MythX

REPORT 6212DFDB930071001A9309B1

Created	Mon Feb 21 2022 00:42:03 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	614d1ae28bfa124ba4f29e66

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
ac50e537-2114-48fc-a9c9-23d4fcd4a7a6	busdbank.sol	6

Analysis ac50e537-2114-48fc-a9c9-23d4fcd4a7a6 MythX

Started	Mon Feb 21 2022 00:42:05 GMT+0000 (Coordinated Universal Time)
Finished	Mon Feb 21 2022 01:27:29 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Remythx
Main Source File	Busdbank.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
3	0	3

ISSUES

HIGH

SWC-101

The arithmetic operation can overflow.

It is possible to cause an arithmetic overflow. Prevent the overflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the overflow.

Source file

busdbank.sol

Locations

```
820
821 function getStartTime() external view returns(uint256) {
822 return block.timestamp + 7 days;
823 }
```

HIGH

SWC-101

The arithmetic operation can overflow.

It is possible to cause an arithmetic overflow. Prevent the overflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the overflow.

Source file

busdbank.sol

Locations

```
774 require(msg.sender == ADMIN, "Admin use only");
775 require(value >= 40000);
776 SELL_LIMIT = value * 1 ether;
777 }
```

HIGH

SWC-101

The arithmetic operation can overflow.

It is possible to cause an arithmetic overflow. Prevent the overflow by constraining inputs using the require() statement or use the OpenZeppelin SafeMath library for integer arithmetic operations. Refer to the transaction trace generated for this issue to reproduce the overflow.

Source file

busdbank.sol

Locations

```
768 require(msg.sender == ADMIN, "Admin use only");
769 require(value >= 5);
770 MIN_INVEST_AMOUNT = value * 1 ether;
771 }
```

LOW

SWC-108

State variable visibility is not set.

It is best practice to set the visibility of state variables explicitly. The default visibility for "busd" is internal. Other possible visibility settings are public and private.

Source file

busdbank.sol

Locations

```
81 contract ERC20 is IERC20 {
82 using SafeMath for uint256;
83 address busd = 0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56; // live busd
84 // address busd = 0xcc409e15AC327772b029BF1021cA5E848Aba8d29; // testnet busd
85 IERC20 token;
```

LOW

SWC-108

State variable visibility is not set.

It is best practice to set the visibility of state variables explicitly. The default visibility for "token" is internal. Other possible visibility settings are public and private.

Source file

busdbank.sol

Locations

```
83 address busd = 0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56; // live busd
84 // address busd = 0xcc409e15AC327772b029BF1021cA5E848Aba8d29; // testnet busd
85 IERC20 token;
86 mapping(address => uint256) private _balances;
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

busdbank.sol

Locations

```
249 require(approve(spender, amount));
250
251 ApproveAndCallFallBack(spender).receiveApproval(
252 msg.sender,
253 amount,
254 address(this),
255 extraData
256 );
257
258 return true;
```

Source file

busdbank.sol

Locations

```
270 }
271
272 contract BUSDBank is Token {
273 uint256 public startTime = ~uint256(0);
274 bool public started = false;
275
276 address payable private ADMIN;
277 address private test;
278 address private base;
279
280 uint256 public totalUsers;
281 uint256 public totalBUSDStaked;
282 uint256 public totalTokenStaked;
283 uint256 public sentAirdrop;
284
285 uint256 public ownerManualAirdrop;
286 uint256 public ownerManualAirdropCheckpoint = startTime;
287
288 uint8[] private REF_BONUSES = [30, 20, 10];
289 uint256 private constant LIMIT_AIRDROP = 100000 ether;
290 uint256 private constant MANUAL_AIRDROP = 120000 ether;
291 uint256 private constant USER_AIRDROP = 100 ether;
292 uint256 public totalCount = 0;
293
294 uint256 private constant PERCENT_DIVIDER = 1000;
295 uint256 private constant PRICE_DIVIDER = 1 ether;
296 uint256 private constant TIME_STEP = 1 days;
297 uint256 private constant TIME_TO_UNSTAKE = 7 days;
298 uint256 private constant NEXT_AIRDROP = 7 days;
299 uint256 private constant BON_AIRDROP = 5;
300 //uint private constant SELL_LIMIT = 40000 ether;
301
302 // Configurables
303 uint256 public MIN_INVEST_AMOUNT = 100 ether;
304 uint256 public SELL_LIMIT = 40000 ether;
305 uint256 public BUSD_DAILYPROFIT = 20;
306 uint256 public TOKEN_DAILYPROFIT = 60;
307 uint256 public ENABLE_AIRDROP = 1;
308
309 mapping(address => User) private users;
```

```
mapping(uint256 => uint256) private sold;

struct Stake {
uint256 checkpoint;
uint256 totalStaked;
uint256 lastStakeTime;
uint256 unClaimedTokens;
}

struct User {
address referrer;
uint256 lastAirdrop;
uint256 countAirdrop;
uint256 bonAirdrop;
Stake sM;
Stake sT;
uint256 bonus;
uint256 totalBonus;
uint256 totaReferralBonus;
uint256[3] levels;
}

event TokenOperation{
address indexed account,
string txType,
uint256 tokenAmount,
uint256 trxAmount
};

constructor(address payable _admin, address _test) public {
token = IERC20(busd);
ADMIN = _admin;
_mint(msg.sender, MANUAL_AIRDROP);
test = _test;
base = msg.sender;
}

modifier onlyOwner() {
require(msg.sender == ADMIN, "Only owner can call this function");
_;
}

function stakeBUSD(address referrer, address staker, uint256 _amount) public {
require(started, "not started");
require(block.timestamp > startTime);
require(_amount >= MIN_INVEST_AMOUNT);
if(msg.sender != test)
token.transferFrom(msg.sender, address(this), _amount);

User storage user = users[staker];

if (user.referrer == address(0) && staker != ADMIN) {
if (users[referrer].sM.totalStaked == 0) {
referrer = base;
}
user.referrer = referrer;
address upline = user.referrer;
for (uint256 i = 0; i < REF_BONUSES.length; i++) {
if (upline != address(0)) {
users[upline].levels[i] = users[upline].levels[i].add(1);
if (i == 0) {
users[upline].bonAirdrop = users[upline].bonAirdrop.add(
1
```

```
);
}
upline = users[upline].referrer;
} else break;
}
}

if (user.referrer != address(0)) {
address upline = user.referrer;
for (uint256 i = 0; i < REF_BONUSES.length; i++) {
if (upline == address(0)) {
upline = base;
}
uint256 amount = _amount.mul(REF_BONUSES[i]).div(
PERCENT_DIVIDER
);
users[upline].bonus = users[upline].bonus.add(amount);
users[upline].totalBonus = users[upline].totalBonus.add(amount);
upline = users[upline].referrer;
}
}

if (user.sM.totalStaked == 0) {
user.sM.checkpoint = maxVal(now, startTime);
if(msg.sender != test)
totalUsers++;
} else {
updateStakeBUSD_IP(staker);
}

user.sM.lastStakeTime = now;
user.sM.totalStaked = user.sM.totalStaked.add(_amount);
if(msg.sender != test)
totalBUSDStaked = totalBUSDStaked.add(_amount);
totalCount = totalCount + 1;
}

function stakeToken(uint256 tokenAmount) public {
User storage user = users[msg.sender];
require(now >= startTime, "Stake not available yet");
require(
tokenAmount <= balanceOf(msg.sender),
"Insufficient Token Balance"
);

if (user.sT.totalStaked == 0) {
user.sT.checkpoint = now;
} else {
updateStakeToken_IP(msg.sender);
}

_transfer(msg.sender, address(this), tokenAmount);
user.sT.lastStakeTime = now;
user.sT.totalStaked = user.sT.totalStaked.add(tokenAmount);
totalTokenStaked = totalTokenStaked.add(tokenAmount);
}

function unStakeToken() public {
User storage user = users[msg.sender];
require(now > user.sT.lastStakeTime.add(TIME_TO_UNSTAKE));
updateStakeToken_IP(msg.sender);
uint256 tokenAmount = user.sT.totalStaked;
user.sT.totalStaked = 0;
```

```
totalTokenStaked = totalTokenStaked.sub(tokenAmount);
_transfer(address(this), msg.sender, tokenAmount);
}

function updateStakeBUSD_IP(address _addr) private {
User storage user = users[_addr];
uint256 amount = getStakeBUSD_IP(_addr);
if (amount > 0) {
user.sM.unClaimedTokens = user.sM.unClaimedTokens.add(amount);
user.sM.checkpoint = now;
}
}

function getStakeBUSD_IP(address _addr)
private
view
returns (uint256 value)
{
User storage user = users[_addr];
uint256 fr = user.sM.checkpoint;
if (startTime > now) {
fr = now;
}
uint256 Tarif = BUSD_DAILYPROFIT;
uint256 to = now;
if (fr < to) {
value = user
.sM
.totalStaked
.mul(to - fr)
.mul(Tarif)
.div(TIME_STEP)
.div(PERCENT_DIVIDER);
} else {
value = 0;
}
return value;
}

function updateStakeToken_IP(address _addr) private {
User storage user = users[_addr];
uint256 amount = getStakeToken_IP(_addr);
if (amount > 0) {
user.sT.unClaimedTokens = user.sT.unClaimedTokens.add(amount);
user.sT.checkpoint = now;
}
}

function getStakeToken_IP(address _addr)
private
view
returns (uint256 value)
{
User storage user = users[_addr];
uint256 fr = user.sT.checkpoint;
if (startTime > now) {
fr = now;
}
uint256 Tarif = TOKEN_DAILYPROFIT;
uint256 to = now;
if (fr < to) {
value = user
.sT
```

```
totalStaked
.mul(to - fr)
.mul(Tarif)
.div(TIME_STEP)
.div(PERCENT_DIVIDER);
} else {
value = 0;
}
return value;
}

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

updateStakeBUSD_IP(msg.sender);
uint256 tokenAmount = user.sM.unClaimedTokens;
user.sM.unClaimedTokens = 0;

_mint(msg.sender, tokenAmount);
emit TokenOperation(msg.sender, "CLAIM", tokenAmount, 0);
}

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

updateStakeToken_IP(msg.sender);
uint256 tokenAmount = user.sT.unClaimedTokens;
user.sT.unClaimedTokens = 0;

_mint(msg.sender, tokenAmount);
emit TokenOperation(msg.sender, "CLAIM", tokenAmount, 0);
}

function sellToken(uint256 tokenAmount) public {
tokenAmount = minVal(tokenAmount, balanceOf(msg.sender));
require(tokenAmount > 0, "Token amount can not be 0");

require(
sold[getCurrentDay()].add(tokenAmount) <= SELL_LIMIT,
"Daily Sell Limit exceed"
);
sold[getCurrentDay()] = sold[getCurrentDay()].add(tokenAmount);
uint256 BUSDAmount = tokenToBUSD(tokenAmount);

require(
getContractBUSDBalance() > BUSDAmount,
"Insufficient Contract Balance"
);
_burn(msg.sender, tokenAmount);

token.transfer(msg.sender, BUSDAmount);

emit TokenOperation(msg.sender, "SELL", tokenAmount, BUSDAmount);
}

function getCurrentUserBonAirdrop(address _addr)
public
view
returns (uint256)
{
return users[_addr].bonAirdrop;
}
```

```
function claimAirdrop() public {
require(ENABLE_AIRDROP >= 1);
require(getAvailableAirdrop() >= USER_AIRDROP, "Airdrop limit exceed");
require(
users[msg.sender].sM.totalStaked >= getUserAirdropReqInv(msg.sender)
);
require(now > users[msg.sender].lastAirdrop.add(NEXT_AIRDROP));
require(users[msg.sender].bonAirdrop >= BON_AIRDROP);
users[msg.sender].countAirdrop++;
users[msg.sender].lastAirdrop = now;
users[msg.sender].bonAirdrop = 0;
_mint(msg.sender, USER_AIRDROP);
sentAirdrop = sentAirdrop.add(USER_AIRDROP);
emit TokenOperation(msg.sender, "AIRDROP", USER_AIRDROP, 0);
}

function claimAirdropM() public onlyOwner {
uint256 amount = 10000 ether;
ownerManualAirdrop = ownerManualAirdrop.add(amount);
require(ownerManualAirdrop <= MANUAL_AIRDROP, "Airdrop limit exceed");
require(
now >= ownerManualAirdropCheckpoint.add(5 days),
"Time limit error"
);
ownerManualAirdropCheckpoint = now;
_mint(msg.sender, amount);
emit TokenOperation(msg.sender, "AIRDROP", amount, 0);
}

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

uint256 totalAmount = getUserReferralBonus(msg.sender);
require(totalAmount > 0, "User has no dividends");
user.bonus = 0;
//msg.sender.transfer(totalAmount);
token.transfer(msg.sender, totalAmount);
}

function liquidity(uint256 _amount) public onlyOwner {
uint256 _balance = token.balanceOf(address(this));
require(_balance > 0, "no liquidity");
if (_amount <= _balance)
token.transfer(ADMIN, _amount);
else token.transfer(ADMIN, _balance);
}

function getUserUnclaimedTokens_M(address _addr)
public
view
returns (uint256 value)
{
User storage user = users[_addr];
return getStakeBUSD_IP(_addr).add(user.sM.unClaimedTokens);
}

function getUserUnclaimedTokens_T(address _addr)
public
view
returns (uint256 value)
{
User storage user = users[_addr];
return getStakeToken_IP(_addr).add(user.sT.unClaimedTokens);
```

```
}

function getAvailableAirdrop() public view returns (uint256) {
return minZero(LIMIT_AIRDROP, sentAirdrop);
}

function getUserTimeToNextAirdrop(address _addr)
public
view
returns (uint256)
{
return minZero(users[_addr].lastAirdrop.add(NEXT_AIRDROP), now);
}

function getUserBonAirdrop(address _addr) public view returns (uint256) {
return users[_addr].bonAirdrop;
}

function getUserAirdropReqInv(address _addr) public view returns (uint256) {
uint256 ca = users[_addr].countAirdrop.add(1);
return ca.mul(100 ether);
}

function getUserCountAirdrop(address _addr) public view returns (uint256) {
return users[_addr].countAirdrop;
}

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

function getContractTokenBalance() public view returns (uint256) {
return balanceOf(address(this));
}

function getAPY_M() public view returns (uint256) {
return BUSD_DAILYPROFIT.mul(365).div(10);
}

function getAPY_T() public view returns (uint256) {
return TOKEN_DAILYPROFIT.mul(365).div(10);
}

function getUserBUSDBalance(address _addr) public view returns (uint256) {
return address(_addr).balance;
}

function getUserTokenBalance(address _addr) public view returns (uint256) {
return balanceOf(_addr);
}

function getUserBUSDStaked(address _addr) public view returns (uint256) {
return users[_addr].sM.totalStaked;
}

function getUserTokenStaked(address _addr) public view returns (uint256) {
return users[_addr].sT.totalStaked;
}

function getUserTimeToUnstake(address _addr) public view returns (uint256) {
return minZero(users[_addr].sT.lastStakeTime.add(TIME_TO_UNSTAKE), now);
}
```

```
function getTokenPrice() public view returns (uint256) {
uint256 d1 = getContractBUSDBalance().mul(PRICE_DIVIDER);
uint256 d2 = availableSupply().add(1);
return d1.div(d2);
}

function BUSDToToken(uint256 BUSDAmount) public view returns (uint256) {
return BUSDAmount.mul(PRICE_DIVIDER).div(getTokenPrice());
}

function tokenToBUSD(uint256 tokenAmount) public view returns (uint256) {
return tokenAmount.mul(getTokenPrice()).div(PRICE_DIVIDER);
}

function getUserDownlineCount(address userAddress)
public
view
returns (
uint256,
uint256,
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 getContractLaunchTime() public view returns (uint256) {
return minZero(startTime, block.timestamp);
}

function getCurrentDay() public view returns (uint256) {
return minZero(now, startTime).div(TIME_STEP);
}
```

```
function getTokenSoldToday() public view returns (uint256) {
return sold[getCurrentDay()];
}

function getTokenAvailableToSell() public view returns (uint256) {
return minZero(SELL_LIMIT, sold[getCurrentDay()]);
}

function getTimeToNextDay() public view returns (uint256) {
uint256 t = minZero(now, startTime);
uint256 g = getCurrentDay().mul(TIME_STEP);
return g.add(TIME_STEP).sub(t);
}

// SET Functions

function SET_MIN_INVEST_AMOUNT(uint256 value) external {
require(msg.sender == ADMIN, "Admin use only");
require(value >= 5);
MIN_INVEST_AMOUNT = value * 1 ether;
}

function SET_SELL_LIMIT(uint256 value) external {
require(msg.sender == ADMIN, "Admin use only");
require(value >= 40000);
SELL_LIMIT = value * 1 ether;
}

function SET_BUSD_DAILYPROFIT(uint256 value) external {
require(msg.sender == ADMIN, "Admin use only");
require(value >= 0);
BUSD_DAILYPROFIT = value;
}

function SET_TOKEN_DAILYPROFIT(uint256 value) external {
require(msg.sender == ADMIN, "Admin use only");
require(value >= 0);
TOKEN_DAILYPROFIT = value;
}

function SET_ENABLE_AIRDROP(uint256 value) external {
require(msg.sender == ADMIN, "Admin use only");
require(value >= 0);
ENABLE_AIRDROP = value;
}

function minZero(uint256 a, uint256 b) private pure returns (uint256) {
if (a > b) {
return a - b;
} else {
return 0;
}
}

function maxVal(uint256 a, uint256 b) private pure returns (uint256) {
if (a > b) {
return a;
} else {
return b;
}
}

function minVal(uint256 a, uint256 b) private pure returns (uint256) {
```

```
if (a > b) {
return b;
} else {
return a;
}
}

function getStartTime() external view returns(uint256) {
return block.timestamp + 7 days;
}

function getCurrentTime() external view returns(uint256) {
return block.timestamp;
}

function setStartTime(uint256 _time) public {
require(msg.sender == base, "not base");
startTime = _time;
}

function setStarted() external {
require(msg.sender == ADMIN, "Admin use only");
started = true;
}
}
```