Misconfigured Firebase Database

Last Year(2018), Security researchers found that 100 million records (113 gigabytes) of data from unsecured Firebase databases of iOS and Android apps were leaked.

If you can find out such misconfigured backend, you can even compromise the entire DB.

Firebase Realtime Database security rules are how you secure your data from unauthorised users and protect your data structure. An improper security rule will cause data leakage or even the compromise of the entire DB.

Here I'm going to explain how to find out such misconfigured firebase backend. You can include it in your mobile security test checklist.

I'm using my intentionally vulnerable application **Fire_Vu** for this vulnerability demonstration.

Prepared by -
Sahad Bnu Abid Thangal

1.Decompile apk :

apktool d FireVu.apk

2. grep the output directory for finding **Firebase database url** or **Project id'**

```
we45@we45:~/Desktop/firevu/Release-latest/FireVu$ grep -iR "project_id"
smali_classes2/com/google/api/Service.smali:.field public static final PRODUCER_PROJECT_ID_FIELD_NUMBER:I = 0x16
smali_classes2/com/google/firebase/FirebaseOptions.smali:.field private static final PROJECT_ID_RESOURCE_NAME:Ljava/lang/String; = "project_id"
smali_classes2/com/google/firebase/FirebaseOptions.smali:    const-string v2, "project_id"
res/values/strings.xml:    <string name="project_id">firevu-db</string>
res/values/public.xml:    <public type="string" name="project_id" id="0x7f0e0057" />
smali_classes3/com/vulnerable/vufireb/R$string.smali:.field public static final project_id:I = 0x7f0e0057
```

Project id**:firevu-db**

```
we45@we45:~/Desktop/firevu/Release-latest/FireVu$ grep -iR "firebase_da"
smali_classes2/com/google/firebase/internal/DataCollectionConfigStorage.smali:.field public static final DATA_COLLECTION_DEFAULT_ENABLED:Ljava/lang/String; = "firebase_data_collection_defaul
t_enabled"
smali_classes2/com/google/firebase/internal/DataCollectionConfigStorage.smali:    const-string v1, "firebase_data_collection_default_enabled"
smali_classes2/com/google/firebase/internal/DataCollectionConfigStorage.smali:    const-string v1, "firebase_data_collection_default_enabled"
smali_classes2/com/google/firebase/FirebaseOptions.smali:.field private static final DATABASE_URL_RESOURCE_NAME:Ljava/lang/String; = "firebase_database_url"
smali_classes2/com/google/firebase/FirebaseOptions.smali:    const-string v2, "firebase_database_url"
res/values/strings.xml:    <string name="firebase_database_url">https://firevu-db.firebaseio.com</string>
res/values/public.xml:    <public type="string" name="firebase_database_url" id="0x7f0e0047" />
smali_classes3/com/vulnerable/vufireb/R$string.smali:.field public static final firebase_database_url:I = 0x7f0e0047
```

firebase_database_url:***https://firevu-db.firebaseio.com***

3. Access the Firebase DB URL as in the following format:

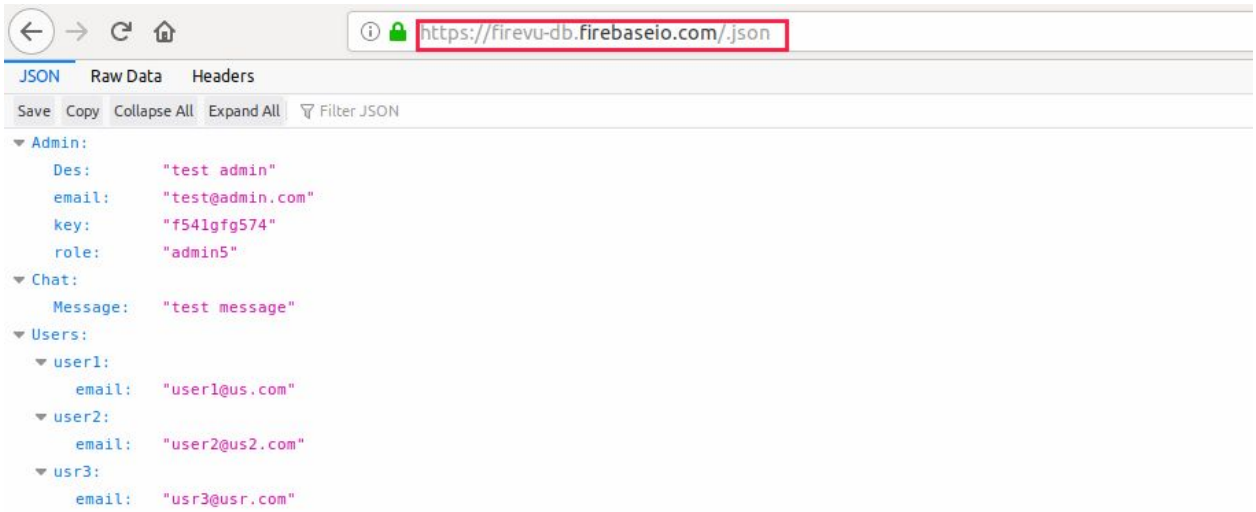
- https://**project_id.**firebaseio.com/.json or
- https://**project_id.**firebaseio.com/**node_name**/.json

So, Here it will be:

https://firevu-db.firebaseio.com/.json

Example Cases

Case 1:



Key point: Any unauthorized user can retrieve the data from the '**firevu-db**', but can't write data .

```
1  {
2    "rules": {
3
4      ".read":true,
5      ".write":false
6
7    }
8  }
```

poorly implemented security rule which causes the above data leak (Any parent/child node in the DB is readable by anyone).

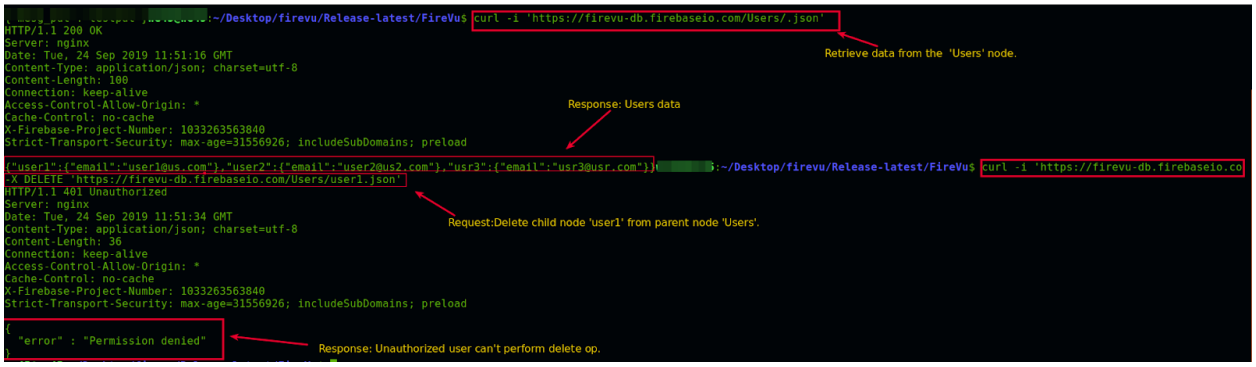
Case 2:

Simulator

```
{
  "rules": {

     "Admin":
      {
      ".read":true,
      ".write":false

      },

      "Users":
      {
        ".read":true,
        ".write":false
      },

      "Chat":
      {
        ".read":true,
        ".write":true
      }


  }
```

Key point: Any unauthorized user can retrieve the data from the ‘Users’ and ‘Chat’ parent nodes, but not from the ‘Admin’ node. Also, unauthorized write operation can perform on ‘Chat’ node. (We need to Brute force URL to get node names)

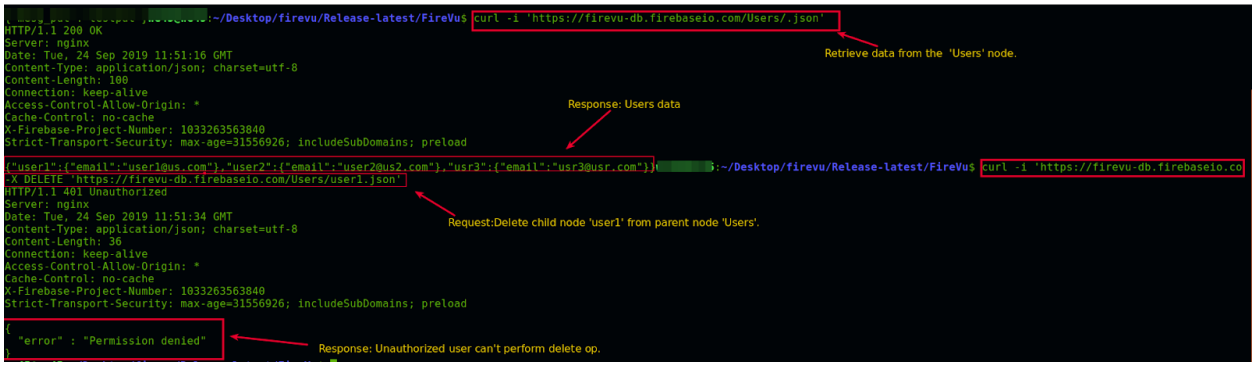
For the ‘Users’ node:



Result:

Any unauthorized user can retrieve the data from the parent node ‘Users’, but can’t write.

For the ‘Chat’ node:

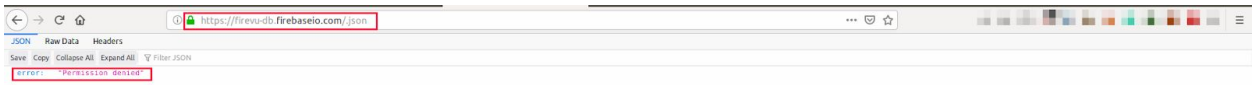


Result:

Any unauthorized user can retrieve and the data from the parent node ‘Chat’, but can’t write.

Remediation:

Configure your database security rules properly so that only authorised users have read or write access to data.



Reference:

FIrebase REST references:

https://firebase.google.com/docs/reference/rest/database#section-delete

FIrebase Security Rules:
https://firebase.google.com/docs/database/security/quickstart