SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

Galo Swap

Audit

Security Assessment
15. April, 2023

For

GALOSWAP

SolidProof_io

@solidproof_io

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	12. April 2023 - 14. April 2023	• Layout project • Automated- /Manual-Security Testing • Summary

Network
zkSync

Website
https://galoswap.io/

Telegram
https://t.me/GaloSwap

Twitter
https://twitter.com/GaloSwap

Description

Galoswap is a community-driven project that aims to solve the liquidity problem on ZKsync ecosystem, using ZK-rollups technology for optimizing security and scalability The project offers a comprehensive range of products and services, including a Decentralized Exchange, a Decentralized Perpetual Exchange, NFT marketplace support trading vesting tokens, farming, staking, and a Launchpad.

Project Engagement

During the 11th of April 2023, **GaloSwap 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 Links

v1.0

https://github.com/GaloSwap/GALO_SC_PUBLIC_SC

Commit: 8f0f336

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:

```
./interfaces/IGaloPair.sol
./libraries/UniswapV2ERC20.sol
./libraries/Math.sol
./libraries/interfaces/IERC20.sol
./interfaces/IGaloFactory.sol
./libraries/interfaces/IUniswapV2Callee.sol
```

```
./libraries/TransferHelper.sol
./interfaces/IGaloFactory.sol
./interfaces/IGaloPair.sol
./libraries/interfaces/IERC20.sol
./interfaces/IGaloRouter.sol
./libraries/UniswapV2Library.sol
./libraries/SafeMath.sol
./interfaces/IWETH.sol
```

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.

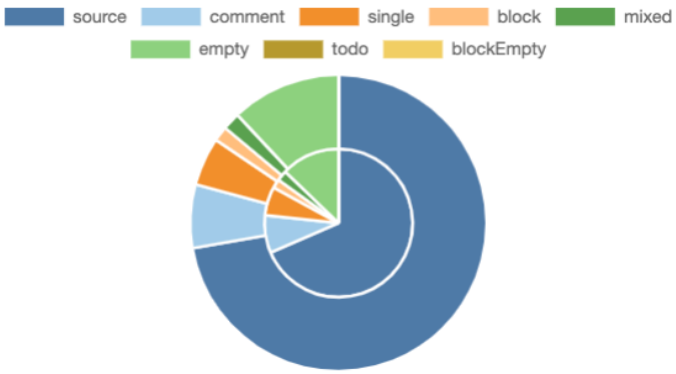
v1.0

File Name	SHA-1 Hash
contracts/interfaces/IGaloRouter.sol	b281b6973faf0eca501fea54f0d8d6bb9e6e29b4
contracts/interfaces/IWETH.sol	c7de86ffc444d87cb8d773b41fb2081a97733adf
contracts/interfaces/IGaloFactory.sol	ba6336fc12f313ffd0c756a589f396f22b9c4786
contracts/interfaces/IGaloPair.sol	4f6aa6b0f35ac744140c77c18c0525a1d4fb5d7b
contracts/GaloPair.sol	92e8f0443cd2bc7920a12be79f2e363f373bbcc2
contracts/libraries/TransferHelper.sol	b4998089dd6c3986b5acb6a61d094b976c2b88fe
contracts/libraries/Math.sol	56b937b584ca03d4e1a25730f28a63ee965d526c
contracts/libraries/SafeMath.sol	57593f7a53069fa66755b6d533d0448764f3965f
contracts/libraries/UniswapV2Library.sol	f30dade45639300771ce4fdde1cba446f4f0b43d
contracts/libraries/GaloFactory.sol	b7b0b8887fa4da5e08c99e4e927176e6de74361a
contracts/libraries/UniswapV2ERC20.sol	ddf341917086421d9d7c458a07150bbe5dc2dd5c

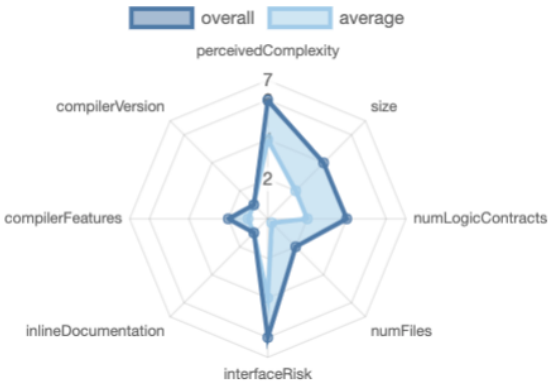
contracts/GaloRouter.sol	f6eb6a03067a055724f20118b81ffd5aa6f478fa

Metrics

Source Lines
v1.0



Risk Level
v1.0



Capabilities

Components

📝Contracts	📚Libraries	🔍Interfaces	🎨Abstract
4	4	4	0

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
96	5

External	Internal	Private	Pure	View
88	93	5	17	34

StateVariables

Total	🌐Public
39	35

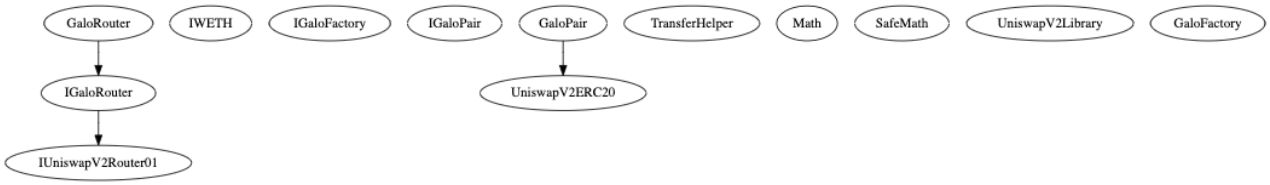
Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥️ Uses Assembly	💣 Has Destroyable Contracts
`0.6.6`		`yes`	`yes` (2 asm blocks)	—

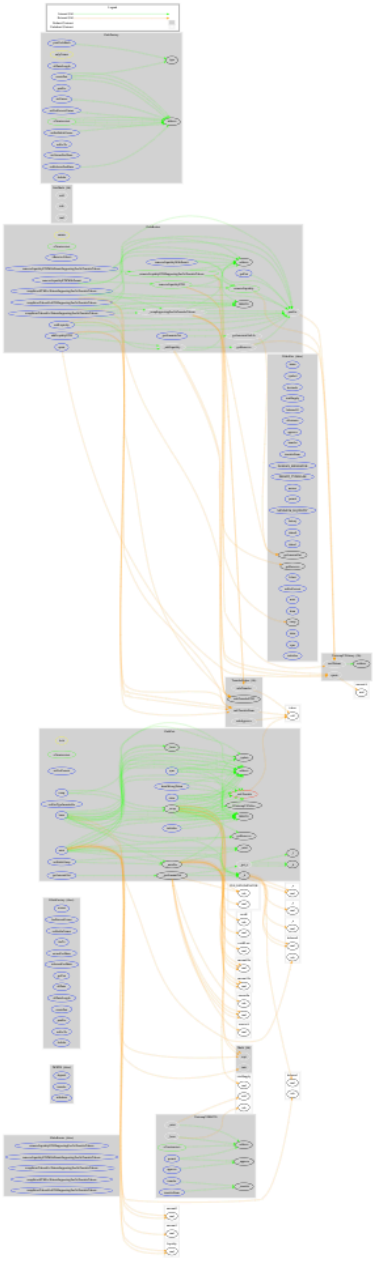
💸 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧮 Uses Hash Functions	🔏 ECRecover	🌀 New/Create/Create2
`yes`	—	—	`yes`	`yes`	`yes` → `AssemblyCall:Name:create2`

♻️ TryCatch	Σ Unchecked
—	—

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. Overall checkup (Smart Contract Security)

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

GaloFactory

```
createPair
setOwner
onlyOwner
setFeePercentOwner
onlyOwner
setSetStableOwner
setFeeTo
onlyOwner
setOwnerFeeShare
onlyOwner
setReferrerFeeShare
onlyOwner
```

Note:

- General fork from Uniswap with some added functionalities
  - Contracts inside are inspired from the Uniswap contracts

Ownership Privileges

- *Set a new owner*
- *Set fee percent, and stable owner address*
- *Set owner fee percent share up to 100% which is not recommended, beware of it*
- *Set referrer fee percent share up to a maximum of 20%*
- *Set feeTo address*
- *Factory owner address can also withdraw any tokens sent to the Pair Contract*
-

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

Source Units in Scope
v1.0

File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/interfaces/IGaloRouter.sol	—	1	52	6	4	—	16
contracts/interfaces/IWETH.sol	—	1	7	4	3	—	10
contracts/interfaces/IGaloFactory.sol	—	1	23	6	4	—	27
contracts/interfaces/IGaloPair.sol	—	1	53	7	5	—	57
contracts/GaloPair.sol	1	—	426	426	336	50	328
contracts/libraries/TransferHelper.sol	1	—	51	38	28	5	26
contracts/libraries/Math.sol	1	—	23	23	18	2	5
contracts/libraries/SafeMath.sol	1	—	17	17	12	1	4
contracts/libraries/UniswapV2Library.sol	1	—	24	24	16	2	8
contracts/libraries/GaloFactory.sol	1	—	133	133	98	20	98
contracts/libraries/UniswapV2ERC20.sol	1	—	98	98	81	1	60
contracts/GaloRouter.sol	1	—	346	247	203	16	231
Totals	**8**	**4**	**1253**	**1029**	**808**	**97**	**870**

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

Issue	File	Type	Line	Description
#1	All	Old Compiler Version	—	The contract uses an old compiler version which is not recommended for deployment because it may be susceptible to known vulnerabilities
#2	GaloFactory.sol	Missing Zero Address Validation	100	Check that the address is not zero

Informational issues

Issue	File	Type	Line	Description
#1	All	Contract doesn't import npm packages from source (like OpenZeppelin etc.)	—	We recommend importing all packages from npm directly without flattening the contract. Functions could be modified or can be susceptible to vulnerabilities

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.

15. April 2023:

- This project consists of the following forks
  - UniSwap
- Read whole report and modifiers section for more information
- The low issues that exist in the Uniswap codebase still exist in the forked code.
- We recommend using a multisig wallet for the owner address to prevent any risk of the loss of private key
- We recommend GaloSwap team to conduct unit tests to rule out any calculation errors.
- Do your own research here

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	NOT 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

SolidProofed

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY