Approved

SMART CONTRACT
FREE AI-BASED AUDIT

ChartIQ

Scan and check this report
was posted at Approved Github

June, 2024

Website: Approved.ltd

Table of Contents

Disclaimer

This is a comprehensive report based on our automated and manual examination of cybersecurity vulnerabilities and framework flaws of the project's smart contract.

Reading the full analysis report is essential to build your understanding of project's security level. It is crucial to take note, though we have done our best to perform this analysis and report, that you should not rely on the our research and cannot claim what it states or how we created it.

Before making any judgments, you have to conduct your own independent research.

We will discuss this in more depth in the following disclaimer - please read it fully.

DISCLAIMER: You agree to the terms of this disclaimer by reading this report or any portion thereof. Please stop reading this report and remove and delete any copies of this report that you download and/or print if you do not agree to these conditions. Scan and verify report's presence in the GitHub repository by a qr-code at the title page. This report is for non-reliability information only and does not represent investment advice. No one shall be entitled to depend on the report or its contents, and Approved and its affiliates shall not be held responsible to you or anyone else, nor shall Approved provide any guarantee or representation to any person with regard to the accuracy or integrity of the report.

Without any terms, warranties or other conditions other than as set forth in that exclusion and Approved excludes hereby all representations, warrants, conditions and other terms (including, without limitation, guarantees implied by the law of satisfactory quality, fitness for purposes and the use of reasonable care and skills).

The report is provided as "as is" and does not contain any terms and conditions. Except as legally banned, Approved disclaims all responsibility and responsibilities and no claim against Approved is made to any amount or type of loss or damages (without limitation, direct, indirect, special, punitive, consequential or pure economic loses or losses) that may be caused by you or any other person, or any damages or damages, including without limitations (whether innocent or negligent).

Security analysis is based only on the smart contracts. No applications or operations were reviewed for security. No product code has been reviewed.

Procedure

Our analysis contains following steps:

1. Project Analysis;

2. Unit Testing:
    - Smart contract functions will be unit tested on multiple parameters and under multiple conditions to ensure that all paths of functions are functioning as intended.
    - In this phase intended behaviour of smart contract is verified.
    - In this phase, we would also ensure that smart contract functions are not consuming unnecessary gas.
    - Gas limits of functions will be verified in this stage.

3. Automated Testing:
    - Mythril
    - Oyente
    - Manticore
    - Solgraph

4. Testing code with artificial intelligence

Terminology

We categorize the finding into 4 categories based on their vulnerability:

- Low-severity issue — less important, must be analyzed
- Medium-severity issue — important, needs to be analyzed and fixed
- High-severity issue —important, might cause vulnerabilities, must be analyzed and fixed
- Critical-severity issue —serious bug causes, must be analyzed and fixed.

Limitations

The security audit of Smart Contract cannot cover all vulnerabilities. Even if no vulnerabilities are detected in the audit, there is no guarantee that future smart contracts are safe. Smart contracts are in most cases safeguarded against specific sorts of attacks. In order to find as many flaws as possible, we carried out a comprehensive smart contract audit. Audit is a document that is not legally binding and guarantees nothing.

Basic Security Recommendation

Unlike hardware and paper wallets, hot wallets are connected to the internet and store private keys online, which exposes them to greater risk. If a company or an individual holds significant amounts of cryptocurrency in a hot wallet, they should consider using MultiSig addresses. Wallet security is enhanced when private keys are stored in different locations and are not controlled by a single entity.

Token Contract Details for 07.06.2024

Contract Name: **ChartIQ**

Deployed address: **0x301739698100845693116d8ae7c0df9b5d07e1e2**

Total Supply: **1,000,000**

Token Tracker: **ChartIQ**

Token holders: **1**

Transactions count: **1**

Top 100 holders dominance: **100.00%**

Audit Details

Project Name: **ChartIQ**

Language: **Solidity**

Compiler Version: **v0.8.26**

Blockchain: **Etherium**

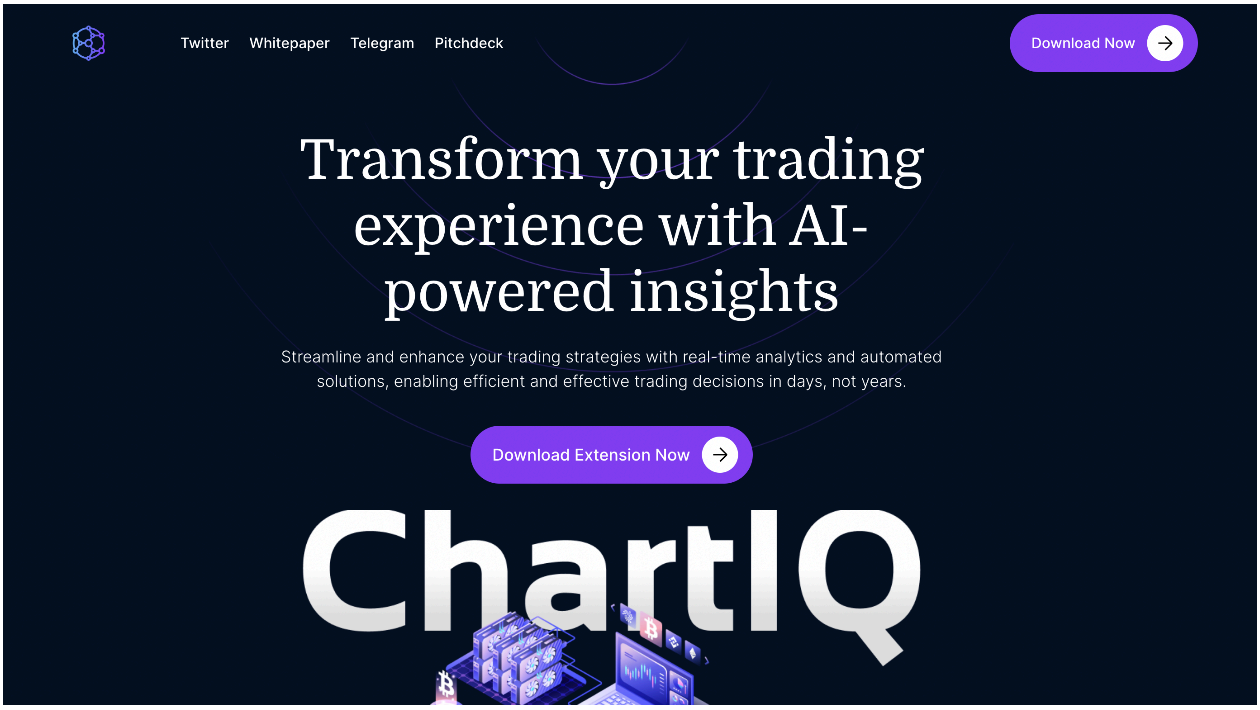
Social Profiles

Project Website: **https://www.chartiq.ai/**

Project Twitter: **https://x.com/chartiqglobal**

Project Telegram: **https://t.me/chartiqglobal**

Project Website Overview



✓JavaScript errors hasn’t been found.
✓Malware pop-up windows hasn’t been detected.
✓No issues with loading elements, code, or stylesheets.

Vulnerabilities checking

Issue Description	Checking Status
Compiler Errors	Completed
Delays in Data Delivery	Completed
Re-entrancy	Completed
Transaction-Ordering Dependence	Completed
Timestamp Dependence	Completed
Shadowing State Variables	Completed
DoS with Failed Call	Completed
DoS with Block Gas Limit	Completed
Outdated Complier Version	Completed
Assert Violation	Completed
Use of Deprecated Solidity Functions	Completed
Integer Overflow and Underflow	Completed
Function Default Visibility	Completed
Malicious Event Log	Completed
Math Accuracy	Completed
Design Logic	Completed
Fallback Function Security	Completed
Cross-function Race Conditions	Completed
Safe Zeppelin Module	Completed

Security Issues

1) Issue Type: **REENTRANCY**

Severity: **High**

```
367     function clearStuckToken(address tokenAddress, uint256 tokens) external authorized returns
    (bool success) {
368         require(tokenAddress != address(this),"Cannot withdraw native token");
369         if(tokenAddress == pair){
370             require(block.timestamp > launchedAt + 500 days,"Locked for 1 year");
371         }
372
373         if(tokens == 0){
374             tokens = BEP20(tokenAddress).balanceOf(address(this));
375         }
376
377         emit clearToken(tokenAddress, tokens);
378
379         return BEP20(tokenAddress).transfer(msg.sender, tokens);
380     }
```

L367 - L380

Description:

In a Re-entrancy attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls.

2) Issue Type: **STRICT EQUALITY CHECK IN BLOCK.TIMESTAMP**

Severity: **Medium**

```
369            if(tokenAddress == pair){
370                require(block.timestamp > launchedAt + 500 days,"Locked for 1 year");
371            }
372
```

L370 - L370

Description:

Contracts require time values for certain functions, but block.timestamp or now are not precise enough for most use cases, especially strict equality checks.
Ethereum's decentralized nature means nodes can only synchronize time to a certain extent, and malicious miners can manipulate timestamps.
Developers can't rely on the accuracy of the timestamps provided.

3) Issue Type: **HARDCODED SLIPPAGE FOR UNISWAP**

Severity: **Medium**

```
404
405            router.swapExactTokensForETHSupportingFeeOnTransferTokens(
406                amountToSwap,
407                0,
408                path,
409                address(this),
410                block.timestamp
411            );
412
```

L405 - L411

Description:

This contract was found to be using hardcoded slippage. This can lead to a sandwich attack where the attacker can track market orders and replicate them whenever the order amount to be executed is large enough to compensate for exchange manipulation costs.

3) Issue Type: **PRECISION LOSS DURING DIVISION BY LARGE NUMBERS**

Severity: **Medium**

```
40          uint256 c = a * b;
41          require(c / a == b, "SafeMath: multiplication overflow");
42
```

L41

```
50          uint256 c = a / b;
51          return c;
```

L50

```
397
398             uint256 amountToLiquify = (swapThreshold * liquidityFee)/(totalETHFee * 2);
399             uint256 amountToSwap = swapThreshold - amountToLiquify;
400
```

L398

```
420
421             uint256 amountETHLiquidity = (amountETH * liquidityFee) / (totalETHFee * 2);
422             uint256 amountETHMarketing = (amountETH * marketingFee) / totalETHFee;
423             uint256 amountETHbuyback = (amountETH * buybackFee) / totalETHFee;
```

L421

```
421             uint256 amountETHLiquidity = (amountETH * liquidityFee) / (totalETHFee * 2);
422             uint256 amountETHMarketing = (amountETH * marketingFee) / totalETHFee;
423             uint256 amountETHbuyback = (amountETH * buybackFee) / totalETHFee;
```

L422

```
422             uint256 amountETHMarketing = (amountETH * marketingFee) / totalETHFee;
423             uint256 amountETHbuyback = (amountETH * buybackFee) / totalETHFee;
424
```

L423

Description:

In Solidity, when dividing large numbers, precision loss can occur due to limitations in the Ethereum Virtual Machine (EVM). Solidity lacks native support for decimal or fractional numbers, leading to truncation of division results to integers. This can result in inaccuracies or unexpected behaviors, especially when the numerator is not significantly larger than the denominator.

3) Issue Type: **EVENT BASED REENTRANCY**

Severity: **Low**

```
367     function clearStuckToken(address tokenAddress, uint256 tokens) external authorized returns
    (bool success) {
368         require(tokenAddress != address(this),"Cannot withdraw native token");
369         if(tokenAddress == pair){
370             require(block.timestamp > launchedAt + 500 days,"Locked for 1 year");
371         }
372
373         if(tokens == 0){
374             tokens = BEP20(tokenAddress).balanceOf(address(this));
375         }
376
377         emit clearToken(tokenAddress, tokens);
378
379         return BEP20(tokenAddress).transfer(msg.sender, tokens);
380     }
381
```

L367 - L380

Description:

In a Re-entrancy attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls. In the case of event-based Re-entrancy attacks, events are emitted after an external call leading to missing event calls.

Conclusion for project owner

High, Medium and Low-severity issues exist within smart contracts.

NOTE: Please check the disclaimer above and note, that the audit makes no statements or warranties on the business model, investment attractiveness, or code sustainability. Contract security report for community

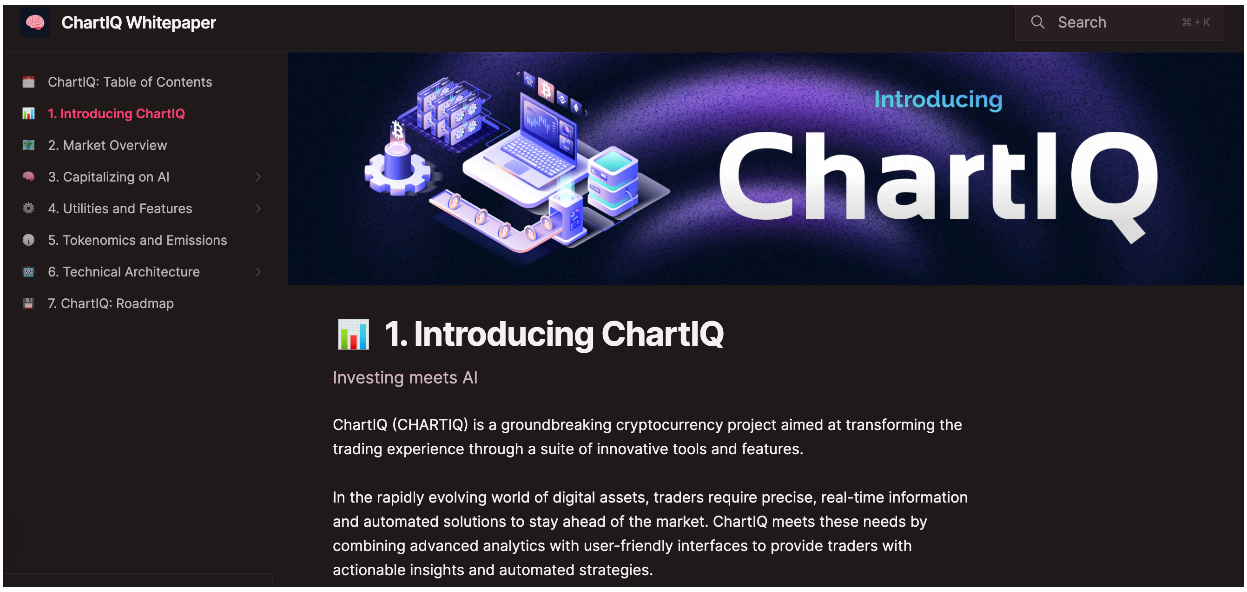
SECURITY REPORT
FOR COMMUNITY

ChartIQ

Approved

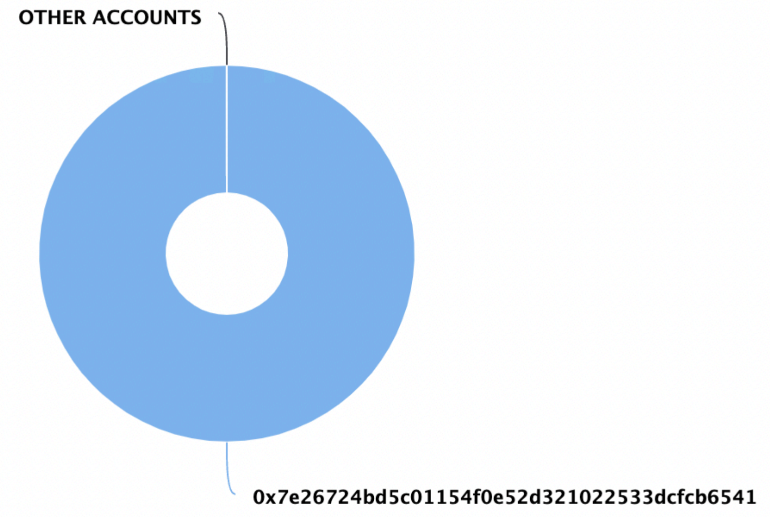
Whitepaper of the project

The whitepaper of ChartIQ project has been verified on behalf of Approved team.



Whitepaper link: **https://chartiq-whitepaper.gitbook.io/chartiq-whitepaper/1.-introducing-chartiq**

ChartIQ Token Distribution



ChartIQ Top 10 Holders

Rank	Address	Quantity (Token)	Percentage
1	0x7E26724b...DcFCB6541	1,000,000	100.0000%

Approved Contact Info

Website: https://approved.ltd

Telegram: @team_approved

GitHub: https://github.com/Approved-Audits/smart_contracts