Day-3

Sensitive Data Exposure

[Task 9] [Day 3] Sensitive Data Exposure

When a webapp accidentally divulges sensitive data, we refer to it as "Sensitive Data Exposure". This is often data directly linked to customers (e.g. names, dates-of-birth, financial information, etc), but could also be more technical information, such as usernames and passwords. At more complex levels this often involves techniques such as a "Man in The Middle Attack", whereby the attacker would force user connections through a device which they control, then take advantage of weak encryption on any transmitted data to gain access to the intercepted information (if the data is even encrypted in the first place...). Of course, many examples are much simpler, and vulnerabilities can be found in web apps which can be exploited without any advanced networking knowledge. Indeed, in some cases, the sensitive data can be found directly on the webserver itself...

The web application in this box contains one such vulnerability. Deploy the machine, then read through the supporting material in the following tasks as the box boots up.

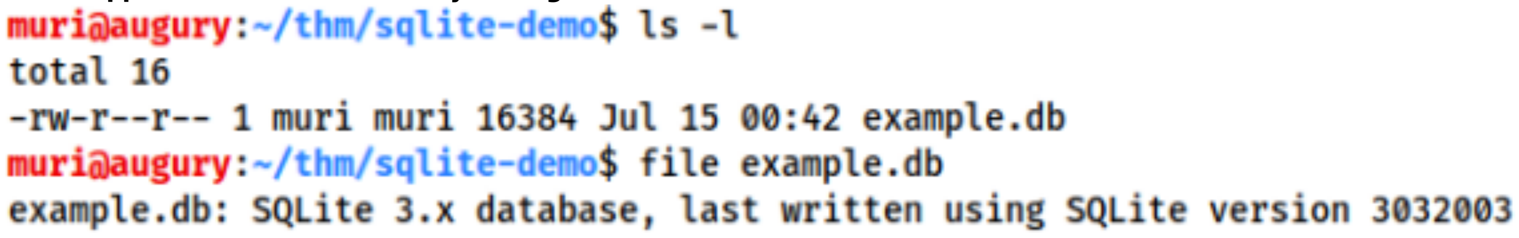
#1
Read the introduction to Sensitive Data Exposure and deploy the machine.

No answer needed

[Task 10] [Day 3] Sensitive Data Exposure (Supporting Material 1)

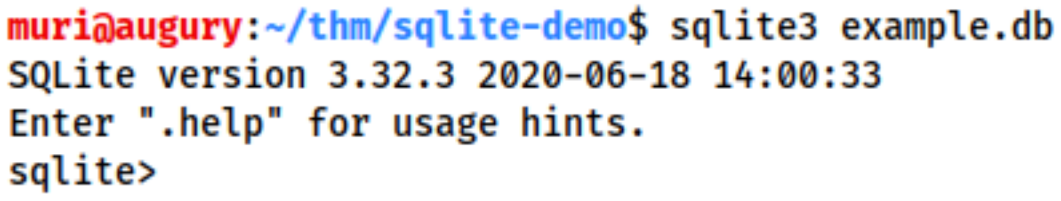
The most common way to store a large amount of data in a format that is easily accessible from many locations at once is in a database. This is obviously perfect for something like a web application, as there may be many users interacting with the website at any one time. Database engines usually follow the Structured Query Language (SQL) syntax; however, alternative formats (such as NoSQL) are rising in popularity.
In a production environment it is common to see databases set up on dedicated servers, running a database service such as MySQL or MariaDB; however, databases can also be stored as files. These databases are referred to as "flat-file" databases, as they are stored as a single file on the computer. This is much easier than setting up a full database server, and so could potentially be seen in smaller web applications. Accessing a database server is outwith the scope of today's task, so let's focus instead on flat-file databases.
As mentioned previously, flat-file databases are stored as a file on the disk of a computer. Usually this would not be a problem for a webapp, but what happens if the database is stored underneath the root directory of the website (i.e. one of the files that a user connecting to the website is able to access)? Well, we can download it and query it on our own machine, with full access to everything in the database. Sensitive Data Exposure indeed!
That is a big hint for the challenge, so let's briefly cover some of the syntax we would use to query a flat-file database.
The most common (and simplest) format of flat-file database is an *sqlite* database. These can be interacted with in most programming languages, and have a dedicated client for querying them on the command line. This client is

called "*sqlite3*", and is installed by default on Kali.
Let's suppose we have successfully managed to download a database:



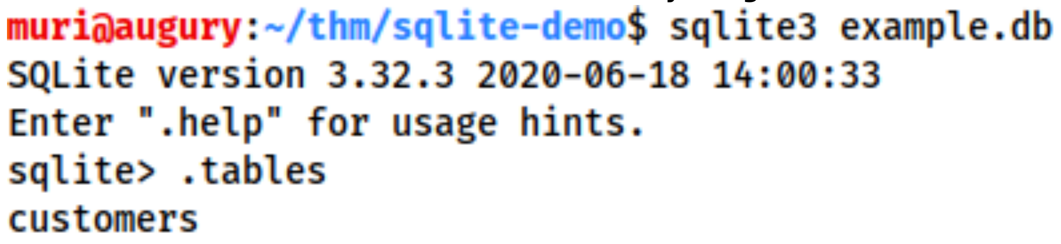
```
muri@augury:~/thm/sqlite-demo$ ls -l
total 16
-rw-r--r-- 1 muri muri 16384 Jul 15 00:42 example.db
muri@augury:~/thm/sqlite-demo$ file example.db
example.db: SQLite 3.x database, last written using SQLite version 3032003
```

We can see that there is an SQlite database in the current folder.
To access it we use: sqlite3 <database-name>:



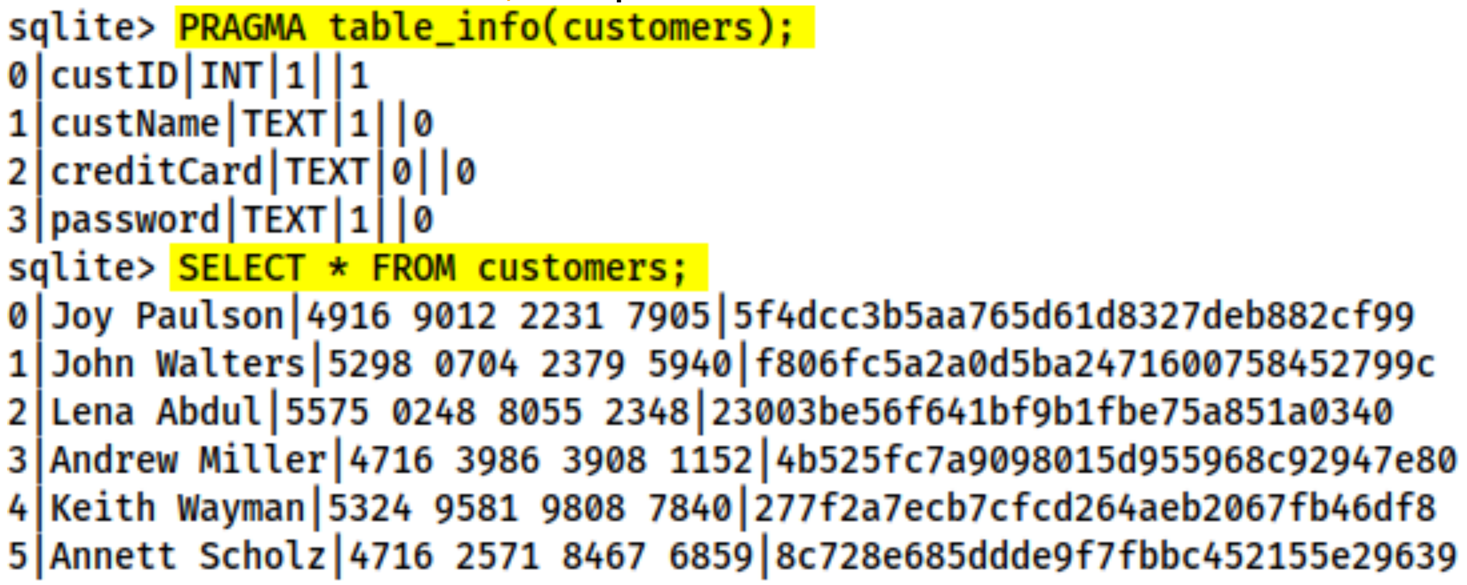
```
muri@augury:~/thm/sqlite-demo$ sqlite3 example.db
SQLite version 3.32.3 2020-06-18 14:00:33
Enter ".help" for usage hints.
sqlite>
```

From here we can see the tables in the database by using the .tables command:



```
muri@augury:~/thm/sqlite-demo$ sqlite3 example.db
SQLite version 3.32.3 2020-06-18 14:00:33
Enter ".help" for usage hints.
sqlite> .tables
customers
```

At this point we can dump all of the data from the table, but we won't necessarily know what each column means unless we look at the table information. First let's use `PRAGMA table_info(customers);` to see the table information, then we'll use `SELECT * FROM customers;` to dump the information from the table:



```
sqlite> PRAGMA table_info(customers);
0|custID|INT|1||1
1|custName|TEXT|1||0
2|creditCard|TEXT|0||0
3|password|TEXT|1||0
sqlite> SELECT * FROM customers;
0|Joy Paulson|4916 9012 2231 7905|5f4dcc3b5aa765d61d8327deb882cf99
1|John Walters|5298 0704 2379 5940|f806fc5a2a0d5ba2471600758452799c
2|Lena Abdul|5575 0248 8055 2348|23003be56f641bf9b1fbe75a851a0340
3|Andrew Miller|4716 3986 3908 1152|4b525fc7a9098015d955968c92947e80
4|Keith Wayman|5324 9581 9808 7840|277f2a7ecb7cfcd264aeb2067fb46df8
5|Annett Scholz|4716 2571 8467 6859|8c728e685ddde9f7fbbc452155e29639
```

We can see from the table information that there are four columns: custID, custName, creditCard and password. You may notice that this matches up with the results. Take the first row:

`0|Joy Paulson|4916 9012 2231 7905|5f4dcc3b5aa765d61d8327deb882cf99`

We have the custID (0), the custName (Joy Paulson), the creditCard (4916 9012 2231 7905) and a password hash (5f4dcc3b5aa765d61d8327deb882cf99).
In the next task we'll look at cracking this hash.

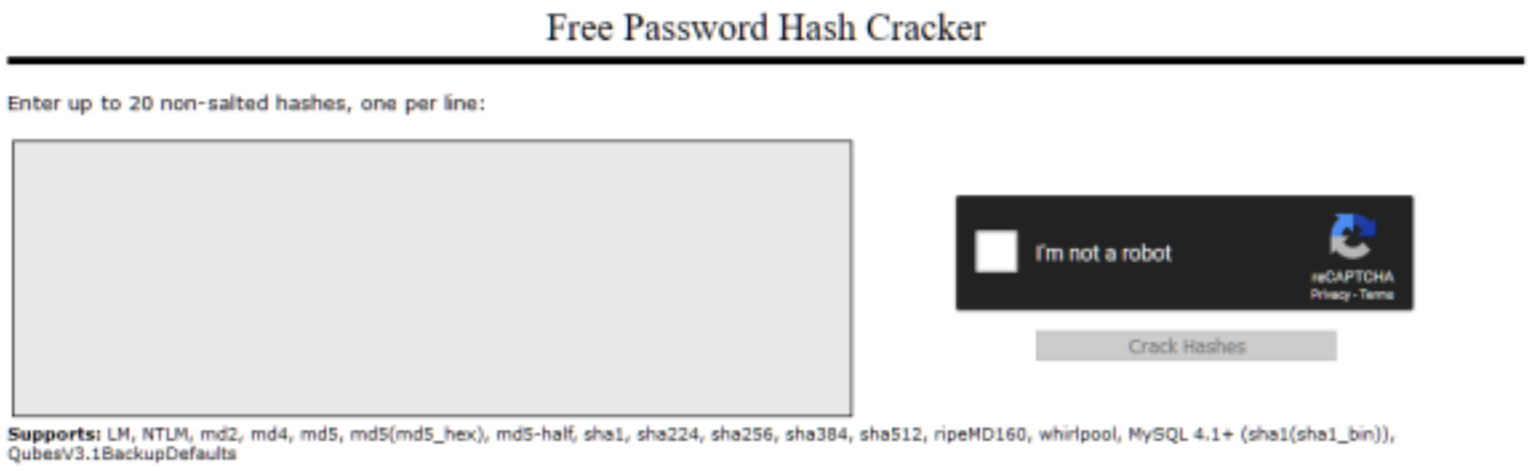
#1
Read and understand the supporting material on SQLite Databases.

No answer needed

[Task 11] [Day 3] Sensitive Data Exposure (Supporting Material 2)

In the previous task we saw how to query an SQLite database for sensitive data. We found a collection of password hashes, one for each user. In this task we will briefly cover how to crack these.
When it comes to hash cracking, Kali comes pre-installed with various tools -- if you know how to use these then feel free to do so; however, they are outwith the scope of this material.

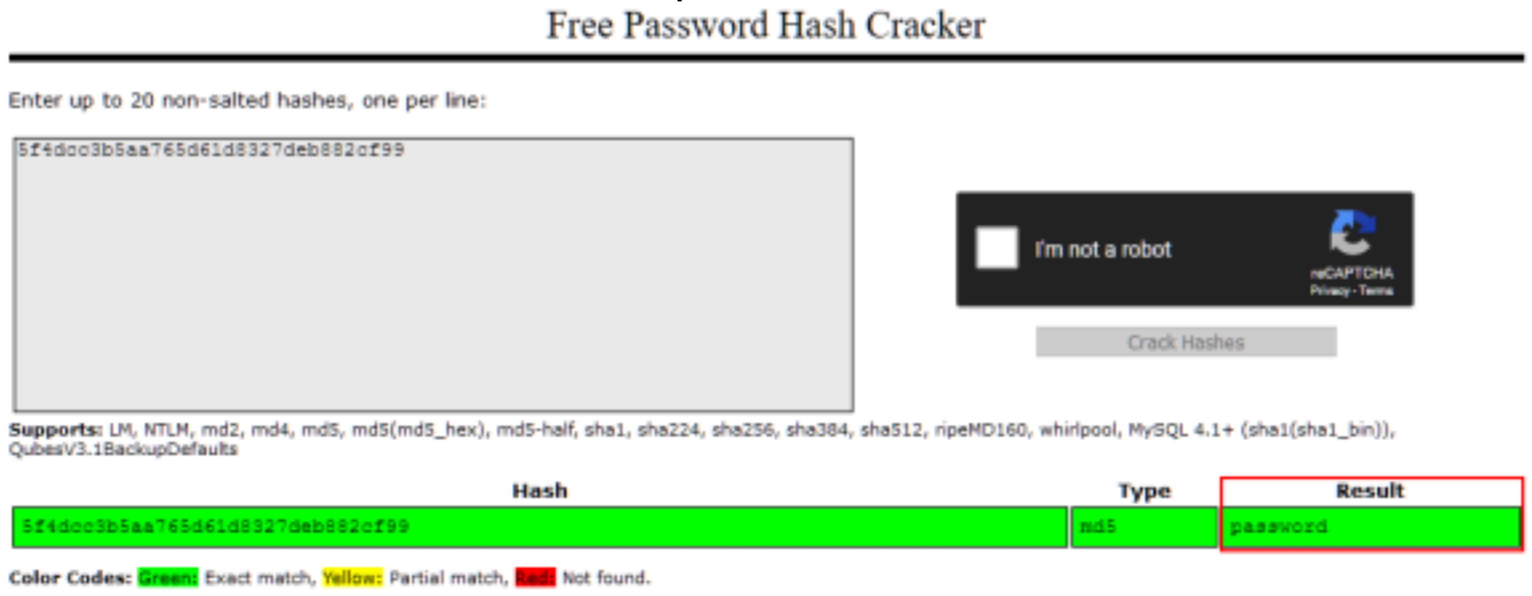
Instead we will be using the online tool: Crackstation. This website is extremely good at cracking weak password hashes. For more complicated hashes we would need more sophisticated tools; however, all of the crackable password hashes used in today's challenge are weak MD5 hashes, which Crackstation should handle very nicely indeed.
When we navigate to the website we are met with the following interface:


Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Let's try pasting in the password hash for Joy Paulson which we found in the previous task (`5f4dcc3b5aa765d61d8327deb882cf99`). We solve the Captcha, then click the "Crack Hashes" button:


Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

```
5f4dcc3b5aa765d61d8327deb882cf99
```

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
5f4dcc3b5aa765d61d8327deb882cf99	md5	password

Color Codes: 🟩 **Green:** Exact match, 🟨 **Yellow:** Partial match, 🟥 **Red:** Not found.

We see that the hash was successfully broken, and that the user's password was "password" -- how secure!
It's worth noting that Crackstation works using a massive wordlist. If the password is not in the wordlist then Crackstation will not be able to break the hash.

The challenge is guided, so if Crackstation fails to break a hash in today's box you can assume that the hash has been specifically designed to not be crackable.

#1
Read the supporting material about cracking hashes.

No answer needed

[Task 12] [Day 3] Sensitive Data Exposure (Challenge)

**It's now time to put what you've learnt into practice with today's challenge.
To spice things up a bit, in addition to the usual daily prize draw this box also harbours a special prize: a voucher for a one month subscription to TryHackMe. There may or may not be another hint hidden on the box, should you need it, but for the time being here's a starting point: boxes are boring, escape 'em at every opportunity.**

IP Address: 10.10.202.147

#1
Have a look around the webapp. The developer has left themselves a note indicating that there is sensitive data in a specific directory. What is the name of the mentioned directory?

/assets

#2
Navigate to the directory you found in question one. What file stands out as being likely to contain sensitive data?

webapp.db

#3
Use the supporting material to access the sensitive data. What is the password hash of the admin user?

6eea9b7ef19179a06954edd0f6c05ceb

#4
Crack the hash. What is the admin's plaintext password?

qwertyuiop

click me
Login as the admin. What is the flag?

THM{Yzc2YjdkMjE5N2VjMzNhOTE3NjdiMjdl}

scans

nmap-quick

```
taj702@kali:~$ nmap -sC -sV 10.10.202.147
Starting Nmap 7.80 ( https://nmap.org ) at 2020-07-16 08:58 EDT
Nmap scan report for 10.10.202.147
Host is up (0.23s latency).
Not shown: 999 closed ports
```

PORT   STATE SERVICE VERSION

```
80/tcp open  http    Apache httpd 2.4.29 ((Ubuntu))
|_http-server-header: Apache/2.4.29 (Ubuntu)
|_http-title: Sense and Sensitivity
```

busters

Gobuster:

```
-----------------------------
===============================================================
[+] Url:            http://10.10.202.147
[+] Threads:        10
[+] Wordlist:       /home/taj702/Desktop/wordlists/dirbuster/directory-list-2.3-medium.txt
[+] Status codes:   200,204,301,302,307,401,403
[+] User Agent:     gobuster/3.0.1
[+] Timeout:        10s
===============================================================
2020/07/16 09:01:20 Starting gobuster
===============================================================
/login          (Status: 301)
/assets         (Status: 301)
/api            (Status: 301)
/console        (Status: 301)
```