QuillAudits

Lite Technical Assessment

For

Table of Content

Project Information

Project Name	RunesBridge
Project Website	*https://runesbridge.xyz*
Date	18th April 2024 - 23rd April 2024
Executive Summary	This contract inherits the standard Openzeppelin library for the creation of ERC20 tokens. On first deployment, there's a maximum amount of tokens that token holders can hold and this would be disabled along the contract lifecycle.
Source Code	https://github.com/runesbridge/runesbridge-contracts

Assessment Summary

This document details our lite technical due diligence on RunesBridge, specifically focusing on analyzing:

- **Coding Best Practices:** We evaluated RunesBridge's codebase for adherence to established best practices in smart contract development. This ensures the code is well-structured, maintainable, and less prone to vulnerabilities.
- **Security Best Practices:** We assessed RunesBridge's implementation of security best practices. This includes measures to mitigate common attack vectors and protect user funds and assets.
- **Audit Readiness and Maturity:** We analyzed RunesBridge's overall preparedness for a full security audit. This involves factors like code documentation, clear architecture, and the presence of internal controls.

Project Maturity Assessment

1. Code Completion

Are all smart contracts for the core functionality written and finalized?

- Yes, The core functionalities of the RunesBridge, RunesRouter, and UnlockandBurn contracts have been properly implemented and finalized.

Is the Codebase Complete and Frozen for Full Audit?

- The codebase has been completed and frozen in preparation for a full audit.

Are unit tests covering all critical functionalities implemented?

- Yes, Unit Test Cases with Good Coverage is Available. Test Coverage Screenshot added on bottom section.

2. Documentation

Is there comprehensive documentation explaining the purpose and functionality of each smart contract?

- No, We will Recommend creating one document and also add a detailed natspec comment on the smart contracts.

Is the documentation clear, concise, and easy to understand for auditors?

- No.

3. External Dependencies

Are all external libraries and dependencies used in the smart contracts well-established and secure?

- The external dependencies utilized in the smart contracts are well-established and secure. These dependencies include OpenZeppelin and Uniswap, both of which have undergone thorough testing and auditing.

Are the versions of these dependencies up-to-date and free of known vulnerabilities?

- Yes.

4. Best Coding Practices

Does the code adhere to secure coding practices such as access control best practices (e.g., least privilege), proper error handling with custom errors, and secure coding practices?

- The smart contract code demonstrates adherence to secure coding practices:

1. Proper Transfer of Ether: The code correctly utilizes the call function for transferring Ether, ensuring secure and reliable transactions.
2. Access Control with “onlyOwner”: The adoption of the onlyOwner modifier effectively regulates function calls, limiting access to authorized parties.
3. Custom Errors for Error Handling: Custom error messages enhance clarity and facilitate better error handling within the contract.
4. Latest Solidity Version: The codebase utilizes the most recent version of Solidity, incorporating the latest security enhancements and features.

Are reentrancy vulnerabilities mitigated?

- Yes.

Is the code written in a gas-efficient manner? (See Gas Optimization section).

- Yes.

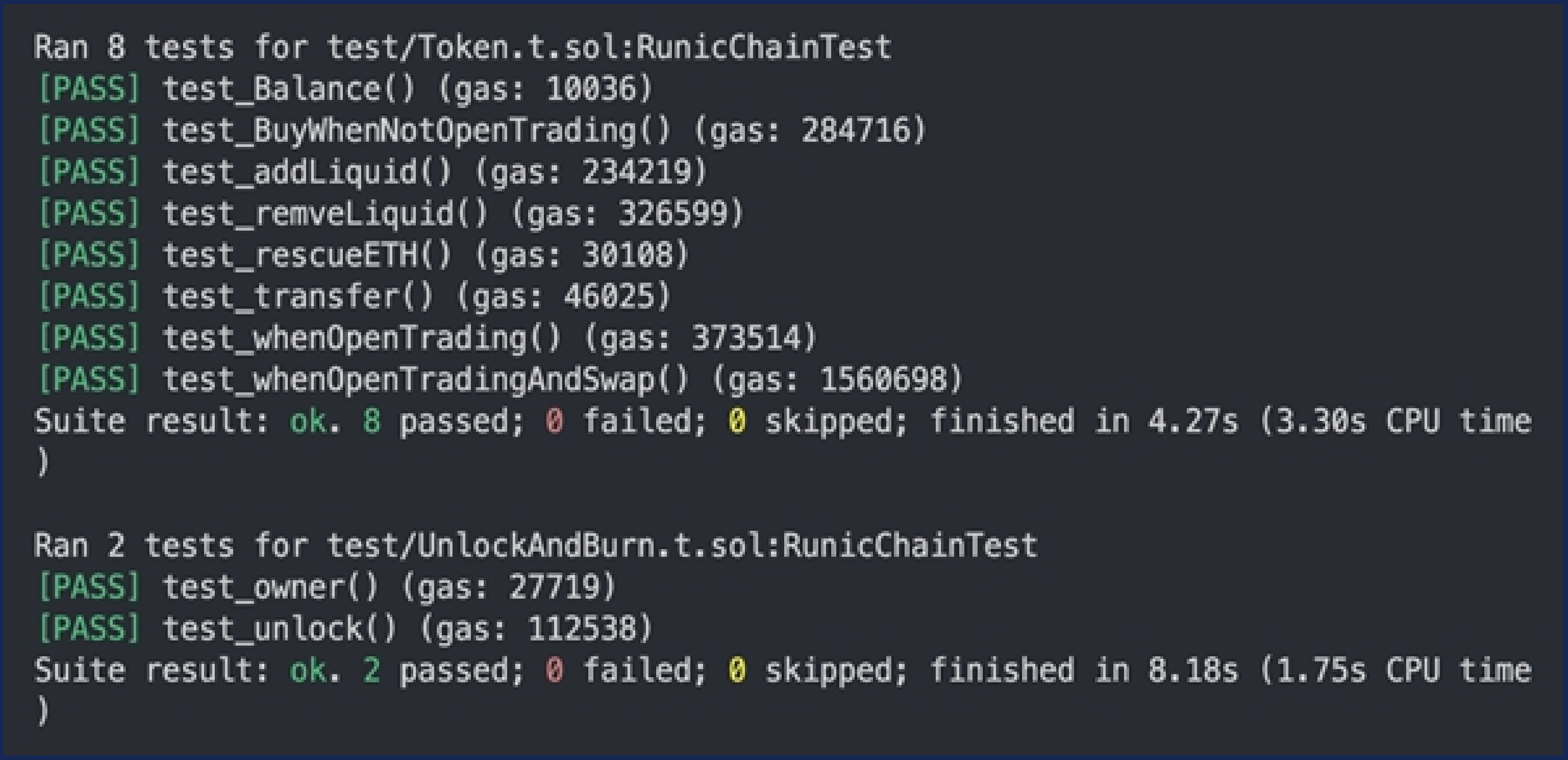
5. Gas Optimization

Have techniques been employed to optimize gas consumption (e.g., using libraries for common functions, avoiding unnecessary storage writes)?

- Yes.

Is Inline assembly adopted within the contracts for Gas Optimization?

- No.



```
Ran 8 tests for test/Token.t.sol:RunicChainTest
[PASS] test_Balance() (gas: 10036)
[PASS] test_BuyWhenNotOpenTrading() (gas: 284716)
[PASS] test_addLiquid() (gas: 234219)
[PASS] test_remveLiquid() (gas: 326599)
[PASS] test_rescueETH() (gas: 30108)
[PASS] test_transfer() (gas: 46025)
[PASS] test_whenOpenTrading() (gas: 373514)
[PASS] test_whenOpenTradingAndSwap() (gas: 1560698)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 4.27s (3.30s CPU time
)

Ran 2 tests for test/UnlockAndBurn.t.sol:RunicChainTest
[PASS] test_owner() (gas: 27719)
[PASS] test_unlock() (gas: 112538)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 8.18s (1.75s CPU time
)
```

Code Assessment Summary

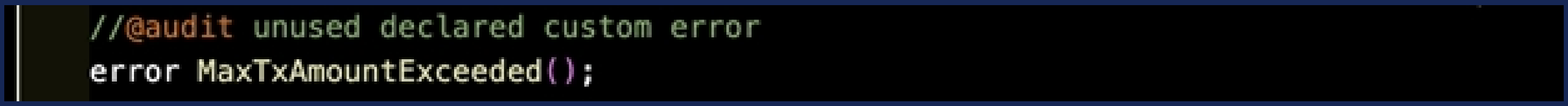
SWC ID	SWC Checklist	Test Result	Notes
SWC-100	Function Default Visibility	Not Vulnerable	Not applicable after **v0.5.X** (Currently using solidity **v >= 0.8.23**).
SWC-101	Integer Overflow and Underflow	Not Vulnerable	The issue persists in versions before v0.8.X
SWC-102	Outdated Compiler Version	Not Vulnerable	Version 0^.8.0 and above is used.
SWC-103	Floating Pragma	Not Vulnerable	Contract uses floating pragma.
SWC-104	Unchecked Call Return Value	Not Vulnerable	**call()** is used and returned value was checked.
SWC-105	Unprotected Ether Withdrawal	Not Vulnerable	Appropriate require validations are used on sensitive functions to prevent unprotected ether withdrawal.
SWC-106	Unprotected SELFDESTRUCT Instruction	Not Vulnerable	**selfdestruct()** is not used anywhere.
SWC-107	Reentrancy	Not Vulnerable	No notable functions were vulnerable to it.
SWC-108	State Variable Default Visibility	Not Vulnerable	Not Vulnerable.

SWC ID	SWC Checklist	Test Result	Notes
SWC-109	Uninitialized Storage Pointer	Not Vulnerable	Not vulnerable after compiler version, **v0.5.0**
SWC-110	Assert Violation	Not Vulnerable	Asserts are not in use.
SWC-111	Use of Deprecated Solidity Functions	Not Vulnerable	None of the deprecated functions like **block.blockhash()**, **msg.gas**, **throw**, **sha3()**, **callcode()**, **suicide()** are in use.
SWC-112	Delegatecall to Untrusted Callee	Not Vulnerable	Not Vulnerable.
SWC-113	DoS with Failed Call	Not Vulnerable	No such function was found.
SWC-114	Transaction Order Dependence	Not Vulnerable	Not Vulnerable.
SWC-115	Authorization through tx.origin	Not Vulnerable	**tx.origin** is not used anywhere in the code.
SWC-116	Block values as a proxy for time	Not Vulnerable	**Block.timestamp** is not used.
SWC-117	Signature Malleability	Not Vulnerable	Contract does not allow users to manually pass in the v, r, and s values of a signature.

SWC ID	SWC Checklist	Test Result	Notes
SWC-118	Incorrect Constructor Name	Not Vulnerable	All the constructors are created using the **constructor** keyword rather than functions.
SWC-119	Shadowing State Variables	Not Vulnerable	There are no function parameters name that shadow state variables.
SWC-120	Weak Sources of Randomness from Chain Attributes from Chain Attributes	Not Vulnerable	Random generators are not used.
SWC-121	Missing Protection against Signature Replay Attacks	Not Vulnerable	The use of chainid in relation to the signature will prevent replaying transactions in another chain.

Top Level Issues Identified

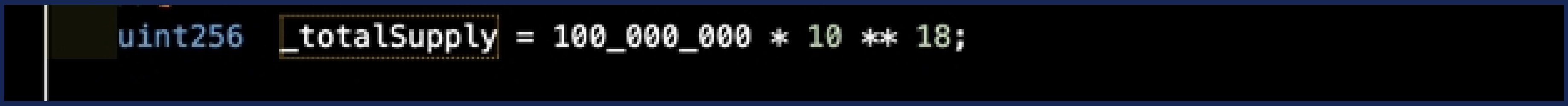
1. Unused Custom Errors



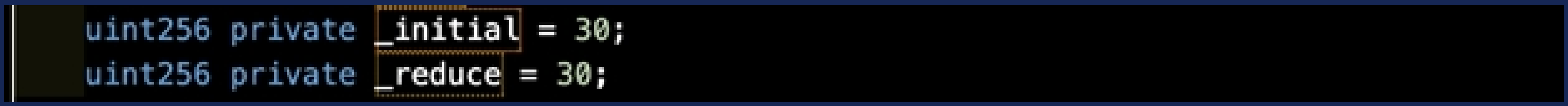
```
//@audit unused declared custom error
error MaxTxAmountExceeded();
```

The MaxTxAmountExceeded custom error was declared but it was not invoked to handle any contract error.

2. Variables certain not to be changed should be marked as constant to save gas



```
    uint256  _totalSupply = 100_000_000 * 10 ** 18;
```



```
uint256 private _initial = 30;
uint256 private _reduce = 30;
```

Few state variables were declared and called within some functions. However, to minimize gas and avoid these variables occupying some storage, these variables should be marked as constant to save gas.

3. Missing Critical State Changes

In the RunesBridge contract, a critical state variable named platformAndRewards is an address which has some permissioned function. However, there is a function to update the address yet there is no event emitted to track the changes of address after function invocation.

4. Floating Solidity Version

```
pragma solidity ^0.8.23;
```

Smart contracts with a floating solidity version are prone to either use an older or recent solidity version when deployed and this could introduce some unexpected or unknown bugs.

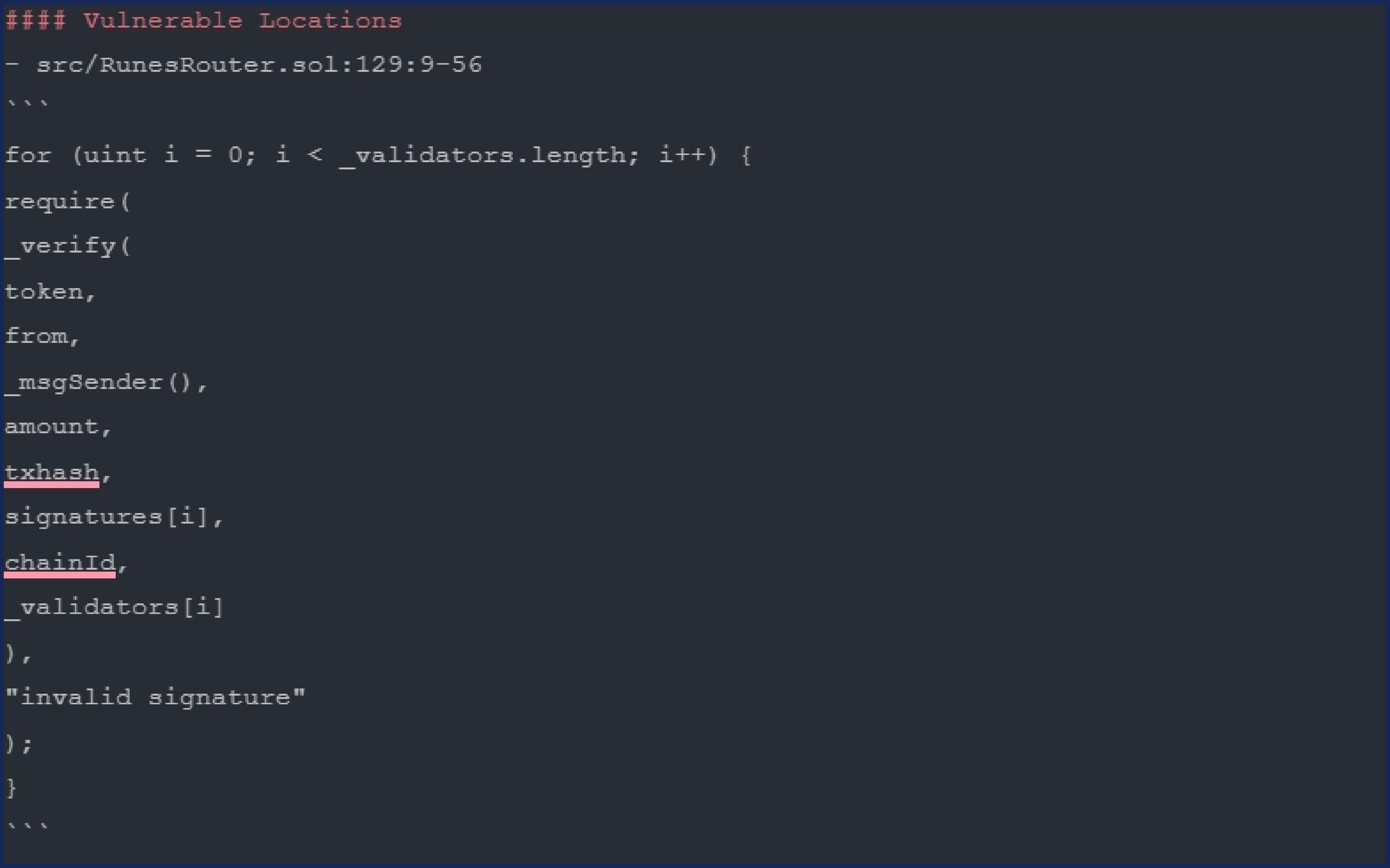
5. Missing Zero Address Checks

In Solidity, contracts often interact with external addresses. Failing to check for a possible 0 address input (especially in constructors,setters, and initializer functions) before such interactions can lead to unexpected dangerous behavior. A zero address check ensures that address are explicitly provided and not left uninitialized or set to a default, invalid state.

~~~
#### Vulnerable Locations
- src/RunesBridge.sol:174:38-57
```
address _newAddress
```
- src/RunesRouter.sol:55:9-28
```
address _validator1
```
- src/RunesRouter.sol:56:9-28
```
address _validator2
```
- src/RunesRouter.sol:57:9-28
```
address _validator3
```
~~~

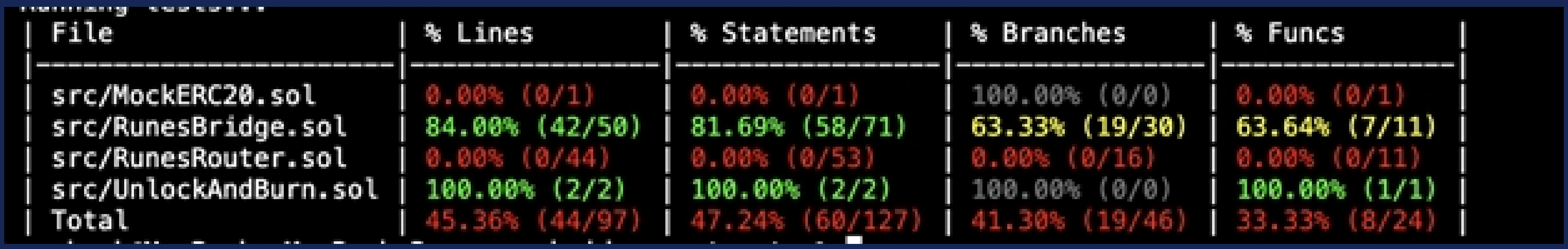
6. Unbounded Loop

Unbounded loops, specifically, for loops that can modify state and have no apparent max restriction to the number of iterations possible, can lead to excessive gas consumption, which may cause transactions to fail or become prohibitively expensive.



~~~~
#### Vulnerable Locations
- src/RunesRouter.sol:129:9-56
```
for (uint i = 0; i < _validators.length; i++) {
require(
_verify(
token,
from,
_msgSender(),
amount,
txhash,
signatures[i],
chainId,
_validators[i]
),
"invalid signature"
);
}
```
~~~~

Codebase Test Coverage



```
| File                    | % Lines         | % Statements     | % Branches      | % Funcs        |
|-------------------------|-----------------|------------------|-----------------|----------------|
| src/MockERC20.sol       | 0.00% (0/1)     | 0.00% (0/1)      | 100.00% (0/0)   | 0.00% (0/1)    |
| src/RunesBridge.sol     | 84.00% (42/50)  | 81.69% (58/71)   | 63.33% (19/30)  | 63.64% (7/11)  |
| src/RunesRouter.sol     | 0.00% (0/44)    | 0.00% (0/53)     | 0.00% (0/16)    | 0.00% (0/11)   |
| src/UnlockAndBurn.sol   | 100.00% (2/2)   | 100.00% (2/2)    | 100.00% (0/0)   | 100.00% (1/1)  |
| Total                   | 45.36% (44/97)  | 47.24% (60/127)  | 41.30% (19/46)  | 33.33% (8/24)  |
```

Social Engineering Checks

Discord Phishing

- Not Present.

Checking web page source code in browser developer mode

- Works for both Environment [Mobile/Desktop].

Suspicious links on Linktree

- No Linktree present.

Web Spoofing / Phishing

- No suspicious link spotted so far on search engines.

Domain Name and Subdomains

- runesbridge.xyz

Subdomains:

- https://bridge-testnet.runesbridge.xyz
- https://r-testnet.runesbridge.xyz
- https://app.runesbridge.xyz
- https://api-dev.runesbridge.xyz

- https://runes.runesbridge.xyz
- https://www.runesbridge.xyz
- https://api-bridge.runesbridge.xyz
- https://docs.runesbridge.xyz
- https://r.runesbridge.xyz
- https://api-bridge-testnet.runesbridge.xyz
- https://testnet.runesbridge.xyz
- https://runes-testnet.runesbridge.xyz
- https://app-dev.runesbridge.xyz

Domain Certificate

- Subject	:	runesbridge.xyz
SAN	:	runesbridge.xyz
Valid from	:	Saturday, 6th April 2024, 02:57:57 GMT
Valid until	:	Friday, 5th July 2024, 02:57:56 GMT
Issuer	:	R3

Telegram Phishing

- 2 TG pages	:	https://t.me/RunesBridge
		https://t.me/RunesBridgeNews

Checking the remotely loaded JavaScript script in the web page source code

- No Remote JS from Malicious source

Server

- Vercel

Login Panel Disclosed

- http://api-dev.runesbridge.xyz:81/login

Readiness Assessment

Based on the evaluation of the aforementioned criteria, this report concludes that the RunesBridge smart contracts:

- ✅ Ready for a full-fledged security audit.
- 🟨 Partially ready for a security audit. Further work is needed on [Specify areas for improvement].
- 🟥 Not yet ready for a security audit. Significant development and testing are required.

Disclaimer

RunesBridge report presents the findings of a lite technical due diligence conducted on the provided codebase. It is intended to identify potential areas for improvement in coding practices and security but does not constitute a comprehensive security audit.

While this review has focused on best practices and secure coding principles, it is important to acknowledge that vulnerabilities may still exist within the codebase. A more thorough security audit is strongly recommended before deploying the code to a production environment (mainnet).

QuillAudits shall not be held liable for any security breaches or vulnerabilities that may be discovered after the completion of this lite technical due diligence.

About QuillAudits

QuillAudits is a leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers. With six years of expertise, we've secured over 1000 projects globally, averting over $30 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

1000+
Audits Completed

$30B
Secured

1M
Lines of Code Audited

Follow Our Journey

Lite Technical Assessment

For

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com