慢雾科技
slow mist

智能合约安全审计报告

慢雾安全团队于 2018-06-05 日，收到 AWARE Token 团队对 AT 项目智能合约安全审计申请。如下为本次智能合约安全审计细节及结果：

Token 名称：

AT

合约地址：

0xaa26931ebba45b69834eb35fe3315cab4b1e97fe

链接地址：

https://etherscan.io/address/0xaa26931ebba45b69834eb35fe3315cab4b1e97fe#code

本次审计项及结果：

（其他未知安全漏洞不包含在本次审计责任范围）

序号	审计大类	审计子类	审计结果
1	溢出审计	-	通过
2	条件竞争审计	-	通过
3	权限控制审计	权限漏洞审计	通过
		权限过大审计	通过
4	安全设计审计	Zeppelin 模块使用安全	通过
		编译器版本安全	通过
		硬编码地址安全	通过
		Fallback 函数使用安全	通过
		显现编码安全	通过
		函数返回值安全	通过
5	拒绝服务审计	-	通过
6	Gas 优化审计	-	通过
7	设计逻辑审计	-	通过

备注：审计意见及建议见代码注释 ***//SlowMist//......***

审计结果：**通过**

审计编号：0X001806100002

审计日期：2018 年 06 月 10 日

审计团队：慢雾安全团队

（**声明**：慢雾仅就本报告出具前已经发生或存在的事实出具本报告，并就此承担相应责任。对于出具以后发生或存在的事实，慢雾无法判断其智能合约安全状况，亦不对此承担责任。本报告所作的安全审计分析及其他内容，仅基于信息提供者截至本报告出具时向慢雾提供的文件和资料（简称“已提供资料”）。慢雾假设：已提供资料不存在缺失、被篡改、删减或隐瞒的情形。如已提供资料信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符的，慢雾对由此而导致的损失和不利影响不承担任何责任。）

合约源代码如下：

说明：此为代币(token)合约，不包含锁仓(tokenVault)部分。

```
pragma solidity ^0.4.21;
```

//SlowMist// 合约不存在溢出、条件竞争问题

//SlowMist// 使用了大量 OpenZeppelin 的 SafeMath 及 ERC20 标准模块，值得称赞的做法

```
/**
 * openzeppelin-solidity@1.9.0/contracts/math/SafeMath.sol
 */


/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {

  /**
  * @dev Multiplies two numbers, throws on overflow.
  */
  function mul(uint256 a, uint256 b) internal pure returns (uint256 c) {
    if (a == 0) {
      return 0;
    }
    c = a * b;
    assert(c / a == b);
    return c;
  }

  /**
  * @dev Integer division of two numbers, truncating the quotient.
  */
  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    // uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return a / b;
  }

  /**
  * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater than minuend).
  */
  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }

  /**
  * @dev Adds two numbers, throws on overflow.
  */
  function add(uint256 a, uint256 b) internal pure returns (uint256 c) {
    c = a + b;
    assert(c >= a);
    return c;
  }
}


/**
 * openzeppelin-solidity@1.9.0/contracts/ownership/Ownable.sol
 */

/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic authorization control
 * functions, this simplifies the implementation of "user permissions".
 */
contract Ownable {
  address public owner;


  event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
  /**
   * @dev The Ownable constructor sets the original `owner` of the contract to the sender
   * account.
   */
  function Ownable() public {
    owner = msg.sender;
  }

  /**
   * @dev Throws if called by any account other than the owner.
   */
  modifier onlyOwner() {
    require(msg.sender == owner);
    _;
  }

  /**
   * @dev Allows the current owner to transfer control of the contract to a newOwner.
   * @param newOwner The address to transfer ownership to.
   */
  function transferOwnership(address newOwner) public onlyOwner {
    require(newOwner != address(0)); //SlowMist// 这个检查很好，避免操作失误失去合约控制权
    emit OwnershipTransferred(owner, newOwner);
    owner = newOwner;
  }

}


/**
 * openzeppelin-solidity@1.9.0/contracts/token/ERC20/ERC20Basic.sol
 */

/**
 * @title ERC20Basic
 * @dev Simpler version of ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/179
 */
contract ERC20Basic {
  function totalSupply() public view returns (uint256);
  function balanceOf(address who) public view returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
  event Transfer(address indexed from, address indexed to, uint256 value);
}



/**
 * openzeppelin-solidity@1.9.0/contracts/token/ERC20/ERC20.sol
 */

/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
contract ERC20 is ERC20Basic {
  function allowance(address owner, address spender) public view returns (uint256);
  function transferFrom(address from, address to, uint256 value) public returns (bool);
  function approve(address spender, uint256 value) public returns (bool);
  event Approval(address indexed owner, address indexed spender, uint256 value);
}



/**
 * openzeppelin-solidity@1.9.0/contracts/token/ERC20/BasicToken.sol
 */

/**
 * @title Basic token
 * @dev Basic version of StandardToken, with no allowances.
 */
contract BasicToken is ERC20Basic {
  using SafeMath for uint256;

  mapping(address => uint256) balances;

  uint256 totalSupply_;

  /**
  * @dev total number of tokens in existence
  */
  function totalSupply() public view returns (uint256) {
    return totalSupply_;
  }
  /**
  * @dev transfer token for a specified address
  * @param _to The address to transfer to.
  * @param _value The amount to be transferred.
  */
  function transfer(address _to, uint256 _value) public returns (bool) {
    require(_to != address(0)); //SlowMist// 这类检查很好，避免用户失误导致 Token 转丢
    require(_value <= balances[msg.sender]);

    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    emit Transfer(msg.sender, _to, _value);
    return true; //SlowMist// 返回值符合 EIP20 规范
  }

  /**
  * @dev Gets the balance of the specified address.
  * @param _owner The address to query the the balance of.
  * @return An uint256 representing the amount owned by the passed address.
  */
  function balanceOf(address _owner) public view returns (uint256) {
    return balances[_owner];
  }

}


/**
 * openzeppelin-solidity@1.9.0/contracts/token/ERC20/StandardToken.sol
 */

/**
 * @title Standard ERC20 token
 *
 * @dev Implementation of the basic standard token.
 * @dev https://github.com/ethereum/EIPs/issues/20
 *         @dev         Based         on         code         by         FirstBlood:
https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
 */
```

```
contract StandardToken is ERC20, BasicToken {

  mapping (address => mapping (address => uint256)) internal allowed;


  /**
   * @dev Transfer tokens from one address to another
   * @param _from address The address which you want to send tokens from
   * @param _to address The address which you want to transfer to
   * @param _value uint256 the amount of tokens to be transferred
   */
  function transferFrom(address _from, address _to, uint256 _value) public returns (bool) {

    require(_to != address(0)); //SlowMist// 这类检查很好，避免用户失误导致 Token 转丢

    require(_value <= balances[_from]);
    require(_value <= allowed[_from][msg.sender]);

    balances[_from] = balances[_from].sub(_value);
    balances[_to] = balances[_to].add(_value);
    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
    emit Transfer(_from, _to, _value);

    return true; //SlowMist// 返回值符合 EIP20 规范

  }

  /**
   * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
   *
   * 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
   * @param _spender The address which will spend the funds.
   * @param _value The amount of tokens to be spent.
   */
  function approve(address _spender, uint256 _value) public returns (bool) {
    allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value);

    return true; //SlowMist// 返回值符合 EIP20 规范

  }
```

```
/**
 * @dev Function to check the amount of tokens that an owner allowed to a spender.
 * @param _owner address The address which owns the funds.
 * @param _spender address The address which will spend the funds.
 * @return A uint256 specifying the amount of tokens still available for the spender.
 */
function allowance(address _owner, address _spender) public view returns (uint256) {
  return allowed[_owner][_spender];
}

/**
 * @dev Increase the amount of tokens that an owner allowed to a spender.
 *
 * approve should be called when allowed[_spender] == 0. To increment
 * allowed value is better to use this function to avoid 2 calls (and wait until
 * the first transaction is mined)
 * From MonolithDAO Token.sol
 * @param _spender The address which will spend the funds.
 * @param _addedValue The amount of tokens to increase the allowance by.
 */
function increaseApproval(address _spender, uint _addedValue) public returns (bool) {
  allowed[msg.sender][_spender] = allowed[msg.sender][_spender].add(_addedValue);
  emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
  return true;
}

/**
 * @dev Decrease the amount of tokens that an owner allowed to a spender.
 *
 * approve should be called when allowed[_spender] == 0. To decrement
 * allowed value is better to use this function to avoid 2 calls (and wait until
 * the first transaction is mined)
 * From MonolithDAO Token.sol
 * @param _spender The address which will spend the funds.
 * @param _subtractedValue The amount of tokens to decrease the allowance by.
 */
function decreaseApproval(address _spender, uint _subtractedValue) public returns (bool) {
  uint oldValue = allowed[msg.sender][_spender];

  if (_subtractedValue > oldValue) { //SlowMist// 溢出检查

    allowed[msg.sender][_spender] = 0;
  } else {
      allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
    }
    emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
  }

}



/**
 * openzeppelin-solidity@1.9.0/contracts/token/ERC20/BurnableToken.sol
 */

/**
 * @title Burnable Token
 * @dev Token that can be irreversibly burned (destroyed).
 */
contract BurnableToken is BasicToken {

  event Burn(address indexed burner, uint256 value);

  /**
   * @dev Burns a specific amount of tokens.
   * @param _value The amount of token to be burned.
   */
  function burn(uint256 _value) public {
    _burn(msg.sender, _value);
  }

  function _burn(address _who, uint256 _value) internal {
    require(_value <= balances[_who]);
    // no need to require value <= totalSupply, since that would imply the
    // sender's balance is greater than the totalSupply, which *should* be an assertion failure

    balances[_who] = balances[_who].sub(_value);
    totalSupply_ = totalSupply_.sub(_value);
    emit Burn(_who, _value);
    emit Transfer(_who, address(0), _value);
  }
}
```

```
/**
 * openzeppelin-solidity@1.9.0/contracts/token/ERC20/MintableToken.sol
 */

/**
 * @title Mintable token
 * @dev Simple ERC20 Token example, with mintable token creation
 * @dev Issue: * https://github.com/OpenZeppelin/openzeppelin-solidity/issues/120
 *               Based               on               code               by               TokenMarketNet:
https://github.com/TokenMarketNet/ico/blob/master/contracts/MintableToken.sol
 */
contract MintableToken is StandardToken, Ownable {
  event Mint(address indexed to, uint256 amount);
  event MintFinished();

  bool public mintingFinished = false;


  modifier canMint() {
    require(!mintingFinished);
    _;
  }

  /**
   * @dev Function to mint tokens
   * @param _to The address that will receive the minted tokens.
   * @param _amount The amount of tokens to mint.
   * @return A boolean that indicates if the operation was successful.
   */
  function mint(address _to, uint256 _amount) onlyOwner canMint public returns (bool) {
    totalSupply_ = totalSupply_.add(_amount);
    balances[_to] = balances[_to].add(_amount);
    emit Mint(_to, _amount);
    emit Transfer(address(0), _to, _amount);
    return true;
  }

  /**
   * @dev Function to stop minting new tokens.
   * @return True if the operation was successful.
   */
  function finishMinting() onlyOwner canMint public returns (bool) {
    mintingFinished = true;
    emit MintFinished();
    return true;
  }
}



/**
 * openzeppelin-solidity@1.9.0/contracts/token/ERC20/CappedToken.sol
 */

/**
 * @title Capped token
 * @dev Mintable token with a token cap.
 */
contract CappedToken is MintableToken {

  uint256 public cap;

  function CappedToken(uint256 _cap) public {
    require(_cap > 0);
    cap = _cap;
  }

  /**
   * @dev Function to mint tokens
   * @param _to The address that will receive the minted tokens.
   * @param _amount The amount of tokens to mint.
   * @return A boolean that indicates if the operation was successful.
   */
  function mint(address _to, uint256 _amount) onlyOwner canMint public returns (bool) {
    require(totalSupply_.add(_amount) <= cap); //SlowMist// 这个检查很好，避免 Token 总量无限增发

    return super.mint(_to, _amount);
  }

}
```

```
/**
 * AWARE Token, totalSupply 100000000000000000
 */
contract AwareToken is BurnableToken, CappedToken(100000000000000000) {
    string public name = "AWARE Token";
    string public symbol = "AT";
    uint8 public decimals = 8;

    function burn(uint256 _value) onlyOwner public {
        super.burn(_value);
    }
}
```