Cyberscope

Audit Report

NFT STRIKE

November 2023

Network ETH

Address 0xa7e218AA56add84f46E5F0E45eeAc064F7e7Bb9

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	Unresolved
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	IPC	Incomplete Pair Creation	Unresolved
●	ZD	Zero Division	Unresolved
●	MRM	Missing Revert Messages	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	TUU	Time Units Usage	Unresolved
●	AOI	Arithmetic Operations Inconsistency	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RCS	Redundant Conditional Statement	Unresolved
●	RAS	Redundant Assert Statement	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	IDI	Immutable Declaration Improvement	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

Review

Contract Name	NFTSTRIKE
Compiler Version	v0.8.18+commit.87f61d96
Optimization	200 runs
Explorer	https://etherscan.io/address/0xa7e218aa56addb84f46e5f0e45eeac064f7e7bb9
Address	0xa7e218aa56addb84f46e5f0e45eeac064f7e7bb9
Network	ETH
Symbol	NFTS
Decimals	18
Total Supply	10,000,000,000

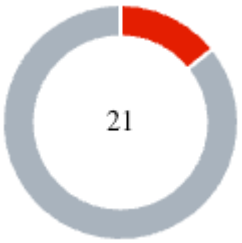
Audit Updates

Initial Audit	01 Nov 2023

Source Files

Filename	SHA256
NFTSTRIKE.sol	86fc34cac6cdc681c2d1e4d770b401e92bd04e95e297080ef1a8cfbee123229d

Findings Breakdown



Critical	3
Medium	0
Minor / Informative	18

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	3	0	0	0
Medium	0	0	0	0
Minor / Informative	18	0	0	0

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	NFTSTRIKE.sol#L780
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `setFees` function with a high percentage value.

```
function setFees(
    uint256 liquidityFee,
    uint256 buybackFee,
    uint256 devFee,
    uint256 marketingFee,
    uint256 feeDenominator
) external onlyOwner {
    _liquidityFee = liquidityFee;
    _devFee = devFee;
    _buybackFee = buybackFee;
    _marketingFee = marketingFee;
    _totalFee = liquidityFee.add(buybackFee).add(marketingFee);
    _feeDenominator = feeDenominator;
}
```

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.

IPC - Incomplete Pair Creation

Criticality	Critical
Location	NFTSTRIKE.sol#L811
Status	Unresolved

Description

The contract contains a function that does not handle the scenario where a pair already exists prior to its execution. If a pair for the given tokens has already been established, the `createPair` function will revert and not proceed with the creation of a new pair. As a result, if a pair has been previously set up before the function is invoked, the contract will encounter an error when trying to call the `createPair` function. This will prevent the successful execution, essentially leading the function to revert.

Additionally, the contract initializes the `uniswapV2Factory` variable during its deployment, which is a crucial component for creating Uniswap pairs. However, the contract does not create a pair between the Uniswap router and itself during this initialization. Instead, it relies on the owner to set the router and pair by calling the `setUniswapRouter` function. If the owner neglects to call this function or another user creates the pair before the owner and the contract attempts to execute its swap functionality, the transaction will revert, causing potential disruptions.

```
function setUniswapRouter(address routerAddress) external onlyOwner {
    uniswapV2Router = IUniswapV2Router02(routerAddress);
    uniswapV2Pair = uniswapV2Factory.createPair(address(this), WETH);
}
```

Recommendation

To mitigate the risks associated with attempting to create an already existing pair, it is recommended to implement a check to determine whether the pair already exists before proceeding to create a new pair. This can be achieved by utilizing the getPair function of the Factory contract to retrieve the address of the pair contract for the specified tokens. If the address returned by the getPair function is the zero address, it indicates that the pair does not exist, and the contract can proceed with the createPair function. Conversely, if a non-zero address is returned, it indicates that the pair already exists, and the createPair function will revert.

Lastly, to ensure the contract's functionality and avoid potential issues, the team is advised to perform the Uniswap pair creation within the contract's initialization during deployment. This eliminates the reliance on the owner to set the router and pair, thereby preventing transaction reverts due to missing configurations. By creating the Uniswap pair during deployment, the contract can function independently and seamlessly interact with the Uniswap ecosystem.

ZD - Zero Division

Criticality	Critical
Location	NFTSTRIKE.sol#L608,644
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.

Both `_totalFee` and `_feeDenominator` can be set to zero.

```
uint256 amountToLiquify = contractTokenBalance
    .mul(_liquidityFee)
    .div(_totalFee)
    .div(2);

uint256 feeAmount = amount
    .mul(getTotalFee(receiver == uniswapV2Pair))
    .div(_feeDenominator);
```

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.

MRM - Missing Revert Messages

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L579
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);
```

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.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L803
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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L772,777,791,792,799,800,813
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.

```
_maxTxAmount = amount;
_maxWalletSize = amount;
_liquidityFee = liquidityFee;
_devFee = devFee;
_buybackFee = buybackFee;
_marketingFee = marketingFee;
_totalFee = liquidityFee.add(buybackFee).add(marketingFee);
_feeDenominator = feeDenominator;
_marketingFeeReceiver = marketingFeeReceiver;
_buybackFeeReceiver = buybackFeeReceiver;
uniswapV2Pair = uniswapV2Factory.createPair(address(this), WETH);
```

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.

TUU - Time Units Usage

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L626
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.

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.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L680,685,690
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.

RCS - Redundant Conditional Statement

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L574
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 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.

RAS - Redundant Assert Statement

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L47
Status	Unresolved

Description

The contract utilizes a `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.

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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L458
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
uniswapV2Factory
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L419,437,444,445,447,448,449
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))

address private _devFeeReceiver =
        0x1Fc4a0124cD269A8E13533e100aCA377A7975Cb2
bool private _inSwap
bool private _isSwapBackEnabled
address private WETH = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2
address private DEAD = 0x000000000000000000000000000000000000dEaD
address private ZERO = 0x0000000000000000000000000000000000000000
```

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	NFTSTRIKE.sol#L261,447,448,449,624,722
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.

```
function WETH() external pure returns (address);
address private WETH = 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2
address private DEAD = 0x000000000000000000000000000000000000dEaD
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#L445,449
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.

```
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

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L787
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.

```
_liquidityFee = liquidityFee
```

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#L736
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#L443
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 prediction.

```
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

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L799,800
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.

```
_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

Criticality	Minor / Informative
Location	NFTSTRIKE.sol#L6
Status	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**
SafeMath	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		

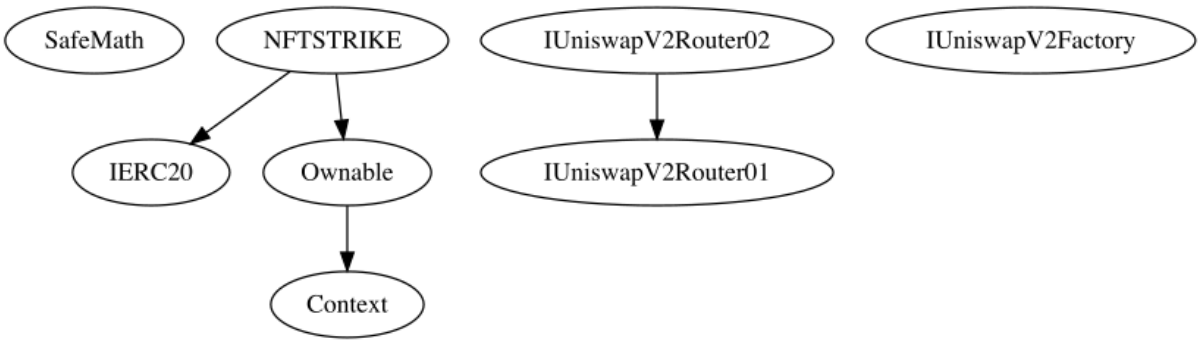
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	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		-
IUniswapV2Router02	Interface	IUniswapV2 Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-

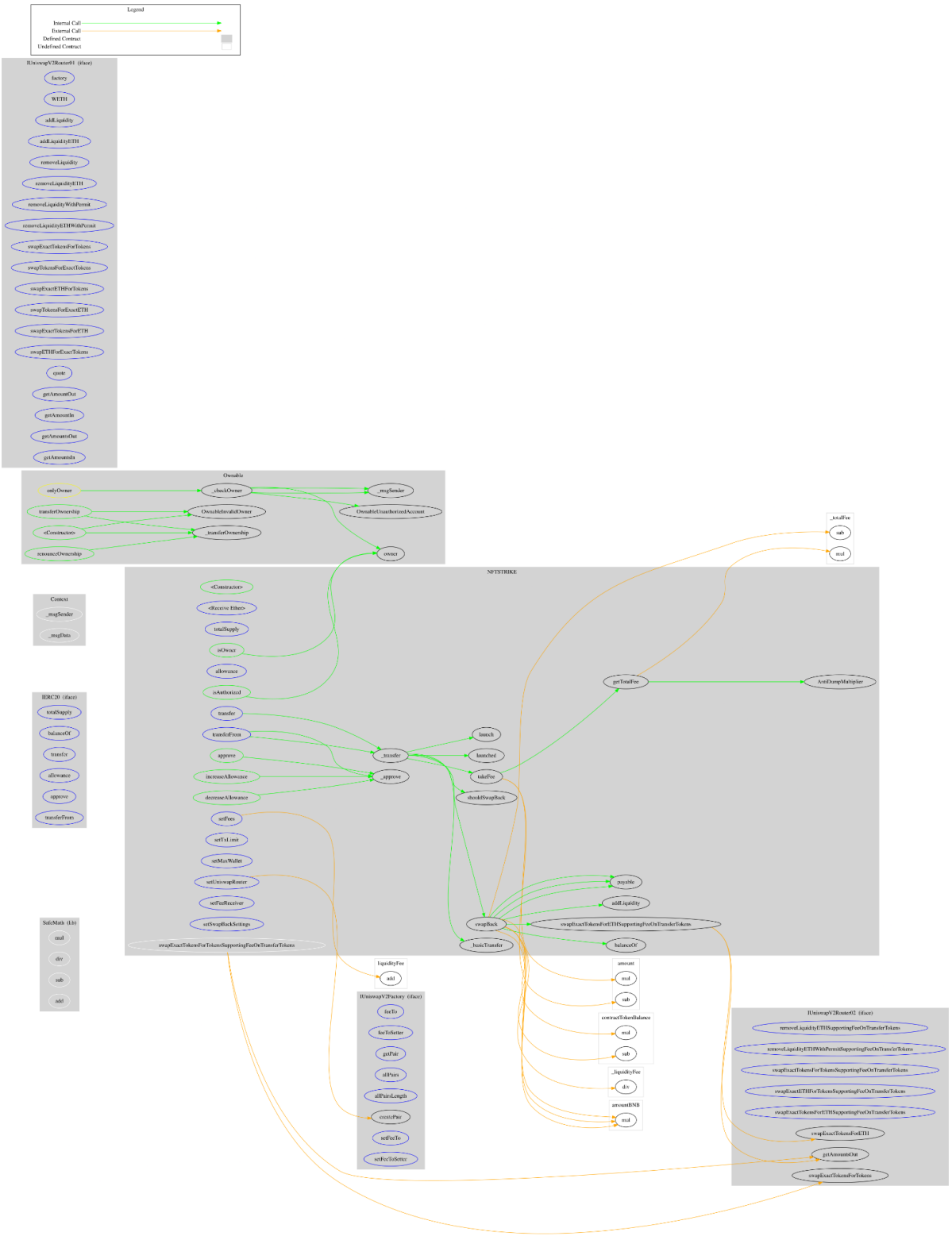
NFTSTRIKE	Implementation	IERC20, Ownable		
		Public	✓	Ownable
		External	Payable	-
	totalSupply	External		-
	balanceOf	Public		-
	transfer	External	✓	-
	allowance	External		-
	_approve	Internal	✓	
	approve	Public	✓	-
	transferFrom	External	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓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

	addLiquidity	Private	✓	
	swapExactTokensForTokensSupportingFeeOnTransferTokens	Internal	✓	
	launched	Internal		
	launch	Internal	✓	
	setTxLimit	External	✓	onlyOwner
	setMaxWallet	External	✓	onlyOwner
	setFees	External	✓	onlyOwner
	setFeeReceiver	External	✓	onlyOwner
	setSwapBackSettings	External	✓	onlyOwner
	setUniswapRouter	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IUniswapV2Router01 (iface)
factory
WETH
addLiquidity
addLiquidityETH
removeLiquidity
removeLiquidityETH
removeLiquidityWithPermit
removeLiquidityETHWithPermit
swapExactTokensForTokens
swapTokensForExactTokens
swapExactETHForTokens
swapTokensForExactETH
swapExactTokensForETH
swapETHForExactTokens
quote
getAmountOut
getAmountIn
getAmountsOut
getAmountsIn
Ownable
onlyOwner
transferOwnership
<Constructor>
renounceOwnership
_checkOwner
OwnableInvalidOwner
_transferOwnership
_msgSender
OwnableUnauthorizedAccount
owner
Context
_msgSender
_msgData
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
SafeMath (Lib)
mul
div
sub
add
NFTSTRIKE
<Constructor>
<Receive Ether>
totalSupply
isOwner
allowance
isAuthorized
transfer
transferFrom
approve
increaseAllowance
decreaseAllowance
setFees
setTxLimit
setMaxWallet
setUniswapRouter
setFeeReceiver
setSwapBackSettings
swapExactTokensForTokensSupportingFeeOnTransferTokens
_transfer
_approve
launch
launched
takeFee
shouldSwapBack
swapBack
_basicTransfer
getTotalFee
AntiDumpMultiplier
_totalFee
sub
mul
payable
addLiquidity
swapExactTokensForETHSupportingFeeOnTransferTokens
balanceOf
liquidityFee
add
amount
mul
sub
contractTokenBalance
mul
sub
_liquidityFee
div
amountBNB
mul
IUniswapV2Factory (iface)
feeTo
feeToSetter
getPair
allPairs
allPairsLength
createPair
setFeeTo
setFeeToSetter
IUniswapV2Router02 (iface)
removeLiquidityETHSupportingFeeOnTransferTokens
removeLiquidityETHWithPermitSupportingFeeOnTransferTokens
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactETHForTokensSupportingFeeOnTransferTokens
swapExactTokensForETHSupportingFeeOnTransferTokens
swapExactTokensForETH
getAmountsOut
swapExactTokensForTokens

Summary

NFT STRIKE 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 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 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