Armors Labs

Revenant (GAMEFI) Token

Smart Contract Audit

Revenant (GAMEFI) Token Audit Summary

Project name : Revenant (GAMEFI) Token Contract

Project address: None

Code URL : https://andromeda-explorer.metis.io/address/0x966B25D174bE6Ba703E0fF80c68BB6e167236BA2

Code URL : https://andromeda-explorer.metis.io/address/0xAD80144d19C48570Af8091D90dDaD7Cc269AEbE9

Commit : None

Project target : Revenant (GAMEFI) Token Contract Audit

Blockchain : Metis

Test result : PASSED

Audit Info

Audit NO : 0X202205170026

Audit Team : Armors Labs

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

Revenant (GAMEFI) Token Audit

The Revenant (GAMEFI) Token team asked us to review and audit their Revenant (GAMEFI) 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
Revenant (GAMEFI) Token Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2022-05-17

Audit results

Notices

```
1.Owner can transfer the token and platform currency in the contract.
2. The initial dividendTracker address of the contract
  is 0xad80144d19c48570af8091d90ddad7cc269aebe9.
```

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

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

Disclaimer

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

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

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

Audited target file

file	md5
Revenant (GAMEFI) Token.sol	b8f99b3d2cca6da462cda389ef929687

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

Vulnerability	status
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

```
// Sources flattened with hardhat v2.9.3 https://hardhat.org

// File @openzeppelin/contracts/utils/Context.sol@v4.5.0

// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts v4.4.1 (utils/Context.sol)

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) {
        return msg.data;
    }
}


// File @openzeppelin/contracts/access/Ownable.sol@v4.5.0


// OpenZeppelin Contracts v4.4.1 (access/Ownable.sol)

pragma solidity ^0.8.0;

/**
```

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

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor() {
        _transferOwnership(_msgSender());
    }

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

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

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public virtual onlyOwner {
        _transferOwnership(address(0));
    }

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

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Internal function without access restriction.
     */
    function _transferOwnership(address newOwner) internal virtual {
        address oldOwner = _owner;
        _owner = newOwner;
        emit OwnershipTransferred(oldOwner, newOwner);
    }
}


// File @openzeppelin/contracts/token/ERC20/IERC20.sol@v4.5.0


// OpenZeppelin Contracts (last updated v4.5.0) (token/ERC20/IERC20.sol)

pragma solidity ^0.8.0;

/**
 * @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 `to`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address to, 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 `from` to `to` 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 from,
        address to,
        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);
}


// File @openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol@v4.5.0


// OpenZeppelin Contracts v4.4.1 (token/ERC20/extensions/IERC20Metadata.sol)

pragma solidity ^0.8.0;

/**
 * @dev Interface for the optional metadata functions from the ERC20 standard.
 *
 * _Available since v4.1._
 */
interface IERC20Metadata is IERC20 {
    /**
     * @dev Returns the name of the token.
     */
    function name() external view returns (string memory);

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

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


// File @openzeppelin/contracts/token/ERC20/ERC20.sol@v4.5.0


// OpenZeppelin Contracts (last updated v4.5.0) (token/ERC20/ERC20.sol)

pragma solidity ^0.8.0;



/**
 * @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[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin Contracts guidelines: functions revert
 * instead returning `false` on failure. This behavior is nonetheless
 * conventional and does not conflict with the expectations of ERC20
 * applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts 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 Context, IERC20, IERC20Metadata {
    mapping(address => uint256) private _balances;

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

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;

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

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

    /**
     * @dev Returns the number of decimals used to get its user representation.
     * For example, if `decimals` equals `2`, a balance of `505` tokens 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
     * overridden;
     *
     * 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 override returns (uint8) {
        return 18;
    }

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

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

    /**
     * @dev See {IERC20-transfer}.
     *
     * Requirements:
     *
     * - `to` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    function transfer(address to, uint256 amount) public virtual override returns (bool) {
        address owner = _msgSender();
        _transfer(owner, to, 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}.
     *
     * NOTE: If `amount` is the maximum `uint256`, the allowance is not updated on
     * `transferFrom`. This is semantically equivalent to an infinite approval.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        address owner = _msgSender();
        _approve(owner, 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 {ERC20}.
     *
     * NOTE: Does not update the allowance if the current allowance
     * is the maximum `uint256`.
     *
     * Requirements:
     *
     * - `from` and `to` cannot be the zero address.
     * - `from` must have a balance of at least `amount`.
     * - the caller must have allowance for ``from``'s tokens of at least
     * `amount`.
     */
    function transferFrom(
        address from,
        address to,
        uint256 amount
    ) public virtual override returns (bool) {
        address spender = _msgSender();
        _spendAllowance(from, spender, amount);
        _transfer(from, to, 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 returns (bool) {
        address owner = _msgSender();
        _approve(owner, spender, _allowances[owner][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 returns (bool
        address owner = _msgSender();
        uint256 currentAllowance = _allowances[owner][spender];
        require(currentAllowance >= subtractedValue, "ERC20: decreased allowance below zero");
    unchecked {
        _approve(owner, spender, currentAllowance - subtractedValue);
    }

        return true;
    }
```

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

    _beforeTokenTransfer(from, to, amount);

    uint256 fromBalance = _balances[from];
    require(fromBalance >= amount, "ERC20: transfer amount exceeds balance");
unchecked {
    _balances[from] = fromBalance - amount;
}
    _balances[to] += amount;

    emit Transfer(from, to, amount);

    _afterTokenTransfer(from, to, 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:
 *
 * - `account` cannot be the zero address.
 */
function _mint(address account, uint256 amount) internal virtual {
    require(account != address(0), "ERC20: mint to the zero address");

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

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

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

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

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

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

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `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 Spend `amount` form the allowance of `owner` toward `spender`.
     *
     * Does not update the allowance amount in case of infinite allowance.
     * Revert if not enough allowance is available.
     *
     * Might emit an {Approval} event.
     */
    function _spendAllowance(
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        uint256 currentAllowance = allowance(owner, spender);
        if (currentAllowance != type(uint256).max) {
            require(currentAllowance >= amount, "ERC20: insufficient allowance");
        unchecked {
            _approve(owner, spender, currentAllowance - 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 {}

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


// File @openzeppelin/contracts/utils/Address.sol@v4.5.0


// OpenZeppelin Contracts (last updated v4.5.0) (utils/Address.sol)

pragma solidity ^0.8.1;

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     *
     * [IMPORTANT]
     * ====
     * You shouldn't rely on `isContract` to protect against flash loan attacks!
     *
     * Preventing calls from contracts is highly discouraged. It breaks composability, breaks support
     * like Gnosis Safe, and does not provide security since it can be circumvented by calling from a
     * constructor.
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies on extcodesize/address.code.length, which returns 0
        // for contracts in construction, since the code is only stored at the end
        // of the constructor execution.

        return account.code.length > 0;
    }

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

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

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain `call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.de
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
        return functionCall(target, data, "Address: low-level call failed");
    }
```

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

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

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

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

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

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

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

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

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

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

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

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


// File @openzeppelin/contracts/utils/math/SafeMath.sol@v4.5.0


// OpenZeppelin Contracts v4.4.1 (utils/math/SafeMath.sol)

pragma solidity ^0.8.0;

// CAUTION
// This version of SafeMath should only be used with Solidity 0.8 or later,
// because it relies on the compiler's built in overflow checks.

/**
 * @dev Wrappers over Solidity's arithmetic operations.
 *
 * NOTE: `SafeMath` is generally not needed starting with Solidity 0.8, since the compiler
 * now has built in overflow checking.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryAdd(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    unchecked {
        uint256 c = a + b;
        if (c < a) return (false, 0);
        return (true, c);
    }
    }

    /**
     * @dev Returns the substraction of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function trySub(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    unchecked {
        if (b > a) return (false, 0);
        return (true, a - b);
    }
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, with an overflow flag.
     *
     * _Available since v3.4._
     */
    function tryMul(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    unchecked {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) return (true, 0);
        uint256 c = a * b;
        if (c / a != b) return (false, 0);
        return (true, c);
    }
    }

    /**
     * @dev Returns the division of two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryDiv(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    unchecked {
        if (b == 0) return (false, 0);
        return (true, a / b);
    }
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers, with a division by zero flag.
     *
     * _Available since v3.4._
     */
    function tryMod(uint256 a, uint256 b) internal pure returns (bool, uint256) {
    unchecked {
        if (b == 0) return (false, 0);
        return (true, a % b);
    }
    }

    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        return a + b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return a - b;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        return a * b;
    }

    /**
```

```
     * @dev Returns the integer division of two unsigned integers, reverting on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator.
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return a / b;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return a % b;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {trySub}.
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
    unchecked {
        require(b <= a, errorMessage);
        return a - b;
    }
    }

    /**
     * @dev Returns the integer division of two unsigned integers, reverting with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
    unchecked {
        require(b > 0, errorMessage);
        return a / b;
    }
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * reverting with custom message when dividing by zero.
     *
     * CAUTION: This function is deprecated because it requires allocating memory for the error
     * message unnecessarily. For custom revert reasons use {tryMod}.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
    unchecked {
        require(b > 0, errorMessage);
        return a % b;
    }
    }
}


// File contracts/Revenant.sol


pragma solidity ^0.8.11;
pragma experimental ABIEncoderV2;




contract Revenant is Ownable, IERC20 {


    address public dead = 0x000000000000000000000000000000000000dEaD;

    string private _name = "Revenant";
    string private _symbol = "GAMEFI";


    uint256 private _totalSupply;


    mapping(address => uint256) private _balances;
    mapping(address => mapping(address => uint256)) private _allowances;
    mapping(address => bool) private _isExcludedFromFees;

    event SendDividends(uint256 tokensSwapped, uint256 amount);
    event BlacklistEnabled(bool enabled);
    DividendTracker public dividendTracker;

    bool public isOpen = false;

    constructor() {

        dividendTracker = new DividendTracker(address(this));

        dividendTracker.excludeFromDividends(address(dividendTracker), true);
        dividendTracker.excludeFromDividends(address(this), true);
        dividendTracker.excludeFromDividends(owner(), true);

        _mint(owner(), 3000000 * (10**18));
    }

    receive() external payable {}


    function changeDividendTracker(address payable _contractAddress) public onlyOwner{
        dividendTracker = DividendTracker(_contractAddress);
    }

    function name() public view returns (string memory) {
        return _name;
    }

    function symbol() public view returns (string memory) {
        return _symbol;
    }

    function decimals() public pure returns (uint8) {
        return 18;
    }

    function totalSupply() public view virtual override returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address account)
    public
    view
    virtual
    override
    returns (uint256)
    {
        return _balances[account];
    }

    function allowance(address owner, address spender)
    public
    view
    virtual
    override
    returns (uint256)
    {
        return _allowances[owner][spender];
    }

    function increaseAllowance(address spender, uint256 addedValue)
    public
    returns (bool)
    {
        _approve(
            _msgSender(),
            spender,
            _allowances[_msgSender()][spender] + addedValue
        );
        return true;
    }

    function decreaseAllowance(address spender, uint256 subtractedValue)
    public
    returns (bool)
    {
        uint256 currentAllowance = _allowances[_msgSender()][spender];
        require(
            currentAllowance >= subtractedValue,
            "Decreased allowance below zero"
        );
        _approve(_msgSender(), spender, currentAllowance - subtractedValue);
        return true;
    }

    function approve(address spender, uint256 amount)
    public
    virtual
    override
    returns (bool)
    {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    function transfer(address recipient, uint256 amount)
    public
    virtual
    override
    returns (bool)
    {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) public virtual override returns (bool) {
        _transfer(sender, recipient, amount);

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

        return true;
    }

    function openTrading() external onlyOwner {
        isOpen = true;
    }

    function _transfer(
        address sender,
        address recipient,
        uint256 amount
    ) internal {
        require(
            isOpen ||
            sender == owner() ||
            recipient == owner(),
            "Not Open"
        );


        require(sender != address(0), "Transfer from the 0 address");
        require(recipient != address(0), "Transfer to the 0 address");

        uint256 senderBalance = _balances[sender];
        require(
            senderBalance >= amount,
            "Transfer amount exceeds balance"
        );

        _executeTransfer(sender, recipient, amount);

        dividendTracker.setBalance(payable(sender), balanceOf(sender));
        dividendTracker.setBalance(payable(recipient), balanceOf(recipient));
    }

    function _executeTransfer(
        address sender,
        address recipient,
        uint256 amount
    ) private {
        require(sender != address(0), "Transfer from the 0 address");
        require(recipient != address(0), "Transfer to the 0 address");
        uint256 senderBalance = _balances[sender];
        require(
            senderBalance >= amount,
            "Transfer amount exceeds balance"
        );
        _balances[sender] = senderBalance - amount;
        _balances[recipient] += amount;
        emit Transfer(sender, recipient, amount);
    }

    function _approve(
        address owner,
        address spender,
        uint256 amount
    ) private {
        require(owner != address(0), "Approve from the 0 address");
        require(spender != address(0), "Approve to the 0 address");
        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }

    function _mint(address account, uint256 amount) private {
        require(account != address(0), "Mint to the 0 address");
        _totalSupply += amount;
        _balances[account] += amount;
        emit Transfer(address(0), account, amount);
    }

    function _burn(address account, uint256 amount) private {
        require(account != address(0), "Burn from the 0 address");
        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "Burn amount exceeds balance");
        _balances[account] = accountBalance - amount;
        _totalSupply -= amount;
        emit Transfer(account, address(0), amount);
    }

    function manualSendDividend(uint256 amount, address holder)
    external
    onlyOwner
    {
        dividendTracker.manualSendDividend(amount, holder);
    }

    function excludeFromDividends(address account, bool excluded)
    public
    onlyOwner
    {
        dividendTracker.excludeFromDividends(account, excluded);
    }

    function isExcludedFromDividends(address account)
    public
    view
    returns (bool)
    {
        return dividendTracker.isExcludedFromDividends(account);
    }

    function claim() public {
        dividendTracker.processAccount(payable(_msgSender()));
    }

    function withdrawableDividendOf(address account)
    public
    view
    returns (uint256)
    {
        return dividendTracker.withdrawableDividendOf(account);
    }

    function withdrawnDividendOf(address account)
    public
    view
    returns (uint256)
    {
        return dividendTracker.withdrawnDividendOf(account);
    }

    function accumulativeDividendOf(address account)
    public
    view
    returns (uint256)
    {
        return dividendTracker.accumulativeDividendOf(account);
    }

    function getAccountInfo(address account)
    public
    view
    returns (
        address,
        uint256,
        uint256,
        uint256,
        uint256
    )
    {
        return dividendTracker.getAccountInfo(account);
    }

    function getLastClaimTime(address account) public view returns (uint256) {
        return dividendTracker.getLastClaimTime(account);
    }
    function rescueToken(address _token, uint256 _amount) external onlyOwner {
        IERC20(_token).transfer(msg.sender, _amount);
    }

    function rescueMetis(uint256 _amount) external onlyOwner {
        payable(msg.sender).transfer(_amount);
    }

}

contract DividendTracker is Ownable, IERC20 {


    string private _name = "GAMEFI_DividendTracker";
    string private _symbol = "GAMEFI_DividendTracker";

    uint256 public lastProcessedIndex;

    uint256 private _totalSupply;
    mapping(address => uint256) private _balances;

    uint256 private constant MAGNITUDE = 2**128;
    uint256 public immutable minTokenBalanceForDividends;
    uint256 private magnifiedDividendPerShare;
    uint256 public totalDividendsDistributed;
    uint256 public totalDividendsWithdrawn;

    address public tokenAddress;

    mapping(address => bool) public excludedFromDividends;
    mapping(address => int256) private magnifiedDividendCorrections;
    mapping(address => uint256) private withdrawnDividends;
    mapping(address => uint256) private lastClaimTimes;

    event DividendsDistributed(address indexed from, uint256 weiAmount);
    event DividendWithdrawn(address indexed to, uint256 weiAmount);
    event ExcludeFromDividends(address indexed account, bool excluded);
    event Claim(address indexed account, uint256 amount);


    struct AccountInfo {
        address account;
        uint256 withdrawableDividends;
        uint256 totalDividends;
        uint256 lastClaimTime;
    }

    constructor(address _tokenAddress) {
        minTokenBalanceForDividends = 10 * (10**18);
        tokenAddress = _tokenAddress;
    }

    receive() external payable {
        distributeDividends();
    }

    function distributeDividends() public payable{
        require(_totalSupply > 0, "Total supply must be > than 0");
        if (msg.value > 0) {
            magnifiedDividendPerShare =
            magnifiedDividendPerShare +
            ((msg.value * MAGNITUDE) / _totalSupply);
            emit DividendsDistributed(msg.sender, msg.value);
            totalDividendsDistributed += msg.value;
        }
    }
```

```
function setBalance(address payable account, uint256 newBalance)
external
onlyOwner
{
    if (excludedFromDividends[account]) {
        return;
    }
    if (newBalance >= minTokenBalanceForDividends) {
        _setBalance(account, newBalance);
    } else {
        _setBalance(account, 0);
    }
}

function excludeFromDividends(address account, bool excluded)
external
onlyOwner
{
    require(
        excludedFromDividends[account] != excluded,
        "Account already excluded"
    );
    excludedFromDividends[account] = excluded;
    if (excluded) {
        _setBalance(account, 0);
    } else {
        uint256 newBalance = IERC20(tokenAddress).balanceOf(account);
        if (newBalance >= minTokenBalanceForDividends) {
            _setBalance(account, newBalance);
        } else {
            _setBalance(account, 0);
        }
    }
    emit ExcludeFromDividends(account, excluded);
}

function isExcludedFromDividends(address account)
public
view
returns (bool)
{
    return excludedFromDividends[account];
}


function manualSendDividend(uint256 amount, address holder)
external
onlyOwner
{
    uint256 contractBalance = address(this).balance;
    payable(holder).transfer(amount > 0 ? amount : contractBalance);
}

function _setBalance(address account, uint256 newBalance) internal {
    uint256 currentBalance = _balances[account];
    if (newBalance > currentBalance) {
        uint256 addAmount = newBalance - currentBalance;
        _mint(account, addAmount);
    } else if (newBalance < currentBalance) {
        uint256 subAmount = currentBalance - newBalance;
        _burn(account, subAmount);
    }
}

function _mint(address account, uint256 amount) private {
    require(
        account != address(0),
        "Mint to the zero address"
    );
    _totalSupply += amount;
    _balances[account] += amount;
    emit Transfer(address(0), account, amount);
    magnifiedDividendCorrections[account] =
    magnifiedDividendCorrections[account] -
    int256(magnifiedDividendPerShare * amount);
}

function _burn(address account, uint256 amount) private {
    require(
        account != address(0),
        "Burn from the zero address"
    );
    uint256 accountBalance = _balances[account];
    require(
        accountBalance >= amount,
        "Burn amount exceeds balance"
    );
    _balances[account] = accountBalance - amount;
    _totalSupply -= amount;
    emit Transfer(account, address(0), amount);
    magnifiedDividendCorrections[account] =
    magnifiedDividendCorrections[account] +
    int256(magnifiedDividendPerShare * amount);
}

function processAccount(address payable account)
public
onlyOwner
returns (bool)
{
    uint256 amount = _withdrawDividendOfUser(account);
    if (amount > 0) {
        lastClaimTimes[account] = block.timestamp;
        emit Claim(account, amount);
        return true;
    }
    return false;
}

function _withdrawDividendOfUser(address payable account)
private
returns (uint256)
{
    uint256 _withdrawableDividend = withdrawableDividendOf(account);
    if (_withdrawableDividend > 0) {
        withdrawnDividends[account] += _withdrawableDividend;
        totalDividendsWithdrawn += _withdrawableDividend;
        emit DividendWithdrawn(account, _withdrawableDividend);
        (bool success, ) = account.call{
        value: _withdrawableDividend,
        gas: 3000
        }("");
        if (!success) {
            withdrawnDividends[account] -= _withdrawableDividend;
            totalDividendsWithdrawn -= _withdrawableDividend;
            return 0;
        }
        return _withdrawableDividend;
    }
    return 0;
}
```

```
function withdrawableDividendOf(address account)
public
view
returns (uint256)
{
    return accumulativeDividendOf(account) - withdrawnDividends[account];
}

function withdrawnDividendOf(address account)
public
view
returns (uint256)
{
    return withdrawnDividends[account];
}

function accumulativeDividendOf(address account)
public
view
returns (uint256)
{
    int256 a = int256(magnifiedDividendPerShare * balanceOf(account));
    int256 b = magnifiedDividendCorrections[account]; // this is an explicit int256 (signed)
    return uint256(a + b) / MAGNITUDE;
}

function getAccountInfo(address account)
public
view
returns (
    address,
    uint256,
    uint256,
    uint256,
    uint256
)
{
    AccountInfo memory info;
    info.account = account;
    info.withdrawableDividends = withdrawableDividendOf(account);
    info.totalDividends = accumulativeDividendOf(account);
    info.lastClaimTime = lastClaimTimes[account];
    return (
    info.account,
    info.withdrawableDividends,
    info.totalDividends,
    info.lastClaimTime,
    totalDividendsWithdrawn
    );
}

function getLastClaimTime(address account) public view returns (uint256) {
    return lastClaimTimes[account];
}

function name() public view returns (string memory) {
    return _name;
}

function symbol() public view returns (string memory) {
    return _symbol;
}

function decimals() public pure returns (uint8) {
    return 18;
    }

    function totalSupply() public view override returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address account) public view override returns (uint256) {
        return _balances[account];
    }

    function transfer(address, uint256) public pure override returns (bool) {
        revert("Method not implemented");
    }

    function allowance(address, address)
    public
    pure
    override
    returns (uint256)
    {
        revert("Method not implemented");
    }

    function approve(address, uint256) public pure override returns (bool) {
        revert("Method not implemented");
    }

    function transferFrom(
        address,
        address,
        uint256
    ) public pure override returns (bool) {
        revert("Method not implemented");
    }

    function setTokenAddress (address _tokenAddress)
    external
    onlyOwner
    {
        tokenAddress = _tokenAddress;
    }

    function getTokenAddress() public view returns (address) {
        return tokenAddress;
    }
}
```

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!
```

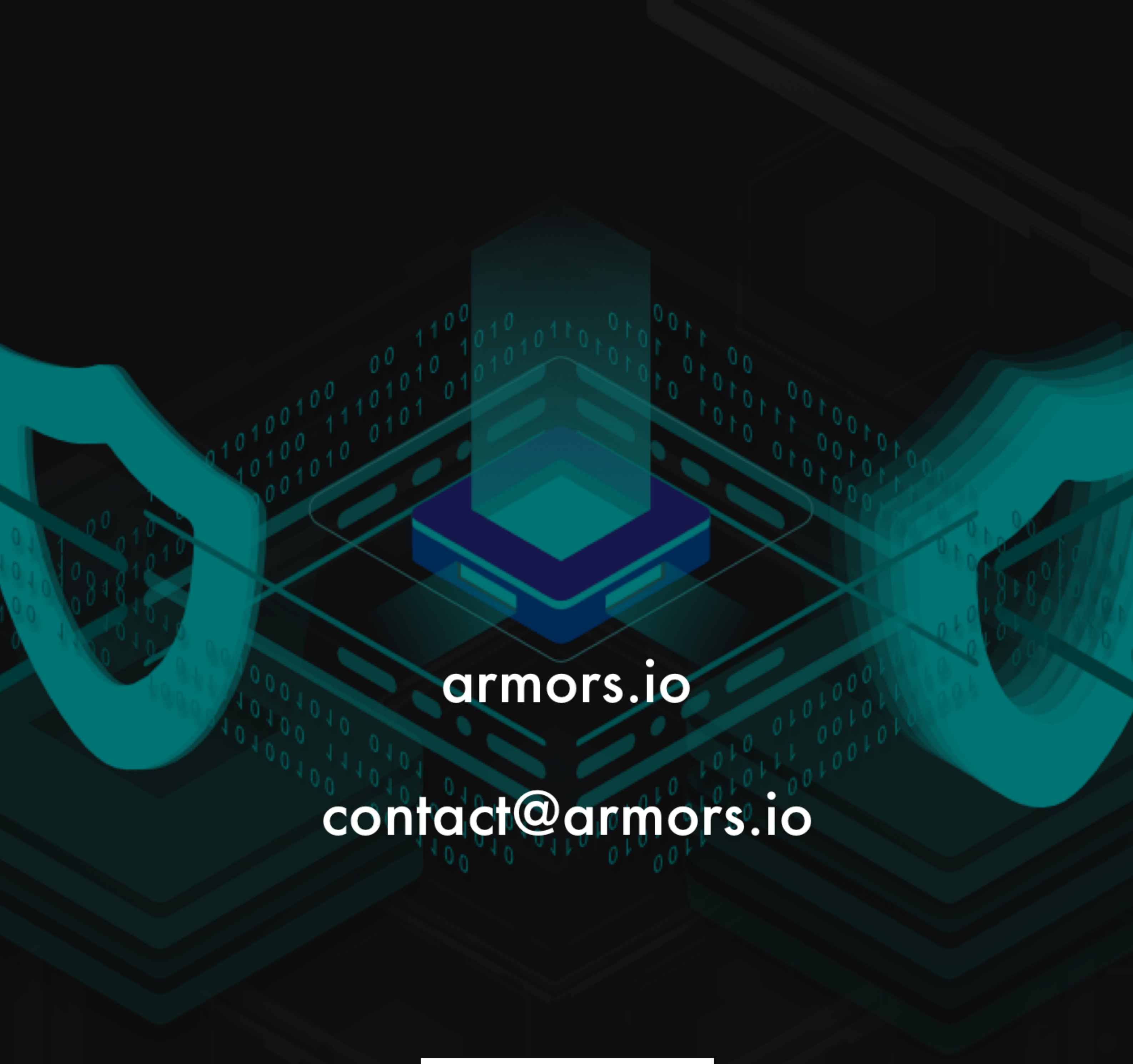
- **Security suggestion:**
  no.

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