Smart Contract Security Audit Report

The SlowMist Security Team received the PANTHEON X TOKEN team's application for smart contract security audit of the XPN on February 05, 2020. The following are the details and results of this smart contract security audit:

Token name :

XPN

The Contract address :

0x3b9e094d56103611f0acefdab43182347ba60df4

Link address :

https://etherscan.io/address/0x3b9e094d56103611f0acefdab43182347ba60df4

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 : 0X002002070001

Audit Date : February 07, 2020

Audit Team : SlowMist Security Team

Summary: This is a token contract that does not contain the tokenVault section. The total amount of contract tokens can be changed, users can burn their tokens through the burn function. 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:

```
/**
 *Submitted for verification at Etherscan.io on 2019-03-27
*/

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

pragma solidity ^0.5.0;


//SlowMist// OpenZeppelin’ s SafeMath security Module is used, which is a recommend
```

approach

```
/**
 * @title SafeMath
 * @dev Unsigned math operations with safety checks that revert on error
 */
library SafeMath {
    /**
    * @dev Multiplies two unsigned integers, reverts on overflow.
    */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b);

        return c;
    }

    /**
    * @dev Integer division of two unsigned integers truncating the quotient, reverts on division by zero.
    */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0);
        uint256 c = a / b;

        return c;
    }

    /**
    * @dev Subtracts two unsigned integers, reverts on overflow (i.e. if subtrahend is greater than minuend).
    */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a);
        uint256 c = a - b;

        return c;
    }

    /**
```

```
    * @dev Adds two unsigned integers, reverts on overflow.
    */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }

    /**
    * @dev Divides two unsigned integers and returns the remainder (unsigned integer modulo),
    * reverts when dividing by zero.
    */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }
}


/**
  * @title ERC20 interface
  * @dev see https://github.com/ethereum/EIPs/issues/20
  */
interface ERC20 {
    function transfer(address to, uint256 value) external returns (bool);

    function approve(address spender, uint256 value) external returns (bool);

    function transferFrom(address from, address to, uint256 value) external returns (bool);

    function totalSupply() external view returns (uint256);

    function balanceOf(address who) external view returns (uint256);

    function allowance(address owner, address spender) external view returns (uint256);

    event Transfer(address indexed from, address indexed to, uint256 value);

    event Approval(address indexed owner, address indexed spender, uint256 value);
}
```

```
/**
 * @title XPN ERC20 token
 */
contract XPN is ERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

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

    string public name = "PANTHEON X TOKEN";

    string public symbol = "XPN";

    uint public decimals = 18;

    uint256 private _totalSupply = 800000000 * 10 ** decimals;

    constructor() public {
        _balances[msg.sender] = _totalSupply;
    }

    /**
    * @dev Total number of tokens in existence
    */
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    /**
    * @dev Gets the balance of the specified address.
    * @param owner The address to query 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];
    }

    /**
     * @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 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) {
        _transfer(msg.sender, to, value);
        return true; //SlowMist// The return value conforms to the EIP20 specification
    }

    /**
     * @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) {
        require(spender != address(0)); //SlowMist// This kind of check is very good, avoiding user mistake leading to approve errors

        _allowed[msg.sender][spender] = value;
        emit Approval(msg.sender, spender, value);
        return true; //SlowMist// The return value conforms to the EIP20 specification
    }

    /**
     * @dev Transfer tokens from one address to another.
     * Note that while this function emits an Approval event, this is not required as per the specification,
```

```
 * and other compliant implementations may not emit the event.
 * @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) {

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

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

    _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value);
    _transfer(from, to, value);
    emit Approval(from, msg.sender, _allowed[from][msg.sender]);

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

}

/**
 * @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
 * Emits an Approval event.
 * @param spender The address which will spend the funds.
 * @param addedValue The amount of tokens to increase the allowance by.
 */
function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
    require(spender != address(0));

    _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
 * Emits an Approval event.
```

```
     * @param spender The address which will spend the funds.
     * @param subtractedValue The amount of tokens to decrease the allowance by.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
        require(spender != address(0));

        _allowed[msg.sender][spender] = _allowed[msg.sender][spender].sub(subtractedValue);
        emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
        return true;
    }

    /**
     * @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);
    }

    /**
     * @dev Burns a specific amount of tokens from the target address and decrements allowance
     * @param from address The address which you want to send tokens from
     * @param value uint256 The amount of token to be burned
     */
    function burnFrom(address from, uint256 value) public {
        _burnFrom(from, value);
    }

    /**
    * @dev Transfer token for a specified addresses
    * @param from The address to transfer from.
    * @param to The address to transfer to.
    * @param value The amount to be transferred.
    */
    function _transfer(address from, address to, uint256 value) internal {

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

        _balances[from] = _balances[from].sub(value);
        _balances[to] = _balances[to].add(value);
```

```
        emit Transfer(from, to, value);
    }

    /**
     * @dev Internal function that burns an amount of the token of a given
     * account.
     * @param account The account whose tokens will be burnt.
     * @param value The amount that will be burnt.
     */
    function _burn(address account, uint256 value) internal {
        require(account != address(0));

        _totalSupply = _totalSupply.sub(value);
        _balances[account] = _balances[account].sub(value);
        emit Transfer(account, address(0), value);
    }

    /**
     * @dev Internal function that burns an amount of the token of a given
     * account, deducting from the sender's allowance for said account. Uses the
     * internal burn function.
     * Emits an Approval event (reflecting the reduced allowance).
     * @param account The account whose tokens will be burnt.
     * @param value The amount that will be burnt.
     */
    //SlowMist// Because _burnFrom() and transferFrom() share the _allowed amount of approve(), if the agent be evil, there is the possibility of malicious burn
    function _burnFrom(address account, uint256 value) internal {
        _allowed[account][msg.sender] = _allowed[account][msg.sender].sub(value);
        _burn(account, value);
        emit Approval(account, msg.sender, _allowed[account][msg.sender]);
    }
}
```

慢雾科技
slow mist

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account