



# Armors Labs

FFA

## Smart Contract Audit

- FFA Token Audit Summary
- FFA Token Audit
  - Document information
    - Audit results
    - Audited target file
  - Vulnerability analysis
    - Vulnerability distribution
    - Summary of audit results
    - Contract file
    - Analysis of audit results
      - Re-Entrancy
      - Arithmetic Over/Under Flows
      - Unexpected Blockchain Currency
      - Delegatecall
      - Default Visibilities
      - Entropy Illusion
      - External Contract Referencing
      - Unsolved TODO comments
      - Short Address/Parameter Attack
      - Unchecked CALL Return Values
      - Race Conditions / Front Running
      - Denial Of Service (DOS)
      - Block Timestamp Manipulation
      - Constructors with Care
      - Unintialised Storage Pointers
      - Floating Points and Numerical Precision
      - tx.origin Authentication
      - Permission restrictions

# FFA Token Audit Summary

Project name : FFA Token Contract

Project address: <https://cryptofifa.com>

Code URL : <https://github.com/cryptofifa/contract>

Commit : c18c45b8c95a9b8a8729eb710a2c22a9becd5898

Project target : FFA Token Contract Audit

Blockchain : Binance Smart Chain (BSC)

Test result : PASSED

Audit Info

Audit NO : 0X202105100008

Audit Team : Armors Labs

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

## FFA Token Audit

The CryptoFifa team asked us to review and audit their FFA 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
FFA Token Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2021-05-10

### Audit results

Note:

This audit only includes CryptoFifaUtils.sol and CryptoFifaToken.sol. FFA is not only ERC20, but also includes authority management, suspension, blacklist, chain token and ERC20 token that the project party can transfer out of the contract address, Miner management, miner minting (with upper limit), and the project can only destroy the token.

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

(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
./CryptoFifaUtils.sol	ba34fd4148a7a08484873135cde98677
./CryptoFifaToken.sol	55626593a38c9a898faf60a724882ebd

## 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
Block Timestamp Manipulation	safe

Vulnerability	status
Constructors with Care	safe
Uninitialised Storage Pointers	safe
Floating Points and Numerical Precision	safe
tx.origin Authentication	safe
Permission restrictions	safe

## Contract file

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

pragma solidity ^0.8.0;

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * 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) {
        this; // silence state mutability warning without generating bytecode - see https://github.com
        return msg.data;
    }
}

/**
 * @dev Contract module which allows children to implement an emergency stop
 * mechanism that can be triggered by an authorized account.
 *
 * This module is used through inheritance. It will make available the
 * modifiers `whenNotPaused` and `whenPaused`, which can be applied to
 * the functions of your contract. Note that they will not be pausable by
 * simply including this module, only once the modifiers are put in place.
 */
abstract contract Pausable is Context {
    /**
     * @dev Emitted when the pause is triggered by `account`.
     */
    event Paused(address account);

    /**
     * @dev Emitted when the pause is lifted by `account`.
     */
    event Unpaused(address account);

    bool private _paused;

    /**
```

```

    * @dev Initializes the contract in unpaused state.
    */
    constructor () {
        _paused = false;
    }

    /**
     * @dev Returns true if the contract is paused, and false otherwise.
     */
    function paused() public view virtual returns (bool) {
        return _paused;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is not paused.
     *
     * Requirements:
     *
     * - The contract must not be paused.
     */
    modifier whenNotPaused() {
        require(!paused(), "Pausable: paused");
        _;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is paused.
     *
     * Requirements:
     *
     * - The contract must be paused.
     */
    modifier whenPaused() {
        require(paused(), "Pausable: not paused");
        _;
    }

    /**
     * @dev Triggers stopped state.
     *
     * Requirements:
     *
     * - The contract must not be paused.
     */
    function _pause() internal virtual whenNotPaused {
        _paused = true;
        emit Paused(_msgSender());
    }

    /**
     * @dev Returns to normal state.
     *
     * Requirements:
     *
     * - The contract must be paused.
     */
    function _unpause() internal virtual whenPaused {
        _paused = false;
        emit Unpaused(_msgSender());
    }
}

/// @title
/// @author
/// @dev
contract AccessControl is Pausable {

```

```

// The addresses of the accounts (or contracts) that can execute actions within each roles.
address public ceoAddress;
address public cfoAddress;
address public cooAddress;

/// @dev Access modifier for CEO-only functionality
modifier onlyCEO() {
    require(msg.sender == ceoAddress);
    _;
}

/// @dev Access modifier for CFO-only functionality
modifier onlyCFO() {
    require(msg.sender == cfoAddress);
    _;
}

/// @dev Access modifier for COO-only functionality
modifier onlyCOO() {
    require(msg.sender == cooAddress);
    _;
}

modifier onlyCLevel() {
    require(
        msg.sender == cooAddress ||
        msg.sender == ceoAddress ||
        msg.sender == cfoAddress
    );
    _;
}

/// @dev Assigns a new address to act as the CEO. Only available to the current CEO.
/// @param _newCEO The address of the new CEO
function setCEO(address _newCEO) external onlyCEO {
    require(_newCEO != address(0));

    ceoAddress = _newCEO;
}

/// @dev Assigns a new address to act as the CFO. Only available to the current CEO.
/// @param _newCFO The address of the new CFO
function setCFO(address _newCFO) external onlyCEO {
    require(_newCFO != address(0));

    cfoAddress = _newCFO;
}

/// @dev Assigns a new address to act as the COO. Only available to the current CEO.
/// @param _newCOO The address of the new COO
function setCOO(address _newCOO) external onlyCEO {
    require(_newCOO != address(0));

    cooAddress = _newCOO;
}

/// @dev Called by any "C-level" role to pause the contract. Used only when
/// a bug or exploit is detected and we need to limit damage.
function pause() public onlyCLevel {
    _pause();
}

/// @dev Unpauses the smart contract. Can only be called by the CEO, since

```



```

    /// one reason we may pause the contract is when CFO or COO accounts are
    /// compromised.
    /// @notice This is public rather than external so it can be called by
    /// derived contracts.
    function unpause() public onlyCEO {
        // can't unpause if contract was upgraded
        _unpause();
    }
}

/**
 * @dev Library for managing
 * https://en.wikipedia.org/wiki/Set_(abstract_data_type)[sets] of primitive
 * types.
 *
 * Sets have the following properties:
 *
 * - Elements are added, removed, and checked for existence in constant time
 *   (O(1)).
 * - Elements are enumerated in O(n). No guarantees are made on the ordering.
 *
 *
 * contract Example {
 *     // Add the library methods
 *     using EnumerableSet for EnumerableSet.AddressSet;
 *
 *     // Declare a set state variable
 *     EnumerableSet.AddressSet private mySet;
 * }
 *
 * As of v3.3.0, sets of type `bytes32` (`Bytes32Set`), `address` (`AddressSet`)
 * and `uint256` (`UintSet`) are supported.
 */
library EnumerableSet {
    // To implement this library for multiple types with as little code
    // repetition as possible, we write it in terms of a generic Set type with
    // bytes32 values.
    // The Set implementation uses private functions, and user-facing
    // implementations (such as AddressSet) are just wrappers around the
    // underlying Set.
    // This means that we can only create new EnumerableSets for types that fit
    // in bytes32.

    struct Set {
        // Storage of set values
        bytes32[] _values;

        // Position of the value in the `values` array, plus 1 because index 0
        // means a value is not in the set.
        mapping (bytes32 => uint256) _indexes;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function _add(Set storage set, bytes32 value) private returns (bool) {
        if (!_contains(set, value)) {
            set._values.push(value);
            // The value is stored at length-1, but we add 1 to all indexes
            // and use 0 as a sentinel value
            set._indexes[value] = set._values.length;
        }
    }

```



```

        return true;
    } else {
        return false;
    }
}

/**
 * @dev Removes a value from a set. O(1).
 *
 * Returns true if the value was removed from the set, that is if it was
 * present.
 */
function _remove(Set storage set, bytes32 value) private returns (bool) {
    // We read and store the value's index to prevent multiple reads from the same storage slot
    uint256 valueIndex = set._indexes[value];

    if (valueIndex != 0) { // Equivalent to contains(set, value)
        // To delete an element from the _values array in O(1), we swap the element to delete with
        // the last element, and then remove the last element (sometimes called as 'swap and pop').
        // This modifies the order of the array, as noted in {at}.

        uint256 toDeleteIndex = valueIndex - 1;
        uint256 lastIndex = set._values.length - 1;

        // When the value to delete is the last one, the swap operation is unnecessary. However,
        // so rarely, we still do the swap anyway to avoid the gas cost of adding an 'if' statement

        bytes32 lastvalue = set._values[lastIndex];

        // Move the last value to the index where the value to delete is
        set._values[toDeleteIndex] = lastvalue;
        // Update the index for the moved value
        set._indexes[lastvalue] = toDeleteIndex + 1; // All indexes are 1-based

        // Delete the slot where the moved value was stored
        set._values.pop();

        // Delete the index for the deleted slot
        delete set._indexes[value];

        return true;
    } else {
        return false;
    }
}

/**
 * @dev Returns true if the value is in the set. O(1).
 */
function _contains(Set storage set, bytes32 value) private view returns (bool) {
    return set._indexes[value] != 0;
}

/**
 * @dev Returns the number of values on the set. O(1).
 */
function _length(Set storage set) private view returns (uint256) {
    return set._values.length;
}

/**
 * @dev Returns the value stored at position `index` in the set. O(1).
 *
 * Note that there are no guarantees on the ordering of values inside the
 * array, and it may change when more values are added or removed.
 */

```

```

* Requirements:
*
* - `index` must be strictly less than {length}.
*/
function _at(Set storage set, uint256 index) private view returns (bytes32) {
    require(set._values.length > index, "EnumerableSet: index out of bounds");
    return set._values[index];
}

// Bytes32Set

struct Bytes32Set {
    Set _inner;
}

/**
 * @dev Add a value to a set. O(1).
 *
 * Returns true if the value was added to the set, that is if it was not
 * already present.
 */
function add(Bytes32Set storage set, bytes32 value) internal returns (bool) {
    return _add(set._inner, value);
}

/**
 * @dev Removes a value from a set. O(1).
 *
 * Returns true if the value was removed from the set, that is if it was
 * present.
 */
function remove(Bytes32Set storage set, bytes32 value) internal returns (bool) {
    return _remove(set._inner, value);
}

/**
 * @dev Returns true if the value is in the set. O(1).
 */
function contains(Bytes32Set storage set, bytes32 value) internal view returns (bool) {
    return _contains(set._inner, value);
}

/**
 * @dev Returns the number of values in the set. O(1).
 */
function length(Bytes32Set storage set) internal view returns (uint256) {
    return _length(set._inner);
}

/**
 * @dev Returns the value stored at position `index` in the set. O(1).
 *
 * Note that there are no guarantees on the ordering of values inside the
 * array, and it may change when more values are added or removed.
 *
 * Requirements:
 *
 * - `index` must be strictly less than {length}.
 */
function at(Bytes32Set storage set, uint256 index) internal view returns (bytes32) {
    return _at(set._inner, index);
}

// AddressSet

struct AddressSet {

```

```

        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(AddressSet storage set, address value) internal returns (bool) {
        return _add(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(AddressSet storage set, address value) internal returns (bool) {
        return _remove(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(AddressSet storage set, address value) internal view returns (bool) {
        return _contains(set._inner, bytes32(uint256(uint160(value))));
    }

    /**
     * @dev Returns the number of values in the set. O(1).
     */
    function length(AddressSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

    /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function at(AddressSet storage set, uint256 index) internal view returns (address) {
        return address(uint160(uint256(_at(set._inner, index))));
    }

    // UIntSet

    struct UIntSet {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(UIntSet storage set, uint256 value) internal returns (bool) {
        return _add(set._inner, bytes32(value));
    }

```

```

}

/**
 * @dev Removes a value from a set. O(1).
 *
 * Returns true if the value was removed from the set, that is if it was
 * present.
 */
function remove(UintSet storage set, uint256 value) internal returns (bool) {
    return _remove(set._inner, bytes32(value));
}

/**
 * @dev Returns true if the value is in the set. O(1).
 */
function contains(UintSet storage set, uint256 value) internal view returns (bool) {
    return _contains(set._inner, bytes32(value));
}

/**
 * @dev Returns the number of values on the set. O(1).
 */
function length(UintSet storage set) internal view returns (uint256) {
    return _length(set._inner);
}

/**
 * @dev Returns the value stored at position `index` in the set. O(1).
 *
 * Note that there are no guarantees on the ordering of values inside the
 * array, and it may change when more values are added or removed.
 *
 * Requirements:
 *
 * - `index` must be strictly less than {length}.
 */
function at(UintSet storage set, uint256 index) internal view returns (uint256) {
    return uint256(_at(set._inner, index));
}
}

/**
 * @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 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 ^0.8.0;

import "./CryptoFifaUtils.sol";

contract Blockable is AccessControl {
    using EnumerableSet for EnumerableSet.AddressSet;
    EnumerableSet.AddressSet private _blocklist;

    function getBlocklistLength() public view returns (uint256) {
        return EnumerableSet.length(_blocklist);
    }
}

```

```

function isInBlocklist(address account) public view returns (bool) {
    if (EnumerableSet.length(_blocklist) == 0) {
        return false;
    }
    return EnumerableSet.contains(_blocklist, account);
}

function getBlocklist(uint256 _index) public view onlyCLevel returns (address){
    require(_index <= getBlocklistLength() - 1, "CryptoFifa: index out of bounds");
    return EnumerableSet.at(_blocklist, _index);
}

function addBlocklist(address _addBlocklist) public onlyCLevel returns (bool) {
    require(_addBlocklist != address(0), "CryptoFifa: _addBlocklist is the zero address");
    return EnumerableSet.add(_blocklist, _addBlocklist);
}

function delBlocklist(address _delBlocklist) public onlyCEO returns (bool) {
    require(_delBlocklist != address(0), "CryptoFifa: _delBlocklist is the zero address");
    return EnumerableSet.remove(_blocklist, _delBlocklist);
}

// modifier for blacklist function
modifier notInBlocklist() {
    require(!isInBlocklist(_msgSender()), "caller is not the blocklist");
    _;
}

}

/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * 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
 * to implement supply mechanisms.
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of 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 just
 * 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 IERC20, Blockable {
    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 three of these values are immutable: they can only be set once during
* construction.
*/
constructor (string memory name_, string memory symbol_) {
    _name = name_;
    _symbol = symbol_;
}

/**
 * @dev Returns the name of the token.
 */
function name() public view virtual returns (string memory) {
    return _name;
}

/**
 * @dev Returns the symbol of the token, usually a shorter version of the
 * name.
 */
function symbol() public view virtual 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 should
 * be displayed to a user as `5,05` (`505 / 10 ** 2`).
 *
 * Tokens usually opt for a value of 18, imitating the relationship between
 * Ether and Wei. This is the value {ERC20} uses, unless this function is
 * overloaded;
 *
 * NOTE: This information is only used for _display_ purposes: it in
 * no way affects any of the arithmetic of the contract, including
 * {IERC20-balanceOf} and {IERC20-transfer}.
 */
function decimals() public view virtual 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 whenNotPaused notInB
    _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 whenNotPaused notInBloc
    _approve(_msgSender(), spender, amount);
    return true;
}

/**
 * @dev See {IERC20-transferFrom}.
 *
 * Emits an {Approval} event indicating the updated allowance. This is not
 * required by the EIP. See the note at the beginning of {IERC20}.
 *
 * 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
    _transfer(sender, recipient, amount);

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

    return true;
}

/**
 * @dev Atomically increases the allowance granted to `spender` by the caller.
 *
 * This is an alternative to {approve} that can be used as a mitigation for
 * problems described in {IERC20-approve}.
 *
 * Emits an {Approval} event indicating the updated allowance.
 *
 * Requirements:
 *
 * - `spender` cannot be the zero address.
 */
function increaseAllowance(address spender, uint256 addedValue) public virtual whenNotPaused notI
    _approve(_msgSender(), spender, _allowances[_msgSender()][spender] + addedValue);
    return true;
}

/**
 * @dev Atomically decreases the allowance granted to `spender` by the caller.

```

```

*
* This is an alternative to {approve} that can be used as a mitigation for
* problems described in {IERC20-approve}.
*
* Emits an {Approval} event indicating the updated allowance.
*
* 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 whenNotPaused
    uint256 currentAllowance = _allowances[_msgSender()][spender];
    require(currentAllowance >= subtractedValue, "ERC20: decreased allowance below zero");
    _approve(_msgSender(), spender, currentAllowance - subtractedValue);

    return true;
}

/**
* @dev Moves tokens `amount` from `sender` to `recipient`.
*
* This is 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");
    _balances[sender] = senderBalance - amount;
    _balances[recipient] += amount;

    emit Transfer(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 address.
*
* Requirements:
*
* - `to` 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);
}

```

```

/**
 * @dev Destroys `amount` tokens from `account`, reducing the
 * total supply.
 *
 * Emits a {Transfer} event with `to` set to the zero address.
 *
 * 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");
    _balances[account] = accountBalance - amount;
    _totalSupply -= amount;

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

/**
 * @dev Sets `amount` as the allowance of `spender` over the `owner`'s tokens.
 *
 * 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-hooks [Using Hooks]
 */
function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }

}

/**
 * @title ReclaimerToken

```

```

* @dev ERC20 token which allows owner to reclaim ERC20 tokens
* or ether sent to this contract
*/
abstract contract ReclaimerToken is ERC20 {
    /**
     * @dev send all eth balance in the contract to another address
     * @param _to address to send eth balance to
     */
    function reclaimEther(address payable _to) external onlyCLevel {
        _to.transfer(address(this).balance);
    }

    /**
     * @dev send all token balance of an arbitrary ERC20 token
     * in the contract to another address
     * @param token token to reclaim
     * @param _to address to send eth balance to
     */
    function reclaimToken(IERC20 token, address _to) external onlyCLevel {
        uint256 balance = token.balanceOf(address(this));
        token.transfer(_to, balance);
    }
}

contract CryptoFifaToken is ReclaimerToken {
    uint256 private constant preMineSupply = 10000000000 * 1e18;
    uint256 private constant maxSupply = 10000000000 * 1e18; // the total supply

    using EnumerableSet for EnumerableSet.AddressSet;
    EnumerableSet.AddressSet private _minters;

    constructor() ERC20("CryptoFifa Token", "FFA"){
        // the creator of the contract is the initial CEO
        ceoAddress = _msgSender();
        cooAddress = _msgSender();
        cfoAddress = _msgSender();
        _mint(_msgSender(), preMineSupply);
        EnumerableSet.add(_minters, _msgSender());
    }

    // mint with max supply
    function mint(address _to, uint256 _amount) public onlyMinter returns (bool) {
        if ((_amount+totalSupply()) > maxSupply) {
            return false;
        }
        _mint(_to, _amount);
        return true;
    }

    function addMinter(address _addMinter) public onlyCEO returns (bool) {
        require(_addMinter != address(0), "CryptoFifa: _addMinter is the zero address");
        require(!isInBlocklist(_addMinter), "minter is not the blocklist");
        return EnumerableSet.add(_minters, _addMinter);
    }

    function delMinter(address _delMinter) public onlyCLevel returns (bool) {
        require(_delMinter != address(0), "CryptoFifa: _delMinter is the zero address");
        return EnumerableSet.remove(_minters, _delMinter);
    }

    function getMinterLength() public view returns (uint256) {
        return EnumerableSet.length(_minters);
    }

    function isMinter(address account) public view returns (bool) {

```

```

    return EnumerableSet.contains(_minters, account);
}

function getMinter(uint256 _index) public view onlyCLevel returns (address){
    require(_index <= getMinterLength() - 1, "CryptoFifa: index out of bounds");
    return EnumerableSet.at(_minters, _index);
}

// modifier for mint function
modifier onlyMinter() {
    require(isMinter(_msgSender()), "caller is not the minter");
    _;
}

function burn(address account, uint256 amount) public onlyCEO {
    ERC20._burn(account, 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 whether 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 devastating vulnerabilities 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 are 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 (initialised 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 initialising 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 Labs

The background is a dark teal color with a complex, layered geometric pattern. In the center, there is a 3D cube with a blue base and a teal top. Above the cube is a transparent teal rectangular prism. The background is filled with floating binary code (0s and 1s) and two large, stylized teal shields on the left and right sides. The shields have a grid-like pattern and are slightly tilted.

[armors.io](https://armors.io)

[contact@armors.io](mailto:contact@armors.io)

