ARtQuest

Security Audit Report

Agile Project 2023

Dublin Group

Kristiania

Table of Contents

Summary 3
Methodology 3
SCOPE 4
Executive Summary 4
Findings Overview 5
Review of the Testing 6
Technical details of each discovery 8
1. App using cleartext protocol HTTP– Critical 8
3. JavaScript heap out of memory attack against the frontend server – Critical. 12
4. Outdatet modules in the Frontend server - Medium 14
5. Hardcoded sensitive data - Medium 16
6. App can write to app directory – Information. 18
7. App logs Information Are stored – Information. 19
8. Defining Methods 20

Summary

An internal audit was conducted by members of the team to solidify the development of the application, identify the weaknesses of the application. This test lasted approx. 4 hours. It was conducted on the 08 of June 2023.

Methodology

The application is a mobile application prototype that consist in a backend server with a database communicating with the frontend through an Api. The Application lets a user register an account and that account gets stored in the database. The main functionality of the application is to use the camera to collect “items” when scanning a QR code. The objects also get stored in the database.

The audit was short because of the nature of the Sprint. Testers had full access to both Server and Frontend Server. Testers had full access to all the source code, passwords, databases. Testers had access to IP addresses.

The goal was to find the most misconfigurations, weaknesses to then gather all, present them in with a scale from **Critical** – **Medium** – **Low** – **Info**.

Testers then has gathered flaws in the application, and they will conclude with assessing the team of which bugs, weaknesses need immediate patching, change or other.

The report will conclude with a small analysis, that defines which findings can be put int the *C.I.A (Confidentiality, Integrity, Availability)* model to access.

SCOPE

The scope of this audit vas internal and external. These are the Services testers did audit.

Host/Service	Description
Database	Local PostgreSQL Database
NPM	Backend Server
Expo	Frontend Server
APK file of the Mobile APP	Analysing the Source Code

Table 1: Scope Details

Executive Summary

According to the nature of the exercise, the time frame of the sprint, we need to consider that the tested application is a prototype. Therefore, the timeframe was limited. The recommendation to procced after the prototype phase, is to patch all the weaknesses found in the testing.

Testers where able to discover a total of **(7)** vulnerabilities on the application. **(3)** are critical, **(2)** are medium, **(0)** are low and **(2)** are informative.

- The application intents to use cleartext network traffic, such as cleartext HTTP.
- The backend server lacks error handling for parsing broken .json files.
- The frontend server crashes and its vulnerable to DoS attack as it doesn't fortify functions that provides checks for buffer overflow.
- The NPM application is working on an older version exposes update weaknesses when user use audit.

According to the CIA model mentioned earlier, the availability of the application gets compromised if the application is putted offline if both services and servers are not available.

The confidentiality of the application gets compromised when external actors may read, manipulate sensitive information extracted from the current application network protocol.

If the application is to continue to the next Phase, it would require parching and updating.

Findings Overview

In the audit testers found a total of **7** security weaknesses. The weaknesses that are categorized as informative is not considered as a security flaw, but including the informative aspect may help, developers' patch.

Security Level				
Critical	Medium	Low	Informative	Total
3	2	0	2	7

Table 2. Security Level

Here we have a more detailed aspect of the security flaws discovered.

Discovery#	Severity Level	Name/Type
1.	Critical	Application use cleartext protocol HTTP.
2.	Critical	Error handling when parsing .json files.
3.	Critical	Doesn't fortify functions to check for buffer overflow.
4.	Medium	Some modules in Frontend Server are outdated.
5.	Medium	Some Files may contain hardcoded sensitive data
6.	Informative	App can write to app directory risking encrypting the data
7.	Informative	App Logs information are stored

Table 3, Discoveries

Review of the Testing

The testing was conducted in 3 phases. The first phase was an internal audit on the database and the server with audit tools and manual testing. The second phase was to test the front-end server with security audit tools on the login area, and the server. The final phase was to audit the application with forensic tools to extract valid data to confirm the weaknesses in the application.

Detailed Review of the testing

We will dive a bit deeper into how testers made the audit, explaining with a more technical view.

1. The application uses cleartext network protocol HTTP. This information was given when testers reviewed the source code, and general scanning tools on the frontend side like Nikto, and mobSF framework.

2. The backend server lacked error handing when parsing .json files, this error occurred when testers replaced the login credentials with a random input, causing the server crash and leave the application unavailable for users. This testing was performed manually.

3. The frontend server was first scanned with external network scanning tools, testers made then several sql injections with sqlMap causing the server to no able to handle the buffer and crashed because it ran out of memory.

4. Some modules of the frontend server holding the application are out of date. Testers used the npm audit command to expose the unpatched dependencies in the server. The testing was done manual, and with mobSF framework.

5. Files in the source code contains sensitive data like API, keys, username, and password. This was obtained by reviewing the source code from the fronted server.

The third phase of the audit consists of reviewing the source code of a fully compiled mobile APK file. This would simulate a fully finished mobile application ready to deploy. The platform would be android. The Framework used to test the application is open source and is named mobSF. The framework confirmed the already mentioned flaws and others. Two that are worth to mention are.

6. The app can write to app directory risking encrypting or tampering the data that is already written in the directory. Information exposed with mobSF.

7. App log data is stored locally. Information exposed with mobSF.

Technical Details Of Each Discovery

1. App using cleartext protocol HTTP– Critical

Type	Cleartext protocol HTTP
Score	Critical
Cause	The application use cleartext protocol HTTP, this is to communicate between servers, from the frontend to backend server to store the data in the database. Users will then add a new account, store the account in the database in cleartext.
Impact	The impact is that the app doesn't safeguard the user's data properly. The network protocol is outdated, vulnerable to man-in-the-middle attacks to external threats. The confidentiality of the whole application is at risk.
Affected Directory	• Whole application.
Remediation	• Implement encryption if data in the database • Update • Use sa safe and encrypted network protocol.
Tools used	Manual testing, reviewing the source code, Nikto, mobSF

Table 4, vuln1

Validation of discovery nr1

The first validation for this vulnerability is a security scan from a test computer with Nmap security testing tool against the server.

```
PORT      STATE SERVICE VERSION
19000/tcp open  igrid?
| fingerprint-strings:
|   GetRequest, HTTPOptions:
|     HTTP/1.1 200 OK
|     Exponent-Server: {"host":"9235be44-016c-45e3-a05e-b8f319cfd25c","server":"expo","serverVersion":"0.7.1","serverD
river":"expo-cli","serverOS":"linux","serverOSVersion":"6.1.0-kali9-amd64"}
|     Date: Sun, 11 Jun 2023 16:02:15 GMT
|     Connection: close
|_    {"name":"ARtQuest","slug":"ARtQuest","version":"1.0.0","orientation":"portrait","icon":"./assets/icon.png","user
InterfaceStyle":"light","splash":{"image":"./assets/splash.png","resizeMode":"contain","backgroundColor":"#ffffff","im
ageUrl":"http://192.168.136.128:19000/assets/./assets/splash.png"},"assetBundlePatterns":["**/*"],"ios":{"supportsTabl
et":true},"android":{"adaptiveIcon":{"foregroundImage":"./assets/adaptive-icon.png","backgroundColor":"#ffffff","foreg
```

Picture 01, Validations

The second validation es the tool mobSF security framework, that scans mobile applications.

2	Clear text traffic is Enabled For App [android:usesCleartextTraffic=true]	high	The app intends to use cleartext network traffic, such as cleartext HTTP, FTP stacks, DownloadManager, and MediaPlayer. The default value for apps that target API level 27 or lower is "true". Apps that target API level 28 or higher default to "false". The key reason for avoiding cleartext traffic is the lack of confidentiality, authenticity, and protections against tampering; a network attacker can eavesdrop on transmitted data and also modify it without being detected.	

Picture 02, Validations

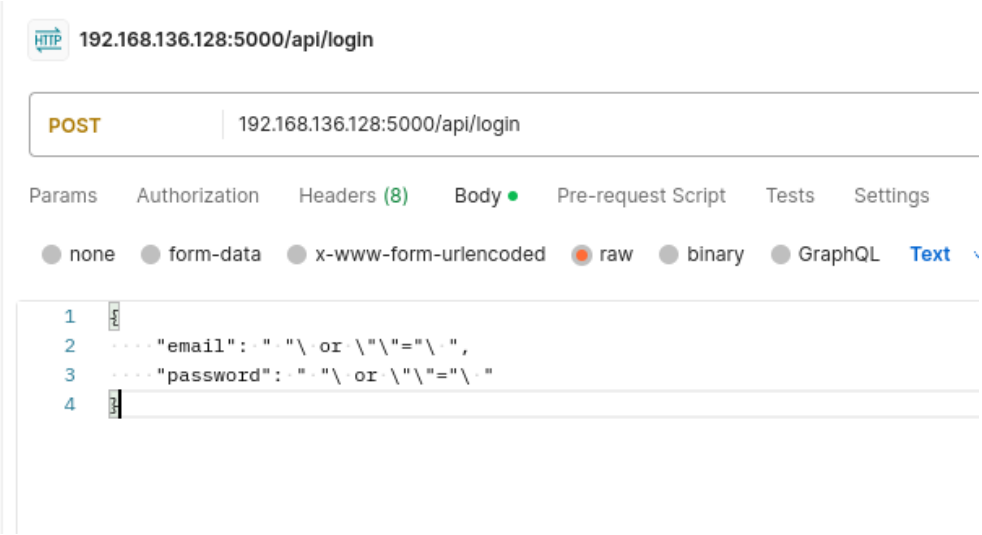
2. Error when parsing .json files – Critical.

Type	Server Parsing error
Score	Critical
Cause	The application on the server side, is not handling errors properly, failing to parse inputs in .json files. This error happens in the login side, where user is sending a post message to the server to login to the application.
Impact	This causes the server to be out of service and compromising the availability of the application.
Affected Directory	• Backend Server
Remediation	• Patching the server to handle errors
Tools used	Manual testing, reviewing the source code.

Table 5, vuln2

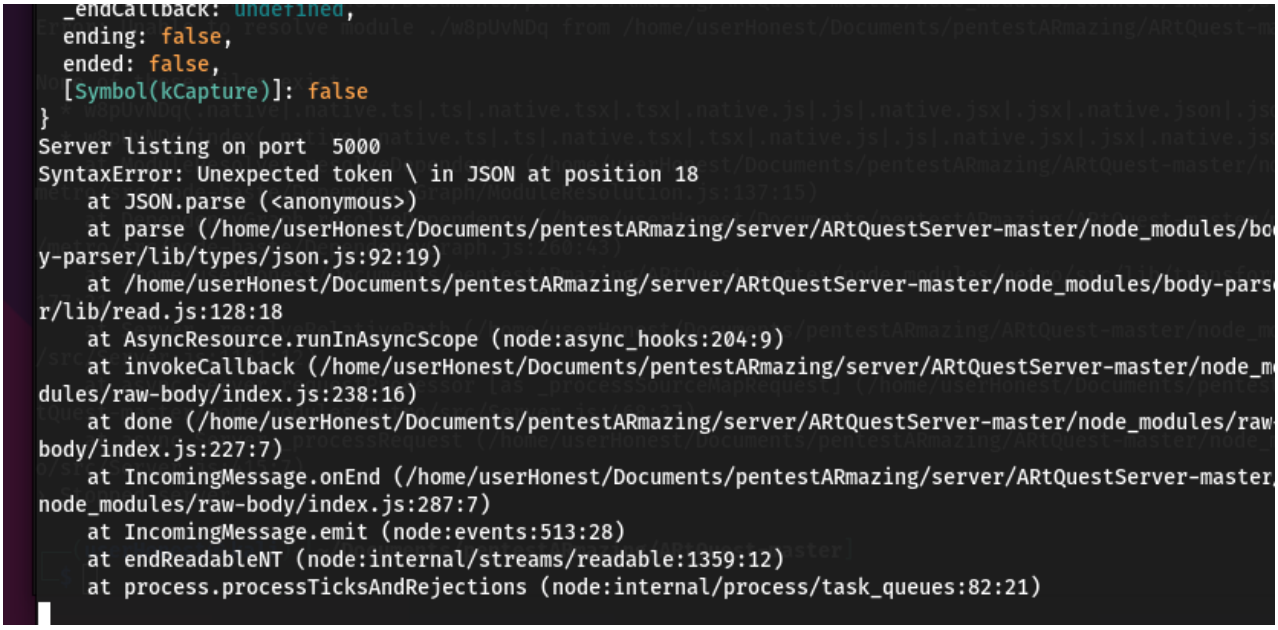
Validation of discovery nr 2

From a test computer to the server, testers sent a post request, simulating a login intent from an external source.



Picture 3, validation 2

This resulted in an error in parsing this input and causing the server to be crash.



Picture 4, validation 2

3. JavaScript heap out of memory attack against the frontend server – Critical.

Type	Heap Attack
Score	Critical
Cause	Testers acquired the IP address and port number of the frontend server. The scanning gave away several directories in the main directory where the server was running. Then they run a sql injection attack with the URL directories found.
Impact	The server got a fatal error and crashed, leaving the front end application unavailable
Affected Directory	• Frontend server
Remediation	• Optimizing the memory usage where the server is running. • Increase memory limits in the application.
Tools used	Nmap, sqlMap

Table 6, vuln3

Validation of discovery nr 3

The frontend server revealed a few directories and attack vectors for testers to perform sql injections to see if one of the gave an input to the backends server and database. This resulted instead in revealing a weakness in the server that is related to handling the memory, and if the applications are to be online and handle thousands of transactions daily it should handle memory more efficiently.

The continuous sql injection attack put the frontend server out of service.

JavaScript heap out of memory.

```
HAR(49) ELSE CHAR(48) END))+CHAR(113)+CHAR(113)+CHAR(118)+CHAR(98)+CHAR(113))) AND 'yaiA'='yaiA Bundling failed
3601ms
Unable to resolve "./Libraries/Components/DatePicker/DatePickerIOS" from "node_modules/react-native/index.js"
ios) AND 5750 IN (SELECT (CHAR(113)+CHAR(113)+CHAR(112)+CHAR(106)+CHAR(113)+(SELECT (CASE WHEN (5750=5750) THEN
HAR(49) ELSE CHAR(48) END))+CHAR(113)+CHAR(113)+CHAR(118)+CHAR(98)+CHAR(113))) AND (8907=8907 node_modules/expo/
ppEntry.js                    0.0% (0/1)
<--- Last few GCs --->

[10862:0x556e35a88580]   348897 ms: Mark-sweep (reduce) 2016.3 (2078.2) -> 2016.3 (2063.2) MB, 1043.1 / 0.0 ms
average mu = 0.367, current mu = 0.000) last resort; GC in old space requested
[10862:0x556e35a88580]   349993 ms: Mark-sweep (reduce) 2016.3 (2063.2) -> 2016.3 (2063.2) MB, 1095.8 / 0.0 ms
average mu = 0.222, current mu = 0.000) last resort; GC in old space requested

<--- JS stacktrace --->

FATAL ERROR: CALL_AND_RETRY_LAST Allocation failed - JavaScript heap out of memory
 1: 0x7feee47f4a98 node::Abort() [/lib/x86_64-linux-gnu/libnode.so.108]
 2: 0x7feee46f40ab [/lib/x86_64-linux-gnu/libnode.so.108]
 3: 0x7feee4b68e60 v8::Utils::ReportOOMFailure(v8::internal::Isolate*, char const*, bool) [/lib/x86_64-linux-gnu
libnode.so.108]
 4: 0x7feee4b6921b v8::internal::V8::FatalProcessOutOfMemory(v8::internal::Isolate*, char const*, bool) [/lib/x8
```

Picture 5, Validation 3

4. Outdatet modules in the Frontend server - Medium

Type	Project dependencies detected vulnerabilities
Score	Medium
Cause	The dependencies show a few vulnerabilities, for example Fast-xml-parser (<4.2.4.) Has a high severity and it's related to regex injection via doctype entities. Other two are related to prototype pollution via parse method.
Impact	Dependencies can be exploited compromising the security of the application. The application can result in functional issues. Regulatory and compliance concerns Entry points for attackers to exploit. More challenging to update later if not addressed.
Affected Directory	Frontend server , npm , node js
Remediation	Running the command npm audit fix –force And it will require that developers address the changes that the update comes with.
Tools Used	Npm , manual testing.

Table 7, Vuln 4

Validation of discovery nr 4

Testers used the command npm audit to get a demonstration and display of the list of vulnerabilities available in the project.

```
└$ npm audit
# npm audit report

fast-xml-parser  <4.2.4
Severity: high
fast-xml-parser vulnerable to Regex Injection via Doctype Entities - https://github.com/advisories/GHSA-6w63-h3fj-q4vw
fix available via `npm audit fix`
node_modules/fast-xml-parser

json5  <1.0.2
Severity: high
Prototype Pollution in JSON5 via Parse Method - https://github.com/advisories/GHSA-9c47-m6qq-7p4h
fix available via `npm audit fix --force`
Will install expo@46.0.21, which is a breaking change
node_modules/find-babel-config/node_modules/json5
  find-babel-config  <=1.2.0
  Depends on vulnerable versions of json5
  node_modules/find-babel-config
    babel-plugin-module-resolver  2.3.0 - 4.1.0
    Depends on vulnerable versions of find-babel-config
    node_modules/babel-plugin-module-resolver
      babel-preset-expo  *
      Depends on vulnerable versions of babel-plugin-module-resolver
      node_modules/babel-preset-expo
        expo  >=14.0.0
        Depends on vulnerable versions of @expo/cli
        Depends on vulnerable versions of @expo/config
        Depends on vulnerable versions of @expo/config-plugins
        Depends on vulnerable versions of babel-preset-expo
        Depends on vulnerable versions of expo-asset
        Depends on vulnerable versions of expo-constants
        node_modules/expo

xml2js  <0.5.0
Severity: moderate
xml2js is vulnerable to prototype pollution  - https://github.com/advisories/GHSA-776f-qx25-q3cc
fix available via `npm audit fix --force`
Will install expo@46.0.21, which is a breaking change
node_modules/xml2js
  @expo/config-plugins  *
  Depends on vulnerable versions of xml2js
  node_modules/@expo/config-plugins
    @expo/cli  >=0.1.0
    Depends on vulnerable versions of @expo/config
    Depends on vulnerable versions of @expo/config-plugins
```

Picture 6, validation 4

5. Hardcoded sensitive data - Medium

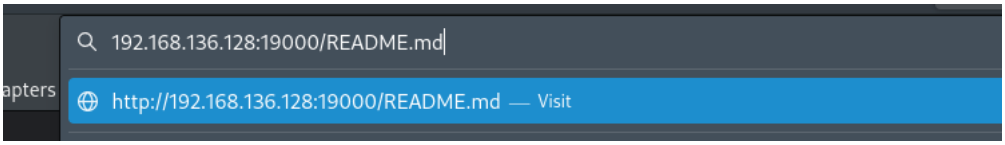
Type	Some files may contain sensitive data
Score	Medium
Cause	A file named README.md that contains a password to log in to the application, testers suspect that the password may belong to developers to test the application and forgot to delete it. File was retrieved from external testing on the frontend.
Impact	Outsiders can have access to the application with credentials that are higher privilege than normal users.
Affected Directory	Frontend server
Remediation	Remove any unnecessary files that may contain sensitive data, remove the test credentials from database in backend.
Tools Used	Nikto, FireFox

Table 8, Vuln 5

Validation of discovery nr 5

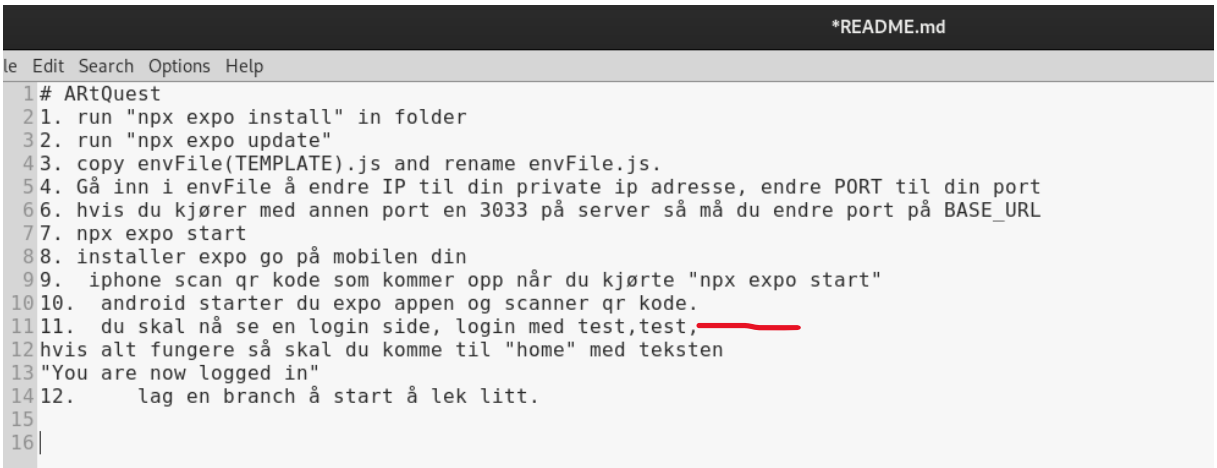
Testers performed a vulnerability scan and discovered that it's possible to access a README.MD file from a testing computed to the front-end server. A file was downloaded and revealed instructions that was intended for the team by storing login credentials to the application in the testing phase.

Testers accessing the file from Firefox.



Picture 7, validation 5

The content of the file



Picture 8, validation 6

6. App can write to app directory – Information.

Type	App can write to App Directory
Description	Sensitive data is not being encrypted; it's getting stored in app directory
Impact and Remediation	The data could be exposed. Use secure storage mechanisms
Affected Directory	Forensic analysis on the apk file
Tools	Mobile Security Framework (mobSF).

Table 9. Vuln 6

Validation of discovery nr 6

Testers submitted a fully compiled APK file of the application to the mobile security framework, the application was opened, and evaluated.

4	App can write to App Directory. Sensitive Information should be encrypted.	info	CWE: CWE-276: Incorrect Default Permissions OWASP MASVS: MSTG-STORAGE-14	expo/modules/adapters/react/permissions/PermissionsService.java expo/modules/constants/ExponentInstallationId.java

Picture 9, Validation 6

7. App logs Information Are stored – Information.

Type	Sensitive information is being logged
Description	The sensitive of the information is being logged
Impact and Remediation	Could lead to data breaches. Apps need to follow the MASVS standard that pertains to secure storage.
Affected Directory	Forensic analysis on the apk file
Tools	Mobile Security Framework (mobSF).

Table 10, Vuln 7

Validation of discovery nr 7

Testers submitted a fully compiled APK file of the application to the mobile security framework, the application was opened, and evaluated.

1	The App logs information. Sensitive information should never be logged.	info	CWE: CWE-532: Insertion of Sensitive Information into Log File OWASP MASVS: MSTG-STORAGE-3	com/bumptech/glide/load/resource/bitmap/DrawableToBitmapConverter.java com/bumptech/glide/load/resource/bitmap/HardwareConfigState.java com/bumptech/glide/load/resource/bitmap/TransformationUtils.java com/bumptech/glide/load/resource/bitmap/VideoDecoder.java com/bumptech/glide/load/resource/gif/ByteBufferGifDecoder.java

Picture 10, Validation 7

8. Defining Methods

Tools used in the audit total (5)

Tools	Description
Nmap	Network Vulnerability Scanner
Nikto	Network Vulnerability scanner
SQLmap	SQL injections
Manual Testing	-
mobSF	Mobile Security Framework Testing