SCRL

Full Audit Report

Never Back Down Security Assessment

Never Back Down Security Assessment

FULL AUDIT REPORT

Security Assessment by SCRL on **Tuesday, January 23, 2024**

SCRL is deliver a security solution for Web3 projects by expert security researchers.

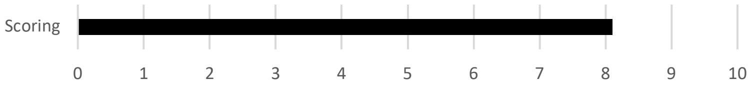
Executive Summary

For this security assessment, SCRL received a request on Monday, January 22, 2024

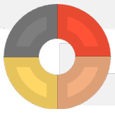
Client	Language	Audit Method	Confidential	Network Chain	Contract
Never Back Down	**Solidity**	**Whitebox**	**Public**	**BNB-Chain**	**0x199D07aa6723e9324F44f89885101FF79e11919A**

Report Version	Twitter	Telegram	Website
1.1	**https://x.com/NbdToken**	**https://t.me/NBDToken**	**https://neverbackdown.space/**

Scoring:



Vulnerability Summary



9	9	0	0	9	0
Total Findings	Unresolved	Resolved	Mitigate	Acknowledge	Decline

0	Critical		Critical severity is assigned to security vulnerabilities that pose a severe threat to the smart contract and the entire blockchain ecosystem.
1	High	1 Unresolved	High-severity issues should be addressed quickly to reduce the risk of exploitation and protect users' funds and data.
1	Medium	1 Unresolved	It's essential to fix medium-severity issues in a reasonable timeframe to enhance the overall security of the smart contract.
1	Low	1Unresolved	While low-severity issues can be less urgent, it's still advisable to address them to improve the overall security posture of the smart contract.
0	Very Low		Very Low severity is used for minor security concerns that have minimal impact and are generally of low risk.
1	Informational	1 Unresolved	Used to categorize security findings that do not pose a direct security threat to the smart contract or its users. Instead, these findings provide additional information, recommendations
5	Gas-optimization	5 Unresolved	Suggestions for more efficient algorithms or improvements in gas usage, even if the current code is already secure.

Audit Scope:

File	SHA-1 Hash
src/NeverBackDown.sol	97ad02bd4e594815ad775cc90ce7f488a47b624b

Audit Version History:

Version	Date	Description
1.0	Tuesday, January 23, 2024	Preliminary Report
1.1	Tuesday, January 23, 2024	Full Audit Report

Audit information:

Request Date	Audit Date	Re-assessment Date
Monday, January 22, 2024	Tuesday, January 23, 2024	-

Smart Contract Audit Summary

SCRL

SCRL has assessed
the security of this smart contract.

The results of the security
assessment revealed

No Critical Vulnerabilities.

Full Audit Report by SCRL on January 23, 2024

Security Assessment Author

Auditor:	**Mark K.**	[Security Researcher | Redteam]
	Kevin N.	[Security Researcher | Web3 Dev]
	Yusheng T.	[Security Researcher | Incident Response]
Document Approval:	**Ronny C.**	CTO & Head of Security Researcher
	Chinnakit J.	CEO & Founder

Digital Sign

Digitally signed by Chinnakit J.
DN: cn=Chinnakit J., o=SCRL,
ou=CEO & Founder,
email=contact@scrl.io, c=TH
Date: 2024.01.23 21:44:33
+07'00'

Disclaimer

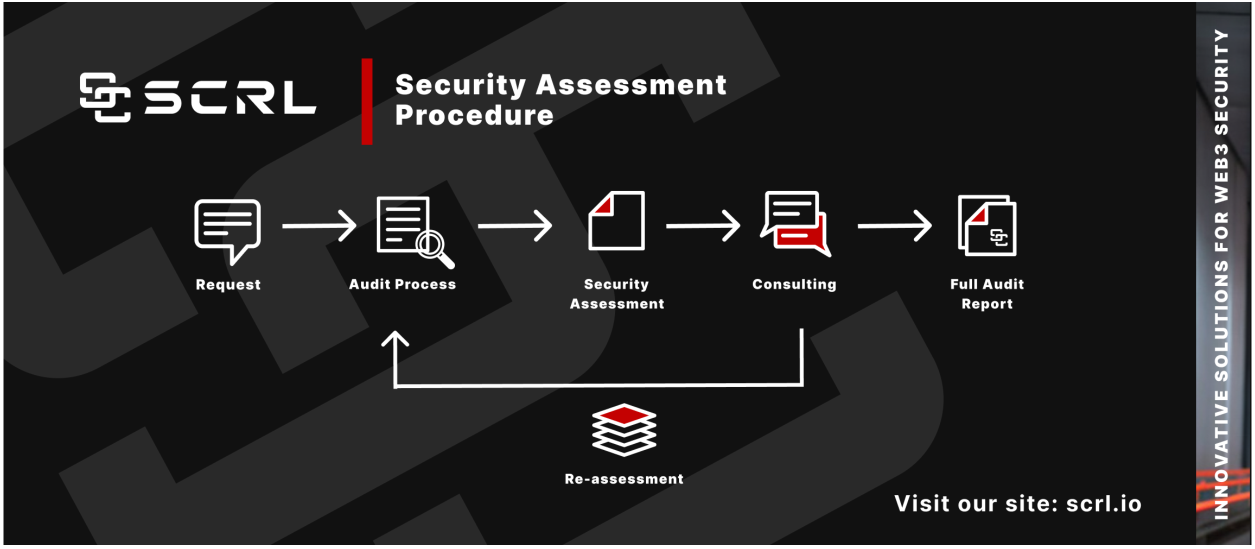
Regarding this security assessment, there are no guarantees about the security of the program instruction received from the client is hereinafter referred to as “**Source code**”.
And **SCRL** hereinafter referred to as “**Service Provider**”, the **Service Provider** will not be held liable for any legal liability arising from errors in the security assessment. The responsibility will be the responsibility of the **Client**, hereinafter referred to as “**Service User**” and the
Service User agrees not to be held liable to the **service provider** in any case. By contract
Service Provider to conduct security assessments with integrity with professional ethics, and transparency to deliver security assessments to users The **Service Provider** has the right to postpone the delivery of the security assessment. If the security assessment is delayed whether caused by any reason and is not responsible for any delayed security assessments.
If **the service provider** finds a vulnerability The **service provider** will notify the **service user** via the Preliminary Report, which will be kept confidential for security. The **service provider** disclaims responsibility in the event of any attacks occurring whether before conducting a security assessment. Or happened later All responsibility shall be sole with the **service user**.

Security Assessment Is Not Financial/Investment Advice Any loss arising from any investment in any project is the responsibility of the investor.

SCRL disclaims any liability incurred. Whether it's Rugpull, Abandonment, Soft Rugpull, Exploit, Exit Scam.

Security Assessment Procedure

1. **Request** The client must submit a formal request and follow the procedure. By submitting the source code and agreeing to the terms of service.
2. **Audit Process** Check for vulnerabilities and vulnerabilities from source code obtained by experts using formal verification methods, including using powerful tools such as Static Analysis, SWC Registry, Dynamic Security Analysis, Automated Security Tools, CWE, Syntax & Parameter Check with AI ,WAS (Warning Avoidance System a python script tools powered by SCRL).
3. **Security Assessment** Deliver Preliminary Security Assessment to clients to acknowledge the risks and vulnerabilities.
4. **Consulting** Discuss on risks and vulnerabilities encountered by clients to apply to their source code to mitigate risks.
    a. **Re-assessment** Reassess the security when the client implements the source code improvements and if the client is satisfied with the results of the audit. We will proceed to the next step.
5. **Full Audit Report** SCRL provides clients with official security assessment reports informing them of risks and vulnerabilities. Officially and it is assumed that the client has been informed of all the information.



Risk Rating

Risk rating using this commonly defined: $Risk\ rating\ =\ impact\ *\ confidence$

Impact	The severity and potential impact of an attacker attack
Confidence	Ensuring that attackers expose and use this vulnerability

Confidence Impact [Likelihood]	Low	Medium	High
Low	**Very Low**	**Low**	**Medium**
Medium	**Low**	**Medium**	**High**
High	**Medium**	**High**	**Critical**

Severity is a risk assessment It is calculated from the Impact and Confidence values using the following calculation methods,

$Risk\ rating\ =\ impact\ *\ confidence$

It is categorized into

7 categories severity based

Gas-optimization	Informational	Very Low	Low	Medium	High	Critical

For **Informational** & **Non-class/Optimization/Best-practices will** <u>not be counted</u> as **severity**

Category

Centralization

Centralization Risk is The risk incurred by a sole proprietor, such as the Owner being able to change something without permission

Economics Risk

Economics Risk is Risks that may affect the economic mechanism system, such as the ability to increase Mint token

Logical Issue

Logical Issue is that can cause errors to core processing, such as any prior operations that cause background processes to crash.

Authorization

Authorization is Possible pitfalls from weak coding allows unrelated people to take any action to modify the values.

Mathematical

Mathematical Any erroneous arithmetic operations affect the operation of the system or lead to erroneous values.

Naming Conventions

Naming Conventions naming variables that may affect code understanding or naming inconsistencies

Security Risk

Security Risk of loss or damage if it's no mitigate

Coding Style

Coding Style is Tips coding for efficiency performance

Best Practices

Best Practices is suggestions for improvement

Optimization

Optimization is performance improvement

Gas Optimization

Gas Optimization is increase performance to avoid expensive gas

Dead Code

Dead Code having unused code This may result in wasted resources and gas fees.

Table Of Content

Summary

- Executive Summary
- CVSS Scoring
- Vulnerability Summary
- Audit Scope
- Audit Version History
- Audit Information
- Smart Contract Audit Summary
- Security Assessment Author
- Digital Sign
- Disclaimer
- Security Assessment Procedure
- Risk Rating
- Category

Source Code Detail

- Dependencies / External Imports
- Visibility, Mutability, Modifier function testing

Vulnerability Finding

- Vulnerability
- SWC Findings
- Contract Description
- Inheritance Relational Graph
- UML Diagram

About SCRL

Source Units in Scope

Source Units Analyzed: **1**

Source Units in Scope: **1** (**100%**)

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝 🔍	src/NeverBackDown.sol	2	5	552	496	402	8	336	🖥💰📤 🔆
📝 🔍	**Totals**	**2**	**5**	**552**	**496**	**402**	**8**	**336**	🖥💰📤 🔆

Legend: [➖]

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

Visibility, Mutability, Modifier function testing

Components

📝Contracts	📚Libraries	🔍Interfaces	🎨Abstract
2	0	5	0

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

🌐Public	💰Payable
44	3

External	Internal	Private	Pure	View
18	44	19	3	25

StateVariables

Total	🌐Public
28	11

Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
0.8.18		yes	yes (1 asm blocks)	

📤 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧮 Uses Hash Functions	🔖 ECRecover	🌀 New/Create/Create2
yes					

♻️ TryCatch	Σ Unchecked

Dependencies / External Imports

Dependency / Import Path	Count

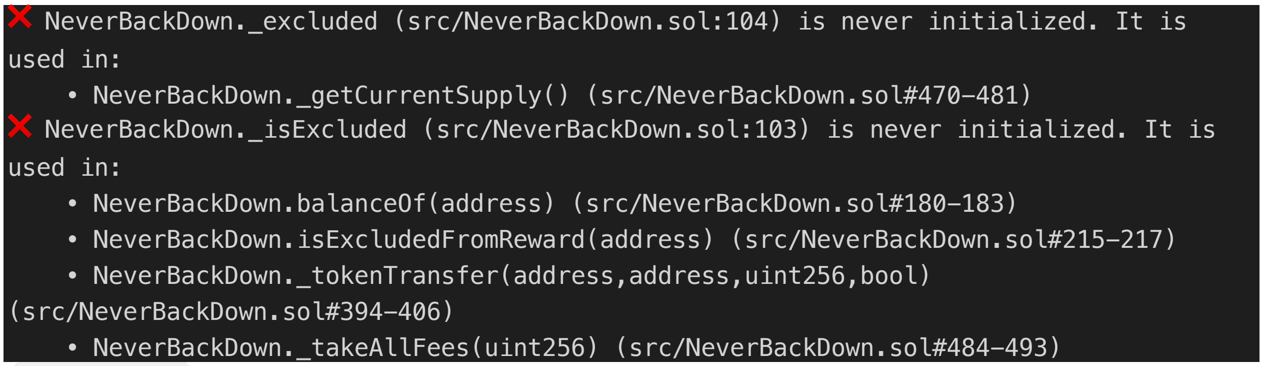
Vulnerability Findings

ID	Vulnerability Detail	Severity	Category	Status
SEC-01	Uninitialized state variables (uninitialized-state)	High	Best Practices	Acknowledge
SEC-02	Centralization Risk	Medium	Centralization	Acknowledge
SEC-03	Missing Events Arithmetic (events-maths)	Low	Mathematical	Acknowledge
SEC-04	Function initializing state variables (function-init-state)	Informational	Best Practices	Acknowledge
GAS-01	Using bools for storage incurs overhead	Gas-optimization	Gas Optimization	Acknowledge
GAS-02	Cache array length outside of loop	Gas-optimization	Gas Optimization	Acknowledge
GAS-03	Use Custom Errors	Gas-optimization	Gas Optimization	Acknowledge
GAS-04	Long revert strings	Gas-optimization	Gas Optimization	Acknowledge
GAS-05	Use != 0 instead of > 0 for unsigned integer comparison	Gas-optimization	Gas Optimization	Acknowledge

SEC-01: Uninitialized state variables (uninitialized-state)

Vulnerability Detail	Severity	Location	Category	Status
Uninitialized state variables (uninitialized-state)	High	Check on finding	Best Practices	**Acknowledge**

Finding:



```
❌ NeverBackDown._excluded (src/NeverBackDown.sol:104) is never initialized. It is
used in:
    • NeverBackDown._getCurrentSupply() (src/NeverBackDown.sol#470-481)
❌ NeverBackDown._isExcluded (src/NeverBackDown.sol:103) is never initialized. It is
used in:
    • NeverBackDown.balanceOf(address) (src/NeverBackDown.sol#180-183)
    • NeverBackDown.isExcludedFromReward(address) (src/NeverBackDown.sol#215-217)
    • NeverBackDown._tokenTransfer(address,address,uint256,bool)
(src/NeverBackDown.sol#394-406)
    • NeverBackDown._takeAllFees(uint256) (src/NeverBackDown.sol#484-493)
```

Recommendation:
Initialize all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero to improve code readability.

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables

Alleviation:
Never Back Down team has acknowledged this issue.

SEC-02: Centralization Risk

Vulnerability Detail	Severity	Location	Category	Status
Centralization Risk	Medium	Check on finding	Centralization	**Acknowledge**

Finding:

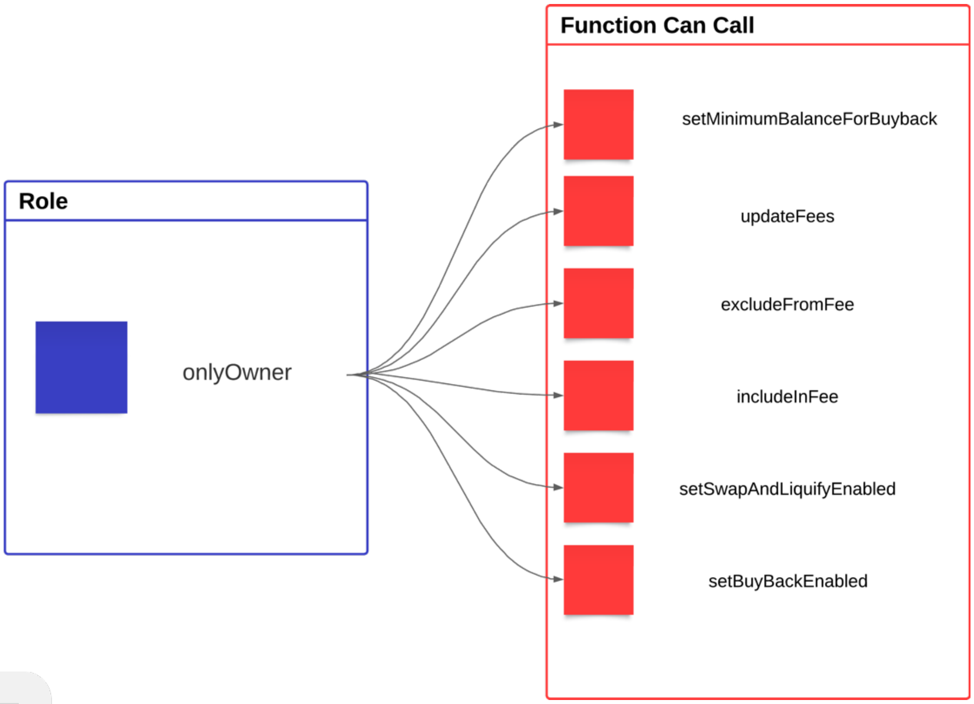
```
97: contract NeverBackDown is IERC20, Ownable

255:     function setMinimumBalanceForBuyback(uint256 _amount) public onlyOwner

361:     function updateFees(uint256 liquidityFee, uint256 buybackFee, uint256
teamFee) public onlyOwner

507:     function excludeFromFee(address account) public onlyOwner {

511:     function includeInFee(address account) public onlyOwner

540:     function setSwapAndLiquifyEnabled(bool _enabled) public onlyOwner {

545:     function setBuyBackEnabled(bool _enabled) public onlyOwner {
```

Explain Function Capability:

The contract provides several functions:

1. setMinimumBalanceForBuyback(uint256 _amount):
   - Purpose: Allows the contract owner to set the minimum balance required for the buyback mechanism to be triggered.
   - Usage: The _amount parameter represents the new minimum balance required for buyback.
2. updateFees(uint256 liquidityFee, uint256 buybackFee, uint256 teamFee):
   - Purpose: Enables the contract owner to update the fee structure of the token.
   - Usage: The parameters liquidityFee, buybackFee, and teamFee represent the new values for the respective fees.
3. excludeFromFee(address account):
   - Purpose: Allows the contract owner to exclude a specific address from paying transaction fees.
   - Usage: The account parameter represents the address that will be excluded from fees.
4. includeInFee(address account):
   - Purpose: Allows the contract owner to include a previously excluded address in the list of addresses subject to transaction fees.
   - Usage: The account parameter represents the address that will be included in fees.
5. setSwapAndLiquifyEnabled(bool _enabled):
   - Purpose: Permits the contract owner to enable or disable the automatic swapping of tokens for liquidity.
   - Usage: The _enabled parameter determines whether the swapping mechanism is turned on (true) or off (false).
6. setBuyBackEnabled(bool _enabled):
   - Purpose: Allows the contract owner to enable or disable the buyback mechanism.
   - Usage: The _enabled parameter determines whether the buyback mechanism is turned on (true) or off (false).

Centralization Risk



Recommendation:

In terms of timeframes, there are three categories: short-term, long-term, and permanent.

For short-term solutions, a combination of timelock and multi-signature (2/3 or 3/5) can be used to mitigate risk by delaying sensitive operations and avoiding a single point of failure in key management. This includes implementing a timelock with a reasonable latency, such as 48 hours, for privileged operations; assigning privileged roles to multi-signature wallets to prevent private key compromise; and sharing the timelock contract and multi-signer addresses with the public via a medium/blog link.

For long-term solutions, a combination of timelock and DAO can be used to apply decentralization and transparency to the system. This includes implementing a timelock with a reasonable latency, such as 48 hours, for privileged operations; introducing a DAO/governance/voting module to increase transparency and user involvement; and sharing the timelock contract, multi-signer addresses, and DAO information with the public via a medium/blog link.

Finally, permanent solutions should be implemented to ensure the ongoing security and protection of the system.

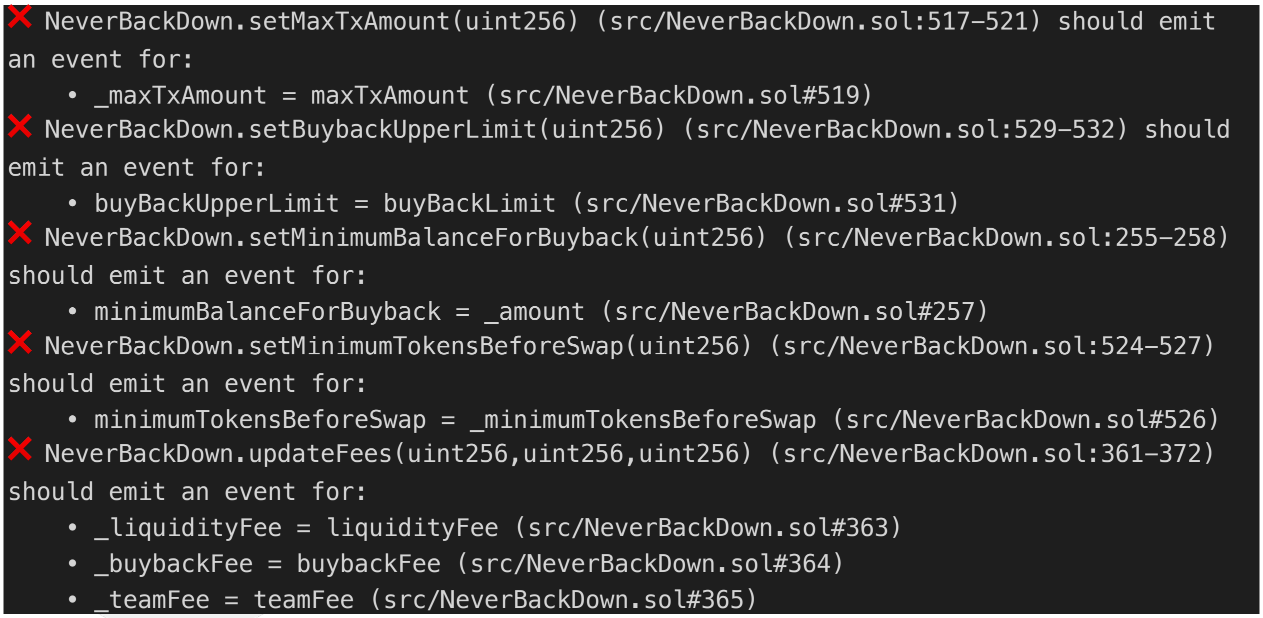
Alleviation:

Never Back Down team has acknowledged this issue.

SEC-03: Missing Events Arithmetic (events-maths)

Vulnerability Detail	Severity	Location	Category	Status
Missing Events Arithmetic (events-maths)	Low	Check on finding	Mathematical	Acknowledge

Finding:



```
❌ NeverBackDown.setMaxTxAmount(uint256) (src/NeverBackDown.sol:517-521) should emit
an event for:
    • _maxTxAmount = maxTxAmount (src/NeverBackDown.sol#519)
❌ NeverBackDown.setBuybackUpperLimit(uint256) (src/NeverBackDown.sol:529-532) should
emit an event for:
    • buyBackUpperLimit = buyBackLimit (src/NeverBackDown.sol#531)
❌ NeverBackDown.setMinimumBalanceForBuyback(uint256) (src/NeverBackDown.sol:255-258)
should emit an event for:
    • minimumBalanceForBuyback = _amount (src/NeverBackDown.sol#257)
❌ NeverBackDown.setMinimumTokensBeforeSwap(uint256) (src/NeverBackDown.sol:524-527)
should emit an event for:
    • minimumTokensBeforeSwap = _minimumTokensBeforeSwap (src/NeverBackDown.sol#526)
❌ NeverBackDown.updateFees(uint256,uint256,uint256) (src/NeverBackDown.sol:361-372)
should emit an event for:
    • _liquidityFee = liquidityFee (src/NeverBackDown.sol#363)
    • _buybackFee = buybackFee (src/NeverBackDown.sol#364)
    • _teamFee = teamFee (src/NeverBackDown.sol#365)
```

Recommendation:
Emit an event for critical parameter changes.

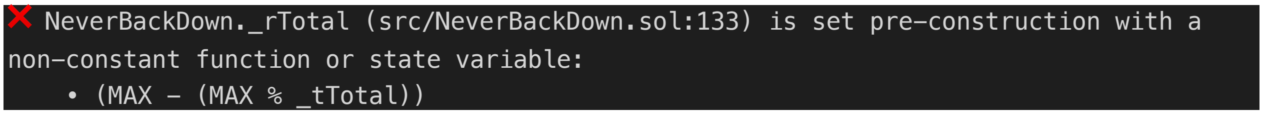
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

Alleviation:
Never Back Down team has acknowledged this issue.

SEC-04: Function initializing state variables (function-init-state)

Vulnerability Detail	Severity	Location	Category	Status
Function initializing state variables (function-init-state)	Informational	Check on finding	Best Practices	**Acknowledge**

Finding:



```
❌ NeverBackDown._rTotal (src/NeverBackDown.sol:133) is set pre-construction with a
non-constant function or state variable:
     • (MAX - (MAX % _tTotal))
```

Recommendation:

Remove any initialization of state variables via non-constant state variables or function calls. If variables must be set upon contract deployment, locate initialization in the constructor instead.

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#function-initializing-state

Alleviation:

Never Back Down team has acknowledged this issue.

GAS-01: Using bools for storage incurs overhead

Vulnerability Detail	Severity	Location	Category	Status
Using bools for storage incurs overhead	-	Check on finding	Gas Optimization	**Acknowledge**

Finding:

```
File: NeverBackDown.sol

102:      mapping (address => bool) private _isExcludedFromFee;

103:      mapping (address => bool) private _isExcluded;

115:      bool public swapAndLiquifyEnabled = true;

116:      bool public buyBackEnabled = true;
```

Recommendation:

Use uint256(1) and uint256(2) for true/false to avoid a Gwarmaccess (100 gas), and to avoid Gsset (20000 gas) when changing from 'false' to 'true', after having been 'true' in the past. See [source](https://github.com/OpenZeppelin/openzeppelin-contracts/blob/58f635312aa21f947cae5f8578638a85aa2519f5/contracts/security/ReentrancyGuard.sol#L23-L27).

Alleviation:

Never Back Down team has acknowledged this issue.

GAS-02: Cache array length outside of loop

Vulnerability Detail	Severity	Location	Category	Status
Cache array length outside of loop	-	Check on finding	Gas Optimization	**Acknowledge**

Finding:

```
File: NeverBackDown.sol

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

Recommendation:

If not cached, the solidity compiler will always read the length of the array during each iteration. That is, if it is a storage array, this is an extra sload operation (100 additional extra gas for each iteration except for the first) and if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first).

Alleviation:

Never Back Down team has acknowledged this issue.

GAS-03: Long revert strings

Vulnerability Detail	Severity	Location	Category	Status
Long revert strings	-	Check on finding	Gas Optimization	Acknowledge

Finding:

```
File: NeverBackDown.sol

45:         require(newOwner != address(0), "Ownable: new owner is the zero address");

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

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

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

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

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

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

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

Alleviation:

Never Back Down team has acknowledged this issue.

GAS-04: Use != 0 instead of > 0 for unsigned integer comparison

Vulnerability Detail	Severity	Location	Category	Status
Use != 0 instead of > 0 for unsigned integer comparison	-	Check on finding	Gas Optimization	**Acknowledge**

Finding:

```
File: NeverBackDown.sol

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

273:         if (!inSwapAndLiquify && swapAndLiquifyEnabled && from != uniswapV2Pair
&& balanceOf(uniswapV2Pair)>0)

314:         if (amount > 0) {

```
```

Alleviation:

Never Back Down team has acknowledged this issue.
```

SWC Findings

ID	Title	Scanning	Result
SWC-100	Function Default Visibility	Complete	No risk
SWC-101	Integer Overflow and Underflow	Complete	No risk
SWC-102	Outdated Compiler Version	Complete	No risk
SWC-103	Floating Pragma	Complete	No risk
SWC-104	Unchecked Call Return Value	Complete	No risk
SWC-105	Unprotected Ether Withdrawal	Complete	No risk
SWC-106	Unprotected SELFDESTRUCT Instruction	Complete	No risk
SWC-107	Reentrancy	Complete	No risk
SWC-108	State Variable Default Visibility	Complete	No risk
SWC-109	Uninitialized Storage Pointer	Complete	No risk
SWC-110	Assert Violation	Complete	No risk
SWC-111	Use of Deprecated Solidity Functions	Complete	No risk
SWC-112	Delegatecall to Untrusted Callee	Complete	No risk
SWC-113	DoS with Failed Call	Complete	No risk
SWC-114	Transaction Order Dependence	Complete	No risk
SWC-115	Authorization through tx.origin	Complete	No risk

SWC-116	Block values as a proxy for time	Complete	No risk
SWC-117	Signature Malleability	Complete	No risk
SWC-118	Incorrect Constructor Name	Complete	No risk
SWC-119	Shadowing State Variables	Complete	No risk
SWC-120	Weak Sources of Randomness from Chain Attributes	Complete	No risk
SWC-121	Missing Protection against Signature Replay Attacks	Complete	No risk
SWC-122	Lack of Proper Signature Verification	Complete	No risk
SWC-123	Requirement Violation	Complete	No risk
SWC-124	Write to Arbitrary Storage Location	Complete	No risk
SWC-125	Incorrect Inheritance Order	Complete	No risk
SWC-126	Insufficient Gas Griefing	Complete	No risk
SWC-127	Arbitrary Jump with Function Type Variable	Complete	No risk
SWC-128	DoS With Block Gas Limit	Complete	No risk
SWC-129	Typographical Error	Complete	No risk
SWC-130	Right-To-Left-Override control character (U+202E)	Complete	No risk
SWC-131	Presence of unused variables	Complete	No risk
SWC-132	Unexpected Ether balance	Complete	No risk

SWC-133	Hash Collisions With Multiple Variable Length Arguments	Complete	No risk
SWC-134	Message call with hardcoded gas amount	Complete	No risk
SWC-135	Code With No Effects	Complete	No risk
SWC-136	Unencrypted Private Data On-Chain	Complete	No risk

Contracts Description Table

Contract	**Type**	**Bases**		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
└	totalSupply	External ❗		NO ❗
└	balanceOf	External ❗		NO ❗
└	transfer	External ❗	🛑	NO ❗
└	allowance	External ❗		NO ❗
└	approve	External ❗	🛑	NO ❗
└	transferFrom	External ❗	🛑	NO ❗
Ownable	Implementation			
└		Public ❗	🛑	NO ❗
└	_msgSender	Internal 🔒		
└	owner	Public ❗		NO ❗
└	renounceOwnership	Public ❗	🛑	onlyOwner
└	transferOwnership	Public ❗	🛑	onlyOwner
└	isContract	Internal 🔒		
IUniswapV2Factory	Interface			
└	createPair	External ❗	🛑	NO ❗
IUniswapV2Pair	Interface			

Contract	Type	Bases		
└	factory	External ❗		NO ❗
IUniswapV2 Router01	Interface			
└	factory	External ❗		NO ❗
└	WETH	External ❗		NO ❗
└	addLiquidityETH	External ❗	💵	NO ❗
IUniswapV2 Router02	Interface	IUniswapV2 Router01		
└	swapExactETHForTokensSupportingFeeOnTransferTokens	External ❗	💵	NO ❗
└	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🛑	NO ❗
NeverBackDown	Implementation	IERC20, Ownable		
└		Public ❗	🛑	NO ❗
└	name	Public ❗		NO ❗
└	symbol	Public ❗		NO ❗
└	decimals	Public ❗		NO ❗
└	totalSupply	Public ❗		NO ❗
└	balanceOf	Public ❗		NO ❗
└	transfer	Public ❗	🛑	NO ❗
└	allowance	Public ❗		NO ❗
└	approve	Public ❗	🛑	NO ❗
└	transferFrom	Public ❗	🛑	NO ❗

Contract	Type	Bases		
└	increaseAllowance	Public ❗️	🛑	NO❗️
└	decreaseAllowance	Public ❗️	🛑	NO❗️
└	isExcludedFromReward	Public ❗️		NO❗️
└	minimumTokensBeforeSwapAmount	Public ❗️		NO❗️
└	buyBackUpperLimitAmount	Public ❗️		NO❗️
└	reflectionFromToken	Public ❗️		NO❗️
└	tokenFromReflection	Public ❗️		NO❗️
└	_approve	Private 🔐	🛑	
└	setMinimumBalanceForBuyback	Public ❗️	🛑	onlyOwner
└	_transfer	Private 🔐	🛑	
└	swapTokens	Private 🔐	🛑	lockTheSwap
└	buyBackTokens	Private 🔐	🛑	lockTheSwap
└	swapTokensForEth	Private 🔐	🛑	
└	swapETHForTokens	Private 🔐	🛑	
└	addLiquidity	Private 🔐	🛑	
└	updateFees	Public ❗️	🛑	onlyOwner
└	removeAllFee	Internal 🔒	🛑	
└	restoreAllFee	Internal 🔒	🛑	
└	totalFee	Internal 🔒		
└	_tokenTransfer	Private 🔐	🛑	
└	_transferStandard	Private 🔐	🛑	

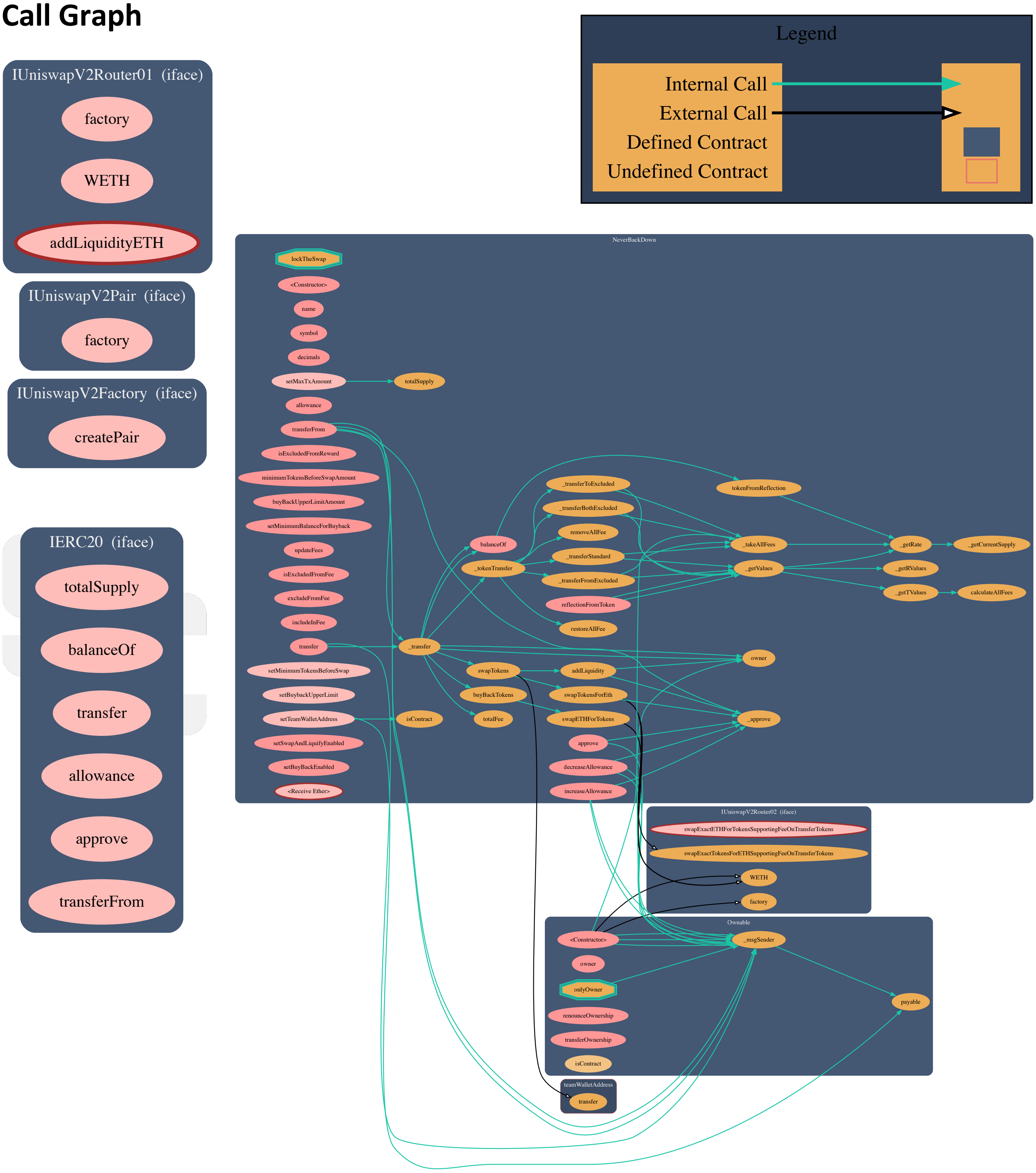
Contract	Type	Bases		
└	_transferToExcluded	Private 🔐	🛑	
└	_transferFromExcluded	Private 🔐	🛑	
└	_transferBothExcluded	Private 🔐	🛑	
└	_getValues	Private 🔐		
└	_getTValues	Private 🔐		
└	_getRValues	Private 🔐		
└	_getRate	Private 🔐		
└	_getCurrentSupply	Private 🔐		
└	_takeAllFees	Private 🔐	🛑	
└	calculateAllFees	Private 🔐		
└	isExcludedFromFee	Public ❗		NO ❗
└	excludeFromFee	Public ❗	🛑	onlyOwner
└	includeInFee	Public ❗	🛑	onlyOwner
└	setMaxTxAmount	External ❗	🛑	onlyOwner
└	setMinimumTokensBeforeSwap	External ❗	🛑	onlyOwner
└	setBuybackUpperLimit	External ❗	🛑	onlyOwner
└	setTeamWalletAddress	External ❗	🛑	onlyOwner
└	setSwapAndLiquifyEnabled	Public ❗	🛑	onlyOwner
└	setBuyBackEnabled	Public ❗	🛑	onlyOwner

Contract	Type	Bases		
└		External ❗️	💵	NO❗️

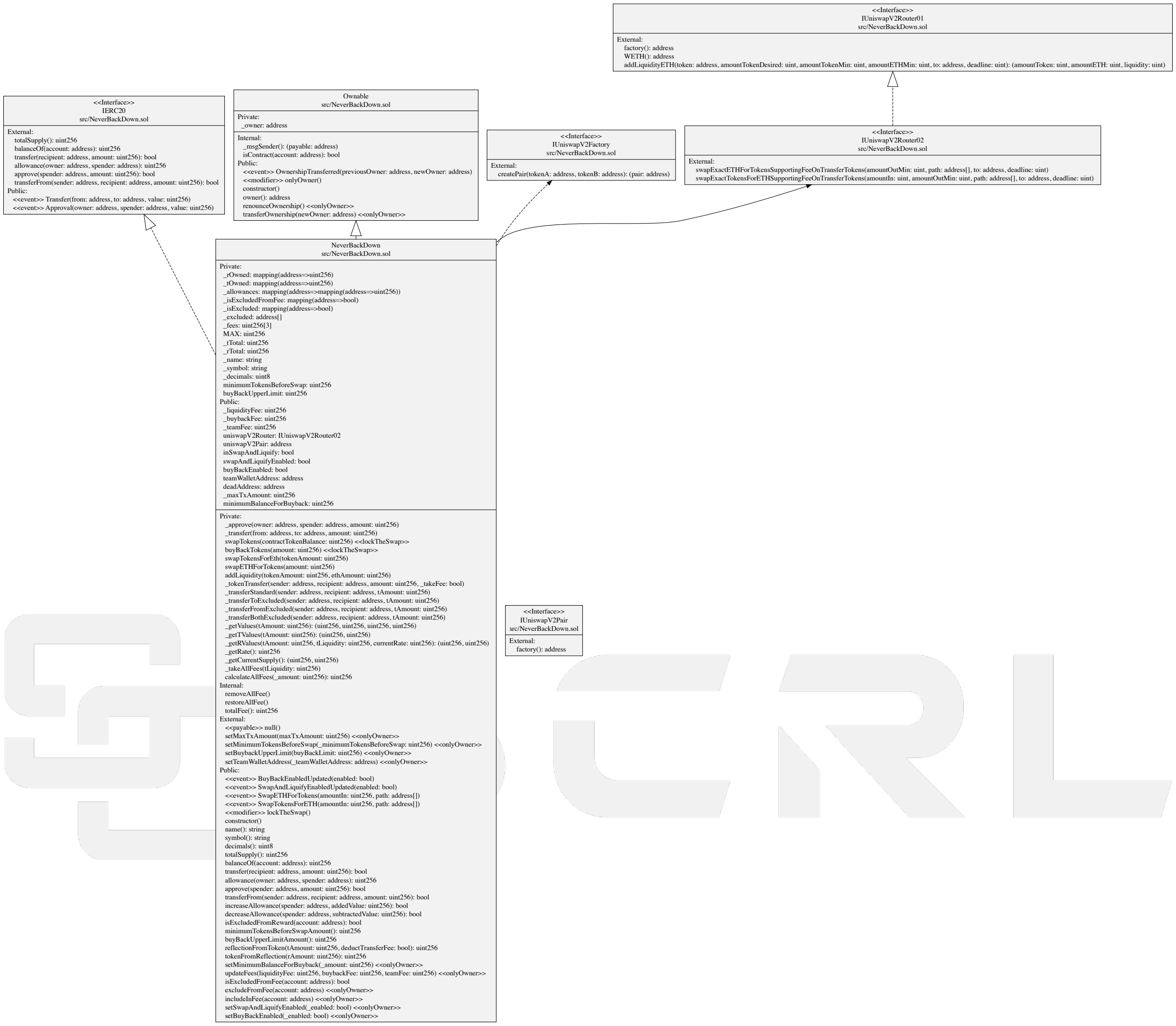
Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

Call Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IUniswapV2Router01 (iface)
factory
WETH
addLiquidityETH
IUniswapV2Pair (iface)
factory
IUniswapV2Factory (iface)
createPair
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
NeverBackDown
lockTheSwap
<Constructor>
name
symbol
decimals
setMaxTxAmount
totalSupply
allowance
transferFrom
isExcludedFromReward
minimumTokensBeforeSwapAmount
buyBackUpperLimitAmount
setMinimumBalanceForBuyback
updateFees
isExcludedFromFee
excludeFromFee
includeInFee
transfer
setMinimumTokensBeforeSwap
setBuybackUpperLimit
setTeamWalletAddress
setSwapAndLiquifyEnabled
setBuyBackEnabled
<Receive Ether>
isContract
_transfer
balanceOf
_tokenTransfer
swapTokens
buyBackTokens
totalFee
_transferToExcluded
_transferBothExcluded
removeAllFee
_transferStandard
_transferFromExcluded
reflectionFromToken
restoreAllFee
addLiquidity
swapTokensForEth
swapETHForTokens
approve
decreaseAllowance
increaseAllowance
tokenFromReflection
_takeAllFees
_getValues
owner
_approve
_getRate
_getRValues
_getTValues
_getCurrentSupply
calculateAllFees
IUniswapV2Router02 (iface)
swapExactETHForTokensSupportingFeeOnTransferTokens
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
factory
Ownable
<Constructor>
_msgSender
owner
onlyOwner
payable
renounceOwnership
transferOwnership
isContract
teamWalletAddress
transfer

UML Class Diagram



```
<<Interface>>
IUniswapV2Router01
src/NeverBackDown.sol

External:
   factory(): address
   WETH(): address
   addLiquidityETH(token: address, amountTokenDesired: uint, amountTokenMin: uint, amountETHMin: uint, to: address, deadline: uint): (amountToken: uint, amountETH: uint, liquidity: uint)
```

```
<<Interface>>
IERC20
src/NeverBackDown.sol

External:
   totalSupply(): uint256
   balanceOf(account: address): uint256
   transfer(recipient: address, amount: uint256): bool
   allowance(owner: address, spender: address): uint256
   approve(spender: address, amount: uint256): bool
   transferFrom(sender: address, recipient: address, amount: uint256): bool
Public:
  <<event>> Transfer(from: address, to: address, value: uint256)
  <<event>> Approval(owner: address, spender: address, value: uint256)
```

```
Ownable
src/NeverBackDown.sol

Private:
  _owner: address

Internal:
   _msgSender(): (payable: address)
   isContract(account: address): bool
Public:
   <<event>> OwnershipTransferred(previousOwner: address, newOwner: address)
   <<modifier>> onlyOwner()
   constructor()
   owner(): address
   renounceOwnership() <<onlyOwner>>
   transferOwnership(newOwner: address) <<onlyOwner>>
```

```
<<Interface>>
IUniswapV2Factory
src/NeverBackDown.sol

External:
   createPair(tokenA: address, tokenB: address): (pair: address)
```

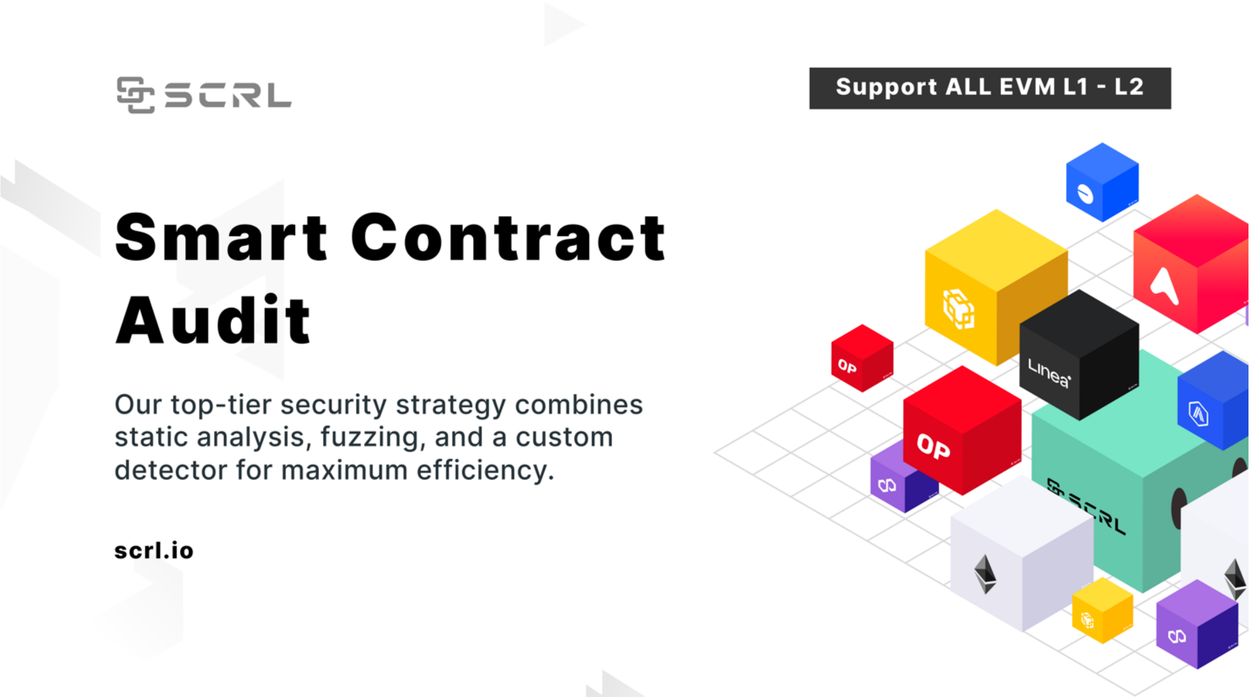
```
<<Interface>>
IUniswapV2Router02
src/NeverBackDown.sol

External:
   swapExactETHForTokensSupportingFeeOnTransferTokens(amountOutMin: uint, path: address[], to: address, deadline: uint)
   swapExactTokensForETHSupportingFeeOnTransferTokens(amountIn: uint, amountOutMin: uint, path: address[], to: address, deadline: uint)
```

```
NeverBackDown
src/NeverBackDown.sol

Private:
  _rOwned: mapping(address=>uint256)
  _tOwned: mapping(address=>uint256)
  _allowances: mapping(address=>mapping(address=>uint256))
  _isExcludedFromFee: mapping(address=>bool)
  _isExcluded: mapping(address=>bool)
  _excluded: address[]
  _fees: uint256[3]
  MAX: uint256
  _tTotal: uint256
  _rTotal: uint256
  _name: string
  _symbol: string
  _decimals: uint8
  minimumTokensBeforeSwap: uint256
  buyBackUpperLimit: uint256
Public:
  _liquidityFee: uint256
  _buybackFee: uint256
  _teamFee: uint256
  uniswapV2Router: IUniswapV2Router02
  uniswapV2Pair: address
  inSwapAndLiquify: bool
  swapAndLiquifyEnabled: bool
  buyBackEnabled: bool
  teamWalletAddress: address
  deadAddress: address
  _maxTxAmount: uint256
  minimumBalanceForBuyback: uint256

Private:
   _approve(owner: address, spender: address, amount: uint256)
   _transfer(from: address, to: address, amount: uint256)
   swapTokens(contractTokenBalance: uint256) <<lockTheSwap>>
   buyBackTokens(amount: uint256) <<lockTheSwap>>
   swapTokensForEth(tokenAmount: uint256)
   swapETHForTokens(amount: uint256)
   addLiquidity(tokenAmount: uint256, ethAmount: uint256)
   _tokenTransfer(sender: address, recipient: address, amount: uint256, _takeFee: bool)
   _transferStandard(sender: address, recipient: address, tAmount: uint256)
   _transferToExcluded(sender: address, recipient: address, tAmount: uint256)
   _transferFromExcluded(sender: address, recipient: address, tAmount: uint256)
   _transferBothExcluded(sender: address, recipient: address, tAmount: uint256)
   _getValues(tAmount: uint256): (uint256, uint256, uint256, uint256)
   _getTValues(tAmount: uint256): (uint256, uint256)
   _getRValues(tAmount: uint256, tLiquidity: uint256, currentRate: uint256): (uint256, uint256)
   _getRate(): uint256
   _getCurrentSupply(): (uint256, uint256)
   _takeAllFees(tLiquidity: uint256)
   calculateAllFees(_amount: uint256): uint256
Internal:
   removeAllFee()
   restoreAllFee()
   totalFee(): uint256
External:
   <<payable>> null()
   setMaxTxAmount(maxTxAmount: uint256) <<onlyOwner>>
   setMinimumTokensBeforeSwap(_minimumTokensBeforeSwap: uint256) <<onlyOwner>>
   setBuybackUpperLimit(buyBackLimit: uint256) <<onlyOwner>>
   setTeamWalletAddress(_teamWalletAddress: address) <<onlyOwner>>
Public:
   <<event>> BuyBackEnabledUpdated(enabled: bool)
   <<event>> SwapAndLiquifyEnabledUpdated(enabled: bool)
   <<event>> SwapETHForTokens(amountIn: uint256, path: address[])
   <<event>> SwapTokensForETH(amountIn: uint256, path: address[])
   <<modifier>> lockTheSwap()
   constructor()
   name(): string
   symbol(): string
   decimals(): uint8
   totalSupply(): uint256
   balanceOf(account: address): uint256
   transfer(recipient: address, amount: uint256): bool
   allowance(owner: address, spender: address): uint256
   approve(spender: address, amount: uint256): bool
   transferFrom(sender: address, recipient: address, amount: uint256): bool
   increaseAllowance(spender: address, addedValue: uint256): bool
   decreaseAllowance(spender: address, subtractedValue: uint256): bool
   isExcludedFromReward(account: address): bool
   minimumTokensBeforeSwapAmount(): uint256
   buyBackUpperLimitAmount(): uint256
   reflectionFromToken(tAmount: uint256, deductTransferFee: bool): uint256
   tokenFromReflection(rAmount: uint256): uint256
   setMinimumBalanceForBuyback(_amount: uint256) <<onlyOwner>>
   updateFees(liquidityFee: uint256, buybackFee: uint256, teamFee: uint256) <<onlyOwner>>
   isExcludedFromFee(account: address): bool
   excludeFromFee(account: address) <<onlyOwner>>
   includeInFee(account: address) <<onlyOwner>>
   setSwapAndLiquifyEnabled(_enabled: bool) <<onlyOwner>>
   setBuyBackEnabled(_enabled: bool) <<onlyOwner>>
```

```
<<Interface>>
IUniswapV2Pair
src/NeverBackDown.sol

External:
   factory(): address
```

About SCRL

SCRL (Previously name SECURI LAB) was established in 2020, and its goal is to deliver a security solution for Web3 projects by expert security researchers. To verify the security of smart contracts, they have developed internal tools and KYC solutions for Web3 projects using industry-standard technology. SCRL was created to solve security problems for Web3 projects. They focus on technology for conciseness in security auditing. They have developed Python-based tools for their internal use called WAS and SCRL. Their goal is to drive the crypto industry in Thailand to grow with security protection technology.



SCRL

Support ALL EVM L1 - L2

Smart Contract Audit

Our top-tier security strategy combines static analysis, fuzzing, and a custom detector for maximum efficiency.

scrl.io

Follow Us On:

Website	https://scrl.io/
Twitter	https://twitter.com/scrl_io
Telegram	https://t.me/scrl_io
Medium	https://scrl.medium.com/