



GSPI Club

SMART CONTRACT AUDIT

02.09.2021

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 & Risk	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 Missing natspec documentation	15
5.1.2 A floating pragma is set	16
5.2. SWC Attacks	17
5.3 Verify Claims	21
6. Executive Summary	22
7. 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 (Shopping.io). 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 (01.09.2021)	Layout
0.5 (01.09.2021)	Automated Security Testing Manual Security Testing
0.7 (02.09.2021)	Test Deployment
0.8 (02.09.2021)	Verify Claims
0.9 (02.09.2021)	Summary and Recommendation
1.0 (02.09.2021)	Final Document
1.1 (02.09.2021)	Added 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/

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.



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

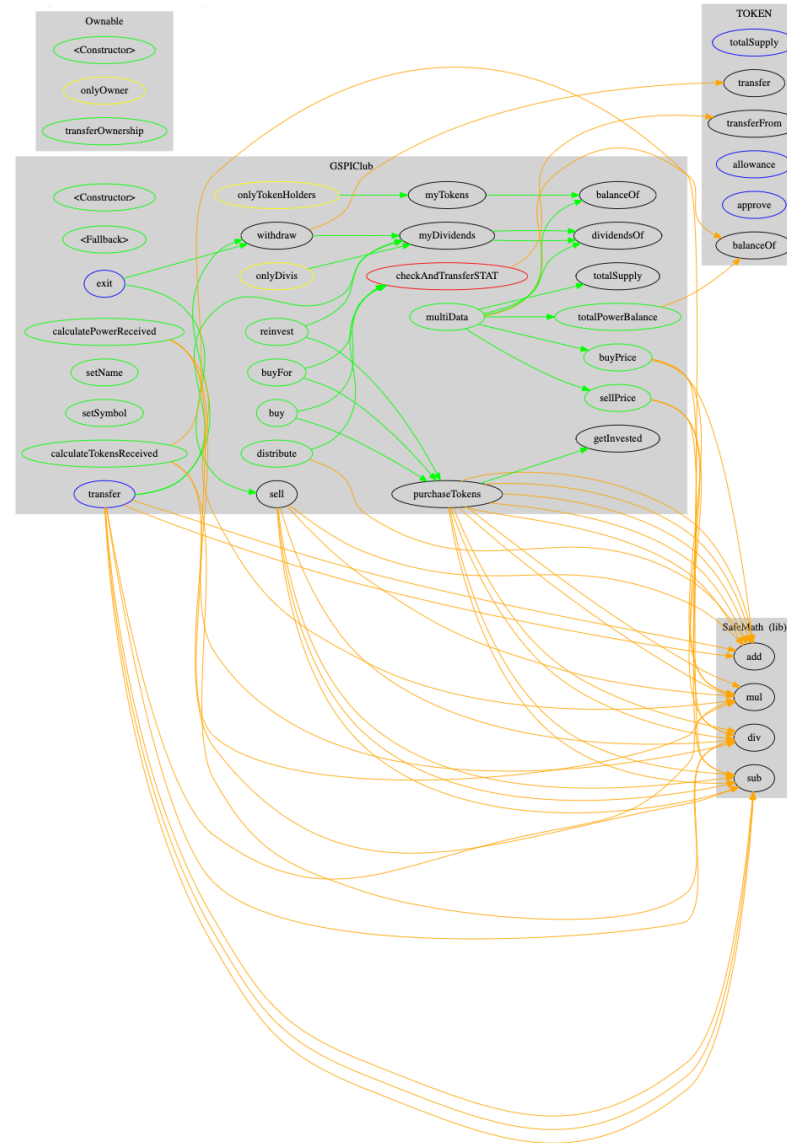
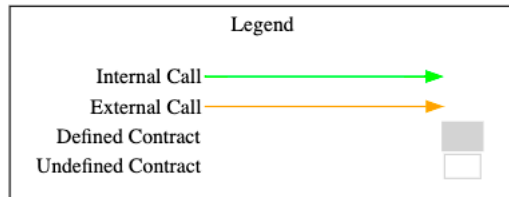
Dependency / Import Path	Source
SafeMath	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.0.1/contracts/math/SafeMath.sol
IERC20Transfer	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.0.1/contracts/token/ERC20/IERC20.sol
Ownable	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v2.0.1/contracts/access/Ownable.sol

4.3 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

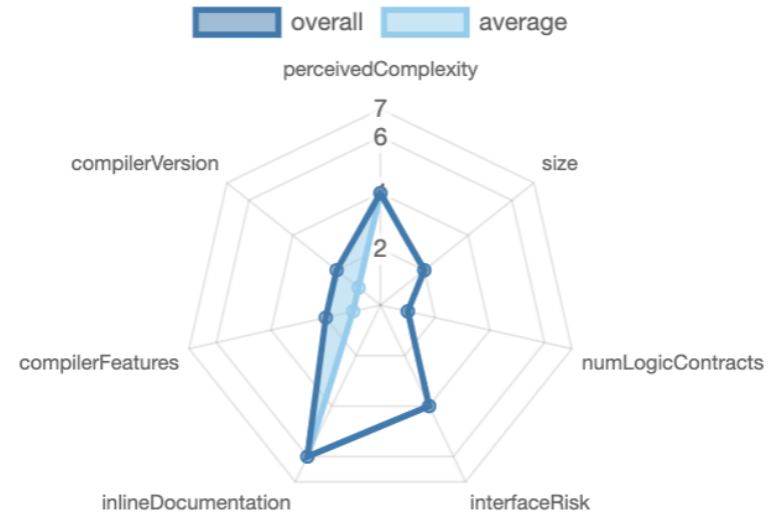
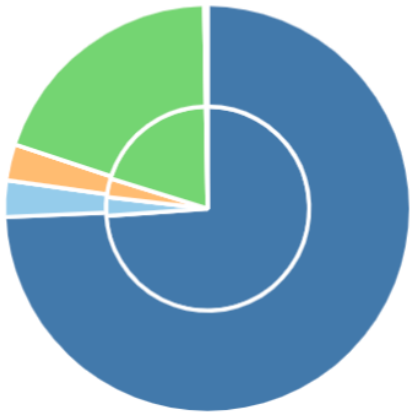
File	Fingerprint (MD5)
gspl_club.sol	9f08a6e417c39c2c901f63cdc9e490e3

4.4 Metrics / CallGraph



4.5 Metrics / Source Lines & Risk

source comment single block mixed
empty todo blockEmpty





4.6 Metrics / Capabilities


Solidity Versions observed		 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts
<code>^0.4.26</code>			<code>yes</code>	**** (0 asm blocks)	
 Transfers ETH	 Low-Level Calls	 DelegateCall	 Uses Hash Functions	 ECTrecover	 New/Create/Create2
<code>yes</code>					

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				
32	1				
External	Internal	Private	Pure	View	
8	23	1	4	15	

StateVariables

Total	 Public
19	7

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	gspi_club.sol	4	_____	369	358	274	11	211	
	Totals	4	_____	369	358	274	11	211	

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 Club contract.

Following contracts with the direct imports has been tested:

- gspi_club.sol

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

- The user can stake GSPI at any time.
- Contract deployer or owner is not able to withdraw staked GSPI from user
- The fees and dividends are correctly calculated
- Mathematical operations within the contract are safe
- Deployer/Owner cannot burn any staked funds
- 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.



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

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

MEDIUM ISSUES

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

LOW ISSUES

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

INFORMATIONAL ISSUES

5.1.1 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).	NA	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.



--	--	--

5.1.2 A floating pragma is set

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
The current pragma Solidity directive is ^0.4.26 ; 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 ^ 0.4.26;	It is recommended to follow the example(0.4.26), as future compiler versions may handle certain language constructions in a way the developer did need foresee. Not effecting the overall contract functionality.

5.2. 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	✗
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	✓

5.3 Verify Claims

5.3.1 The user can stake GSPI at any time.

Status: tested and verified ✓

5.3.2 Contract deployer or owner is not able to withdraw staked GSPI from user

Status: tested and verified ✓

There is no functionality in the contract, which allows the owner of the contract to withdraw or claim users staked funds and rewards.

5.3.3 The fees and dividends are correctly calculated

Status: tested and verified ✓

5.3.4 Mathematical operations within the contract are safe

Status: tested and verified ✓

The contract uses SafeMath library from OpenZeppelin, which makes mathematical calculations secure.

5.3.5 Deployer/Owner cannot burn any staked funds

Status: tested and verified ✓

There is no functionality in the contract, which allows the owner of the contract to burn users staked funds.

5.3.6 The smart contract is coded according to the newest standards and in a secure way.

Status: tested and verified ✓

6. 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 September 02, 2021. The code is not overloaded with unnecessary functions; these is greatly benefiting the security of the contract.

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. Only informational issues were found, to increase the code quality.

7. Deployed Smart Contract

VERIFIED

GSPI Club

<https://bscscan.com/address/0xb5C8fa958Cd4E0b9f465269459Eb28fd3fc21D2b#code>

