



Cyberscope

# Audit Report

## **NFT STRIKE**

November 2023

Network    ETH Goerli

Address    0xe28a9769816606e900DDAC7eD383699BCBbBF5a1

Audited by    © cyberscope

# Analysis

● Critical ● Medium ● Minor / Informative ● Pass

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

# Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	SUF	Swap Unhandled Fees	Unresolved
●	MAA	Missing Amount Approval	Unresolved
●	RC	Redundant Checks	Unresolved
●	CR	Code Repetition	Unresolved
●	FSA	Fixed Swap Address	Unresolved
●	RAS	Redundant Assert Statement	Unresolved
●	RCS	Redundant Conditional Statement	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	AOI	Arithmetic Operations Inconsistency	Unresolved
●	TUU	Time Units Usage	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	MRM	Missing Revert Messages	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved

●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

# Table of Contents

<b>Analysis</b>	<b>1</b>
<b>Diagnostics</b>	<b>2</b>
<b>Table of Contents</b>	<b>4</b>
<b>Review</b>	<b>6</b>
Audit Updates	6
Source Files	6
<b>Findings Breakdown</b>	<b>7</b>
SUF - Swap Unhandled Fees	8
Description	8
Recommendation	9
MAA - Missing Amount Approval	10
Description	10
Recommendation	11
RC - Redundant Checks	12
Description	12
Recommendation	12
CR - Code Repetition	13
Description	13
Recommendation	13
FSA - Fixed Swap Address	14
Description	14
Recommendation	14
RAS - Redundant Assert Statement	15
Description	15
Recommendation	15
RCS - Redundant Conditional Statement	16
Description	16
Recommendation	16
PTRP - Potential Transfer Revert Propagation	17
Description	17
Recommendation	17
AOI - Arithmetic Operations Inconsistency	18
Description	18
Recommendation	18
TUU - Time Units Usage	19
Description	19
Recommendation	19
MEE - Missing Events Emission	20
Description	20

Recommendation	20
PVC - Price Volatility Concern	21
Description	21
Recommendation	21
MRM - Missing Revert Messages	22
Description	22
Recommendation	22
RSML - Redundant SafeMath Library	23
Description	23
Recommendation	23
L02 - State Variables could be Declared Constant	24
Description	24
Recommendation	24
L04 - Conformance to Solidity Naming Conventions	25
Description	25
Recommendation	26
L05 - Unused State Variable	27
Description	27
Recommendation	27
L07 - Missing Events Arithmetic	28
Description	28
Recommendation	28
L09 - Dead Code Elimination	29
Description	29
Recommendation	29
L13 - Divide before Multiply Operation	31
Description	31
Recommendation	31
L16 - Validate Variable Setters	32
Description	32
Recommendation	32
L19 - Stable Compiler Version	33
Description	33
Recommendation	33
<b>Functions Analysis</b>	<b>34</b>
<b>Inheritance Graph</b>	<b>39</b>
<b>Flow Graph</b>	<b>40</b>
<b>Summary</b>	<b>41</b>
<b>Disclaimer</b>	<b>42</b>
<b>About Cyberscope</b>	<b>43</b>

## Review

Contract Name	NFTSTRIKE
Compiler Version	v0.8.18+commit.87f61d96
Optimization	200 runs
Explorer	<a href="https://goerli.etherscan.io/address/0xe28a9769816606e900ddac7ed383699bcbbbf5a1">https://goerli.etherscan.io/address/0xe28a9769816606e900ddac7ed383699bcbbbf5a1</a>
Address	0xe28a9769816606e900ddac7ed383699bcbbbf5a1
Network	GOERLI
Symbol	NFTS
Decimals	18
Total Supply	10,000,000,000

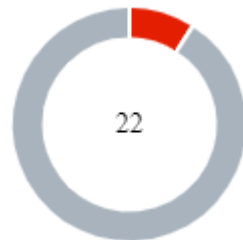
## Audit Updates

Initial Audit	01 Nov 2023
Corrected Phase 2	03 Nov 2023
Corrected Phase 3	09 Nov 2023
Corrected Phase 4	13 Nov 2023

## Source Files

Filename	SHA256
NFTSTRIKE.sol	6140479accac1bd0b0f4ae906c531d072ff78bf22b6017bdf2414c815f202351

## Findings Breakdown



Critical	2
Medium	0
Minor / Informative	20

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	20	0	0	0



## SUF - Swap Unhandled Fees

Criticality	Critical
Location	NFTSTRIKE.sol#L634
Status	Unresolved

### Description

The contract's implementation applies fees to all transactions, including those involving the router address. However, when the contract executes the `swapExactTokensForETHSupportingFeeOnTransferTokens` function, which internally invokes the `swapExactTokensForETH` method of the Uniswap router, it does not account for the fees applied to the transaction. This inconsistency in fee handling can lead to issues, specifically causing sell transactions to revert during the `swapBack` process.

```
function swapExactTokensForETHSupportingFeeOnTransferTokens(
    uint256 amountIn,
    uint256 amountOutMin,
    address[] memory path
) internal {
    ...
    uniswapV2Router.swapExactTokensForETH(
        amountIn,
        amountOutMin,
        path,
        address(this),
        block.timestamp
    );
}
```

## Recommendation

The team is advised to review and refactor the

`swapExactTokensForETHSupportingFeeOnTransferTokens` function, considering

the fees applied to the transaction. This may involve modifying the invoked function from the Uniswap router. The Uniswap router contract includes functions for such cases.

Alternatively, the team could make sure that the pair's address is excluded from fees to prevent reverts during the swap process.

## MAA - Missing Amount Approval

Criticality	Critical
Location	NFTSTRIKE.sol#L638,665
Status	Unresolved

### Description

The `swapExactTokensForETHSupportingFeeOnTransferTokens` and `addLiquidity` functions within the contract do not check or provide the required allowance of at least `amountIn` and `tokenAmount` respectively, as specified in the router documentation. This missing allowance verification can lead to a reversion when executing the `swapExactTokensForTokens` or `addLiquidityETH` functions within `swapBack`. Consequently, the execution flow of the contract may be disrupted, potentially causing unexpected reverts.

```
uniswapV2Router.swapExactTokensForETH(  
    amountIn,  
    amountOutMin,  
    path,  
    address(this),  
    block.timestamp  
);  
  
uniswapV2Router.addLiquidityETH{value: BNBAmount}(  
    address(this),  
    tokenAmount,  
    0,  
    0,  
    address(this),  
    block.timestamp  
);
```

## Recommendation

The team is advised to address this issue and ensure the smooth execution of the `swapBack` function. Specifically, the team should ensure that the contract has the required allowance for `amountIn` and `tokenAmount` before calling the `swapExactTokensForTokens` and `addLiquidityETH` functions within the `swapBack` process. This involves checking the allowance using the `approve` function for the respective tokens. By address this issue, the team can prevent reverts caused by missing allowances and ensure that the contract's execution flow remains uninterrupted during the `swapBack` process.

## RC - Redundant Checks

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L594,634
Status	Unresolved

### Description

The contract includes specific checks before calling the `swapExactTokensForETH` function of the Uniswap router. However, upon reviewing the Uniswap router's implementation, it's evident that these checks are already performed within the Uniswap router's function. Therefore, the duplicate checks in the contract's function are redundant and result in increased gas consumption for each transaction that executes the swap process, without providing additional security or benefits.

```
require(
    path[0] == address(this) && path[path.length - 1] == WETH,
    "Invalid path"
);
require(path.length > 1, "Invalid path");
uint256[] memory amounts = uniswapV2Router.getAmountsOut(
    amountIn,
    path
);
require(
    amounts[amounts.length - 1] >= amountOutMin,
    "Slippage too high"
);
```

### Recommendation

To optimize the contract's gas efficiency and remove redundancy, it is advisable to eliminate the duplicate checks in the `swapExactTokensForETHSupportingFeeOnTransferTokens` function. Instead, rely on the Uniswap router's built-in checks within its `swapExactTokensForETH` function, which are designed to ensure safe and efficient execution. By removing the redundant checks, you can streamline the contract's transaction processing and reduce gas costs while maintaining the necessary security measures implemented in the Uniswap router.

## CR - Code Repetition

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L494,498
Status	Unresolved

### Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
function isOwner(address account) public view returns (bool) {  
    return owner() == account;  
}  
  
function isAuthorized(address adr) public view returns (bool) {  
    return owner() == adr;  
}
```

### Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

## FSA - Fixed Swap Address

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L396
Status	Unresolved

### Description

The swap address is assigned once and it can not be changed. It is a common practice in decentralized exchanges to create new swap versions. A contract that cannot change the swap address may not be able to catch up to the upgrade. As a result, the contract will not be able to migrate to a new liquidity pool pair or decentralized exchange.

```
IUniswapV2Router02 private uniswapV2Router =  
IUniswapV2Router02(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);
```

### Recommendation

The team is advised to add the ability to change the pair and router address in order to cover potential liquidity pool migrations. It would be better to support multiple pair addresses so the token will be able to have the same behavior in all the decentralized liquidity pairs.

## RAS - Redundant Assert Statement

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L23
Status	Unresolved

### Description

The contract utilizes an `assert` 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.

```
function add(uint256 a, uint256 b) internal pure returns (uint256 c) {  
    c = a + b;  
    assert(c >= a);  
    return c;  
}
```

### Recommendation

It is recommended to remove the `assert` 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.



## RCS - Redundant Conditional Statement

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L515
Status	Unresolved

### Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

Within the contract's transfer flow, there is a conditional check that verifies if the recipient is not equal to both `uniswapV2Pair` and `DEAD`. However, this condition serves no practical purpose, as there is no associated code or behavior executed when the condition is met. This redundancy should be addressed to streamline the contract and enhance code readability.

```
if (recipient != uniswapV2Pair && recipient != DEAD) {}
```

### Recommendation

To improve the contract's efficiency and clarity, the team is advised to remove the redundant condition `if (recipient != uniswapV2Pair && recipient != DEAD) {}`. This will result in a cleaner and more concise codebase, focusing on essential logic and conditions.

## PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L619,624,629
Status	Unresolved

### Description

The contract sends funds to a `_marketingFeeReceiver`, `_buybackFeeReceiver`, and `_devFeeReceiver` as part of the transfer flow. These addresses can either be a wallet address or a contract. If the address 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.

```
(bool MarketingSuccess, ) = payable(_marketingFeeReceiver).call{
    value: amountBNBMarketing,
    gas: 30000
}("");
require(MarketingSuccess, "receiver rejected ETH transfer");
(bool BuyBackSuccess, ) = payable(_buybackFeeReceiver).call{
    value: amountBNBbuyback,
    gas: 30000
}("");
require(BuyBackSuccess, "receiver rejected ETH transfer");
(bool DevSuccess, ) = payable(_devFeeReceiver).call{
    value: amountBNBDev,
    gas: 30000
}("");
require(DevSuccess, "receiver rejected ETH transfer");
```

### 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 not allowing set contract addresses or by sending the funds in a non-revertable way.

## AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	NFTSTRIKE.sol
Status	Unresolved

### Description

The contract uses both the SafeMath library and native arithmetic operations. The SafeMath library is commonly used to mitigate vulnerabilities related to integer overflow and underflow issues. However, it was observed that the contract also employs native arithmetic operators (such as +, -, \*, /) in certain sections of the code.

The combination of SafeMath library and native arithmetic operations can introduce inconsistencies and undermine the intended safety measures. This discrepancy creates an inconsistency in the contract's arithmetic operations, increasing the risk of unintended consequences such as inconsistency in error handling, or unexpected behavior.

```
_balances[sender] = _balances[sender].sub(  
    amount  
);  
  
uint256 amountBNBDev = amountBNB -  
    amountBNBLiquidity -  
    amountBNBbuyback -  
    amountBNBMarketing;
```

### Recommendation

To address this finding and ensure consistency in arithmetic operations, it is recommended to standardize the usage of arithmetic operations throughout the contract. The contract should be modified to either exclusively use SafeMath library functions or entirely rely on native arithmetic operations, depending on the specific requirements and design considerations. This consistency will help maintain the contract's integrity and mitigate potential vulnerabilities arising from inconsistent arithmetic operations.

## TUU - Time Units Usage

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L564
Status	Unresolved

### Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
uint256 hour = 3600;
```

### Recommendation

It is a good practice to use the time units reserved keywords like `seconds`, `minutes`, `hours`, `days` and `weeks` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

## MEE - Missing Events Emission

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L720,721,722,723,724,732,732
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.

```
_liquidityFee = liquidityFee;  
_devFee = devFee;  
_buybackFee = buybackFee;  
_marketingFee = marketingFee;  
_totalFee = liquidityFee.add(buybackFee).add(marketingFee);  
_marketingFeeReceiver = marketingFeeReceiver;  
_buybackFeeReceiver = buybackFeeReceiver;
```

### 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.

## PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L735
Status	Unresolved

### Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `_swapThreshold` 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.

```
function setSwapBackSettings(bool enabled, uint256 amount)
    external
    onlyOwner
{
    _swapEnabled = enabled;
    _swapThreshold = amount;
}
```

### 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.

## MRM - Missing Revert Messages

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L521,715,7198,719
Status	Unresolved

### Description

The contract is missing error messages. These missing error messages are 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.

```
require(_balances[sender] > 0);  
require(  
    liquidityFee.add(buybackFee).add(marketingFee).add(devFee) < 25  
);  
require(_liquidityFee > 0);  
require(_totalFee > 0 );
```

### Recommendation

The team is advised to carefully review the source code in order to address these issues. To accelerate the debugging process and mitigate these issues, the team should use more specific and descriptive error messages.

## RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	NFTSTRIKE.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.

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

### Recommendation

The team is advised to remove the SafeMath library. 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/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes>.



## L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L364,376,382,390,392,393,394,396,397
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.

```
uint256 private _totalSupply = 10000000000 * (10**uint256(decimals))
uint256 private _feeDenominator = 100

address private _devFeeReceiver =
    0x1Fc4a0124cD269A8E13533e100aCA377A7975Cb2
bool private _isSwapBackEnabled
address private WETH = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2
address private DEAD = 0x00000000000000000000000000000000dEaD
address private ZERO = 0x0000000000000000000000000000000000000000
IUniswapV2Router02 private uniswapV2Router =
    IUniswapV2Router02(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D)
IUniswapV2Factory private uniswapV2Factory =
    IUniswapV2Factory(0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f)
```

### 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

<b>Criticality</b>	Minor / Informative
<b>Location</b>	NFTSTRIKE.sol#L124,392,393,394,562,661
<b>Status</b>	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.

```
function WETH() external pure returns (address);
address private WETH = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2
address private DEAD = 0x00000000000000000000000000000000dEaD
address private ZERO = 0x0000000000000000000000000000000000000000

function AntiDumpMultiplier() private view returns (uint256) {
    uint256 timeSinceStart = block.timestamp - _launchedAt;
    uint256 hour = 3600;
    if (timeSinceStart > 1 * hour) {
        return 1;
    } else {
        return 2;
    }
}

uint256 BNBAmount
```

## 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/v0.8.17/style-guide.html#naming-convention>.

## L05 - Unused State Variable

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L365,366,390,394
Status	Unresolved

### Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
uint256 private _maxTxAmount = (_totalSupply * 2) / 100
uint256 private _maxWalletSize = (_totalSupply * 2) / 100
bool private _isSwapBackEnabled
address private ZERO = 0x0000000000000000000000000000000000000000
```

### Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

## L07 - Missing Events Arithmetic

<b>Criticality</b>	Minor / Informative
<b>Location</b>	NFTSTRIKE.sol#L720,740
<b>Status</b>	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.

```
_liquidityFee = liquidityFee  
_swapThreshold = amount
```

### 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.

## L09 - Dead Code Elimination

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

### Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function swapExactTokensForTokensSupportingFeeOnTransferTokens(  
    uint256 amountIn,  
    uint256 amountOutMin,  
    address[] memory path,  
    address to,  
    uint256 deadline  
    ...  
    amountIn,  
    amountOutMin,  
    path,  
    to,  
    deadline  
);  
}
```

### Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

## L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L388
Status	Unresolved

### Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of precision.

```
uint256 private _swapThreshold = (_totalSupply / 1000) * 3
```

### Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.



## L16 - Validate Variable Setters

<b>Criticality</b>	Minor / Informative
<b>Location</b>	NFTSTRIKE.sol#L731,732
<b>Status</b>	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.

```
_marketingFeeReceiver = marketingFeeReceiver  
_buybackFeeReceiver = buybackFeeReceiver
```

### 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.

## L19 - Stable Compiler Version

<b>Criticality</b>	Minor / Informative
<b>Location</b>	NFTSTRIKE.sol#L2
<b>Status</b>	Unresolved

### Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.5;
```

### Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

## Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
<b>SafeMath</b>	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
<b>IERC20</b>	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
<b>Context</b>	Implementation			
	_msgSender	Internal		
	_msgData	Internal		

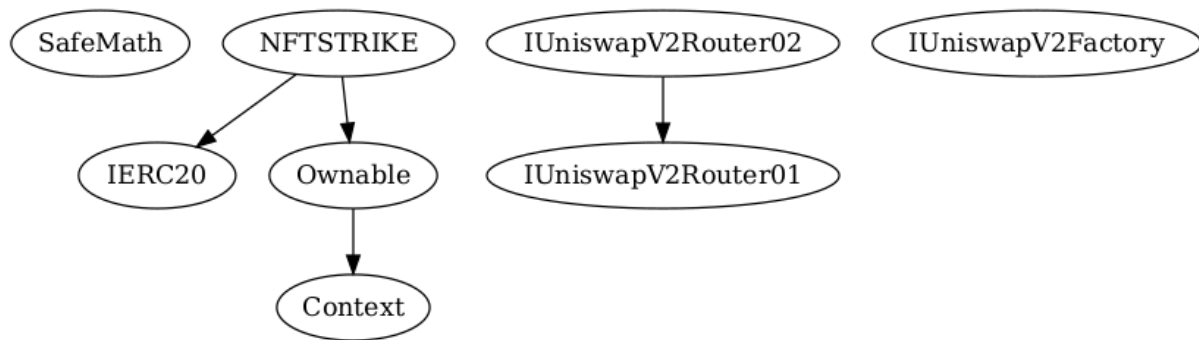
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
<b>IUniswapV2Router01</b>	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-

	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
<b>IUniswapV2Router02</b>	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
<b>IUniswapV2Factory</b>	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-

<b>NFTSTRIKE</b>	Implementation	IERC20, Ownable		
		Public	✓	Ownable
		External	Payable	-
	totalSupply	External		-
	balanceOf	Public		-
	transfer	External	✓	-
	allowance	External		-
	_approve	Internal	✓	
	approve	Public	✓	-
	transferFrom	External	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	isOwner	Public		-
	isAuthorized	Public		-
	_transfer	Internal	✓	
	_basicTransfer	Internal	✓	
	takeFee	Internal	✓	
	getTotalFee	Public		-
	AntiDumpMultiplier	Private		
	shouldSwapBack	Internal		
	swapBack	Internal	✓	
	swapExactTokensForETHSupportingFee OnTransferTokens	Internal	✓	

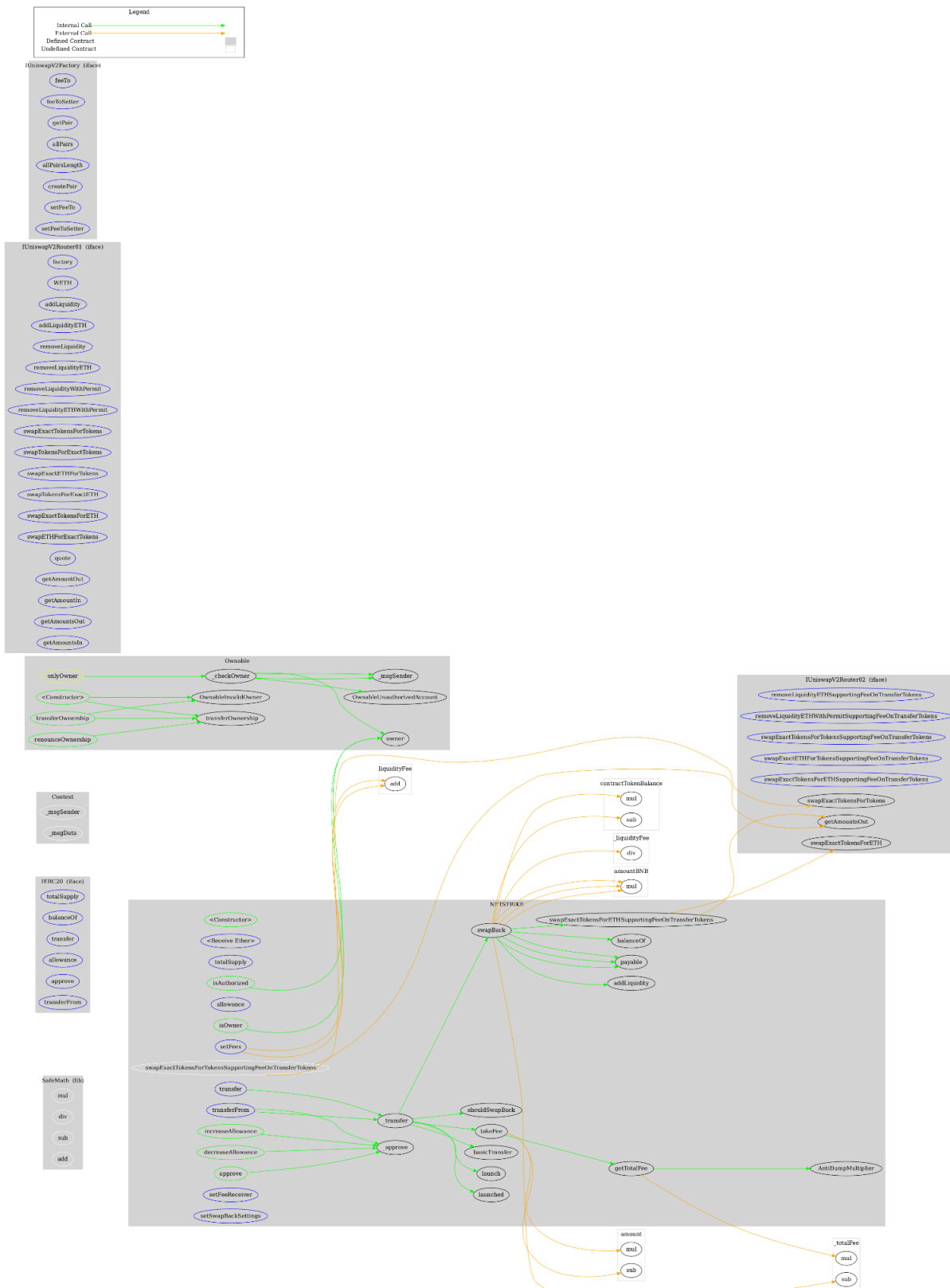
	addLiquidity	Private	✓	
	swapExactTokensForTokensSupporting FeeOnTransferTokens	Internal	✓	
	launched	Internal		
	launch	Internal	✓	
	setFees	External	✓	onlyOwner
	setFeeReceiver	External	✓	onlyOwner
	setSwapBackSettings	External	✓	onlyOwner

## Inheritance Graph





# Flow Graph



## Summary

NFT STRIKE contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. NFT STRIKE is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors, but two critical issues. The Contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a limit of max 24% fee. Additionally, the contract doubles the fees on sell transactions for the first hour.

## 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 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.



**The Cyberscope team**

<https://www.cyberscope.io>