



## Smart Contract Security Audit Report





The SlowMist Security Team received the Dam team's application for smart contract security audit of the Dam Token on Oct 09, 2020. The following are the details and results of this smart contract security audit:

**Token name :**

DAM

**The Contract address :**

0xf80d589b3dbe130c270a69f1a69d050f268786df

**Link address :**

<https://etherscan.io/address/0xf80d589b3dbe130c270a69f1a69d050f268786df>

**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 : 0X002010130003**

**Audit Date : October 13, 2020**

**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 does not contain the tokenVault section. The total amount of contract tokens can changed. Users can burn their own tokens. Operator can burn the balance of the account but need authorization. OpenZeppelin's SafeMath security Module is used, which is a recommend approach. The contract does not have the Overflow and the Race Conditions issue.**

The source code:

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

```
/** *Submitted for verification at Etherscan.io on 2020-05-08*/  
// File: @openzeppelin/contracts/GSN/Context.sol  
pragma solidity ^0.6.0;  
  
/* * @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 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 { }

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

function _msgData() internal view virtual returns (bytes memory) {
    this; // silence state mutability warning without generating bytecode - see
https://github.com/ethereum/solidity/issues/2691
    return msg.data;
}

// File: @openzeppelin/contracts/token/ERC777/IERC777.sol
pragma solidity ^0.6.0;

/** * @dev Interface of the ERC777Token standard as defined in the EIP. * * This contract uses the *
https://eips.ethereum.org/EIPS/eip-1820[ERC1820 registry standard] to let * token holders and
recipients react to token movements by using setting implementers * for the associated interfaces in
said registry. See {IERC1820Registry} and * {ERC1820Implementer}. */
interface IERC777 {
    /** * @dev Returns the name of the token. */
    function name() external view returns (string memory);

    /** * @dev Returns the symbol of the token, usually a shorter version of the * name. */
    function symbol() external view returns (string memory);

    /** * @dev Returns the smallest part of the token that is not divisible. This * means all
token operations (creation, movement and destruction) must have * amounts that are a multiple of
this number. * * For most token contracts, this value will equal 1. */
    function granularity() external view returns (uint256);

    /** * @dev Returns the amount of tokens in existence. */
    function totalSupply() external view returns (uint256);

    /** * @dev Returns the amount of tokens owned by an account (`owner`). */
    function balanceOf(address owner) external view returns (uint256);

    /** * @dev Moves `amount` tokens from the caller's account to `recipient`. * * If send
or receive hooks are registered for the caller and `recipient`, * the corresponding functions will
be called with `data` and empty * `operatorData`. See {IERC777Sender} and {IERC777Recipient}. *
* Emits a {Sent} event. * * Requirements * * - the caller must have at least `amount` tokens.
```



```
* - `recipient` cannot be the zero address.      * - if `recipient` is a contract, it must implement the
{IERC777Recipient}      * interface.      */

function send(address recipient, uint256 amount, bytes calldata data) external;

/**      * @dev Destroys `amount` tokens from the caller's account, reducing the      * total supply.
*      * If a send hook is registered for the caller, the corresponding function      * will be called with
`data` and empty `operatorData`. See {IERC777Sender}.      *      * Emits a {Burned} event.      *      *
Requirements      *      * - the caller must have at least `amount` tokens.      */

function burn(uint256 amount, bytes calldata data) external;

/**      * @dev Returns true if an account is an operator of `tokenHolder`.      * Operators can send
and burn tokens on behalf of their owners. All      * accounts are their own operator.      *      * See
{operatorSend} and {operatorBurn}.      */

function isOperatorFor(address operator, address tokenHolder) external view returns (bool);

/**      * @dev Make an account an operator of the caller.      *      * See {isOperatorFor}.      *      *
Emits an {AuthorizedOperator} event.      *      * Requirements      *      * - `operator` cannot be calling
address.      */

function authorizeOperator(address operator) external;

/**      * @dev Revoke an account's operator status for the caller.      *      * See {isOperatorFor}
and {defaultOperators}.      *      * Emits a {RevokedOperator} event.      *      * Requirements      *      *
- `operator` cannot be calling address.      */

function revokeOperator(address operator) external;

/**      * @dev Returns the list of default operators. These accounts are operators      * for all token
holders, even if {authorizeOperator} was never called on      * them.      *      * This list is immutable,
but individual holders may revoke these via      * {revokeOperator}, in which case {isOperatorFor} will
return false.      */

function defaultOperators() external view returns (address[] memory);

/**      * @dev Moves `amount` tokens from `sender` to `recipient`. The caller must      * be an operator
of `sender`.      *      * If send or receive hooks are registered for `sender` and `recipient`,      * the
corresponding functions will be called with `data` and      * `operatorData`. See {IERC777Sender} and
{IERC777Recipient}.      *      * Emits a {Sent} event.      *      * Requirements      *      * - `sender` cannot
be the zero address.      * - `sender` must have at least `amount` tokens.      * - the caller must be an
operator for `sender`.      * - `recipient` cannot be the zero address.      * - if `recipient` is a contract,
it must implement the {IERC777Recipient}      * interface.      */

function operatorSend(
    address sender,
```



```
        address recipient,
        uint256 amount,
        bytes calldata data,
        bytes calldata operatorData
    ) external;

    /**      * @dev Destroys `amount` tokens from `account`, reducing the total supply.      * The caller
must be an operator of `account`.      *      * If a send hook is registered for `account`, the corresponding
function      * will be called with `data` and `operatorData`. See {IERC777Sender}.      *      * Emits a
{Burned} event.      *      * Requirements      *      * - `account` cannot be the zero address.      * -
`account` must have at least `amount` tokens.      * - the caller must be an operator for `account`.      */
    function operatorBurn(
        address account,
        uint256 amount,
        bytes calldata data,
        bytes calldata operatorData
    ) external;

    event Sent(
        address indexed operator,
        address indexed from,
        address indexed to,
        uint256 amount,
        bytes data,
        bytes operatorData
    );

    event Minted(address indexed operator, address indexed to, uint256 amount, bytes data, bytes
operatorData);

    event Burned(address indexed operator, address indexed from, uint256 amount, bytes data, bytes
operatorData);

    event AuthorizedOperator(address indexed operator, address indexed tokenHolder);

    event RevokedOperator(address indexed operator, address indexed tokenHolder);}
// File: @openzeppelin/contracts/token/ERC777/IERC777Recipient.sol
pragma solidity ^0.6.0;

/** * @dev Interface of the ERC777TokensRecipient standard as defined in the EIP. * * Accounts can be
notified of {IERC777} tokens being sent to them by having a * contract implement this interface (contract
```



```
holders can be their own * implementer) and registering it on the *
https://eips.ethereum.org/EIPS/eip-1820[ERC1820 global registry]. * * See {IERC1820Registry} and
{ERC1820Implementer}. */

interface IERC777Recipient {

    /**      * @dev Called by an {IERC777} token contract whenever tokens are being      * moved or created
into a registered account (`to`). The type of operation      * is conveyed by `from` being the zero address
or not.      *      * This call occurs after the token contract's state is updated, so      *
{IERC777-balanceOf}, etc., can be used to query the post-operation state.      *      * This function may
revert to prevent the operation from being executed.      */

    function tokensReceived(
        address operator,
        address from,
        address to,
        uint256 amount,
        bytes calldata userData,
        bytes calldata operatorData
    ) external;
}

// File: @openzeppelin/contracts/token/ERC777/IERC777Sender.sol
pragma solidity ^0.6.0;

/** * @dev Interface of the ERC777TokensSender standard as defined in the EIP. * * {IERC777} Token holders
can be notified of operations performed on their * tokens by having a contract implement this interface
(contract holders can be * their own implementer) and registering it on the *
https://eips.ethereum.org/EIPS/eip-1820[ERC1820 global registry]. * * See {IERC1820Registry} and
{ERC1820Implementer}. */

interface IERC777Sender {

    /**      * @dev Called by an {IERC777} token contract whenever a registered holder's      * (`from`)
tokens are about to be moved or destroyed. The type of operation      * is conveyed by `to` being the
zero address or not.      *      * This call occurs before the token contract's state is updated, so      *
{IERC777-balanceOf}, etc., can be used to query the pre-operation state.      *      * This function may
revert to prevent the operation from being executed.      */

    function tokensToSend(
        address operator,
        address from,
        address to,
        uint256 amount,
        bytes calldata userData,
        bytes calldata operatorData
    ) external;
}

// File: @openzeppelin/contracts/token/ERC20/IERC20.sol
pragma solidity ^0.6.0;
```



```
/** * @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 `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);
}
// File: @openzeppelin/contracts/math/SafeMath.sol
pragma solidity ^0.6.0;
```





```
/** * @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 recommend 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.      */

    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.
 */
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.      */
function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
    require(b != 0, errorMessage);
    return a % b;
}
}

// File: @openzeppelin/contracts/utils/Address.sol
pragma solidity ^0.6.2;

/**
 * @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
     * ====      */
    function isContract(address account) internal view returns (bool) {
        // 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 != accountHash && codehash != 0x0);
    }

    /**
     * @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-int
eractions-pattern[checks-effects-interactions pattern]. */

function sendValue(address payable recipient, uint256 amount) internal {
    require(address(this).balance >= amount, "Address: insufficient balance");

    // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
    (bool success, ) = recipient.call{ value: amount }("");
    require(success, "Address: unable to send value, recipient may have reverted");
}

// File: @openzeppelin/contracts/introspection/IERC1820Registry.sol
pragma solidity ^0.6.0;

/** * @dev Interface of the global ERC1820 Registry, as defined in the
https://eips.ethereum.org/EIPS/eip-1820[EIP]. Accounts may register * implementers for interfaces in
this registry, as well as query support. * * Implementers may be shared by multiple accounts, and can
also implement more * than a single interface for each account. Contracts can implement interfaces *
for themselves, but externally-owned accounts (EOA) must delegate this to a * contract. * * {IERC165}
interfaces can also be queried via the registry. * * For an in-depth explanation and source code analysis,
see the EIP text. */

interface IERC1820Registry {

    /** * @dev Sets `newManager` as the manager for `account`. A manager of an * account is able
to set interface implementers for it. * * By default, each account is its own manager. Passing
a value of `0x0` in * `newManager` will reset the manager to this initial state. * * Emits
a {ManagerChanged} event. * * Requirements: * * - the caller must be the current manager
for `account`. */

    function setManager(address account, address newManager) external;

    /** * @dev Returns the manager for `account`. * * See {setManager}. */

    function getManager(address account) external view returns (address);

    /** * @dev Sets the `implementer` contract as ``account``'s implementer for * `interfaceHash`.
* * `account` being the zero address is an alias for the caller's address. * The zero address
can also be used in `implementer` to remove an old one. * * See {interfaceHash} to learn how these
are created. * * Emits an {InterfaceImplementerSet} event. * * Requirements: * *
- the caller must be the current manager for `account`. * - `interfaceHash` must not be an {IERC165}
interface id (i.e. it must not * end in 28 zeroes). * - `implementer` must implement
{IERC1820Implementer} and return true when * queried for support, unless `implementer` is the caller.
See * {IERC1820Implementer-canImplementInterfaceForAddress}. */
```



```
function setInterfaceImplementer(address account, bytes32 interfaceHash, address implementer)
external;

/**      * @dev Returns the implementer of `interfaceHash` for `account`. If no such      * implementer
is registered, returns the zero address.      *      * If `interfaceHash` is an {IERC165} interface id (i.e.
it ends with 28      * zeroes), `account` will be queried for support of it.      *      * `account` being
the zero address is an alias for the caller's address.      */

function getInterfaceImplementer(address account, bytes32 interfaceHash) external view returns
(address);

/**      * @dev Returns the interface hash for an `interfaceName`, as defined in the      * corresponding
* https://eips.ethereum.org/EIPS/eip-1820#interface-name\[section of the EIP\].      */
function interfaceHash(string calldata interfaceName) external pure returns (bytes32);

/**      * @notice Updates the cache with whether the contract implements an ERC165 interface or not.
* @param account Address of the contract for which to update the cache.      * @param interfaceId ERC165
interface for which to update the cache.      */

function updateERC165Cache(address account, bytes4 interfaceId) external;

/**      * @notice Checks whether a contract implements an ERC165 interface or not.      * If the result
is not cached a direct lookup on the contract address is performed.      * If the result is not cached
or the cached value is out-of-date, the cache MUST be updated manually by calling      * {updateERC165Cache}
with the contract address.      * @param account Address of the contract to check.      * @param
interfaceId ERC165 interface to check.      * @return True if `account` implements `interfaceId`, false
otherwise.      */

function implementsERC165Interface(address account, bytes4 interfaceId) external view returns
(bool);

/**      * @notice Checks whether a contract implements an ERC165 interface or not without using nor
updating the cache.      * @param account Address of the contract to check.      * @param interfaceId
ERC165 interface to check.      * @return True if `account` implements `interfaceId`, false otherwise.
*/

function implementsERC165InterfaceNoCache(address account, bytes4 interfaceId) external view
returns (bool);

event InterfaceImplementerSet(address indexed account, bytes32 indexed interfaceHash, address
indexed implementer);

event ManagerChanged(address indexed account, address indexed newManager);}

// File: @openzeppelin/contracts/token/ERC777/ERC777.sol
```



```
pragma solidity ^0.6.0;

/** * @dev Implementation of the {IERC777} 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}. * * Support for ERC20 is included in this contract, as specified by the EIP: both * the ERC777
and ERC20 interfaces can be safely used when interacting with it. * Both {IERC777-Sent} and
{IERC20-Transfer} events are emitted on token * movements. * * Additionally, the {IERC777-granularity}
value is hard-coded to `1`, meaning that there * are no special restrictions in the amount of tokens
that created, moved, or * destroyed. This makes integration with ERC20 applications seamless. */
contract ERC777 is Context, IERC777, IERC20 {
    using SafeMath for uint256;
    using Address for address;

    IERC1820Registry constant internal _ERC1820_REGISTRY =
IERC1820Registry(0x1820a4B7618BdE71Dce8cdc73aAB6C95905faD24);

    mapping(address => uint256) private _balances;

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;

    // We inline the result of the following hashes because Solidity doesn't resolve them at compile
time.
    // See https://github.com/ethereum/solidity/issues/4024.

    // keccak256("ERC777TokensSender")
    bytes32 constant private _TOKENS_SENDER_INTERFACE_HASH =
        0x29ddb589b1fb5fc7cf394961c1adf5f8c6454761adf795e67fe149f658abe895;

    // keccak256("ERC777TokensRecipient")
    bytes32 constant private _TOKENS_RECIPIENT_INTERFACE_HASH =
        0xb281fc8c12954d22544db45de3159a39272895b169a852b314f9cc762e44c53b;

    // This isn't ever read from - it's only used to respond to the defaultOperators query.
    address[] private _defaultOperatorsArray;

    // Immutable, but accounts may revoke them (tracked in __revokedDefaultOperators).
    mapping(address => bool) private _defaultOperators;
```



```
// For each account, a mapping of its operators and revoked default operators.
mapping(address => mapping(address => bool)) private _operators;
mapping(address => mapping(address => bool)) private _revokedDefaultOperators;

// ERC20-allowances
mapping (address => mapping (address => uint256)) private _allowances;

/**      * @dev `defaultOperators` may be an empty array.      */
constructor(
    string memory name,
    string memory symbol,
    address[] memory defaultOperators
) public {
    _name = name;
    _symbol = symbol;

    _defaultOperatorsArray = defaultOperators;
    for (uint256 i = 0; i < _defaultOperatorsArray.length; i++) {
        _defaultOperators[_defaultOperatorsArray[i]] = true;
    }

    // register interfaces
    _ERC1820_REGISTRY.setInterfaceImplementer(address(this), keccak256("ERC777Token"),
address(this));
    _ERC1820_REGISTRY.setInterfaceImplementer(address(this), keccak256("ERC20Token"),
address(this));
}

/**      * @dev See {IERC777-name}.      */
function name() public view override returns (string memory) {
    return _name;
}

/**      * @dev See {IERC777-symbol}.      */
function symbol() public view override returns (string memory) {
    return _symbol;
}
```



```
/**      * @dev See {ERC20-decimals}.      *      * Always returns 18, as per the      * [ERC777  
EIP] (https://eips.ethereum.org/EIPS/eip-777#backward-compatibility).      */  
  
function decimals() public pure returns (uint8) {  
    return 18;  
}  
  
/**      * @dev See {IERC777-granularity}.      *      * This implementation always returns `1`.      */  
function granularity() public view override returns (uint256) {  
    return 1;  
}  
  
/**      * @dev See {IERC777-totalSupply}.      */  
function totalSupply() public view override(IERC20, IERC777) returns (uint256) {  
    return _totalSupply;  
}  
  
/**      * @dev Returns the amount of tokens owned by an account (`tokenHolder`).      */  
function balanceOf(address tokenHolder) public view override(IERC20, IERC777) returns (uint256) {  
    return _balances[tokenHolder];  
}  
  
/**      * @dev See {IERC777-send}.      *      * Also emits a {IERC20-Transfer} event for ERC20  
compatibility.      */  
function send(address recipient, uint256 amount, bytes memory data) public override {  
    _send(_msgSender(), recipient, amount, data, "", true);  
}  
  
/**      * @dev See {IERC20-transfer}.      *      * Unlike `send`, `recipient` is not required to  
implement the {IERC777Recipient}      * interface if it is a contract.      *      * Also emits a {Sent} event.  
*/  
function transfer(address recipient, uint256 amount) public override returns (bool) {  
    require(recipient != address(0), "ERC777: transfer to the zero address");  
  
    address from = _msgSender();  
  
    _callTokensToSend(from, from, recipient, amount, "", "");  
  
    _move(from, from, recipient, amount, "", "");  
  
    _callTokensReceived(from, from, recipient, amount, "", "", false);  
}
```





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

}

/**      * @dev See {IERC777-burn}.      *      * Also emits a {IERC20-Transfer} event for ERC20
compatibility.      */

//SlowMist// Users can burn their own tokens.

function burn(uint256 amount, bytes memory data) public override {
    _burn(_msgSender(), amount, data, "");
}

/**      * @dev See {IERC777-isOperatorFor}.      */
function isOperatorFor(
    address operator,
    address tokenHolder
) public view override returns (bool) {
    return operator == tokenHolder ||
        (_defaultOperators[operator] && !_revokedDefaultOperators[tokenHolder][operator]) ||
        _operators[tokenHolder][operator];
}

/**      * @dev See {IERC777-authorizeOperator}.      */
function authorizeOperator(address operator) public override {
    require(_msgSender() != operator, "ERC777: authorizing self as operator");

    if (_defaultOperators[operator]) {
        delete _revokedDefaultOperators[_msgSender()][operator];
    } else {
        _operators[_msgSender()][operator] = true;
    }

    emit AuthorizedOperator(operator, _msgSender());
}

/**      * @dev See {IERC777-revokeOperator}.      */
function revokeOperator(address operator) public override {
    require(operator != _msgSender(), "ERC777: revoking self as operator");
}
```



```
    if (_defaultOperators[operator]) {
        _revokedDefaultOperators[_msgSender()][operator] = true;
    } else {
        delete _operators[_msgSender()][operator];
    }

    emit RevokedOperator(operator, _msgSender());
}

/**      * @dev See {IERC777-defaultOperators}.      */
function defaultOperators() public view override returns (address[] memory) {
    return _defaultOperatorsArray;
}

/**      * @dev See {IERC777-operatorSend}.      *      * Emits {Sent} and {IERC20-Transfer} events.
*/
function operatorSend(
    address sender,
    address recipient,
    uint256 amount,
    bytes memory data,
    bytes memory operatorData
)
public override
{
    require(isOperatorFor(_msgSender(), sender), "ERC777: caller is not an operator for holder");
    _send(sender, recipient, amount, data, operatorData, true);
}

/**      * @dev See {IERC777-operatorBurn}.      *      * Emits {Burned} and {IERC20-Transfer} events.
*/

//SlowMist// Operator can burn the balance of the account but need authorization.

function operatorBurn(address account, uint256 amount, bytes memory data, bytes memory operatorData)
public override {
    require(isOperatorFor(_msgSender(), account), "ERC777: caller is not an operator for holder");
    _burn(account, amount, data, operatorData);
}
```



```
/**      * @dev See {IERC20-allowance}.      *      * Note that operator and allowance concepts are
orthogonal: operators may      * not have allowance, and accounts with allowance may not be operators
* themselves.      */

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

/**      * @dev See {IERC20-approve}.      *      * Note that accounts cannot have allowance issued by
their operators.      */

function approve(address spender, uint256 value) public override returns (bool) {
    address holder = _msgSender();
    _approve(holder, spender, value);

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

/**      * @dev See {IERC20-transferFrom}.      *      * Note that operator and allowance concepts are
orthogonal: operators cannot      * call `transferFrom` (unless they have allowance), and accounts with
* allowance cannot call `operatorSend` (unless they are operators).      *      * Emits {Sent},
{IERC20-Transfer} and {IERC20-Approval} events.      */

function transferFrom(address holder, address recipient, uint256 amount) public override returns
(bool) {
    require(recipient != address(0), "ERC777: transfer to the zero address");
    require(holder != address(0), "ERC777: transfer from the zero address");

    address spender = _msgSender();

    _callTokensToSend(spender, holder, recipient, amount, "", "");

    _move(spender, holder, recipient, amount, "", "");
    _approve(holder, spender, _allowances[holder][spender].sub(amount, "ERC777: transfer amount
exceeds allowance"));

    _callTokensReceived(spender, holder, recipient, amount, "", "", false);

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



```
/**      * @dev Creates `amount` tokens and assigns them to `account`, increasing      * the total supply.
*      * If a send hook is registered for `account`, the corresponding function      * will be called with
`operator`, `data` and `operatorData`.      *      * See {IERC777Sender} and {IERC777Recipient}.      *
* Emits {Minted} and {IERC20-Transfer} events.      *      * Requirements      *      * - `account` cannot
be the zero address.      * - if `account` is a contract, it must implement the {IERC777Recipient}      *
interface.      */
function _mint(
    address account,
    uint256 amount,
    bytes memory userData,
    bytes memory operatorData
)
internal virtual
{
    require(account != address(0), "ERC777: mint to the zero address");

    address operator = _msgSender();

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

    // Update state variables
    _totalSupply = _totalSupply.add(amount);
    _balances[account] = _balances[account].add(amount);

    _callTokensReceived(operator, address(0), account, amount, userData, operatorData, true);

    emit Minted(operator, account, amount, userData, operatorData);
    emit Transfer(address(0), account, amount);
}

/**      * @dev Send tokens      * @param from address token holder address      * @param to address
recipient address      * @param amount uint256 amount of tokens to transfer      * @param userData bytes
extra information provided by the token holder (if any)      * @param operatorData bytes extra information
provided by the operator (if any)      * @param requireReceptionAck if true, contract recipients are
required to implement ERC777TokensRecipient      */
function _send(
    address from,
    address to,
    uint256 amount,
    bytes memory userData,
```



```
        bytes memory operatorData,
        bool requireReceptionAck
    )

    internal
    {
        require(from != address(0), "ERC777: send from the zero address");
        require(to != address(0), "ERC777: send to the zero address");

        address operator = _msgSender();

        _callTokensToSend(operator, from, to, amount, userData, operatorData);

        _move(operator, from, to, amount, userData, operatorData);

        _callTokensReceived(operator, from, to, amount, userData, operatorData, requireReceptionAck);
    }

    /**
     * @dev Burn tokens
     * @param from address token holder address
     * @param amount uint256
     amount of tokens to burn
     * @param data bytes extra information provided by the token holder
     * @param operatorData bytes extra information provided by the operator (if any)
     */
    function _burn(
        address from,
        uint256 amount,
        bytes memory data,
        bytes memory operatorData
    )
    internal virtual
    {
        require(from != address(0), "ERC777: burn from the zero address");

        address operator = _msgSender();

        _beforeTokenTransfer(operator, from, address(0), amount);

        _callTokensToSend(operator, from, address(0), amount, data, operatorData);

        // Update state variables
        _balances[from] = _balances[from].sub(amount, "ERC777: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
    }
}
```

```
emit Burned(operator, from, amount, data, operatorData);
emit Transfer(from, address(0), amount);
}

function _move(
    address operator,
    address from,
    address to,
    uint256 amount,
    bytes memory userData,
    bytes memory operatorData
)
{
    private

    _beforeTokenTransfer(operator, from, to, amount);

    _balances[from] = _balances[from].sub(amount, "ERC777: transfer amount exceeds balance");
    _balances[to] = _balances[to].add(amount);

    emit Sent(operator, from, to, amount, userData, operatorData);
    emit Transfer(from, to, amount);
}

/**
 * @dev See {ERC20-_approve}.
 * Note that accounts cannot have allowance issued by
their operators.
 */
function _approve(address holder, address spender, uint256 value) internal {
    require(holder != address(0), "ERC777: approve from the zero address");
    require(spender != address(0), "ERC777: approve to the zero address");

    _allowances[holder][spender] = value;
    emit Approval(holder, spender, value);
}

/**
 * @dev Call from.tokensToSend() if the interface is registered
 * @param operator address
operator requesting the transfer
 * @param from address token holder address
 * @param to address
recipient address
 * @param amount uint256 amount of tokens to transfer
 * @param userData bytes
extra information provided by the token holder (if any)
 * @param operatorData bytes extra information
provided by the operator (if any)
 */
function _callTokensToSend(
    address operator,
```

```

        address from,
        address to,
        uint256 amount,
        bytes memory userData,
        bytes memory operatorData
    )

    private
    {
        address implementer = _ERC1820_REGISTRY.getInterfaceImplementer(from,
_TOKENS_SENDER_INTERFACE_HASH);

        if (implementer != address(0)) {
            IERC777Sender(implementer).tokensToSend(operator, from, to, amount, userData,
operatorData);
        }
    }

    /**
     * @dev Call to.tokensReceived() if the interface is registered. Reverts if the recipient is
a contract but
     * tokensReceived() was not registered for the recipient
     * @param operator address
operator requesting the transfer
     * @param from address token holder address
     * @param to address
recipient address
     * @param amount uint256 amount of tokens to transfer
     * @param userData bytes
extra information provided by the token holder (if any)
     * @param operatorData bytes extra information
provided by the operator (if any)
     * @param requireReceptionAck if true, contract recipients are
required to implement ERC777TokensRecipient
     */
    function _callTokensReceived(
        address operator,
        address from,
        address to,
        uint256 amount,
        bytes memory userData,
        bytes memory operatorData,
        bool requireReceptionAck
    )

    private
    {
        address implementer = _ERC1820_REGISTRY.getInterfaceImplementer(to,
_TOKENS_RECIPIENT_INTERFACE_HASH);

        if (implementer != address(0)) {
            IERC777Recipient(implementer).tokensReceived(operator, from, to, amount, userData,
operatorData);
        } else if (requireReceptionAck) {

```

```
require(!to.isContract(), "ERC777: token recipient contract has no implementer for
ERC777TokensRecipient");

    }
}

/**      * @dev Hook that is called before any token transfer. This includes      * calls to {send},
{transfer}, {operatorSend}, minting and burning.      *      * Calling conditions:      *      * - when `from`
and `to` are both non-zero, ``from``'s `tokenId` will be      * transferred to `to`.      * - when `from`
is zero, `tokenId` will be minted for `to`.      * - when `to` is zero, ``from``'s `tokenId` 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 operator, address from, address to, uint256 tokenId) internal
virtual { }

// File: contracts/DamToken.sol
pragma solidity ^0.6.0;

contract DamToken is ERC777 {
    constructor () public ERC777("Datamine", "DAM", new address[] (0)) {
        _mint(msg.sender, 25000000 * (10 ** 18), "", "");
    }
}
```





# SLOWMIST

**Official Website**

[www.slowmist.com](http://www.slowmist.com)



**E-mail**

[team@slowmist.com](mailto:team@slowmist.com)



**Twitter**

[@SlowMist\\_Team](https://twitter.com/SlowMist_Team)



**Github**

<https://github.com/slowmist>