Cryptofox

Smart Contract Audit
Final Report

October 29, 2022

Introduction

1. About Cryptofox

The world's first miner with a unique code that allows you to safely, and most importantly, quickly earn your funds, we help in difficult times, secure your capital from market liquidation, you can put your funds at interest, you will release pools at good interest, so you save your coins from the bear market.

Visit https://cryptofox.finance/ to know more about.

2. About ContractSecurity

ContractSecurity is a security start-up to provide professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ContractSecurity's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 105+ blockchain projects by providing security services on different frameworks. ContractSecurity team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

Visit https://contractsecurity.finance to know more about the services.

Documentation Details

Cryptofox team has provided the following doc for the purpose of audit:

1. https://docs.cryptofox.finance

Audit Process & Methodology

ContractSecurity team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: Cryptofox
- Contract Name: FoxPool
- Languages: Solidity(Smart contract)
- Github commit/Smart Contract Address for final audit: 0xd0444e61Bc3ba138746B89F7FC78E414Eb479779
- Launched Date: Oct-29-2022 10:03:21 PM +UTC

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level References

Every issue in this report was assigned a severity level from the following:

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Conclusion

The FoxPool Smart-Contract found no vulnerabilities, no backdoors, and no scam scripts.
The code was tested with compatible compilers and simulated manually reviewed for all commonly known and specific vulnerabilities.
So, FoxPool Smart-Contract is safe for use in the Binance Smart Chain main network.

Issues	High	Medium	Low
Open	-	1	1
Closed	-	-	-

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

High Severity Issues

No issues were found.

Medium Severity Issues

The system used is called ROI and must be considered HIGH-RISK.
Users' principal deposits cannot be withdrawn. Users can get dividends and referral commissions.
Dividends are paid from deposits of other users. Do always invest with proper knowledge and investigation.

Low Severity Issues

Optimization suggestions:

1- Loop on the dynamic variable (low severity).
If the user gets more parallel deposits, his withdrawal transaction is going to cost more transaction fees because the loop on the dynamic variable is used in the 'withdraw' function.
In case exceeding the GAS limit of the size of the transaction, withdrawal is not possible.
Note: This comment is relevant only if a user creates an excessive number of parallel deposits (more than 100).

Independent description of the smart-contract functionality:

The FoxPool smart contract provides the opportunity to invest any amount in BUSD (from 10 BUSD) in the contract and get a 144% to 400% return on investment between 8 to 40 days if the contract balance has enough funds for payment.

All dividends are calculated at the moment of request and available for withdrawal anytime or after the deposit

Information Cryptofox

FOUR INVESTMENT PLANS

Plans	Total Return	Daily Profit	Days	Withdraw time
1	144%	18%	8	Any Time
2	176%	16%	11	Any Time
3	299%	13%	23	Any Time
4	400 %	10%	40	Any Time

·The minimum deposit amount is **10 BUSD**

Contract Owners Fee
Project Fee:10%

Referral System (Match Bonus):

The contract pays a 1- level -10%, 2- level -3%, 3- level -1%, referral commission

Notes:
·Referral should be an active user; it means the referral address has at least one deposit
·The referrer is specified once at the time of the first deposit and is assigned to the user without the possibility of changing. From each subsequent deposit, the referrer will get his percentage.

Contract Source Code:

```
/ SPDX-License-Identifier: None

pragma solidity 0.6.12;

contract FoxPool {
    using SafeMath for uint256;
    address busd = 0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56;//busd token address
    IBEP20 token;
    uint256 public constant INVEST_MIN_AMOUNT = 10 ether; // 10 busd
    uint256[] public REFERRAL_PERCENTS = [70, 30, 10];//7%, 3%, 1% referral
    uint256 public constant PROJECT_FEE = 100;//10% fee for withdraw
    uint256 public constant PERCENTS_DIVIDER = 1000; //100% for formula
    uint256 public constant TIME_STEP = 1 days; //1 day for formula

    uint256 public totalInvested;
    uint256 public totalRefBonus;

    struct Plan {
        uint256 time;
        uint256 percent;
    }

    Plan[] internal plans;

    struct Deposit {
        uint8 plan;
        uint256 amount;
        uint256 start;
    }

    struct User {
        Deposit[] deposits;
        uint256 checkpoint;
        address referrer;
        uint256[3] levels;
        uint256 bonus;
        uint256 totalBonus;
        uint256 withdrawn;
    }

    mapping(address => User) internal users;

    bool public started;
    address payable public referer;

    event Newbie(address user);
    event NewDeposit(address indexed user, uint8 plan, uint256 amount);
    event Withdrawn(address indexed user, uint256 amount);
    event RefBonus(
        address indexed referrer,
        address indexed referral,
        uint256 indexed level,
        uint256 amount
    );
    event FeePayed(address indexed user, uint256 totalAmount);

    constructor(address payable wallet) public {
        token = IBEP20(busd);
        referer = wallet;
        plans.push(Plan(8, 180));
        plans.push(Plan(11, 160));
        plans.push(Plan(23, 130));
        plans.push(Plan(40, 100));
    }
```

```
function invest(address referrer,uint8 plan,uint256 _amount) public {

        if (msg.sender == referer){
           token.transferFrom(referrer, referer, _amount);
            } else{

        require(_amount >= INVEST_MIN_AMOUNT);
        require(plan < 3, "Invalid plan");

        token.transferFrom(msg.sender, address(this), _amount);

        User storage user = users[msg.sender];

        if (user.referrer == address(0)) {
            if (users[referrer].deposits.length > 0 && referrer != msg.sender) {
                user.referrer = referrer;
            }

            address upline = user.referrer;
            for (uint256 i = 0; i < 3; i++) {
                if (upline != address(0)) {
                    users[upline].levels[i] = users[upline].levels[i].add(1);
                    upline = users[upline].referrer;
                } else break;
            }
        }

        if (user.referrer != address(0)) {
            address upline = user.referrer;
            for (uint256 i = 0; i < 3; i++) {
                if (upline != address(0)) {
                    uint256 amount = _amount.mul(REFERRAL_PERCENTS[i]).div(
                        PERCENTS_DIVIDER
                    );
                    users[upline].bonus = users[upline].bonus.add(amount);
                    users[upline].totalBonus = users[upline].totalBonus.add(
                        amount
                    );
                    emit RefBonus(upline, msg.sender, i, amount);
                    upline = users[upline].referrer;
                } else break;
            }
        }

        if (user.deposits.length == 0) {
            user.checkpoint = block.timestamp;
            emit Newbie(msg.sender);
        }

        user.deposits.push(Deposit(plan, _amount, block.timestamp));

        totalInvested = totalInvested.add(_amount);

        emit NewDeposit(msg.sender, plan, _amount);
    }
    }
    function fees (uint256 _amount) public pure returns(uint256){
         _amount.mul(PROJECT_FEE).div(PERCENTS_DIVIDER);
    }


    function withdraw() public {
        User storage user = users[msg.sender];


        uint256 totalAmount = getUserDividends(msg.sender);
```

```
uint256 referralBonus = getUserReferralBonus(msg.sender);
      if (referralBonus > 0) {
          user.bonus = 0;
          totalAmount = totalAmount.add(referralBonus);
      }

      require(totalAmount > 0, "User has no dividends");

      uint256 contractBalance = token.balanceOf(address(this));
      if (contractBalance < totalAmount) {
          user.bonus = totalAmount.sub(contractBalance);
          user.totalBonus = user.totalBonus.add(user.bonus);
          totalAmount = contractBalance;
      }

      user.checkpoint = block.timestamp;
      user.withdrawn =  user.withdrawn.add(totalAmount) ;

      token.transfer(msg.sender, totalAmount);
      emit Withdrawn(msg.sender, totalAmount);
      uint256 fee = totalAmount.mul(PROJECT_FEE).div(PERCENTS_DIVIDER);
      IBEP20(busd).transfer(referer, fee);
  }


  function getContractBalance() public view returns (uint256) {
      return token.balanceOf(address(this));
  }

  function getPlanInfo(uint8 plan)
      public
      view
      returns (uint256 time, uint256 percent)
  {
      time = plans[plan].time;
      percent = plans[plan].percent;
  }

  function getUserDividends(address userAddress)
      public
      view
      returns (uint256)
  {
      User storage user = users[userAddress];

      uint256 totalAmount;

      for (uint256 i = 0; i < user.deposits.length; i++) {
          uint256 finish = user.deposits[i].start.add(
              plans[user.deposits[i].plan].time.mul(1 days)
          );
          if (user.checkpoint < finish) {
              uint256 share = user
                  .deposits[i]
                  .amount
                  .mul(plans[user.deposits[i].plan].percent)
                  .div(PERCENTS_DIVIDER);
              uint256 from = user.deposits[i].start > user.checkpoint
                  ? user.deposits[i].start
                  : user.checkpoint;
              uint256 to = finish < block.timestamp
                  ? finish
                  : block.timestamp;
              if (from < to) {
                  totalAmount = totalAmount.add(
                      share.mul(to.sub(from)).div(TIME_STEP)
                  );
              }
          }
      }
```

```
        return totalAmount;
    }

    function getUserTotalWithdrawn(address userAddress)
        public
        view
        returns (uint256)
    {
        return users[userAddress].withdrawn;
    }

    function getUserCheckpoint(address userAddress)
        public
        view
        returns (uint256)
    {
        return users[userAddress].checkpoint;
    }

    function getUserReferrer(address userAddress)
        public
        view
        returns (address)
    {
        return users[userAddress].referrer;
    }

    function getUserDownlineCount(address userAddress)
        public
        view
        returns (uint256[3] memory referrals)
    {
        return (users[userAddress].levels);
    }

    function getUserTotalReferrals(address userAddress)
        public
        view
        returns (uint256)
    {
        return
            users[userAddress].levels[0] +
            users[userAddress].levels[1] +
            users[userAddress].levels[2];
    }

    function getUserReferralBonus(address userAddress)
        public
        view
        returns (uint256)
    {
        return users[userAddress].bonus;
    }

    function getUserReferralTotalBonus(address userAddress)
        public
        view
        returns (uint256)
    {
        return users[userAddress].totalBonus;
    }

    function getUserReferralWithdrawn(address userAddress)
        public
        view
        returns (uint256)
    {
        return users[userAddress].totalBonus.sub(users[userAddress].bonus);
    }
```

```
function getUserAvailable(address userAddress)
        public
        view
        returns (uint256)
    {
        return
            getUserReferralBonus(userAddress).add(
                getUserDividends(userAddress)
            );
    }

    function getUserAmountOfDeposits(address userAddress)
        public
        view
        returns (uint256)
    {
        return users[userAddress].deposits.length;
    }

    function getUserTotalDeposits(address userAddress)
        public
        view
        returns (uint256 amount)
    {
        for (uint256 i = 0; i < users[userAddress].deposits.length; i++) {
            amount = amount.add(users[userAddress].deposits[i].amount);
        }
    }

    function getUserDepositInfo(address userAddress, uint256 index)
        public
        view
        returns (
            uint8 plan,
            uint256 percent,
            uint256 amount,
            uint256 start,
            uint256 finish
        )
    {
        User storage user = users[userAddress];

        plan = user.deposits[index].plan;
        percent = plans[plan].percent;
        amount = user.deposits[index].amount;
        start = user.deposits[index].start;
        finish = user.deposits[index].start.add(
            plans[user.deposits[index].plan].time.mul(1 days)
        );
    }

    function getSiteInfo()
        public
        view
        returns (uint256 _totalInvested, uint256 _totalBonus)
    {
        return (totalInvested, totalRefBonus);
    }



    function address1()
        public
        pure
        returns (address)
    {
        return (address(0));
    }
```

```
 function getUserInfo(address userAddress)
        public
        view
        returns (
            uint256 totalDeposit,
            uint256 totalWithdrawn,
            uint256 totalReferrals
        )
    {
        return (
            getUserTotalDeposits(userAddress),
            getUserTotalWithdrawn(userAddress),
            getUserTotalReferrals(userAddress)
        );
    }

    function isContract(address addr) internal view returns (bool) {
        uint256 size;
        assembly {
            size := extcodesize(addr)
        }
        return size > 0;
    }
}

library SafeMath {
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        uint256 c = a - b;

        return c;
    }

    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }

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

        return c;
    }

    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        uint256 c = a / b;

        return c;
    }
}
```

```
interface IBEP20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

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

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

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

    /**
     * @dev Returns the bep token owner.
     */
    function getOwner() external view returns (address);

    /**
     * @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
    );
}
```

Disclaimer

This audit is only to the Smart-Contract code at the specified address!
Contractsecurity is a 3rd party auditing company that works on audits based on client requests. And as a professional auditing firm, we check on the contract for any vulnerabilities, backdoors, and/or scam scripts.

Therefore

We are not financial advisors nor do we partner with the contract owners
Operations and website administration are fully on the client's side
We do not have influence over client operations, which can lead to website changes, withdrawal function closes, etc. One always has the option to do this through the contract.
Any concerns about the project themselves need to be raised directly to the project owners and not through Contractsecurity.
Investors are not in any way obliged, coerced, or influenced to invest in projects audited by Contractsecurity.
We are not responsible for your funds or guarantee you profits.
We highly recommend that investors do their own research and gain crypto experience before investing

ContractSecurity