CredShields

Smart Contract Audit

Nov 23rd, 2023 • CONFIDENTIAL

Description

This document details the process and result of the Quest Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of DappRadar between Oct 16th, 2023, and Nov 1st, 2023. A retest was performed on Nov 22nd, 2023.

Author

Shashank (Co-founder, CredShields)

shashank@CredShields.com

Reviewers

Aditya Dixit (Research Team Lead)

aditya@CredShields.com

Prepared for

DappRadar

Table of Contents

CRED SHIELDS

1. Executive Summary

DappRadar engaged CredShields to perform a smart contract audit from Oct 16th, 2023, to Nov 1st, 2023. During this timeframe, Nine (9) vulnerabilities were identified. **A retest was performed on Nov 22nd, 2023, and all the bugs have been addressed.**

During the audit, Zero (0) vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "DappRadar" and should be prioritized for remediation, and fortunately, none were found.

The table below shows the in-scope assets and a breakdown of findings by severity per asset. Section 2.3 contains more information on how severity is calculated.

Assets in Scope	Critical	High	Medium	Low	info	Gas	Σ
Quest Smart Contract	0	0	2	1	2	4	**9**
	0	**0**	**2**	**1**	**2**	**4**	**9**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in Quest Smart Contract's scope during the testing window while abiding by the policies set forth by DappRader team.

State of Security

To maintain a robust security posture, it is essential to continuously review and improve upon current security processes. Utilizing CredShields' continuous audit feature allows both DappRadar's internal security and development teams to not only identify specific vulnerabilities, but also gain a deeper understanding of the current security threat landscape.

To ensure that vulnerabilities are not introduced when new features are added, or code is refactored, we recommend conducting regular security assessments. Additionally, by analyzing the root cause of resolved vulnerabilities, the internal teams at DappRadar can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, DappRadar can future-proof its security posture and protect its assets.

2. Methodology

DappRadar engaged CredShields to perform a DappRadar Smart Contract audit. The following sections cover how the engagement was put together and executed.

2.1 Preparation phase

The CredShields team meticulously reviewed all provided documents and comments in the smart-contract code to gain a thorough understanding of the contract's features and functionalities. They meticulously examined all functions and created a mind map to systematically identify potential security vulnerabilities, prioritizing those that were more critical and business-sensitive for the refactored code. To confirm their findings, the team deployed a self-hosted version of the smart contract and performed verifications and validations during the audit phase.

A testing window from Oct 16th, 2023, to Nov 1st, 2023, was agreed upon during the preparation phase.

2.1.1 Scope

During the preparation phase, the following scope for the engagement was agreed upon:

IN SCOPE ASSETS
https://github.com/dappradar/quest-smart-contract/tree/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts

Table: List of Files in Scope

2.1.2 Documentation

Documentation was not required as the code was self-sufficient for understanding the project.

2.1.3 Audit Goals

CredShields uses both in-house tools and manual methods for comprehensive smart contract security auditing. The majority of the audit is done by manually reviewing the contract source code, following SWC registry standards, and an extended industry standard self-developed checklist. The team places emphasis on understanding core concepts, preparing test cases, and evaluating business logic for potential vulnerabilities.

2.2 Retesting phase

DappRadar is actively partnering with CredShields to validate the remediations implemented towards the discovered vulnerabilities.

2.3 Vulnerability Classification and Severity

CredShields follows OWASP's Risk Rating Methodology to determine the risk associated with discovered vulnerabilities. This approach considers two factors - Likelihood and Impact - which are evaluated with three possible values - **Low**, **Medium**, and **High**, based on factors such as Threat agents, Vulnerability factors, Technical and Business Impacts. The overall severity of the risk is calculated by combining the likelihood and impact estimates.

Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

Overall, the categories can be defined as described below -

1. **Informational**

   We prioritize technical excellence and pay attention to detail in our coding practices. Our guidelines, standards, and best practices help ensure software stability and reliability. Informational vulnerabilities are opportunities for improvement and do

not pose a direct risk to the contract. Code maintainers should use their own judgment on whether to address them.

2. Low

Low-risk vulnerabilities are those that either have a small impact or can't be exploited repeatedly or those the client considers insignificant based on their specific business circumstances.

3. Medium

Medium-severity vulnerabilities are those caused by weak or flawed logic in the code and can lead to exfiltration or modification of private user information. These vulnerabilities can harm the client's reputation under certain conditions and should be fixed within a specified timeframe.

4. High

High-severity vulnerabilities pose a significant risk to the Smart Contract and the organization. They can result in the loss of funds for some users, may or may not require specific conditions, and are more complex to exploit. These vulnerabilities can harm the client's reputation and should be fixed immediately.

5. Critical

Critical issues are directly exploitable bugs or security vulnerabilities that do not require specific conditions. They often result in the loss of funds and Ether from Smart Contracts or users and put sensitive user information at risk of compromise

or modification. The client's reputation and financial stability will be severely impacted if these issues are not addressed immediately.

6. **Gas**

   To address the risk and volatility of smart contracts and the use of gas as a method of payment, CredShields has introduced a "Gas" severity category. This category deals with optimizing code and refactoring to conserve gas.

2.4 CredShields staff

The following individual at CredShields managed this engagement and produced this report:

- **Shashank, Co-founder CredShields**
    - shashank@CredShields.com

Please feel free to contact this individual with any questions or concerns you have around the engagement or this document.

3. Findings

This chapter contains the results of the security assessment. Findings are sorted by their severity and grouped by the asset and SWC classification. Each asset section will include a summary. The table in the executive summary contains the total number of identified security vulnerabilities per asset per risk indication.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, Nine (9) security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Accounting Issue with Winner Rewards	Medium	Business Logic
Handling ERC-20 Tokens with Fees On Transfer	Medium	Business Logic
Lack of ERC-721 Accounting in claimReward and RemoveWinner Functions	Informational	Business Logic
Outdated Pragma version	Low	Outdated Pragma
Gas Optimization in Increments	Gas	Gas optimization
Missing NatSpec Comments	Informational	Missing best practices

Custom Errors instead of Revert	Gas	Gas Optimization
Gas Optimization in Require/Revert Statements	Gas	Gas Optimization
Array Length Caching	Gas	Gas Optimization

Table: Findings in Smart Contracts

3.1.2 Findings 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 >= 0.8.6)
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 not used
SWC-105	Unprotected Ether Withdrawal	Not Vulnerable	Appropriate function modifiers and require validations are used on sensitive functions that allow token or 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-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	Not used anywhere
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	Not applicable as this won't work during compile time after version 0.6.0
SWC-120	Weak Sources of Randomness from Chain Attributes	Not Vulnerable	Random generators are not used.
SWC-121	Missing Protection against Signature Replay Attacks	Not Vulnerable	No such scenario was found

SWC-122	Lack of Proper Signature Verification	Not Vulnerable	Not used anywhere
SWC-123	Requirement Violation	Not Vulnerable	Not vulnerable
SWC-124	Write to Arbitrary Storage Location	Not Vulnerable	No such scenario was found
SWC-125	Incorrect Inheritance Order	Not Vulnerable	No such scenario was found
SWC-126	Insufficient Gas Griefing	Not Vulnerable	No such scenario was found
SWC-127	Arbitrary Jump with Function Type Variable	Not Vulnerable	Jump is not used.
SWC-128	DoS With Block Gas Limit	Not Vulnerable	Not Vulnerable.
SWC-129	Typographical Error	Not Vulnerable	No such scenario was found
SWC-130	Right-To-Left-Override control character (U+202E)	Not Vulnerable	No such scenario was found
SWC-131	Presence of unused variables	Not Vulnerable	No such scenario was found
SWC-132	Unexpected Ether balance	Not Vulnerable	No such scenario was found
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Not Vulnerable	abi.encodePacked() or other functions are not used.
SWC-134	Message call with hardcoded gas amount	Not Vulnerable	Not used anywhere in the code
SWC-135	Code With No Effects	Not Vulnerable	No such scenario was found
SWC-136	Unencrypted Private Data On-Chain	Not Vulnerable	No such scenario was found

4. Remediation Status

DappRadar is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on Nov 22nd, 2023, and all the issues have been addressed.**

Also, the table shows the remediation status of each finding.

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Accounting Issue with Winner Rewards	Medium	**Fixed [12/11/2023]**
Handling ERC-20 Tokens with Fees On Transfer	Medium	**Fixed [12/11/2023]**
Lack of ERC-721 Accounting in claimReward and RemoveWinner Functions	Informational	**Fixed [12/11/2023]**
Outdated Pragma version	Low	**Fixed [12/11/2023]**
Gas Optimization in Increments	Gas	**Fixed [12/11/2023]**
Missing NatSpec Comments	Informational	**Fixed [12/11/2023]**
Custom Errors instead of Revert	Gas	**Fixed [12/11/2023]**
Gas Optimization in Require/Revert Statements	Gas	**Fixed [12/11/2023]**

Array Length Caching	Gas	**Fixed [12/11/2023]**

Table: Summary of findings and status of remediation

5. Bug Reports

Bug ID #1[Fixed]

Accounting Issue with Winner Rewards

Vulnerability Type
Accounting Issue

Severity
Medium

Description
The contract includes functions `setWinnerReward()` and `removeWinner()` for handling rewards for winners. The `setWinnerReward()` function is used to set the rewards for winners and increases the `totalRewardAmount`. However, the `removeWinner()` function is intended to remove rewards for a winner, but it also increases the `totalRewardAmount`.

This leads to an accounting issue in the contract. In normal operation, when a winner's rewards are removed, the `totalRewardAmount` should decrease by the amount of the rewards removed. However, due to the incorrect behavior of `removeWinner()`, the `totalRewardAmount` is increasing when rewards are removed. This can lead to inaccurate accounting and reporting of the total reward amount in the contract.

Affected Code

- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L194-L196
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L197-L199

Impacts

The accounting issue caused by the incorrect behavior of removeWinner() may lead to inaccurate reporting of the total reward amount in the contract. While this issue does not directly result in the loss of funds, it can affect the transparency and trustworthiness of the contract's reward distribution system.

Remediation

To address this accounting issue, it is recommended to modify the `removeWinner()` function to decrease the `totalRewardAmount` by the amount of rewards being removed. This ensures that the total reward amount accurately reflects the remaining rewards in the contract. Regularly audit and test the contract's functionality to identify and address such accounting issues and maintain the integrity of the reward distribution system.

Retest

This issue has been fixed by removing totalRewardAmount & totalERC1155Amount variables in the contract.

Bug ID #2 [Fixed]

Handling ERC-20 Tokens with Fees On Transfer

Vulnerability Type
Accounting Error

Severity
Medium

Description
The smart contracts use ERC-20 tokens for transfers but do not account for cases where some of these ERC-20 tokens charge fees on transfers. When tokens charge fees on transfers, the actual received amount by the recipient may be less than the transferred amount. This can result in accounting errors and potential financial losses for the protocol.

Affected Code

- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L144-L174

Impacts
In cases where transfer fees are charged but not considered, the protocol may incur financial losses, as it may distribute more tokens than it holds. The contract may incorrectly account for the transferred amounts, leading to discrepancies between recorded and actual balances.

Remediation
To address this issue, it is recommended to modify the contract to consider the possibility of transfer fees for ERC-20 tokens. This can be done by implementing transfer handling mechanisms that account for fees, either by querying the token's balance after a transfer or by using token-specific functions designed to handle transfers with fees.

Retest
This has been taken care of by internal and UI validations.

Bug ID #3 [Fixed]

Lack of ERC-721 Accounting in claimReward and RemoveWinner Functions

Vulnerability Type
Accounting Error

Severity
Informational

Description
The smart contracts have implemented functions for claiming and removing rewards, which involve ERC-20, ERC-1155, and ERC-721 tokens. In the claimReward function, when rewards are claimed, accounting and transfers are performed for ERC-20 and ERC-1155 tokens, but there is no accounting for ERC-721 tokens. Similarly, in the removeWinner function, accounting is only done for ERC-20 and ERC-1155 tokens, leaving out ERC-721 tokens. This can lead to inconsistencies in data and accounting, potentially causing issues with the ERC-721 tokens' rewards.

Affected Code
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L144-L174
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L192-L203

Impacts
The lack of accounting can make it challenging to audit and understand the distribution of ERC-721 tokens, potentially leading to a loss of transparency. Failure to account for ERC-721 tokens can lead to inconsistencies in the total reward amounts and amounts held by users.

Remediation

To address this issue, it is recommended to implement accounting and transfer logic for ERC-721 tokens in the claimReward and removeWinner functions. Ensure that the total amounts of ERC-721 tokens are accurately updated and transferred when rewards are claimed or removed.

Retest

Contract used to account total tokens using totalRewardAmount & totalERC1155Amount For ERC20 & ERC1155 but not for ERC721. This issue has been fixed by removing both variables.

Bug ID #4 [Fixed]

Outdated Pragma version

Vulnerability Type
Outdated Pragma

Severity
Low

Description
Using an outdated compiler version can be problematic, especially if there are publicly disclosed bugs and issues that affect the current compiler version.
The contracts found in the repository were allowing an old compiler version to be used, i.e., 0.8.18.

Affected Code

- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L2
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/AssetsHolder.sol#L2

Impacts
If the smart contract gets compiled and deployed with an older or too recent version of the solidity compiler, there's a chance that it may get compromised due to the bugs present in the older versions or unidentified exploits in the new versions.
Incompatibility issues may also arise if the contract code does not support features in other compiler versions, therefore, breaking the logic.
The likelihood of exploitation is really low therefore this is only Low severity.

Remediation
Keep the compiler versions updated in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.21 pragma version which is stable and not too recent.

Reference: https://swcregistry.io/docs/SWC-103

Retest

The pragma version has been updated to 0.8.21.

Bug ID #5 [Fixed]

Gas Optimization in Increments

Vulnerability Type
Gas optimization

Severity
Gas optimization

Description
The contract uses **for** loops that use post increments for the variable "**i**". The contract can save some gas by changing this to **++i**.
++i costs less gas compared to **i++** or **i += 1** for unsigned integers. In **i++**, the compiler has to create a temporary variable to store the initial value. This is not the case with **++i** in which the value is directly incremented and returned, thus, making it a cheaper alternative.

Vulnerable Code

- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L63
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L67
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L71
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L101
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L108
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L114
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L194
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L197

Impacts
Using **i++** instead of **++i** costs the contract deployment around 600 more gas units.

Remediation
It is recommended to switch to **++i** and change the code accordingly so the function logic remains the same and saves some gas.
If you use pragma 0.8.22 this will automatically be fixed.

Retest
Inside setWinnerRewards() its still using **i++**

Bug ID #6 [Fixed]

Missing NatSpec Comments

Vulnerability Type
Missing best practices

Severity
Informational

Description:
Solidity contracts use a special form of comments to document code. This special form is named the Ethereum Natural Language Specification Format (NatSpec).
The document is divided into descriptions for developers and end-users along with the title and the author.
The contracts in the scope were missing these comments.

Affected Code

- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/AssetsHolder.sol

Impacts:
Without Natspec comments, it can be challenging for other developers to understand the code's intended behavior and purpose. This can lead to errors or bugs in the code, making it difficult to maintain and update the codebase. Additionally, it can make it harder for auditors to evaluate the code for security vulnerabilities, increasing the risk of potential exploits.

Remediation:
Developers should review their codebase and add Natspec comments to all relevant functions, variables, and events. Natspec comments should include a description of the function or event, its parameters, and its return values.

Retest

NatSpec comments have been added to QuestsDappRadar but not in AssetsHolder.

Bug ID#7 [Fixed]

Custom Errors instead of Revert

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The contract was found to be using a revert() statement. Since Solidity v0.8.4, custom errors have been introduced which are a better alternative to the revert.
This allows the developers to pass custom errors with dynamic data while reverting the transaction and also makes the whole implementation a bit cheaper than using revert.

Vulnerable Code

- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L210
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L218
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L227

Impacts
Using revert() instead of error() costs more gas.

Remediation
It is recommended to replace the instances of revert() statements with error() to save gas.

Retest
The contracts are not using revert statements anymore.

Bug ID#8 [Fixed]

Gas Optimization in Require/Revert Statements

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The **require()** statement takes an input string to show errors if the validation fails.
The strings inside these functions that are longer than **32 bytes** require at least one additional MSTORE, along with additional overhead for computing memory offset and other parameters. For this purpose, having strings lesser than 32 bytes saves a significant amount of gas.

Vulnerable Code

- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L60
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L210
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L218
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L227

Impacts
Having longer require/revert strings than 32 bytes costs a significant amount of gas.

Remediation
It is recommended to shorten the strings passed inside **require()** statements to fit under **32 bytes**. This will decrease the gas usage at the time of deployment and at runtime when the validation condition is met.

Retest
This is fixed as the strings have been shortened.

Bug ID #9 [Fixed]

Array Length Caching

Vulnerability Type
Gas Optimization

Severity
Gas

Description
During each iteration of the loop, reading the length of the array uses more gas than is necessary. In the most favorable scenario, in which the length is read from a memory variable, storing the array length in the stack can save about 3 gas per iteration. In the least favorable scenario, in which external calls are made during each iteration, the amount of gas wasted can be significant.

Affected Code

- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L63-L65
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L67-L69
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L71-L73
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L101-L106
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L108-L112
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L114-L119
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L135-L138
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L154-L159

- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L161-L165
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L167-L172
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L194-L196
- https://github.com/dappradar/quest-smart-contract/blob/dae00d8e4a377f9f758cf58ca50c61bbd924e081/contracts/QuestsDappRadar.sol#L197-L199

Impacts

Reading the length of an array multiple times in a loop by calling .length costs more gas.

Remediation

Consider storing the array length of the variable before the loop and use the stored length instead of fetching it in each iteration.

Retest

The length variable is now being cached and used inside loops to optimize gas.

6. Disclosure

The Reports provided by CredShields is not an endorsement or condemnation of any specific project or team and do not guarantee the security of any specific project. The contents of this report are not intended to be used to make decisions about buying or selling tokens, products, services, or any other assets and should not be interpreted as such.

Emerging technologies such as Smart Contracts and Solidity carry a high level of technical risk and uncertainty. CredShields does not provide any warranty or representation about the quality of code, the business model or the proprietors of any such business model, or the legal compliance of any business. The report is not intended to be used as investment advice and should not be relied upon as such.

CredShields Audit team is not responsible for any decisions or actions taken by any third party based on the report.