SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Audit

Security Assessment

17. August, 2021

For

DYNAMIS

Disclaimer

SolidProof.io reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team. SolidProof.io do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc’...)

SolidProof.io Audits do not provide any warranty or guarantee regarding the absolute bug- free nature of the technology analyzed, nor do they provide any indication of the technology proprietors. SolidProof Audits should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

SolidProof.io Reports represent an extensive auditing 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. SolidProof’s position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of security or functionality of the technology we agree to analyze.

Network
Polygon (PoS Chain)

Website
https://dynamis.finance/

Telegram
https://t.me/Dynamis_Finance

Twitter
https://twitter.com/Dynamis_Finance

Github
https://github.com/Dynamis-Finance

Medium
https://medium.com/@DynamisFinance

Description

Dynamis Finance is a decentralized exchange running on the Polygon network, which in itself is a sidechain running on the Ethereum network. The platform provides crosscompatibility with other ecosystems.

The name Dynamis was chosen because it embodies three key principles - power, potential and ability. These principles underly our core vision for the platform – a place that enables our users to achieve financial freedom by exploring the full potential of decentralized finance.

On the Dynamis' platform, users can create liquidity pools and interact with smart contracts while earning rewards. After researching the demands and desires of our users, we have enhanced the platform with incentivized utilitarian functionality through several features that will be detailed in the following sections.

Project Engagement

During the 29th of July 2021, **DYNAToken Team** engaged Solidproof.io to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. **DYNAToken Team** provided Solidproof.io with access to their code repository and whitepaper.

Logo

DYNAMIS

Contract Link

https://polygonscan.com/address/0x15b74087e37d3168e25E127f02000D1A4aF2288f#code

Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 - 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon aspossible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
   i) Review of the specifications, sources, and instructions provided to SolidProof to make sure we understand the size, scope, and functionality of the smart contract.
   ii) Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii) Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to SolidProof describe.

2. Testing and automated analysis that includes the following:
   i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii) Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.

3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

Used Code from other Frameworks/Smart Contracts (direct imports)

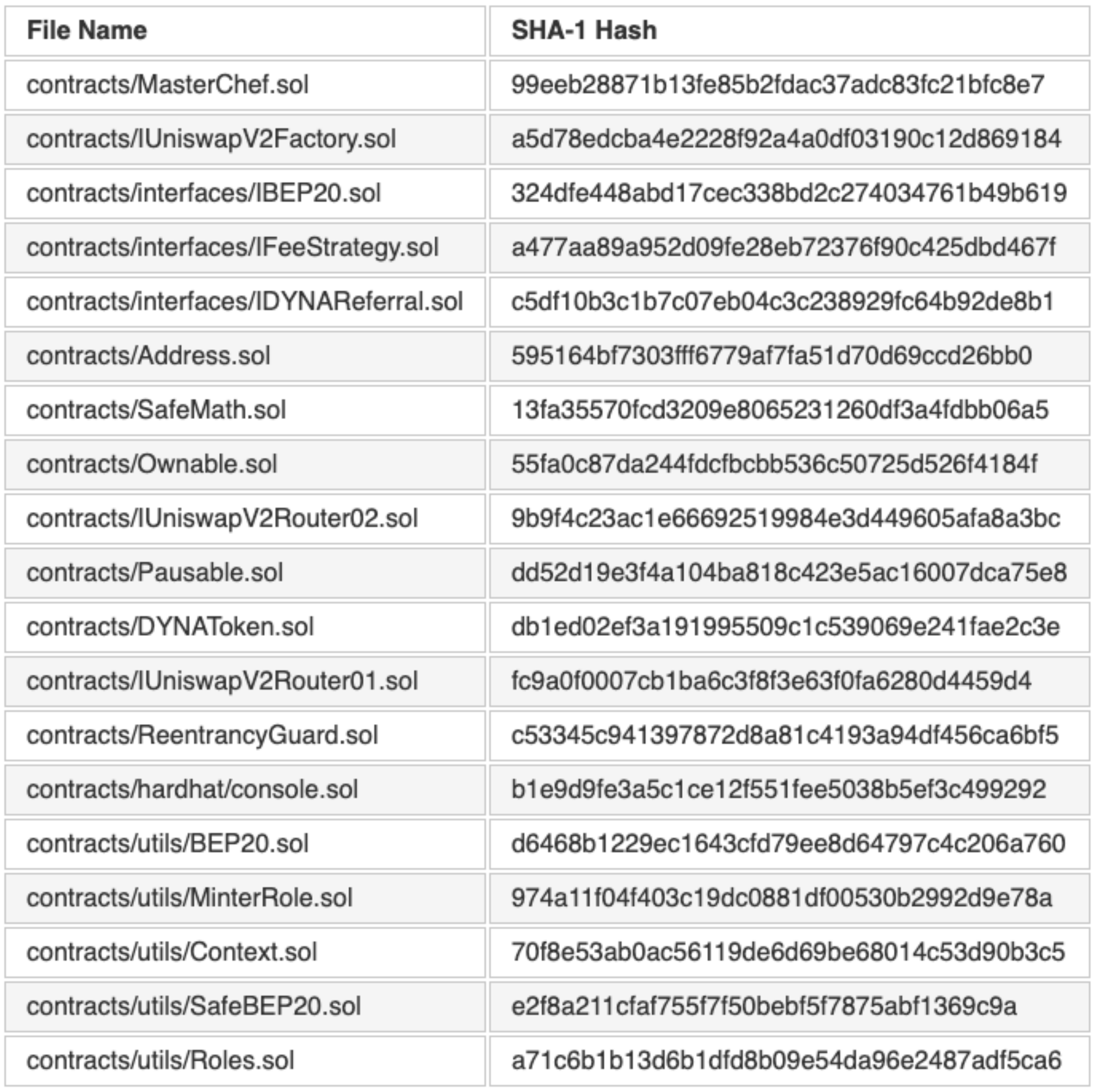
Imported packages:

- OpenZeppelin
  - Address
  - Ownable
  - SafeMatch
- Uniswap
  - UniswapV2Factory
  - UniswapV2Pair
  - UniswapV2Router01
  - UniswapV2Router02

Tested Contract Files

This audit covered the following files listed below with a SHA-1 Hash.

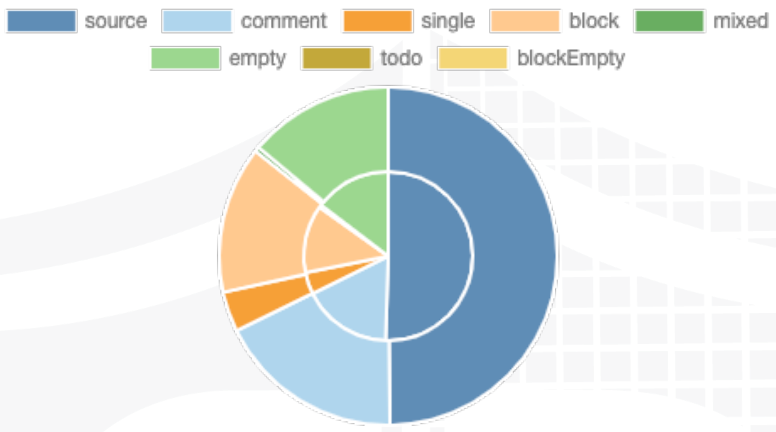
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.



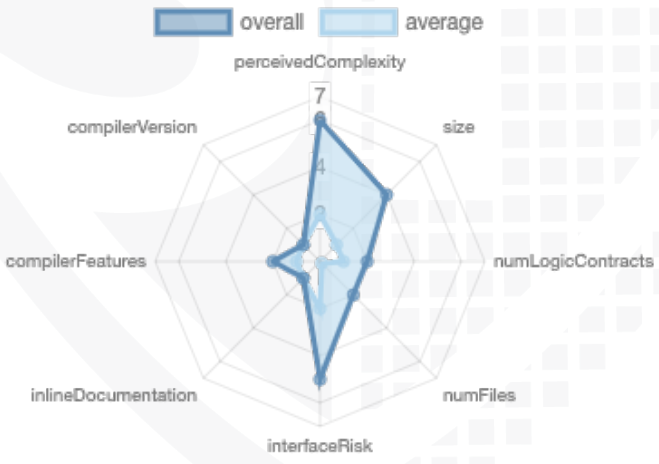
File Name	SHA-1 Hash
contracts/MasterChef.sol	99eeb28871b13fe85b2fdac37adc83fc21bfc8e7
contracts/IUniswapV2Factory.sol	a5d78edcba4e2228f92a4a0df03190c12d869184
contracts/interfaces/IBEP20.sol	324dfe448abd17cec338bd2c274034761b49b619
contracts/interfaces/IFeeStrategy.sol	a477aa89a952d09fe28eb72376f90c425dbd467f
contracts/interfaces/IDYNAReferral.sol	c5df10b3c1b7c07eb04c3c238929fc64b92de8b1
contracts/Address.sol	595164bf7303fff6779af7fa51d70d69ccd26bb0
contracts/SafeMath.sol	13fa35570fcd3209e8065231260df3a4fdbb06a5
contracts/Ownable.sol	55fa0c87da244fdcfbcbb536c50725d526f4184f
contracts/IUniswapV2Router02.sol	9b9f4c23ac1e66692519984e3d449605afa8a3bc
contracts/Pausable.sol	dd52d19e3f4a104ba818c423e5ac16007dca75e8
contracts/DYNAToken.sol	db1ed02ef3a191995509c1c539069e241fae2c3e
contracts/IUniswapV2Router01.sol	fc9a0f0007cb1ba6c3f8f3e63f0fa6280d4459d4
contracts/ReentrancyGuard.sol	c53345c941397872d8a81c4193a94df456ca6bf5
contracts/hardhat/console.sol	b1e9d9fe3a5c1ce12f551fee5038b5ef3c499292
contracts/utils/BEP20.sol	d6468b1229ec1643cfd79ee8d64797c4c206a760
contracts/utils/MinterRole.sol	974a11f04f403c19dc0881df00530b2992d9e78a
contracts/utils/Context.sol	70f8e53ab0ac56119de6d69be68014c53d90b3c5
contracts/utils/SafeBEP20.sol	e2f8a211cfaf755f7f50bebf5f7875abf1369c9a
contracts/utils/Roles.sol	a71c6b1b13d6b1dfd8b09e54da96e2487adf5ca6

Metrics

Source Lines



Risk Level



Capabilities

Components

Contracts	Libraries	Interfaces	Abstract
4	5	6	4

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
102	4

External	Internal	Private	Pure	View
64	526	6	21	421

State Variables

Total	Public
53	35

Capabilities

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
`^0.8.0` `>=0.5.0` `>=0.4.0` `>=0.6.12` `>=0.6.2` `>=0.8.0` `>=0.4.22` `<0.9.0` `>=0.7.0` `>=0.6.0`		`yes`	`yes` (4 asm blocks)	

Transfers ETH	Low-Level Calls	Delegate Call	Uses Hash Functions	ECRecover	New/ Create/ Create2
`yes`		`yes`	`yes`		

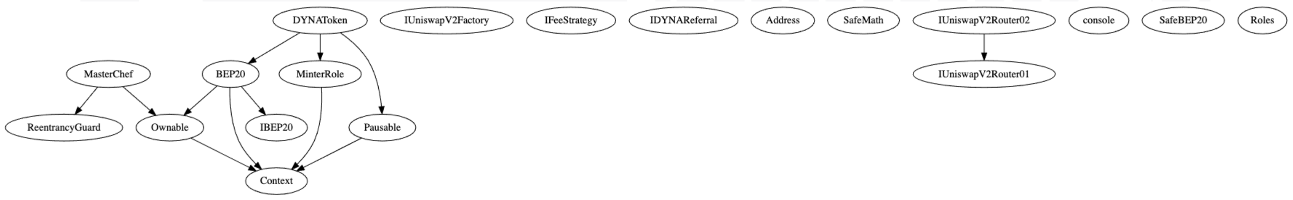
Scope of Work

The above token Team provided us with the files that needs to be tested (Github, Bscscan, Etherscan, files, etc.). The scope of the audit is the main contract (usual the same name as team appended with .sol).

We will verify the following claims:

1. Correct implementation of Token standard
2. Deployer cannot mint any new tokens
3. Deployer cannot burn or lock user funds
4. Deployer cannot pause the contract
5. Overall checkup (Smart Contract Security)

Inheritance Graph



Verify Claims

Correct implementation of Token standard

Tested	Verified
✓	✓

Function	Description	Exist	Tested	Verified
TotalSupply	provides information about the total token supply	✓	✓	✓
BalanceOf	provides account balance of the owner's account	✓	✓	✓
Transfer	executes transfers of a specified number of tokens to a specified address	✓	✓	✓
TransferFrom	executes transfers of a specified number of tokens from a specified address	✓	✓	✓
Approve	allow a spender to withdraw a set number of tokens from a specified account	✓	✓	✓
Allowance	returns a set number of tokens from a spender to the owner	✓	✓	✓

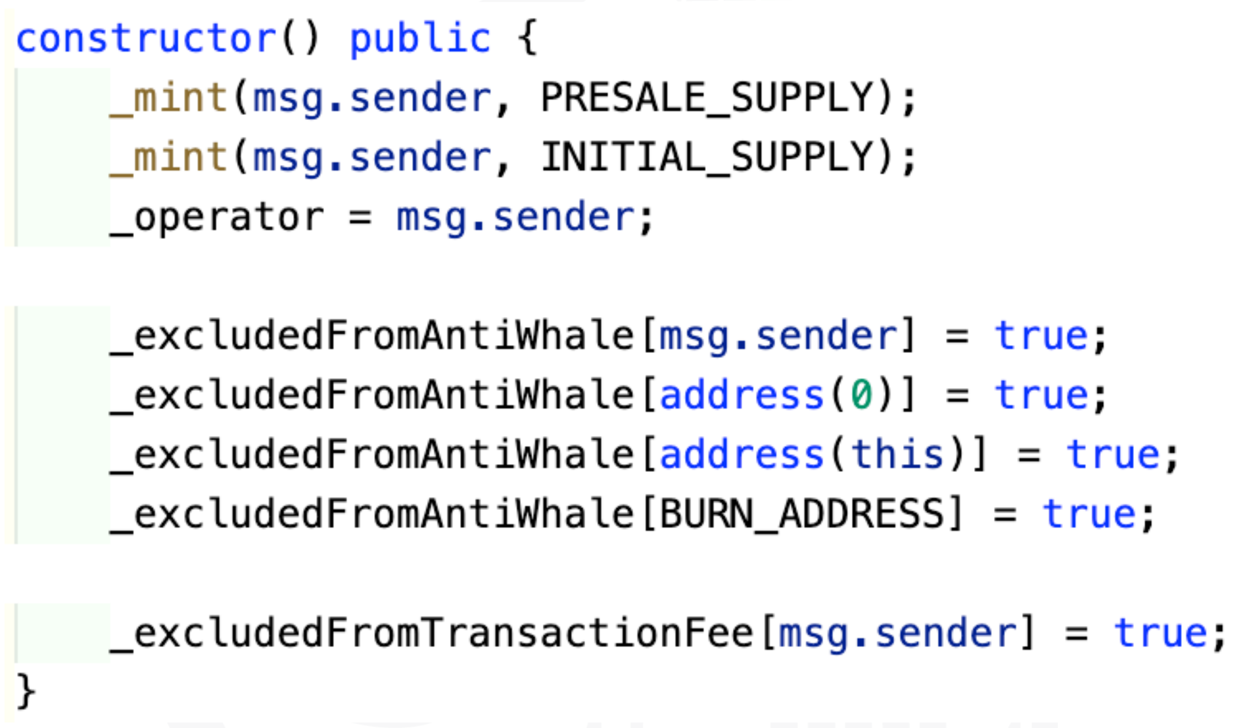
Optional implementations

Function	Description	Exist	Tested	Verified
renounceOwnership	Owner renounce ownership for more trust	✓	✓	✓

Deployer cannot mint any new tokens

Tested	Deployer cannot mint	File	Comment
✓	✗	Main	Line: 118

Max / Total Supply: 10.184.000

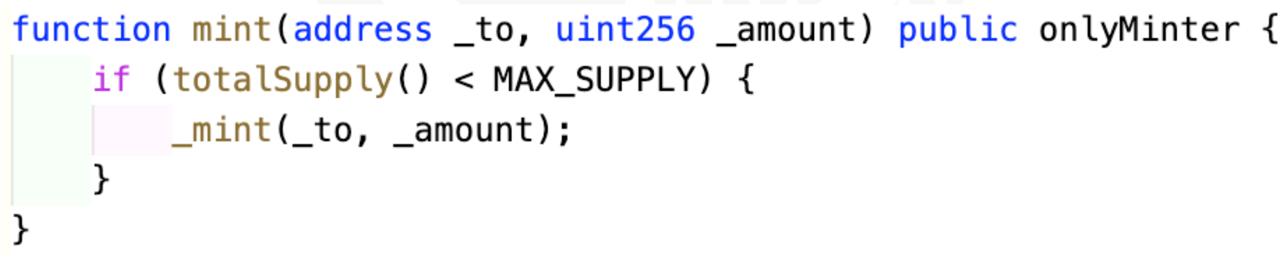


```
constructor() public {
        _mint(msg.sender, PRESALE_SUPPLY);
        _mint(msg.sender, INITIAL_SUPPLY);
        _operator = msg.sender;

        _excludedFromAntiWhale[msg.sender] = true;
        _excludedFromAntiWhale[address(0)] = true;
        _excludedFromAntiWhale[address(this)] = true;
        _excludedFromAntiWhale[BURN_ADDRESS] = true;

        _excludedFromTransactionFee[msg.sender] = true;
}
```

DYNAToken.sol



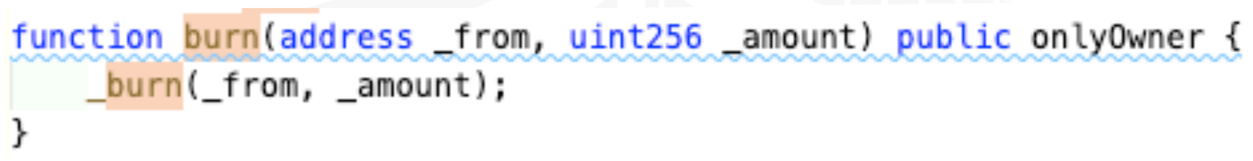
```
function mint(address _to, uint256 _amount) public onlyMinter {
    if (totalSupply() < MAX_SUPPLY) {
        _mint(_to, _amount);
    }
}
```

Deployer cannot burn or lock user funds

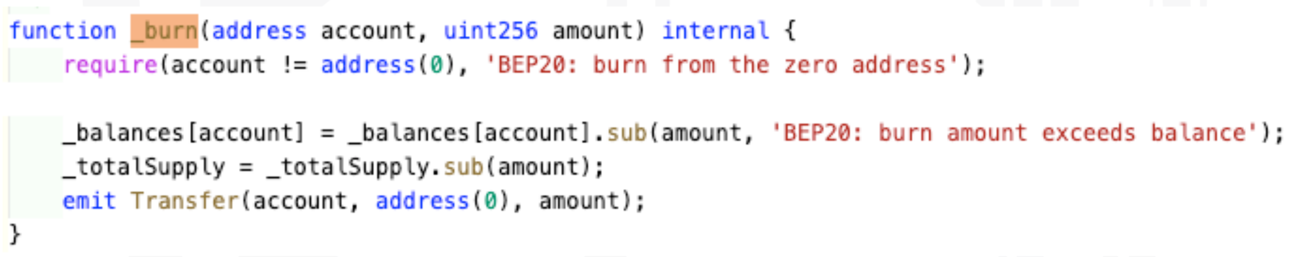
Name	Tested	Exist	Verified
No Lock function	✓	✓	✓
No Burn function	✓	✓	✗

DYNAToken.so

- Burn function can only called by the owner (l)



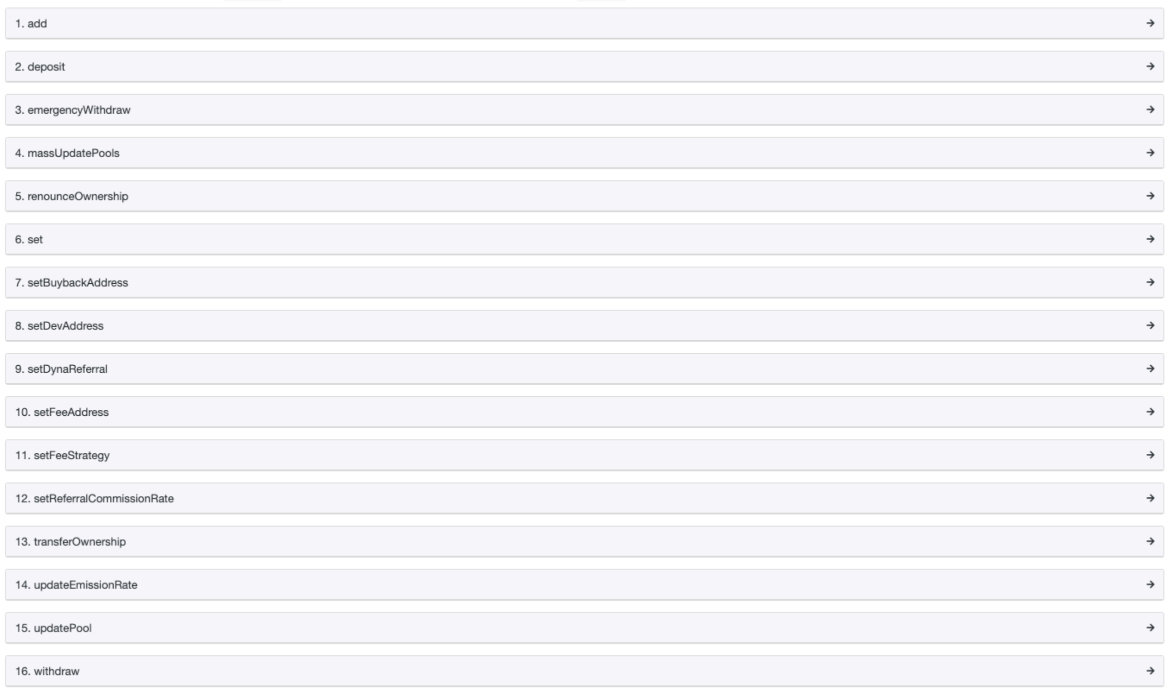
```
function burn(address _from, uint256 _amount) public onlyOwner {
    _burn(_from, _amount);
}
```



```
function _burn(address account, uint256 amount) internal {
    require(account != address(0), 'BEP20: burn from the zero address');

    _balances[account] = _balances[account].sub(amount, 'BEP20: burn amount exceeds balance');
    _totalSupply = _totalSupply.sub(amount);
    emit Transfer(account, address(0), amount);
}
```

MasterChef.sol



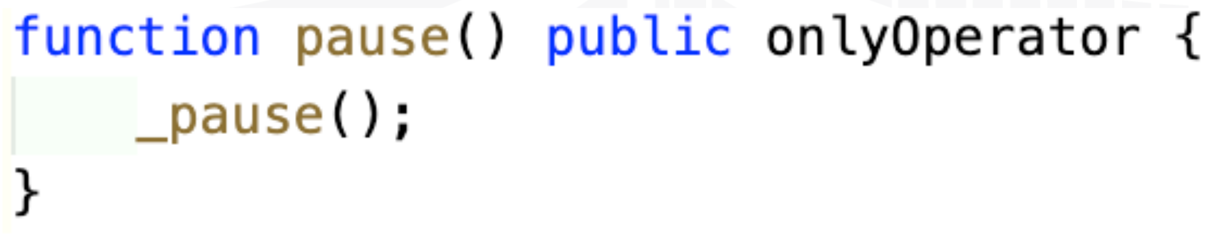
1. add
2. deposit
3. emergencyWithdraw
4. massUpdatePools
5. renounceOwnership
6. set
7. setBuybackAddress
8. setDevAddress
9. setDynaReferral
10. setFeeAddress
11. setFeeStrategy
12. setReferralCommissionRate
13. transferOwnership
14. updateEmissionRate
15. updatePool
16. withdraw

Deployer cannot pause the contract

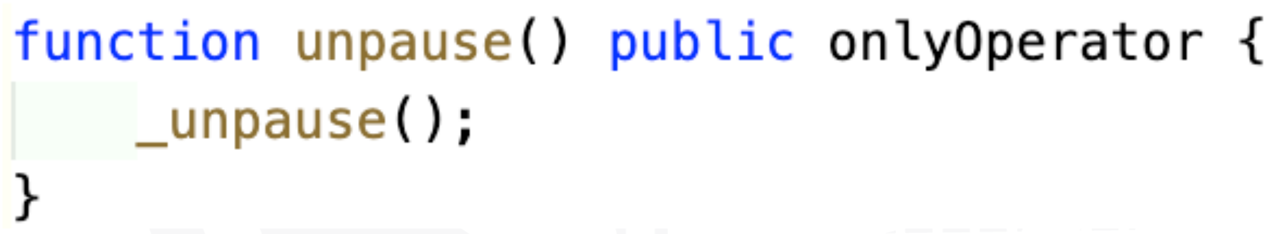
Tested	Deployer cannot pause	Exist
✓	✗	✓

DYNAToken.sol

- Operator is the creator of contract



```
function pause() public onlyOperator {
        _pause();
}
```



```
function unpause() public onlyOperator {
        _unpause();
}
```

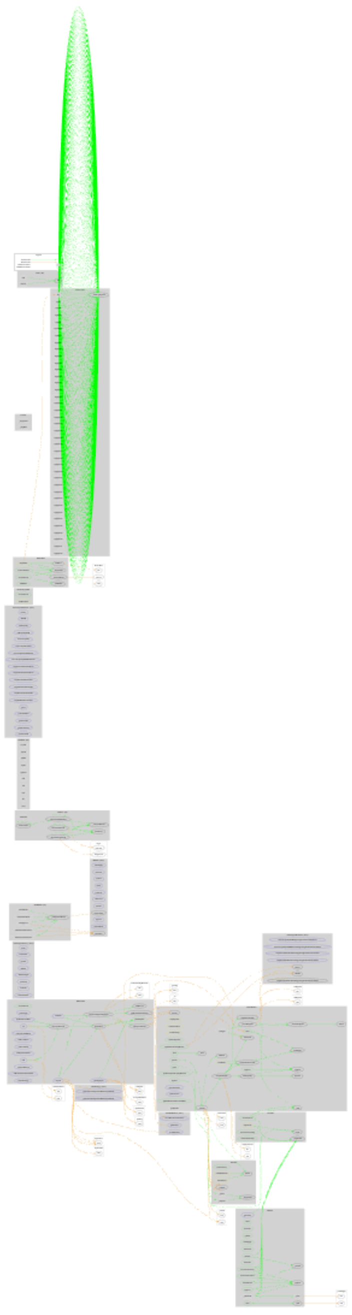
Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

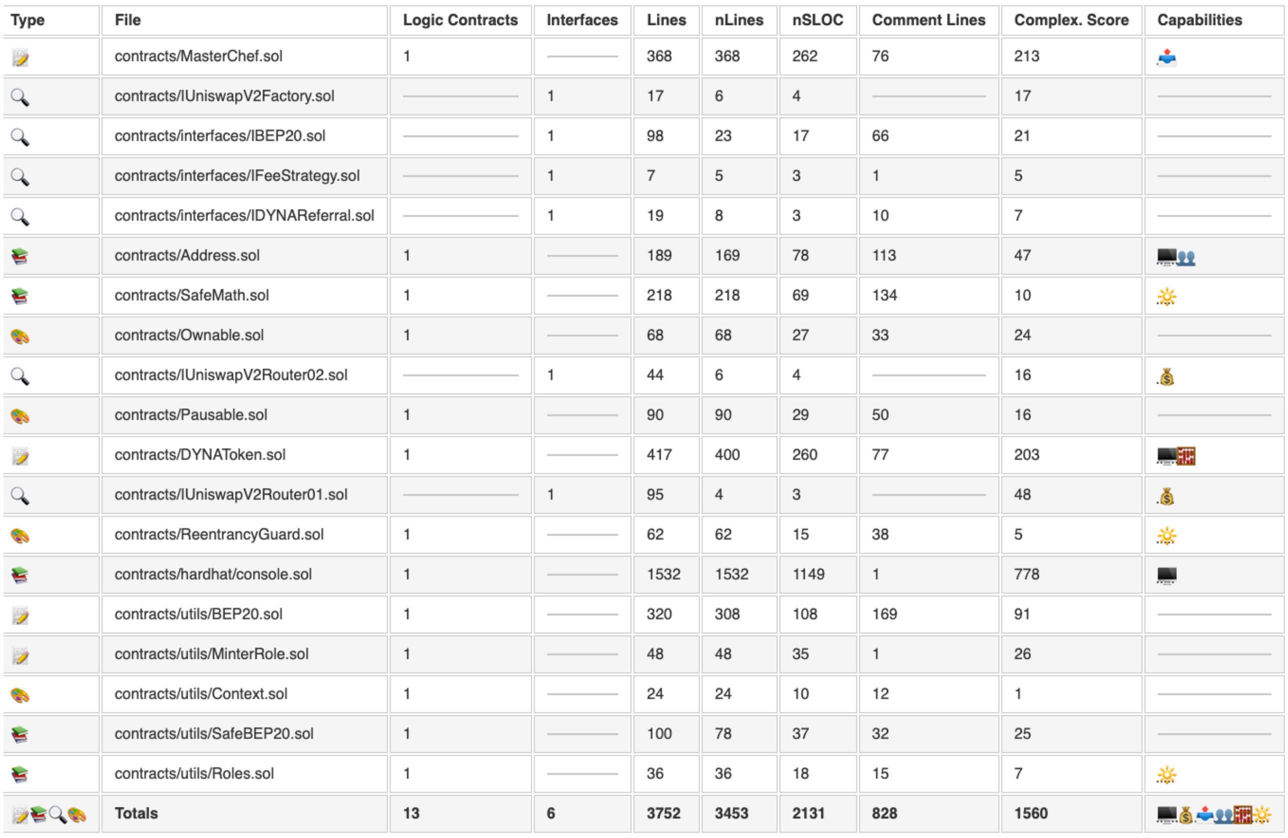
Legend

Attribute	Symbol
Verfified / Checked	✓
Partly Verified	⚑
Unverified / Not checked	✗

CallGraph



Source Units in Scope



Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝	contracts/MasterChef.sol	1	----	368	368	262	76	213	📥
🔍	contracts/IUniswapV2Factory.sol	----	1	17	6	4	----	17	----
🔍	contracts/interfaces/IBEP20.sol	----	1	98	23	17	66	21	----
🔍	contracts/interfaces/IFeeStrategy.sol	----	1	7	5	3	1	5	----
🔍	contracts/interfaces/IDYNAReferral.sol	----	1	19	8	3	10	7	----
📚	contracts/Address.sol	1	----	189	169	78	113	47	🖥👥
📚	contracts/SafeMath.sol	1	----	218	218	69	134	10	🔆
🎨	contracts/Ownable.sol	1	----	68	68	27	33	24	----
🔍	contracts/IUniswapV2Router02.sol	----	1	44	6	4	----	16	💰
🎨	contracts/Pausable.sol	1	----	90	90	29	50	16	----
📝	contracts/DYNAToken.sol	1	----	417	400	260	77	203	🖥🧮
🔍	contracts/IUniswapV2Router01.sol	----	1	95	4	3	----	48	💰
🎨	contracts/ReentrancyGuard.sol	1	----	62	62	15	38	5	🔆
📚	contracts/hardhat/console.sol	1	----	1532	1532	1149	1	778	🖥
📝	contracts/utils/BEP20.sol	1	----	320	308	108	169	91	----
📝	contracts/utils/MinterRole.sol	1	----	48	48	35	1	26	----
🎨	contracts/utils/Context.sol	1	----	24	24	10	12	1	----
📚	contracts/utils/SafeBEP20.sol	1	----	100	78	37	32	25	----
📚	contracts/utils/Roles.sol	1	----	36	36	18	15	7	🔆
📝📚🔍🎨	**Totals**	**13**	**6**	**3752**	**3453**	**2131**	**828**	**1560**	🖥💰📥👥🧮🔆

Legend

Attribute	Description
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, ...)

Audit Results

AUDIT PASSED

Critical issues

- no critical issues found -

High issues

- no high issues found -

Medium issues

- no medium issues found -

Low issues

Issue	File	Type	Line	Description
#1	Main	A floating pragma is set	2	The current pragma Solidity directive is "">=0.8.0"".

Informational issues

- no informational issues found -

Commented Code exist

There are some instances of code being commented out in the following files that should be removed:

File	Line	Comment
MasterChef.sol	203	`// dyna.mint(devAddress, dynaReward.div(10));`

	220-223	`// if (address(pool.lpToken) == address(dyna)) {`
		`    //     uint256 transferTax =`
		`_amount.mul(dyna.transferTaxRate()).div(10000);`
		`    //     _amount = _amount.sub(transferTax);`
		`    // }`

Recommendation

Remove the commented code, or address them properly.

Audit Comments

31. July 2021:

DYNAToken.sol

- Owner can mint tokens lower than set MAX_SUPPLY variable
- Owner may indefinitely pause the contract. When the contract is paused you are not able to transfer any token
- Owner can enable disable swap and liquify variable
- Owner has not renounced ownership

MasterChef.sol

- Owner has not renounced ownership
- Owner can set referral commission rate to 0, so the payment of the referral commission to the referrer who referred that user is thus prevented.

SWC Attacks

ID	Title	Relationships	Status
SWC-136	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SWC-135	Code With No Effects	CWE-1164: Irrelevant Code	PASSED
SWC-134	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SWC-133	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED
SWC-132	Unexpected Ether balance	CWE-667: Improper Locking	PASSED
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	PASSED
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	PASSED
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	PASSED
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	PASSED

SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	**PASSED**
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	**PASSED**
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	**PASSED**
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	**PASSED**
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	**PASSED**
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	**PASSED**
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	**PASSED**
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	**PASSED**
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	**PASSED**
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	**PASSED**

SW C-11 6	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SW C-11 5	Authorization through tx.origin	CWE-477: Use of Obsolete Function	PASSED
SW C-11 4	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	PASSED
SW C-11 3	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	PASSED
SW C-11 2	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SW C-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	PASSED
SW C-11 0	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	PASSED
SW C-10 9	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	PASSED
SW C-10 8	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	PASSED
SW C-10 7	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED
SW C-10 6	Unprotected SELFDESTRUC T Instruction	CWE-284: Improper Access Control	PASSED

SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	**PASSED**
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	**PASSED**
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	**NOT PASSED**
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	**PASSED**
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	**PASSED**
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	**PASSED**

Solid Proofed

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY