Smart Contract Security Audit Report

The SlowMist Security Team received the AXL team's application for smart contract security audit of the AXiaL Entertainment Digital Asset on November 25, 2019. The following are the details and results of this smart contract security audit:

Token name :

AXL

File name and HASH(SHA256) :

AXL_UDIA_newDeploy_191123.zip:

9047b061103aa4cf3ee6b435e1c6aa28e99f64e2ad98a51fdc6a67e564b4d929

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

No.	Audit Items	Audit Subclass	Audit Subclass Result
1	Overflow Audit	-	Passed
2	Race Conditions Audit	-	Passed
3	Authority Control Audit	Permission vulnerability audit	Passed
		Excessive auditing authority	Passed
4	Safety Design Audit	Zeppelin module safe use	Passed
		Compiler version security	Passed
		Hard-coded address security	Passed
		Fallback function safe use	Passed
		Show coding security	Passed
		Function return value security	Passed
		Call function security	Passed
5	Denial of Service Audit	-	Passed
6	Gas Optimization Audit	-	Passed
7	Design Logic Audit	-	Passed
8	“False Deposit” vulnerability Audit	-	Passed
9	Malicious Event Log Audit	-	Passed

10	Scoping and Declarations Audit	-	Passed
11	Replay Attack Audit	ECDSA's Signature Replay Audit	Passed
12	Uninitialized Storage Pointers Audit	-	Passed
13	Arithmetic Accuracy Deviation Audit	-	Passed

Audit Result : **Passed**

Audit Number : 0X001911290001

Audit Date : November 29, 2019

Audit Team : SlowMist Security Team

(**Statement :** SlowMist only issues this report based on the fact that has occurred or existed before the report is issued, and bears the corresponding responsibility in this regard. For the facts occur or exist later after the report, SlowMist cannot judge the security status of its smart contract. SlowMist is not responsible for it. The security audit analysis and other contents of this report are based on the documents and materials provided by the information provider to SlowMist as of the date of this report (referred to as "the provided information"). SlowMist assumes that: there has been no information missing, tampered, deleted, or concealed. If the information provided has been missed, modified, deleted, concealed or reflected and is inconsistent with the actual situation, SlowMist will not bear any responsibility for the resulting loss and adverse effects. SlowMist will not bear any responsibility for the background or other circumstances of the project.)

Summary: This is a token contract that contain the tokenVault and Multisignature section. The total amount of contract tokens can be changed, users can burn their tokens through the burn function. Minter can use the mint function to unlimited mint tokens. OpenZeppelin's SafeMath security module is used, which is a commendable approach. The contract does not have the Overflow and the Race Conditions issue. The comprehensive evaluation contract is no risk.

The source code:

AXL_deploy_191123.sol:

//SlowMist// The contract does not have the Overflow and the Race Conditions issue

```
pragma solidity ^0.5.0;
/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
 * the optional functions; to access them see {ERC20Detailed}.
 */
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);
}
/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * This test is non-exhaustive, and there may be false-negatives: during the
     * execution of a contract's constructor, its address will be reported as
     * not containing 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.
     */
    function isContract(address account) internal view returns (bool) {
```

```
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }

    /**
     * @dev Converts an `address` into `address payable`. Note that this is
     * simply a type cast: the actual underlying value is not changed.
     *
     * _Available since v2.4.0._
     */
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }

    /**
     * @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-interactions-pattern[checks-effects-interactions pattern].
     *
```

```
     * _Available since v2.4.0._
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
}
/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
```

//SlowMist// OpenZeppelin's SafeMath security module is used, which is a commendable approach

```
library SafeMath {
    /**
     * @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) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
```

```
        return c;
    }

    /**
     * @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 sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     *
     * _Available since v2.4.0._
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @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) {
        // 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 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts 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) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts 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.
     *
     * _Available since v2.4.0._
```

```
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts 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 mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message 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.
     *
     * _Available since v2.4.0._
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
```

```
}
library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreased allowance
below zero");
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxing the requirement
     * on the return value: the return value is optional (but if data is returned, it must not be false).
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function callOptionalReturn(IERC20 token, bytes memory data) private {
```

```
        // We need to perform a low level call here, to bypass Solidity's return data size checking mechanism, since
        // we're implementing it ourselves.

        // A Solidity high level call has three parts:
        //   1. The target address is checked to verify it contains contract code
        //   2. The call itself is made, and success asserted
        //   3. The return value is decoded, which in turn checks the size of the returned data.
        // solhint-disable-next-line max-line-length
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}
contract TokenTimelock {
  using SafeERC20 for IERC20;

  // ERC20 basic token contract being held
  IERC20 private _token;

  // beneficiary of tokens after they are released
  address private _beneficiary;

  // timestamp when token release is enabled
  uint256 private _releaseTime;

  // generator of the tokenLock
  address private _owner;
  bool private _ownable;

  event UnLock(address _receiver, uint256 _amount);

  constructor(IERC20 token, address beneficiary, uint256 releaseTime) public {
    _token = token;
    _beneficiary = beneficiary;
    _releaseTime = releaseTime;
```

```
    }

    /**
     * @return the token being held.
     */
    function token() public view returns (IERC20) {
        return _token;
    }

    /**
     * @return the beneficiary of the tokens.
     */
    function beneficiary() public view returns (address) {
        return _beneficiary;
    }

    /**
     * @return the time when the tokens are released.
     */
    function releaseTime() public view returns (uint256) {
        return _releaseTime;
    }

    /**
     * @notice Transfers tokens held by timelock to beneficiary.
     */
    function release() public {
        require(block.timestamp >= _releaseTime);

        uint256 amount = _token.balanceOf(address(this));
        require(amount > 0);

        _token.safeTransfer(_beneficiary, amount);
        emit UnLock(_beneficiary, amount);
    }
}
/*
 * @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 GSN 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.
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}
/**
 * @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 {ERC20Mintable}.
 *
 * 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 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 Context, IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

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

    uint256 private _totalSupply;

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

    /**
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account) public view 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 returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true; //SlowMist// The return value conforms to the EIP20 specification
    }

    /**
     * @dev See {IERC20-allowance}.
     */
```

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

    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public returns (bool) {
        _approve(_msgSender(), spender, amount);

        return true; //SlowMist// The return value conforms to the EIP20 specification

    }

    /**
     * @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};
     *
     * 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 returns (bool) {

        //SlowMist// require(value <= _allowed[sender][msg.sender]);

        //SlowMist// It is recommended to add to code above, can optimize Gas

        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));

        return true; //SlowMist// The return value conforms to the EIP20 specification

    }

    /**
```

```
     * @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 returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(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 returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below zero"));
        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 {
        require(sender != address(0), "ERC20: transfer from the zero address");

        require(recipient != address(0), "ERC20: transfer to the zero address"); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer


        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(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 {

        require(account != address(0), "ERC20: mint to the zero address"); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during mint


        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(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 {
        require(account != address(0), "ERC20: burn from the zero address");

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens.
     *
     * This is 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 {
        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 Destroys `amount` tokens from `account`.`amount` is then deducted
     * from the caller's allowance.
     *
```

```
     * See {_burn} and {_approve}.
     */

    //SlowMist// Because _burnFrom() and transferFrom() share the _allowances amount of _approve(), if the agent be evil, there is the possibility of malicious burn
    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
        _approve(account, _msgSender(), _allowances[account][_msgSender()].sub(amount, "ERC20: burn amount exceeds allowance"));
    }

    function _multiTransfer(address[] memory _to, uint256[] memory _amount) internal {
        require(_to.length == _amount.length);

        uint256 ui;
        uint256 amountSum = 0;

        for (ui = 0; ui < _to.length; ui++) {
            require(_to[ui] != address(0)); //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer

            amountSum = amountSum.add(_amount[ui]);
        }

        require(amountSum <= _balances[msg.sender]);

        for (ui = 0; ui < _to.length; ui++) {
            _balances[msg.sender] = _balances[msg.sender].sub(_amount[ui]);
            _balances[_to[ui]] = _balances[_to[ui]].add(_amount[ui]);

            emit Transfer(msg.sender, _to[ui], _amount[ui]);
        }
    }
}
/**
 * @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.
```

```
 *
 * 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.
 */
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 () internal {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

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

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

    /**
     * @dev Returns true if the caller is the current owner.
     */
    function isOwner() public view returns (bool) {
        return _msgSender() == _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 onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = 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 onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address"); //SlowMist// This check is
quite good in avoiding losing control of the contract caused by user mistakes
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}
/**
 * @title Roles
 * @dev Library for managing addresses assigned to a Role.
 */
library Roles {
    struct Role {
        mapping (address => bool) bearer;
    }

    /**
     * @dev Give an account access to this role.
```

```
     */
    function add(Role storage role, address account) internal {
        require(!has(role, account), "Roles: account already has role");
        role.bearer[account] = true;
    }

    /**
     * @dev Remove an account's access to this role.
     */
    function remove(Role storage role, address account) internal {
        require(has(role, account), "Roles: account does not have role");
        role.bearer[account] = false;
    }

    /**
     * @dev Check if an account has this role.
     * @return bool
     */
    function has(Role storage role, address account) internal view returns (bool) {
        require(account != address(0), "Roles: account is the zero address");
        return role.bearer[account];
    }
}
contract PauserRole is Context {
    using Roles for Roles.Role;

    event PauserAdded(address indexed account);
    event PauserRemoved(address indexed account);

    Roles.Role private _pausers;

    constructor () internal {
        _addPauser(_msgSender());
    }

    modifier onlyPauser() {
        require(isPauser(_msgSender()), "PauserRole: caller does not have the Pauser role");
        _;
    }

    function isPauser(address account) public view returns (bool) {
        return _pausers.has(account);
```

```
    }

    function addPauser(address account) public onlyPauser {
        _addPauser(account);
    }

    function renouncePauser() public {
        _removePauser(_msgSender());
    }

    function _addPauser(address account) internal {
        _pausers.add(account);
        emit PauserAdded(account);
    }

    function _removePauser(address account) internal {
        _pausers.remove(account);
        emit PauserRemoved(account);
    }
}
/**
 * @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.
 */
contract Pausable is Context, PauserRole {
    /**
     * @dev Emitted when the pause is triggered by a pauser (`account`).
     */
    event Paused(address account);

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

    bool private _paused;
```

```
    /**
     * @dev Initializes the contract in unpaused state. Assigns the Pauser role
     * to the deployer.
     */
    constructor () internal {
        _paused = false;
    }

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

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

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

    /**
     * @dev Called by a pauser to pause, triggers stopped state.
     */
```

//SlowMist// Suspending all transactions upon major abnormalities is a recommended approach

```
    function pause() public onlyPauser whenNotPaused {
        _paused = true;
        emit Paused(_msgSender());
```

```
    }

    /**
     * @dev Called by a pauser to unpause, returns to normal state.
     */
    function unpause() public onlyPauser whenPaused {
        _paused = false;
        emit Unpaused(_msgSender());
    }
}
/**
 * @title Pausable token
 * @dev ERC20 with pausable transfers and allowances.
 *
 * Useful if you want to stop trades until the end of a crowdsale, or have
 * an emergency switch for freezing all token transfers in the event of a large
 * bug.
 */
contract ERC20Pausable is ERC20, Pausable {
    function transfer(address to, uint256 value) public whenNotPaused returns (bool) {
        return super.transfer(to, value);
    }

    function transferFrom(address from, address to, uint256 value) public whenNotPaused returns (bool) {
        return super.transferFrom(from, to, value);
    }

    function approve(address spender, uint256 value) public whenNotPaused returns (bool) {
        return super.approve(spender, value);
    }

    function increaseAllowance(address spender, uint256 addedValue) public whenNotPaused returns (bool) {
        return super.increaseAllowance(spender, addedValue);
    }

    function decreaseAllowance(address spender, uint256 subtractedValue) public whenNotPaused returns (bool) {
        return super.decreaseAllowance(spender, subtractedValue);
    }
}
/**
 * @dev Extension of {ERC20} that allows token holders to destroy both their own
 * tokens and those that they have an allowance for, in a way that can be
```

```
 * recognized off-chain (via event analysis).
 */
contract ERC20Burnable is Context, ERC20 {
    /**
     * @dev Destroys `amount` tokens from the caller.
     *
     * See {ERC20-_burn}.
     */
    function burn(uint256 amount) public {
        _burn(_msgSender(), amount);
    }

    /**
     * @dev See {ERC20-_burnFrom}.
     */
    function burnFrom(address account, uint256 amount) public {
        _burnFrom(account, amount);
    }
}
contract MinterRole is Context {
    using Roles for Roles.Role;

    event MinterAdded(address indexed account);
    event MinterRemoved(address indexed account);

    Roles.Role private _minters;

    constructor () internal {
        _addMinter(_msgSender());
    }

    modifier onlyMinter() {
        require(isMinter(_msgSender()), "MinterRole: caller does not have the Minter role");
        _;
    }

    function isMinter(address account) public view returns (bool) {
        return _minters.has(account);
    }

    function addMinter(address account) public onlyMinter {
        _addMinter(account);
```

```
    }

    function renounceMinter() public {
        _removeMinter(_msgSender());
    }

    function _addMinter(address account) internal {
        _minters.add(account);
        emit MinterAdded(account);
    }

    function _removeMinter(address account) internal {
        _minters.remove(account);
        emit MinterRemoved(account);
    }
}
/**
 * @dev Extension of {ERC20} that adds a set of accounts with the {MinterRole},
 * which have permission to mint (create) new tokens as they see fit.
 *
 * At construction, the deployer of the contract is the only minter.
 */
contract ERC20Mintable is ERC20, MinterRole {
    /**
     * @dev See {ERC20-_mint}.
     *
     * Requirements:
     *
     * - the caller must have the {MinterRole}.
     */
    function mint(address account, uint256 amount) public onlyMinter returns (bool) {
        _mint(account, amount);
        return true;
    }
}
// ----------------------------------------------------------------------------
// @title MultiTransfer Token
// @dev Only Admin
// ----------------------------------------------------------------------------
contract MultiTransferToken is ERC20, Ownable {

    function multiTransfer(address[] memory _to, uint256[] memory _amount) onlyOwner public returns (bool) {
```

```
            _multiTransfer(_to, _amount);

            return true;
        }
}
contract AXLCoin is ERC20Pausable, ERC20Burnable, ERC20Mintable, MultiTransferToken {
    string public constant name = "AXiaL Entertainment Digital Asset";
    string public constant symbol = "AXL";
    uint public constant decimals = 18;

    // Lock
    mapping (address => address) public lockStatus;
    event Lock(address _receiver, uint256 _amount);

    // Airdrop
    mapping (address => uint256) public airDropHistory;
    event AirDrop(address _receiver, uint256 _amount);

    constructor() public {}

    function dropToken(address[] memory receivers, uint256[] memory values) public {
        require(receivers.length != 0);
        require(receivers.length == values.length);

        for (uint256 i = 0; i < receivers.length; i++) {
            address receiver = receivers[i];
            uint256 amount = values[i];

            transfer(receiver, amount);
            airDropHistory[receiver] += amount;

            emit AirDrop(receiver, amount);
        }
    }

    function timeLockToken(address beneficiary, uint256 amount, uint256 releaseTime) onlyOwner public {
        //SlowMist// require(beneficiary != address(0));

        //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of
token during transfer
```

```
        TokenTimelock lockContract = new TokenTimelock(this, beneficiary, releaseTime);

        transfer(address(lockContract), amount);
        lockStatus[beneficiary] = address(lockContract);
        emit Lock(beneficiary, amount);
    }
}
```

MultiSig_deploy.sol:

```
pragma solidity ^0.4.15;


/// @title Multisignature wallet - Allows multiple parties to agree on transactions before execution.
/// @author Stefan George - <stefan.george@consensys.net>
contract MultiSigWallet {

    /*
     *  Events
     */
    event Confirmation(address indexed sender, uint indexed transactionId);
    event Revocation(address indexed sender, uint indexed transactionId);
    event Submission(uint indexed transactionId);
    event Execution(uint indexed transactionId);
    event ExecutionFailure(uint indexed transactionId);
    event Deposit(address indexed sender, uint value);
    event OwnerAddition(address indexed owner);
    event OwnerRemoval(address indexed owner);
    event RequirementChange(uint required);

    /*
     *  Constants
     */
    uint constant public MAX_OWNER_COUNT = 50;

    /*
     *  Storage
     */
    mapping (uint => Transaction) public transactions;
    mapping (uint => mapping (address => bool)) public confirmations;
    mapping (address => bool) public isOwner;
    address[] public owners;
```

```
    uint public required;
    uint public transactionCount;

    struct Transaction {
        address destination;
        uint value;
        bytes data;
        bool executed;
    }

    /*
     *  Modifiers
     */
    modifier onlyWallet() {
        require(msg.sender == address(this));
        _;
    }

    modifier ownerDoesNotExist(address owner) {
        require(!isOwner[owner]);
        _;
    }

    modifier ownerExists(address owner) {
        require(isOwner[owner]);
        _;
    }

    modifier transactionExists(uint transactionId) {
        require(transactions[transactionId].destination != 0);
        _;
    }

    modifier confirmed(uint transactionId, address owner) {
        require(confirmations[transactionId][owner]);
        _;
    }

    modifier notConfirmed(uint transactionId, address owner) {
        require(!confirmations[transactionId][owner]);
        _;
    }
```

```
modifier notExecuted(uint transactionId) {
    require(!transactions[transactionId].executed);
    _;
}

modifier notNull(address _address) {
    require(_address != 0);
    _;
}

modifier validRequirement(uint ownerCount, uint _required) {
    require(ownerCount <= MAX_OWNER_COUNT
        && _required <= ownerCount
        && _required != 0
        && ownerCount != 0);
    _;
}

/// @dev Fallback function allows to deposit ether.
function()
    payable
{
    if (msg.value > 0)
        Deposit(msg.sender, msg.value);
}

/*
 * Public functions
 */
/// @dev Contract constructor sets initial owners and required number of confirmations.
/// @param _owners List of initial owners.
/// @param _required Number of required confirmations.
function MultiSigWallet(address[] _owners, uint _required)
    public
    validRequirement(_owners.length, _required)
{
    for (uint i=0; i<_owners.length; i++) {
        require(!isOwner[_owners[i]] && _owners[i] != 0);
        isOwner[_owners[i]] = true;
    }
    owners = _owners;
```

```
        required = _required;
    }

    /// @dev Allows to add a new owner. Transaction has to be sent by wallet.
    /// @param owner Address of new owner.
    function addOwner(address owner)
        public //Slowmist// use external to optimize Gas
        onlyWallet
        ownerDoesNotExist(owner)
        notNull(owner)
        validRequirement(owners.length + 1, required)
    {
        isOwner[owner] = true;
        owners.push(owner);
        OwnerAddition(owner);
    }

    /// @dev Allows to remove an owner. Transaction has to be sent by wallet.
    /// @param owner Address of owner.
    function removeOwner(address owner)
        public
        onlyWallet
        ownerExists(owner)
    {
        isOwner[owner] = false;
        for (uint i=0; i<owners.length - 1; i++)
            if (owners[i] == owner) {
                owners[i] = owners[owners.length - 1];
                break;
            }
        owners.length -= 1;
        if (required > owners.length)
            changeRequirement(owners.length);
        OwnerRemoval(owner);
    }

    /// @dev Allows to replace an owner with a new owner. Transaction has to be sent by wallet.
    /// @param owner Address of owner to be replaced.
    /// @param newOwner Address of new owner.
    function replaceOwner(address owner, address newOwner)
        public
```

```
        onlyWallet
        ownerExists(owner)
        ownerDoesNotExist(newOwner)
    {
        for (uint i=0; i<owners.length; i++)
            if (owners[i] == owner) {
                owners[i] = newOwner;
                break;
            }
        isOwner[owner] = false;
        isOwner[newOwner] = true;
        OwnerRemoval(owner);
        OwnerAddition(newOwner);
    }

    /// @dev Allows to change the number of required confirmations. Transaction has to be sent by wallet.
    /// @param _required Number of required confirmations.
    function changeRequirement(uint _required)
        public
        onlyWallet
        validRequirement(owners.length, _required)
    {
        required = _required;
        RequirementChange(_required);
    }

    /// @dev Allows an owner to submit and confirm a transaction.
    /// @param destination Transaction target address.
    /// @param value Transaction ether value.
    /// @param data Transaction data payload.
    /// @return Returns transaction ID.
    function submitTransaction(address destination, uint value, bytes data)
        public
        returns (uint transactionId)
    {
        transactionId = addTransaction(destination, value, data);
        confirmTransaction(transactionId);
    }

    /// @dev Allows an owner to confirm a transaction.
    /// @param transactionId Transaction ID.
```

//SlowMist// Multisignature to confirm transaction

```
function confirmTransaction(uint transactionId)
    public
    ownerExists(msg.sender)
    transactionExists(transactionId)
    notConfirmed(transactionId, msg.sender)
{
    confirmations[transactionId][msg.sender] = true;
    Confirmation(msg.sender, transactionId);
    executeTransaction(transactionId);
}

/// @dev Allows an owner to revoke a confirmation for a transaction.
/// @param transactionId Transaction ID.
function revokeConfirmation(uint transactionId)
    public
    ownerExists(msg.sender)
    confirmed(transactionId, msg.sender)
    notExecuted(transactionId)
{
    confirmations[transactionId][msg.sender] = false;
    Revocation(msg.sender, transactionId);
}

/// @dev Allows anyone to execute a confirmed transaction.
/// @param transactionId Transaction ID.
function executeTransaction(uint transactionId)
    public
    ownerExists(msg.sender)
    confirmed(transactionId, msg.sender)
    notExecuted(transactionId)
{
    if (isConfirmed(transactionId)) {
        Transaction storage txn = transactions[transactionId];
        txn.executed = true;
        if (external_call(txn.destination, txn.value, txn.data.length, txn.data))
            Execution(transactionId);
        else {
            ExecutionFailure(transactionId);
            txn.executed = false;
        }
```

```
        }
    }

    // call has been separated into its own function in order to take advantage
    // of the Solidity's code generator to produce a loop that copies tx.data into memory.
    function external_call(address destination, uint value, uint dataLength, bytes data) internal returns (bool) {
        bool result;
        assembly {
            let x := mload(0x40)    // "Allocate" memory for output (0x40 is where "free memory" pointer is stored by convention)
            let d := add(data, 32) // First 32 bytes are the padded length of data, so exclude that
            result := call(
                sub(gas, 34710),    // 34710 is the value that solidity is currently emitting
                                    // It includes callGas (700) + callVeryLow (3, to pay for SUB) + callValueTransferGas (9000) +
                                    // callNewAccountGas (25000, in case the destination address does not exist and needs creating)
                destination,
                value,
                d,
                dataLength,         // Size of the input (in bytes) - this is what fixes the padding problem
                x,
                0                   // Output is ignored, therefore the output size is zero
            )
        }
        return result;
    }

    /// @dev Returns the confirmation status of a transaction.
    /// @param transactionId Transaction ID.
    /// @return Confirmation status.
    function isConfirmed(uint transactionId)
        public
        constant
        returns (bool)
    {
        uint count = 0;
        for (uint i=0; i<owners.length; i++) {
            if (confirmations[transactionId][owners[i]])
                count += 1;
            if (count == required)
                return true;
```

```
        }
    }

    /*
     * Internal functions
     */
    /// @dev Adds a new transaction to the transaction mapping, if transaction does not exist yet.
    /// @param destination Transaction target address.
    /// @param value Transaction ether value.
    /// @param data Transaction data payload.
    /// @return Returns transaction ID.
    function addTransaction(address destination, uint value, bytes data)
        internal
        notNull(destination)
        returns (uint transactionId)
    {
        transactionId = transactionCount;
        transactions[transactionId] = Transaction({
            destination: destination,
            value: value,
            data: data,
            executed: false
        });
        transactionCount += 1;
        Submission(transactionId);
    }

    /*
     * Web3 call functions
     */
    /// @dev Returns number of confirmations of a transaction.
    /// @param transactionId Transaction ID.
    /// @return Number of confirmations.
    function getConfirmationCount(uint transactionId)
        public
        constant
        returns (uint count)
    {
        for (uint i=0; i<owners.length; i++)
            if (confirmations[transactionId][owners[i]])
                count += 1;
    }
```

```
    /// @dev Returns total number of transactions after filers are applied.
    /// @param pending Include pending transactions.
    /// @param executed Include executed transactions.
    /// @return Total number of transactions after filters are applied.
    function getTransactionCount(bool pending, bool executed)
        public
        constant
        returns (uint count)
    {
        for (uint i=0; i<transactionCount; i++)
            if (    pending && !transactions[i].executed
                || executed && transactions[i].executed)
                count += 1;
    }

    /// @dev Returns list of owners.
    /// @return List of owner addresses.
    function getOwners()
        public
        constant
        returns (address[])
    {
        return owners;
    }

    /// @dev Returns array with owner addresses, which confirmed transaction.
    /// @param transactionId Transaction ID.
    /// @return Returns array of owner addresses.
    function getConfirmations(uint transactionId)
        public
        constant
        returns (address[] _confirmations)
    {
        address[] memory confirmationsTemp = new address[](owners.length);
        uint count = 0;
        uint i;
        for (i=0; i<owners.length; i++)
            if (confirmations[transactionId][owners[i]]) {
                confirmationsTemp[count] = owners[i];
                count += 1;
            }
```

```
        _confirmations = new address[](count);
        for (i=0; i<count; i++)
            _confirmations[i] = confirmationsTemp[i];
    }

    /// @dev Returns list of transaction IDs in defined range.
    /// @param from Index start position of transaction array.
    /// @param to Index end position of transaction array.
    /// @param pending Include pending transactions.
    /// @param executed Include executed transactions.
    /// @return Returns array of transaction IDs.
    function getTransactionIds(uint from, uint to, bool pending, bool executed)
        public
        constant
        returns (uint[] _transactionIds)
    {
        uint[] memory transactionIdsTemp = new uint[](transactionCount);
        uint count = 0;
        uint i;
        for (i=0; i<transactionCount; i++)
            if (   pending && !transactions[i].executed
                || executed && transactions[i].executed)
            {
                transactionIdsTemp[count] = i;
                count += 1;
            }
        _transactionIds = new uint[](to - from);
        for (i=from; i<to; i++)
            _transactionIds[i - from] = transactionIdsTemp[i];
    }
}
```

慢雾科技
slow mist

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account