NASDEX(NSDX) Token

Smart Contract Audit

NASDEX(NSDX) Token Audit Summary

Project name : NASDEX(NSDX) Token Contract

Project address: None

Code URL : https://polygonscan.com/address/0xe8d17b127ba8b9899a160d9a07b69bca8e08bfc6

Commit : None

Project target : NASDEX(NSDX) Token Contract Audit

Blockchain : Polygon

Test result : PASSED

Audit Info

Audit NO : 0X202109190026

Audit Team : Armors Labs

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

NASDEX(NSDX) Token Audit

The NASDEX(NSDX) Token team asked us to review and audit their NASDEX(NSDX) 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
NASDEX(NSDX) Token Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2021-09-19

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 NASDEX(NSDX) 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
NSDXToken.sol	b9870da17cee0e1a6506d803ffd76da2

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 polygonscan.com on 2021-09-18
*/

// Sources flattened with hardhat v2.6.2 https://hardhat.org

// File @openzeppelin/contracts/token/ERC20/IERC20.sol@v4.3.1

// SPDX-License-Identifier: MIT

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@v4.3.1
```

```
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@v4.3.1


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@v4.3.1


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/access/Ownable.sol@v4.3.1

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 @openzeppelin/contracts/token/ERC20/extensions/draft-IERC20Permit.sol@v4.3.1


pragma solidity ^0.8.0;

/**
 * @dev Interface of the ERC20 Permit extension allowing approvals to be made via si
 * https://eips.ethereum.org/EIPS/eip-2612[EIP-2612].
 *
 * Adds the {permit} method, which can be used to change an
 * presenting a message signed by the account. By not rely
 * need to send a transaction, and thus is not required to hold Ether at all.
 */
interface IERC20Permit {
    /**
     * @dev Sets `value` as the allowance of `spender` over ``owner``'s tokens,
     * given ``owner``'s signed approval.
     *
     * IMPORTANT: The same issues {IERC20-approve} has related to transaction
     * ordering also apply here.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `deadline` must be a timestamp in the future.
     * - `v`, `r` and `s` must be a valid `secp256k1` signature from `owner`
```

```
     * over the EIP712-formatted function arguments.
     * - the signature must use ``owner``'s current nonce (see {nonces}).
     *
     * For more information on the signature format, see
     * https://eips.ethereum.org/EIPS/eip-2612#specification[relevant EIP
     * section].
     */
    function permit(
        address owner,
        address spender,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external;

                /**
     * @dev Returns the current nonce for `owner`. This value must be
     * included whenever a signature is generated for {permit}.
     *
     * Every successful call to {permit} increases ``owner``'s nonce by one. This
     * prevents a signature from being used multiple times.
     */
    function nonces(address owner) external view returns (uint256);

                /**
     * @dev Returns the domain separator used in the enco
     */
    // solhint-disable-next-line func-name-mixedcase
    function DOMAIN_SEPARATOR() external view returns (bytes32);
}


// File @openzeppelin/contracts/utils/cryptography/ECDSA.sol@v4.3.1


pragma solidity ^0.8.0;

            /**
 * @dev Elliptic Curve Digital Signature Algorithm (ECDSA) operations.
 *
 * These functions can be used to verify that a message was signed by the
 * of the private keys of a given address.
 */
library ECDSA {
    enum RecoverError {
        NoError,
        InvalidSignature,
        InvalidSignatureLength,
        InvalidSignatureS,
        InvalidSignatureV
    }

    function _throwError(RecoverError error) private pure {
        if (error == RecoverError.NoError) {
            return; // no error: do nothing
        } else if (error == RecoverError.InvalidSignature) {
            revert("ECDSA: invalid signature");
```

```
        } else if (error == RecoverError.InvalidSignatureLength) {
            revert("ECDSA: invalid signature length");
        } else if (error == RecoverError.InvalidSignatureS) {
            revert("ECDSA: invalid signature 's' value");
        } else if (error == RecoverError.InvalidSignatureV) {
            revert("ECDSA: invalid signature 'v' value");
        }
    }

    /**
     * @dev Returns the address that signed a hashed mess
     * `signature` or error string. This address can then be used for verification purposes.
     *
     * The `ecrecover` EVM opcode allows for malleable (non-unique) signatures:
     * this function rejects them by requiring the `s` value to be in the
     * half order, and the `v` value to be either 27 or 28.
     *
     * IMPORTANT: `hash` _must_ be the result of a hash op
     * verification to be secure: it is possible to craft signatures that
     * recover to arbitrary addresses for non-hashed data. A safe way to ensure
     * this is by receiving a hash of the original message (wh
     * be too long), and then calling {toEthSignedMessageHash} on it.
     *
     * Documentation for signature generation:
     * - with https://web3js.readthedocs.io/en/v1.3.4/web3-eth-accounts.html#sign[Web3.js]
     * - with https://docs.ethers.io/v5/api/signer/#Signer-signMessage[ethers]
     *
     * _Available since v4.3._
     */
    function tryRecover(bytes32 hash, bytes memory signature) internal pure returns (address, Recover
        // Check the signature length
        // - case 65: r,s,v signature (standard)
        // - case 64: r,vs signature (cf https://eips.ethereum.org/EIPS/eip-2098) _Available since v4
        if (signature.length == 65) {
            bytes32 r;
            bytes32 s;
            uint8 v;
            // ecrecover takes the signature parameters, and the only way to get them
            // currently is to use assembly.
            assembly {
                r := mload(add(signature, 0x20))
                s := mload(add(signature, 0x40))
                v := byte(0, mload(add(signature, 0x60)))
            }
            return tryRecover(hash, v, r, s);
        } else if (signature.length == 64) {
            bytes32 r;
            bytes32 vs;
            // ecrecover takes the signature parameters, and the only way to get them
            // currently is to use assembly.
            assembly {
                r := mload(add(signature, 0x20))
                vs := mload(add(signature, 0x40))
            }
            return tryRecover(hash, r, vs);
        } else {
            return (address(0), RecoverError.InvalidSignatureLength);
        }
    }
```

```
    /**
 * @dev Returns the address that signed a hashed mes
 * `signature`. This address can then be used for verification purposes.
 *
 * The `ecrecover` EVM opcode allows for malleable (non-unique) signatures:
 * this function rejects them by requiring the `s` value to be in the
 * half order, and the `v` value to be either 27 or 28.
 *
 * IMPORTANT: `hash` _must_ be the result of a hash o
 * verification to be secure: it is possible to craft signatures that
 * recover to arbitrary addresses for non-hashed data. A safe way to ensure
 * this is by receiving a hash of the original message (wh
 * be too long), and then calling {toEthSignedMessageHash} on it.
 */
  function recover(bytes32 hash, bytes memory signature) internal pure returns (address) {
      (address recovered, RecoverError error) = tryRecover(hash, signature);
      _throwError(error);
      return recovered;
  }

    /**
 * @dev Overload of {ECDSA-tryRecover} that receives the `r` and `vs` short-signatu
 *
 * See https://eips.ethereum.org/EIPS/eip-2098[EIP-2098 short signatures]
 *
 * _Available since v4.3._
 */
  function tryRecover(
      bytes32 hash,
      bytes32 r,
      bytes32 vs
  ) internal pure returns (address, RecoverError) {
      bytes32 s;
      uint8 v;
      assembly {
          s := and(vs, 0x7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff)
          v := add(shr(255, vs), 27)
      }
      return tryRecover(hash, v, r, s);
  }

    /**
 * @dev Overload of {ECDSA-recover} that receives the `r and `vs` short-signature fi
 *
 * _Available since v4.2._
 */
  function recover(
      bytes32 hash,
      bytes32 r,
      bytes32 vs
  ) internal pure returns (address) {
      (address recovered, RecoverError error) = tryRecover(hash, r, vs);
      _throwError(error);
      return recovered;
  }

    /**
```

```
 * @dev Overload of {ECDSA-tryRecover} that receives the `v`,
 * `r` and `s` signature fields separately.
 *
 * _Available since v4.3._
 */
  function tryRecover(
      bytes32 hash,
      uint8 v,
      bytes32 r,
      bytes32 s
  ) internal pure returns (address, RecoverError) {
      // EIP-2 still allows signature malleability for ecrecover(). Remove this possibility and mak
      // unique. Appendix F in the Ethereum Yellow paper (https://ethereum.github.io/yellowpaper/pa
      // the valid range for s in (301): 0 < s < secp256k1n ÷ 2 + 1, and for v in (302): v ∈ {27, 2
      // signatures from current libraries generate a unique signature with an s-value in the lower
      //
      // If your library generates malleable signatures, such as s-values in the upper range, calcu
      // with 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141 - s1 and flip v fr
      // vice versa. If your library also generates signatures with 0/1 for v instead 27/28, add 27
      // these malleable signatures as well.
      if (uint256(s) > 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0) {
          return (address(0), RecoverError.InvalidSignatureS);
      }
      if (v != 27 && v != 28) {
          return (address(0), RecoverError.InvalidSignatureV);
      }

      // If the signature is valid (and not malleable), return the signer address
      address signer = ecrecover(hash, v, r, s);
      if (signer == address(0)) {
          return (address(0), RecoverError.InvalidSignature);
      }

      return (signer, RecoverError.NoError);
  }

              /**
 * @dev Overload of {ECDSA-recover} that receives the `v`,
 * `r` and `s` signature fields separately.
 */
  function recover(
      bytes32 hash,
      uint8 v,
      bytes32 r,
      bytes32 s
  ) internal pure returns (address) {
      (address recovered, RecoverError error) = tryRecover(hash, v, r, s);
      _throwError(error);
      return recovered;
  }

              /**
 * @dev Returns an Ethereum Signed Message, created from a
 * produces hash corresponding to the one signed with the
 * https://eth.wiki/json-rpc/API#eth_sign[`eth_sign`]
 * JSON-RPC method as part of EIP-191.
 *
 * See {recover}.
 */
  function toEthSignedMessageHash(bytes32 hash) internal pure returns (bytes32) {
      // 32 is the length in bytes of hash,
```

```
        // enforced by the type signature above
        return keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", hash));
    }

    /**
     * @dev Returns an Ethereum Signed Typed Data, created from a
     * `domainSeparator` and a `structHash`. This produces hash corresponding
     * to the one signed with the
     * https://eips.ethereum.org/EIPS/eip-712[`eth_signTypedData`]
     * JSON-RPC method as part of EIP-712.
     *
     * See {recover}.
     */
    function toTypedDataHash(bytes32 domainSeparator, bytes32 structHash) internal pure returns (byte
        return keccak256(abi.encodePacked("\x19\x01", domainSeparator, structHash));
    }
}


// File @openzeppelin/contracts/utils/cryptography/draft-EIP712.sol@v4.3.1


pragma solidity ^0.8.0;

/**
 * @dev https://eips.ethereum.org/EIPS/eip-712[EIP 712] is a standard for hashing and
 *
 * The encoding specified in the EIP is very generic, and such
 * thus this contract does not implement the encoding itself. Protocols need to impleme
 * they need in their contracts using a combination of `abi.e
 *
 * This contract implements the EIP 712 domain separator ({_domainSeparatorV4}) tha
 * scheme, and the final step of the encoding to obtain
 * ({_hashTypedDataV4}).
 *
 * The implementation of the domain separator was designed to be as efficient as poss
 * the chain id to protect against replay attacks on an event
 *
 * NOTE: This contract implements the version of
 * https://docs.metamask.io/guide/signing-data.html[`eth_signTypedDataV4` in MetaMask].
 *
 * _Available since v3.4._
 */
abstract contract EIP712 {
    /* solhint-disable var-name-mixedcase */
    // Cache the domain separator as an immutable value, but also store the chain id that it correspo
    // invalidate the cached domain separator if the chain id changes.
    bytes32 private immutable _CACHED_DOMAIN_SEPARATOR;
    uint256 private immutable _CACHED_CHAIN_ID;

    bytes32 private immutable _HASHED_NAME;
    bytes32 private immutable _HASHED_VERSION;
    bytes32 private immutable _TYPE_HASH;

    /* solhint-enable var-name-mixedcase */

    /**
```

```
 * @dev Initializes the domain separator and parameter caches.
 *
 * The meaning of `name` and `version` is specified in
 * https://eips.ethereum.org/EIPS/eip-712#definition-of-domainseparator[EIP 712]:
 *
 * - `name`: the user readable name of the signing doma
 * - `version`: the current major version of the signing do
 *
 * NOTE: These parameters cannot be changed except through a
 * contract upgrade].
 */
  constructor(string memory name, string memory version) {
      bytes32 hashedName = keccak256(bytes(name));
      bytes32 hashedVersion = keccak256(bytes(version));
      bytes32 typeHash = keccak256(
          "EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)"
      );
      _HASHED_NAME = hashedName;
      _HASHED_VERSION = hashedVersion;
      _CACHED_CHAIN_ID = block.chainid;
      _CACHED_DOMAIN_SEPARATOR = _buildDomainSeparator(typeHash, hashedName, hashedVersion);
      _TYPE_HASH = typeHash;
  }

                /**
 * @dev Returns the domain separator for the current ch
 */
  function _domainSeparatorV4() internal view returns (bytes32) {
      if (block.chainid == _CACHED_CHAIN_ID) {
          return _CACHED_DOMAIN_SEPARATOR;
      } else {
          return _buildDomainSeparator(_TYPE_HASH, _HASHED_NAME, _HASHED_VERSION);
      }
  }

  function _buildDomainSeparator(
      bytes32 typeHash,
      bytes32 nameHash,
      bytes32 versionHash
  ) private view returns (bytes32) {
      return keccak256(abi.encode(typeHash, nameHash, versionHash, block.chainid, address(this)));
  }

                /**
 * @dev Given an already https://eips.ethereum.org/EIPS/eip-712#definition-of-hashs
 * function returns the hash of the fully encoded EIP712
 *
 * This hash can be used together with {ECDSA-recover} to obtain the signer of
 *
 * ``solidity
 * bytes32 digest = _hashTypedDataV4(keccak256(abi.encode(
 *    keccak256("Mail(address to,string contents)"),
 *    mailTo,
 *    keccak256(bytes(mailContents))
 * )));
 * address signer = ECDSA.recover(digest, signature);
 * ``
```

```
   */
    function _hashTypedDataV4(bytes32 structHash) internal view virtual returns (bytes32) {
        return ECDSA.toTypedDataHash(_domainSeparatorV4(), structHash);
    }
}


// File @openzeppelin/contracts/utils/Counters.sol@v4.3.1


pragma solidity ^0.8.0;

            /**
 * @title Counters
 * @author Matt Condon (@shrugs)
 * @dev Provides counters that can only be incremented, decremented or reset. This can be used e.g. to track
 * of elements in                a                mapping, issuing ERC721 ids, or counting request ids.
 *
 * Include with `using Counters for Counters.Counter;`
 */
library Counters {
    struct Counter {
        // This variable should never be directly accessed by users of the library: interactions must
        // the library's function. As of Solidity v0.5.2, this cannot be enforced, though there is a
        // this feature: see https://github.com/ethereum/solidity/issues/4637
        uint256 _value; // default: 0
    }

    function current(Counter storage counter) internal view returns (uint256) {
        return counter._value;
    }

    function increment(Counter storage counter) internal {
        unchecked {
            counter._value += 1;
        }
    }

    function decrement(Counter storage counter) internal {
        uint256 value = counter._value;
        require(value > 0, "Counter: decrement overflow");
        unchecked {
            counter._value = value - 1;
        }
    }

    function reset(Counter storage counter) internal {
        counter._value = 0;
    }
}


// File @openzeppelin/contracts/token/ERC20/extensions/draft-ERC20Permit.sol@v4.3.1


pragma solidity ^0.8.0;




            /**
```

```
 * @dev Implementation of the ERC20 Permit extension allowing approvals to be made
 * https://eips.ethereum.org/EIPS/eip-2612[EIP-2612].
 *
 * Adds the {permit} method, which can be used to change an
 * presenting a message signed by the account. By not rely
 * need to send a transaction, and thus is not required to hold Ether at all.
 *
 * _Available since v3.4._
 */
abstract contract ERC20Permit is ERC20, IERC20Permit, EIP712 {
    using Counters for Counters.Counter;

    mapping(address => Counters.Counter) private _nonces;

    // solhint-disable-next-line var-name-mixedcase
    bytes32 private immutable _PERMIT_TYPEHASH =
        keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)

    /**
   * @dev Initializes the {EIP712} domain separator using the
   *
   * It's a good idea to use the same `name` that is defined
   */
    constructor(string memory name) EIP712(name, "1") {}

    /**
   * @dev See {IERC20Permit-permit}.
   */
    function permit(
        address owner,
        address spender,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) public virtual override {
        require(block.timestamp <= deadline, "ERC20Permit: expired deadline");

        bytes32 structHash = keccak256(abi.encode(_PERMIT_TYPEHASH, owner, spender, value, _useNonce(

        bytes32 hash = _hashTypedDataV4(structHash);

        address signer = ECDSA.recover(hash, v, r, s);
        require(signer == owner, "ERC20Permit: invalid signature");

        _approve(owner, spender, value);
    }

    /**
   * @dev See {IERC20Permit-nonces}.
   */
    function nonces(address owner) public view virtual override returns (uint256) {
        return _nonces[owner].current();
    }

    /**
   * @dev See {IERC20Permit-DOMAIN_SEPARATOR}.
   */
```

```
    // solhint-disable-next-line func-name-mixedcase
    function DOMAIN_SEPARATOR() external view override returns (bytes32) {
        return _domainSeparatorV4();
    }

    /**
     * @dev "Consume a nonce": return the current value ar
     *
     * _Available since v4.1._
     */
    function _useNonce(address owner) internal virtual returns (uint256 current) {
        Counters.Counter storage nonce = _nonces[owner];
        current = nonce.current();
        nonce.increment();
    }
}


// File @openzeppelin/contracts/utils/math/Math.sol@v4.3.1


pragma solidity ^0.8.0;

/**
 * @dev Standard math utilities missing in the Solidity language.
 */
library Math {
    /**
     * @dev Returns the largest of two numbers.
     */
    function max(uint256 a, uint256 b) internal pure returns (uint256) {
        return a >= b ? a : b;
    }

    /**
     * @dev Returns the smallest of two numbers.
     */
    function min(uint256 a, uint256 b) internal pure returns (uint256) {
        return a < b ? a : b;
    }

    /**
     * @dev Returns the average of two numbers. The result is rounded towards
     * zero.
     */
    function average(uint256 a, uint256 b) internal pure returns (uint256) {
        // (a + b) / 2 can overflow.
        return (a & b) + (a ^ b) / 2;
    }

    /**
     * @dev Returns the ceiling of the division of two numbe
     *
     * This differs from standard division with `/` in that it rounds up instead
     * of rounding down.
     */
    function ceilDiv(uint256 a, uint256 b) internal pure returns (uint256) {
        // (a + b - 1) / b can overflow on addition, so we distribute.
```

```
        return a / b + (a % b == 0 ? 0 : 1);
    }
}

// File @openzeppelin/contracts/utils/math/SafeCast.sol@v4.3.1

pragma solidity ^0.8.0;

/**
 * @dev Wrappers over Solidity's uintXX/intXX casting operators with added overflow
 * checks.
 *
 * Downcasting from uint256/int256 in Solidity does not revert on overflow. This can
 * easily result in undesired exploitation or bugs, since developers usually
 * assume that overflows raise errors. `SafeCast` restores this intuition by
 * reverting the transaction when such an
 *
 * Using this library instead of the unchecked operations eliminates an
 * class of bugs, so it's recommended to use it always.
 *
 * Can be combined with {SafeMath} and {SignedSafeMath} to extend it to smaller types, by performing
 * all math on `uint256` and `int256` and then downcasting.
 */
library SafeCast {
    /**
     * @dev Returns the downcasted uint224 from uint256, reverting on
     * overflow (when the input is greater than largest uint224).
     *
     * Counterpart to Solidity's `uint224` operator.
     *
     * Requirements:
     *
     * - input must fit into 224 bits
     */
    function toUint224(uint256 value) internal pure returns (uint224) {
        require(value <= type(uint224).max, "SafeCast: value doesn't fit in 224 bits");
        return uint224(value);
    }

    /**
     * @dev Returns the downcasted uint128 from uint256, reverting on
     * overflow (when the input is greater than largest uint128).
     *
     * Counterpart to Solidity's `uint128` operator.
     *
     * Requirements:
     *
     * - input must fit into 128 bits
     */
    function toUint128(uint256 value) internal pure returns (uint128) {
        require(value <= type(uint128).max, "SafeCast: value doesn't fit in 128 bits");
        return uint128(value);
    }
```

```
    /**
     * @dev Returns the downcasted uint96 from uint256, reverting on
     * overflow (when the input is greater than largest uint96).
     *
     * Counterpart to Solidity's `uint96` operator.
     *
     * Requirements:
     *
     * - input must fit into 96 bits
     */
    function toUint96(uint256 value) internal pure returns (uint96) {
        require(value <= type(uint96).max, "SafeCast: value doesn't fit in 96 bits");
        return uint96(value);
    }

    /**
     * @dev Returns the downcasted uint64 from uint256, reverting on
     * overflow (when the input is greater than largest uint64).
     *
     * Counterpart to Solidity's `uint64` operator.
     *
     * Requirements:
     *
     * - input must fit into 64 bits
     */
    function toUint64(uint256 value) internal pure returns (uint64) {
        require(value <= type(uint64).max, "SafeCast: value doesn't fit in 64 bits");
        return uint64(value);
    }

    /**
     * @dev Returns the downcasted uint32 from uint256, reverting on
     * overflow (when the input is greater than largest uint32).
     *
     * Counterpart to Solidity's `uint32` operator.
     *
     * Requirements:
     *
     * - input must fit into 32 bits
     */
    function toUint32(uint256 value) internal pure returns (uint32) {
        require(value <= type(uint32).max, "SafeCast: value doesn't fit in 32 bits");
        return uint32(value);
    }

    /**
     * @dev Returns the downcasted uint16 from uint256, reverting on
     * overflow (when the input is greater than largest uint16).
     *
     * Counterpart to Solidity's `uint16` operator.
     *
     * Requirements:
     *
```

```
 * - input must fit into 16 bits
 */
  function toUint16(uint256 value) internal pure returns (uint16) {
      require(value <= type(uint16).max, "SafeCast: value doesn't fit in 16 bits");
      return uint16(value);
  }

                /**
 * @dev Returns the downcasted uint8 from uint256, reverting on
 * overflow (when the input is greater than largest uint8).
 *
 * Counterpart to Solidity's `uint8` operator.
 *
 * Requirements:
 *
 * - input must fit into 8 bits.
 */
  function toUint8(uint256 value) internal pure returns (uint8) {
      require(value <= type(uint8).max, "SafeCast: value doesn't fit in 8 bits");
      return uint8(value);
  }

                /**
 * @dev Converts a signed int256 into an unsigned uint2
 *
 * Requirements:
 *
 * - input must be greater than or equal to 0.
 */
  function toUint256(int256 value) internal pure returns (uint256) {
      require(value >= 0, "SafeCast: value must be positive");
      return uint256(value);
  }

                /**
 * @dev Returns the downcasted int128 from int256, reverting on
 * overflow (when the input is less than smallest int128 or
 * greater than largest int128).
 *
 * Counterpart to Solidity's `int128` operator.
 *
 * Requirements:
 *
 * - input must fit into 128 bits
 *
 * _Available since v3.1._
 */
  function toInt128(int256 value) internal pure returns (int128) {
      require(value >= type(int128).min && value <= type(int128).max, "SafeCast: value doesn't fit
      return int128(value);
  }

                /**
 * @dev Returns the downcasted int64 from int256, reverting on
 * overflow (when the input is less than smallest int64 or
```

```
 * greater than largest int64).
 *
 * Counterpart to Solidity's `int64` operator.
 *
 * Requirements:
 *
 * - input must fit into 64 bits
 *
 * _Available since v3.1._
 */
  function toInt64(int256 value) internal pure returns (int64) {
      require(value >= type(int64).min && value <= type(int64).max, "SafeCast: value doesn't fit in
      return int64(value);
  }

                /**
 * @dev Returns the downcasted int32 from int256, reverting on
 * overflow (when the input is less than smallest int32 or
 * greater than largest int32).
 *
 * Counterpart to Solidity's `int32` operator.
 *
 * Requirements:
 *
 * - input must fit into 32 bits
 *
 * _Available since v3.1._
 */
  function toInt32(int256 value) internal pure returns (int32) {
      require(value >= type(int32).min && value <= type(int32).max, "SafeCast: value doesn't fit in
      return int32(value);
  }

                /**
 * @dev Returns the downcasted int16 from int256, reverting on
 * overflow (when the input is less than smallest int16 or
 * greater than largest int16).
 *
 * Counterpart to Solidity's `int16` operator.
 *
 * Requirements:
 *
 * - input must fit into 16 bits
 *
 * _Available since v3.1._
 */
  function toInt16(int256 value) internal pure returns (int16) {
      require(value >= type(int16).min && value <= type(int16).max, "SafeCast: value doesn't fit in
      return int16(value);
  }

                /**
 * @dev Returns the downcasted int8 from int256, reverting on
 * overflow (when the input is less than smallest int8 or
```

```
     * greater than largest int8).
     *
     * Counterpart to Solidity's `int8` operator.
     *
     * Requirements:
     *
     * - input must fit into 8 bits.
     *
     * _Available since v3.1._
     */
    function toInt8(int256 value) internal pure returns (int8) {
        require(value >= type(int8).min && value <= type(int8).max, "SafeCast: value doesn't fit in 8
        return int8(value);
    }

                /**
     * @dev Converts an unsigned uint256 into a signed int2
     *
     * Requirements:
     *
     * - input must be less than or equal to maxInt256.
     */
    function toInt256(uint256 value) internal pure returns (int256) {
        // Note: Unsafe cast below is okay because `type(int256).max` is guaranteed to be positive
        require(value <= uint256(type(int256).max), "SafeCast: value doesn't fit in an int256");
        return int256(value);
    }
}


// File @openzeppelin/contracts/token/ERC20/extensions/ERC20Votes.sol@v4.3.1


pragma solidity ^0.8.0;




            /**
 * @dev Extension of ERC20 to support Compound-like voting and delegation. This v
 * and supports token supply up to 2^224^ - 1, while COMP is limited to 2^96^ - 1.
 *
 * NOTE: If exact COMP compatibility is required, use the {E
 *
 * This extension keeps a history (checkpoints) of each account's vote power. Vote pow
 * by calling the {delegate} function directly, or by providing a
 * power can be queried through the public accessors {getVotes} and {getPastVotes}.
 *
 * By default, token balance does not account for voting power. This makes transfers cheaper. The downside is that
 * requires users to delegate to themselves in order to activate checkpoints and have their voting power tracked.
 * Enabling self-delegation can easily be done by overriding the {delegates} function. k
 * will significantly increase the base gas cost of transfers
 *
 * _Available since v4.2._
 */
```

```
abstract contract ERC20Votes is ERC20Permit {
    struct Checkpoint {
        uint32 fromBlock;
        uint224 votes;
    }

    bytes32 private constant _DELEGATION_TYPEHASH =
        keccak256("Delegation(address delegatee,uint256 nonce,uint256 expiry)");

    mapping(address => address) private _delegates;
    mapping(address => Checkpoint[]) private _checkpoints;
    Checkpoint[] private _totalSupplyCheckpoints;

                /**
   * @dev Emitted when an account changes their delegate.
   */
    event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed to

                /**
   * @dev Emitted when a token transfer or delegate change results in changes to
   */
    event DelegateVotesChanged(address indexed delegate, uint256 previousBalance, uint256 newBalance)

                /**
   * @dev Get the `pos`-th checkpoint for `account`.
   */
    function checkpoints(address account, uint32 pos) public view virtual returns (Checkpoint memory)
        return _checkpoints[account][pos];
    }

                /**
   * @dev Get number of checkpoints for `account`.
   */
    function numCheckpoints(address account) public view virtual returns (uint32) {
        return SafeCast.toUint32(_checkpoints[account].length);
    }

                /**
   * @dev Get the address `account` is currently delegating to.
   */
    function delegates(address account) public view virtual returns (address) {
        return _delegates[account];
    }

                /**
   * @dev Gets the current votes balance for `account`
   */
    function getVotes(address account) public view returns (uint256) {
        uint256 pos = _checkpoints[account].length;
        return pos == 0 ? 0 : _checkpoints[account][pos - 1].votes;
    }

                /**
   * @dev Retrieve the number of votes for `account` at the
   *
   * Requirements:
   *
   * - `blockNumber` must have been already mined
```

```
 */
  function getPastVotes(address account, uint256 blockNumber) public view returns (uint256) {
      require(blockNumber < block.number, "ERC20Votes: block not yet mined");
      return _checkpointsLookup(_checkpoints[account], blockNumber);
  }

              /**
 * @dev Retrieve the `totalSupply` at the end of `blockN
 * It is but NOT the sum of all the
 *
 * Requirements:
 *
 * - `blockNumber` must have been already mined
 */
  function getPastTotalSupply(uint256 blockNumber) public view returns (uint256) {
      require(blockNumber < block.number, "ERC20Votes: block not yet mined");
      return _checkpointsLookup(_totalSupplyCheckpoints, blockNumber);
  }

              /**
 * @dev Lookup a value in a list of (sorted) checkpoints.
 */
  function _checkpointsLookup(Checkpoint[] storage ckpts, uint256 blockNumber) private view returns
      // We run a binary search to look for the earliest checkpoint taken after `blockNumber`.
      //
      // During the loop, the index of the wanted checkpoint remains in the range [low-1, high).
      // With each iteration, either `low` or `high` is moved towards the middle of the range to ma
      // - If the middle checkpoint is after `blockNumber`, we look in [low, mid)
      // - If the middle checkpoint is before or equal to `blockNumber`, we look in [mid+1, high)
      // Once we reach a single value (when low == high), we've found the right checkpoint at the i
      // out of bounds (in which case we're looking too far in the past and the result is 0).
      // Note that if the latest checkpoint available is exactly for `blockNumber`, we end up with
      // past the end of the array, so we technically don't find a checkpoint after `blockNumber`,
      // the same.
      uint256 high = ckpts.length;
      uint256 low = 0;
      while (low < high) {
          uint256 mid = Math.average(low, high);
          if (ckpts[mid].fromBlock > blockNumber) {
              high = mid;
          } else {
              low = mid + 1;
          }
      }

      return high == 0 ? 0 : ckpts[high - 1].votes;
  }

              /**
 * @dev Delegate votes from the sender to `delegatee`.
 */
  function delegate(address delegatee) public virtual {
      return _delegate(_msgSender(), delegatee);
  }

              /**
 * @dev Delegates votes from signer to `delegatee`
 */
  function delegateBySig(
      address delegatee,
```

```
        uint256 nonce,
        uint256 expiry,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) public virtual {
        require(block.timestamp <= expiry, "ERC20Votes: signature expired");
        address signer = ECDSA.recover(
            _hashTypedDataV4(keccak256(abi.encode(_DELEGATION_TYPEHASH, delegatee, nonce, expiry))),
            v,
            r,
            s
        );
        require(nonce == _useNonce(signer), "ERC20Votes: invalid nonce");
        return _delegate(signer, delegatee);
    }

    /**
     * @dev Maximum token supply. Defaults to `type(uint224).max` (2^224^ - 1).
     */
    function _maxSupply() internal view virtual returns (uint224) {
        return type(uint224).max;
    }

    /**
     * @dev Snapshots the totalSupply after it has been increased.
     */
    function _mint(address account, uint256 amount) internal virtual override {
        super._mint(account, amount);
        require(totalSupply() <= _maxSupply(), "ERC20Votes: total supply risks overflowing votes");

        _writeCheckpoint(_totalSupplyCheckpoints, _add, amount);
    }

    /**
     * @dev Snapshots the totalSupply after it has been decreased.
     */
    function _burn(address account, uint256 amount) internal virtual override {
        super._burn(account, amount);

        _writeCheckpoint(_totalSupplyCheckpoints, _subtract, amount);
    }

    /**
     * @dev Move voting power when tokens are transferred.
     *
     * Emits a {DelegateVotesChanged} event.
     */
    function _afterTokenTransfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual override {
        super._afterTokenTransfer(from, to, amount);

        _moveVotingPower(delegates(from), delegates(to), amount);
    }

    /**
     * @dev Change delegation for `delegator` to `delegatee`.
     *
```

```
    * Emits events {DelegateChanged} and {DelegateVotesChanged}.
    */
    function _delegate(address delegator, address delegatee) internal virtual {
        address currentDelegate = delegates(delegator);
        uint256 delegatorBalance = balanceOf(delegator);
        _delegates[delegator] = delegatee;

        emit DelegateChanged(delegator, currentDelegate, delegatee);

        _moveVotingPower(currentDelegate, delegatee, delegatorBalance);
    }

    function _moveVotingPower(
        address src,
        address dst,
        uint256 amount
    ) private {
        if (src != dst && amount > 0) {
            if (src != address(0)) {
                (uint256 oldWeight, uint256 newWeight) = _writeCheckpoint(_checkpoints[src], _subtrac
                emit DelegateVotesChanged(src, oldWeight, newWeight);
            }

            if (dst != address(0)) {
                (uint256 oldWeight, uint256 newWeight) = _writeCheckpoint(_checkpoints[dst], _add, am
                emit DelegateVotesChanged(dst, oldWeight, newWeight);
            }
        }
    }

    function _writeCheckpoint(
        Checkpoint[] storage ckpts,
        function(uint256, uint256) view returns (uint256) op,
        uint256 delta
    ) private returns (uint256 oldWeight, uint256 newWeight) {
        uint256 pos = ckpts.length;
        oldWeight = pos == 0 ? 0 : ckpts[pos - 1].votes;
        newWeight = op(oldWeight, delta);

        if (pos > 0 && ckpts[pos - 1].fromBlock == block.number) {
            ckpts[pos - 1].votes = SafeCast.toUint224(newWeight);
        } else {
            ckpts.push(Checkpoint({fromBlock: SafeCast.toUint32(block.number), votes: SafeCast.toUint
        }
    }

    function _add(uint256 a, uint256 b) private pure returns (uint256) {
        return a + b;
    }

    function _subtract(uint256 a, uint256 b) private pure returns (uint256) {
        return a - b;
    }
}


// File @openzeppelin/contracts/token/ERC20/extensions/ERC20VotesComp.sol@v4.3.1


pragma solidity ^0.8.0;

            /**
 * @dev Extension of ERC20 to support Compound's voting and delegation. This version exactly matches Compou
 * interface, with the drawback of only supporting supply up to (2^96^ - 1).
```

```
 *
 * NOTE: You should use this contract if you
 * with Governor Alpha or Bravo) and if you are sure
 * {ERC20Votes} variant of this module.
 *
 * This extension keeps a history (checkpoints) of each account's vote pow
 * by calling the {delegate} function directly, or by providing a
 * power can be queried through the public accessors {getCurrentVotes} and {getPriorV
 *
 * By default, token balance does not account for voting power. This makes transfers cheaper. The downside is that
 * requires users to delegate to themselves in order to activate checkpoints and have their voting power tracked.
 * Enabling self-delegation can easily be done by overriding the {delegates} function. K
 * will significantly increase the base gas cost of transfers
 *
 * _Available since v4.2._
 */
abstract contract ERC20VotesComp is ERC20Votes {
    /**
     * @dev Comp version of the {getVotes} accessor, with `uint96` return type.
     */
    function getCurrentVotes(address account) external view returns (uint96) {
        return SafeCast.toUint96(getVotes(account));
    }

    /**
     * @dev Comp version of the {getPastVotes} accessor, with `uint96` return type.
     */
    function getPriorVotes(address account, uint256 blockNumber) external view returns (uint96) {
        return SafeCast.toUint96(getPastVotes(account, blockNumber));
    }

    /**
     * @dev Maximum token supply. Reduced to `type(uint96).max` (2^96^ - 1) to fit COMP interface.
     */
    function _maxSupply() internal view virtual override returns (uint224) {
        return type(uint96).max;
    }
}


// File contracts/NSDXToken.sol


pragma solidity ^0.8.0;
// NSDXToken with Governance
contract NSDXToken is Ownable, ERC20, ERC20Permit, ERC20VotesComp {
    constructor() ERC20("NASDEX Token", "NSDX") ERC20Permit("NASDEX Token") {}

    function _afterTokenTransfer(address from, address to, uint256 amount)
    internal
    override(ERC20, ERC20Votes)
    {
        super._afterTokenTransfer(from, to, amount);
    }

    function _mint(address to, uint256 amount)
    internal
    override(ERC20, ERC20Votes)
```

```
    {
        super._mint(to, amount);
    }

    function _burn(address account, uint256 amount)
    internal
    override(ERC20, ERC20Votes)
    {
        super._burn(account, amount);
    }


    // @notice Creates `_amount` token to `_to`. Must only be called by the owner.
    function mint(address _to, uint256 _amount) public onlyOwner {
        _mint(_to, _amount);
    }
}
```

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