Bitcoin.com

Liquidity Maker

SMART CONTRACT AUDIT

20.06.2023

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 Saint Bitts 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 (13.06.2022)	Layout
0.4 (14.06.2022)	Automated Security Testing Manual Security Testing
0.5 (15.06.2023)	Verify Claims and Test Deployment
0.6 (16.06.2023)	Testing SWC Checks
0.9 (17.06.2023)	Summary and Recommendation
1.0 (20.06.2023)	Final document

2. About the Project and Company

Company address:

Saint Bitts LLC
858 Zenway Blvd, Unit 15-203
C/O Corporate Solutions Ltd
Bayview Commercial Complex Nevis
Frigate Bay, Saint Kitts
Saint Kitts & Nevis

Website: https://www.bitcoin.com

Twitter: https://twitter.com/bitcoincom

Telegram: https://t.me/www_Bitcoin_com

LinkedIn: https://linkedin.com/company/bitcoin.com

Medium: https://medium.com/@Bitcoin_Com

Instagram: https://www.instagram.com/bitcoin.com_official

Facebook: https://www.facebook.com/buy.bitcoin.news

YouTube: https://www.youtube.com/channel/UCetxkZolEBHX47BqtZktbkg

2.1 Project Overview

Established in 2015, Bitcoin.com is a leading platform in the cryptocurrency space, largely recognized for introducing newcomers to the world of crypto. Founded by Roger Ver, the platform serves as a comprehensive suite for all things related to cryptocurrency and blockchain technology.

Roger Ver's vision was to provide a platform that would democratize access to financial services, breaking down barriers typically associated with traditional finance. Under his leadership, Bitcoin.com has effectively made it easy for anyone to buy, spend, trade, invest, earn, and stay up-to-date on cryptocurrency and the future of finance.

At the heart of Bitcoin.com is its mobile and web-enabled platform, which serves as a comprehensive gateway for users to access all Bitcoin.com products and services. This platform offers a variety of features, such as the ability to buy, sell, and trade cryptocurrencies, in addition to earning opportunities and educational resources.

An integral part of the platform is a multi-chain, web3 wallet, enabling users to have complete control of their crypto assets. This wallet function not only offers a secure place to store digital currencies but also allows users to utilize their crypto holdings in various ways according to their needs and wishes.

One of the unique aspects of Bitcoin.com is its commitment to providing timely, objective, and relevant news content about the crypto industry. This feature ensures that users are always informed about the latest trends, market changes, and innovations in the blockchain technology space.

Furthering its mission to educate users, Bitcoin.com hosts a Learning Center that offers comprehensive and up-to-date content about cryptocurrency. This resource caters to a wide spectrum of users, from those learning about the basic value proposition of crypto to the more experienced enthusiasts seeking deeper understanding of technologies, applications, and market insights.

Finally, a key element of the Bitcoin.com ecosystem is VERSE, the platform's own rewards and utility token. VERSE is designed to be the gateway to the world of Decentralized Finance (DeFi) for Bitcoin.com users.

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

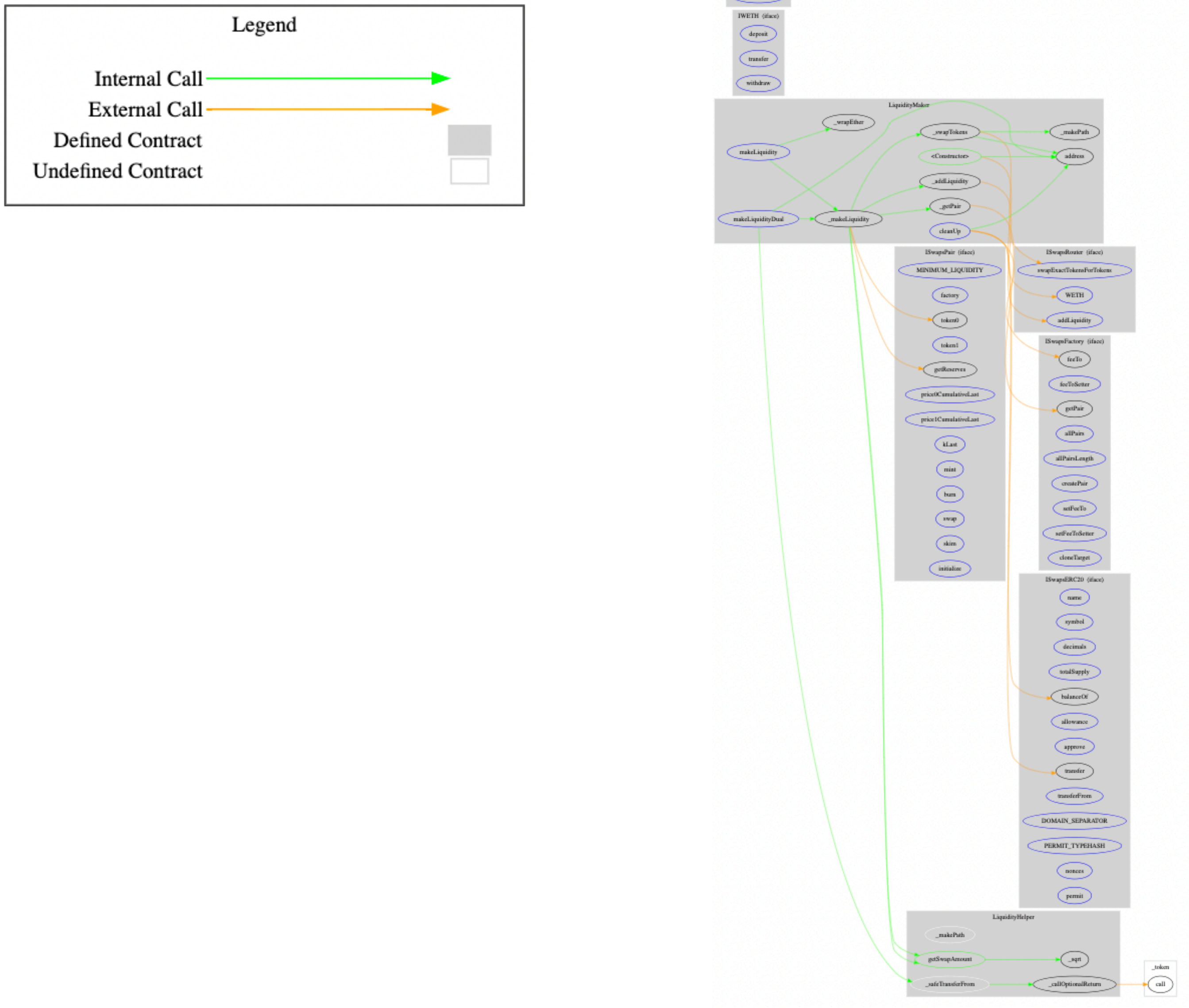
The metrics section should give the reader an overview on the size, quality, flows and capabilities of the codebase, without the knowledge to understand the actual code.

5.1 Tested Contract Files

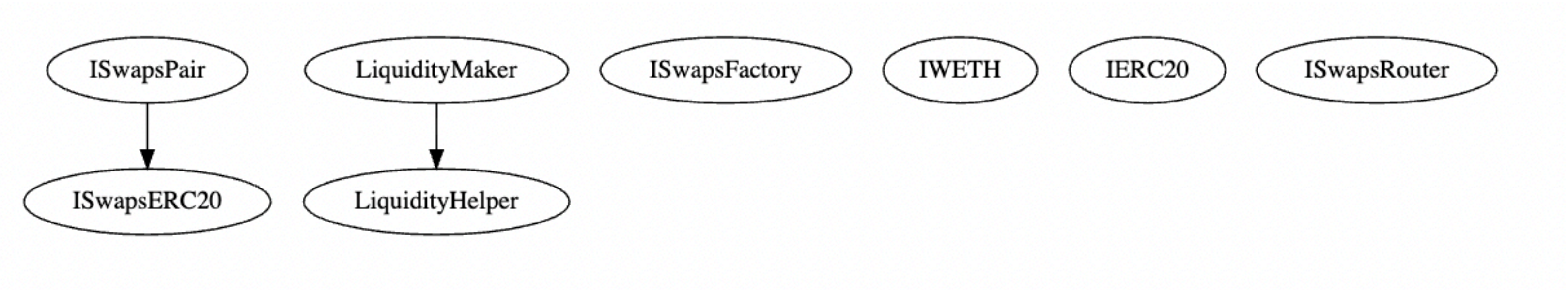
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability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

File	Fingerprint (MD5)
./ISwapsRouter.sol	ccedbe9527f4f420d6084ca63a8eec6a
./IERC20.sol	68fc36df6868a91d7eea6eb80ee4c939
./LiquidityMaker.sol	e5b0502a391215ed33a2c90f0d64f90a
./ISwapsPair.sol	c986fb2730894ade57e8d4eb092036d1
./IWETH.sol	063803e60b4a7083a76b7f4e2574f397
./ISwapsFactory.sol	575478feca2fe1afa616e117f3a427d8
./LiquidityHelper.sol	70def61301e9de9db5fca2907a33dde7
./ISwapsERC20.sol	bbf38bf21cbf9320e5bc9315414689b2

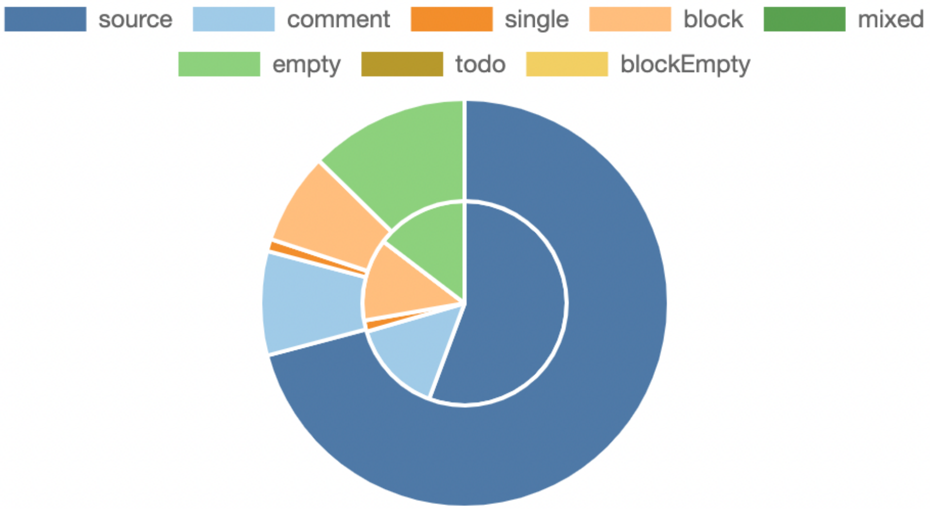
5.2 CallGraph

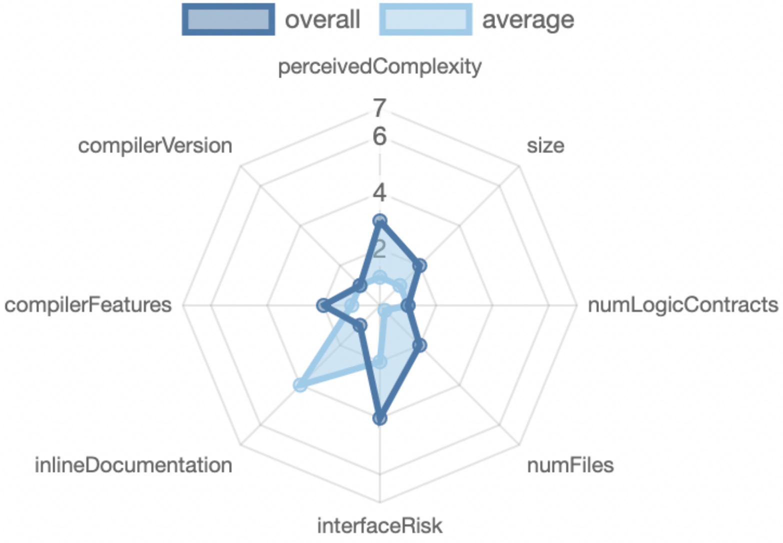


5.3 Inheritance Graph



5.4 Source Lines & Risk





5.5 Capabilities

Solidity Versions observed	🧪 **Experimental Features**	💰 **Can Receive Funds**	🖥 **Uses Assembly**	💣 **Has Destroyable Contracts**

`=0.8.19`
`^0.8.19`
		`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**
47	2

External	**Internal**	**Private**	**Pure**	**View**
46	31	5	8	20

StateVariables

Total	🌐**Public**
6	3

5.6 Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
🔍	ISwapsERC20.sol		1	91	7	3	1	27	
📝	LiquidityHelper.sol	1		128	96	61	21	29	
🔍	ISwapsFactory.sol		1	56	7	3	1	19	
🔍	IWETH.sol		1	22	7	3	1	10	💰
🔍	ISwapsPair.sol		1	84	9	4	1	29	
📝	LiquidityMaker.sol	1		306	244	171	40	51	💰📤
🔍	IERC20.sol		1	21	7	3	1	5	
🔍	ISwapsRouter.sol		1	39	7	3	1	7	
📝🔍	**Totals**	**2**	**6**	**747**	**384**	**251**	**67**	**177**	💰📤

- **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

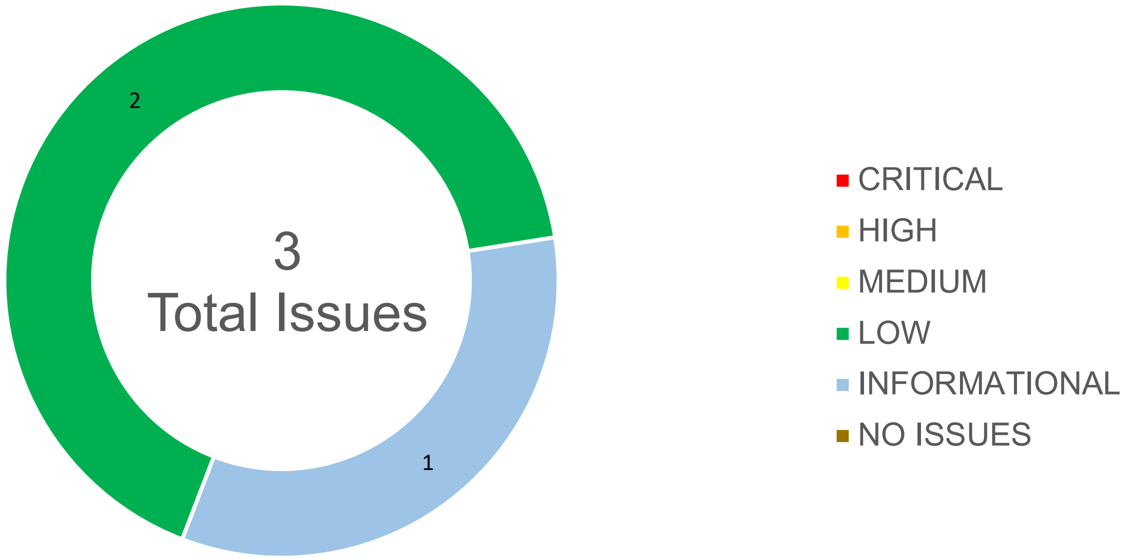
The Bitcoin.com Team provided us with the files that needs to be tested. The scope of the audit is the Liquidity Maker contract.

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

1. The contract enables users to efficiently provide liquidity to a specific token pair using ETH, making optimal swaps to balance the liquidity provision.
2. The contract allows users to add liquidity to any two ERC20 token pairs by swapping a desired amount of one token to another to ensure optimal liquidity provision.
3. The contract provides a mechanism to clear any tokens that are mistakenly sent to the contract, thereby ensuring the contract is not cluttered with unwanted tokens.
4. The contract calculates the optimal swap amount, allowing users to maximize their liquidity provision and potential return.
5. The contract is responsible for adding the specified amount of tokens to the liquidity pool, ensuring the correct transfer of tokens and the minting of liquidity provider tokens.
6. The contract developers have made sure that the contract logic is gas-efficient and optimized to minimize the risk of running out of gas during contract execution.
7. 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	Redundant References	LOW	ACKNOWLEDGED
6.2.2	Over-Approval of Tokens	LOW	ACKNOWLEDGED
6.2.3	Floating Compiler Version	INFORMATIONAL	ACKNOWLEDGED

6.2 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 **2 Low issues** in the code of the smart contract

6.2.1 Redundant References

Severity: LOW
Status: ACKNOWLEDGED
Code: CWE-398
File(s) affected: LiquidityMaker.sol

Attack / Description	In LiquidityMaker.sol, references to router and WETH are stored twice: once as an address and once as an interface. This redundancy could lead to higher gas costs and bytecode size.

Code	Line 13 – 17 (LiquidityMaker.sol)
	`address immutable WETH_ADDRESS;`
	`address immutable ROUTER_ADDRESS;`

	IWETH public immutable WETH;
	ISwapsRouter public immutable ROUTER;

Result/Recommendation	Consider using a single state variable per reference. For instance, you could remove the address references, and obtain the address when needed via address(WETH) or address(ROUTER)

6.2.2 Over-Approval of Tokens

Severity: LOW
Status: ACKNOWLEDGED
Code: CWE-400
File(s) affected: LiquidityMaker.sol

Attack / Description	By calling the makeLiquidity or makeLiquidityDual function, the contract first swaps the given token and adds afterwards the two tokens to a liquidity pool. In the internal swap function, the LiquidityMaker approves the router with a max uint256 amount. A malicious router could potentially abuse this by pulling over-approved tokens out of the contract. In the current implementation, this has no affect on the business logic because the contract should not hold any tokens and only pass them between the caller and the router. However, if the logic will be extended in the future and the LiquidityMaker may hold funds, this must be kept in mind.
Code	Line 186 – 189 (LiquidityMaker.sol) ISwapsERC20(_tokenIn).approve(ROUTER_ADDRESS, MAX_VALUE);
Result/Recommendation	It is recommended to approve the lowest required amount of tokens to prevent any attack vectors. In this case it would be sufficient to approve the swap amount and the amount to add liquidity to the pool to decrease the over-approved token amount.

INFORMATIONAL ISSUES

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

6.2.3 Floating Compiler Version

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

Attack / Description	The current pragma Solidity directive is floating. 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	`pragma solidity ^0.8.19;`
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. i.e. Pragma solidity 0.8.19 See SWC-103: https://swcregistry.io/docs/SWC-103

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 The contract enables users to efficiently provide liquidity to a specific token pair using ETH, making optimal swaps to balance the liquidity provision.
Status: tested and verified ✅

6.4.2 The contract allows users to add liquidity to any two ERC20 token pairs by swapping a desired amount of one token to another to ensure optimal liquidity provision.
Status: tested and verified ✅

6.4.3 The contract provides a mechanism to clear any tokens that are mistakenly sent to the contract, thereby ensuring the contract is not cluttered with unwanted tokens.
Status: tested and verified ✅

6.4.4 The contract calculates the optimal swap amount, allowing users to maximize their liquidity provision and potential return.
Status: tested and verified ✅

6.4.5 The contract is responsible for adding the specified amount of tokens to the liquidity pool, ensuring the correct transfer of tokens and the minting of liquidity provider tokens.
Status: tested and verified ✅

6.4.6 The contract developers have made sure that the contract logic is gas-efficient and optimized to minimize the risk of running out of gas during contract execution.
Status: tested and verified ✅

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

7. Executive Summary

Two independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase provided by Bitcoin.com Team. The main objective of the audit was to verify the security and functionality claims of the smart contract. The audit process involved a thorough manual code review and automated security testing.

Overall, the audit identified a total of 3 issues, classified as follows:
- No critical issues were found.
- No high severity issues were found.
- No medium severity issues were found.
- Two low severity issues were discovered, including an over-approval of token and redundant reference.
- One informational issues were identified, including code optimizations.

The audit report provides detailed descriptions of each identified issue, including severity levels, CWE classifications, and recommendations for mitigation. It also includes code snippets, where applicable, to demonstrate the issues and suggest possible fixes.

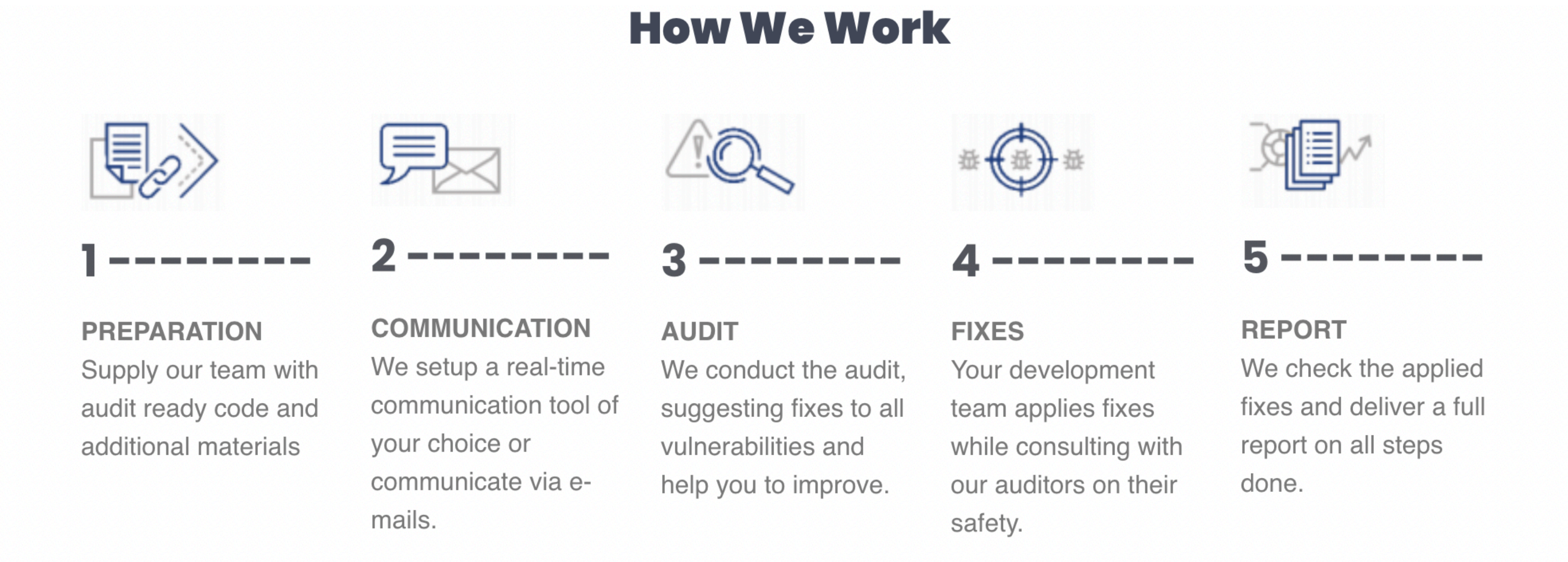
8. About the Auditor

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 many 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.