OpenZeppelin

PROPS Token Contracts Audit

OPENZEPPELIN SECURITY | SEPTEMBER 2, 2019

Security Audits

The YouNow team asked us to review and audit their PROPS Token contracts. We looked at the code and now publish our results.

The audited commit is `eaf0c6fddd320a258a9d7531d72d07a830fae58a` and the files included in scope were TokenVesting.sol, ERC865.sol, ERC865Token.sol, PropsSidechainCompatible.sol, PropsTimeBasedTransfers.sol and PropsToken.sol.

While the rest of the project is still a work in progress and was therefore *not* included in this audit's scope, additional general notes regarding the project's setup, documentation and software testability were still included to help improve its overall quality.

Here are our assessment and recommendations, in order of importance.

Update: *the YouNow team made some fixes based on our recommendations. We address below the fixes introduced up to commit* `7ead8c55da3770c4d39db8c89a8aa20ec62e97b1`.

Critical severity

None.

High severity

Incorrect 4-bytes function identifier in ERC865Token contract

`48664c16`) is obtained from hashing the signature `transferPreSignedHashing(address,address,address,uint256,uint256,uint256)`, whereas the actual signature of the function being used is `transferPreSignedHashing(address,address,uint256,uint256,uint256)`. This mismatch may potentially cause problems in higher application layers or in the interaction with other third-party systems that use the correct signature.

Hence, the 4-bytes identifier must be replaced by the correct one, which is `15420b71`. Consider updating this value in line 242 of the `ERC865Token`, as well as modifying the comment in the previous line so as to match both the correct identifier and function signature.

Update: *Fixed. The 4-bytes identifier now matches the signature of the function. Note that the function was renamed, so the signature is now* `0x0d98dcb1`.

Medium severity

Implementation of the ERC865 interface does not fully match EIP 865 specification

The “Specification” section of the EIP 865 details the process for how users are expected to generate and sign token transactions off-chain and how trusted parties (called *delegates*) are expected to handle and submit them, in order to be paid fixed fees for their services. However, comparing what is specified in the EIP and the actual implementation in the `ERC865Token` contract, some inconsistencies were detected.

According to the spec *“With their private key, the user generates {V,R,S} for the sha3 of the payload P {N,A,B,D,X,Y,T}”*, where N is the nonce, A the sender of the payment, B the recipient of the payment, D is the delegate who pays for the gas, X is the amount of tokens sent from A to B, Y is the fee A pays D, and T is the token. The `ERC865Token` contract is then expected to reconstruct the hash of the payload P with all data provided by the delegate. Nonetheless, the current implementation in this contract rebuilds payloads placing data in a different order than the spec (the nonce, for instance, is placed last), which might potentially result in differences between on-chain and off-chain generated hashes if the off-chain client is not strictly aligned with the implementation in the `ERC865Token` contract.

again, may result in differences between on-chain and off-chain generated hashes.

Consider applying the necessary changes to fully match the EIP 865 specification, or provide enough end-user documentation that details how off-chain clients are expected to interact with the `ERC865Token` contract. Should the development team consider their current implementation of the EIP to be more reliable, consider proposing changes to the public EIP while it is still a work in progress.

Potential signature malleability in ERC865Token

As highlighted in the EIP 865's ongoing discussion, its current full implementation is affected by a signature malleability issue, steaming from the fact that in the current EIP, the `recover` function:

- Allows both values 0/1 and 27/28 for `v`
- Allows both lower and upper `s` values

Signature malleability poses a security risk in systems that use these kind of signatures as unique identifiers; as is the case of the `ERC865Token` contract, where a `signatures` mapping takes care of tracking signed operations using a signature as the identifier for each operation.

The `recover` function implementation found in the `ERC865Token` contract appears to be copied from OpenZeppelin-eth's ECDSA library. Although a set of lines were commented out in an attempt to patch the signature malleability issue reported in the library, there is still debate around how to best approach the fix and whether only restricting the `v` value to 27/28 is sufficient to prevent signature malleability.

Taking into account these issues, consider tracking unique operations using a hash of the payload (including a nonce to prevent equal operations from producing the same hash) instead of using potentially malleable signatures. This way, the custom implementation of the `recover` function in the `ERC865Token` contract could be removed from the codebase and instead directly imported from OpenZeppelin's ECDSA library, since the potential signature malleability issue in its `recover` function would no longer affect the `ERC865Token` implementation. In addition, the project will be able to benefit from thoroughly tested bug-fixes in future releases of the library.

render the audited `ERC865` and `ERC865Token` contracts outdated and non-compliant.

For more details on what signature malleability is and how it can be exploited by attackers, please refer to:

- How Not to Use ECDSA, by Yondon Fu.
- Bitcoin Transaction Malleability, by Evan Klitzke.

Update: *Fixed. The contract is now using the `recover` function from OpenZeppelin. The hash of the payload and the signer address is now used as the identifier of the operation, for example, see L215. Note that the EIP is still in draft.*

Lack of event emission in `transferFromPreSigned` function of ERC865Token contract

The `ERC865Token` contract's purpose is to extend ERC20 `transfer` and `transferFrom` operations with pre-signed transfers of tokens. This is achieved via the `transferPreSigned` and `transferFromPreSigned` public functions.

Additionally, a new event called `TransferPreSigned` takes care of logging pre-signed transfers . Even though this event is properly emitted after a successful `transferPreSigned` operation, `TransferPreSigned` is not emitted before the execution of the `transferFromPreSigned` function is completed, which may be difficult tracking this operation via event logs.

To favor consistency and operations traceability, and taking into account that the counterpart ERC20 `transferFrom` does emit a `Transfer` event, consider emitting a `TransferPreSigned` event after a `transferFromPreSigned` operation is successfully registered.

Update: *Fixed. The `transferFromPreSigned` function now emits the `TransferPreSigned` event.*

Missing test coverage report

manual test suite has to be executed to make sure that nothing is broken or misbehaving.

Consider adding the test coverage report, and making it reach at least 95% of the source code.

Update: *Fixed. A coverage report has been added. Note that this report has to be generated every time the code is changed, so consider removing it from the repository and using instead a reporting service like Codecov or Coveralls. Also note that although reported line coverage is 100%, branch coverage is 53%.*

Low severity

Transaction token fees may be locked forever in ERC865Token contract

The `ERC865Token` contract allows the owner of tokens to execute token transfers and approvals without the need to own ETH in the first place. The owner, instead, pre-signs transactions and sends them off-chain to a trusted party that in turn submits the actual transactions to the token contract. As a reward, this trusted party gets paid a pre-arranged fixed amount of tokens.

In the current implementation, fees are paid in tokens to the `msg.sender` of the transaction, which may in some cases be a contract and not an externally-owned account. In a scenario where the `msg.sender` is indeed a smart contract without the necessary logic to operate with tokens, the fees may forever be locked in the token contract, resulting in financial losses for the party submitting the actual transactions to the network.

If there are no restrictions on whether contracts can relay transactions to the `ERC865Token` contract, it is recommended to explicitly warn users about the potential locking of tokens, providing detailed end-user documentation on how to best tackle this issue (e.g. providing a sample contract containing the necessary logic to both relay transactions and operate with received fees).

Update: *Partially fixed. A notice comment has been added to the functions to warn users about calling them from smart contracts. For example, see L182.*

Unexpected behavior in decreaseApprovalPreSigned function of ERC865Token contract

functions behaved exactly the same when decreasing an allowance. Yet, `decreaseApprovalPreSigned` does not revert when the value to be subtracted is greater than the current allowance, as `decreaseAllowance` does. Instead, it sets the allowance to zero.

While this issue does not pose a security risk, since the allowance is indeed decreased, the differences in behavior between functions is not documented at all and might be entirely unexpected for developers and users, potentially causing errors in off-chain clients interacting with it. Thus, consider not implementing custom behavior in interfaces that have become de-facto standard. If strictly required, documenting differences in behavior is highly recommended.

Update: *Fixed. The `decreaseApprovalPreSigned` function now reverts when the value subtracted is greater than the current allowance.*

Code repetition in PropsSidechainCompatible contract

Following code reusability best practices, the `settle` function of the `PropsSidechainCompatible` contract may benefit from internally calling its public `transfer` function (instead of calling `super.transfer`) and on success only emit the `Settlement` event, as the `TransferDetails` event will already be emitted by the `transfer` function.

Furthermore, consider explicitly documenting the rationale behind including two extremely similar functions such as `transfer` and `settle`, where the only difference between the two is the emission of a `Settlement` event in the latter. Off-chain clients accidentally calling `transfer` instead of `settle` to transfer tokens may expect a `Settlement` event that is never going to be emitted. Additionally, this behavior is inconsistent, considering that the `settle` function only works with `transfer` and not with `transferFrom`.

Update: *The `PropsSidechainCompatible` contract has been removed.*

Missing error messages in require statements

There are several `require` statements in the `ERC865Token` contract without error messages. Consider including specific and informative error messages in all require statements.

Missing docstrings for public functions

There are several public functions without docstrings in both `PropsSidechainCompatible` and `PropsTimeBasedTransfers` contracts. Consider adding Natspec docstrings to everything that is part of the contracts' public API.

Update: *Fixed. The `PropsSidechainCompatible` contract has been removed and now all the functions of the PropsTimeBasedTransfers contract have docstrings.*

Outdated OpenZeppelin vesting contract in use

The vesting contract `TokenVesting` is outdated and has two known low-severity issues, thoroughly described in the last LevelK's OpenZeppelin audit report (refer to page 15).

In particular, these issues were fixed in newer versions of the `TokenVesting` contract and now include additional safety checks during construction and restricted private functions that are not yet intended to be part of the contract's public API (such as `releasableAmount` and `vestedAmount` functions).

As these changes are already ported to newer versions of the openzeppelin-eth package, consider upgrading the `TokenVesting` contract to include these fixes, or at least properly documenting the rationale behind not including them.

Update: *The `TokenVesting` contract has been removed.*

OpenZeppelin MIT license violation

The contract `TokenVesting` in the `TokenVesting.sol` file was copied from the openzeppelin-eth public repository. However, it does not include the MIT license required by the project. As stated by the this license:

"The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software."

When not importing the contracts directly from installed packages and instead copying them into your project (which is not the recommended way of use), consider including all necessary licenses

Update: *The `TokenVesting` contract has been removed.*

Untested Solidity optimizations may cause unpredictable behavior

In the `truffle.js` configuration file, Solidity optimizations are enabled. Solidity has some default optimizations always executed, and some others that are optional. Enabling the optional ones increases the risk of unexpected behavior, since they are not as battle-tested as the default optimizations. Consider adding full test coverage without optimizations before enabling them, so as to verify that the introduced optimizations preserve expected behavior.

Update: *Partially fixed. Reported line coverage is 100%, but branch coverage is 53%.*

Commented out code in several functions

Several functions in the `ERC865Token` contract include commented out lines of code without giving developers enough context on why those lines have been discarded, thus providing them with little to no value at all. The same issue was identified in the contract's import statements.

As the purpose of these lines is unclear and may confuse future developers and external contributors, consider removing them from the codebase. If they are to provide alternate implementation options, consider extracting them to a separate document where a deeper and more thorough explanation could be included.

Update: *Fixed. Commented out lines have been removed from the `ERC865Token` contract.*

Erroneous docstrings in codebase

A first instance of this issue can be found at the inline documentation provided before the `PropsTimeBasedTransfers` contract declaration, where the current docstrings correspond to the `PropsSidechainCompatible` contract. Similarly, contract `ERC865` docstrings correspond to the `ERC865Token` contract.

Moreover, there are references in the `PropsToken` contract docstrings to an integration with the TPL protocol that is not currently implemented in the token. Additionally, in this same contract, the docstrings state that the `PropsToken` is pausable, but no such behavior was found in the code's logic.

Furthermore, an inline comment in the `ERC865Token` contract mistakenly describes the `signatures` mapping as a data structure to hold nonces, while it is actually used to store signatures.

As inaccurate docstrings may confuse users, developers and auditors alike, consider fixing all instances of this issue.

Update: *Partially fixed. The docstring of the* `IERC865` *contract is still wrong.*

Erroneous and complex project set up

The setup instructions found in the README file require developers to do a minor change in the openzeppelin-eth ERC20 contract at `node_modules/openzeppelin-eth/contracts/token/ERC20.sol`. For a more developer-friendly experience while setting up the project's dependencies, consider directly including the modified contract in the `contracts` folder (changing the `import` paths were needed), instead of instructing to modify a package in the `node_modules` folder, which requires developers to apply the modifications every time a fresh `npm install` command is run. Also, consider fixing the path specified in the README to `node_modules/openzeppelin-eth/contracts/token/ERC20/ERC20.sol`.

Additionally, one step in the testing instructions in the the README file mentions a `logic-contract-deployer-address`, from which contracts are expected to be deployed. However, it is never explained to contributors what that address is nor how to obtain it.

Finally, further hindering the developers and potential contributors' experience, the openzeppelin-eth version in the `package.json` file is not fixed to `2.0.2`, but to `^2.0.2`, which currently leads to version `2.1.3` of the openzeppelin-eth package being installed after `npm install` is run, a version of the package meant to be used with Solidity compiler's version 0.5, while the rest of the project's contracts are to be compiled with a version of Solidity up to 0.4.25. Until the rest of the contracts in the PROPS Token project is upgraded to Solidity 0.5, consider pinning this dependency to `2.0.2`, or `~2.0.2` to benefit from future patch versions.

Erroneous test suite

The testing suite does not run after following the instructions in the README file of the project.

As the test suite was left outside the audit's scope, please consider thoroughly reviewing it to make sure all tests run successfully after following the instructions in the README file.

Update: *Fixed. The test suite now can be run after following the instructions in the README. However, note that the tests Name should be correct and Symbol should be correct are failing.*

Erroneous Truffle configuration

An error in the `truffle.js` configuration file prevents Truffle 5 from running with the specified solc version (0.4.24). Current configuration is:

```
solc: {
  version: '0.4.24',
  optimizer: {
    enabled: true,
    runs: 500,
  },
},
```

According to Truffle docs, it should be:

```
compilers: {
  solc: {
    version: '0.4.24',
    optimizer: {
        enabled: true,
        runs: 500,
    },
  },
 }
```

An outdated solc version, 0.4.24, is currently in use. Consider upgrading to the latest Solidity v0.5 or at least to the latest version of the 0.4 branch, currently 0.4.25.

Updated: *Fixed. The truffle configuration now specifies solidity v0.4.25.*

Notes & Additional Information

- In the `PropsTimeBasedTransfers` contract, consider refactoring the functionality of the `canTransfer` function to a `canTransfer` modifier that includes a require statement, so as to avoid repeating code in both `transfer` and `transferFrom` functions.
  Update: *`canTransfer` is now a modifier.*
- It is recommended to always first inherit from the `zos-lib/contracts/Initializable.sol` contract when dealing with upgradeable contracts, so as to prevent any unexpected C3-linearization issues down the inheritance chain, as explained in Zeppelin's forum. This is a common practice applied throughout the openzeppelin-eth project (the official OpenZeppelin fork adapted to upgradeable contracts), as well as in the code examples at ZeppelinOS docs.
  The `PropsToken` correctly follows this practice, so consider explicitly adding the `Initializable` contract first in the inheritance chain in both `PropsTimeBasedTransfers` and `PropsSidechainCompatible` contracts as well.
  Update: *The `PropsTimeBasedTransfers` contract now inherits first from Initializable. The `PropsSidechainCompatible` contract has been removed.*
- In the `PropsToken` contract's `initialize` function, consider clearly stating the time units of the `_transfersStartTime` argument.
  Update: *The comment of the `initialize` function now mentions that `_transfersStartTime` is a Unix timestamp.*
- The project's documentation refer to the token as "PROPS token", while the name used in the contract's code is DEV_Token. Consider modifying the hardcoded name before deploying the token to a public network so as to avoid confusion in users.
  Update: *The name of the token is now "Props Token" and the symbol "PROPS".*
- To favor readability in the `PropsToken` contract, consider refactoring the token's total supply calculation to `6 * 1e8 * (10 ** uint256(decimals))`, adding a short

using block timestamps as opposed to e.g. block numbers), and is therefore sensitive to timestamp manipulation (something miners can do, to a certain degree). Therefore, it is recommended to avoid using a short time duration (less than a minute). Typical vesting schemes, such as those with a cliff period of a year and a duration of four years, are safe to use. While this does not pose a security issue *per se*, it is advisable to properly document this behavior and bear it in mind when defining the token's vesting scheme.

Update: *The `TokenVesting` contract has been removed. It is still used by distribution scripts, which are out of the scope of this audit.*

- To favor explicitness and readability, consider changing all instances of `uint` to `uint256` (see for example `ERC865Token`: L159).

  Update: *Some `u` `int256` in PropsTimeBasedTransfers and PropsToken are still not declared explicitly.*

- The `ERC865` interface may benefit from renaming to `IERC865`, so as to explicitly denote that it is an interface. Moreover, following good practices already applied in community-vetted frameworks, consider moving event definitions such as `TransferPreSigned` and `ApprovalPreSigned` from the implementation `ERC865Token` contract to the interface. These events will then be inherited by all implementations of the `ERC865` interface.

  Update: *The interface is now named IERC865. It now includes the definition of the events.*

- Although not strictly required, it is always advisable to follow interfaces proposed by most popular smart contract frameworks, which have become the de-facto standard supported by developers. As the OpenZeppelin's ERC20 contract from which the `ERC865` contract inherits includes two functions `increaseAllowance` and `decreaseAllowance`, it would be sensible to rename both `increaseApprovalPreSigned` and `decreaseApprovalPreSigned` functions to `increaseAllowancePreSigned` and `decreaseAllowancePreSigned` respectively. Be aware that the suggested naming would not be aligned to the current state of the EIP 865, so make sure these changes are first applied in the EIP before modifying the interface (`ERC865)` and implementation `(ERC865Token)` contracts.

  Update: *The functions are now named `increaseAllowancePreSigned` and `decreaseAllowancePreSigned`. Note that the renames have not been proposed to the EIP yet.*

any transfer operation, following the Check-Effects-Interactions pattern.

Update: state variables are now modified before the transfer operations. For example, see L91.

- To favor explicitness and readability, several functions in the `ERC865Token` contract may benefit from better naming. Our suggestions are:
- `transferPreSignedHashing` to `getTransferPreSignedHash`
- `transferFromPreSignedHashing` to `getTransferFromPreSignedHash`
- `approvePreSignedHashing` to `getApprovePreSignedHash` – `increaseApprovalPreSignedHashing` to `getIncreaseApprovalPreSignedHash` – `decreaseApprovalPreSignedHashing` to `getDecreaseApprovalPreSignedHash` Consider applying these modifications only after they are proposed and accepted in the EIP 865, so as to avoid rendering the current implemented interfaces non-compliant.

  ***Update:** The functions have been renamed. Note that the renames have not been proposed to the EIP yet.*
- Function parameters of `transferFrom` function in `PropsTimeBasedTransfers` contract and function `transferFrom` in `PropsSidechainCompatible` should be indented one extra level.

  ***Update:** The `PropsSidechainCompatible` contract has been removed. The `transferFrom` function in `PropsTimeBasedTransfers` is now properly indented. However, note that now the require statements are not properly indented.*
- Consider always following Solidity's style guide, where it is suggested that functions' visibility modifiers should come before any custom modifiers. A recommended practice to help enforce coding style is the integration of a linter tool, such as Solhint, into the development process.
- In "truffle.js file, there are several different networks with the exact same parameters, such as: `test0`, `test1`, `test2`, `testValidator`, `test`, `local`. Additionally, the naming of the network called `rinkebydev` is misleading and may confuse developers, considering that it actually points to a `localhost` network.

  ***Update:** The mentioned `test` networks have been removed, and `rinkebydev` has been renamed.*

these paths and instead use environment variables.

Update: *The paths have been removed.*

- There is an unnecessary ZeppelinOS development configuration file called `zos.dev-5777.json` in the root directory of the repository that should be removed. Refer to ZeppelinOS docs on configuration files to learn more about which files are intended to be tracked in version control.

  Update: *The ZeppelinOS development configuration file has been removed.*

- `web3` is currently included as a dependency in the `package.json` file as `"web3": "^1.0.0-beta.35"`. Developers should be aware of issue #2266 introduced in web3 v1.0.0-beta.38, where custom providers are no longer accepted, which could introduce unexpected bugs in the project. Pinning the `web3` dependency to a fixed version (e.g. `"web3": "1.0.0-beta.37"`) is the recommended course of action, which other projects have already taken, until this issue is solved.

  Update: *The web3 dependency is now pinned to 1.0.0-beta.35.*

- The `package.json` file includes a `fs` dependency which should be removed, considering that the `fs` package in NPM is only a security holding package.

  Update: *The fs dependency has been removed.*

- The dependency `"zeppelin-solidity": "^1.6.0"` that is included in the `package.json` file is deprecated and no longer maintained. To use OpenZeppelin up-to-date audited contracts, install the openzeppelin-solidity package.

  Update: *The `zeppelin-solidity` dependency has been removed.*

Conclusion

No critical and one high severity issue were found. Some changes were proposed to follow best practices and reduce the potential attack surface.

> *Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the PROPS Token's contracts. The above should not be construed as investment advice. For general information about smart contract security, check out our thoughts here.*

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits

OpenZeppelin