



**GSPI Staking**  
**SMART CONTRACT AUDIT**  
**01.05.2021**

**Made in Germany by Chainsulting.de**



## Table of contents

1. Disclaimer .....	3
2. About the Project and Company .....	4
2.1 Project Overview .....	5
3. Vulnerability & Risk Level.....	6
4. Auditing Strategy and Techniques Applied .....	7
4.1 Methodology.....	7
4.2 Used Code from other Frameworks/Smart Contracts .....	8
4.3 Tested Contract Files.....	9
4.4 Metrics / CallGraph .....	10
4.5 Metrics / Source Lines .....	11
4.6 Metrics / Capabilities.....	12
4.7 Metrics / Source Unites in Scope.....	13
5. Scope of Work.....	14
5.1 Manual and Automated Vulnerability Test .....	15
5.1.1 Unsecure calculation .....	15
5.1.2 Wrong import of OpenZeppelin library.....	16
5.1.3 A floating pragma is set .....	17
5.1.4 Hardcoded address .....	17
5.1.5 Missing natspec documentation .....	18
5.1.6 Old pragma version .....	18
6. Test Deployment.....	19
6.1 Verify Claims .....	20



7. Executive Summary .....	22
8. Deployed Smart Contract.....	22

## 1. Disclaimer

The audit makes no statements or warranties 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 AZ EXPRESS RETAIL LLC. 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 (28.04.2021)	Layout
0.5 (29.04.2021)	Automated Security Testing Manual Security Testing
0.7 (29.04.2021)	Test Deployment
0.8 (29.04.2021)	Verify Claims
0.9 (29.04.2021)	Summary and Recommendation
1.0 (30.04.2021)	Final Document
1.1 (01.05.2021 )	Deployed Contract

## 2. About the Project and Company

### **Company address:**

AZ EXPRESS RETAIL LLC  
4281 EXPRESS LN  
SARASOTA  
FLORIDA, 34249  
United States

**Website:** <https://shopping.io>

**Instagram:** [https://www.instagram.com/shopping.io\\_official/](https://www.instagram.com/shopping.io_official/)

**Twitter:** [https://twitter.com/shopping\\_io](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.

## 4.2 Used Code from other Frameworks/Smart Contracts

1. IERC20.sol

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.2.0/contracts/token/ERC20/IERC20.sol>

2. SafeMath.sol

<https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.2.0/contracts/math/SafeMath.sol>



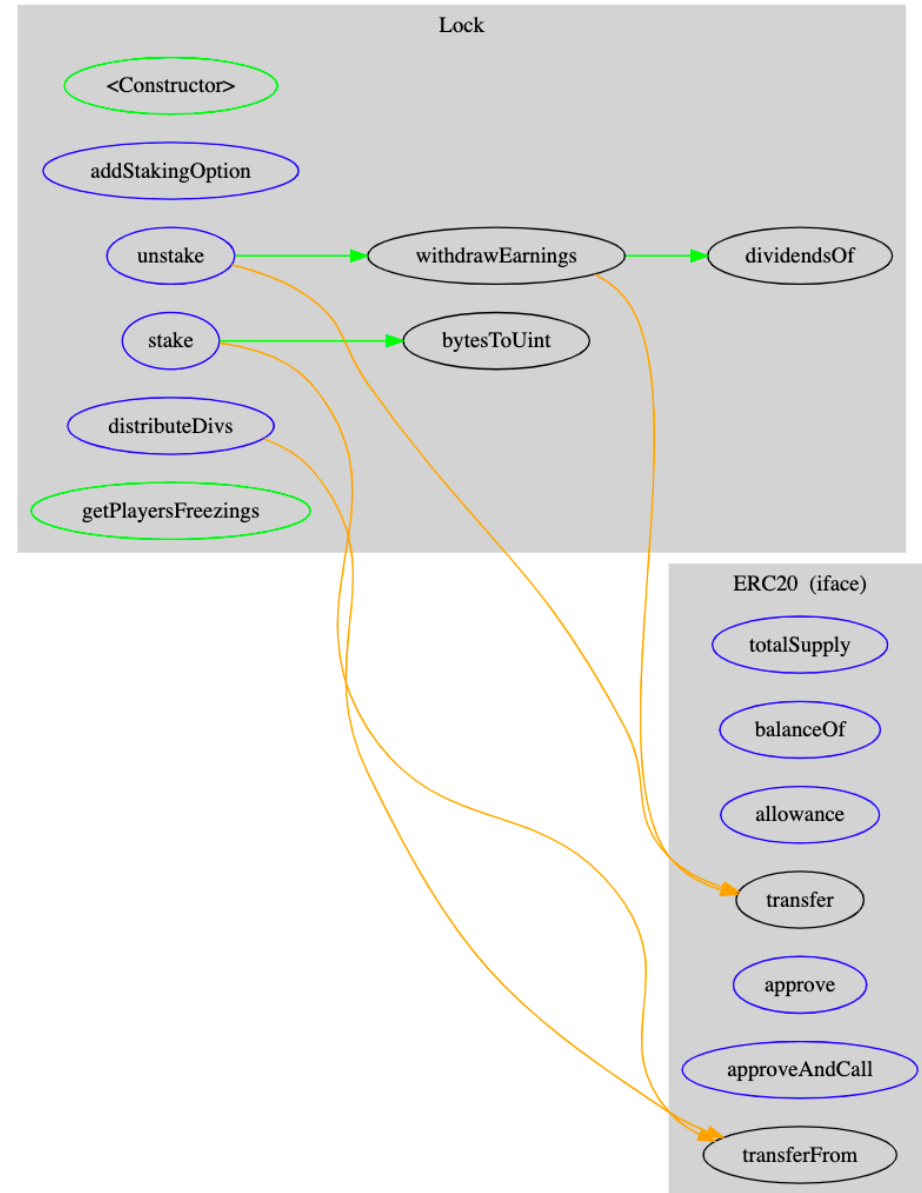
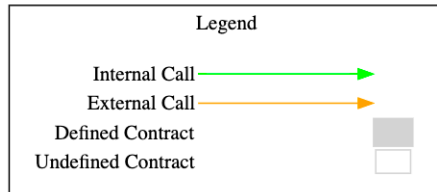


## 4.3 Tested Contract Files

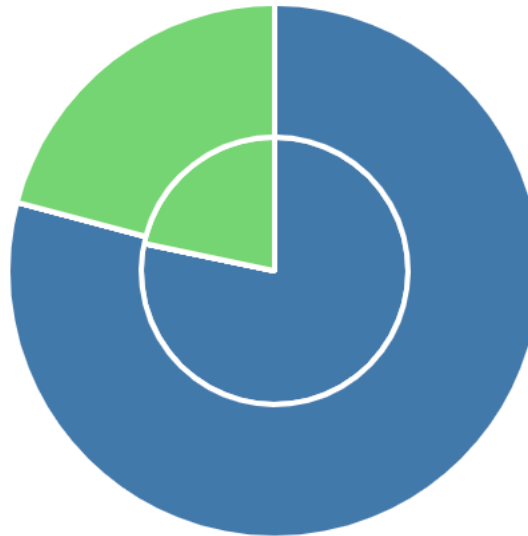
The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

File	Fingerprint (SHA256)
gspl_staking.sol	dcfe76addbb93d9fcc3af47cace560c













## 4.4 Metrics / CallGraph




## 4.5 Metrics / Source Lines







## 4.6 Metrics / Capabilities

Solidity Versions observed		 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts
^0.5.13				**** (0 asm blocks)	
 Transfers ETH	 Low-Level Calls	 DelegateCall	 Uses Hash Functions	 ECTRecover	 New/Create/Create2
yes					
 Public	 Payable				
16	0				
External	Internal	Private	Pure	View	
11	10	0	1	5	

### StateVariables

Total	 Public
10	7

## 4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	gspi_staking.sol	1	1	139	129	101		104	
	<b>Totals</b>	<b>1</b>	<b>1</b>	<b>139</b>	<b>129</b>	<b>101</b>	<b>0</b>	<b>104</b>	

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, ...)

## 5. Scope of Work

The Shopping.io Team provided us with the file that needs to be tested. The scope of the audit is the GSPI Staking contract.

Following contracts with the direct imports has been tested:

- gspi\_staking.sol

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

- The users can only withdraw locked amount and bonus, after the unlockEpoch is reached.
- Contract deployer is not able to withdraw users locked funds or bonus
- The user is not able to withdraw bonus and locked amount, before unlockEpoch
- The stakingBonus is correctly calculated
- Mathematical operations within the contract are safe
- Check the overall smart contract security and functions

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.

## 5.1 Manual and Automated Vulnerability Test

### CRITICAL ISSUES

During the audit, Chainsulting's experts found **no Critical issues** in the code of the smart contract.

### HIGH ISSUES

#### 5.1.1 Unsecure calculation

Severity: HIGH

Status: **Fixed**

File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
The calculations could cause unhandled errors such as integer under- or overflow and division by zero.	Overall	We recommend using the OpenZeppelin library SafeMath to avoid unsecure calculations.

## MEDIUM ISSUES

### 5.1.2 Wrong import of OpenZeppelin library

Severity: MEDIUM

Status: Acknowledged

File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation, OpenZeppelin files are added directly into the code. This violates OpenZeppelin's MIT license, which requires the license and copyright notice to be included if its code is used. Moreover, updating code manually is error-prone.	SafeMath, IERC20	We highly recommend using npm (import "@openzeppelin/contracts/..") in order to guarantee that original OpenZeppelin contracts are used with no modifications. This also allows for any bug-fixes to be easily integrated into the codebase.

## LOW ISSUES

During the audit, Chainsulting's experts found **no Low issues** in the code of the smart contract.



## INFORMATIONAL ISSUES

### 5.1.3 A floating pragma is set

Severity: INFORMATIONAL

Status: Fixed

File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
The current pragma Solidity directive is <b>^0.5.13</b> ; It is recommended to specify a fixed 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.	Line 1 pragma solidity ^0.5.13;	It is recommended to follow the example(0.5.13), as future compiler versions may handle certain language constructions in a way the developer did need foresee. Not effecting the overall contract functionality.

### 5.1.4 Hardcoded address

Severity: INFORMATIONAL

Status: ACKNOWLEDGED (GSPI Contract address is verified)

<https://bscscan.com/address/0x95CBb5c3f3c23c37D5be5414C591BA3B8B3bbcF1#code>

File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
The contract contains unknown address. This address might be used for some malicious activity. Please check	Line 5 ERC20 <b>constant</b> token = ERC20( <b>0xB42e1c3902b85b410334f5fff79cDc51fBeE6950</b> );	We recommend checking the address. Also, it is required to check the code of the called contract for vulnerabilities.



hardcoded address and it's usage.		
-----------------------------------	--	--

#### 5.1.5 Missing natspec documentation

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
Solidity contracts can use a special form of comments to provide rich documentation for functions, return variables and more. This special form is named the Ethereum Natural Language Specification Format (NatSpec).	Overall	It is recommended to include natspec documentation and follow the doxygen style including @author, @title, @notice, @dev, @param, @return and make it easier to review and understand your smart contract.

#### 5.1.6 Old pragma version

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
Old pragma version can lead to errors	Line 1 pragma solidity ^0.5.13;	Use at least 0.6.12

## 6. Test Deployment

Deployment and tests have been performed on the Ethereum testnet, as Binance Smart Chain is using the same EVM

### 6.0.1 Deployment of ERC20-Token as test token (GSPIX)

Tx: <https://kovan.etherscan.io/tx/0xc0583b34f053f0a39ed205b178c19461d65b5beb54ee35c24206fd42b379ba1d>

Contract: <https://kovan.etherscan.io/address/0xd54f70ee738866ac18560ce842cdb337c4d73560>

### 6.0.2 Deployment of lock contract

Tx: <https://kovan.etherscan.io/tx/0x0813242a5033cd55158a0941d284648ef2d74522e9070bcc12941198dd9c3918>

Contract: <https://kovan.etherscan.io/address/0x717813753081f678cfa8e90849f660aa41c9087f>

## 6.1 Verify Claims

### 6.1.1 The user is not able to withdraw bonus and locked amount, before unlockEpoch

The user is not able to withdraw the staked amount before *unlockEpoch*. The function call *unstake* requires the current timestamp to be greater than the timestamp of the *unlockEpoch*.

```
66 | | require(freeze.unlockEpoch <= now);
```

Withdraw staked amount before unlockEpoch fails as expected:

Tx: <https://kovan.etherscan.io/tx/0x0254124c29ce2950e322667db4257bea98f2184e50d5fade550862a44eb80de6>

But the user is able to withdraw earnings (bonus) at any time independent from *unlockEpoch* by calling *withdrawEarnings* function.

Tx: <https://kovan.etherscan.io/tx/0x73110c12abbd9c3d896aa76bc7f2bfc06d1d2a31a1db9fdc11a3bfb75cea91b0>

### 6.1.2 The users can only withdraw locked amount and bonus, after the unlockEpoch is reached.

After the unlockEpoch the user is able to withdraw staked amount and earnings (bonus). The withdrawing of earnings is also possible before unlockEpoch is reached (see 2.1).

Tx: <https://kovan.etherscan.io/tx/0xae0fd7d1b8aeb93bea60094532038b23d1804ca62e57e53bfea3862c8303aca1>

### 6.1.3 Contract deployer is not able to withdraw users locked funds or bonus

The staked funds and bonus are strictly mapped to the users address. The owner has no opportunity to withdraw or claim bonus and funds staked by other users.

#### 6.1.4 The stakingBonus is correctly calculated

The staking bonus is calculated correctly, but if no one calls the *distributeDivs* function and sends token to the contract, there are no rewards for staking. Users can only get earnings if someone sends token to the contract by calling *distributeDivs* function.

Sending 10 GSPIX as reward to the contract by calling *distributeDivs* function:

‣ From 0x4126041020a00... To 0x717813753081f... For 10  GSPIX (GSPIX)

Tx: <https://kovan.etherscan.io/tx/0xa41b4e1bfa8dc21e74eedf44c7115debd7fe36eac544745cc1239a4050725234>

Get equally split amount as reward for same staked amount for two different stakers:

‣ From 0x717813753081f... To 0xdd90261f9a6a4... For 4.9999999999999999  GSPIX (GSPIX)

‣ From 0x717813753081f... To 0xdd90261f9a6a4... For 10  GSPIX (GSPIX)

Tx: <https://kovan.etherscan.io/tx/0xf5fb5cb72993046e125e417f6f58e9f2bb9bdaed23bef00787a8db1647321a16>

‣ From 0x717813753081f... To 0x4126041020a00... For 4.9999999999999999  GSPIX (GSPIX)

‣ From 0x717813753081f... To 0x4126041020a00... For 10  GSPIX (GSPIX)

Tx: <https://kovan.etherscan.io/tx/0x21fb2749648d83ba94aada71b566eae4223f7b96a130dc10705110688bb5839>

#### 6.1.5 Mathematical operations within the contract are safe

All mathematical operations are not safe and can potentially under- or overflow and cause division by zero errors. It is strongly recommended to use the SafeMath library of OpenZeppelin to secure mathematical calculations.

## 7. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase. The final debriefs took place on the April 30, 2021. The main goal of the audit was to verify the claims regarding the security of the smart contract and the claims inside the scope of work. During the audit, no critical issues were found after the manual and automated security testing. The high issue regarding the import of SafeMath and similar medium issue, should be considered before deploying the code.

## 8. Deployed Smart Contract

VERIFIED

GSPI Staking

<https://bscscan.com/address/0x95CBb5c3f3c23c37D5be5414C591BA3B8B3bbcF1#code>

