Roles.sol, preventing add() from adding already assigned roles and preventing remove() from removing unassigned roles will keep the role contracts from emitting false events such as CapperAdded when the role was already assigned to the added address.

Update: resolved in #1421

Notes:

- CapperRole.sol#L15 - Emit a `CapperAdded` event in the `CapperRole` constructor so that the contract's set of cappers can be determined from the contract's events. This recommendation applies to `MinterRole`, `PauserRole`, and `SignerRole` as well.

  Update: resolved in #1329

crowdsale/

Crowdsale.sol

HIGH

1. Stop crowdsale manipulation via reentrancy by adding the nonReentrant modifier to the buyTokens() function

The Crowdsale contract is at risk for reentrancy if a call to an unknown address is made during the execution of the `buyTokens()` function. One way this could happen is if the ERC20 token executes code at the receiver's address when it is transferred. An example of this class of token is described by the ERC677 standard, an extension of ERC20. Reentrancy would allow a malicious actor to bypass protections such as `IndividuallyCappedCrowdsale`'s purchase cap. Consider adding the `nonReentrant` modifier to the `buyTokens()` function to ensure this attack is not possible.

Update: resolved in #1438

LOW

2. Use the internal keyword to force correct contract usage

Because all of the crowdsale contracts are meant to be extended, consider making every crowdsale contract's constructor `internal`.

Update: resolved in #1439

LOW

3. Mark validation functions as view to ensure they don’t change state

Consider using the view keyword on both `_preValidatePurchase` and `_postValidatePurchase`. All state changes should be made in `_updatePurchasingState`, so enforcing view for the validation functions ensures that state changes are implemented where they should be.

Update: resolved in #1439

Notes:

- *Comment:* Crowdsale.sol#L78 - Crowdsale’s fallback function can be used to purchase tokens but requires more than 2300 gas. Any tokens purchased via `transfer()` from another contract will fail due to the imposed gas limit. One case where this might be implemented is a contract that pools funds to make a group purchase with a single transaction. A comment advising users to use the `buyTokens()` function when purchasing from another contract will help avoid unexpected transaction failures.

  Update: resolved in #1446

- *Typo:* Crowdsale.sol#L104 - the mount -> the amount

  Update: resolved in #1446

- *Grammar:* The natspec comment for `buyTokens` says `@param beneficiary Address performing the token purchase`, but the beneficiary doesn't necessarily perform the buyTokens transaction. The `msg.sender` "performs" it. Consider rewording the comment to say that the beneficiary is the address that will be receiving the purchased ERC20 tokens.

  Update: resolved in #1446

- *Grammar:* `Not necessarily emits/sends` -> `Does not necessarily emit/send`

  Update: resolved in #1446

crowdsale/validation/

TimedCrowdsale.sol

MEDIUM

1. Require closingTime to be strictly greater than openingTime

TimedCrowdsale.sol#L33 - `closingTime` should be strictly greater than `openingTime`. If opening time and closing time are equal, `IncreasingPriceCrowdsale`'s `getCurrentRate()` will always revert due to division by 0.

Update: resolved in #1440

crowdsale/price/

IncreasingPriceCrowdsale.sol

LOW

1. Require initialRate is strictly greater than finalRate

IncreasingPriceCrowdsale.sol#L26 - Make `initialRate` strictly greater than `finalRate` (that is, the price should increase by some amount or `TimedCrowdsale` could be used instead).

Update: resolved in #1441

LOW

2. Consider overriding the rate() function to avoid any confusion

IncreasingPriceCrowdsale.sol#L50 - The `rate()` function will return the static rate that is passed into `Crowdsale`'s constructor and is never used. This will differ from what is returned from `getCurrentRate()`. Consider overriding `rate()` to revert in `IncreasingPriceCrowdsale`.

Update: resolved in #1441

Notes:

- IncreasingPriceCrowdsale.sol#L50 - We believe `getCurrentRate()` should return `0` when called outside of the crowdsale time period. The current implementation will throw in some cases or return a nonzero rate in others. Ideally the function should return early like this: `if (!isOpen()) { return 0 }`

  Update: resolved in #1442

crowdsale/emission/

AllowanceCrowdsale.sol

MEDIUM

1. Return the true value of remainingTokens()

AllowanceCrowdsale.sol#L40 - `remainingTokens()` may return more tokens than the `_tokenWallet` address contains. It should return the minimum (using `Math.min`) of the `_tokenWallet`'s `balance` and the `allowance`.

Update: resolved in #1449

crowdsale/distribution/

RefundableCrowdsale.sol

Notes:

- *Note:* RefundableCrowdsale.sol#L44 - For `claimRefund`, `beneficiary` is a loaded word that implies the intended beneficiary from the escrow's perspective. This is because `Escrow` has a `beneficiaryWithdraw` function and it isn't the same `beneficiary` that is meant here. Consider calling the parameter `refundee` in the `claimRefund` function of this contract.

crowdsale/distribution/

PostDeliveryCrowdsale.sol

Notes:

- *Note:* PostDeliveryCrowdsale.sol#L4 - Consider removing the `IERC20` import, as it is never used.

  Update: resolved in #1437

crowdsale/distribution/

FinalizableCrowdsale.sol

MEDIUM

1. Prevent reentrancy in finalize()

FinalizableCrowdsale.sol#L37 - We recommend setting `_finalized` to true before calling `_finalization()`. If `_finalization()` is overridden to make a call to an unknown address, a malicious actor could reenter `finalize()`. One case where this may happen is if the caller of `finalize()` is rewarded with a small ETH payment.

Update: resolved in #1447

Notes:

- *Note:* FinalizableCrowdsale.sol#L15 - We recommend removing the right-hand operand of `_finalized` in order to save gas on deployment.

  Update: resolved in #1403

drafts/

BreakInvariantBounty.sol

CRITICAL

1. Avoid frontrunning by a malicious actor or the contract owner

BreakInvariantBounty.sol#L49 - Claims can be frontrun by both a malicious third-party and the contract owner. A malicious third-party can frontrun the claim by repeating the researcher's transactions that deploy the target, break the invariant, and make the claim with higher gas prices. The contract owner can frontrun a claim with a call to `destroy()`, revoking the bounty before the researcher can be rewarded while the researcher already revealed the broken invariant.

Update: resolved in #1424. This contract will be removed while new approaches are considered.

LOW

2. Checks-Effects-Interactions

BreakInvariantBounty.sol#L55: `_claimed` should be moved above `_asyncTransfer(researcher, address(this).balance);` in order to comply with the "check-effects-interaction" rule.

Update: resolved in #1424. This contract will be removed while new approaches are considered.

drafts/

SignatureBouncer.sol

Notes:

- *Comment:* SignatureBouncer.sol - Add a comment warning that users are responsible for preventing replay attacks when inheriting from this contract.

  Update: resolved in #1434

drafts/

TokenVesting.sol

LOW

1. Consider being more explicit upon contract creation to force correct usage of the contract

TokenVesting.sol#L45 - Consider adding `require(start.add(duration) > now);` and `require(duration > 0);` to the constructor. As it stands, if either were to return false, the contract would allow the beneficiary to claim all tokens immediately, which is likely not desired (this can be achieved with a simple transaction). Adding this check adds an additional sanity check to confirm that the contract executes as expected.

Update: resolved in #1431

LOW

2. Consider being explicit about the number of tokens that are meant to exist within the contract

TokenVesting.sol#L162 - `vestedAmount()` returns a different amount if tokens are added to the contract. The beneficiary could withdraw their currently vested tokens and send them back to the contract in order to increase the amount returned by `vestedAmount()`. Changing the function to `vestedPercentage()` and using that to calculate `releasableAmount()` will prevent unexpected manipulation of the amount returned by `vestedAmount()`.

Update: resolved in #1427

Notes:

- *Note:* TokenVesting.sol#L20-L21 - Since the contract can accept and pay out multiple types of tokens, we recommend adding a variable `tokenAddress` to both the `Released()` and `Revoked()` events.

  Update: resolved in #1431

- *Note:* TokenVesting.sol#L163: We recommend adding an `address` typecast to `this`.

examples/

Notes:

- *Note:* Both SampleCrowdsaleToken and SimpleToken declare state variables for `name`, `symbol`, and `decimals`. We recommend inheriting from `ERC20Detailed` instead to demonstrate its usage.

  Update: resolved in #1448

- *Comment:* SampleCrowdsale.sol#L28 - `MintedCrowdsale` is not listed as an extension in the comment.

  Update: resolved in #1448

introspection/

LOW

1. _supportedInterfaces should be private

ERC165.sol#L22 - `_supportedInterfaces` can be `private` for increased encapsulation.

Update: resolved in #1379

Notes:

- *Typo:* ERC165.sol#L46 - `_registerInterface` comment says `@dev private method` but it's `internal`. The comment should be changed to say `internal`.

  Update: resolved in #1422

- *Style:* ERC165Checker.sol#L44-L81 - `supportsInterfaces()` as a name is very similar to `supportsInterface()` which may be error prone. Consider renaming the function to something slightly more verbose but easily distinguishable, like `supportsManyInterfaces`.

  Update: resolved in #1435

- *Style:* ERC165Checker.sol#L94-L147 - Adding underscores to `ERC165Checker`'s private functions and changing `supportsERC165Interface()` to `_supportsInterface()` will help differentiate the functions and their usages.

  Update: resolved in #1435

lifecycle/

Pausable.sol

Notes:

- *Note:* Pausable.sol#L11-L12 - In `Paused` and `Unpaused` events, consider including the pauser's address in the event since there can be multiple pausers.

  Update: resolved in #1410

- *Note:* Subclasses of Pausable currently have no access to the `_paused` state variable. Adding internal functions for `_pause` and `_unpause` would allow for subclasses that provide additional functionality. (e.g. a contract that allows for unpausing by any address after a time period has expired)

math/

Math.sol

Notes:

- *Comment:* Consider adding natural specification comments to each function in `Math.sol` for consistency and clarity.

  Update: resolved in #1423

math/

SafeMath.sol

LOW

1. Additional test cases are needed for SafeMath to have full coverage

The following test cases are needed for SafeMath to have full coverage:

- `div` with numbers that aren't divisible evenly
- `div` with the first argument being 0
- `mul` with the second argument being 0

Update: Tracked in issue #1386

ownership/

Ownable.sol

LOW

1. Allow subclasses to renounce ownership

Subclasses of `Ownable` are able to transfer ownership but are not able to set `_owner` to a 0 address like `renounceOwnership` does. Consider exposing an internal function to allow `Ownable` subclasses to remove the owner.

Notes:

- *API:* The ERC20 and ERC721 standards require a `Transfer` event be emitted when tokens are created or destroyed. Consider following this pattern with `Ownable` and emitting an `OwnershipTransferred` event from the 0 address when `_owner` is set in the constructor and to the 0 address in `renounceOwnership`. This allows off-chain applications to recreate the ownership state from the events.

  Update: resolved in #1397

ownership/

Secondary.sol

LOW

1. Secondary's constructor should be internal because it is meant to be extended

Update: resolved in #1433

Notes:

- *Comments:* Consider adding natural specification comments to the `primary()` and `transferPrimary()` functions.

  Update: resolved in #1425

- *API:* `transferPrimary()` in `Secondary.sol` should emit an event.

  Update: resolved in #1425

payment/

RefundEscrow.sol

Notes:

- *Note:* RefundEscrow.sol#L13 - `RefundEscrow` is already `Secondary` via `ConditionalEscrow`. Remove the redundant inheritance of `Secondary`.

  Update: resolved in #1381

- *Style:* We recommend replacing all `_state` checks such as `require(_state == State.Active);` with a modifier. (e.g. `isState(State.Active)`)

- *Style:* RefundEscrow.sol#L16 - `Closed` is the name of both a state and an event. Consider renaming the event to `RefundClosed` to avoid confusion later on in the code.

  Update: resolved in #1418

payment/

Escrow.sol

Notes:

- *Word choice:* Escrow.sol#L9 - Use more common English word choice: “destinated to” -> “sent to”

  Update: resolved in #1430

payment/

SplitPayment.sol

LOW

1. Explicitly prevent adding payees late

SplitPayment.sol#L101 - Adding payees after payments have been released is not supported by the current code. Consider making this function private or requiring that `_totalReleased` is 0. If the second option is chosen, adding payees after funds have been received but not released will dilute existing payees which may or may not be the desired behavior.

Update: resolved in #1417

Notes:

- *Naming:* We recommend being more clear about the desired functionality of `SplitPayment.sol`. An unknowing user may not realize that funds can be added throughout the lifetime of the contract. Consider changing the name of the contract to `SplitPayments.sol`.

  Update: resolved in #1417

- *Note:* Adding `PayeeAdded`, `PaymentReceived`, and `PaymentReleased` events will log interactions with this contract.

  Update: resolved in #1417

payment/

PullPayment.sol

LOW

1. PullPayment‘s constructor should be internal because it is meant to be extended.

Update: resolved in #1433

token/

ERC20.sol

LOW

1. Cast 0 to an address type on lines 169 and 182

LOW

2. Expose an internal _transfer() function

Exposing an internal `_transfer()` function may be useful when subclassing ERC20 for use cases such as security tokens where a central operator may need to reverse a transfer or recover frozen shares. Additionally, an internal function such as `_clearAllowance()` may be useful when a transaction requires a certain amount of token be approved for transfer but, in some cases, transfers none or a fraction of the approved amount.

Update: An internal _transfer() function was added in #1370

token/

ERC20Burnable.sol

LOW

1. Unused function _burn()

ERC20Burnable.sol#L33 - The `_burn()` function is not changed by the override in `ERC20Burnable` and is no longer emitting an event. Consider removing this function override.

Update: resolved in #1373

token/

ERC20Capped.sol

LOW

1. ERC20Capped should override _mint() instead of mint()

Now that the base ERC20 contract implements `_mint()`, `ERC20Capped` can inherit directly from the base ERC20 contract and override `_mint()` instead of `mint()`. This will ensure the cap is not exceeded even when tokens are minted through functions other than `mint()`.

Update: resolved in #1443

token/

SafeERC20.sol

HIGH

1. Allow for safe changes to allowances through the SafeERC20 interface

`SafeERC20`'s `safeApprove` is still susceptible to this attack and may be misleading. Reverting when the allowance is not being set to or from 0 will help protect users from this vulnerability. Additionally, adding `safeIncreaseAllowance` and `safeDecreaseAllowance` functions will allow users to still safely adjust allowances when they are not setting the allowance to or from 0.

Update: resolved in #1407

Notes:

- *Note:* SafeERC20.sol#L3 - Importing `ERC20.sol` is unnecessary.

  Update: resolved in #1437

token/ERC721/

ERC721.sol

MEDIUM

1. Make _clearApproval() private

Subclasses that call `_clearApproval()` without emitting an `Approval` event will not be ERC721 compliant. Consider making this function `private` or emitting an `Approval` event in `_clearApproval()`.

Update: resolved in #1450

Notes:

- *Note:* `_checkAndCallSafeTransfer` does not call any transfer-related functions as its name implies (although it is used by one). Consider renaming the function to `_checkOnERC721Received`.

  Update: resolved in #1445

token/ERC721/

ERC721Burnable.sol

Notes:

- *Note:* ERC721Burnable.sol#L7 - `burn()` is missing a NatSpec comment.

token/ERC721/

ERC721Enumerable.sol

Notes:

- *Note:* Consider adding a natspec comment for the contract itself.

token/ERC721/

ERC721Metadata.sol

Notes:

- *Note:* `tokenURI` should be `external` rather than `public`.

  Update: resolved in #1444

- *Note:* `_name` and `_symbol` should be `private`.

  Update: resolved in #1426

token/ERC721/

ERC721Mintable.sol

Notes:

- *Note:* ERC721Mintable.sol#L47 - `mintWithTokenURI()` is missing a NatSpec comment.

  Update: resolved in #1365

utils/

LOW

1. Consider changing the name of isContract()

Address.sol#L16 - Because this function returns false when called from a contract's constructor, consider calling this function `isInitializedContract()` for clarity.

General

Notes:

- *Note:* There are a number of times throughout the code base where a variable is assigned a default value. We recommend removing the right-hand operand for each of these instances that will reduce the gas cost.

    - drafts/ERC1046/TokenMetadata.sol#L18: `_tokenURI` (8,170 gas)
    - payment/SplitPayment.sol#L14-15: `_totalShares` and `_totalReleased` (10,560 gas)
    - token/ERC20/ERC20Mintable.sol#L14: _mintingFinished (6,911 gas)

  Update: resolved in #1432 and #1451

- *Note:* We recommend mirroring the testing directory with the contracts directory. This is already followed for the most part, but there are some notable exceptions after the reorganization of the contracts directory. The following are some adjustments that should be made to achieve this:
    - Move `TokenVesting.test.js` into `test/drafts/`
    - Move `Math.test.js` into `test/math/`
    - Tests are in the wrong folder for `ECDSA`. They're still in the `library/` folder.

  Update: resolved in #1428

- *Style:* It would be good to consistently use decimal or hex numbers in assembly. Switching for different contracts using assembly will confuse users. See `ECDSA.sol` vs. `ERC165Checker.sol`.

  Update: resolved in #1429

- *Note:* The escrow contracts can be separated out into their own folder to help them stand out as first-class contracts in the library. Also, consider adding the following comments to the escrow contracts to clarify their usage. See this Gist for specific recomendations: https://gist.github.com/cwhinfrey/8d995081483906796d634d0373a16c15.

Update: resolved in #1430

No Issues

The following contracts were reviewed but no issues were found:

- crowdsale/validation/IndividuallyCappedCrowdsale.sol
- crowdsale/validation/CappedCrowdsale.sol
- crowdsale/emission/MintedCrowdsale.sol
- cryptography/
- drafts/ERC20Migrator.sol
- drafts/ERC1046/TokenMetadata.sol
- token/ERC20Detailed.sol
- token/ERC20/ERC20Mintable.sol

- token/ERC20Pausable.sol
- token/IERC20.sol
- token/ERC20/TokenTimelock.sol
- token/ERC721/ERC721Full.sol
- token/ERC721/ERC721Holder.sol
- token/ERC721/ERC721Pausable.sol
- token/ERC721/IERC721.sol
- token/ERC721/IERC721Enumberable.sol
- token/ERC721/IERC721Full.sol
- token/ERC721/IERC721Metadata.sol
- token/ERC721/IERC721Receiver.sol

Conclusion

One issue of critical severity was found and relayed to the Zeppelin team immediately. A solution was discussed and implemented as quickly as possible. Only two high severity issues were found and explained, along with recommendations on how to fix them. Most of the issues above are there in order to mitigate the likelihood of an attack by a malicious actor so that developers can focus on writing application-specific code and not have to worry about the implementation of these contracts.

It has been a wonderful experience working with the Zeppelin team and we look forward to seeing the many scenarios in which these contracts are used!

Note that the above audit reflects the Level K analysis of the OpenZeppelin contracts based on currently known security patterns in Solidity and the EVM. We have not reviewed any other Zeppelin or OpenZeppelin products. The above is not investment advice and we do not endorse any token sale related to or created by this code. We do not guarantee that this code is unexploitable and assume no liability for any funds lost in these contracts.