
Armors Labs
Bright Union BV
Smart Contract Audit

Bright Union BV Audit Summary

Project name : Bright Union BV Contract

Project address: None

Code URL : https://etherscan.io/address/0xbeab712832112bd7664226db7cd025b153d3af55#code

Code URL : https://etherscan.io/address/0xfa6980a9151a989ca4ee719faab998aef1d522a3#code

Commit : None

Project target : Bright Union BV Contract Audit

Blockchain : Ethereum

Test result : PASSED

Audit Info

Audit NO : 0X202108040016

Audit Team : Armors Labs

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

Bright Union BV Audit

The Bright Union BV team asked us to review and audit their Bright Union BV 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
Bright Union BV Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2021-08-04

Audit results

Note that: This contract is upgradable and this audit is only valid for the current version.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Bright Union BV 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.

(Statement: Armors Labs reports only on facts that have occurred or existed before this report is issued and assumes corresponding responsibilities. Armors Labs is not able to determine the security of its smart contracts and is not responsible for any subsequent or existing facts after this report is issued. The security audit analysis and other content of this report are only based on the documents and information provided by the information provider to

Armors Labs at the time of issuance of this report (" information provided " for short). 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.)

Audited target file

file	md5
IERC20Permit.sol	032552b15db2480f4ea77b61d60eac86
ISTKBrightToken.sol	2c440036082e4868388a27d93603dcec
IBrightStaking.sol	af3f4b4243016c856ed9b9b6a993fecd
IContractsRegistry.sol	f5533fb403d3829a2b5eda43445e9a85
BRIGHTToken.sol	db3ecb6bacb9e6ffe938a14ff65e7675
EIP712.sol	22e068e33d7cbc33cab6225653d1f924
ERC20Permit.sol	ab174c2aea1ff30995b01b0191014cb5
STKBrightToken.sol	7e7e5eb71aaa4367beaed57a036d14ab
BrightTokenVesting.sol	776c0e77ec71aab13ae562247bcbf7ef
BrightStaking.sol	2fdaa0b3f431a56ba3846095242ab09a
ContractsRegistry.sol	fa00a9eea7aa7cf32c09cbe89946f1c4
Globals.sol	252a8651210fcd2f74c7590cb42eca73

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

Vulnerability	status
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
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

IERC20Permit.sol

```
// SPDX-License-Identifier: MIT

pragma solidity =0.7.4;

/**
 * @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 rel
 * 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 encoc
     */
    // solhint-disable-next-line func-name-mixedcase
    function DOMAIN_SEPARATOR() external view returns (bytes32);
}
```

ISTKBrightToken.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.4;

import "@openzeppelin/contracts-upgradeable/token/ERC20/IERC20Upgradeable.sol";

interface ISTKBrightToken is IERC20Upgradeable {
    function mint(address account, uint256 amount) external;

    function burn(address account, uint256 amount) external;
}
```

IBrightStaking.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.4;

import "./token/ISTKBrightToken.sol";

interface IBrightStaking {
       event StakedBright(
        uint256 stakedBright,
        uint256 mintedStkBright,
        address indexed recipient
    );

    event WithdrawnBright(
        uint256 withdrawnBright,
        uint256 burnedStkBright,
        address indexed recipient
    );

    struct WithdrawalInfo {
        uint256 coolDownTimeEnd;
        uint256 amountStkBrightRequested;
    }

    event SweepedUnusedRewards(address recipient, uint256 amount);

       function stkBrightToken() external returns (ISTKBrightToken);

       function stake(uint256 _amountBright) external;

    function stakeWithPermit(uint256 _amountBright, uint8 _v, bytes32 _r, bytes32 _s) external;

    function whenCanWithdrawBrightReward(address _address) external view returns (uint256);

    function callWithdraw(uint256 _amountStkBrightUnlock) external;

       function withdraw() external;

    function getWithdrawalInfo(address _userAddr) external view
    returns (
        uint256 _amountStkBrightRequested,
        uint256 _amountBrightRequested,
        uint256 _unlockPeriod,
        uint256 _availableFor
    );

       function stakingReward(uint256 _amount) external view returns (uint256);

       function getStakedBright(address _address) external view returns (uint256);

       function setRewardPerBlock(uint256 _amount) external;

    function sweepUnusedRewards() external;
}
```

IContractsRegistry.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.4;

interface IContractsRegistry {

    function getBrightContract() external view returns (address);
```

```
    function getBrightStakingContract() external view returns (address);

    function getSTKBrightContract() external view returns (address);

}
```

BRIGHTToken.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.4;

import "./ERC20Permit.sol";

contract BRIGHTToken is ERC20Permit {
  uint256 constant TOTAL_SUPPLY = 110 * (10**6) * (10**18);

  constructor(address tokenReceiver) ERC20Permit("Bright Union") ERC20("Bright Union", "BRIGHT") {
    _mint(tokenReceiver, TOTAL_SUPPLY);
  }
}
```

EIP712.sol

```
// SPDX-License-Identifier: MIT

pragma solidity =0.7.4;

/**
 * @dev https://eips.ethereum.org/EIPS/eip-712[EIP 712] is a standard for hashing and signing of type
 *
 * The encoding specified in the EIP is very generic, and such a generic implementation in Solidity i
 * thus this contract does not implement the encoding itself. Protocols need to implement the type-sp
 * they need in their BXH using a combination of `abi.encode` and `keccak256`.
 *
 * This contract implements the EIP 712 domain separator ({_domainSeparatorV4}) that is used as part
 * scheme, and the final step of the encoding to obtain the message digest that is then signed via EC
 * ({_hashTypedDataV4}).
 *
 * The implementation of the domain separator was designed to be as efficient as possible while still
 * the chain id to protect against replay attacks on an eventual fork of the chain.
 *
 * NOTE: This contract implements the version of the encoding known as "v4", as implemented by the JS
 * https:///docs.metamask.io/guide/signing-data.html[`eth_signTypedDataV4` in MetaMask].
 */
abstract contract EIP712 {
  /* solhint-disable var-name-mixedcase */
  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 domain, i.e. the name of the DApp or the protoco
   * - `version`: the current major version of the signing domain.
   *
   * NOTE: These parameters cannot be changed except through a xref:learn::upgrading-smart-BXH.adoc[s
```

```
     * contract upgrade].
     */
    constructor(string memory name, string memory version) internal {
        bytes32 hashedName = keccak256(bytes(name));
        bytes32 hashedVersion = keccak256(bytes(version));
        bytes32 typeHash = keccak256("EIP712Domain(string name,string version,uint256 chainId,address ver
        _HASHED_NAME = hashedName;
        _HASHED_VERSION = hashedVersion;
        _TYPE_HASH = typeHash;
    }

    /**
     * @dev Returns the domain separator for the current chain.
     */
    function _domainSeparatorV4() internal view returns (bytes32) {
        return _buildDomainSeparator(_TYPE_HASH, _HASHED_NAME, _HASHED_VERSION);
    }

    function _buildDomainSeparator(
        bytes32 typeHash,
        bytes32 name,
        bytes32 version
    ) private view returns (bytes32) {
        return keccak256(abi.encode(typeHash, name, version, _getChainId(), address(this)));
    }

    /**
     * @dev Given an already https://eips.ethereum.org/EIPS/eip-712#definition-of-hashstruct[hashed str
     * function returns the hash of the fully encoded EIP712 message for this domain.
     *
     * This hash can be used together with {ECDSA-recover} to obtain the signer of a message. For examp
     *
     * ```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 returns (bytes32) {
        return keccak256(abi.encodePacked("\x19\x01", _domainSeparatorV4(), structHash));
    }

    function _getChainId() private view returns (uint256 chainId) {
        this; // silence state mutability warning without generating bytecode - see https://github.com/et
        // solhint-disable-next-line no-inline-assembly
        assembly {
            chainId := 1
        }
    }
}
```

ERC20Permit.sol

```
// SPDX-License-Identifier: MIT

pragma solidity =0.7.4;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/utils/Counters.sol";
import "./EIP712.sol";
```

```
import "../interfaces/token/IERC20Permit.sol";

/**
 * @dev Implementation of the ERC20 Permit extension allowing approvals to be made via signatures, as
 * https://eips.ethereum.org/EIPS/eip-2612[EIP-2612].
 *
 * Adds the {permit} method, which can be used to change an account's ERC20 allowance (see {IERC20-al
 * presenting a message signed by the account. By not relying on `{IERC20-approve}`, the token holder
 * need to send a transaction, and thus is not required to hold Ether at all.
 */
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 `name` parameter, and setting `version`
   *
   * It's a good idea to use the same `name` that is defined as the ERC20 token name.
   */
  constructor(string memory name) internal 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 {
    // solhint-disable-next-line not-rely-on-time
    require(block.timestamp <= deadline, "ERC20Permit: expired deadline");

    bytes32 structHash =
      keccak256(abi.encode(_PERMIT_TYPEHASH, owner, spender, value, _nonces[owner].current(), deadlin

    bytes32 hash = _hashTypedDataV4(structHash);

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

    _nonces[owner].increment();
    _approve(owner, spender, value);
  }

  /**
   * @dev See {IERC20Permit-nonces}.
   */
  function nonces(address owner) public view 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 Overload of {ECDSA-recover-bytes32-bytes-} that receives the `v`,
   * `r` and `s` signature fields separately.
   */
  function recover(
    bytes32 hash,
    uint8 v,
    bytes32 r,
    bytes32 s
  ) internal pure returns (address) {
    // EIP-2 still allows signature malleability for ecrecover(). Remove this possibility and make th
    // unique. Appendix F in the Ethereum Yellow paper (https://ethereum.github.io/yellowpaper/paper.
    // the valid range for s in (281): 0 < s < secp256k1n ÷ 2 + 1, and for v in (282): v ∈ {27, 28}.
    // signatures from current libraries generate a unique signature with an s-value in the lower hal
    //
    // If your library generates malleable signatures, such as s-values in the upper range, calculate
    // with 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141 - s1 and flip v from 2
    // vice versa. If your library also generates signatures with 0/1 for v instead 27/28, add 27 to
    // these malleable signatures as well.
    require(
      uint256(s) <= 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0,
      "ECDSA: invalid signature 's' value"
    );
    require(v == 27 || v == 28, "ECDSA: invalid signature 'v' value");

    // If the signature is valid (and not malleable), return the signer address
    address signer = ecrecover(hash, v, r, s);
    require(signer != address(0), "ECDSA: invalid signature");

    return signer;
  }
}
```

STKBrightToken.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.4;

import "@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol";

import "../interfaces/IContractsRegistry.sol";

import "../interfaces/token/ISTKBrightToken.sol";

contract STKBrightToken is ISTKBrightToken, ERC20Upgradeable {
    IContractsRegistry public contractsRegistry;

    modifier onlyBrightStaking() {
        require(contractsRegistry.getBrightStakingContract() == _msgSender(), "Caller is not the Brig
        _;
    }

    function initialize(IContractsRegistry _contractsRegistry) external initializer {
        __ERC20_init("Stake BRIGHT", "stkBRIGHT");

        contractsRegistry = _contractsRegistry;
      }

    function mint(address account, uint256 amount) public override onlyBrightStaking {
        _mint(account, amount);
```

```
    }

    function burn(address account, uint256 amount) public override onlyBrightStaking {
        _burn(account, amount);
    }
}
```

BrightTokenVesting.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.4;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts-upgradeable/math/MathUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/math/SafeMathUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/proxy/Initializable.sol";


contract BrightTokenVesting is Initializable, OwnableUpgradeable {
  using MathUpgradeable for uint256;
  using SafeMathUpgradeable for uint256;

  enum VestingSchedule {
    SEEDROUND,
    PRIVATEROUND,
    TREASURY,
    FOUNDERS,
    TEAM,
    ADVISORS,
    COMMUNITY
  }

  struct Vesting {
    bool isValid;
    address beneficiary;
    uint256 amount;
    VestingSchedule vestingSchedule;
    uint256 paidAmount;
    bool isCancelable;
  }

  struct LinearVestingSchedule {
    uint256 portionOfTotal;
    uint256 startDate;
    uint256 periodInSeconds;
    uint256 portionPerPeriod;
    uint256 cliffInPeriods;
  }

  uint256 public constant SECONDS_IN_MONTH = 60 * 60 * 24 * 30;
  uint256 public constant PORTION_OF_TOTAL_PRECISION = 10**10;
  uint256 public constant PORTION_PER_PERIOD_PRECISION = 10**10;

  IERC20 public token;
  Vesting[] public vestings;
  uint256 public amountInVestings;
  uint256 public tgeTimestamp;
  mapping(VestingSchedule => LinearVestingSchedule[]) public vestingSchedules;

  event TokenSet(IERC20 token);
  event VestingAdded(uint256 vestingId, address beneficiary);
  event VestingCanceled(uint256 vestingId);
```

```
event VestingWithdraw(uint256 vestingId, uint256 amount);

function initialize(uint256 _tgeTimestamp) public initializer {
  __Ownable_init();
  tgeTimestamp = _tgeTimestamp;

  initializeVestingSchedules();
}


function initializeVestingSchedules() internal {
  addLinearVestingSchedule(
    VestingSchedule.SEEDROUND,
    LinearVestingSchedule({
      portionOfTotal: PORTION_OF_TOTAL_PRECISION.div(10),             //10% at TGE
      startDate: tgeTimestamp.sub(SECONDS_IN_MONTH),
      periodInSeconds: SECONDS_IN_MONTH,
      portionPerPeriod: PORTION_PER_PERIOD_PRECISION,
      cliffInPeriods: 0
    })
  );

  addLinearVestingSchedule(
    VestingSchedule.SEEDROUND,
    LinearVestingSchedule({
      portionOfTotal: PORTION_OF_TOTAL_PRECISION.div(100).mul(90),    // 90% after TGE
      startDate: tgeTimestamp,
      periodInSeconds: SECONDS_IN_MONTH,
      portionPerPeriod: PORTION_PER_PERIOD_PRECISION.div(6),          // over 6 month
      cliffInPeriods: 0
    })
  );

  addLinearVestingSchedule(
    VestingSchedule.PRIVATEROUND,
    LinearVestingSchedule({
      portionOfTotal: PORTION_OF_TOTAL_PRECISION.div(10),             // 10% at TGE
      startDate: tgeTimestamp.sub(SECONDS_IN_MONTH),
      periodInSeconds: SECONDS_IN_MONTH,
      portionPerPeriod: PORTION_PER_PERIOD_PRECISION,
      cliffInPeriods: 0
    })
  );

  addLinearVestingSchedule(
    VestingSchedule.PRIVATEROUND,
    LinearVestingSchedule({
      portionOfTotal: PORTION_OF_TOTAL_PRECISION.div(100).mul(90),    // 90% after TGE
      startDate: tgeTimestamp,
      periodInSeconds: SECONDS_IN_MONTH,
      portionPerPeriod: PORTION_PER_PERIOD_PRECISION.div(6),          // over 6 month
      cliffInPeriods: 0
    })
  );

  addLinearVestingSchedule(
    VestingSchedule.TREASURY,
    LinearVestingSchedule({
      portionOfTotal: PORTION_OF_TOTAL_PRECISION,                     // starting TGE date
      startDate: tgeTimestamp.sub(SECONDS_IN_MONTH),
      periodInSeconds: SECONDS_IN_MONTH,
      portionPerPeriod: PORTION_PER_PERIOD_PRECISION.div(25),         // 4% per month, over 1 (TGE)
      cliffInPeriods: 0
    })
  );
```

```
addLinearVestingSchedule(
  VestingSchedule.FOUNDERS,
  LinearVestingSchedule({
    portionOfTotal: PORTION_OF_TOTAL_PRECISION.div(100),            // 1% at TGE
    startDate: tgeTimestamp.sub(SECONDS_IN_MONTH),
    periodInSeconds: SECONDS_IN_MONTH,
    portionPerPeriod: PORTION_PER_PERIOD_PRECISION,
    cliffInPeriods: 0
  })
);

addLinearVestingSchedule(
  VestingSchedule.FOUNDERS,
  LinearVestingSchedule({
    portionOfTotal: PORTION_OF_TOTAL_PRECISION.div(100).mul(99),    // 99% there after
    startDate: tgeTimestamp,
    periodInSeconds: SECONDS_IN_MONTH,
    portionPerPeriod: PORTION_PER_PERIOD_PRECISION.div(36),         // 2.75% per month, over 36 m
    cliffInPeriods: 0
  })
);

addLinearVestingSchedule(
  VestingSchedule.TEAM,
  LinearVestingSchedule({
    portionOfTotal: PORTION_OF_TOTAL_PRECISION,                     // starting TGE date
    startDate: tgeTimestamp.sub(SECONDS_IN_MONTH),
    periodInSeconds: SECONDS_IN_MONTH,
    portionPerPeriod: PORTION_PER_PERIOD_PRECISION.div(25),         // 4% per month, over 1 (TGE)
    cliffInPeriods: 0
  })
);

addLinearVestingSchedule(
  VestingSchedule.ADVISORS,
  LinearVestingSchedule({
    portionOfTotal: PORTION_OF_TOTAL_PRECISION.div(10),             // 10% at TGE
    startDate: tgeTimestamp.sub(SECONDS_IN_MONTH),
    periodInSeconds: SECONDS_IN_MONTH,
    portionPerPeriod: PORTION_PER_PERIOD_PRECISION,
    cliffInPeriods: 0
  })
);

addLinearVestingSchedule(
  VestingSchedule.ADVISORS,
  LinearVestingSchedule({
    portionOfTotal: PORTION_OF_TOTAL_PRECISION.div(100).mul(90),    // 90% there after
    startDate: tgeTimestamp,
    periodInSeconds: SECONDS_IN_MONTH,
    portionPerPeriod: PORTION_PER_PERIOD_PRECISION.div(24),         // 3,75% per month, over 24 m
    cliffInPeriods: 0
  })
);

addLinearVestingSchedule(
  VestingSchedule.COMMUNITY,
  LinearVestingSchedule({
    portionOfTotal: PORTION_OF_TOTAL_PRECISION.div(100),            // 1% at TGE
    startDate: tgeTimestamp.sub(SECONDS_IN_MONTH),
    periodInSeconds: SECONDS_IN_MONTH,
    portionPerPeriod: PORTION_PER_PERIOD_PRECISION,
    cliffInPeriods: 0
  })
);
```

```
    addLinearVestingSchedule(
      VestingSchedule.COMMUNITY,
      LinearVestingSchedule({
        portionOfTotal: PORTION_OF_TOTAL_PRECISION.div(100).mul(99),    // 99% there after
        startDate: tgeTimestamp,
        periodInSeconds: SECONDS_IN_MONTH,
        portionPerPeriod: PORTION_PER_PERIOD_PRECISION.div(36),         // 2.75% per month, over 36 m
        cliffInPeriods: 0
      })
    );

  }

  function addLinearVestingSchedule(VestingSchedule _type, LinearVestingSchedule memory _schedule) in
    vestingSchedules[_type].push(_schedule);
  }

  function setToken(IERC20 _token) external onlyOwner {
    require(address(token) == address(0), "token is already set");
    token = _token;
    emit TokenSet(token);
  }

  function createPartlyPaidVestingBulk(
    address[] calldata _beneficiary,
    uint256[] calldata _amount,
    VestingSchedule[] calldata _vestingSchedule,
    bool[] calldata _isCancelable,
    uint256[] calldata _paidAmount
  ) external onlyOwner {
    require(
      _beneficiary.length == _amount.length &&
        _beneficiary.length == _vestingSchedule.length &&
        _beneficiary.length == _isCancelable.length &&
        _beneficiary.length == _paidAmount.length,
      "Parameters length mismatch"
    );

    for (uint256 i = 0; i < _beneficiary.length; i++) {
      _createVesting(_beneficiary[i], _amount[i], _vestingSchedule[i], _isCancelable[i], _paidAmount[
    }
  }

  function createVestingBulk(
    address[] calldata _beneficiary,
    uint256[] calldata _amount,
    VestingSchedule[] calldata _vestingSchedule,
    bool[] calldata _isCancelable
  ) external onlyOwner {
    require(
      _beneficiary.length == _amount.length &&
        _beneficiary.length == _vestingSchedule.length &&
        _beneficiary.length == _isCancelable.length,
      "Parameters length mismatch"
    );

    for (uint256 i = 0; i < _beneficiary.length; i++) {
      _createVesting(_beneficiary[i], _amount[i], _vestingSchedule[i], _isCancelable[i], 0);
    }
  }

  function createVesting(
    address _beneficiary,
    uint256 _amount,
    VestingSchedule _vestingSchedule,
    bool _isCancelable
```

```
  ) external onlyOwner returns (uint256 vestingId) {
    return _createVesting(_beneficiary, _amount, _vestingSchedule, _isCancelable, 0);
  }

  function _createVesting(
    address _beneficiary,
    uint256 _amount,
    VestingSchedule _vestingSchedule,
    bool _isCancelable,
    uint256 _paidAmount
  ) internal returns (uint256 vestingId) {
    require(_beneficiary != address(0), "Cannot create vesting for zero address");

    uint256 amountToVest = _amount.sub(_paidAmount);
    require(getTokensAvailable() >= amountToVest, "Not enough tokens");
    amountInVestings = amountInVestings.add(amountToVest);

    vestingId = vestings.length;
    vestings.push(
      Vesting({
        isValid: true,
        beneficiary: _beneficiary,
        amount: _amount,
        vestingSchedule: _vestingSchedule,
        paidAmount: _paidAmount,
        isCancelable: _isCancelable
      })
    );

    emit VestingAdded(vestingId, _beneficiary);
  }

  function cancelVesting(uint256 _vestingId) external onlyOwner {
    Vesting storage vesting = getVesting(_vestingId);
    require(vesting.isCancelable, "Vesting is not cancelable");

    _forceCancelVesting(_vestingId, vesting);
  }

  function _forceCancelVesting(uint256 _vestingId, Vesting storage _vesting) internal {
    require(_vesting.isValid, "Vesting is canceled");
    _vesting.isValid = false;
    uint256 amountReleased = _vesting.amount.sub(_vesting.paidAmount);
    amountInVestings = amountInVestings.sub(amountReleased);

    emit VestingCanceled(_vestingId);
  }

  function withdrawFromVestingBulk(uint256 _offset, uint256 _limit) external {
    uint256 to = (_offset + _limit).min(vestings.length).max(_offset);
    for (uint256 i = _offset; i < to; i++) {
      Vesting storage vesting = getVesting(i);
      if (vesting.isValid) {
        _withdrawFromVesting(vesting, i);
      }
    }
  }

  function withdrawFromVesting(uint256 _vestingId) external {
    Vesting storage vesting = getVesting(_vestingId);
    require(vesting.isValid, "Vesting is canceled");

    _withdrawFromVesting(vesting, _vestingId);
  }

  function _withdrawFromVesting(Vesting storage _vesting, uint256 _vestingId) internal {
```

```
    uint256 amountToPay = _getWithdrawableAmount(_vesting);
    if (amountToPay > 0) {
      _vesting.paidAmount = _vesting.paidAmount.add(amountToPay);
      amountInVestings = amountInVestings.sub(amountToPay);
      token.transfer(_vesting.beneficiary, amountToPay);
      emit VestingWithdraw(_vestingId, amountToPay);
    }
  }

  function getWithdrawableAmount(uint256 _vestingId) external view returns (uint256) {
    Vesting storage vesting = getVesting(_vestingId);
    require(vesting.isValid, "Vesting is canceled");

    return _getWithdrawableAmount(vesting);
  }

  function _getWithdrawableAmount(Vesting storage _vesting) internal view returns (uint256) {
    return calculateAvailableAmount(_vesting).sub(_vesting.paidAmount);
  }

  function calculateAvailableAmount(Vesting storage _vesting) internal view returns (uint256) {
    LinearVestingSchedule[] storage vestingSchedule = vestingSchedules[_vesting.vestingSchedule];
    uint256 amountAvailable = 0;
    for (uint256 i = 0; i < vestingSchedule.length; i++) {
      LinearVestingSchedule storage linearSchedule = vestingSchedule[i];
      if (linearSchedule.startDate > block.timestamp) return amountAvailable;
      uint256 amountThisLinearSchedule = calculateLinearVestingAvailableAmount(linearSchedule, _vesti
      amountAvailable = amountAvailable.add(amountThisLinearSchedule);
    }
    return amountAvailable;
  }

  function calculateLinearVestingAvailableAmount(LinearVestingSchedule storage _linearVesting, uint25
    internal
    view
    returns (uint256)
  {
    uint256 elapsedPeriods = calculateElapsedPeriods(_linearVesting);
    if (elapsedPeriods <= _linearVesting.cliffInPeriods) return 0;
    uint256 amountThisVestingSchedule = _amount.mul(_linearVesting.portionOfTotal).div(PORTION_OF_TOT
    uint256 amountPerPeriod =
      amountThisVestingSchedule.mul(_linearVesting.portionPerPeriod).div(PORTION_PER_PERIOD_PRECISION
    return amountPerPeriod.mul(elapsedPeriods).min(amountThisVestingSchedule);
  }

  function calculateElapsedPeriods(LinearVestingSchedule storage _linearVesting) private view returns
    return block.timestamp.sub(_linearVesting.startDate).div(_linearVesting.periodInSeconds);
  }

  function getVesting(uint256 _vestingId) internal view returns (Vesting storage) {
    require(_vestingId < vestings.length, "No vesting with such id");
    return vestings[_vestingId];
  }

  function withdrawExcessiveTokens() external onlyOwner {
    token.transfer(owner(), getTokensAvailable());
  }

  function getTokensAvailable() public view returns (uint256) {
    return token.balanceOf(address(this)).sub(amountInVestings);
  }

  function getVestingById(uint256 _vestingId)
    public
    view
    returns (
```

```
      bool isValid,
      address beneficiary,
      uint256 amount,
      VestingSchedule vestingSchedule,
      uint256 paidAmount,
      bool isCancelable
    )
  {
    Vesting storage vesting = getVesting(_vestingId);
    isValid = vesting.isValid;
    beneficiary = vesting.beneficiary;
    amount = vesting.amount;
    vestingSchedule = vesting.vestingSchedule;
    paidAmount = vesting.paidAmount;
    isCancelable = vesting.isCancelable;
  }

  function getVestingsCount() public view returns (uint256 _vestingsCount) {
    return vestings.length;
  }
}
```

BrightStaking.sol

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.4;

import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

import "./interfaces/IContractsRegistry.sol";
import "./interfaces/IBrightStaking.sol";

import "./interfaces/token/ISTKBrightToken.sol";
import "./interfaces/token/IERC20Permit.sol";
import "./Globals.sol";

contract BrightStaking is IBrightStaking, OwnableUpgradeable {
    using SafeMath for uint256;

    uint256 internal constant WITHDRAWING_COOLDOWN_DURATION = 7 days;
    uint256 internal constant WITHDRAWAL_PHASE_DURATION = 2 days;

    IERC20 public brightToken;
    ISTKBrightToken public override stkBrightToken;
    uint256 public lastUpdateBlock;
    uint256 public rewardPerBlock;
    uint256 public totalPool;

    mapping(address => WithdrawalInfo) private withdrawalsInfo;

    modifier updateRewardPool() {
        if (totalPool == 0) {
            lastUpdateBlock = block.number;
        }

        totalPool = totalPool.add(_calculateReward());
        lastUpdateBlock = block.number;
        _;
    }
```

```
    receive() payable external {
        revert('BrightStk: No ETH here');
    }

    function initialize(uint256 _rewardPerBlock, IContractsRegistry _contractsRegistry)
        external
        initializer
    {
        __Ownable_init();

        lastUpdateBlock = block.number;
        rewardPerBlock = _rewardPerBlock;

        brightToken = IERC20(_contractsRegistry.getBrightContract());
        stkBrightToken = ISTKBrightToken(_contractsRegistry.getSTKBrightContract());
    }

    function stake(uint256 _amountBright) external override updateRewardPool {
        brightToken.transferFrom(_msgSender(), address(this), _amountBright);

        _stake(_msgSender(), _amountBright);
    }

    function stakeWithPermit(uint256 _amountBright, uint8 _v, bytes32 _r, bytes32 _s) external overri
        IERC20Permit(address(brightToken)).permit(
            _msgSender(),
            address(this),
            _amountBright,
            MAX_INT,
            _v,
            _r,
            _s
        );

        brightToken.transferFrom(_msgSender(), address(this), _amountBright);
        _stake(_msgSender(), _amountBright);
    }

    function _stake(address _staker, uint256 _amountBright) internal {
        require(_amountBright > 0, "BrightStk: cant stake 0 tokens");
        uint256 amountStkBright = _convertToStkBright(_amountBright);
        stkBrightToken.mint(_staker, amountStkBright);

        totalPool = totalPool.add(_amountBright);

        emit StakedBright(_amountBright, amountStkBright, _staker);
    }

    // There is a second withdrawal phase of 48 hours to actually receive the rewards.
    // If a user misses this period, in order to withdraw he has to wait for 7 days again.
    // It will return:
    // 0 if cooldown time didn't start or if phase duration (48hs) has expired
    // #coolDownTimeEnd Time when user can withdraw.
    function whenCanWithdrawBrightReward(address _address) public view override returns (uint256) {
        return
        withdrawalsInfo[_address].coolDownTimeEnd.add(WITHDRAWAL_PHASE_DURATION) >=
        block.timestamp
        ? withdrawalsInfo[_address].coolDownTimeEnd
        : 0;
    }

    function callWithdraw(uint256 _amountStkBrightUnlock) external override {
        require(_amountStkBrightUnlock > 0, "BrightStk: can't unlock 0 tokens");

        require(
            stkBrightToken.balanceOf(_msgSender()) >= _amountStkBrightUnlock,
```

```
            "BrightStk: not enough stkBright to unlock"
        );

        withdrawalsInfo[_msgSender()] = WithdrawalInfo(
            block.timestamp.add(WITHDRAWING_COOLDOWN_DURATION),
            _amountStkBrightUnlock
        );
    }

    function withdraw() external override updateRewardPool {
        uint256 _whenCanWithdrawBrightReward = whenCanWithdrawBrightReward(_msgSender());
        require(_whenCanWithdrawBrightReward != 0, "BrightStk: unlock not started/exp");
        require(_whenCanWithdrawBrightReward <= block.timestamp, "BrightStk: cooldown not reached");

        uint256 _amountStkBright = withdrawalsInfo[_msgSender()].amountStkBrightRequested;
        delete withdrawalsInfo[_msgSender()];

        require(
            stkBrightToken.balanceOf(_msgSender()) >= _amountStkBright,
            "BrightStk: not enough stkBright tokens to withdraw"
        );

        uint256 amountBright = _convertToBright(_amountStkBright);
        require(
            brightToken.balanceOf(address(this)) >= amountBright,
            "BrightStk: not enough Bright tokens in the pool"
        );
        stkBrightToken.burn(_msgSender(), _amountStkBright);

        totalPool = totalPool.sub(amountBright);

        brightToken.transfer(_msgSender(), amountBright);

        emit WithdrawnBright(amountBright, _amountStkBright, _msgSender());
    }

    function getWithdrawalInfo(address _userAddr) external view override
        returns (
            uint256 _amountStkBrightRequested,
            uint256 _amountBrightRequested,
            uint256 _unlockPeriod,
            uint256 _availableFor
        )
    {
        _unlockPeriod = whenCanWithdrawBrightReward(_userAddr);
        if (_unlockPeriod > 0) {
            _amountStkBrightRequested = withdrawalsInfo[_userAddr].amountStkBrightRequested;
            _amountBrightRequested = _convertToBright(_amountStkBrightRequested);

            uint256 endUnlockPeriod = _unlockPeriod.add(WITHDRAWAL_PHASE_DURATION);
            _availableFor = _unlockPeriod <= block.timestamp ? endUnlockPeriod : 0;
        }
    }

    function stakingReward(uint256 _amount) external view override returns (uint256) {
        return _convertToBright(_amount);
    }

    function getStakedBright(address _address) external view override returns (uint256) {
        uint256 balance = stkBrightToken.balanceOf(_address);
        return balance > 0 ? _convertToBright(balance) : 0;
    }

    function setRewardPerBlock(uint256 _amount) external override onlyOwner updateRewardPool {
        rewardPerBlock = _amount;
    }
```

```
    function sweepUnusedRewards() external override onlyOwner updateRewardPool {
        uint256 contractBalance = brightToken.balanceOf(address(this));

        require(
            contractBalance > totalPool,
            "BrightStk: There are no unused tokens to revoke"
        );

        uint256 unusedTokens = contractBalance.sub(totalPool);

        brightToken.transfer(_msgSender(), unusedTokens);
        emit SweepedUnusedRewards(_msgSender(), unusedTokens);
    }

    function outstandingRewards() external view returns (uint256) {
        return _calculateReward();
    }

    function _convertToStkBright(uint256 _amount) internal view returns (uint256) {
        uint256 tStkBrightToken = stkBrightToken.totalSupply();
        uint256 stakingPool = totalPool.add(_calculateReward());

        if (stakingPool > 0 && tStkBrightToken > 0) {
            _amount = tStkBrightToken.mul(_amount).div(stakingPool);
        }

        return _amount;
    }

    function _convertToBright(uint256 _amount) internal view returns (uint256) {
        uint256 tStkBrightToken = stkBrightToken.totalSupply();
        uint256 stakingPool = totalPool.add(_calculateReward());

        return tStkBrightToken > 0 ? stakingPool.mul(_amount).div(tStkBrightToken) : 0;
    }

    function _calculateReward() internal view returns (uint256) {
        uint256 blocksPassed = block.number.sub(lastUpdateBlock);
        return rewardPerBlock.mul(blocksPassed);
    }
}
```


ContractsRegistry.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.4;
pragma experimental ABIEncoderV2;

import "@openzeppelin/contracts/access/AccessControl.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/proxy/TransparentUpgradeableProxy.sol";

import "./interfaces/IContractsRegistry.sol";

contract ContractsRegistry is IContractsRegistry, Ownable, AccessControl {
    mapping (bytes32 => address) private _contracts;
    mapping (address => bool) private _isProxy;

    bytes32 constant public REGISTRY_ADMIN_ROLE = keccak256("REGISTRY_ADMIN_ROLE");
    bytes32 constant public BRIGHT_STAKING_NAME = keccak256("BRIGHT_STAKING_NAME");
    bytes32 constant public BRIGHT_NAME = keccak256("BRIGHT");
    bytes32 constant public STKBRIGHT_NAME = keccak256("STK_BRIGHT");
```

```
    modifier onlyAdmin() {
        require(hasRole(REGISTRY_ADMIN_ROLE, msg.sender), "ContractsRegistry: Caller is not an admin"
        _;
    }

    constructor() {
        _setupRole(REGISTRY_ADMIN_ROLE, owner());
        _setRoleAdmin(REGISTRY_ADMIN_ROLE, REGISTRY_ADMIN_ROLE);
    }

    function getBrightContract() external view override returns (address) {
        return getContract(BRIGHT_NAME);
    }

    function getBrightStakingContract() external view override returns (address) {
        return getContract(BRIGHT_STAKING_NAME);
    }

    function getSTKBrightContract() external view override returns (address) {
        return getContract(STKBRIGHT_NAME);
    }

    function getContract(bytes32 name) public view returns (address) {
        require(_contracts[name] != address(0), "ContractsRegistry: This mapping doesn't exist");

        return _contracts[name];
    }

    function upgradeContract(bytes32 name, address newImplementation) external onlyAdmin {
        require(_contracts[name] != address(0), "ContractsRegistry: This mapping doesn't exist");

        TransparentUpgradeableProxy proxy = TransparentUpgradeableProxy(payable(_contracts[name]));

        require(_isProxy[address(proxy)], "ContractsRegistry: Can't upgrade not a proxy contract");

        proxy.upgradeTo(newImplementation);
    }

    function addContract(bytes32 name, address contractAddress) external onlyAdmin {
        require(contractAddress != address(0), "ContractsRegistry: Null address is forbidden");

        _contracts[name] = contractAddress;
    }

    function addProxyContract(bytes32 name, address contractAddress) external onlyAdmin {
        require(contractAddress != address(0), "ContractsRegistry: Null address is forbidden");

        TransparentUpgradeableProxy proxy = new TransparentUpgradeableProxy(
            contractAddress, address(this), ""
        );

        _contracts[name] = address(proxy);
        _isProxy[address(proxy)] = true;
    }

    function deleteContract(bytes32 name) external onlyAdmin {
        require(_contracts[name] != address(0), "ContractsRegistry: This mapping doesn't exist");

        delete _isProxy[_contracts[name]];
        delete _contracts[name];
    }
}
```

Globals.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.7.4;
pragma experimental ABIEncoderV2;

uint256 constant SECONDS_IN_THE_YEAR = 365 * 24 * 60 * 60; // 365 days * 24 hours * 60 minutes * 60 s
uint256 constant MAX_INT = 2**256 - 1;

uint256 constant PRECISION = 10**25;
uint256 constant PERCENTAGE_100 = 100 * PRECISION;
```

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!
```

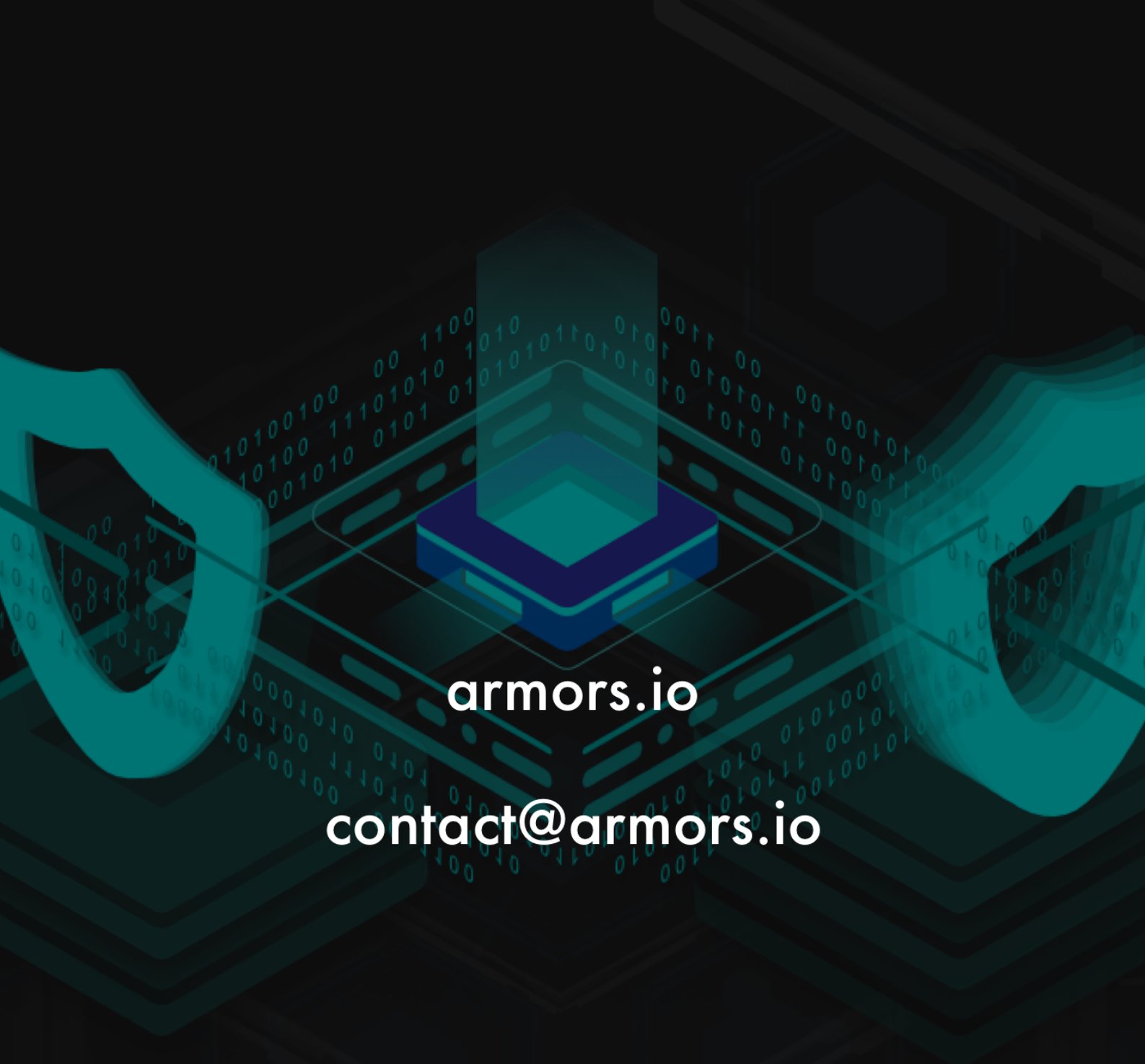
- **Security suggestion:**
  no.

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