Shopping.io

Staking & Exchange

SMART CONTRACT AUDIT

18.09.2022

Made in Germany by Chainsulting.de

Table of contents

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of S.E.T. SHOPPING TECHNOLOGY ENTERPRISES LTD. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (06.07.2022)	Layout
0.4 (08.07.2022)	Automated Security Testing Manual Security Testing
0.5 (12.07.2022)	Verify Claims and Test Deployment
0.6 (13.07.2022)	Testing SWC Checks
0.9 (15.07.2022)	Summary and Recommendation
1.0 (15.07.2022)	Final document
1.1 (23.08.2022)	Re-check
1.2 (18.09.2022)	Deployed contracts

2. About the Project and Company

Company address:
S.E.T. SHOPPING TECHNOLOGY ENTERPRISES LTD
Registration number: HE 426697
Panorama, 35, Flat Oikia 3A
Koyklia, 8500, Pafos
Cyprus

Website: https://shopping.io

Instagram: https://www.instagram.com/shopping.io_official/

Twitter: https://twitter.com/shopping_io

Discord: https://discord.gg/36xNXa6

Telegram: https://t.me/shoppingio

Facebook: https://www.facebook.com/shopping.io

2.1 Project Overview

Shopping.io was established as of December 2020. They are founded by dropshipping veterans with a vision to change how we make purchases with crypto. Shopping.io allows users to purchase goods directly from some of the leading ecommerce giants using over 100 different cryptocurrencies. To avail of this facility, all one has to do is sign up on Shopping.io by entering their email address and setting up the desired password. Once the account is created, they get access to a personal dashboard from which they can start searching and ordering stuff from the likes of Amazon, Walmart, eBay and more. The entire process from scratch takes no longer than a few minutes. Apart from the convenience of making payment in cryptocurrencies, customers on Shopping.io also stand to enjoy additional discounts and freedom from miscellaneous charges like shipping charges, sales tax, and more.

Shopping.io is in the constant process of developing and introducing new features for the cryptocurrency community. Some of the upcoming developments include the launch of its own token that offers an additional 5% discount and international shipping service to token holders. The platform will also be introducing new membership tiers after the conclusion of the limited period 10% discount offer on Free accounts. While users will still be able to purchase goods with cryptocurrencies using a free account, the discounts will be limited to Starter and Pro accounts at 5% and 10% respectively. All products shopped on Shopping.io, irrespective of account type will be eligible for 2-day free delivery within the United States along with a return/refund policy applicable for a maximum of 30 days from the date of delivery. Shopping.io will also be expanding its support for other leading ecommerce platforms including the upcoming AliExpress integration.

3. 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 as possible.
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

4. 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.

4.1 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 Chainsulting 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 Chainsulting 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.

5. Metrics

The metrics section should give the reader an overview on the size, quality, flows and capabilities of the codebase, without the knowledge to understand the actual code.

5.1 Tested Contract Files

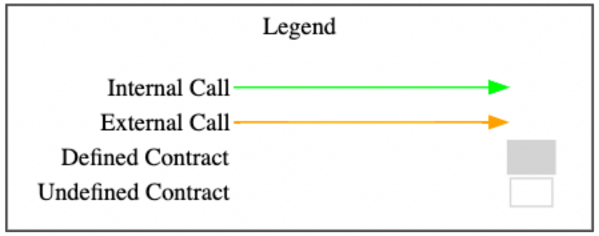
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

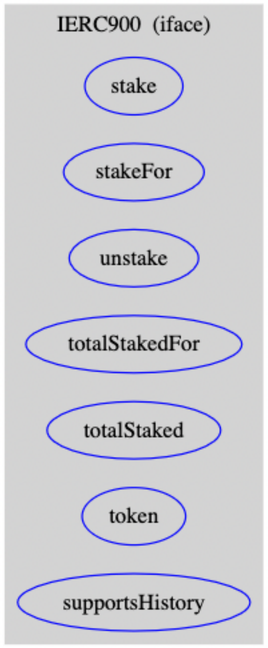
File	Fingerprint (MD5)
./SpiExt.sol	0ed51a5ff11bcb6c49e4b02976cd13ef
./NftStaking.sol	787a4e169967a9c3940f22b024b212dc
./ShovpverseNft.sol	0ff791a78095d8a94c75934b15bb7f25
./SpiExtStaking.sol	a39e760dd29a7ed87928d49bc99465f4
./IER900.sol	8c2fd552cbbaad6bee85e1adfcf65d26
./Exchange.sol	27bff1080491d75092f031031c9c4dea

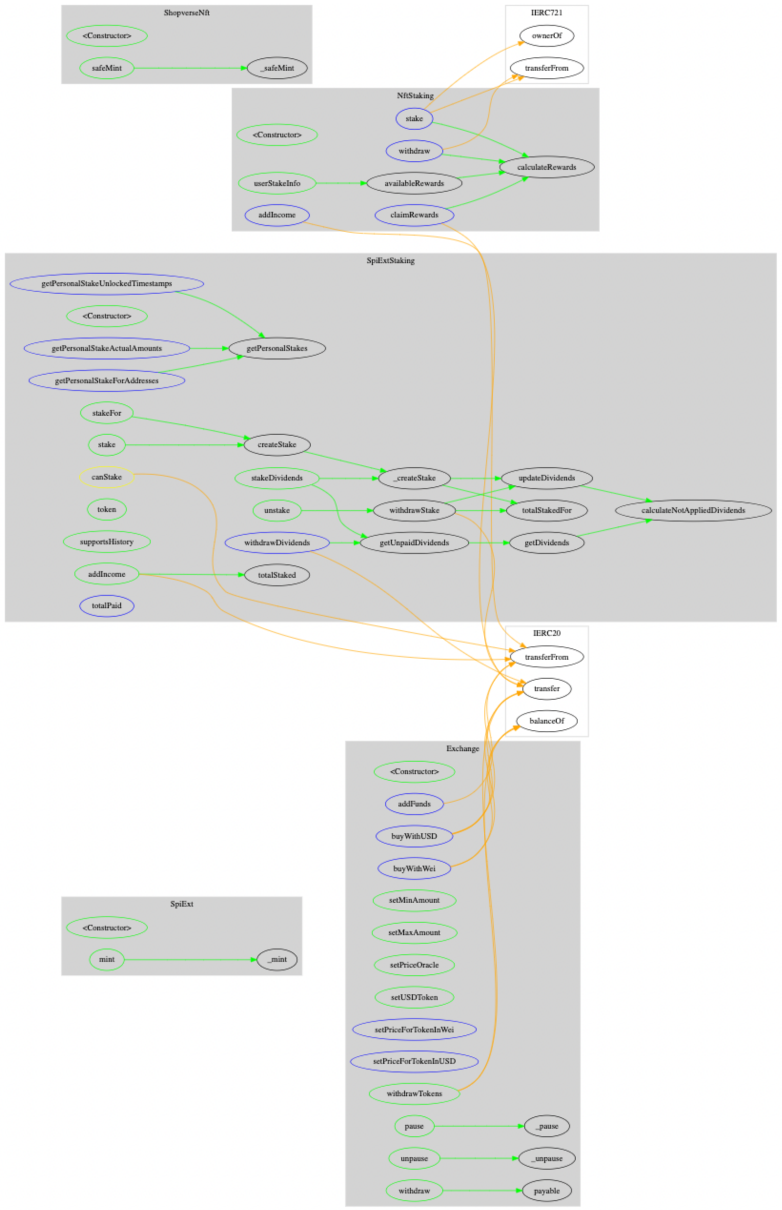
5.2 Used Code from other Frameworks/Smart Contracts (direct imports)

Dependency / Import Path	Source
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.6.0/contracts/access/Ownable.sol
@openzeppelin/contracts/security/Pausable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.6.0/contracts/security/Pausable.sol
@openzeppelin/contracts/security/ReentrancyGuard.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.6.0/contracts/security/ReentrancyGuard.sol
@openzeppelin/contracts/token/ERC1155/IERC1155.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.6.0/contracts/token/ERC1155/IERC1155.sol
@openzeppelin/contracts/token/ERC20/ERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.6.0/contracts/token/ERC20/ERC20.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.6.0/contracts/token/ERC20/IERC20.sol
@openzeppelin/contracts/token/ERC721/ERC721.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.6.0/contracts/token/ERC721/ERC721.sol
@openzeppelin/contracts/token/ERC721/IERC721.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.6.0/contracts/token/ERC721/IERC721.sol
@openzeppelin/contracts/utils/math/SafeMath.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.6.0/contracts/utils/math/SafeMath.sol

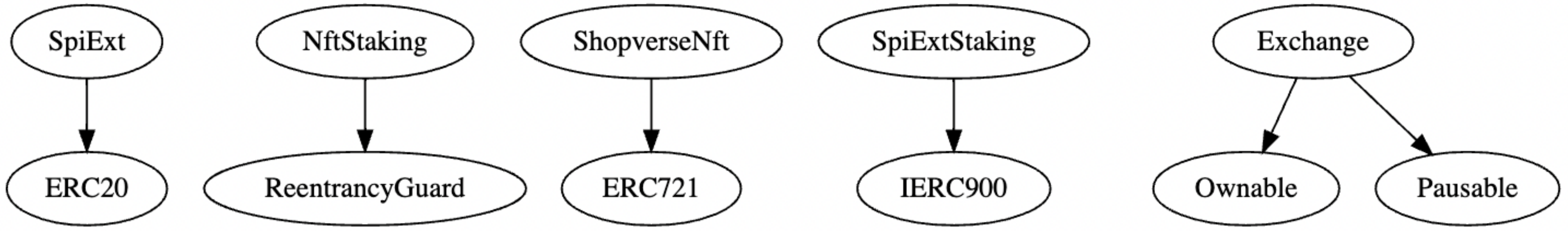
5.3 CallGraph



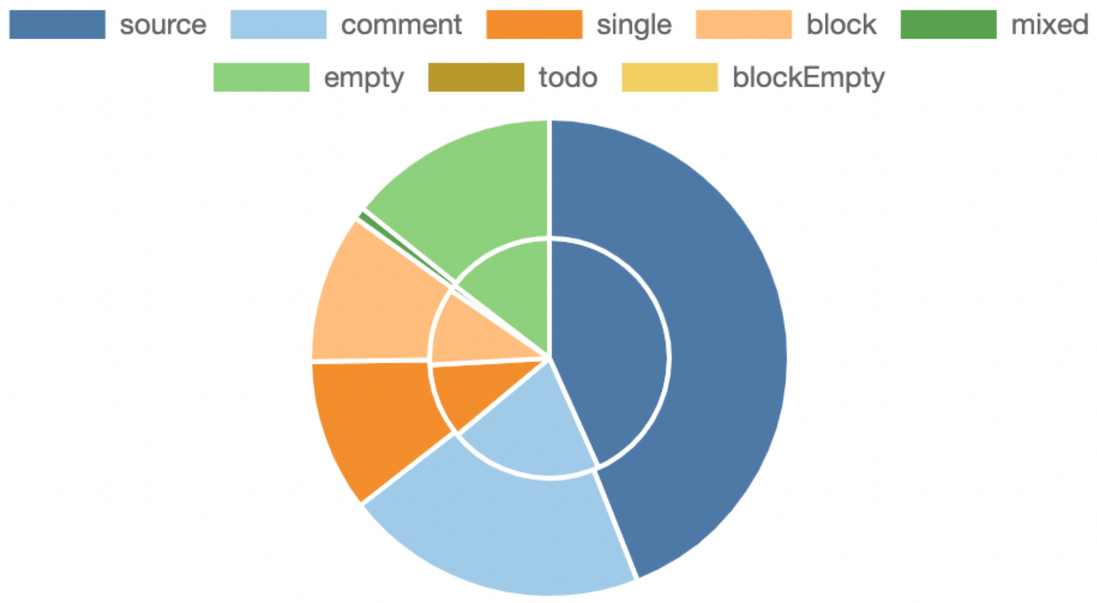


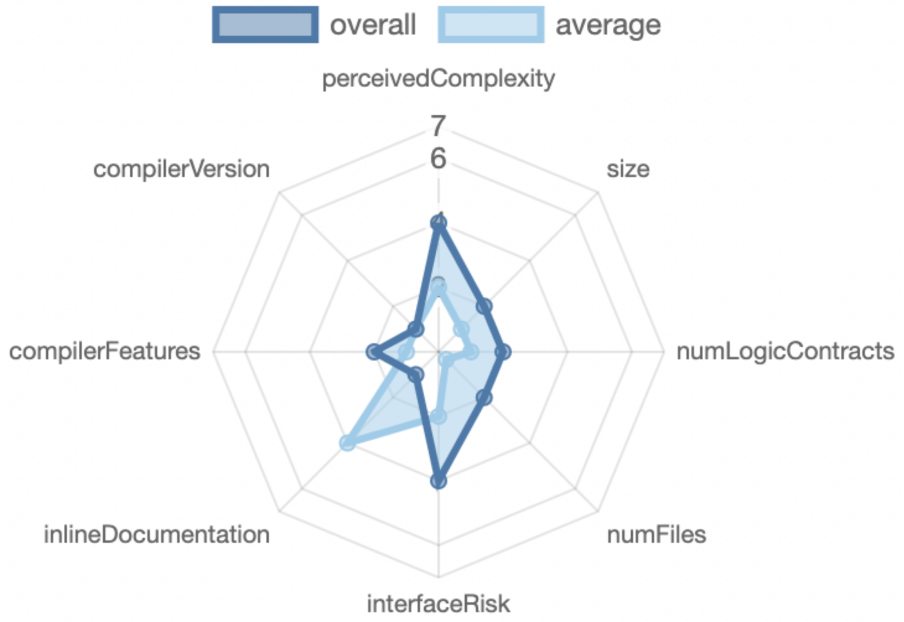


5.4 Inheritance Graph



5.5 Source Lines & Risk





5.6 Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`^0.8.0`		`yes`		

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

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
43	1

External	Internal	Private	Pure	View
21	40	3	2	

StateVariables

Total	🌐Public
19	18

5.7 Source Unites in Scope

Source: https://github.com/dchizhevskiy/shop-staking-truffle
Last commit: 510e8eb14cb43ec891d938c29d2b94977ae333c2

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝	contracts/SpiExt.sol	1		16	16	9	3	7	
📝	contracts/NftStaking.sol	1		171	162	103	33	62	📤
📝	contracts/ShovpverseNft.sol	1		15	15	9	3	7	
📝	contracts/SpiExtStaking.sol	1		338	336	164	110	133	📤
🔍	contracts/IERC900.sol		1	22	10	5	5	15	
📝	contracts/Exchange.sol	1		127	127	81	27	83	💰📤
📝🔍	**Totals**	**5**	**1**	**689**	**666**	**371**	**181**	**307**	

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, ...)

6. Scope of Work

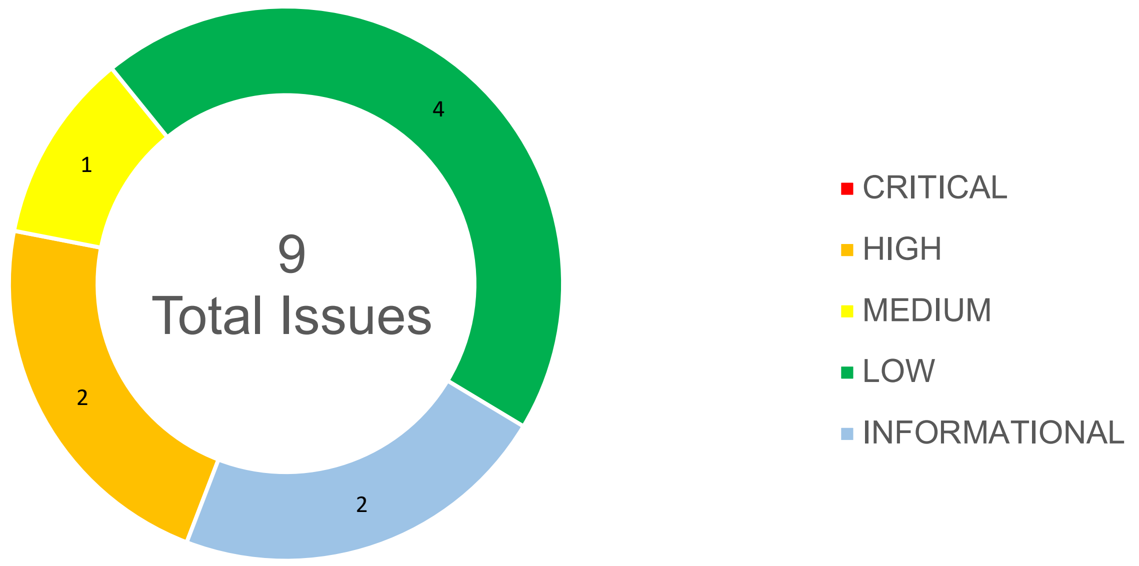
The Shopping.io Team provided us with the files that needs to be tested. The scope of the audit are the staking contracts.

The team put forward the following assumptions regarding the security, usage of the contracts:

- Staking is working as expected and the owner is not able to drain out user funds
- Rewards/Dividends are correctly calculated and possible to withdraw
- The smart contract is coded according to the newest standards and in a secure way.

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.

6.1 Findings Overview



No	Title	Severity	Status
6.2.1	Price Conflict	HIGH	FIXED
6.2.2	Price Change Can Lead To Free Tokens	HIGH	FIXED
6.2.3	Missing Transfer Verification	MEDIUM	FIXED
6.2.4	Using .transfer() to transfer Ether	LOW	FIXED
6.2.5	Missing Value Verification	LOW	FIXED
6.2.6	Zero Address Checks Missing	LOW	FIXED
6.2.7	The Owner Can Renounce Ownership	LOW	ACKNOWLEDGED
6.2.8	Floating Pragma Version Identified	INFORMATIONAL	ACKNOWLEDGED
6.2.9	Spelling Or Grammatical Errors	INFORMATIONAL	OPEN

6.2 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, Chainsulting‘s experts found **0 Critical issues** in the code of the smart contract.

HIGH ISSUES

During the audit, Chainsulting’s experts found **2 High issues** in the code of the smart contract.

6.2.1 Price Conflict

Severity: HIGH
Status: FIXED
Code: NA
File(s) affected: Exchange.sol
Update: 5f7abca1de39c6e87b4184a0e6fe98c8cadcd2d8, e6cc3c2bfa2866ae947c02fbe7df4e3020b7459b

Attack / Description	The users can buy tokens using the buyWithUSD or the buyWithWei function, the first function require 1 USD to buy 1 token, meanwhile the second function requires only 1 Wei to buy 1 token. This conflict between the prices makes the buyWithUSD function biased.

Code	Line 37 - 73 (Exchange.sol)
	
	`function buyWithWei(uint256 _amountTokens) external payable whenNotPaused {`
	`    require(_amountTokens >= minAmount, "below min");`
	`    require(_amountTokens <= maxAmount, "above max");`
	
	`    //you cannot buy more than the contract has`
	`    uint256 balToken = theToken.balanceOf(address(this));`
	`    require(_amountTokens <= balToken, "contract doesn't have enough balance");`

```
        //how much would it pay for them
        uint256 requiredAmount = (priceForTokenInWei * _amountTokens) / (10**18);
        require(requiredAmount == msg.value, "you must send the exact ETH amount");

        //finally give the user the tokens
        theToken.transfer(msg.sender, _amountTokens);
    }

//buy tokens for USD. _amountTokens is with decimals (must give allowance for USD Token)
    function buyWithUSD(uint256 _amountTokens) external whenNotPaused {
        require(_amountTokens >= minAmount, "below min");
        require(_amountTokens <= maxAmount, "above max");

        //you cannot buy more than the contract has
        uint256 balToken = theToken.balanceOf(address(this));
        require(_amountTokens <= balToken, "contract doesn't have enough balance");

        //how much would it pay for them
        uint256 usdRequiredAmount = (priceForTokenInUSD * _amountTokens) / (10**18);

        //transfer the usdtoken
        require(
            usdToken.transferFrom(msg.sender, address(this), usdRequiredAmount),
            "failed to transfer"
        );

        //finally give the user the tokens
        theToken.transfer(msg.sender, _amountTokens);
    }
```

Result/Recommendation	Consider adjusting the priceForTokenInWei and the priceForTokenInUSD variables to give the token the same value in both buying options.

6.2.2 Price Change Can Lead To Free Tokens

Severity: HIGH
Status: FIXED
Code: NA
File(s) affected: Exchange.sol
Update: 5f7abca1de39c6e87b4184a0e6fe98c8cadcd2d8, e6cc3c2bfa2866ae947c02fbe7df4e3020b7459b

Attack / Description	The users can buy tokens using the buyWithUSD or the buyWithWei function. The required amount to pay is in general (price * amountTokens) / (10**18). In the case where the amountTokens variable is lower than 10**18/ price, therefore, if the price is changed to a value that is lower than 10**18, the required amount will be equal to 0, due to the type conversion. Also, if the price is equal to 0 the users will be able to get any amount of tokens for free. Thus, in these cases, the users will be able to get specific amounts of tokens for free.

Code	Line 37 - 73 (Exchange.sol)
	
	`function buyWithWei(uint256 _amountTokens) external payable whenNotPaused {`
	`    require(_amountTokens >= minAmount, "below min");`
	`    require(_amountTokens <= maxAmount, "above max");`
	
	`    //you cannot buy more than the contract has`
	`    uint256 balToken = theToken.balanceOf(address(this));`
	`    require(_amountTokens <= balToken, "contract doesn't have enough balance");`
	
	`    //how much would it pay for them`
	`    uint256 requiredAmount = (priceForTokenInWei * _amountTokens) / (10**18);`
	`    require(requiredAmount == msg.value, "you must send the exact ETH amount");`
	
	`    //finally give the user the tokens`
	`    theToken.transfer(msg.sender, _amountTokens);`

```
    }

//buy tokens for USD. _amountTokens is with decimals (must give allowance for USD Token)
    function buyWithUSD(uint256 _amountTokens) external whenNotPaused {
        require(_amountTokens >= minAmount, "below min");
        require(_amountTokens <= maxAmount, "above max");

        //you cannot buy more than the contract has
        uint256 balToken = theToken.balanceOf(address(this));
        require(_amountTokens <= balToken, "contract doesn't have enough balance");

        //how much would it pay for them
        uint256 usdRequiredAmount = (priceForTokenInUSD * _amountTokens) / (10**18);

        //transfer the usdtoken
        require(
            usdToken.transferFrom(msg.sender, address(this), usdRequiredAmount),
            "failed to transfer"
        );

        //finally give the user the tokens
        theToken.transfer(msg.sender, _amountTokens);
    }
```

Result/Recommendation	We recommend preventing the priceForTokenInWei and the priceForTokenInUSD variables from having values that are lower than 10**18 using a require statement.

MEDIUM ISSUES

During the audit, Chainsulting's experts found **1 Medium issue** in the code of the smart contract.

6.2.3 Missing Transfer Verification

Severity: MEDIUM
Status: FIXED
Code: NA
File(s) affected: Exchange.sol, NftStaking.sol
Update: e6cc3c2bfa2866ae947c02fbe7df4e3020b7459b

Attack / Description	The ERC20 standard token implementation functions return the transaction status as a Boolean. It is best practice to check for the return status of the function call, to ensure that the transaction was executed successfully. It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller's transaction also fails.

Code	Line 37 - 51 (Exchange.sol)
	
	`function buyWithWei(uint256 _amountTokens) external payable whenNotPaused {`
	`    require(_amountTokens >= minAmount, "below min");`
	`    require(_amountTokens <= maxAmount, "above max");`
	
	`    //you cannot buy more than the contract has`
	`    uint256 balToken = theToken.balanceOf(address(this));`
	`    require(_amountTokens <= balToken, "contract doesn't have enough balance");`
	
	`    //how much would it pay for them`
	`    uint256 requiredAmount = (priceForTokenInWei * _amountTokens) / (10**18);`
	`    require(requiredAmount == msg.value, "you must send the exact ETH amount");`
	
	`    //finally give the user the tokens`
	`    theToken.transfer(msg.sender, _amountTokens);`
	`}`
	
	Line 54 - 73 (Exchange.sol)

	```
	//buy tokens for USD. _amountTokens is with decimals (must give allowance for USD Token)
	    function buyWithUSD(uint256 _amountTokens) external whenNotPaused {
	        require(_amountTokens >= minAmount, "below min");
	        require(_amountTokens <= maxAmount, "above max");

	        //you cannot buy more than the contract has
	        uint256 balToken = theToken.balanceOf(address(this));
	        require(_amountTokens <= balToken, "contract doesn't have enough balance");

	        //how much would it pay for them
	        uint256 usdRequiredAmount = (priceForTokenInUSD * _amountTokens) / (10**18);

	        //transfer the usdtoken
	        require(
	            usdToken.transferFrom(msg.sender, address(this), usdRequiredAmount),
	            "failed to transfer"
	        );

	        //finally give the user the tokens
	        theToken.transfer(msg.sender, _amountTokens);
	    }
	```
	Line 121 - 126 (Exchange.sol)
	```
	function withdrawTokens(IERC20 token, uint256 _amount) public onlyOwner {
	        require(address(token) != address(0));
	        uint256 balance = token.balanceOf(address(this));
	        require(_amount <= balance, "not enough balance");
	        token.transfer(msg.sender, _amount);
	    }
	```
	Line 117 - 127 (NftStaking.sol)
	```
	  function claimRewards() external {
	```

```
        uint256 rewards = calculateRewards(msg.sender);
        stakers[msg.sender].rewards += rewards;
        uint256 availableRewardsAmt = stakers[msg.sender].rewards - stakers[msg.sender].paid;
        require(availableRewardsAmt > 0, "You have no rewards to claim");
        stakers[msg.sender].timeOfLastUpdate = block.timestamp;
        stakers[msg.sender].incomeIndex = incomes.length;
        stakers[msg.sender].paid += availableRewardsAmt;
        totalPaid += availableRewardsAmt;
        rewardsToken.transfer(msg.sender, availableRewardsAmt);
    }
```

Result/Recommendation	We recommend using safeTransfer function from the safeERC20 Implementation, or put the transfer call inside an assert or require, to verify that it returned true.

LOW ISSUES

During the audit, Chainsulting's experts found **4 Low issues** in the code of the smart contract.

6.2.4 Using .transfer() to transfer Ether

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: Exchange.sol
Update: e6cc3c2bfa2866ae947c02fbe7df4e3020b7459b

Attack / Description	.transfer() and .send() are recommended as best-practice, to prevent reentrancy attacks, because they only forward 2300 gas, the gas repricing of opcodes in the future may require more than 2300 gas, therefore the transaction will never pass successfully.
Code	Line 16 – 18 (Exchange.sol)

```
//owner can withdraw ETH
    function withdraw() public onlyOwner {
        payable(msg.sender).transfer(address(this).balance);
```

	`}`
Result/Recommendation	We recommend using .call{ value: ... }("") instead, without hardcoded gas limits along with checks-effects-interactions pattern or reentrancy guards for reentrancy protection.

6.2.5 Missing Value Verification

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: Exchange.sol, SpiExtStaking.sol
Update: e6cc3c2bfa2866ae947c02fbe7df4e3020b7459b

Attack / Description	Certain functions lack a value safety check, the values of the arguments should be verified to allow only the ones that comply with the contract's logic. In the constructor, from Exchange.sol the _newMaxAmount variable should be higher than the maxAmount, also, the _newMaxAmount should be lower than the minAmount. In the constructor, the _defaultLockInDuration from SpiExtStaking.sol variable should be different from 0.

Code	Line 76 – 78 (Exchange.sol)
	`    //sets the minimum tokens amount`
	`    function setMinAmount(uint256 _newMinAmount) public onlyOwner {`
	`        minAmount = _newMinAmount;`
	`    }`
	
	Line 81 –83 (Exchange.sol)
	`//sets the maximum tokens amount`
	`    function setMaxAmount(uint256 _newMaxAmount) public onlyOwner {`
	`        maxAmount = _newMaxAmount;`
	`    }`

	Line 86 – 89 (SpiExtStaking.sol)
	
	`constructor(IERC20 _stakingToken, uint256 _defaultLockInDuration) {`
	`      stakingToken = _stakingToken;`
	`      defaultLockInDuration = _defaultLockInDuration;`
	`    }`

Result/Recommendation	Consider verifying the values of the arguments using a require statement, to allow only the ones that comply with the contract's logic.

6.2.6 Zero Address Checks Missing

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: Exchange.sol, NftStaking.sol, SpiExtStaking.sol
Update: e6cc3c2bfa2866ae947c02fbe7df4e3020b7459b

Attack / Description	In the current implementation several functions are not checking for zero addresses. The address-type arguments should include a zero-address test, otherwise, the contract's functionality may become inaccessible. The _theToken ,_usdToken, nftCollection, _rewardsToken and _stakingToken arguments should be verified to be different from the address(0).

Code	Line 23 – 29 (Exchange.sol)
	`    constructor(IERC20 _theToken, IERC20 _usdToken) {`
	`        theToken = _theToken;`
	`        priceOracle = msg.sender;`
	`        priceForTokenInWei = 1 ether;`
	`        priceForTokenInUSD = 1 ether;`
	`        usdToken = _usdToken;`
	`    }`

	Line 86 – 88 (Exchange.sol)
	
	`//sets the price oracle address`
	`    function setPriceOracle(address _newAddress) public onlyOwner {`
	`        priceOracle = _newAddress;`
	`    }`
	
	Line 61 – 64 (NftStaking.sol)
	
	`constructor(IERC721 _nftCollection, IERC20 _rewardsToken) {`
	`        nftCollection = _nftCollection;`
	`        rewardsToken = _rewardsToken;`
	`    }`
	
	Line 86 – 89 (SpiExtStaking.sol)
	
	`constructor(IERC20 _stakingToken, uint256 _defaultLockInDuration) {`
	`      stakingToken = _stakingToken;`
	`      defaultLockInDuration = _defaultLockInDuration;`
	`    }`

Result/Recommendation	It is recommended to check addresses for the zero address, before setting them.

6.2.7 The Owner Can Renounce Ownership
Severity: LOW
Status: ACKNOWLEDGED
Code: NA
File(s) affected: Exchange.sol

Attack / Description	Generally, the account that deploys the contract is also its owner. Consequently, the owner is able to engage in certain privileged activities in his own name. In smart contracts, the

	renounceOwnership function is used to renounce ownership, which means that if the contract's ownership has never been transferred, it will never have an Owner, rendering some owner-exclusive functionality unavailable.
Code	Line 10 (Exchange.sol) `contract Exchange is Ownable, Pausable {`
Result/Recommendation	We recommend that you prevent the owner from calling renounceOwnership without first transferring ownership to a different address. Additionally, if you decide to use a multi-signature wallet, then the execution of the renounceOwnership will require for at least two or more users to be confirmed. Alternatively, you can disable Renounce Ownership functionality by overriding it.

INFORMATIONAL ISSUES

During the audit, Chainsulting's experts found **1 Informational issue** in the code of the smart contract.

6.2.8 Floating Pragma Version Identified

Severity: INFORMATIONAL
Status: ACKNOWLEDGED
Code: SWC-103
File(s) affected: ALL

Attack / Description	It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.
Code	Line 1 `pragma solidity ^0.8.0;`
Result/Recommendation	It is recommended to follow the latter example, as future compiler versions may handle certain

	language constructions in a way the developer did not foresee. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version. i.e. Pragma solidity 0.8.0

6.2.9 Spelling Or Grammatical Errors

Severity: INFORMATIONAL
Status: OPEN
Code: NA
File(s) affected: Exchange.sol

Attack / Description	Spelling and grammatical errors were identified in the codebase. Fixing these mistakes can help improve the end-user experience by providing clear information on errors encountered and improve the maintainability and auditability of the codebase.
Code	Line 13 (Exchange.sol) `uint256 public errorPercetage = 1;`
Result/Recommendation	Better: errorPercentage https://dictionary.cambridge.org/dictionary/english/percentage

6.3 SWC Attacks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	✅
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✅
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✅
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✅
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	✅
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✅
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	✅
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	✅

ID	Title	Relationships	Test Result
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	✓
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	✓
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	✓
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	✓
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	✓
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	✓
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✓
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	✓
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✓

ID	Title	Relationships	Test Result
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	✓
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✓
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	✓
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	✓
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	✓
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	✓
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	✓
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	✓
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	✓
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	✓

ID	Title	Relationships	Test Result
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	X
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	✓
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	✓
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	✓

6.4. Verify Claims

6.4.1 Staking is working as expected and the owner is not able to drain out user funds
Status: tested and verified ✅

6.4.2 Rewards/Dividends are correctly calculated and possible to withdraw
Status: tested and verified ✅

6.4.3 The smart contract is coded according to the newest standards and in a secure way.
Status: tested and verified ✅

7. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase.

The main goal of the audit was to verify the claims regarding the security of the smart contract. During the audit, no critical, two high, one medium, four low and one Informational issues have been found, after the manual and automated security testing. We advise the shopping.io team to implement the recommendations to further enhance the code's security and readability.

8. Deployed Smart Contract

VERIFIED

Staking: https://etherscan.io/address/0xF6e11C0b1dF0A129c9B9Fb424CB8F6617030316A#code

NFT Staking : https://etherscan.io/address/0x86ee966910bd3e29c4acf0f0afcec2f1c4a11c76#code

Exchange: https://etherscan.io/address/0x372Fee526EB3E7906128578451a4Fed367eDE50C#code

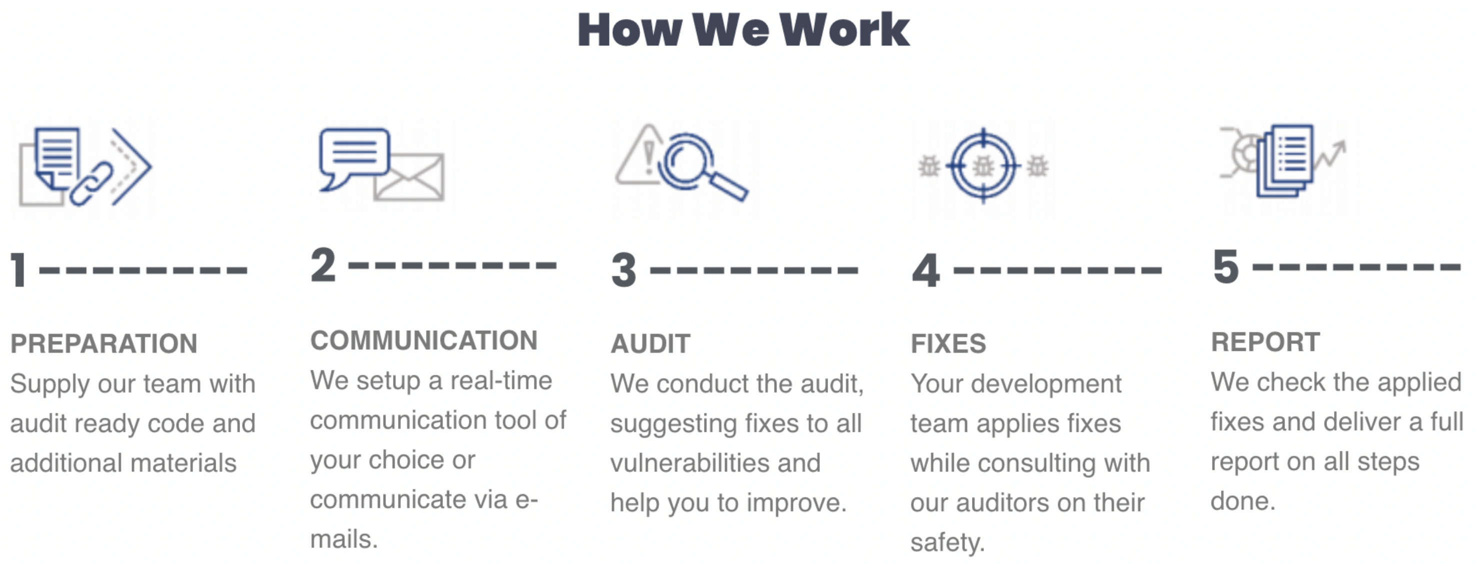
9. About the Auditor

Chainsulting is a professional software development firm, founded in 2017 and based in Germany. They show ways, opportunities, risks and offer comprehensive web3 solutions. Their services include web3 development, security and consulting.

Chainsulting conducts code audits on market-leading blockchains such as Solana, Tezos, Ethereum, Binance Smart Chain, and Polygon to mitigate risk and instil trust and transparency into the vibrant crypto community. They have also reviewed and secure the smart contracts of 1Inch, POA Network, Unicrypt, LUKSO among numerous other top DeFi projects.

Chainsulting currently secures [$100 billion](#) in user funds locked in multiple DeFi protocols. The team behind the leading audit firm relies on their robust technical know-how in the web3 sector to deliver top-notch smart contract audit solutions, tailored to the clients' evolving business needs.

Check our website for further information: https://chainsulting.de



How We Work

1 --------

PREPARATION
Supply our team with audit ready code and additional materials

2 --------

COMMUNICATION
We setup a real-time communication tool of your choice or communicate via e-mails.

3 --------

AUDIT
We conduct the audit, suggesting fixes to all vulnerabilities and help you to improve.

4 --------

FIXES
Your development team applies fixes while consulting with our auditors on their safety.

5 --------

REPORT
We check the applied fixes and deliver a full report on all steps done.