UpOnly (UPO) Token

Smart Contract Audit

UpOnly (UPO) Token Audit Summary

Project name : UpOnly (UPO) Token Contract

Project address: None

Code URL : https://polygonscan.com/address/0x9dBfc1cbf7a1E711503a29B4b5F9130ebeCcaC96#code

Commit : None

Project target : UpOnly (UPO) Token Contract Audit

Blockchain : Polygon

Test result : PASSED

Audit Info

Audit NO : 0X202111300006

Audit Team : Armors Labs

Audit Proofreading: https://armors.io/#project-cases

UpOnly (UPO) Token Audit

The UpOnly (UPO) Token team asked us to review and audit their UpOnly (UPO) Token contract. We looked at the code and now publish our results.

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

Document information

Name	Auditor	Version	Date
UpOnly (UPO) Token Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2021-11-30

Audit results

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the UpOnly (UPO) Token contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 3 months from the date of output.

Disclaimer

Armors Labs Reports is not and should not be regarded as an "approval" or "disapproval" of any particular project or team. These reports are not and should not be regarded as indicators of the economy or value of any "product" or "asset" created by any team. Armors do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'…)

Armors Labs Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Armors does not guarantee the safety or functionality of the technology agreed to be analyzed.

Armors Labs postulates that the information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered, deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused. Armors Labs Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

Audited target file

file	md5
./PowerfulERC20.sol	bd1c0ab075560cb21f5be9634a5f9d98

Vulnerability analysis

Vulnerability distribution

vulnerability level	number
Critical severity	0
High severity	0
Medium severity	0
Low severity	0

Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Arithmetic Over/Under Flows	safe
Unexpected Blockchain Currency	safe
Delegatecall	safe
Default Visibilities	safe
Entropy Illusion	safe
External Contract Referencing	safe
Short Address/Parameter Attack	safe
Unchecked CALL Return Values	safe
Race Conditions / Front Running	safe
Denial Of Service (DOS)	safe

Vulnerability	status
Block Timestamp Manipulation	safe
Constructors with Care	safe
Unintialised Storage Pointers	safe
Floating Points and Numerical Precision	safe
tx.origin Authentication	safe
Permission restrictions	safe

Contract file

```
/**
 *Submitted for verification at Etherscan.io on 2021-09-22
*/

// SPDX-License-Identifier: MIT

// File: @openzeppelin/contracts/token/ERC20/IERC20.sol


pragma solidity ^0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the opera
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
```

```
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the
     *
     * Returns a boolean value indicating whether the opera
     *
     * IMPORTANT: Beware that changing an allowance with this method brings
     * that someone may use both the old and the new allow
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the opera
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

// File: @openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol
```

```
pragma solidity ^0.8.0;


/**
 * @dev Interface for the optional metadata functions from the
 *
 * _Available since v4.1._
 */
interface IERC20Metadata is IERC20 {
    /**
     * @dev Returns the name of the token.
     */
    function name() external view returns (string memory);

    /**
     * @dev Returns the symbol of the token.
     */
    function symbol() external view returns (string memory);

    /**
     * @dev Returns the decimals places of the token.
     */
    function decimals() external view returns (uint8);
}

// File: @openzeppelin/contracts/utils/Context.sol


pragma solidity ^0.8.0;

/**
 * @dev Provides information about the current execution context, including
 * sender of the transaction and its data. While these are g
 * via msg.sender and msg.data, they should not be acces
 * manner, since when dealing with meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an
 * is concerned).
 *
 * This contract is only required for intermediate, library- like contracts.
 */
abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes calldata) {
        return msg.data;
    }
}

// File: @openzeppelin/contracts/token/ERC20/ERC20.sol


pragma solidity ^0.8.0;
```

```
/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are cr
 * that a supply mechanism has to be added in a derived co
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin Contracts guidelines: functions revert
 * instead returning `false` on failure. This behavior is nonetheless
 * conventional and does not conflict with the expectations of ERC20
 * applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts j
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20, IERC20Metadata {
    mapping(address => uint256) private _balances;

    mapping(address => mapping(address => uint256)) private _allowances;

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;

    /**
     * @dev Sets the values for {name} and {symbol}.
     *
     * The default value of {decimals} is 18. To select a different value for
     * {decimals} you should overload it.
     *
     * All two of these values are immutable: they can only b
     * construction.
     */
    constructor(string memory name_, string memory symbol_) {
        _name = name_;
        _symbol = symbol_;
    }

    /**
     * @dev Returns the name of the token.
```

```
 */
  function name() public view virtual override returns (string memory) {
      return _name;
  }

              /**
 * @dev Returns the symbol of the token, usually
 * name.
 */
  function symbol() public view virtual override returns (string memory) {
      return _symbol;
  }

              /**
 * @dev Returns the number of decimals used to get its user representation.
 * For example, if `decimals` equals `2`, a balance of `505` tokens shou
 * be displayed to a user as `5.05` (`505 / 10 ** 2`).
 *
 * Tokens usually opt for a value of 18, imitating the relat
 * Ether and Wei. This is the value {ERC20} uses, unless this function is
 * overridden;
 *
 * NOTE: This information is only used for _display_ purposes: it in
 * no way affects any of the arithmetic of the contract, inc
 * {IERC20-balanceOf} and {IERC20-transfer}.
 */
  function decimals() public view virtual override returns (uint8) {
      return 18;
  }

              /**
 * @dev See {IERC20-totalSupply}.
 */
  function totalSupply() public view virtual override returns (uint256) {
      return _totalSupply;
  }

              /**
 * @dev See {IERC20-balanceOf}.
 */
  function balanceOf(address account) public view virtual override returns (uint256) {
      return _balances[account];
  }

              /**
 * @dev See {IERC20-transfer}.
 *
 * Requirements:
 *
 * - `recipient` cannot be the zero address.
 * - the caller must have a balance of at least `amount`.
 */
  function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
      _transfer(_msgSender(), recipient, amount);
      return true;
  }
```

```
    /**
 * @dev See {IERC20-allowance}.
 */
    function allowance(address owner, address spender) public view virtual override returns (uint256)
        return _allowances[owner][spender];
    }

    /**
 * @dev See {IERC20-approve}.
 *
 * Requirements:
 *
 * - `spender` cannot be the zero address.
 */
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    /**
 * @dev See {IERC20-transferFrom}.
 *
 * Emits an {Approval} event indicating the updated allow
 * required by the EIP. See the note at the
 *
 * Requirements:
 *
 * - `sender` and `recipient` cannot be the zero address.
 * - `sender` must have a balance of at least `amount`.
 * - the caller must have allowance for ``sender``'s tokens of at least
 * `amount`.
 */
    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) public virtual override returns (bool) {
        _transfer(sender, recipient, amount);

        uint256 currentAllowance = _allowances[sender][_msgSender()];
        require(currentAllowance >= amount, "ERC20: transfer amount exceeds allowance");
        unchecked {
            _approve(sender, _msgSender(), currentAllowance - amount);
        }

        return true;
    }

    /**
 * @dev Atomically increases the allowance granted to `spender` by t
 *
 * This is an alternative to {approve} that can be used as a
 * problems described in {IERC20-approve}.
 *
 * Emits an {Approval} event indicating the updated allow
```

```
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender] + addedValue);
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by
     *
     * This is an alternative to {approve} that can be used as a
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allow
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool
        uint256 currentAllowance = _allowances[_msgSender()][spender];
        require(currentAllowance >= subtractedValue, "ERC20: decreased allowance below zero");
        unchecked {
            _approve(_msgSender(), spender, currentAllowance - subtractedValue);
        }

        return true;
    }

    /**
     * @dev Moves `amount` of tokens from `sender` to `recipient`.
     *
     * This internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    function _transfer(
        address sender,
        address recipient,
        uint256 amount
    ) internal virtual {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");
```

```
        _beforeTokenTransfer(sender, recipient, amount);

        uint256 senderBalance = _balances[sender];
        require(senderBalance >= amount, "ERC20: transfer amount exceeds balance");
        unchecked {
            _balances[sender] = senderBalance - amount;
        }
        _balances[recipient] += amount;

        emit Transfer(sender, recipient, amount);

        _afterTokenTransfer(sender, recipient, amount);
    }

    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
     * the total supply.
     *
     * Emits a {Transfer} event with `from` set to the zero ado
     *
     * Requirements:
     *
     * - `account` cannot be the zero address.
     */
    function _mint(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address");

        _beforeTokenTransfer(address(0), account, amount);

        _totalSupply += amount;
        _balances[account] += amount;
        emit Transfer(address(0), account, amount);

        _afterTokenTransfer(address(0), account, amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero addres
     *
     * Requirements:
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");

        _beforeTokenTransfer(account, address(0), amount);

        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
        unchecked {
            _balances[account] = accountBalance - amount;
        }
        _totalSupply -= amount;

        emit Transfer(account, address(0), amount);
```

```
        _afterTokenTransfer(account, address(0), amount);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the
     *
     * This internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    function _approve(
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }

    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     *   will be transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-ho
     */
    function _beforeTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {}

    /**
     * @dev Hook that is called after any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
```

```
     * has been transferred to `to`.
     * - when `from` is zero, `amount` tokens have been minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens have been burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-ho
     */
    function _afterTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual {}
}

// File: @openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol


pragma solidity ^0.8.0;



/**
 * @dev Extension of {ERC20} that allows token holders to destroy both their own
 * tokens and those that they have an allowance for, in
 * recognized off-chain (via event analysis).
 */
abstract contract ERC20Burnable is Context, ERC20 {
    /**
     * @dev Destroys `amount` tokens from the caller.
     *
     * See {ERC20-_burn}.
     */
    function burn(uint256 amount) public virtual {
        _burn(_msgSender(), amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, deducting from the caller's
     * allowance.
     *
     * See {ERC20-_burn} and {ERC20-allowance}.
     *
     * Requirements:
     *
     * - the caller must have allowance for ``accounts``'s tokens of at least
     * `amount`.
     */
    function burnFrom(address account, uint256 amount) public virtual {
        uint256 currentAllowance = allowance(account, _msgSender());
        require(currentAllowance >= amount, "ERC20: burn amount exceeds allowance");
        unchecked {
            _approve(account, _msgSender(), currentAllowance - amount);
        }
        _burn(account, amount);
    }
}
```

```
// File: @openzeppelin/contracts/token/ERC20/extensions/ERC20Capped.sol


pragma solidity ^0.8.0;


/**
 * @dev Extension of {ERC20} that adds a cap to the supp
 */
abstract contract ERC20Capped is ERC20 {
    uint256 private immutable _cap;

    /**
     * @dev Sets the value of the `cap`. This value is immut
     * set once during construction.
     */
    constructor(uint256 cap_) {
        require(cap_ > 0, "ERC20Capped: cap is 0");
        _cap = cap_;
    }

    /**
     * @dev Returns the cap on the token's total supply.
     */
    function cap() public view virtual returns (uint256) {
        return _cap;
    }

    /**
     * @dev See {ERC20-_mint}.
     */
    function _mint(address account, uint256 amount) internal virtual override {
        require(ERC20.totalSupply() + amount <= cap(), "ERC20Capped: cap exceeded");
        super._mint(account, amount);
    }
}

// File: @openzeppelin/contracts/utils/Address.sol


pragma solidity ^0.8.0;

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a
     *
     * Among others, `isContract` will return false for the fol
```

```
 * types of addresses:
 *
 * -             an              externally-owned account
 * -             a              contract in construction
 * -             an              address where              a              contract              will
 * -             an              address where              a              contract lived,              but
 * ====
 */
  function isContract(address account) internal view returns (bool) {
      // This method relies on extcodesize, which returns 0 for contracts in
      // construction, since the code is only stored at the end of the
      // constructor execution.

      uint256 size;
      assembly {
          size := extcodesize(account)
      }
      return size > 0;
  }

              /**
 * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
 * `recipient`, forwarding all available gas and reverting on errors.
 *
 * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases              the              gas cost
 * of certain opcodes, possibly making contracts go over              the              2300 gas limit
 * imposed by `transfer`, making them unable to receive funds via
 * `transfer`. {sendValue} removes this limitation.
 *
 * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn              more
 *
 * IMPORTANT: because control is transferred to `recipient`, care must be
 * taken to not create reentrancy vulnerabilities. Consider using
 * {ReentrancyGuard} or              the
 * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-              the              -che
 */
  function sendValue(address payable recipient, uint256 amount) internal {
      require(address(this).balance >= amount, "Address: insufficient balance");

      (bool success, ) = recipient.call{value: amount}("");
      require(success, "Address: unable to send value, recipient may have reverted");
  }

              /**
 * @dev Performs              a              Solidity function call using              a              low lev
 * plain `call` is              an              unsafe replacement for              a              function call
 * function instead.
 *
 * If `target` reverts with              a              revert reason, it is bubbled up by this
 * function (              like              regular Solidity function calls).
 *
 * Returns              the              raw returned data. To convert to              the              expe
 * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding
 *
 * Requirements:
```

```
 *
 * - `target` must be a contract.
 * - calling `target` with `data` must not revert.
 *
 * _Available since v3.1._
 */
  function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
  }

                /**
 * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
 * `errorMessage` as a fallback revert reason when `target` reverts.
 *
 * _Available since v3.1._
 */
  function functionCall(
      address target,
      bytes memory data,
      string memory errorMessage
  ) internal returns (bytes memory) {
      return functionCallWithValue(target, data, 0, errorMessage);
  }

                /**
 * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
 * but also transferring `value` wei to `target`.
 *
 * Requirements:
 *
 * - the calling contract must have an ETH balance of at
 * - the called Solidity function must be `payable`.
 *
 * _Available since v3.1._
 */
  function functionCallWithValue(
      address target,
      bytes memory data,
      uint256 value
  ) internal returns (bytes memory) {
      return functionCallWithValue(target, data, value, "Address: low-level call with value failed"
  }

                /**
 * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`],
 * with `errorMessage` as a fallback revert reason when `target` reverts.
 *
 * _Available since v3.1._
 */
  function functionCallWithValue(
      address target,
      bytes memory data,
      uint256 value,
      string memory errorMessage
  ) internal returns (bytes memory) {
      require(address(this).balance >= value, "Address: insufficient balance for call");
      require(isContract(target), "Address: call to non-contract");
```

```
        (bool success, bytes memory returndata) = target.call{value: value}(data);
        return verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data) internal view returns (bytes memor
        return functionStaticCall(target, data, "Address: low-level static call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal view returns (bytes memory) {
        require(isContract(target), "Address: static call to non-contract");

        (bool success, bytes memory returndata) = target.staticcall(data);
        return verifyCallResult(success, returndata, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns (bytes memory)
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(
        address target,
        bytes memory data,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(isContract(target), "Address: delegate call to non-contract");

        (bool success, bytes memory returndata) = target.delegatecall(data);
        return verifyCallResult(success, returndata, errorMessage);
    }
```

```
    /**
     * @dev Tool to verifies that a low level call was successful, and revert if it wasn't, eit
     * revert reason using the provided one.
     *
     * _Available since v4.3._
     */
    function verifyCallResult(
        bool success,
        bytes memory returndata,
        string memory errorMessage
    ) internal pure returns (bytes memory) {
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}

// File: @openzeppelin/contracts/utils/introspection/IERC165.sol


pragma solidity ^0.8.0;

/**
 * @dev Interface of the ERC165 standard, as defined in the
 * https://eips.ethereum.org/EIPS/eip-165[EIP].
 *
 * Implementers can declare support of contract interfaces, which can then be
 * queried by others ({ERC165Checker}).
 *
 * For an implementation, see {ERC165}.
 */
interface IERC165 {
    /**
     * @dev Returns true if this contract implements the interface defined by
     * `interfaceId`. See the corresponding
     * https://eips.ethereum.org/EIPS/eip-165#how-interfaces-are-identified[EIP section]
     * to learn more about how these ids are created.
     *
     * This function call must use less than 30 000 gas.
     */
    function supportsInterface(bytes4 interfaceId) external view returns (bool);
}

// File: @openzeppelin/contracts/utils/introspection/ERC165.sol
```

```
pragma solidity ^0.8.0;


/**
 * @dev Implementation of the {IERC165} interface.
 *
 * Contracts that want to implement ERC165 should inherit from this contract and overri
 * for the additional interface id that will be supported. For
 *
 * ```solidity
 * function supportsInterface(bytes4 interfaceId) public view virtual override returns (bool) {
 *     return interfaceId == type(MyInterface).interfaceId || super.supportsInterface(interfaceId);
 * }
 * ```
 *
 * Alternatively, {ERC165Storage} provides an easier to use but
 */
abstract contract ERC165 is IERC165 {
    /**
     * @dev See {IERC165-supportsInterface}.
     */
    function supportsInterface(bytes4 interfaceId) public view virtual override returns (bool) {
        return interfaceId == type(IERC165).interfaceId;
    }
}

// File: erc-payable-token/contracts/token/ERC1363/IERC1363.sol


pragma solidity ^0.8.0;


/**
 * @title IERC1363 Interface
 * @dev Interface for a Payable Token contract as defined in
 *  https://eips.ethereum.org/EIPS/eip-1363
 */
interface IERC1363 is IERC20, IERC165 {
    /**
     * @notice Transfer tokens from `msg.sender` to another address and then call `onTransferReceived` on receiver
     * @param recipient address The address which you want to transfer to
     * @param amount uint256 The amount of tokens to be transferred
     * @return true unless throwing
     */
    function transferAndCall(address recipient, uint256 amount) external returns (bool);

    /**
     * @notice Transfer tokens from `msg.sender` to another address and then call `onTransferReceived` on receiver
     * @param recipient address The address which you want to transfer to
     * @param amount uint256 The amount of tokens to be transferred
     * @param data bytes Additional data with no specified format, sent in call to `recipient`
```

```
 * @return true unless throwing
 */
  function transferAndCall(
      address recipient,
      uint256 amount,
      bytes calldata data
  ) external returns (bool);

                /**
 * @notice Transfer tokens from one address to another and then call `onTransferReceived` on receiver
 * @param sender address The address which you want to send tokens from
 * @param recipient address The address which you want to transfer to
 * @param amount uint256 The amount of tokens to be transferred
 * @return true unless throwing
 */
  function transferFromAndCall(
      address sender,
      address recipient,
      uint256 amount
  ) external returns (bool);

                /**
 * @notice Transfer tokens from one address to another and then call `onTransferReceived` on receiver
 * @param sender address The address which you want to send tokens from
 * @param recipient address The address which you want to transfer to
 * @param amount uint256 The amount of tokens to be transferred
 * @param data bytes Additional data with no specified format, sent in call to `recipient`
 * @return true unless throwing
 */
  function transferFromAndCall(
      address sender,
      address recipient,
      uint256 amount,
      bytes calldata data
  ) external returns (bool);

                /**
 * @notice Approve the passed address to spend the sp
 * and then call `onApprovalReceived` on spender.
 * Beware that changing an allowance with this method brings the
 * and the new allowance by unfortunate transaction ordering. One possible solution
 * race condition is to first reduce the spender's allowance to 0 and set
 * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
 * @param spender address The address which will spend the
 * @param amount uint256 The amount of tokens to be spent
 */
  function approveAndCall(address spender, uint256 amount) external returns (bool);

                /**
 * @notice Approve the passed address to spend the sp
 * and then call `onApprovalReceived` on spender.
 * Beware that changing an allowance with this method brings the
 * and the new allowance by unfortunate transaction ordering. One possible solution
 * race condition is to first reduce the spender's allowance to 0 and set
 * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
```

```
   * @param spender address The address which will spend the
   * @param amount uint256 The amount of tokens to be spent
   * @param data bytes Additional data with no specified format, sent in call to `spender`
   */
    function approveAndCall(
        address spender,
        uint256 amount,
        bytes calldata data
    ) external returns (bool);
}

// File: erc-payable-token/contracts/token/ERC1363/IERC1363Receiver.sol



pragma solidity ^0.8.0;

            /**
 * @title IERC1363Receiver Interface
 * @dev Interface for any contract that wants to support transferAndCall or transferFromAndCall
 *  from ERC1363 token contracts as defined in
 *  https://eips.ethereum.org/EIPS/eip-1363
 */
interface IERC1363Receiver {
                /**
   * @notice Handle the receipt of ERC1363 tokens
   * @dev Any ERC1363 smart contract calls this function on the recipient
   * after a `transfer` or a `transferFrom`. This function MAY
   * transfer. Return of other than the magic value MUST result in the
   * transaction being reverted.
   * Note: the token contract address is always the messa
   * @param operator address The address which called `transferAndCall` or `transferFromAndCall` function
   * @param sender address The address which are token transferred from
   * @param amount uint256 The amount of tokens transferred
   * @param data bytes Additional data with no specified format
   * @return `bytes4(keccak256("onTransferReceived(address,address,uint256,bytes)"))` unless throwing
   */
    function onTransferReceived(
        address operator,
        address sender,
        uint256 amount,
        bytes calldata data
    ) external returns (bytes4);
}

// File: erc-payable-token/contracts/token/ERC1363/IERC1363Spender.sol



pragma solidity ^0.8.0;

            /**
 * @title IERC1363Spender Interface
 * @dev Interface for any contract that wants to support approveAndCall
 *  from ERC1363 token contracts as defined in
 *  https://eips.ethereum.org/EIPS/eip-1363
 */
```

```
interface IERC1363Spender {
    /**
     * @notice Handle the approval of ERC1363 tokens
     * @dev Any ERC1363 smart contract calls this function on the recipient
     * after an `approve`. This function MAY throw to revert and reject the
     * approval. Return of other than the magic value MUST result in the
     * transaction being reverted.
     * Note: the token contract address is always the messa
     * @param sender address The address which called `approveAndCall` function
     * @param amount uint256 The amount of tokens to be spent
     * @param data bytes Additional data with no specified format
     * @return `bytes4(keccak256("onApprovalReceived(address,uint256,bytes)"))` unless throwing
     */
    function onApprovalReceived(
        address sender,
        uint256 amount,
        bytes calldata data
    ) external returns (bytes4);
}

// File: erc-payable-token/contracts/token/ERC1363/ERC1363.sol


pragma solidity ^0.8.0;









/**
 * @title ERC1363
 * @dev Implementation of an ERC1363 interface
 */
abstract contract ERC1363 is ERC20, IERC1363, ERC165 {
    using Address for address;

    /**
     * @dev See {IERC165-supportsInterface}.
     */
    function supportsInterface(bytes4 interfaceId) public view virtual override(ERC165, IERC165) retu
        return interfaceId == type(IERC1363).interfaceId || super.supportsInterface(interfaceId);
    }

    /**
     * @dev Transfer tokens to a specified address and then execute a
     * @param recipient The address to transfer to.
     * @param amount The amount to be transferred.
     * @return A boolean that indicates if the operation was successful.
     */
    function transferAndCall(address recipient, uint256 amount) public virtual override returns (bool
        return transferAndCall(recipient, amount, "");
    }

    /**
```

```
 * @dev Transfer tokens to a specified address and then execute a
 * @param recipient The address to transfer to
 * @param amount The amount to be transferred
 * @param data Additional data with no specified format
 * @return A boolean that indicates if the operation was successful.
 */
  function transferAndCall(
      address recipient,
      uint256 amount,
      bytes memory data
  ) public virtual override returns (bool) {
      transfer(recipient, amount);
      require(_checkAndCallTransfer(_msgSender(), recipient, amount, data), "ERC1363: _checkAndCall
      return true;
  }

              /**
 * @dev Transfer tokens from one address to another and then execute a callback on
 * @param sender The address which you want to send tokens from
 * @param recipient The address which you want to transfer to
 * @param amount The amount of tokens to be transferred
 * @return A boolean that indicates if the operation was successful.
 */
  function transferFromAndCall(
      address sender,
      address recipient,
      uint256 amount
  ) public virtual override returns (bool) {
      return transferFromAndCall(sender, recipient, amount, "");
  }

              /**
 * @dev Transfer tokens from one address to another and then execute a callback on
 * @param sender The address which you want to send tokens from
 * @param recipient The address which you want to transfer to
 * @param amount The amount of tokens to be transferred
 * @param data Additional data with no specified format
 * @return A boolean that indicates if the operation was successful.
 */
  function transferFromAndCall(
      address sender,
      address recipient,
      uint256 amount,
      bytes memory data
  ) public virtual override returns (bool) {
      transferFrom(sender, recipient, amount);
      require(_checkAndCallTransfer(sender, recipient, amount, data), "ERC1363: _checkAndCallTransf
      return true;
  }

              /**
 * @dev Approve spender to transfer tokens and then execute a callback on recipien
 * @param spender The address allowed to transfer to
 * @param amount The amount allowed to be transferred
 * @return A boolean that indicates if the operation was successful.
 */
  function approveAndCall(address spender, uint256 amount) public virtual override returns (bool) {
```

```
        return approveAndCall(spender, amount, "");
    }

    /**
     * @dev Approve spender to transfer tokens and then execute a callback on recipien
     * @param spender The address allowed to transfer to.
     * @param amount The amount allowed to be transferred.
     * @param data Additional data with no specified format.
     * @return A boolean that indicates if the operation was successful.
     */
    function approveAndCall(
        address spender,
        uint256 amount,
        bytes memory data
    ) public virtual override returns (bool) {
        approve(spender, amount);
        require(_checkAndCallApprove(spender, amount, data), "ERC1363: _checkAndCallApprove reverts")
        return true;
    }

    /**
     * @dev Internal function to invoke `onTransferReceived` on a target address
     *  The call is not executed if the target address is not a
     * @param sender address Representing the previous owner of the
     * @param recipient address Target address that will receive the
     * @param amount uint256 The amount mount of tokens to be transferred
     * @param data bytes Optional data to send along with the call
     * @return whether the call correctly returned the expec
     */
    function _checkAndCallTransfer(
        address sender,
        address recipient,
        uint256 amount,
        bytes memory data
    ) internal virtual returns (bool) {
        if (!recipient.isContract()) {
            return false;
        }
        bytes4 retval = IERC1363Receiver(recipient).onTransferReceived(_msgSender(), sender, amount,
        return (retval == IERC1363Receiver(recipient).onTransferReceived.selector);
    }

    /**
     * @dev Internal function to invoke `onApprovalReceived` on a target address
     *  The call is not executed if the target address is not a
     * @param spender address The address which will spend the
     * @param amount uint256 The amount of tokens to be spent
     * @param data bytes Optional data to send along with the call
     * @return whether the call correctly returned the expec
     */
    function _checkAndCallApprove(
        address spender,
        uint256 amount,
        bytes memory data
    ) internal virtual returns (bool) {
        if (!spender.isContract()) {
            return false;
        }
```

```
        bytes4 retval = IERC1363Spender(spender).onApprovalReceived(_msgSender(), amount, data);
        return (retval == IERC1363Spender(spender).onApprovalReceived.selector);
    }
}

// File: @openzeppelin/contracts/access/Ownable.sol


pragma solidity ^0.8.0;


/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclus
 * specific functions.
 *
 * By default, the owner account will be the
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
abstract contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /**
     * @dev Initializes the contract setting the deployer as
     */
    constructor() {
        _setOwner(_msgSender());
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view virtual returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(owner() == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Leaves the contract without owner. It will not b
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave
     * thereby removing any functionality that is only available to the owner.
```

```
    */
    function renounceOwnership() public virtual onlyOwner {
        _setOwner(address(0));
    }

    /**
     * @dev Transfers ownership of the contract to a new ac
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        _setOwner(newOwner);
    }

    function _setOwner(address newOwner) private {
        address oldOwner = _owner;
        _owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);
    }
}

// File: eth-token-recover/contracts/TokenRecover.sol


pragma solidity ^0.8.0;


/**
 * @title TokenRecover
 * @dev Allows owner to recover any ERC20 sent into the contract
 */
contract TokenRecover is Ownable {
    /**
     * @dev Remember that only owner can call so be careful when use on contracts ge
     * @param tokenAddress The token contract address
     * @param tokenAmount Number of tokens to be sent
     */
    function recoverERC20(address tokenAddress, uint256 tokenAmount) public virtual onlyOwner {
        IERC20(tokenAddress).transfer(owner(), tokenAmount);
    }
}

// File: contracts/token/ERC20/behaviours/ERC20Decimals.sol


pragma solidity ^0.8.0;


/**
 * @title ERC20Decimals
 * @dev Implementation of the ERC20Decimals. Extension of {ERC20} that adds deci
 */
abstract contract ERC20Decimals is ERC20 {
    uint8 private immutable _decimals;

    /**
     * @dev Sets the value of the `decimals`. This value is i
```

```
     * set once during construction.
     */
    constructor(uint8 decimals_) {
        _decimals = decimals_;
    }

    function decimals() public view virtual override returns (uint8) {
        return _decimals;
    }
}

// File: contracts/token/ERC20/behaviours/ERC20Mintable.sol



pragma solidity ^0.8.0;


/**
 * @title ERC20Mintable
 * @dev Implementation of the ERC20Mintable. Extension of {ERC20} that adds
 */
abstract contract ERC20Mintable is ERC20 {
    // indicates if minting is finished
    bool private _mintingFinished = false;

    /**
     * @dev Emitted during finish minting
     */
    event MintFinished();

    /**
     * @dev Tokens can be minted only before minting finished.
     */
    modifier canMint() {
        require(!_mintingFinished, "ERC20Mintable: minting is finished");
        _;
    }

    /**
     * @return if minting is finished or not.
     */
    function mintingFinished() external view returns (bool) {
        return _mintingFinished;
    }

    /**
     * @dev Function to mint tokens.
     *
     * WARNING: it allows everyone to mint new tokens. Access controls MUST be defined in derived contracts.
     *
     * @param account The address that will receive the m
     * @param amount The amount of tokens to mint
     */
    function mint(address account, uint256 amount) external canMint {
        _mint(account, amount);
    }
```

```
    /**
     * @dev Function to stop minting new tokens.
     *
     * WARNING: it allows everyone to finish minting. Access controls MUST be defined in derived contracts.
     */
    function finishMinting() external canMint {
        _finishMinting();
    }

    /**
     * @dev Function to stop minting new tokens.
     */
    function _finishMinting() internal virtual {
        _mintingFinished = true;

        emit MintFinished();
    }
}

// File: @openzeppelin/contracts/access/IAccessControl.sol


pragma solidity ^0.8.0;

/**
 * @dev External interface of AccessControl declared to support ERC165 detection.
 */
interface IAccessControl {
    /**
     * @dev Emitted when `newAdminRole` is set as ``role``'s admin role, replacing `previousAdminRole`
     *
     * `DEFAULT_ADMIN_ROLE` is the starting admin for all roles, despite
     * {RoleAdminChanged} not being emitted signaling this.
     *
     * _Available since v3.1._
     */
    event RoleAdminChanged(bytes32 indexed role, bytes32 indexed previousAdminRole, bytes32 indexed n

    /**
     * @dev Emitted when `account` is granted `role`.
     *
     * `sender` is the account that originated the contract ca
     * bearer except when using {AccessControl-_setupRole}.
     */
    event RoleGranted(bytes32 indexed role, address indexed account, address indexed sender);

    /**
     * @dev Emitted when `account` is revoked `role`.
     *
     * `sender` is the account that originated the contract ca
     *  - if using `revokeRole`, it is the admin role bearer
     *  - if using `renounceRole`, it is the role bearer (i.e. `account`)
     */
    event RoleRevoked(bytes32 indexed role, address indexed account, address indexed sender);
```

```
    /**
     * @dev Returns `true` if `account` has been granted `role`.
     */
    function hasRole(bytes32 role, address account) external view returns (bool);

    /**
     * @dev Returns the admin role that controls `role`. See {grantRole} and
     * {revokeRole}.
     *
     * To change a role's admin, use {AccessControl-_setRoleAdmin}.
     */
    function getRoleAdmin(bytes32 role) external view returns (bytes32);

    /**
     * @dev Grants `role` to `account`.
     *
     * If `account` had not been already granted `role`, emits a {RoleGranted}
     * event.
     *
     * Requirements:
     *
     * - the caller must have ``role``'s admin role.
     */
    function grantRole(bytes32 role, address account) external;

    /**
     * @dev Revokes `role` from `account`.
     *
     * If `account` had been granted `role`, emits a {RoleRevoked} event.
     *
     * Requirements:
     *
     * - the caller must have ``role``'s admin role.
     */
    function revokeRole(bytes32 role, address account) external;

    /**
     * @dev Revokes `role` from the calling account.
     *
     * Roles are often managed via {grantRole} and {revokeRole}: this function's
     * purpose is to provide a mechanism for accounts to lose their privileges
     * if they are compromised (such
     *
     * If the calling account had been granted `role`, emits a
     * event.
     *
     * Requirements:
     *
     * - the caller must be `account`.
     */
    function renounceRole(bytes32 role, address account) external;
}
```

```
// File: @openzeppelin/contracts/utils/Strings.sol



pragma solidity ^0.8.0;

            /**
 * @dev String operations.
 */
library Strings {
    bytes16 private constant _HEX_SYMBOLS = "0123456789abcdef";

                /**
   * @dev Converts a `uint256` to its ASCII `string` decimal representation.
   */
    function toString(uint256 value) internal pure returns (string memory) {
        // Inspired by OraclizeAPI's implementation - MIT licence
        // https://github.com/oraclize/ethereum-api/blob/b42146b063c7d6ee1358846c198246239e9360e8/ora

        if (value == 0) {
            return "0";
        }
        uint256 temp = value;
        uint256 digits;
        while (temp != 0) {
            digits++;
            temp /= 10;
        }
        bytes memory buffer = new bytes(digits);
        while (value != 0) {
            digits -= 1;
            buffer[digits] = bytes1(uint8(48 + uint256(value % 10)));
            value /= 10;
        }
        return string(buffer);
    }

                /**
   * @dev Converts a `uint256` to its ASCII `string` hexadecimal representation.
   */
    function toHexString(uint256 value) internal pure returns (string memory) {
        if (value == 0) {
            return "0x00";
        }
        uint256 temp = value;
        uint256 length = 0;
        while (temp != 0) {
            length++;
            temp >>= 8;
        }
        return toHexString(value, length);
    }

                /**
   * @dev Converts a `uint256` to its ASCII `string` hexadecimal representation with fix
   */
    function toHexString(uint256 value, uint256 length) internal pure returns (string memory) {
        bytes memory buffer = new bytes(2 * length + 2);
        buffer[0] = "0";
        buffer[1] = "x";
        for (uint256 i = 2 * length + 1; i > 1; --i) {
            buffer[i] = _HEX_SYMBOLS[value & 0xf];
```

```
            value >>= 4;
        }
        require(value == 0, "Strings: hex length insufficient");
        return string(buffer);
    }
}

// File: @openzeppelin/contracts/access/AccessControl.sol


pragma solidity ^0.8.0;




/**
 * @dev Contract module that allows children to implement role-based access
 * control mechanisms. This is a lightweight version that doesn't
 * members except through off-chain means by accessing the contract event logs. Som
 * applications may benefit from on-chain enumerability, for those cases see
 * {AccessControlEnumerable}.
 *
 * Roles are referred to by their `bytes32` identifier. These should
 * in the external API and be unique. The best way to achieve this is by
 * using `public constant` hash digests:
 *
 * ```
 * bytes32 public constant MY_ROLE = keccak256("MY_ROLE");
 * ```
 *
 * Roles can be used to represent a set of permissions. To restrict access to
 * function call, use {hasRole}:
 *
 * ```
 * function foo() public {
 *     require(hasRole(MY_ROLE, msg.sender));
 *     ...
 * }
 * ```
 *
 * Roles can be granted and revoked dynamically via the {grantRole} and
 * {revokeRole} functions. Each role has an associated admin role, and only
 * accounts that have a role's admin role can call {grantRole} and {revokeRole}.
 *
 * By default, the admin role for all roles is `DEFAULT_ADMIN_ROLE`, which means
 * that only accounts with this role will be able to grant or revoke other
 * roles. More complex role relationships can be created by using
 * {_setRoleAdmin}.
 *
 * WARNING: The `DEFAULT_ADMIN_ROLE` is also its own admin: it has permission to
 * grant and revoke this role. Extra precautions should be taken to secure
 * accounts that have been granted it.
```

```
 */
abstract contract AccessControl is Context, IAccessControl, ERC165 {
    struct RoleData {
        mapping(address => bool) members;
        bytes32 adminRole;
    }

    mapping(bytes32 => RoleData) private _roles;

    bytes32 public constant DEFAULT_ADMIN_ROLE = 0x00;

                /**
   * @dev Modifier that checks that an account has a spec
   * with a standardized message including the required r
   *
   * The format of the revert reason is given by the followi
   *
   * /^AccessControl: account (0x[0-9a-f]{40}) is missing role (0x[0-9a-f]{64})$/
   *
   * _Available since v4.1._
   */
    modifier onlyRole(bytes32 role) {
        _checkRole(role, _msgSender());
        _;
    }

                /**
   * @dev See {IERC165-supportsInterface}.
   */
    function supportsInterface(bytes4 interfaceId) public view virtual override returns (bool) {
        return interfaceId == type(IAccessControl).interfaceId || super.supportsInterface(interfaceId
    }

                /**
   * @dev Returns `true` if `account` has been granted `role`.
   */
    function hasRole(bytes32 role, address account) public view override returns (bool) {
        return _roles[role].members[account];
    }

                /**
   * @dev Revert with a standard message if `account` is missing `role`.
   *
   * The format of the revert reason is given by the followi
   *
   * /^AccessControl: account (0x[0-9a-f]{40}) is missing role (0x[0-9a-f]{64})$/
   */
    function _checkRole(bytes32 role, address account) internal view {
        if (!hasRole(role, account)) {
            revert(
                string(
                    abi.encodePacked(
                        "AccessControl: account ",
                        Strings.toHexString(uint160(account), 20),
                        " is missing role ",
                        Strings.toHexString(uint256(role), 32)
                    )
                )
```

```
            );
        }
    }

    /**
     * @dev Returns the admin role that controls `role`. See {grantRole} and
     * {revokeRole}.
     *
     * To change a role's admin, use {_setRoleAdmin}.
     */
    function getRoleAdmin(bytes32 role) public view override returns (bytes32) {
        return _roles[role].adminRole;
    }

    /**
     * @dev Grants `role` to `account`.
     *
     * If `account` had not been already granted `role`, emits a {RoleGranted}
     * event.
     *
     * Requirements:
     *
     * - the caller must have ``role``'s admin role.
     */
    function grantRole(bytes32 role, address account) public virtual override onlyRole(getRoleAdmin(r
        _grantRole(role, account);
    }

    /**
     * @dev Revokes `role` from `account`.
     *
     * If `account` had been granted `role`, emits a {RoleRevoked} event.
     *
     * Requirements:
     *
     * - the caller must have ``role``'s admin role.
     */
    function revokeRole(bytes32 role, address account) public virtual override onlyRole(getRoleAdmin(
        _revokeRole(role, account);
    }

    /**
     * @dev Revokes `role` from the calling account.
     *
     * Roles are often managed via {grantRole} and {revokeRole}: this function's
     * purpose is to provide a mechanism for accounts to lose their privileges
     * if they are compromised (such
     *
     * If the calling account had been granted `role`, emits a
     * event.
     *
     * Requirements:
     *
     * - the caller must be `account`.
```

```
    */
    function renounceRole(bytes32 role, address account) public virtual override {
        require(account == _msgSender(), "AccessControl: can only renounce roles for self");

        _revokeRole(role, account);
    }

    /**
     * @dev Grants `role` to `account`.
     *
     * If `account` had not been already granted `role`, emits a {RoleGranted}
     * event. Note that unlike {grantRole}, this function doesn't perform any
     * checks on the calling account.
     *
     * [WARNING]
     * ====
     * This function should only be called from the constructor when setting
     * up the initial roles for the system.
     *
     * Using this function in any other way is effectively circumventing the admin
     * system imposed by {AccessControl}.
     * ====
     */
    function _setupRole(bytes32 role, address account) internal virtual {
        _grantRole(role, account);
    }

    /**
     * @dev Sets `adminRole` as ``role``'s admin role.
     *
     * Emits a {RoleAdminChanged} event.
     */
    function _setRoleAdmin(bytes32 role, bytes32 adminRole) internal virtual {
        bytes32 previousAdminRole = getRoleAdmin(role);
        _roles[role].adminRole = adminRole;
        emit RoleAdminChanged(role, previousAdminRole, adminRole);
    }

    function _grantRole(bytes32 role, address account) private {
        if (!hasRole(role, account)) {
            _roles[role].members[account] = true;
            emit RoleGranted(role, account, _msgSender());
        }
    }

    function _revokeRole(bytes32 role, address account) private {
        if (hasRole(role, account)) {
            _roles[role].members[account] = false;
            emit RoleRevoked(role, account, _msgSender());
        }
    }
}

// File: contracts/access/Roles.sol


pragma solidity ^0.8.0;
```

```
contract Roles is AccessControl {
    bytes32 public constant MINTER_ROLE = keccak256("MINTER");

    constructor() {
        _setupRole(DEFAULT_ADMIN_ROLE, _msgSender());
        _setupRole(MINTER_ROLE, _msgSender());
    }

    modifier onlyMinter() {
        require(hasRole(MINTER_ROLE, _msgSender()), "Roles: caller does not have the MINTER role");
        _;
    }
}

// File: contracts/service/ServicePayer.sol



pragma solidity ^0.8.0;

interface IPayable {
    function pay(string memory serviceName) external payable;
}

            /**
 * @title ServicePayer
 * @dev Implementation of the ServicePayer
 */
abstract contract ServicePayer {
    constructor(address payable receiver, string memory serviceName) payable {
        IPayable(receiver).pay{value: msg.value}(serviceName);
    }
}

// File: contracts/token/ERC20/PowerfulERC20.sol



pragma solidity ^0.8.0;









            /**
 * @title PowerfulERC20
 * @dev Implementation of the PowerfulERC20
 */
contract PowerfulERC20 is
    ERC20Decimals,
    ERC20Capped,
    ERC20Mintable,
    ERC20Burnable,
    ERC1363,
    TokenRecover,
    Roles,
    ServicePayer
{
```

```
    constructor(
        string memory name_,
        string memory symbol_,
        uint8 decimals_,
        uint256 cap_,
        uint256 initialBalance_,
        address payable feeReceiver_
    )
        payable
        ERC20(name_, symbol_)
        ERC20Decimals(decimals_)
        ERC20Capped(cap_)
        ServicePayer(feeReceiver_, "PowerfulERC20")
    {
        // Immutable variables cannot be read during contract creation time
        // https://github.com/ethereum/solidity/issues/10463
        require(initialBalance_ <= cap_, "ERC20Capped: cap exceeded");
        ERC20._mint(_msgSender(), initialBalance_);
    }

    function decimals() public view virtual override(ERC20, ERC20Decimals) returns (uint8) {
        return super.decimals();
    }

    function supportsInterface(bytes4 interfaceId) public view virtual override(AccessControl, ERC136
        return super.supportsInterface(interfaceId);
    }

                /**
   * @dev Function to mint tokens.
   *
   *              NOTE:              restricting access to addresses with MINTER role. See {ERC20Mintable-mi
   *
   * @param account The address that              will              receive              the              m
   * @param amount The amount of tokens to mint
   */
    function _mint(address account, uint256 amount) internal override(ERC20, ERC20Capped) onlyMinter
        super._mint(account, amount);
    }

                /**
   * @dev Function to stop minting new tokens.
   *
   *              NOTE:              restricting access to owner only. See {ERC20Mintable-finishMinting}.
   */
    function _finishMinting() internal override onlyOwner {
        super._finishMinting();
    }
}
```


Analysis of audit results

Re-Entrancy

- **Description:**
  One of the features of smart contracts is the ability to call and utilise code of other external contracts. Contracts

also typically handle Blockchain Currency, and as such often send Blockchain Currency to various external user addresses. The operation of calling external contracts, or sending Blockchain Currency to an address, requires the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the contract to execute further code (i.e. through a fallback function) , including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Arithmetic Over/Under Flows

- **Description:**
  The Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unexpected Blockchain Currency

- **Description:**
  Typically when Blockchain Currency is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where Blockchain Currency can exist in a contract without having executed any code. Contracts which rely on code execution for every Blockchain Currency sent to the contract can be vulnerable to attacks where Blockchain Currency is forcibly sent to a contract.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Delegatecall

- **Description:**
  The CALL and DELEGATECALL opcodes are useful in allowing developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Default Visibilities

- **Description:**
  Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whBlockchain Currency a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs. Functions default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devestating vulernabilities in smart contracts as will be discussed in this section.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Entropy Illusion

- **Description:**
  All transactions on the blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no rand() function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

External Contract Referencing

- **Description:**
  One of the benefits of the global computer is the ability to re-use code and interact with contracts already deployed on the network. As a result, a large number of contracts reference external contracts and in general operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unsolved TODO comments

- **Description:**
  Check for Unsolved TODO comments
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Short Address/Parameter Attack

- **Description:**
  This attack is not specifically performed on Solidity contracts themselves but on third party applications that may interact with them. I add this attack for completeness and to be aware of how parameters can be manipulated in contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unchecked CALL Return Values

- **Description:**
  There a number of ways of performing external calls in solidity. Sending Blockchain Currency to external accounts is commonly performed via the transfer() method. However, the send() function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The call() and send() functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (intialised by call() or send()) fails, rather the call() or send() will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Race Conditions / Front Running

- **Description:**
  The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds of race conditions that can occur on the blockchain. There is a variety of good posts on this subject, a few are: Wiki - Safety, DASP - Front-Running and the Consensus - Smart Contract Best Practices.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Denial Of Service (DOS)

- **Description:**
  This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap Blockchain Currency in these contracts forever, as was the case with the Second Parity MultiSig hack
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Block Timestamp Manipulation

- **Description:**
  Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Constructors with Care

- **Description:**
  Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unintialised Storage Pointers

- **Description:**
  The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately intialising variables.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Floating Points and Numerical Precision

- **Description:**
  As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

tx.origin Authentication

- **Description:**
  Solidity has a global variable, tx.origin which traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in smart contracts leaves the contract vulnerable to a phishing-like attack.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Permission restrictions

- **Description:**
  Contract managers who can control liquidity or pledge pools, etc., or impose unreasonable restrictions on other users.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

armors.io

contact@armors.io