Cyberscope

A TAC Security Company

Audit Report

DEEPTICS

October 2025

Network BSC

Address 0xf17ed7686e6a7096f962f90f00bdeabb45c7630d

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Unresolved
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	PAMAR	Pair Address Max Amount Restriction	Unresolved
	AAO	Accumulated Amount Overflow	Unresolved
	CCR	Contract Centralization Risk	Unresolved
	MEM	Missing Error Messages	Unresolved
	MEE	Missing Events Emission	Unresolved
	MPV	Missing Parameter Validations	Unresolved
	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
	PMRM	Potential Mocked Router Manipulation	Unresolved
	PSU	Potential Subtraction Underflow	Unresolved
	PTRP	Potential Transfer Revert Propagation	Unresolved
	PVC	Price Volatility Concern	Unresolved
	RRA	Redundant Repeated Approvals	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	RSRS	Redundant SafeMath Require Statement	Unresolved

●	RSV	Redundant State Variables	Unresolved
●	ZD	Zero Division	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	DEEPTICS
Compiler Version	v0.8.24+commit.e11b9ed9
Optimization	200 runs
Explorer	https://bscscan.com/address/0xf17ed7686e6a7096f962f90f00bdeabb45c7630d
Address	0xf17ed7686e6a7096f962f90f00bdeabb45c7630d
Network	BSC
Symbol	$DPTX
Decimals	9
Total Supply	10.000.000
Badge Eligibility	Must Fix Criticals

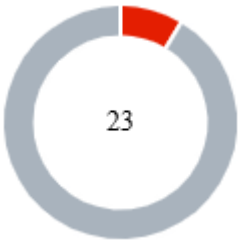
Audit Updates

Initial Audit	15 Oct 2025

Source Files

Filename	SHA256
DEEPTICS.sol	3280ddb10018bd764c80c24e0709749f4b8a80890fa79bfa1bb2747dffded18a

Findings Breakdown



Critical	2
Medium	0
Minor / Informative	21

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	21	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	DEEPTICS.sol#L655,662,674
Status	Unresolved

Description

The contract owner has multiple ways to stop the sales for all users excluding the owner.

Specifically the transactions can be stopped by:

- Including the pair to transaction limit and max transaction amount
- Setting max transaction amount and wallet limit to a very small number or zero
- Set the minimum number of tokens before swap to 0. The owner can set `swapAndLiquifyByLimitOnly` to `true` to enforce using only zero tokens for the swap. This will result in the contract trying to swap zero tokens which will result in the revert of the `transfer`.

Additionally, sales can be stopped from reasons described in `AAO`, `PSU`, `ZD`, `PMRM`, `PAMAR`, `PLPI` and `MPV` findings.

As a result, the contract may operate as a honeypot.

Shell

```
if(!isTxLimitExempt[sender] &&
!isTxLimitExempt[recipient]) {
    require(amount <= _maxTxAmount, "Transfer
amount exceeds the maxTxAmount.");
}
...
if (overMinimumTokenBalance && !inSwapAndLiquify
&& !isMarketPair[sender] && swapAndLiquifyEnabled)
{
    if(swapAndLiquifyByLimitOnly)
        contractTokenBalance =
minimumTokensBeforeSwap;
    swapAndLiquify(contractTokenBalance);
}
...
if(checkWalletLimit &&
!isWalletLimitExempt[recipient])


require(balanceOf(recipient).add(finalAmount) <=
_walletMax);
```

Recommendation

The contract could embody a check for not allowing setting the `_maxTxAmount`, `_walletMax` and `minimumTokensBeforeSwap` to a value less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply.

Additionally the team is advised to not enforce restrictions on the pair contract.

Also, the team should take into consideration the recommendations of the `AAO`, `PSU`, `ZD`, `PMRM`, `PAMAR`, `PLPI` and `MPV` findings.

Finally, the team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	DEEPTICS.sol#L541,549
Status	Unresolved

Description

The contract owner has the authority to increase taxes over the allowed limit of 25%. The owner may take advantage of it by calling the `setBuyTaxes` or `setSellTaxes` functions with a high percentage value.

```
Shell
function setBuyTaxes(uint256 newLiquidityTax,
uint256 newMarketingTax, uint256 newBuyBackTax)
external onlyOwner() {
    ...
    _totalTaxIfBuying =
_buyLiquidityFee.add(_buyMarketingFee).add(_buyBuy
BackFee);
}

function setSellTaxes(uint256 newLiquidityTax,
uint256 newMarketingTax, uint256 newBuyBackTax)
external onlyOwner() {
    ...
    _totalTaxIfSelling =
_sellLiquidityFee.add(_sellMarketingFee).add(_sell
BuyBackFee);
}
```

Recommendation

The contract could embody a check for the maximum acceptable value. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

PAMAR - Pair Address Max Amount Restriction

Criticality	Minor / Informative
Location	DEEPTICS.sol#L674
Status	Unresolved

Description

The contract is configured to enforce a maximum token accumulation limit through checks. This mechanism aims to prevent excessive token concentration by reverting transactions that overcome the specified cap. However, this functionality encounters issues when transactions default to the pair address during sales. If the pair address is not listed in the exceptions, then the sale transactions are inadvertently stopped, effectively disrupting operations and making the contract susceptible to unintended behaviors akin to a honeypot.

```
Shell
if(checkWalletLimit &&
!isWalletLimitExempt[recipient])
    require(balanceOf(recipient).add(finalAmount)
<= _walletMax);
```

Recommendation

It is advised to modify the contract to ensure uninterrupted operations by either permitting the pair address to exceed the established token accumulation limit or by safeguarding its status in the exception list. By recognizing and allowing these essential addresses the flexibility to hold more tokens than typical limits, the contract can maintain seamless transaction flows and uphold the liquidity and stability of the ecosystem. This modification is vital for avoiding disruptions that could impact the functionality and security of the contract.

AAO - Accumulated Amount Overflow

Criticality	Minor / Informative
Location	DEEPTICS.sol#L755,758,762
Status	Unresolved

Description

The contract is using variables to accumulate values. The contract could lead to an overflow when the total value of a variable exceeds the maximum value that can be stored in that variable's data type.

Specifically the `_totalTaxIfBuying` and `_totalTaxIfSelling` can be set to a very large number. If that number is multiplied by the amount the result may exceed the max uint. The case is similar when updating the balance of the contract.

```
Shell
if(isMarketPair[sender]) {
    feeAmount =
amount.mul(_totalTaxIfBuying).div(100);
}
else if(isMarketPair[recipient]) {
    feeAmount =
amount.mul(_totalTaxIfSelling).div(100);
}
if(feeAmount > 0) {
    _balances[address(this)] =
_balances[address(this)].add(feeAmount);
    emit Transfer(sender, address(this),
feeAmount);

}
```

Recommendation

The team is advised to carefully investigate the usage of the variables that accumulate value. A suggestion is to add checks to the code to ensure that the value of a variable does not exceed the maximum value that can be stored in its data type.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	DEEPTICS.sol#L529,533,537,541,549,557,565,570,574,578,582,586,594,599,611
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```shell
function addMarketPair(address account) public
onlyOwner
function setIsTxLimitExempt(address holder, bool
exempt) external onlyOwner
function setIsExcludedFromFee(address account,
bool newValue) public onlyOwner
function setBuyTaxes(uint256 newLiquidityTax,
uint256 newMarketingTax, uint256 newBuyBackTax)
external onlyOwner()
function setSellTaxes(uint256 newLiquidityTax,
uint256 newMarketingTax, uint256 newBuyBackTax)
external onlyOwner()
function setDistributionSettings(uint256
newLiquidityShare, uint256 newMarketingShare,
uint256 newBuyBackShare) external onlyOwner()
function setMaxTxAmount(uint256 maxTxAmount)
external onlyOwner()
function enableDisableWalletLimit(bool newValue)
external onlyOwner
function setIsWalletLimitExempt(address holder,
bool exempt) external onlyOwner
```

```
function setWalletLimit(uint256 newLimit) external
onlyOwner
function setNumTokensBeforeSwap(uint256 newLimit)
external onlyOwner()
function setMarketingWalletAddress(address
newAddress) external onlyOwner()
function setBuyBackWalletAddress(address
newAddress) external onlyOwner()
function setSwapAndLiquifyEnabled(bool _enabled)
public onlyOwner
function setSwapAndLiquifyByLimitOnly(bool
newValue) public onlyOwner

function changeRouterVersion(address
newRouterAddress) public onlyOwner returns(address
newPairAddress)
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	DEEPTICS.sol#L675
Status	Unresolved

Description

The contract is missing error messages. Specifically, there are no error messages to accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
Shell
require(balanceOf(recipient).add(finalAmount) <=
_walletMax)
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	DEEPTICS.sol#L529,533,537,541,549,557,565,570,574,578,582,586,599,607,611,715,735
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```shell
function addMarketPair(address account) public
onlyOwner
function setIsTxLimitExempt(address holder, bool
exempt) external onlyOwner
function setIsExcludedFromFee(address account,
bool newValue) public onlyOwner
function setBuyTaxes(uint256 newLiquidityTax,
uint256 newMarketingTax, uint256 newBuyBackTax)
external onlyOwner()
function setSellTaxes(uint256 newLiquidityTax,
uint256 newMarketingTax, uint256 newBuyBackTax)
external onlyOwner()
function setDistributionSettings(uint256
newLiquidityShare, uint256 newMarketingShare,
uint256 newBuyBackShare) external onlyOwner()
function setMaxTxAmount(uint256 maxTxAmount)
external onlyOwner()
function enableDisableWalletLimit(bool newValue)
external onlyOwner
function setIsWalletLimitExempt(address holder,
bool exempt) external onlyOwner
function setWalletLimit(uint256 newLimit) external
onlyOwner
function setNumTokensBeforeSwap(uint256 newLimit)
external onlyOwner()
function setMarketingWalletAddress(address
newAddress) external onlyOwner()
function setBuyBackWalletAddress(address
newAddress) external onlyOwner()
function setSwapAndLiquifyByLimitOnly(bool
newValue) public onlyOwner
function transferToAddressETH(address payable
recipient, uint256 amount) private
function changeRouterVersion(address
newRouterAddress) public onlyOwner returns(address
newPairAddress)
function swapTokensForEth(uint256 tokenAmount)
private
```

```
function addLiquidity(uint256 tokenAmount, uint256
ethAmount) private
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

MPV - Missing Parameter Validations

Criticality	Minor / Informative
Location	DEEPTICS.sol#L533,537,541,549,557,565,570,574,578,582,586,594,599,611
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

Specifically, the contract is missing essential validation checks for the setter function parameters:

- Exclusion setters do not check if the address is already in the state that is added as input. Specifically for `setIsWalletLimitExempt` and `setIsTxLimitExempt` the pairs should not be able to be included because it will break the `transfer` operations.
- Taxes can be set above the acceptable 25%. Additionally, taxes can be set above 100% which will result in breaking the `transfer` operations as described in the findings `AAO` and `PSU`.
- Distribution settings can be set to zero as described in the `ZD` finding.
- Max transaction amount, wallet limit and minimum tokens before swap can be set to a very small number or even 0 that will also break the `transfer` operations.

```shell
function setIsTxLimitExempt(address holder, bool
exempt) external onlyOwner
function setIsExcludedFromFee(address account,
bool newValue) public onlyOwner
function setBuyTaxes(uint256 newLiquidityTax,
uint256 newMarketingTax, uint256 newBuyBackTax)
external onlyOwner()
function setSellTaxes(uint256 newLiquidityTax,
uint256 newMarketingTax, uint256 newBuyBackTax)
external onlyOwner()
function setDistributionSettings(uint256
newLiquidityShare, uint256 newMarketingShare,
uint256 newBuyBackShare) external onlyOwner()
function setMaxTxAmount(uint256 maxTxAmount)
external onlyOwner()
function enableDisableWalletLimit(bool newValue)
external onlyOwner
function setIsWalletLimitExempt(address holder,
bool exempt) external onlyOwner
function setWalletLimit(uint256 newLimit) external
onlyOwner
function setNumTokensBeforeSwap(uint256 newLimit)
external onlyOwner()
function setMarketingWalletAddress(address
newAddress) external onlyOwner()
function setBuyBackWalletAddress(address
newAddress) external onlyOwner()
function setSwapAndLiquifyEnabled(bool _enabled)
public onlyOwner
function setSwapAndLiquifyByLimitOnly(bool
newValue) public onlyOwner

function changeRouterVersion(address

newRouterAddress) public onlyOwner returns(address

newPairAddress)
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	DEEPTICS.sol#L724,740
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```shell
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount, 0, path, address(this), block.timestamp);

uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this), tokenAmount, 0, 0, owner(), block.timestamp);
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

PMRM - Potential Mocked Router Manipulation

Criticality	Minor / Informative
Location	DEEPTICS.sol#L611
Status	Unresolved

Description

The contract includes a method that allows the owner to modify the router address and create a new pair. While this feature provides flexibility, it introduces a security threat. The owner could set the router address to any contract that implements the router's interface, potentially containing malicious code. In the event of a transaction triggering the swap functionality with such a malicious contract as the router, the transaction may be manipulated.

```shell
function changeRouterVersion(address
newRouterAddress) public onlyOwner returns(address
newPairAddress) {
    IUniswapV2Router02 _uniswapV2Router =
IUniswapV2Router02(newRouterAddress);
    newPairAddress =
IUniswapV2Factory(_uniswapV2Router.factory()).getP
air(address(this), _uniswapV2Router.WETH());
    if(newPairAddress == address(0))
    {
        newPairAddress =
IUniswapV2Factory(_uniswapV2Router.factory())
            .createPair(address(this),
_uniswapV2Router.WETH());
    }
    uniswapPair = newPairAddress;
    uniswapV2Router = _uniswapV2Router;
    isWalletLimitExempt[address(uniswapPair)] =
true;
    isMarketPair[address(uniswapPair)] = true;
```

```
}
```

Recommendation

The team should carefully manage the private keys of the owner’s account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

PSU - Potential Subtraction Underflow

Criticality	Minor / Informative
Location	DEEPTICS.sol#L754,758,766
Status	Unresolved

Description

The contract subtracts two values, the second value may be greater than the first value if the contract owner misuses the configuration. As a result, the subtraction may underflow and cause the execution to revert.

Specifically the `_totalTaxIfBuying` and `_totalTaxIfSelling` can be set into a value greater than 100. This will result in `feeAmount` being greater than `amount`.

```
Shell
if(isMarketPair[sender]) {
    feeAmount =
amount.mul(_totalTaxIfBuying).div(100);
}
else if(isMarketPair[recipient]) {
    feeAmount =
amount.mul(_totalTaxIfSelling).div(100);
}
...

return amount.sub(feeAmount);
```

Recommendation

The team is advised to properly handle the code to avoid underflow subtractions and ensure the reliability and safety of the contract. The contract should ensure that the first value is always greater than the second value. It should add a sanity check in the setters of the variable or not allow executing the corresponding section if the condition is violated.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	DEEPTICS.sol#L705
Status	Unresolved

Description

The contract sends funds to a `marketingWalletAddress` and a `BuyBackWalletAddress` as part of the transfer flow. These addresses can either be wallet addresses or contracts. If either of these addresses belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
Shell
if(amountBNBMarketing > 0)
    transferToAddressETH(marketingWalletAddress,
amountBNBMarketing);
if(amountBNBBuyBack > 0)
    transferToAddressETH(BuyBackWalletAddress,
amountBNBBuyBack);
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by sending the funds in a non-revertable way while also protecting the contract from potential gas exhaustions.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	DEEPTICS.sol#L660
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `minimumTokensBeforeSwap` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```shell
bool overMinimumTokenBalance =
contractTokenBalance >= minimumTokensBeforeSwap;
if (overMinimumTokenBalance && !inSwapAndLiquify
&& !isMarketPair[sender] && swapAndLiquifyEnabled)
{
    if(swapAndLiquifyByLimitOnly)
        contractTokenBalance =
minimumTokensBeforeSwap;
    swapAndLiquify(contractTokenBalance);

}
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RRA - Redundant Repeated Approvals

Criticality	Minor / Informative
Location	DEEPTICS.sol#L721,735
Status	Unresolved

Description

The contract is designed to `approve` token transfers during the contract's operation by calling the _approve function before specific operations. This approach results in additional gas costs since the approval process is repeated for every operation execution, leading to inefficiencies and increased transaction expenses.

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

Recommendation

Since the approved address is a trusted third-party source, it is recommended to optimize the contract by approving the maximum amount of tokens once in the initial set of the variable, rather than before each operation. This change will reduce the overall gas consumption and improve the efficiency of the contract.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	DEEPTICS.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily in cases where the explanatory error message is not used.

```shell
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/stable/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSRS - Redundant SafeMath Require Statement

Criticality	Minor / Informative
Status	Unresolved

Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

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

}
```

Recommendation

It is recommended to remove the `require` statement from the add function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

RSV - Redundant State Variables

Criticality	Minor / Informative
Location	DEEPTICS.sol#L401
Status	Unresolved

Description

The contract includes redundant state variables that hinder code readability and optimization. Specifically, the state variable storing separate fees such as liquidity fee, marketing fee and buyBack fee are not used in any of the contract's functionality except for calculating the `_totalTaxIfBuying` and `_totalTaxIfSelling`.

```
Shell
_buyLiquidityFee
_buyMarketingFee
_buyBuyBackFee
_sellLiquidityFee
_sellMarketingFee

_sellBuyBackFee
```

Recommendation

The team should update the contract so that it does not include unused state variable to enhance optimization and readability.

ZD - Zero Division

Criticality	Minor / Informative
Location	DEEPTICS.sol#L693
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

Specifically, `_totalDistributionShares` can be set to 0 which will revert the `transfer` operation.

```
Shell
uint256 tokensForLP =
tAmount.mul(_liquidityShare).div(_totalDistributio
nShares).div(2);
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow the execution of the corresponding statements.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	DEEPTICS.sol#L385,386,387,416
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```shell
string private _name = "DEEPTICS"
string private _symbol = "$DPTX"
uint8 private _decimals = 9

uint256 private _totalSupply = 100 * 10**5 * 10**9
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	DEEPTICS.sol#L211,212,228,247,390,401,402,403,404,405,406,408,409,410,412,413,414,417,418,594
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```shell
function DOMAIN_SEPARATOR() external view returns
(bytes32);
function PERMIT_TYPEHASH() external pure returns
(bytes32);
function MINIMUM_LIQUIDITY() external pure returns
(uint);
function WETH() external pure returns (address);
address payable public BuyBackWalletAddress =
payable(0xC9aA85A9F12d13E4B68CDCaE9De8662a5d09bff1
)
uint256 public _buyLiquidityFee = 1
uint256 public _buyMarketingFee = 5
uint256 public _buyBuyBackFee = 1
uint256 public _sellLiquidityFee = 2
uint256 public _sellMarketingFee = 4
uint256 public _sellBuyBackFee = 2
uint256 public _liquidityShare = 2
uint256 public _marketingShare = 3
uint256 public _BuyBackShare = 2


...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation

https://docs.soliditylang.org/en/stable/style-guide.html#naming-conventions.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	DEEPTICS.sol#L546,554,558,567,579,583
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
Shell
_totalTaxIfBuying =
_buyLiquidityFee.add(_buyMarketingFee).add(_buyBuy
BackFee)
_totalTaxIfSelling =
_sellLiquidityFee.add(_sellMarketingFee).add(_sell
BuyBackFee)
_liquidityShare = newLiquidityShare
_maxTxAmount = maxTxAmount
_walletMax  = newLimit

minimumTokensBeforeSwap = newLimit
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	DEEPTICS.sol#L587,591
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```shell
marketingWalletAddress = payable(newAddress)
BuyBackWalletAddress = payable(newAddress)
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	DEEPTICS.sol#L98,136
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
Shell
assembly { codehash := extcodehash(account) }

assembly {
                    let returndata_size :=
mload(returndata)
                    revert(add(32, returndata),
returndata_size)
                }
```

Recommendation

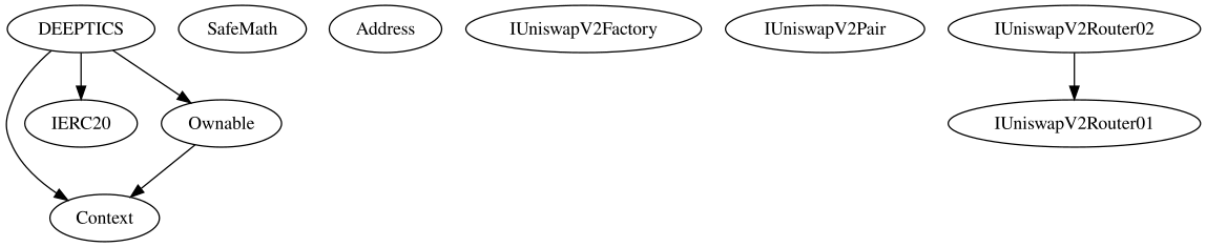
It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

Functions Analysis

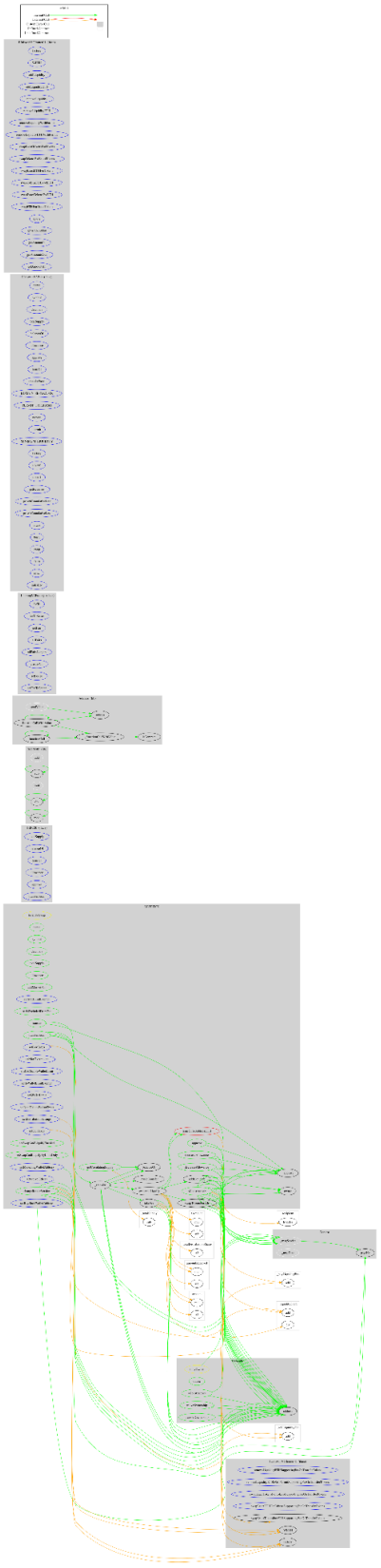
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
DEEPTICS	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	allowance	Public		-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	approve	Public	✓	-
	_approve	Private	✓	
	addMarketPair	Public	✓	onlyOwner
	setIsTxLimitExempt	External	✓	onlyOwner
	setIsExcludedFromFee	Public	✓	onlyOwner
	setBuyTaxes	External	✓	onlyOwner
	setSellTaxes	External	✓	onlyOwner
	setDistributionSettings	External	✓	onlyOwner
	setMaxTxAmount	External	✓	onlyOwner
	enableDisableWalletLimit	External	✓	onlyOwner

	setIsWalletLimitExempt	External	✓	onlyOwner
	setWalletLimit	External	✓	onlyOwner
	setNumTokensBeforeSwap	External	✓	onlyOwner
	setMarketingWalletAddress	External	✓	onlyOwner
	setBuyBackWalletAddress	External	✓	onlyOwner
	setSwapAndLiquifyEnabled	Public	✓	onlyOwner
	setSwapAndLiquifyByLimitOnly	Public	✓	onlyOwner
	getCirculatingSupply	Public		-
	transferToAddressETH	Private	✓	
	changeRouterVersion	Public	✓	onlyOwner
		External	Payable	-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	_transfer	Private	✓	
	_basicTransfer	Internal	✓	
	swapAndLiquify	Private	✓	lockTheSwap
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	takeFee	Internal	✓	

Inheritance Graph



Flow Graph



Summary

DEEPTICS contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions and manipulate the fees. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a TAC blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

A **TAC Security** Company

The Cyberscope team

cyberscope.io