Coinsult

Advanced Manual Smart Contract Audit

Project: Private: Naughty Dog

Website: https://naughtydog.vip/

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

Contract Address

0x27df76451d690643ADbfa920C03D6975189697b4

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

Rank	Address	Quantity (Token)	Percentage
1	0x0ec23376e853bd463502b1cd75886184c9440ffa	7,203,839,935.68	72.0384%
2	0x89baf5715bad7f1f0343f6652e77363b9eb83244	2,796,160,064.32	27.9616%

Source Code

Coinsult was comissioned by Private: Naughty Dog to perform an audit based on the following smart contract:

https://bscscan.com/address/0x27df76451d690643ADbfa920C03D6975189697b4#code

Manual Code Review

In this audit report we will highlight all these issues:

🟢 Low-Risk

6 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 contains Reentrancy vulnerabilities

Additional information: This combination increases risk of malicious intent. While it may be justified by some complex mechanics (e.g. rebase, reflections, buyback).

More information: Slither

```
function _transfer(address sender, address recipient, uint256 amount) private returns (bool) {

    require(sender != address(0), "ERC20: transfer from the zero address");
    require(recipient != address(0), "ERC20: transfer to the zero address");

    if(inSwapAndLiquify)
    {
        return _basicTransfer(sender, recipient, amount);
    }
    else
    {
        uint256 contractTokenBalance = balanceOf(address(this));
        bool overMinimumTokenBalance = contractTokenBalance &gt;= minimumTokensBeforeSwap;

        if (overMinimumTokenBalance &amp;&amp; !inSwapAndLiquify &amp;&amp; !isMarketPair[sender] &a
        {
            if(swapAndLiquifyByLimitOnly)
                contractTokenBalance = minimumTokensBeforeSwap;
            swapAndLiquify(contractTokenBalance);
        }

         balances[sender] =  balances[sender].sub(amount, "Insufficient Balance");
```

Recommendation

Apply the check-effects-interactions pattern.

Exploit scenario

```
function withdrawBalance(){
    // send userBalance[msg.sender] Ether to msg.sender
    // if mgs.sender is a contract, it will call its fallback function
    if( ! (msg.sender.call.value(userBalance[msg.sender])() ) ){
        throw;
    }
    userBalance[msg.sender] = 0;
}
```

Bob uses the re-entrancy bug to call withdrawBalance two times, and withdraw more than its initial deposit to the contract.

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

Avoid relying on block.timestamp

block.timestamp can be manipulated by miners.

```
function lock(uint256 time) public virtual onlyOwner {
    _previousOwner = _owner;
    _owner = address(0);
    _lockTime = block.timestamp + time;
    emit OwnershipTransferred(_owner, address(0));
}
```

Recommendation

Do not use `block.timestamp`, now or `blockhash` as a source of randomness

Exploit scenario

```
contract Game {

    uint reward_determining_number;

    function guessing() external{
      reward_determining_number = uint256(block.blockhash(10000)) % 10;
    }
}
```

Eve is a miner. Eve calls `guessing` and re-orders the block containing the transaction. As a result, Eve wins the game.

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

Too many digits

Literals with many digits are difficult to read and review.

```
_walletMax = 1000000000 * 10 ** 5 * 10 ** 5 * 10 ** _decimals
```

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.

No zero address validation for some functions

Detect missing zero address validation.

```
function setMarketingWalletAddress(address newAddress) external onlyOwner() {
    marketingWalletAddress = payable(newAddress);
}

function setLPAd(address newAddress) external onlyOwner() {
    LPAd = payable(newAddress);
}
```

Recommendation

Check that the new address is not zero.

Exploit scenario

```
contract C {

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

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

Bob calls `updateOwner` without specifying the `newOwner`, soBob loses ownership of the contract.

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

Missing events arithmetic

Detect missing events for critical arithmetic parameters.

```
function setDistributionSettings(uint256 newLiquidityShare, uint256 newMarketingShare, uint256 newLP
    _liquidityShare = newLiquidityShare;
    _marketingShare = newMarketingShare;
    _LPADShare = newLPADShare;

    _totalDistributionShares = _liquidityShare.add(_marketingShare).add(_LPADShare);
}
```

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.

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

Conformance to Solidity naming conventions

Allow _ at the beginning of the mixed_case match for private variables and unused parameters.

```
Function IUniswapV2Pair.DOMAIN_SEPARATOR() (#226) is not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (#227) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (#243) is not in mixedCase
Function IUniswapV2Router01.WETH() (#262) is not in mixedCase
```

Recommendation

Follow the Solidity naming convention.

Rule exceptions

- Allow constant variable name/symbol/decimals to be lowercase (`ERC20`).
- Allow _ at the beginning of the `mixed_case` match for private variables and unused parameters.

Owner privileges

- 🟢 Owner cannot pause trading
- 🟡 Owner can change max transaction amount
- 🟡 Owner can set fees higher than 25%
- 🟡 Owner can exclude from fees
- ⚠️ Owner can set wallet limit
- ⚠️ Owner can exclude from wallet limit

Extra notes by the team

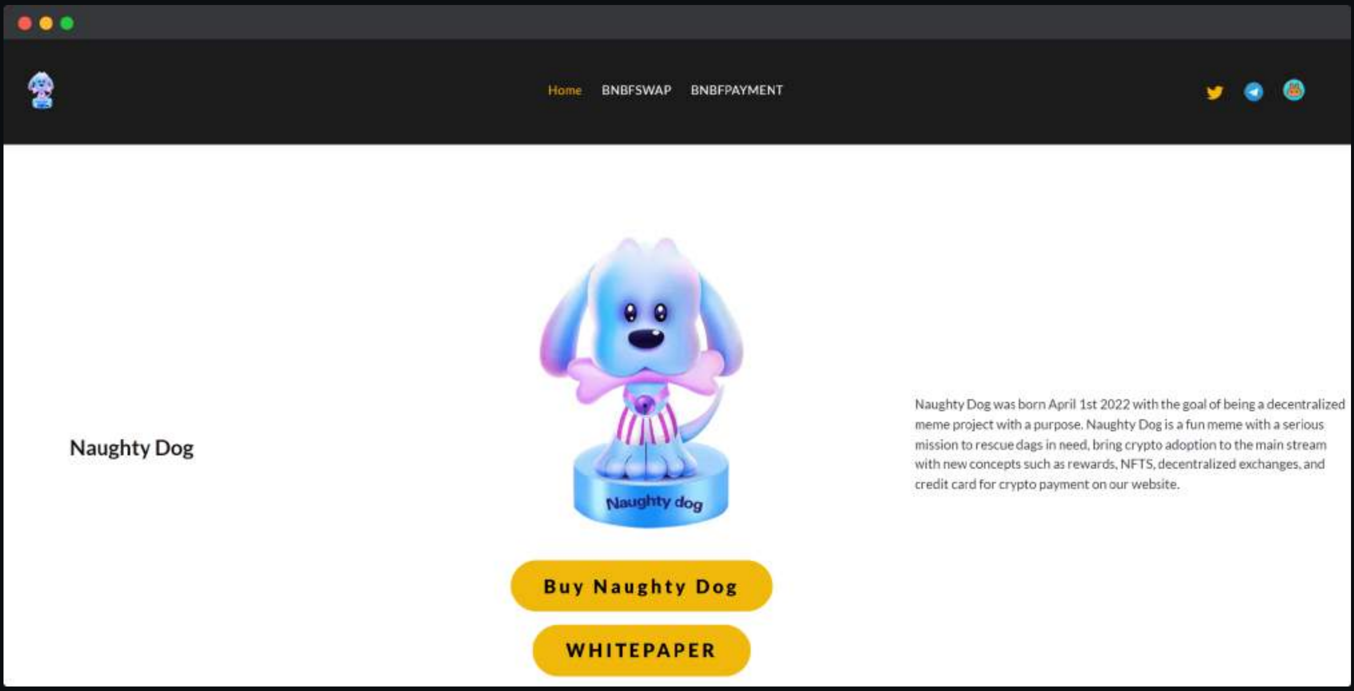
No notes

Contract Snapshot

```
contract NaughtyDoge is Context, IERC20, Ownable {

using SafeMath for uint256;
using Address for address;

string private _name = "Naughty Doge";
string private _symbol = "NDoge";
uint8 private _decimals = 9;

address payable public marketingWalletAddress = payable(0x89baF5715bad7f1f0343f6652e77363B9Eb83244);
address payable public LPAd = payable(0xB3F6E10F9e6705E5863165E6f55D6AdaFBDFE2Ec); // LP Address
address public immutable deadAddress = 0x000000000000000000000000000000000000dEaD;

mapping (address =&gt; uint256) _balances;
mapping (address =&gt; mapping (address =&gt; uint256)) private _allowances;

mapping (address =&gt; bool) public isExcludedFromFee;
mapping (address =&gt; bool) public isWalletLimitExempt;
mapping (address =&gt; bool) public isTxLimitExempt;
mapping (address =&gt; bool) public isMarketPair;

uint256 public _buyLiquidityFee = 1;
uint256 public _buyMarketingFee = 8;
uint256 public _buyLPFee = 1;

uint256 public _sellLiquidityFee = 1;
uint256 public _sellMarketingFee = 8;
uint256 public _sellLPFee = 1 ;

uint256 public _liquidityShare = 2;
uint256 public _marketingShare = 16;
uint256 public _LPADShare = 2;

uint256 public _totalTaxIfBuying = 10;
uint256 public _totalTaxIfSelling = 10;
uint256 public _totalDistributionShares = 20;
```

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.



Naughty Dog

Naughty Dog was born April 1st 2022 with the goal of being a decentralized meme project with a purpose. Naughty Dog is a fun meme with a serious mission to rescue dags in need, bring crypto adoption to the main stream with new concepts such as rewards, NFTS, decentralized exchanges, and credit card for crypto payment on our website.

● Mobile Friendly

● Does not contain jQuery errors

● SSL Secured

● No major spelling errors

Project Overview

● Not KYC verified by Coinsult