SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

Peernetics

Audit

Security Assessment
10. February, 2023

For

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.

Version	Date	Description
1.0	06. February 2023	• Layout project • Automated- /Manual-Security Testing • Summary
1.1	10. February 2023	• Reaudit

Network
Polygon

Website
https://www.peernetics.io/

Telegram
@peernetics

Twitter
https://twitter.com/peernetics

Facebook
https://m.facebook.com/Peernetics

Instagram
https://www.instagram.com/peernetics/

LinkedIn
https://www.linkedin.com/company/peernetics/

Description

A global crypto payment gateway made easy and accessible to everyone. Accept Bitcoin, Ethereum, Peernectics token and other cryptocurrencies, gain new customers globally, and avoid the cost of high fees and chargebacks. From start to finish, Peernetics makes accepting these payments easy.

Project Engagement

During the 3rd of February 2023, **Peernetics 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. They provided Solidproof.io with access to their code repository and whitepaper.

Logo

Contract Link

v1.0

- Github
    - https://github.com/Peernetics-Ltd/token-smart-contract
    - **Commit**
        - c844788106b806a2fca85406d3f2e584ce903436

v1.1

- Github
    - https://github.com/Peernetics-Ltd/token-smart-contract
    - **Commit**
        - 94b6e0b2de1fc5164130a753646e2457ecf998d8

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.

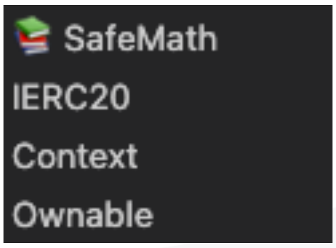
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

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

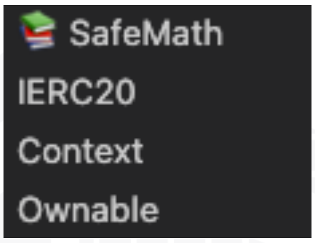
Used Code from other Frameworks/Smart Contracts (direct imports)

Imported packages:

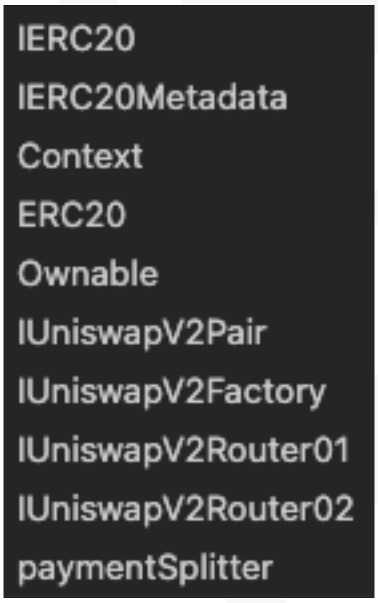
MarketingVesting


SafeMath
IERC20
Context
Ownable

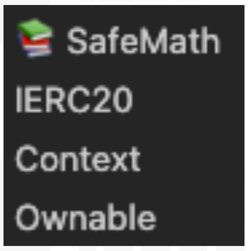
ResearchAndDevelopme
ntVersting


SafeMath
IERC20
Context
Ownable

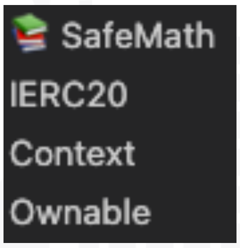
Peernetics


IERC20
IERC20Metadata
Context
ERC20
Ownable
IUniswapV2Pair
IUniswapV2Factory
IUniswapV2Router01
IUniswapV2Router02
paymentSplitter

TeamVesting


SafeMath
IERC20
Context
Ownable

UpdatedICOVesting


SafeMath
IERC20
Context
Ownable

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.

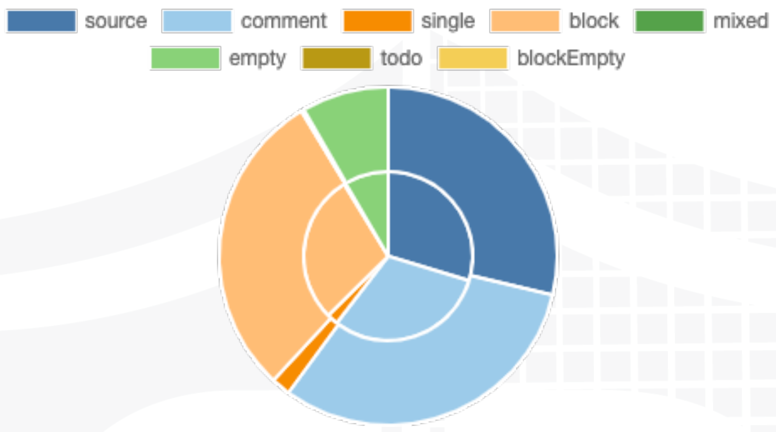
v1.1

File Name	SHA-1 Hash
contracts/MarketingVesting.sol	05308b2be9b3083078b026214bbf231c6f00b332
contracts/researchAndDevelopment.sol	1574fb2231ba6d42486764185dffb44ecd238ee0
contracts/ICOclaim.sol	fb0a4369fae78b61b3f1f74ef002c9aa3c4961ee
contracts/TeamVesting.sol	dcf53ca52159bb237ccdd7e9ba9e2b16b80cb09a
contracts/Peernetics.sol	133b8a66726f2cfae479bf765f806d35a6157fc1

Metrics

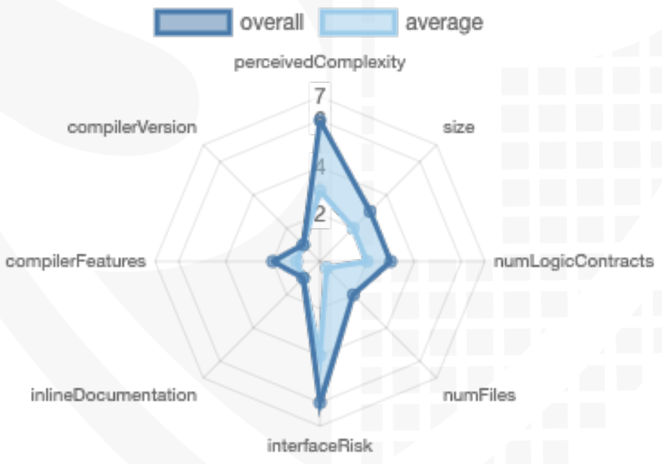
Source Lines

v1.0



Risk Level

v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	7	4	10	10

Exposed Functions

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

Version	Public	Payable
1.0	151	6

Version	External	Internal	Private	Pure	View
1.0	103	190	2	62	63

State Variables

Version	Total	Public
1.0	49	16

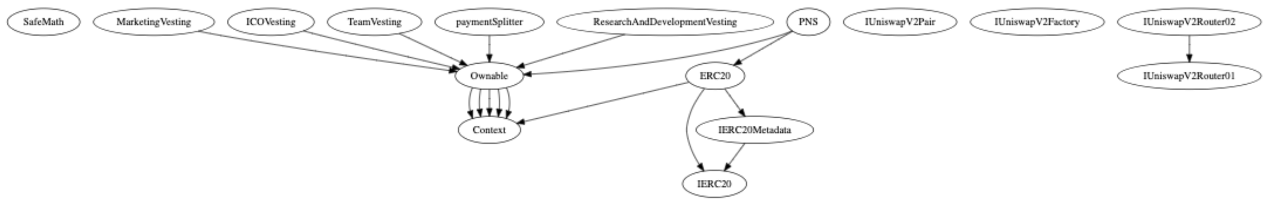
Capabilities

Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	`0.8.17`		`yes`		

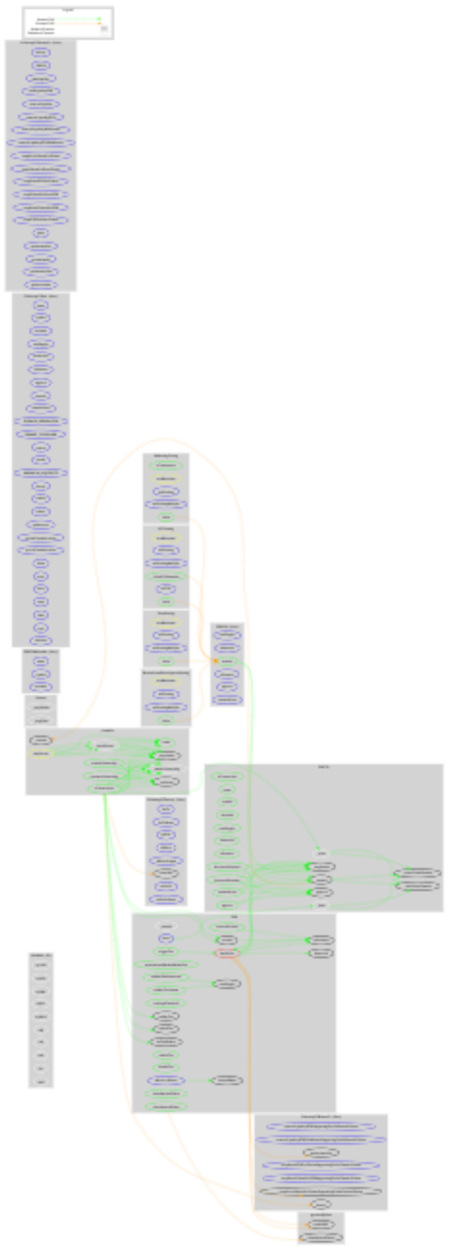
Version	Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2

1.0	`yes`					`yes` → `NewContract:paymentSplitter`

Inheritance Graph
v1.0



CallGraph
v1.0



Scope of Work/Verify Claims

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).

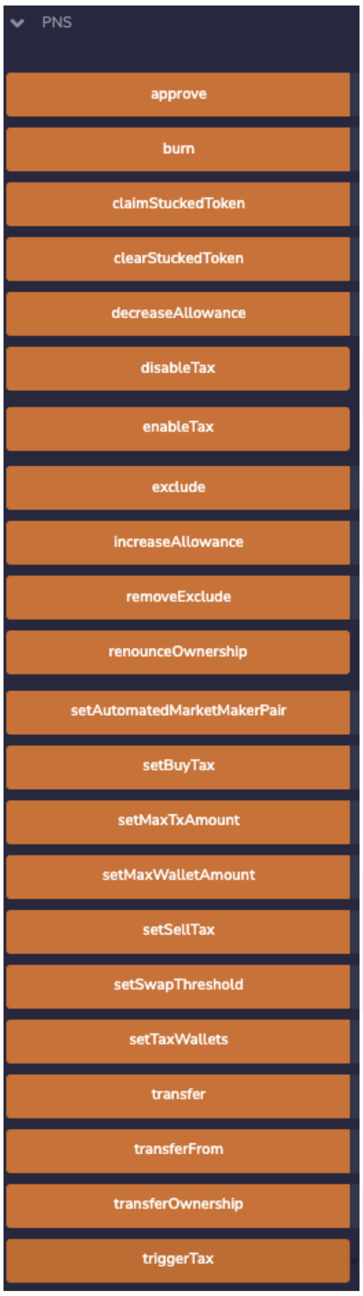
We will verify the following claims:

1. Is contract an upgradeable
2. Deployer cannot mint any new tokens
3. Deployer cannot burn or lock user funds
4. Deployer cannot pause the contract
5. Deployer cannot set fees
6. Deployer cannot blacklist/antisnipe addresses
7. Overall checkup (Smart Contract Security)

Is contract an upgradeable

Name	
Is contract an upgradeable?	No

Write functions of contract v1.0



PNS

approve

burn

claimStuckedToken

clearStuckedToken

decreaseAllowance

disableTax

enableTax

exclude

increaseAllowance

removeExclude

renounceOwnership

setAutomatedMarketMakerPair

setBuyTax

setMaxTxAmount

setMaxWalletAmount

setSellTax

setSwapThreshold

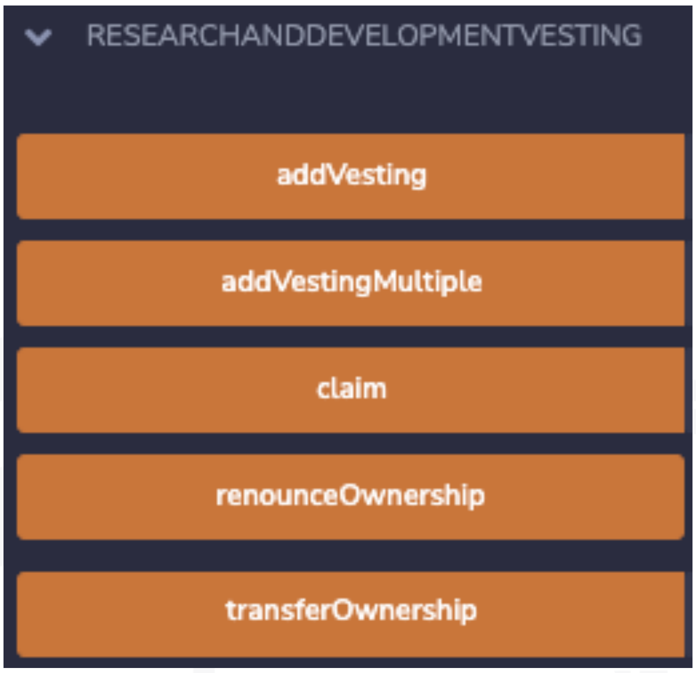
setTaxWallets

transfer

transferFrom

transferOwnership

triggerTax



RESEARCHANDDEVELOPMENTVESTING

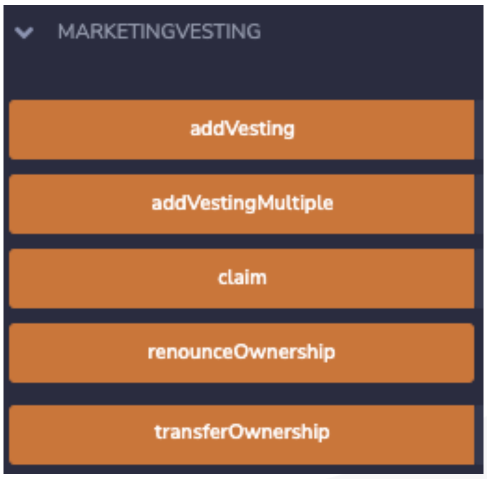
addVesting

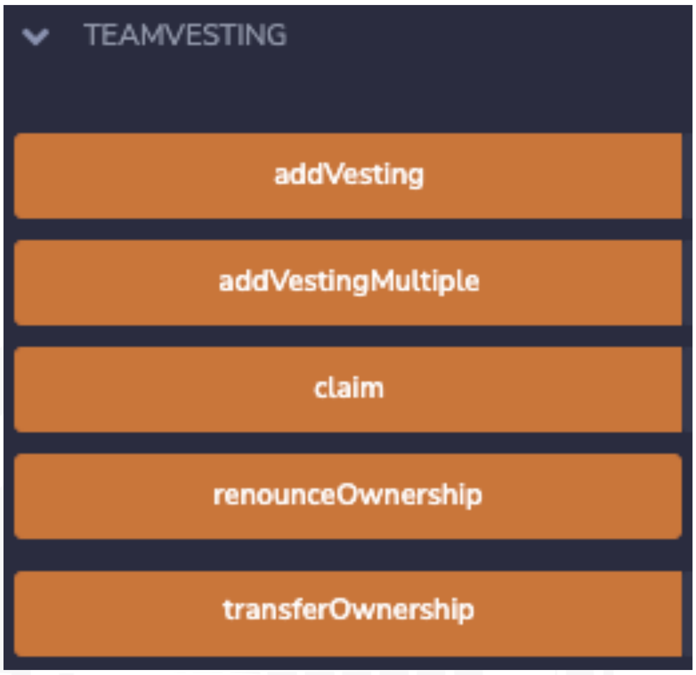
addVestingMultiple

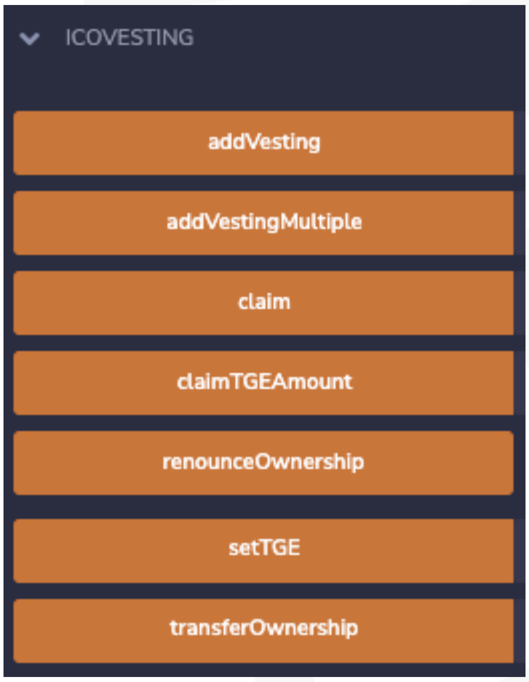
claim

renounceOwnership

transferOwnership


MARKETINGVESTING
addVesting
addVestingMultiple
claim
renounceOwnership
transferOwnership


TEAMVESTING
addVesting
addVestingMultiple
claim
renounceOwnership
transferOwnership


ICOVESTING
addVesting
addVestingMultiple
claim
claimTGEAmount
renounceOwnership
setTGE
transferOwnership

Deployer cannot mint any new tokens

Name	Exist	Tested	Status
Deployer cannot mint	✓	✓	✓
Max / Total Supply			1000000000

Deployer cannot burn or lock user funds

Name	Exist	Tested	Status
Deployer can lock	✓	✓	✓
Deployer cannot burn	✓	✓	✓

Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer cannot pause	–	–	–

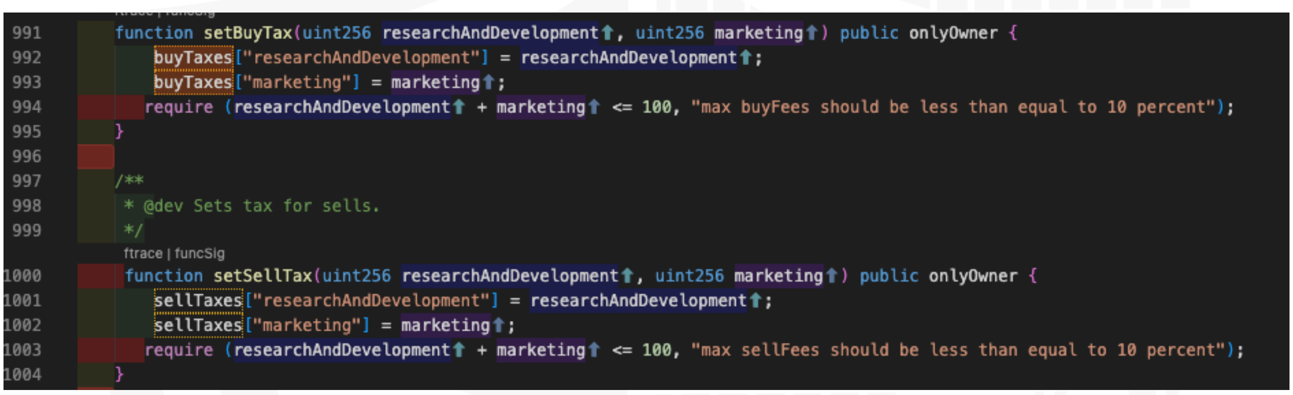
Deployer cannot set fees

Name	Exist	Tested	Status
Deployer cannot set fees over 25%	✓	✓	✓
Deployer cannot set fees to nearly 100% or to 100%	✓	✓	✓

Comments:

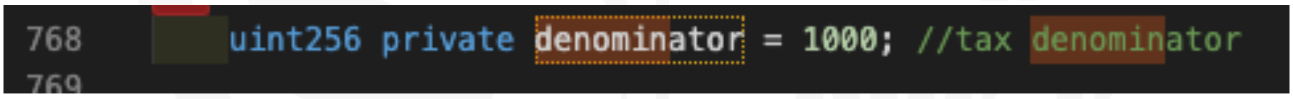
v1.0

- Fees can be set max to 10%



```
function setBuyTax(uint256 researchAndDevelopment, uint256 marketing) public onlyOwner {
    buyTaxes["researchAndDevelopment"] = researchAndDevelopment;
    buyTaxes["marketing"] = marketing;
   require (researchAndDevelopment + marketing <= 100, "max buyFees should be less than equal to 10 percent");
}

/**
 * @dev Sets tax for sells.
 */
ftrace | funcSig
 function setSellTax(uint256 researchAndDevelopment, uint256 marketing) public onlyOwner {
    sellTaxes["researchAndDevelopment"] = researchAndDevelopment;
    sellTaxes["marketing"] = marketing;
   require (researchAndDevelopment + marketing <= 100, "max sellFees should be less than equal to 10 percent");
}
```



```
uint256 private denominator = 1000; //tax denominator
```

Deployer can blacklist/antisnipe addresses

Name	Exist	Tested	Status
Deployer cannot blacklist/antisnipe addresses	–	–	–

Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

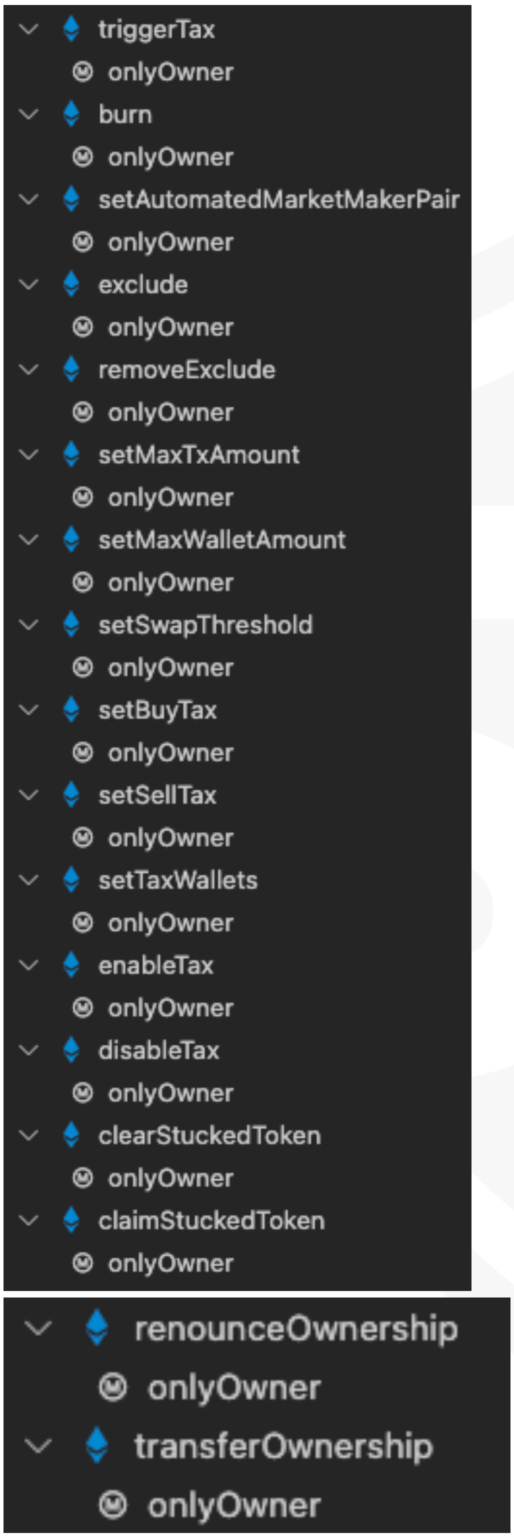
Legend

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

Modifiers and public functions

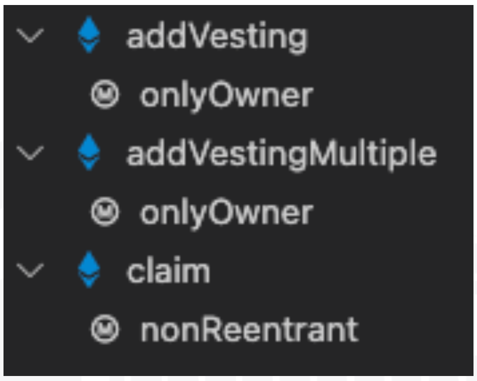
v1.0

Peernetics



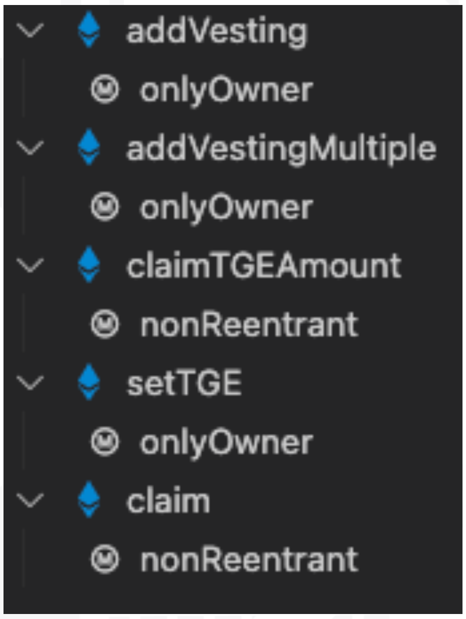
```
triggerTax
  onlyOwner
burn
  onlyOwner
setAutomatedMarketMakerPair
  onlyOwner
exclude
  onlyOwner
removeExclude
  onlyOwner
setMaxTxAmount
  onlyOwner
setMaxWalletAmount
  onlyOwner
setSwapThreshold
  onlyOwner
setBuyTax
  onlyOwner
setSellTax
  onlyOwner
setTaxWallets
  onlyOwner
enableTax
  onlyOwner
disableTax
  onlyOwner
clearStuckedToken
  onlyOwner
claimStuckedToken
  onlyOwner
renounceOwnership
  onlyOwner
transferOwnership
  onlyOwner
```

MarketingVesting/ResearchAndDevelopmentVesting/TeamVesting



```
addVesting
  onlyOwner
addVestingMultiple
  onlyOwner
claim
  nonReentrant
```

UpdatedICOVesting



```
addVesting
  onlyOwner
addVestingMultiple
  onlyOwner
claimTGEAmount
  nonReentrant
setTGE
  onlyOwner
claim
  nonReentrant
```

Comments

- *Deployer can set following state variables without any limitations*
  - Peernetics
    - sellTaxes

        - Only max to 10%
    - buyTaxes
        - Only max to 10%
    - swapThreshold
    - maxWalletAmount
        - Min 1% of total supply
    - maxTxAmount
        - Min 1% of total supply

- *Deployer can enable/disable following state variables*
    - Peernetics
        - taxStatus
        - excludeList
        - automatedMarketMakerPairs

- *Deployer can set following addresses*
    - Peernetics
        - taxWallets

- *Existing Modifiers*
    - Peernetics
        - onlyOwner

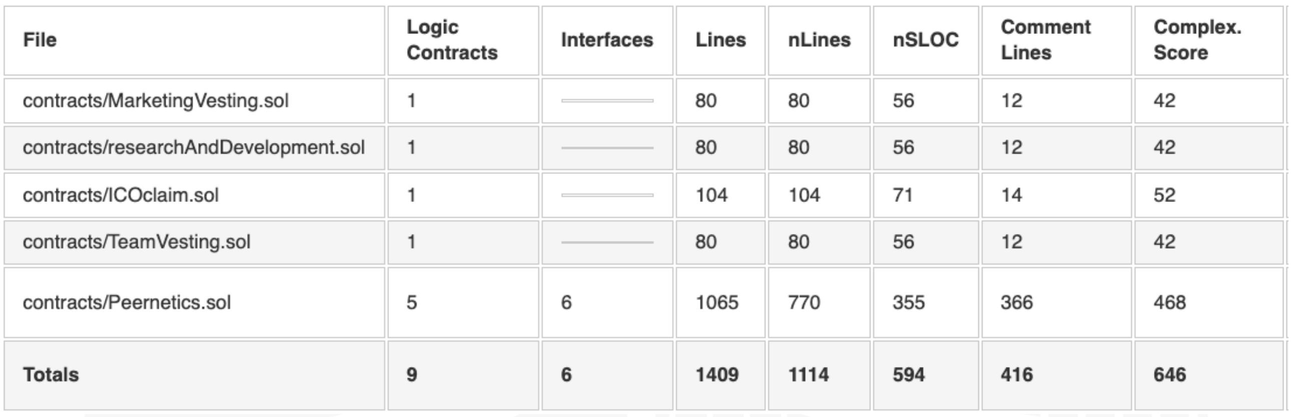
- Peernetics
    - Owner is able to

        - Also the owner is able to call the “clearStuckedToken” function to clear the stucked token from the payment handler to an arbitrary address
    - Accidentally matic send to contract will be automatically transferred to the marketing wallet. If the marketing wallet is set to 0 address the funds will be lost.
- All Vestings
    - Only owner is able to
        - Addvesting
        - addVestingMultiple

Please check if an OnlyOwner or similar restrictive modifier has been forgotten.

Source Units in Scope

v1.1



File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/MarketingVesting.sol	1		80	80	56	12	42
contracts/researchAndDevelopment.sol	1		80	80	56	12	42
contracts/ICOclaim.sol	1		104	104	71	14	52
contracts/TeamVesting.sol	1		80	80	56	12	42
contracts/Peernetics.sol	5	6	1065	770	355	366	468
Totals	**9**	**6**	**1409**	**1114**	**594**	**416**	**646**

Legend

Attribute	Description
Lines	total lines of the source unit
nLines	normalised lines of the source unit (e.g. normalises functions spanning multiple lines)
nSLOC	normalised 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

Critical issues

No critical issues

High issues

No high issues

Medium issues

No medium issues

Low issues

No low issues

Informational issues

Issue	File	Type	Line	Description
#1	Main	NatSpec documentation missing	-	If you started to comment your code, also comment all other functions, variables etc.

Audit Comments

We recommend you to use the special form of comments (NatSpec Format, Follow link for more information https://docs.soliditylang.org/en/latest/natspec-format.html) for your contracts to provide rich documentation for functions, return variables and more. This helps investors to make clear what that variables, functions etc. do.

10. February 2023:

- There is still an owner (Owner still has not renounced ownership)
- Users of the marketing, research and development, and Team vesting contracts will only be able to claim the vested tokens once every 3 months.
- The owner will decide the monthly unlock percent at the time of adding the vesting and if that amount is set to zero then the users will not be able to claim any tokens.
- Read whole report and modifiers section for more information

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

SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	**PASSED**
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	**PASSED**
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	**PASSED**
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	**PASSED**
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	**PASSED**
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	**PASSED**
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	**PASSED**
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	**PASSED**
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	**PASSED**
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	**PASSED**
SWC-106	Unprotected SELFDESTRUCT 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	**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**

SolidProof_io @solidproof_io

Solid Proofed

Blockchain Security | Smart Contract Audits | KYC
Development | Marketing

MADE IN GERMANY