DEEPSPACE Audit Report

Prepared by 0xMilenov

Contents

1 About 0xMilenov

Independent Security Researcher

As a software engineer based in Bulgaria, 0xMilenov is a passionate expert in blockchain security, with a keen focus on Web3, smart contracts, and vulnerability detection.

Collaborating with renowned figures, he's contributed significantly to Web3 security projects and developed impactful tools.

He founded and nurtures the "web3guardians" community, fostering learning and engagement around blockchain security. His content creation and sharing of insights on platforms like Twitter and Medium highlight his commitment to advancing knowledge in the field.

0xMilenov's blend of technical expertise, community building, and proactive security research establish him as a dynamic and influential figure in the blockchain security landscape.

Feel free to reach out on X or 0xMilenov

2 Protocol Summary

DEEPSPACE – A revolutionary evolving platform that leverages risk for dynamic yield generating strategies! It combines frictionless holder incentives and reward structures!

Gameplay rewards holders through sharing transaction tax rewards and allows players to generate revenue through creating, owning, upgrading, and fighting with spacecraft!

A blockchain-based Play-to-Earn game with a supporting NFT marketplace is currently under development.

Buy a starship and explore the galaxy of DEEPSPACE. Travel to the edges of the galaxy to mine, build and battle!

Learn more at `https://deepspace.game`

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of {X} according to the specific commit. Any modifications to the code will require a new security review.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized

- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Overview

Project	DEEPSPACE
Repository	DEEPSPACE
Commit	...
Date	November 2023

Issues Found

Severity	Count
Critical Risk	0
High Risk	0
Medium Risk	4
Low Risk	8
Informational	10
Gas Optimizations	3
Total Issues	**25**

Summary of Findings

Title	Status
[M-1] Incompatibility of increaseAllowance/decreaseAllowance	Open
[M-2] withdrawalToken uses transfer instead of call which can break with future updates to gas costs	Open
[M-3] Inadequate deadline verification opens door to exploitation	Open
[M-4] approve will always revert as the IERC20 interface mismatch	Open

Summary of Findings

Title	Status
[L-1] Use Ownable2Step.acceptOwnership() instead of Ownable.transferOwnership	Open
[L-2] Certain tokens might revert transactions when zero-value transfers are initiated	Open
[L-3] Omitted validations for address(0) when setting values to address state variables	Open
[L-4] Avoid leaving an implementation contract in an uninitialized state	Open
[L-5] Lack of authentication in receive()/payable fallback()	Open
[L-6] Insufficient gas buffer and lock risk in sendValue and functionCallWithValue	Open
[L-7] receive() payable function does not authorize requests	Open
[L-8] Insecure use of address(this).balance in swapAndLiquify function	Open
[I-1] Replace magic numbers with descriptive constant definitions	Open
[I-2] Use delete for clearer semantics instead of setting to zero	Open
[I-3] Unemitted Event Declarations	Open
[I-4] Specify unsigned integer with uint256 instead of uint	Open
[I-5] Separate interface definitions from their implementations	Open
[I-6] No Event logged for vital parameter alterations	Open
[I-7] NatSpec is completely non-existent on functions that should have them	Open
[I-8] Avoid hard-coding address values	Open
[I-9] Unnecessary use of SafeMath Library	Open
[I-10] Avoid floating pragma	Open
[G-1] Using bools for storage incurs overhead	Open
[G-2] Revert strings are less gas-efficient than custom errors	Open
[G-3] Increments/decrements can be unchecked in for-loops	Open

5 Findings

5.1 Medium Risk

[M-1] Incompatibility of increaseAllowance/decreaseAllowance

Context DEEPSPACE.sol

Description Better remove increaseAllowance / decreaseAllowance

- These functions are not part of the EIP-20 specs.
- These functions may allow for further phishing possibilities (instead of the common approve or permit ones; see e.g. just 12 hours ago someone lost $24m since he got tricked into signing a malicious increaseAllowance payload https://etherscan.io/tx/0xcbe7b32e62c7d931a28f747bba3a0afa7da95169fcf380ac2f7d54f3a2f77913).
- The security concerns that fix increaseAllowance and decreaseAllowance are not critical nor high in the wild (and decreaseAllowance can be frontrunned also) and thus I think the responsibility can be delegated to the devs to decide whether to include it or not. Full conversation
- On the mainnet, the remedy to ensure compatibility with increaseAllowance/decreaseAllowance isn't implemented as evidenced here: https://etherscan.io/token/0xdac17f958d2ee523a2206206994597c13d831ec7#code.
- This results in a reversal when establishing a non-zero & non-maximum allowance, unless the set allowance is currently zero.

Affected code:

```
function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {

function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
```

Recommendation Considering removing increaseAllowance / decreaseAllowance

Or implement a function similar to this SafeERC20 library which is still available.

[M-2] withdrawalToken uses transfer instead of call which can break with future updates to gas costs

Context DEEPSPACE.sol

Description The use of the deprecated transfer() function for an address will inevitably make the transaction fail when:

- The claimer smart contract does not implement a payable function.
- The claimer smart contract does implement a payable fallback which uses more than 2300 gas unit.
- The claimer smart contract implements a payable fallback function that needs less than 2300 gas units but is called through proxy, raising the call's gas usage above 2300.
- Additionally, using higher than 2300 gas might be mandatory for some multisig wallets.

Affected code:

```
function withdrawalToken(address _tokenAddr, uint _amount) external onlyOwner() {
  IERC20 token = IERC20(_tokenAddr);
  token.transfer(owner(), _amount);
}
```

There are hundreds of examples. 1, 2

Recommendation Use call instead of transfer.

[M-3] Inadequate deadline verification opens door to exploitation

Context DEEPSPACE.sol

Description The functions `swapTokensForEth` and `addLiquidity` utilize block.timestamp for setting deadlines in Uniswap transactions. This creates a risk, as miners can slightly alter timestamps, potentially **leading to exploitation** through transaction timing, especially during periods of high network congestion or price volatility. Similar vulnerabilities have been observed in other contracts, as documented in cases like this finding.

Affected code:

```
function swapTokensForEth(uint256 tokenAmount) private {
    // generate the uniswap pair path of token -> weth
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = uniswapV2Router.WETH();

    _approve(address(this), address(uniswapV2Router), tokenAmount);

    // make the swap
    uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        0, // accept any amount of ETH
        path,
        address(this),
        block.timestamp
    );
}

function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
    // approve token transfer to cover all possible scenarios
    _approve(address(this), address(uniswapV2Router), tokenAmount);

    // add the liquidity
    uniswapV2Router.addLiquidityETH{value: ethAmount}(
        address(this),
        tokenAmount,
        0, // slippage is unavoidable
        0, // slippage is unavoidable
        owner(),
        block.timestamp
    );
    emit Transfer(address(this), uniswapV2Pair, tokenAmount);
}
```

Here are some additional resources that could be useful for learning about the process of Block Timestamp Manipulation.

Recommendation Replace block.timestamp with a fixed or adjustable deadline that is less susceptible to manipulation. For example, Uniswap sets it to 30 minutes on the Etehreum mainnet and to 5 minutes on L2 networks.

[M-4] approve will always revert as the IERC20 interface mismatch

Context DEEPSPACE.sol

Description Some tokens (like the very popular USDT) do not work when changing the allowance from an existing non-zero allowance value (it will revert if the current approval is not zero to protect against front-running changes of approvals). These tokens must first be approved for zero and then the actual allowance can be approved.

Affected code:

```
function swapTokensForEth(uint256 tokenAmount) private {
    // generate the uniswap pair path of token -> weth
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = uniswapV2Router.WETH();

    _approve(address(this), address(uniswapV2Router), tokenAmount);

function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
    // approve token transfer to cover all possible scenarios
    _approve(address(this), address(uniswapV2Router), tokenAmount);
```

There are hundreds of examples.

Recommendation Set the allowance to zero immediately before each of the existing allowance calls. Check the examples.

5.2 Low Risk

[L-1] Use Ownable2Step.acceptOwnership() instead of Ownable.transferOwnership

Context DEEPSPACE.sol

Description Suggest using a dual-stage process where the owner or admin designates an account, and the chosen account must invoke the acceptOwnership() function for complete ownership transition.

```
function acceptOwnership() public virtual {
    address sender = _msgSender();
    if (pendingOwner() != sender) {
        revert OwnableUnauthorizedAccount(sender);
    }
    _transferOwnership(sender);
}
```

- This verifies the designated EOA account's legitimacy and activity.
- Not having a two-stage process for essential operations makes them susceptible to mistakes.
- Think about integrating a two-stage mechanism for crucial functions.

Affected code:

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

Recommendation Use Ownable2Step Instead

[L-2] Certain tokens might revert transactions when zero-value transfers are initiated

Context DEEPSPACE.sol

Description For instance, you can take a look at this example.

Despite EIP-20's specification stating that zero-value transfers must be accepted, some tokens, such as LEND, will result in a revert if such transfers are attempted.

This can potentially lead to full transaction reversals, including operations involving other tokens (like batch operations).

To mitigate this issue and save gas, consider bypassing the transfer when the transfer amount is zero.

Affected code:

```
function withdrawalToken(address _tokenAddr, uint _amount) external onlyOwner() {
    IERC20 token = IERC20(_tokenAddr);
    token.transfer(owner(), _amount);
}
```

Recommendation To prevent transaction reversals and save gas, modify the withdrawalToken function to bypass the transfer when the _amount is zero.

```
function withdrawalToken(address _tokenAddr, uint _amount) external onlyOwner() {
    // Check if the amount is greater than zero
    if (_amount > 0) {
        IERC20 token = IERC20(_tokenAddr);
        token.transfer(owner(), _amount);
    }
    // If the amount is zero, the transfer is bypassed
}
```

[L-3] Omitted validations for address(0) when setting values to address state variables

Context DEEPSPACE.sol

Description Checking addresses against zero-address during initialization or during setting is a security best-practice. However, such checks are missing in address variable initializations/changes in many places. Given that zero-address is used as an indicator for BNB, there is a greater risk of using it accidentally.

Affected code:

```
function owner() public view virtual returns (address) {
   return _owner;
}
```

Recommendation Add zero-address checks for all initializations/setters of all address state variables.

[L-4] Avoid leaving an implementation contract in an uninitialized state

Context DEEPSPACE.sol

Description An uninitialized implementation contract can be taken over by an attacker, which may impact the proxy. To prevent the implementation contract from being used, it's advisable to invoke the `_disableInitializers` function in the constructor to automatically lock it when it is deployed. This should look similar to this:

```
  /// @custom:oz-upgrades-unsafe-allow constructor
  constructor() {
      _disableInitializers();
  }
```

Sources:

- https://docs.openzeppelin.com/contracts/4.x/api/proxy#Initializable-_disableInitializers--
- https://twitter.com/0xCygaar/status/1621417995905167360?s=20

Affected code:

```
constructor () {
        _rOwned[owner()] = _rTotal;

        _setUniswapV2Router(0x10ED43C718714eb63d5aA57B78B54704E256024E);

        //exclude owner and this contract from fee
        _isExcludedFromFee[owner()] = true;
        _isExcludedFromFee[address(this)] = true;

        emit Transfer(address(0), owner(), _tTotal);
    }
```

Recommendation You should add _disableInitializers to your implementation's constructor. This function will lock the implementation and prevent any function modified with initializer from being called.

[L-5] Lack of authentication in receive()/payable fallback()

Context DEEPSPACE.sol

Description If Ether is meant to be utilized, the function should invoke another specific function; otherwise, it should be reversed (e.g., require(msg.sender == address(eth))).

Absence of proper access control implies that an individual could transfer Ether to the contract but might be unable to retrieve it, leading to lost funds.

If the minor gas expenditure for checking the sender against an unchangeable address is a concern, there should, at a minimum, be a mechanism in place to recover stranded Ether.

Affected code:

```
  receive() external payable {}
```

Recommendation The function should call another function, otherwise it should revert (require(msg.sender == address(eth))

[L-6] Insufficient gas buffer and lock risk in sendValue and functionCallWithValue

Context DEEPSPACE.sol, DEEPSPACE.sol

Description The sendValue and functionCallWithValue functions in the contract check if the contract's balance is sufficient for the intended transaction.

```
function sendValue(address payable recipient, uint256 amount) internal {
    require(address(this).balance >= amount, "Address: insufficient balance");
}

function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage) internal
↪    returns (bytes memory) {
    require(address(this).balance >= value, "Address: insufficient balance for call");
}
```

If the contract's balance exactly equals the amount or value being sent, no Ether remains for covering gas fees, risking **transaction reversion.**

This scenario also poses **a risk of locking the contract** if it needs a balance for certain functionalities, like fallback methods or withdrawal patterns.

Recommendation Consider changing the checks to address(this).balance > amount or value.

[L-7] receive() payable function does not authorize requests

Context DEEPSPACE.sol

Description Having no access control on the function (e.g. `require(msg.sender == address(weth))`) means that someone may send Ether to the contract, and have no way to get anything back out, which is a loss of funds.

Affected code:

```
 receive() external payable {}
```

Recommendation If the concern is having to spend a small amount of gas to check the sender against an immutable address, the code should at least have a function to rescue mistakenly-sent Ether.

[L-8] Insecure use of address(this).balance in swapAndLiquify function

Context DEEPSPACE.sol

Description In the swapAndLiquify function, the contract's total Ether balance is used to figure out how much Ether was made from swapping tokens. The problem is, this total includes all Ether sent to the contract, even in ways not counted by the function, like direct transfers to the contract. This **can mess up how much Ether the function thinks it should use.** Check the Solidity documentation

Affected code:

```
function swapAndLiquify(uint256 contractTokenBalance) private lockTheSwap {
    // split the contract balance into halves
    uint256 half = contractTokenBalance.div(2);
    uint256 otherHalf = contractTokenBalance.sub(half);

    // capture the contract's current ETH balance.
    // this is so that we can capture exactly the amount of ETH that the
    // swap creates, and not make the liquidity event include any ETH that
    // has been manually sent to the contract
    uint256 initialBalance = address(this).balance;

    // swap tokens for ETH
    swapTokensForEth(half); // <- this breaks the ETH -> HATE swap when swap+liquify is triggered

    // how much ETH did we just swap into?
    uint256 newBalance = address(this).balance.sub(initialBalance);

    // add liquidity to uniswap
    addLiquidity(otherHalf, newBalance);

    emit SwapAndLiquify(half, newBalance, otherHalf);
}
```

Recommendation The contract should keep track of Ether from token swaps separately from the total Ether balance.

5.3 Informational

[I-1] Replace magic numbers with descriptive constant definitions

Context DEEPSPACE.sol

Description Using direct numbers (often referred to as magic numbers') in the code can make it difficult for other developers to understand the intent or purpose behind specific values.

This can lead to misunderstandings, errors, or vulnerabilities. It's a good practice to replace these magic numbers with named constant variables that provide a clear, descriptive label for the value.

For clarity and maintainability, even in assembly blocks, it's beneficial to utilize named constants.

Affected code:

```
require(taxFee + _liquidityFee + _DevFee <= 15
```

[I-2] Use delete for clearer semantics instead of setting to zero

Context DEEPSPACE.sol

Description Using the delete keyword conveys the intention more clearly and highlights changes in state, promoting a more detailed review of the surrounding logic.

Affected code:

```
    _tOwned[account] = 0;
    _taxFee = 0;
    _DevFee = 0;
    _liquidityFee = 0;
```

[I-3] Unemitted Event Declarations

Context DEEPSPACE.sol

Description Certain events have been defined in the code but aren't ever triggered. To enhance code clarity and cleanliness, consider eliminating this unused event definition. The instance mentioned below fall into this category.

Affected code:

```
  event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
```

Recommendation Remove it.

[I-4] Specify unsigned integer with uint256 instead of uint

Context DEEPSPACE.sol

Description To enhance clarity and minimize potential ambiguities, it's advisable to use uint256 in place of the generic uint. Explicitly declaring the data type's size can foster a clearer comprehension of the code's intent and expected behavior.

Affected code:

```
  function withdrawalToken(address _tokenAddr, uint _amount) external onlyOwner() {
```

Recommendation

```
  function withdrawalToken(address _tokenAddr, uint256 _amount) external onlyOwner() {
```

[I-5] Separate interface definitions from their implementations

Context DEEPSPACE.sol

Description It's beneficial to define the interfaces in distinct files from where they're utilized. By doing this, it simplifies the process for future projects to import these interfaces. Furthermore, it helps in preventing potential redundancy should these interfaces be required elsewhere within the project later on.

Affected code:

```
interface IUniswapV2Factory

interface IUniswapV2Pair

interface IUniswapV2Router01

interface IUniswapV2Router02 is IUniswapV2Router01
```

Recommendation Separate interface from implementation for clearer code.

[I-6] No Event logged for vital parameter alterations

Context DEEPSPACE.sol

Description Events play a crucial role in alerting off-chain tools about state changes. Without them, users might unexpectedly face changes without prior notice.

Affected code:

```
function setTaxFeePercent(uint256 taxFee) external onlyOwner() {
    require(taxFee + _liquidityFee + _DevFee <= 15, "Total fees exceed 15%");
    _taxFee = taxFee;
}

function setDevFeePercent(uint256 DevFee) external onlyOwner() {
    require(_taxFee + _liquidityFee + DevFee <= 15, "Total fees exceed 15%");
    _DevFee = DevFee;
}

function setLiquidityFeePercent(uint256 liquidityFee) external onlyOwner() {
    require(_taxFee + liquidityFee + _DevFee <= 15, "Total fees exceed 15%");
    _liquidityFee = liquidityFee;
}

function setMaxTxPercent(uint256 maxTxPercent) external onlyOwner() {
    _maxTxAmount = _tTotal.mul(maxTxPercent).div(
        10**2
    );
}

function setSwapEnabled(bool _enabled) external onlyOwner() {
    swapEnabled = _enabled;
}

function setDevWalletAddress(address _address) external onlyOwner() {
    _DevWalletAddress = _address;
}

function setNumTokensSellToAddToLiquidity(uint256 _amount) external onlyOwner() {
    numTokensSellToAddToLiquidity = _amount;
}

function _setUniswapV2Router(address _router) private {
    IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(_router);
    uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).getPair(address(this),
    ↪    _uniswapV2Router.WETH());
    if(uniswapV2Pair == address(0))
        uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),
        ↪    _uniswapV2Router.WETH());
    uniswapV2Router = _uniswapV2Router;
}

function setUniswapV2Router(address _router) external onlyOwner() {
    _setUniswapV2Router(_router);
}
```

Recommendation Add Events

[I-7] NatSpec is completely non-existent on functions that should have them

Context DEEPSPACE.sol

Description It is recommended that Solidity contracts are fully annotated using NatSpec. This documentation is segmented into developer-focused messages and end-user-facing messages. These messages may be shown to the end user (the human) at the time that they will interact with the contract (i.e. sign a transaction).

Affected code:

```
contract DEEPSPACE is Context, IERC20, Ownable
```

Recommendation Add NatSpec

[I-8] Avoid hard-coding address values

Context DEEPSPACE.sol

Description For greater flexibility across different network deployments, it's advisable to use immutable for declaring address values and assign them using constructor parameters. This approach ensures consistent code and eliminates the need for recompilation when address assignments change.

Affected code:

```
address public _DevWalletAddress = 0xaCc34268f5D7Cb9B11BfB1ba4D8bD2bc2B49EE4E;

_setUniswapV2Router(0x10ED43C718714eb63d5aA57B78B54704E256024E);
```

Recommendation You can have a variable (instead of a constant) containing the token address. And this variable can be set from a constructor. So you can effectively pass the value from an environment variable to the contract constructor, to the contract storage.

[I-9] Unnecessary use of SafeMath Library

Context DEEPSPACE.sol

Description Since Solidity v0.8.0, all arithmetic operations are checked by default and revert on over- or underflow. Hence, it is not necessary anymore to use the SafeMath library (or SafeMathUpgradeable). Employing it nonetheless not only wastes gas but also reduces the readability of arithmetic expressions considerably.

Affected code:

```
contract DEEPSPACE is Context, IERC20, Ownable {
    using SafeMath for uint256;
```

Recommendation I recommend using the built-in arithmetic operations instead of SafeMath.

[I-10] Avoid floating pragma

Context DEEPSPACE.sol

Description Using floating `pragma` can introduce unexpected behaviors with future compiler versions.

Affected code:

```
pragma solidity ^0.8.3;
```

Recommendation It's advisable to use fixed compiler versions for non-library/interface files to ensure consistent behavior.

5.4 Gas Optimization

[G-1] Using bools for storage incurs overhead

Context DEEPSPACE.sol

Description "Booleans are more expensive than uint256 or any type that takes up a full word because each write operation emits an extra SLOAD to first read the slot's contents, replace the bits taken up by the boolean, and then write back. This is the compiler's defense against contract upgrades and pointer aliasing, and it cannot be disabled." - OpenZeppelin

Affected code:

```
bool inSwapAndLiquify;
bool public swapAndLiquifyEnabled = true;
bool public swapEnabled = false;
```

Recommendation Use uint256(1) and uint256(2) for true/false to avoid a Gwarmaccess (100 gas) for the extra SLOAD, and to avoid Gsset (20000 gas) when changing from ‘false’ to ‘true’, after having been ‘true’ in the past

[G-2] Revert strings are less gas-efficient than custom errors

Context DEEPSPACE.sol

Description From Solidity v0.8.4 onward, custom errors have been introduced.

These errors provide a savings of approximately 50 gas each instance they're triggered, as they circumvent the need to allocate and keep the revert string. Omitting these strings also conserves gas during deployment.

Furthermore, custom errors are versatile, applicable both inside and outside contracts, including in interfaces and libraries.

As stated in the Solidity blog:

With the introduction of Solidity v0.8.4, a streamlined and gas-efficient method has been provided to elucidate to users the reasons behind an operation's failure through custom errors. Prior to this, although strings could be used to detail failure reasons (e.g., revert('Insufficient funds.');), they were notably costlier, especially in terms of deployment, and incorporating dynamic information into them was challenging.

Affected code:

```
require(!_isExcluded[sender], "Excluded addresses cannot call this function");

require(tAmount <= _tTotal, "Amount must be less than supply");

require(rAmount <= _rTotal, "Amount must be less than total reflections");

require(!_isExcluded[account], "Account is already excluded");

require(_isExcluded[account], "Account is already included");

require(taxFee + _liquidityFee + _DevFee <= 15, "Total fees exceed 15%");

require(_taxFee + _liquidityFee + DevFee <= 15, "Total fees exceed 15%");

require(_taxFee + liquidityFee + _DevFee <= 15, "Total fees exceed 15%");

require(owner != address(0), "ERC20: approve from the zero address");

require(spender != address(0), "ERC20: approve to the zero address");

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

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

require(amount > 0, "Transfer amount must be greater than zero");

require(amount <= _maxTxAmount, "Transfer amount exceeds the maxTxAmount.");

revert("Buying and selling is disabled");
```

Recommendation It's advisable to transition all revert strings to custom errors in your solution, especially focusing on those that appear multiple times.

[G-3] Increments/decrements can be unchecked in for-loops

Context DEEPSPACE.sol

Description In Solidity 0.8+, there's a default overflow check on unsigned integers. It's possible to uncheck this in for-loops and save some gas at each iteration, but at the cost of some code readability, as this uncheck cannot be made inline.

ethereum/solidity#10695

Affected code:

```
for (uint256 i = 0; i < _excluded.length; i++) {
```

Recommendation The change would be:

```
- for (uint256 i; i < numIterations; i++) {
+ for (uint256 i; i < numIterations;) {
 // ...
+   unchecked { ++i; }
}
```

These save around **25 gas saved** per instance. The same can be applied with decrements (which should use `break` when `i == 0`). The risk of overflow is non-existent for `uint256`.