Android System Security Evaluation Manager

Advisor: Leon Reznik
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY 14586

Abstract—Security is an important aspect that needs to be taken care of. With the progress in technology, along with its uses, its misuses have also increased. Hence, there is a strong need to keep oneself secure from the various possible threats. This paper deals with the Android device’s security aspect. There is an existing Android application which evaluates the security level of an Android device based on certain pre-defined parameters. In this paper, we discuss the implementation of a backend system for the Android application. It involves the development of a RESTful API and a Web application to provide the user with some additional functionalities.

Keywords—Android security; REST API; Web application; Score comparison;

I. INTRODUCTION

The first ever mobile phone was produced by Motorola on 3rd April 1973. From being used for just calling, texting to clicking pictures, mobile banking, virtual reality, the mobiles have gone through a lot of evolution [1]. Mobile phones have become a diamond mine of personal information. Mobile phones used to store personal information from the start. But now with the development of the mobile phones, the personal information is exposed to the outer world. Social media is an example of the same. This makes every user vulnerable to the outside world. We all feel safe unless we find out that how our information can be used to influence things, influence our decisions without us knowing about it. A most recent example is that of Facebook and Cambridge Analytica. Cambridge Analytica had access to Facebook user’s information. This information was then used to influence voters during a presidential election [2]. Social media giants like Facebook tend to have a tremendous amount of personal information belonging to users obtained by means of mobiles usage.

This paper focuses on the Android devices. Among all the smartphones, Android-based smartphones are most popular. In the first quarter of the year 2016, Android had a market share of 84.1%, while iOS, Windows, and others had a market share of 14.8%, 0.7% and 0.4% respectively [3]. The Android framework is an open source framework. Android allows to install and use applications from unknown sources, even if they are not available on the Google PlayStore. These are some of the major reasons why one of the widely used smartphone platforms is not secure. Therefore, we have opted to work with the security aspect of the Android devices. Different parameters that should be considered while evaluating the security of an Android device can be narrowed down to specific ones. Now, these parameters can be considered for every Android device and some form of conclusion on the level of security can be obtained. Further, if this level of security is compared with the same of other similar devices, it will give the user an enhanced idea of the same.

II. BACKGROUND WORK

Igor Khokhlov and Leon Reznik have already worked on an Android application by name ‘System Security Evaluation’. The execution of this application is dependent on Android OS API and Google SafeNet library [4]. This application supports devices with Android OS version 5.0 and above. In this application, the system parameters of the Android OS device are evaluated and a score is generated. This score aides the user in understanding the security level of the Android OS device. Along with the score, the application provides some suggestions to assist users in making their Android OS device more secure.
III. ANALYSIS

A. Problem

This application does a good job in evaluating the system parameters, generating a security score and providing suggestions based on the score. There are certain functions that if added to the current Android application will enhance it and make it even more useful for helping the user. Currently, it is not able to give the user an idea of how good the generated security score is, in comparison with other similar Android devices. Also, if the user would like to see a history of the given Android device’s security score generated by the Android application, there were no means to do that. Thus, the user just received the information obtained from the most recent execution of the application.

B. Solution

The Android application could be modified to incorporate the functionalities. But adding more to the Android application would make it bulky in size. Also, if for some reason the application would have crashed, the user could lose all his data. For comparing the scores between various devices, it would be very difficult with the involvement of frequent communications between devices. In this scenario, an Android device would have to make communication with fellow Android devices even when a request was not initiated by its user. Thus, having the Android devices communicate with each other directly is highly undesirable.

The web is an area that is currently being actively researched on. Tremendous efforts are been taken on making smart things more accessible and to make them able for managing them more efficiently following certain Web standards. A huge development has already been achieved in this area [5]. To integrate the additional features, we then planned to for having a backend system for the ‘System Security Evaluation’ Android application. The Android application would communicate with this application and keep the user updated. Thus, the user would always have a backup of his information at a single place other than the Android device. Every time a new score is generated using the ‘System Security Evaluation’ Android application, the score will be sent to the backend system and will be saved against the corresponding user. If at any time the user wishes to see all his previous scores, the request can be served by the backend system.

Our first thought was to create a simple server application but to make it efficiently reusable, a decision was made to develop an Application Programming Interface. Having an interface that would help in connecting the android application to the data, by providing it the required services to operate on the data like the score and the user information would be useful. Also, since we would have an interface, we planned to develop a web application as well which would give the user an option to check his relevant information using a browser.

The Application Programming Interface (API) would act like a plug. Thus, the System Security Evaluation application and the web application would just have to plug themselves into the API and consume the services provided. Now the next question was how would the application programming interface look like. There are different architectural patterns that can be followed while developing such interfaces. Simple Object Access Protocol (SOAP) and Representational State Transfer (REST) are two types of Service-Oriented Architecture (SOA). We then dived into comparing the Simple Object Access Protocol and the Representational State Transfer Protocol for its various advantages and disadvantages to find out a suitable option. The Simple Object Access Protocol is more traditional as compared to the Representational State Transfer Protocol. We studied their differences and how they work. From the experiments performed by the authors of [6] Smita Kumari and Santanu Kumar Pant. It was concluded that the Representation State Transfer is a better service-oriented architecture as compared to its counterpart. Representational State Transfer has a better throughput and a faster response time than Simple Object Access Protocol. The SOAP contains a lot of overhead as a huge drawback making its user undesirable [6]. Thus, we opted to develop an application programming interface following the Representational State Transfer Protocol.

REST provides a client-server type of architecture. It is a hybrid style that is based on 12 other styles [6]. We found the properties of REST-like its simplicity, usability, and scalability to meet our purpose for developing an appropriate backend system [7]. Thus, the backend system is a RESTful API, which will serve the Android application to help make the application more valuable to the user.

IV. SYSTEM ARCHITECTURE

Previously the system consisted only of the Android application. We proposed a modified system containing a RESTful API and a web application as well.

As we see above, the Android application and the web application, both would send requests to and receive responses from the API.

A. REST API

There are many options which can be used to build a REST API. Out of the available options, we chose to build this API using the ASP.NET Core with the MVC middleware. A benefit
of doing so is to have an API which can be executed successfully on different platforms. There are various levels of REST API that can be developed based on the Richardson Maturity Model (RMM). If a REST API provides services that would take hypermedia as an engine of application state, then that REST API can be called as a level 3 REST API. The API developed as for the required setup is a level 3 REST API. The API middleware framework provided by the ASP.NET Core helps in building applications using the Model-View-Controller design pattern. Using the MVC design pattern helped in having the components loosely coupled. The API is hosted on Kestrel server along with Internet Information Services (IIS) server which acts as a reverse proxy server.

1) Outer-facing contract

Now the first task here was to design an outer facing contract for the user. The three important components here are resource identifier i.e. the URIs which will be used by the consumer of the API to interact with it, then the HTTP methods like GET which is used to get resources from the API and an optional payload is like a request body and will be used when there is a creation of resource involved. The payload can contain data in different formats like JSON. For this media types are used. These three components are vital for successful communication between the consumer and the API.

2) Designing URI

Then the next step was to name the resources. Designing the URIs is also an important task. To make URIs appropriate nouns were used rather than actions that can be performed on the resources. URIs would help in understanding its exact ability. To keep access to different resource types simple, URIs have been created accordingly. For example, instead of having ‘api/{id}/resourceType1’ and ‘api/xyz/resourceType2’, URIs like ‘API/resourceType1/{id}’ and ‘API/resourceType2’ were created. The proper hierarchy for working with the resources using the URIs was also kept in mind. If a user must access all his scores, he would have to use URI like ‘API/users/{userID}/scores’. This makes URIs easy to understand and work with. If a user needs to get his scores ordered in a specific order, one option is to have a URI like ‘API/users/{userID}/orderByDate/score’, but a better option would be to have a URI like ‘API/user/{userID}/scores?OrderBy>Date’. Having the second type of URI here helps in minimizing unnecessary fetching of resources. Just having it as some parameter, the data can then be worked on before sending it to the consumer after it has been fetched once.

3) UID

Every user needs to have some form of unique identification. A normal practice would be to have auto-numbered database field for the ID parameter. For the current setup, we are using Microsoft SQL Server to store the data. A problem that can arise here is that if in future the database is changed to MongoDB or any other database, then the unique IDs would change for all the resources. To avoid this issue, we have used GUIDs. It allows switching of the backend data stores. We have used Entity Framework for smoothly working with the database.

4) Status Codes

Status codes is another vital aspect of a REST API. If the user wishes to know if the request that he made has been processed as expected or not, or if the request made has failed then what was the reason behind its failure. Thus, status codes convey meaningful information to the consumer of the API. Without the status codes, it would be difficult to figure out if the error was due to the API or the consumer or due to the network. There are different levels of status codes available that need to be implemented for supporting different aspects. We have implemented the level 200 i.e. the success status codes, level 400 i.e. the status codes which indicate that the client is at fault, level 500 i.e. the status codes indicating if there was something wrong within the server. The issues arising during execution can be categorized as errors and faults, the errors being issues generated due to the client and the faults being the problems caused due to the server. Our API handles both these types of issues.

5) Media types

The different formats in which the request and response have been encoded serve various purposes. The representation which the best for a given request-response pair can be opted for. Our API supports JSON, XML and a customized media type. The customized media type is used for sending the different links to the various actions that can be performed on the given resource. The Accept header is used to specify the format of the content. As a default, if not Accept header is passed in the request JSON is used. Accept header is used for specifying the format of the response expected by the client, while the Content-type header is used to specify the format of the content that is being sent as a part of the request.

6) HTTP methods

There are many actions which can be performed on the resources. Hence, the selection of the HTTP methods for interacting with the resources had to be done wisely. Depending on the resource and the operation, an action which would make sense would be a resource when not needed would have been a problem. So, we appropriately made use of the GET, POST, DELETE, PATCH requests to develop a system which modifies the data only in an expected manner. An example can be the deletion of a specific resource. If a user would like to delete his information or his account information, he should be allowed to do so. But if the user knowingly or unknowingly tries to delete all the users in the database which would be a highly undesirable, unauthorized action, then that would be a major setback.

7) Comparing the security score

One of the important functionalities needed in comparing the security score generated with others to have a better idea of how good the score is. Every time the API receives a new score generated by the Android application, it would then compare it with the latest security scores of the Android devices generated by the System Security Evaluation application and available in the database. Thus, while sending out a confirmation of making an entry of the new score for a device, the API also sends out a comparison parameter which would be like ‘x/y’. This means that it was compared with the latest scores of y devices and x is its rank among them. Meaning it has the x\*th best score among
all the Android devices. For this, it makes sure of accurately noting the latest score in the database and then marking the previous scores from the same device as old. Along with the score, the parameters that resulted in the score obtained are also stored in the database. This is done for every score. This helps if the user at any time in the future wishes to check the parameters for different scores at the time the score was recorded.

8) Repository pattern

There are various functions or operations which can be performed on the similar type of data. Basic CRUD operations or certain other tasks may use same functions for multiple purposes. To avoid duplication of code, make it less error-prone and reduce the complexity we have used a repository which contains the operations that can be performed on data. As and when they are needed for relevant reasons, they can be used. To implement this the Repository design pattern has been used.

9) Logging faults

Faults can occur during the execution of an application. Appropriate means should be available to make a note of the faults along with its details. It would help in reaching out to the exact reason that caused the issue. Faults can occur in any component of the application. It will be a good option if we have a means of record all the faults in a single place and have access to it even at a different time. To achieve this, we have incorporated a library called NLog. With the help of this our API writes all the faults into a single file, which then can be accessed at any given point of time in future.

10) Other API functionalities

Certain other functionalities have also been implemented to ease working with the data. If a certain resource needs to be retrieved in the form of multiple pages, where there needs to be a certain number of items on a single page along with the number of pages in which the information must fit, that has been implemented. Links to previous and next pages will also be sent as a part of it. Thus, paging of resources is available. If the user wishes to filter out his score based on certain rules, that can be done. There is also a function to search through available items. Also, if the certain resource contain x parameters and the user wishes to retrieve only a subset of those parameters, an option is available for that.

11) Hypermedia as Engine of Application State (HATEOAS)

To make the API completely RESTful, we had to make sure it is fully self-descriptive. Usually, if the user has to perform an action on a resource, he would have to go through the API’s documentation to know all the possible actions of a given resource. We have tried to reduce this user’s effort, by attaching links for the authorized actions that can be performed on the given resource. Thus, in this case, all the information required by the user is contained in the response message body itself. The clients do not have to worry about the design aspect, while they just should use the links attached to the response to future actions.

12) Caching

To reduce the number of requests being processed, network bandwidth we have implemented caching as well. There is a component in the API which stores responses that can be cached. They follow an expiration model. A cached response is considered valid only until a certain amount of time mentioned in the API. One the response expires a regular call to retrieve the information from the API is made. It can happen that a response has not yet expired, but its actual value has changed. To handle such situations an Entity tag has been used. It stays in the response header and it’s valued changes if the resource has been modified. These tags also help in avoiding concurrency issues.

13) Protecting the API

As we are dealing with the security aspect of a system. Our API also needs to be secured. An attempt can be made to crash our API by flooding it with multiple requests within a very short span of time. Rate limiting has been implemented to avoid this. The number of requests received from a specific IP or any given IP can be restricted to a certain count within a given time frame. If the client exceeds this count, an error message will be returned and its request will not be able to go through. With respect to user’s privacy, the user’s sensitive information is stored in the database in an encrypted format.

B. Web Application

The main heart of the backend system was implemented. We still wanted to make things easier for the user. We wanted the user to be able to easily access his relevant information. Hence, we decided to develop a web application which would act as a client as consume the developed REST API. This web application would get the information stored in the REST API, entered it by the Android device. Thus, the user will have one more interface to look at all his information at a single place. For this, a client was developed in PHP. With the help of HTML, CSS, and Bootstrap, a visually appealing display was created.

To host the web application, Apache server was used. The user would first have to log in to the system with his credentials and then he would get access to his user information. Then the user will have a further option of looking at his recorded scores.

Like a regular user, there is a provision for an admin as well. The admin will log in to the system using the same login page, but with the admin credentials. The admin would then be able to perform various operations on all the user related information and their score information.

V. RESULTS

Once all the components of the backend system were implemented and started running, we had to make sure they if they are working fine by giving the expected outputs and that there is no error during the entire cycle of execution.

A. Process memory and CPU usage

We had to make sure that the setup is not consuming too much of memory and keeping the CPU busy. Following is a
snapshot of the process memory in megabytes and the percentage of the CPU being used once the REST API is triggered.

As we can see from the above graphs that when the API is triggered it consumes some amount of CPU which is almost under 50% and then uses minimal of it. Now let us have a look at the process memory usage and the CPU usage when the API keeps getting requests from the client, which is as below.

As we can see from the above graphs that when the API is triggered it consumes some amount of CPU which is almost under 50% and then uses minimal of it. Now let us have a look at the process memory usage and the CPU usage when the API keeps getting requests from the client, which is as below.

![Figure 3: Process memory and CPU usage in the first 10 seconds of starting the API](image)

![Figure 4: Process memory and CPU usage during normal functioning of the API](image)

It is clearly seen from the graph shown above that when the API is idle i.e. it is not serving any requests to clients, it does not use a lot of the CPU processors. This is expected for normal functioning of an API. We would not want our backend system to unnecessarily consume resources when it is not needed. Once a set of events are served, the CPU processes again fall back to it minimalistic usage.

**B. API requests-response details**

Then we had to check if the requests that our API was supposed to handle, was it doing it properly. For this, we generated various requests and observed their responses. As a part of their responses along with it content we analyzed the status code returned, the time required for the response to be received and the size of the content received as a part of the response. The details of these have been given in the following table.

<table>
<thead>
<tr>
<th>Request Type</th>
<th>Status Code</th>
<th>Time (ms)</th>
<th>Response size</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET all registered users (JSON)</td>
<td>200 OK</td>
<td>204ms</td>
<td>1.57KB</td>
</tr>
<tr>
<td>GET all registered users (JSON-Cached response)</td>
<td>200 OK</td>
<td>50ms</td>
<td>1.57KB</td>
</tr>
<tr>
<td>GET all registered users (XML)</td>
<td>200 OK</td>
<td>920ms</td>
<td>5.68KB</td>
</tr>
<tr>
<td>GET all registered users (XML-Cached response)</td>
<td>200 OK</td>
<td>71ms</td>
<td>5.38KB</td>
</tr>
<tr>
<td>GET Existing single user</td>
<td>200 OK</td>
<td>188ms</td>
<td>622B</td>
</tr>
<tr>
<td>GET Non-Existing single user</td>
<td>404 Not Found</td>
<td>130ms</td>
<td>0B</td>
</tr>
<tr>
<td>GET scores for a valid user</td>
<td>200 OK</td>
<td>327ms</td>
<td>1.1KB</td>
</tr>
<tr>
<td>GET scores for an invalid user</td>
<td>404 Not Found</td>
<td>81ms</td>
<td>0B</td>
</tr>
<tr>
<td>GET specific score for a valid user</td>
<td>200 OK</td>
<td>192ms</td>
<td>968B</td>
</tr>
<tr>
<td>GET non-existing score for a valid user</td>
<td>404 Not Found</td>
<td>161ms</td>
<td>0B</td>
</tr>
<tr>
<td>GET Existing single user (Unsupported format)</td>
<td>406 Not Acceptable</td>
<td>130ms</td>
<td>0B</td>
</tr>
<tr>
<td>GET users (Order by Name)</td>
<td>200 OK</td>
<td>137ms</td>
<td>765B</td>
</tr>
<tr>
<td>GET score information (Order by, Search, Page Number, Page Size)</td>
<td>200 OK</td>
<td>184ms</td>
<td>395B</td>
</tr>
<tr>
<td>POST score for a valid user</td>
<td>201 Created</td>
<td>1286ms</td>
<td>982B</td>
</tr>
<tr>
<td>POST user information for the new user</td>
<td>201 Created</td>
<td>885ms</td>
<td>1.39KB</td>
</tr>
<tr>
<td>DELETE Existing single user</td>
<td>204 No Content</td>
<td>958ms</td>
<td>0B</td>
</tr>
<tr>
<td>DELETE a score for the Existing user</td>
<td>204 No Content</td>
<td>1174ms</td>
<td>0B</td>
</tr>
</tbody>
</table>

All appropriate responses were received with their expected status codes. We tried sending responses with the Accept header as JSON and XML i.e. specifying the expected return format of the data. It was observed that the time required to receive the XML format data was quite longer almost twice as compared to the time required for receiving data in the JSON format. Also, the size of the data received in the XML format was quite larger as compared to the size of the data received in the JSON format. Here it thus seems to be a good choice of using a RESTful API which supports JSON. If needed by some client applications, we still do have provided our API the ability to respond and understanding XML format data.

For the first time, when a request was served by the API, consider for example a GET request as mentioned in the above table, it took some time to respond with the data. The request for getting all the users in JSON format took 204 milliseconds in the first time. The same request when it was served again, caching took care of it. Thus, this time the response to the same request was served in 50 milliseconds. This is a good amount of reduction in time. This would help in reducing the load on the network, avoid unnecessary processing in the API.

Critical operations like the creation and deletion of the resources consumed more amount of time as compared to the retrieval of resources even with different filtering and sorting
parameters applied. Here we observed that the creation and deletion of a user take lesser time as compared to the creation and deletion of the scores. The reason here is that the for operating on a score, the system would first have to go through its user, which adds up some time to process the request.

C. Comparison of security score

Following is a snippet of the JSON request sent for posting a new score against a user in the database and the response received for it.

```json
{
  "instanceScore": 8,
  "screenLock": 1,
  "os": 2,
  "unknownSources": 1,
  "harmfulApps": 1,
  "devOpt": 1,
  "compatibility": 1
}
```

Figure 5: JSON request for posting a new score

```json
{
  "id": "0a19b4f5-e5a-4d97-a82d-e16df76d7b09",
  "timestamp": "2018-04-19T11:28:00.6183651-04:00",
  "instanceScore": 8,
  "userId": "25320c5e-f58a-4b1f-b63a-8ee07a840bdf",
  "scoreComparison": "1 / 6",
  "latestScore": true,
  "imei": "1234566554798120244",
  "screenLock": 1,
  "os": 2,
  "unknownResources": 1,
  "harmfulApps": 1,
  "devOpt": 1,
  "compatibility": 1
}
```

Figure 6: JSON response for new score entry

Figure 5 contains an example of the request content for posting a new score against a user in the JSON format. While posting the score, user identification number is obtained from the URI of the request. The request contains the score and all the parameter details corresponding to this score. Figure 6 is an example of a response to posting a new score in the database. From figure 6 we can see that for a new score entered, it was compared with six other latest scores of different devices available in the database. Since this device had the perfect score and the best score among all the six devices it was ranked first. Other than this, the JSON object contains information about the score unique identification number, the timestamp when it was recorded, the user’s unique identification number, if this score is latest one or not, the IMEI of the device and the details of the parameters that resulted in this score.

D. List of the score as provided on the web application

Following is a snapshot of how the scores would appear to a user if he checks them using the website.

![User Score Information Table](image)

Figure 7: Web application- Score information for a user

VI. CONCLUSION

Thus, we can conclude that with the system developed as a backend system for the System Security Evaluation Android application solves all the issues mentioned in this paper. It successfully can provide important functionalities like the score comparison. Comparing it with all the available scores would help the user in getting a better idea of the security level of his Android device. Also with the previous scores available the user can prepare or take necessary steps to keep his device safe.

VII. FUTURE WORK

Integrating this backend system with the System Security Android application is one of the important things that need to be done. Given that this backend has been developed following certain standards, the integration can be performed successfully. Currently, the backend system is specifically created keeping the Android system security parameters into consideration. In the future, support can be added for other types of Operating Systems. If more resources are needed for running the backend system, it can be pushed on a cloud system like Microsoft Azure or Amazon AWS.

VIII. ACKNOWLEDGMENT

I would like to thank Dr. Leon Reznik, who advised and motivated me throughout the semester while working on this project and Dr. Joe Geigel for his support by making sure we followed the given timeline, by providing feedback. I would also like to thank Igor Khokhlov for providing insights on the System Security Evaluation Android application, which helped me understand how it works. I would like to thank the Computer Science Department of Rochester Institute of Technology for providing all the valuable resources and facilities that resulted in completing the project successfully.

REFERENCES


