
Armors Labs
FIDO
Smart Contract Audit

FIDO Audit Summary

Project name : FIDO Contract

Project address: None

Code URL : https://github.com/fido-project/audit-contracts

Commit : 4f5387c53123c928fc7207d6b0af03179ae1c46a

Project target : FIDO Contract Audit

Blockchain : Huobi ECO Chain（Heco）

Test result : PASSED

Audit Info

Audit NO : 0X202105080008

Audit Team : Armors Labs

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

FIDO Audit

The FIDO team asked us to review and audit their FIDO 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
FIDO Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2021-05-08

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 FIDO 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
./ReentrancyGuard.sol	f3c1a971bfe306e3330870e4c918b631
./IDOInfo.sol	3d07082e3aba519265ac5604626868a5
./ERC20/ERC20Pausable.sol	5ba41b496341f26393f62f30572c037a
./ERC20/ERC20Mintable.sol	eca19f6cccd0fe57e66a29e93f22a728
./ERC20/ERC20Burnable.sol	0dca068ad242853e73f0cf501f9e3cdd
./ERC20/ERC20.sol	ebf1cff6a039a54f1a0568b38bc0e0f9
./StakeRewardPerBlock.sol	24746a23ec6711d1915c2d96629ee151
./IDOToken.sol	18807f36832c6b409f7b1bceb1e8d151
./StakeLPRewardPerDay.sol	587c5571ac1a36b2d408401628f95e41
./StakeTokenPool.sol	1d2e5278917ba95c720058df654959f1
./IDOFactory.sol	33d95bba0b9326602a0a95945969b408
./MFILPool.sol	17a98a57f950f51cd56002a221ec8dbf
./MFIL-IDOToken-Factory.sol	d252b31c0d8a2a7094b265ef796d21f0
./oracle/RateOracle.sol	7dc71dd75865bd11c522b0188dc15bf3
./FidoMargin.sol	2cce4a0c81f44ba5b62578cdee70e1b2
./Pausable.sol	e70cfad9554608935e5da690a1e8c81a
./libraries/TransferHelper.sol	9bf500f7d995b8348c9b5dcb3aa24312
./libraries/SafeMath.sol	e03e12206057e809eb76c5f681170c32
./Context.sol	2adbd82f6d055a4751566d4671512b03
./StakeLPRewardPerBlock.sol	6354cdbf357428bd5ecdd8a9898de912
./MFIL-IDOToken.sol	0042ccfe460baba7fd36e2222f8391df
./StakeRewardPerDay.sol	d7634f0eb3bad2ca0e8959e0f8852de7
./Ownable.sol	3c73ff1bfb400374dd48f30345945264
./MFIL.sol	eb7df44eaeecc07d4828f02aa0ce25d4
./FIDO-USDT.sol	60ab120afbc00261f1a67997ba7377f9
./FidoMember.sol	36df7526bede4bc66945dc2b74d8e160
./Migrations.sol	50b3b1dc92806dc8f604fc8c5c65518f
./FIDO.sol	943ff11ac96ea389da3b89c7591e404f
./IDOUserRouter.sol	ca9d18b7c322c91091e4b1be48f62bf5
./MdexRouter.sol	3645059c5fd05fed8f8b249fa9151da3

file	md5
./interfaces/IIDOInfo.sol	ce29173dcd612f77336cfba16b621e56
./interfaces/IMdexFactory.sol	13e2c61c92e70f81275a984181fd7176
./interfaces/IFidoUsdtLPPool.sol	1ac8e832f3b8811c5d89bf8a72ad44df
./interfaces/IHFIL.sol	a2214ca7300b49e37218b1494834dcb6
./interfaces/IIDOToken.sol	4b4f9fa3bd54bf924a559014dd8450f3
./interfaces/IMFIL.sol	fb358aad558e600d8c283701e8778b2c
./interfaces/IRateOracle.sol	2d8b9715941c9d7582ed3553d28e55c2
./interfaces/IMdexPair.sol	082fc325db03f7698928f5afa7f9bf54
./interfaces/IERC20.sol	e0a41531d159d3a32f84b7a3ecf9fabb
./interfaces/IERC20Mintable.sol	ceb9c3b25228976d6e5ee7328ceff899
./interfaces/IFidoMember.sol	1e7368dbb56892fe14261b24abc58b2e

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

Vulnerability	status
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

```
    function changeStakeMin(uint256 newStakeMin) external;

    function joinFide(address _inviter) external;

    function changeNFiLPool(address newNFiLpool) external;

    function changeFideOfficial(address newFideOfficial) external;

    function changePool(address newPool) external;

    function getFollowerCount(address member) view external returns (uint256);

    function calcInviteRate(address member) view external returns (address inviter0, address inviter1,

    event AdjustRate(uint256 baseRate, uint256 multiRate);

    event AdjustStakeMin(uint256 stakeMin);

    event NewMember(address indexed member, address indexed inviter, uint256 joinBlockHeight);

    event OperatorshipTransferred(address indexed previousOperator, address indexed newOperator);

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
}// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

/**
 * @dev Interface of the ERC20 standard as defined in the
 */
interface IERC20Mintable {
    /**
     * @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 operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount)
        external
        returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * 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 caller's tokens.
```

```
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * 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 operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) external returns (bool);

    function mint(address recipient, uint256 amount) external;

    /**
     * @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 an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );

    event AddMinter(address indexed minter);
    event RemoveMinter(address indexed minter);
    event Mint(
        address indexed minter,
        address indexed recipient,
        uint256 amount
    );
}
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0 <0.8.6;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
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 operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * 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 caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * 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 owner, 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 operation succeeded.
     *
     * 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 an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
```

```
    event Approval(address indexed owner, address indexed spender, uint256 value);
}
// SPDX-License-Identifier: MIT

pragma solidity ^0.7.6;
interface IPAIRPair {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);

    function symbol() external pure returns (string memory);

    function decimals() external pure returns (uint8);

    function totalSupply() external view returns (uint);

    function balanceOf(address owner) external view returns (uint);

    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);

    function transfer(address to, uint value) external returns (bool);

    function transferFrom(address from, address to, uint value) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);

    function PERMIT_TYPEHASH() external pure returns (bytes32);

    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, by

    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    function MINIMUM_LIQUIDITY() external pure returns (uint);

    function factory() external view returns (address);

    function token0() external view returns (address);

    function token1() external view returns (address);

    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTim

    function price0CumulativeLast() external view returns (uint);

    function price1CumulativeLast() external view returns (uint);

    function kLast() external view returns (uint);

    function mint(address to) external returns (uint liquidity);

    function burn(address to) external returns (uint amount0, uint amount1);
```

```
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;
    function price(address token, uint256 baseDecimal) external view returns (uint256);
    function initialize(address, address) external;
}// SPDX-License-Identifier: MIT
pragma solidity ^0.7.6;

interface IRouterOracle {
  function owner() view external returns (address);
  function paused() view external returns (bool);
  function renounceOwnership() external;
  function requester() view external returns (address);
  function transferOwnership(address newOwner) external;
  function updater() view external returns (address);
  function pause() external;
  function unPause() external;
  function status() view external returns (bool);
  function lastRequestTime() view external returns (uint256);
  function lastUpdateTime() view external returns (uint256);
  function decimals() view external returns (uint8);
  function rate() view external returns (uint256);
  function request() external;
  function update(uint256 rate_) external;
  event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
  event Paused(address account);
  event Request(uint256 indexed timestamp);
  event RequestershipTransferred(address indexed previousRequester, address indexed newRequester);
  event Unpaused(address account);
  event Update(uint256 indexed timestamp, uint256 rate);
  event UpdatershipTransferred(address indexed previousUpdater, address indexed newUpdater);
}// SPDX-License-Identifier: MIT
pragma solidity ^0.7.6;

interface IMPS1 {
  function allowance(address owner, address spender) view external returns (uint256);
  function approve(address spender, uint256 amount) external returns (bool);
  function balanceOf(address account) view external returns (uint256);
```

```
    function burn(uint256 amount) external;

    function burnFrom(address account, uint256 amount) external;

    function decimals() view external returns (uint8);

    function decreaseAllowance(address spender, uint256 subtractedValue) external returns (bool);

    function increaseAllowance(address spender, uint256 addedValue) external returns (bool);

    function isMinter(address) view external returns (bool);

    function mint(address recipient, uint256 amount) external;

    function name() view external returns (string memory);

    function operator() view external returns (address);

    function owner() view external returns (address);

    function paused() view external returns (bool);

    function renounceOwnership() external;

    function symbol() view external returns (string memory);

    function totalSupply() view external returns (uint256);

    function transfer(address recipient, uint256 amount) external returns (bool);

    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    function transferOwnership(address newOwner) external;

    function transferOperatorship(address newOperator) external;

    function pause() external;

    function unPause() external;

    function addMinter(address minter) external;

    function removeMinter(address minter) external;

    function cap() view external returns (uint256);

    event AddMinter(address indexed minter);

    event Approval(address indexed owner, address indexed spender, uint256 value);

    event Mint(address indexed minter, address indexed recipient, uint256 amount);

    event OperatorshipTransferred(address indexed previousOperator, address indexed newOperator);

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

    event Paused(address account);

    event RemoveMinter(address indexed minter);

    event Transfer(address indexed from, address indexed to, uint256 value);

    event Unpaused(address account);
}// SPDX-License-Identifier: MIT
pragma solidity ^0.7.0;
```

```
interface IERC20Token {
  function allowance(address owner, address spender) view external returns (uint256);

  function approve(address spender, uint256 amount) external returns (bool);

  function balanceOf(address account) view external returns (uint256);

  function burn(uint256 amount) external;

  function burnFrom(address account, uint256 amount) external;

  function dailyBaseStage() view external returns (uint256);

  function decimals() view external returns (uint8);

  function decreaseAllowance(address spender, uint256 subtractedValue) external returns (bool);

  function factory() view external returns (address);

  function gasPrice() view external returns (uint256);

  function hardDriveFee() view external returns (uint256);

  function hfil() view external returns (address);

  function hfilRecipient() view external returns (address);

  function idoEndTime() view external returns (uint256);

  function idoStartTime() view external returns (uint256);

  function idoStatus() view external returns (bool);

  function increaseAllowance(address spender, uint256 addedValue) external returns (bool);

  function name() view external returns (string memory);

  function node() view external returns (string memory);

  function paused() view external returns (bool);

  function price() view external returns (uint256);

  function router() view external returns (address);

  function sealFee() view external returns (uint256);

  function sender() view external returns (address);

  function symbol() view external returns (string memory);

  function teamStartTime() view external returns (uint256);

  function totalSupply() view external returns (uint256);

  function transfer(address recipient, uint256 amount) external returns (bool);

  function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

  function upgrator() view external returns (address);

  function setPrice(uint256 gasPrice_, uint256 sealFee_, uint256 hardDriveFee_) external;

  function setTimes(uint256 idoStartTime_, uint256 idoEndTime_) external;
```

```
    }

    function addIDO(address idoToken) external onlyOperator {
        require(!exist[idoToken], "IDOINFO: already here");
        exist[idoToken] = true;
        IDOList.push(idoToken);
        emit AddIDO(block.timestamp, idoToken);
    }

    function setPayer(address pool, address payer) external onlyOperator {
        require(isPool[pool], "IDOINFO: pool not found");
        payer[pool][payer] = pool;
        emit SetPayer(block.timestamp, pool, payer);
    }

    function setStakeAddress(address idoToken, address _stakeAddress)
        external
        onlyOperator
    {
        require(exist[idoToken], "IDOINFO: idoToken not found");
        isPool[_stakeAddress] = true;
        poolIdoToken[_stakeAddress] = idoToken;
        stakeAddress[idoToken] = _stakeAddress;
        emit SetStakeAddress(block.timestamp, idoToken, _stakeAddress);
    }
}
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

abstract contract ReentrancyGuard {

    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;

    uint256 private _status;

    constructor () {
        _status = _NOT_ENTERED;
    }

    modifier nonReentrant() {
        // On the first call to nonReentrant, _notEntered will be true
        require(_status != _ENTERED, "ReentrancyGuard: reentrant call");

        // Any calls to nonReentrant after this point will fail
        _status = _ENTERED;

        _;

        // By storing the original value once again, a refund is triggered (see
        // https://eips.ethereum.org/EIPS/eip-2200)
        _status = _NOT_ENTERED;
    }
}
```

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