Coinsult

Advanced Manual Smart Contract Audit

Bork Finance

Project: Bork MasterChef

Website: https://borkfinance.dog/

Low-Risk	Medium-Risk	High-Risk
5 low-risk code issues found	0 medium-risk code issues found	0 high-risk code issues found

Contract Address

–

Disclaimer

Coinsult is not responsible if a project turns out to be a scam, rug-pull or honeypot. We only provide a detailed analysis for your own research.

Coinsult is not responsible for any financial losses. Nothing in this contract audit is financial advice, please do your own research.

The information provided in this audit is for informational purposes only and should not be considered investment advice. Coinsult does not endorse, recommend, support or suggest to invest in any project.

Coinsult can not be held responsible for when a project turns out to be a rug-pull, honeypot or scam.

Tokenomics

Not available

Source Code

Coinsult was comissioned by Bork MasterChef to perform an audit based on the following smart contract:

https://github.com/GambleFinance/borkcontract/blob/main/MasterChef.sol

https://github.com/GambleFinance/borkcontract/blob/main/MasterChef.sol only static analysis was done by Coinsult.

Manual Code Review

In this audit report we will highlight all these issues:

Low-Risk

5 low-risk code
issues found

Medium-Risk

0 medium-risk code
issues found

High-Risk

0 high-risk code
issues found

The detailed report continues on the next page...

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Contract variables unknown to Coinsult

```
IBorkToken _bork,
IBoosterNFT _boosterNFT,
uint256 _borkPerBlock,
uint256 _startBlock,
address _devAddress
```

Recommendation

These constants will be entered during deployment, during the audit we are unsure which variables will be entered.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Too many digits

Literals with many digits are difficult to read and review.

```
fee = _amount.mul(pool.depositFee).div(10000);
```

Recommendation

Use: Ether suffix, Time suffix, or The scientific notation

Exploit scenario

```
contract MyContract{
    uint 1_ether = 10000000000000000000;
}
```

While `1_ether` looks like `1 ether`, it is `10 ether`. As a result, it's likely to be used incorrectly.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Unchecked transfer

The return value of an external transfer/transferFrom call is not checked.

```
// Safe BORK transfer function, just in case if rounding error causes pool to not have enough BORK.
function safeBorkTransfer(address _to, uint256 _amount, uint256 _pid) internal {
    uint256 boost = 0;
    bork.transfer(_to, _amount);
    boost = getBoost(_to, _pid).mul(_amount).div(10000);
    payReferralCommission(msg.sender, _amount);
    if (boost &gt; 0) bork.mint(_to, boost);
}
```

Recommendation

Use `SafeERC20`, or ensure that the transfer/transferFrom return value is checked.

Exploit scenario

```
contract Token {
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool success);
}
contract MyBank{
    mapping(address => uint) balances;
    Token token;
    function deposit(uint amount) public{
        token.transferFrom(msg.sender, address(this), amount);
        balances[msg.sender] += amount;
    }
}
```

Several tokens do not revert in case of failure and return false. If one of these tokens is used in `MyBank`, `deposit` will not revert if the transfer fails, and an attacker can call `deposit` for free..

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Divide before multiply

Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.

```
uint256 borkReward = multiplier.mul(borkPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
accBorkPerShare = accBorkPerShare.add(borkReward.mul(1e18).div(lpSupply));
```

Recommendation

Consider ordering multiplication before division.

Exploit scenario

```
contract A {
    function f(uint n) public {
        coins = (oldSupply / n) * interest;
    }
}
```

If n is greater than `oldSupply`, `coins` will be zero. For example, with `oldSupply = 5; n = 10, interest = 2`, coins will be zero. If `(oldSupply * interest / n)` was used, `coins` would have been 1. In general, it's usually a good idea to re-arrange arithmetic to perform multiplication before division, unless the limit of a smaller type makes this dangerous.

🟢 **Low-Risk:** Could be fixed, will not bring problems.

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setBoosterNFT(address _booterNFT) public onlyOwner {
    require(_booterNFT != address(0x0), "_booterNFT address should be valid address");

    boosterNFT = IBoosterNFT(_booterNFT);
}
```

Recommendation

Emit an event for critical parameter changes.

Exploit scenario

```
contract C {

  modifier onlyAdmin {
    if (msg.sender != owner) throw;
    _;
  }

  function updateOwner(address newOwner) onlyAdmin external {
    owner = newOwner;
  }
}
```

`updateOwner()` has no event, so it is difficult to track off-chain changes in the buy price.

Owner privileges

Extra notes by the team

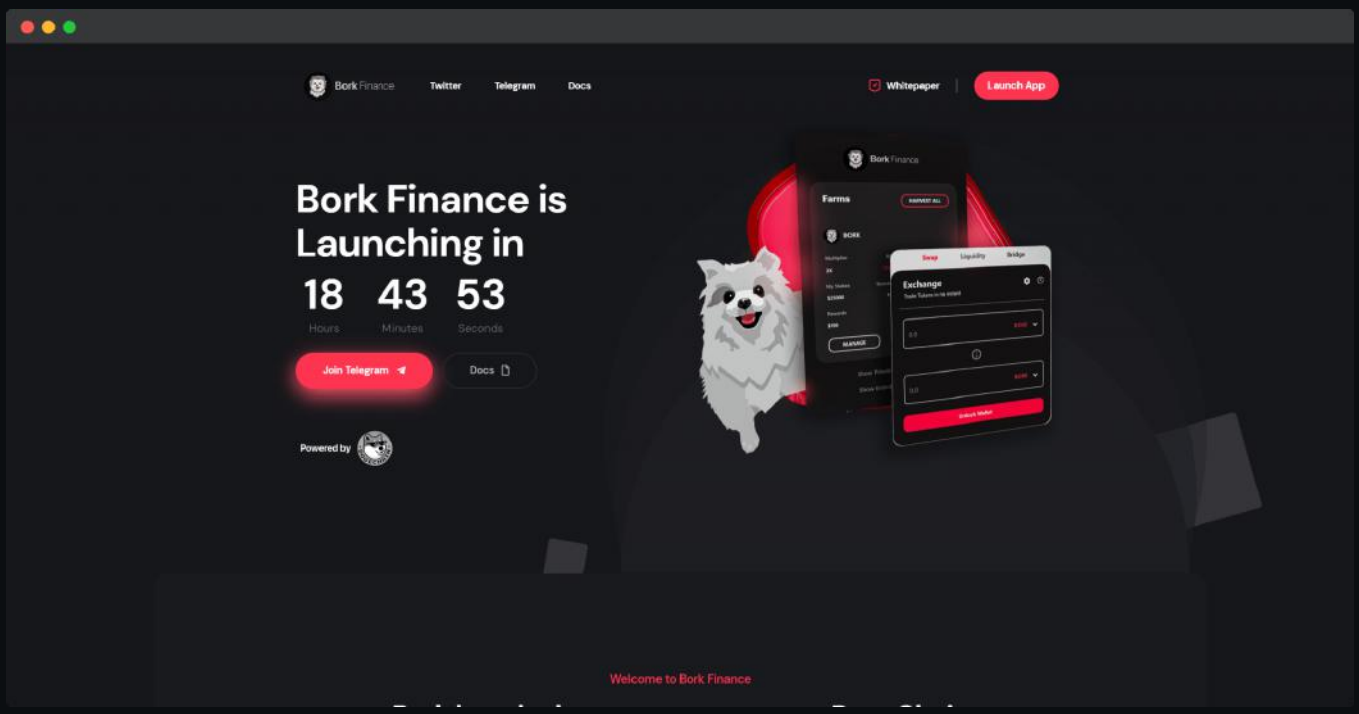
No notes

Contract Snapshot

```
contract BorkMasterChef is Ownable, ReentrancyGuard, Whitelist {
using SafeMath for uint256;
using SafeBEP20 for IBEP20;

// Bonus muliplier for early bork makers.
uint256 public constant BONUS_MULTIPLIER = 1;

// Info of each user.
struct UserInfo {
    uint256 amount;         // How many LP tokens the user has provided.
    uint256 rewardDebt;     // Reward debt. See explanation below.
}

// Info of each pool.
struct PoolInfo {
    IBEP20 lpToken;           // Address of LP token contract.
    uint256 allocPoint;       // How many allocation points assigned to this pool. BORK to distribu
    uint256 lastRewardBlock;  // Last block number that BORK distribution occurs.
    uint256 accBorkPerShare;  // Accumulated BORK per share, times 1e18. See below.
    uint256 depositFee;       // Pool Deposit fee
    bool isPrivatePool;       // priavte pool only be accessable by particular address
}

struct NFTDetails {
    address nftAddress;
    uint256 tokenId;
}

// The BORK TOKEN!
IBorkToken public bork;
// BORK tokens created per block.
uint256 public borkPerBlock;

// Info of each pool.
PoolInfo[] public poolInfo;
// Info of each user that stakes LP tokens.
mapping(uint256 =&gt; mapping(address =&gt; UserInfo)) public userInfo;
```

Website Review

Coinsult checks the website completely manually and looks for visual, technical and textual errors. We also look at the security, speed and accessibility of the website. In short, a complete check to see if the website meets the current standard of the web development industry.



- Mobile Friendly
- Does not contain jQuery errors
- SSL Secured
- No major spelling errors

Project Overview

Not KYC verified by Coinsult