# פרויקט סיום

חקירת שרת פגיע

תמר מרגלית פרץ

# תוכן

# מבוא

בדוח זה חקרתי את אבטחת השרת דרך בדיקה של פגיעויות שונות שהתגלו ביישום אינטרנט מבוסס Flask. המטרה הייתה להבין כיצד פגיעויות אלו נגרמות, כיצד ניתן לנצל אותן ומה הסיכונים הנובעים מהן. במהלך החקירה זיהיתי פגיעויות כמו

SQL Injection, SSRF, File Upload, XSS, Path Traversal ו-IFrame Injection.

לכל פגיעות פירטתי את אופן הביטוי שלה בשרת, דרך הניצול שלה וההמלצות לתיקון.

# פגיעות 1 – SQL Injection

## תיאור כללי של הפגיעות:

SQL Injection היא טכניקה שמאפשרת לתוקף להחדיר קוד SQL זדוני למסד הנתונים דרך קלט משתמשים, ובכך לבצע פעולות לא מורשות כמו גישה, שינוי או מחיקת נתונים.

## הביטוי של הפגיעות בשרת:

בטופס החיפוש וההתחברות של השרת, שדות ה-Username וה-Password מועברים ישירות לשאילתות SQL ללא אימות קלט מתאים. כתוצאה מכך, ניתן להזריק קוד SQL דרך שדות אלו.

## מה אפשר לפגיעות להתקיים:

הכשל בקוד הוא חוסר אימות או סינון של קלט המשתמשים לפני שהקלט מועבר למסד הנתונים. זה מאפשר להכניס קוד SQL דרך השדות הללו ולהשפיע על ביצועי השאילתות.

## דרך הניצול:

1. Payload המשמש בטופס ההתחברות:

Username: ' OR '1'='1

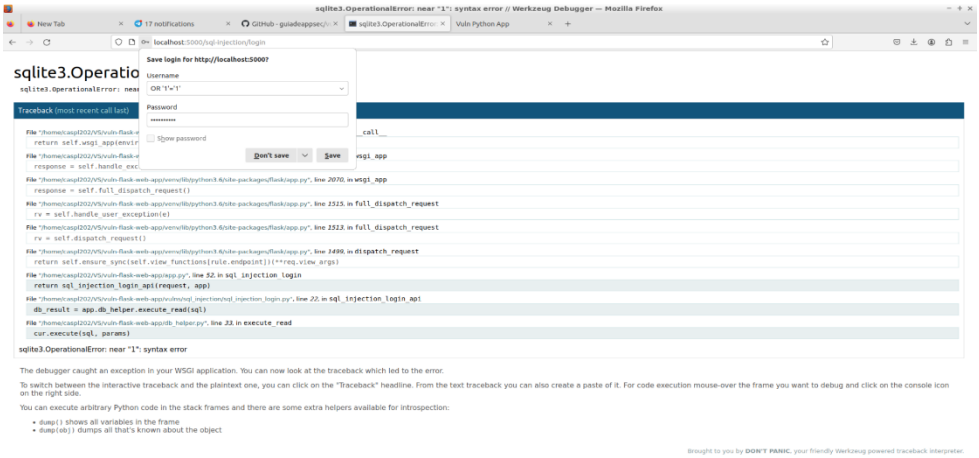
<u>תוצאה בטופס ההתחברות:</u>

- התקלות בשגיאה.

צילום מסך של טופס ההתחברות לפני הזנת הקלט הזדוני:


צילום מסך של הזנת הקלט הזדוני:

צילום מסך של השגיאה שהתקבלה:



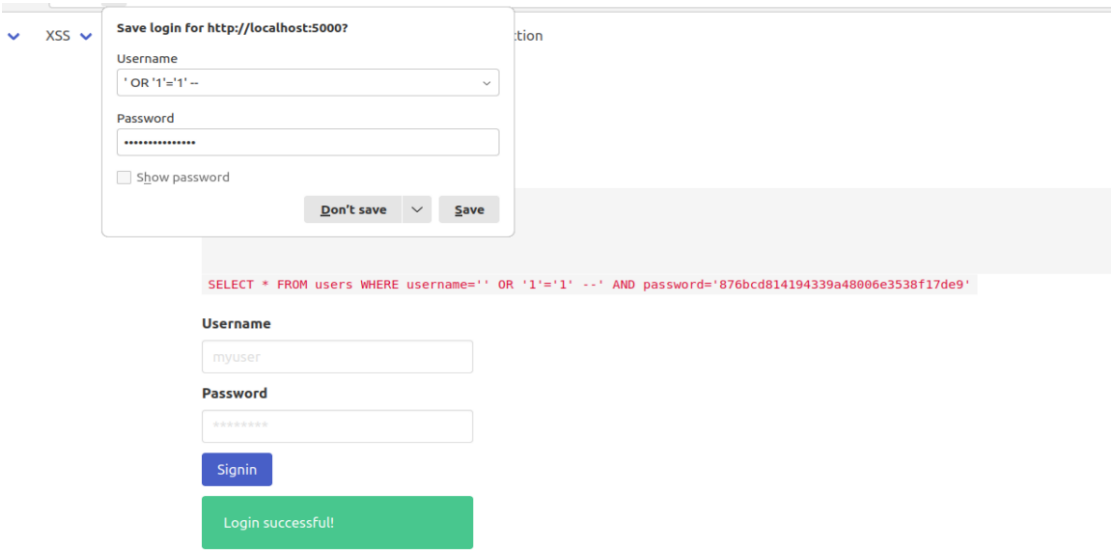

ניסיון שני בטופס ההתחברות:

' OR '1'='1' --

<u>תוצאה בטופס ההתחברות:</u>

- התחברות **מוצלחת** ללא אימות וסיסמא.

צילום של ההתחברות בהצלחה ללא שם משתמש וסיסמא:



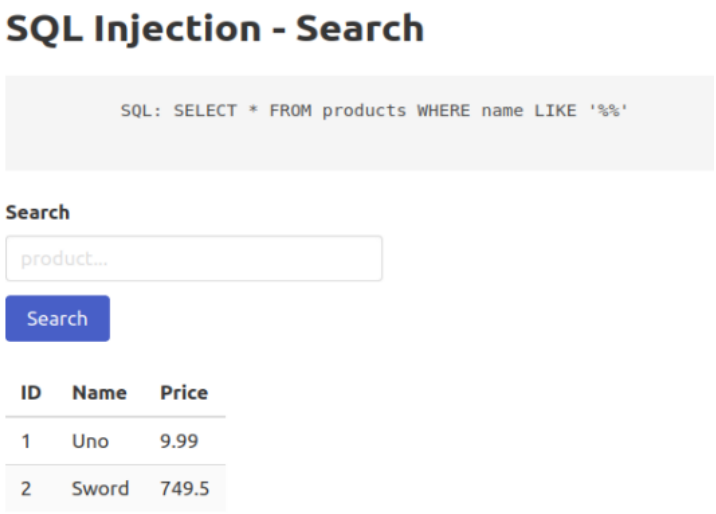
2. Payload המשומש בטופס החיפוש:

' AND 1=2 UNION SELECT 1, username, password FROM users --

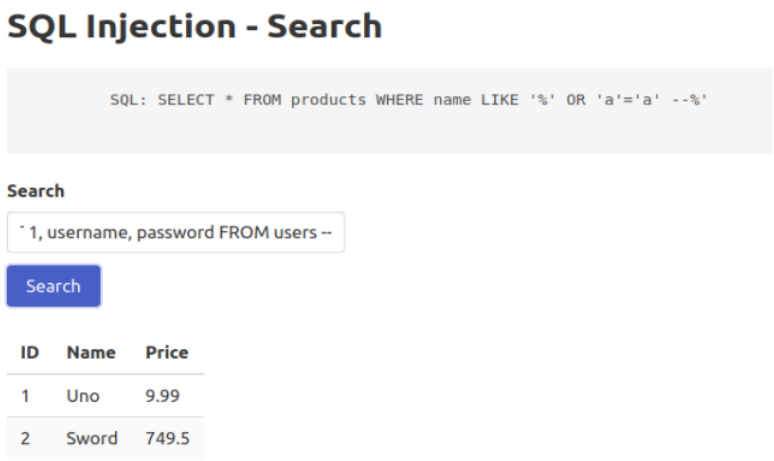
<u>תוצאה בטופס החיפוש:</u>

- קבלת כל שמות המשתמשים והסיסמאות שלהם.

צילום מסך של טופס החיפוש לפני הזנת הקלט הזדוני:




צילום מסך של הזנת הקלט הזדוני:




צילום מסך של התוצאה לאחר לחיצה על כפתור החיפוש:

שאילתות נוספות להוצאת נתונים:

' UNION ALL SELECT 1, username, password FROM users --

תוצאה:

קבלת נתונים רגישים ממסד הנתונים.

צילום מסך של שדות הקלט עם השאילתה הנוספת והנתונים שהתקבלו:

## SQL Injection - Search

```
SQL: SELECT * FROM products WHERE name LIKE '%' UNION SELECT 1, username, password FROM users --%'
```

**Search**

product...

Search

| ID | Name | Price |
|---|---|---|
| 1 | Uno | 9.99 |
| 1 | admin | e64b78fc3bc91bcbc7dc232ba8ec59e0 |
| 1 | robso | b3c634c91e1711c794704a031918a34b |
| 2 | Sword | 749.5 |

שאילתה להוצאת כל הפריטים במערכת:

' OR '1'='1

<u>תוצאה:</u>

קבלת כל הפריטים.

תמונת המוצרים והשאילתה:

```
SQL: SELECT * FROM products WHERE name LIKE '%' OR '1'='1%'
```

**Search**

product...

Search

| ID | Name | Price |
|---|---|---|
| 1 | Uno | 9.99 |
| 2 | Sword | 749.5 |

## המלצות לתיקון:

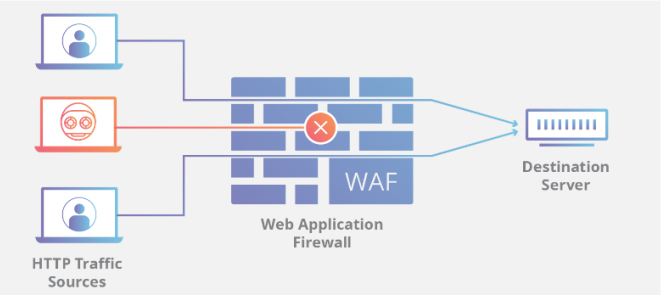
- ברמת הקוד (תיקון לקוד):
  שימוש ב-Prepared Statements שמאמתים את קלט המשתמשים ומונעים הזרקות SQL.
  דוגמת קוד מתוקן :

```
from sqlalchemy import text


query = text("SELECT * FROM products WHERE name=:name")

result = conn.execute(query, name=product_input)
```

- ברמת מערכות הגנה:
  שימוש ב-Web Application Firewall (WAF) כדי לסנן בקשות זדוניות.
  תיאור: תיאור של מערכות ושיטות הגנה נוספות שיכולות למנוע פגיעות SQL Injection.




## מה הסיכונים הגלויים לכל אחת מהפגיעויות:

- תוקף יכול לגשת למידע רגיש כמו שמות משתמשים וסיסמאות.
- תוקף יכול למחוק או לשנות נתונים במסד הנתונים.
- תוקף יכול להשתמש בפגיעות כדי לחדור למערכות נוספות בארגון.

# פגיעות SSRF – 2

## תיאור כללי של הפגיעות:

Server-Side Request Forgery (SSRF) היא טכניקה שמאפשרת לתוקף לשלוח בקשות HTTP מהשרת לצדדים אחרים, תוך ניצול שרת המטרה. תוקף יכול להשתמש בפגיעות זו כדי לגשת למשאבים פנימיים בתוך הרשת של השרת, לקרוא קבצים רגישים, ואפילו לבצע פעולות על שרתים אחרים ברשת.

## הביטוי של הפגיעות בשרת:

בטופס ה-SSRF, השדה Profile Picture URL מועבר ישירות לבקשת HTTP ללא אימות קלט מתאים. כתוצאה מכך, ניתן להזריק כתובת URL פנימית ולהשפיע על הבקשות שנשלחות מהשרת.

## מה אפשר לפגיעות להתקיים:

הכשל בקוד הוא חוסר אימות או סינון של קלט המשתמשים לפני שהקלט מועבר לבקשת HTTP. זה מאפשר להכניס כתובת URL פנימית ולהשפיע על הבקשות שנשלחות מהשרת.

## דרך הניצול:

**Payload המשומש בטופס ה-SSRF:**

http://www.google.com

תוצאה בטופס ה-SSRF:

- השרת ביצע את הבקשה לכתובת החיצונית והוריד את התוכן מהכתובת.

צילום מסך של טופס ה-SSRF לפני הזנת הקלט הזדוני:


צילום מסך של הזנת הקלט הזדוני:

צילום מסך של התוצאה לאחר לחיצה על כפתור "Send":


**Payload נוסף שמשומש בטופס ה-SSRF:**

file:///etc/passwd

<u>תוצאה בטופס ה-SSRF:</u>

השרת ביצע את הבקשה לכתובת הפנימית והציג את תוכן הקובץ.

<u>התוצאה:</u>

## המלצות לתיקון:

- ברמת הקוד (תיקון לקוד):
  שימוש ב-Whitelist של כתובות URL מותרות בלבד.
  דוגמת קוד מתוקן:

```
import re

def is_valid_url(url):
    whitelist = ["example.com", "another-allowed-site.com"]
    for site in whitelist:
        if re.match(f"https?://{site}", url):
            return True
    return False

if is_valid_url(profile_picture_url):
 #    Execute HTTP request
else:
 #    Handle invalid URL
```

- ברמת מערכות הגנה:
  שימוש ב-Firewall ו-WAF -
  שימוש בחומות אש (Firewall) ובמערכות WAF (Web Application Firewall) כדי לסנן בקשות HTTP חשודות ולחסום ניסיונות ניצול. מערכות אלו יכולות לזהות ולחסום בקשות שמנסות לנצל פגיעויות כמו SSRF.

  הגבלת גישה לכתובות פנימיות-
  הגבלת הגישה למשאבים פנימיים כמו כתובות IP פנימיות, פורטים ומשאבים אחרים שאינם אמורים להיות נגישים מבחוץ. ניתן להשתמש בחומת אש פנימית (Internal Firewall) כדי לחסום גישה למשאבים אלו.

  שימוש בכלים לגילוי וניטור
  שימוש בכלים לגילוי וניטור פגיעויות כמו סורקי אבטחה (Security Scanners) וכלי ניתוח סטטיים ודינמיים (Static and Dynamic Analysis Tools) כדי לזהות ולתקן פגיעויות SSRF לפני שהן מנוצלות.

## מה הסיכונים הגלויים לכל אחת מהפגיעויות:

- תוקף יכול לגשת למידע רגיש כמו קבצים פנימיים.
- תוקף יכול להשתמש בפגיעות כדי לבצע בקשות לא מורשות.
- תוקף יכול להשתמש בפגיעות כדי לחדור למערכות נוספות בארגון.

# פגיעות 3 – File Upload

## תיאור כללי של הפגיעות:

פגיעות בהעלאת קבצים מאפשרת לתוקף להעלות קבצים זדוניים לשרת, ולאחר מכן להפעיל אותם על השרת. זה יכול לכלול קבצי סקריפטים (כמו PHP, Python, וכו') שמבצעים פעולות זדוניות על השרת.

## הביטוי של הפגיעות בשרת:

בטופס העלאת הקבצים, השרת מקבל את הקובץ שהמשתמש מעלה ללא אימות מתאים. כתוצאה מכך, ניתן להעלות קובץ זדוני ולהריץ אותו על השרת.

## מה אפשר לפגיעות להתקיים:

הכשל בקוד הוא חוסר אימות או סינון של הקבצים המועלים לשרת. זה מאפשר להעלות קבצים עם סיומות מסוכנות כמו .py, .php וכו'.

## דרך הניצול:

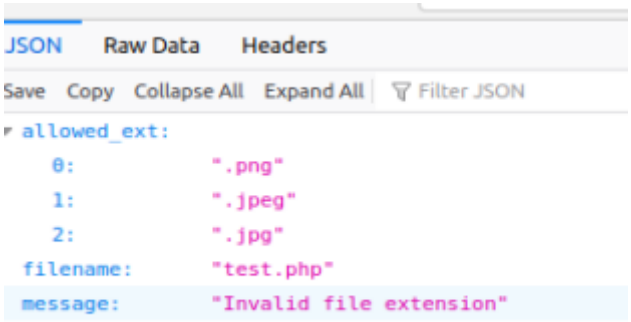
**Payload המשומש בטופס העלאת הקבצים:**

קובץ test.php:

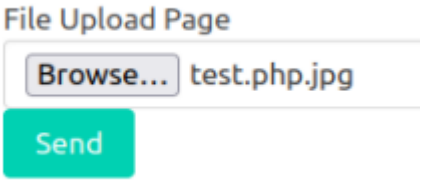
```
<?php phpinfo(); ?>
```

<u>תוצאה בטופס העלאת הקבצים:</u>

- גילינו כי המערכת מסננת קבצים על פי סיומת.




**ניסיון לעקוף את הסינון בעזרת סיומת כפולה:** test.php.jpg




<u>תוצאה בטופס העלאת הקבצים:</u>

- עורר שגיאה:

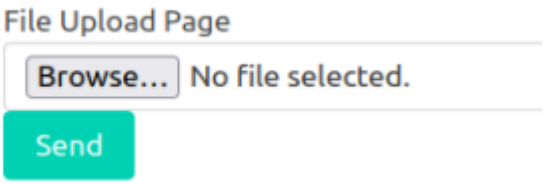
אז שיניתי כיוון וכעת ננסה דרך אחרת:

הבנתי שחסרה לי תקיית Upload אז הוספתי –

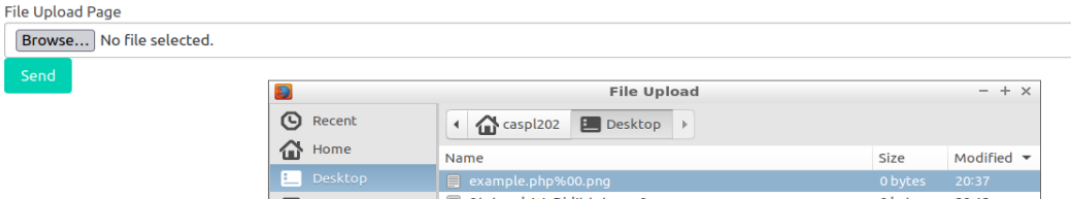
```
caspl202@caspl202-lubuntu:~/VS/vuln-flask-web-app$ ls -ld /home/caspl202/VS/vuln-flask-web-app/temp/uploads/
ls: cannot access '/home/caspl202/VS/vuln-flask-web-app/temp/uploads/': No such file or directory
caspl202@caspl202-lubuntu:~/VS/vuln-flask-web-app$ mkdir -p /home/caspl202/VS/vuln-flask-web-app/temp/uploads/
caspl202@caspl202-lubuntu:~/VS/vuln-flask-web-app$ chmod 777 /home/caspl202/VS/vuln-flask-web-app/temp/uploads
```

**ניסיון תקיפה עם שם קובץ זדוני:**

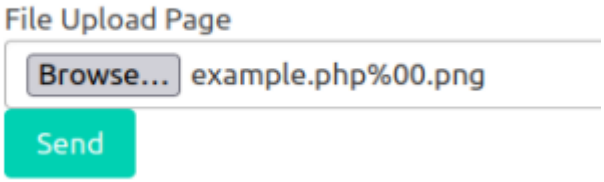
קובץ בשם example.php%00.png



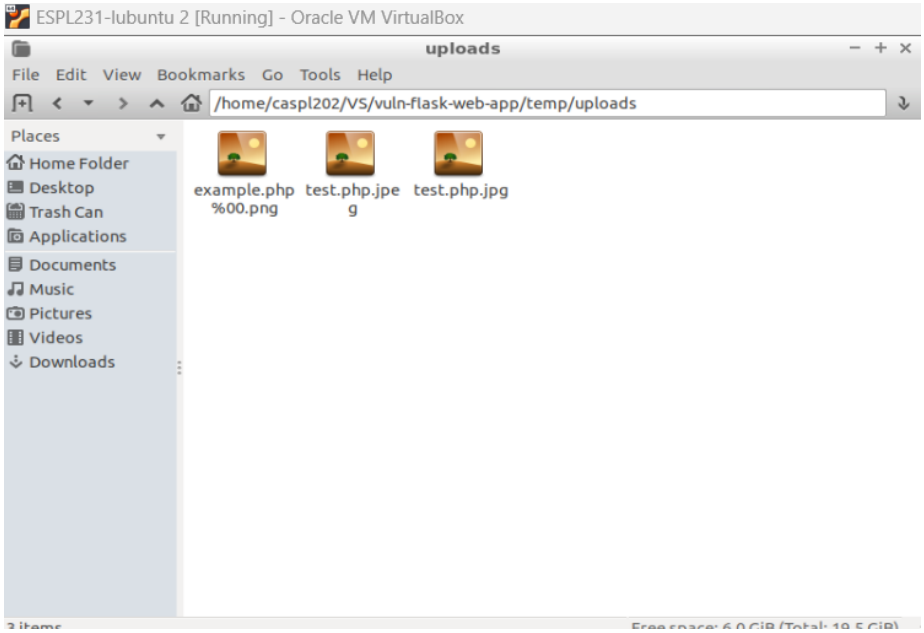

נלחץ על Brows:




נבחר את הקובץ שיצרנו:




נראה כי הקובץ התווסף כהלכה לUploade :



## המלצות לתיקון:

- ברמת הקוד (תיקון לקוד):
  אימות וסינון קפדניים של קבצים לפני השימוש בהם.
  שימוש בפונקציות מובנות לניהול קבצים שמונעות מניפולציה של שמות קבצים.
  דוגמת קוד מתוקן:

```
def secure_upload(file):
    base_dir = "/secure/uploads"
    secure_path = os.path.join(base_dir, os.path.normpath(file.filename))
    if os.path.commonprefix([secure_path, base_dir]) == base_dir:
        file.save(secure_path)
        return "File uploaded successfully"
    else:
        return "Invalid file path"
```

- ברמת מערכות הגנה:
  שימוש ב-Web Application Firewall (WAF) כדי לסנן בקשות זדוניות ולמנוע העלאה של קבצים לא מורשים.
  הגבלת גישה לתיקיות רגישות בעזרת הגדרות אבטחה במערכת ההפעלה ובשרת.

## מה הסיכונים הגלויים לכל אחת מהפגיעויות:

- תוקף יכול לגשת למידע רגיש כמו קבצים פנימיים.
- תוקף יכול להשתמש בפגיעות כדי לבצע בקשות לא מורשות.
- תוקף יכול לשנות קבצים קריטיים בקבצים זדוניים.
- התוקף מנצל את היכולת הזאת של העלאת הקבצים בידי המשתמש, והעובדה שהשרת הפגיע לא עורך בדיקות לאותם קבצים, כדי להעלות קובץ זדוני. הקבצים הזדונים שהתוקף מעלה יכולים להיות קבצים עם סקריפטים זדוניים המאפשרים לו לבצע פעולות מרחוק.

# פגיעות 4 – XSS

## תיאור כללי של הפגיעות:

XSS (Cross-Site Scripting) היא פגיעות אבטחה שמאפשרת לתוקף להזריק קוד זדוני (כגון JavaScript) לתוך עמודי אינטרנט הניגשים למשתמשים אחרים. הקוד הזדוני יכול לרוץ בדפדפן של הקורבן ולקבל גישה למידע רגיש כמו עוגיות או פרטי הזדהות.

## הביטוי של הפגיעות בשרת:

בטפסים שמציגים תוכן שהוזן על ידי המשתמש בלי לנקות או לאמת את התוכן כראוי, ניתן להזריק סקריפטים זדוניים שירוצו בדפדפן של משתמשים אחרים.

## מה אפשר לפגיעות להתקיים:

חוסר סינון וניקוי של הקלטים שמוזנים לטפסים. הצגת התוכן שהוזן למשתמשים אחרים בלי להגן עליו כראוי.

## XSS Reflected - דרך הניצול:

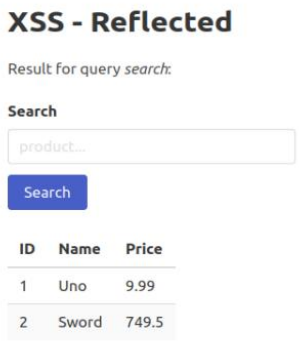
**Payload המשומש בטופס החיפוש:**

```
<script>alert('XSS')</script>
```

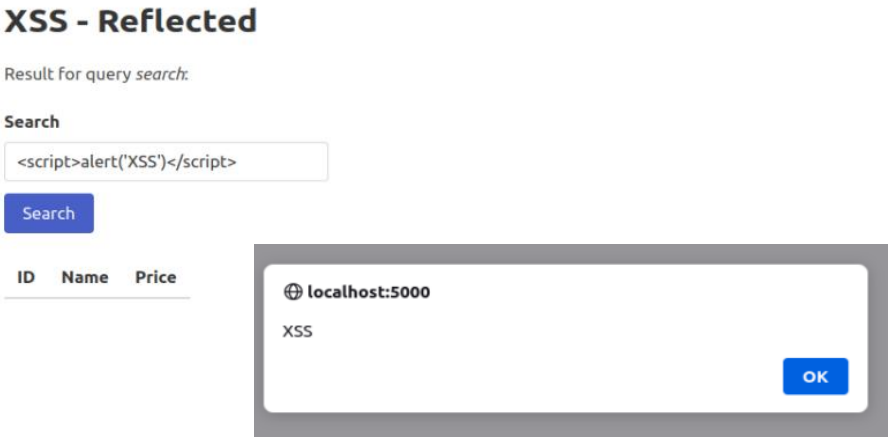
<u>תוצאה בטופס החיפוש:</u>

- הקוד שהוזן ירוץ מיד עם שליחת הטופס ויציג הודעת "alert" בדפדפן.

צילום מסך של הטופס לפני הכנסת הקוד:




צילום מסך של התוצאה לאחר לחיצה על כפתור "Search" והופעת ההודעה.



## XSS Stored דרך הניצול:

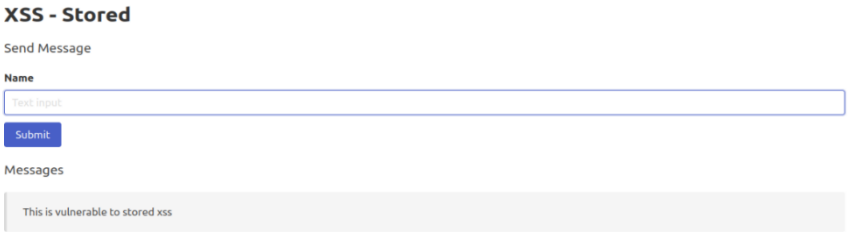
<u>Payload המשומש בטופס השם:</u>

<script>alert('XSS')</script>

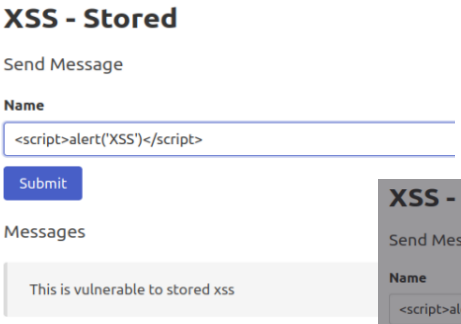
<u>תוצאה בטופס השם:</u>

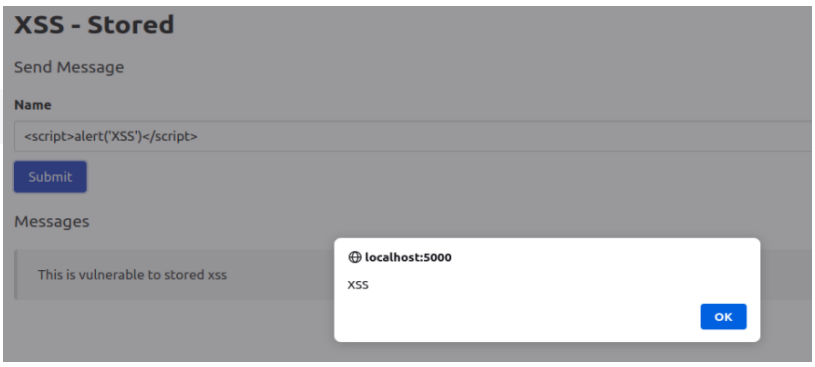
- הקוד שהוזן יישמר בבסיס הנתונים וירוץ בכל פעם שהעמוד ייטען.

צילום מסך של הטופס לפני הכנסת הקוד:

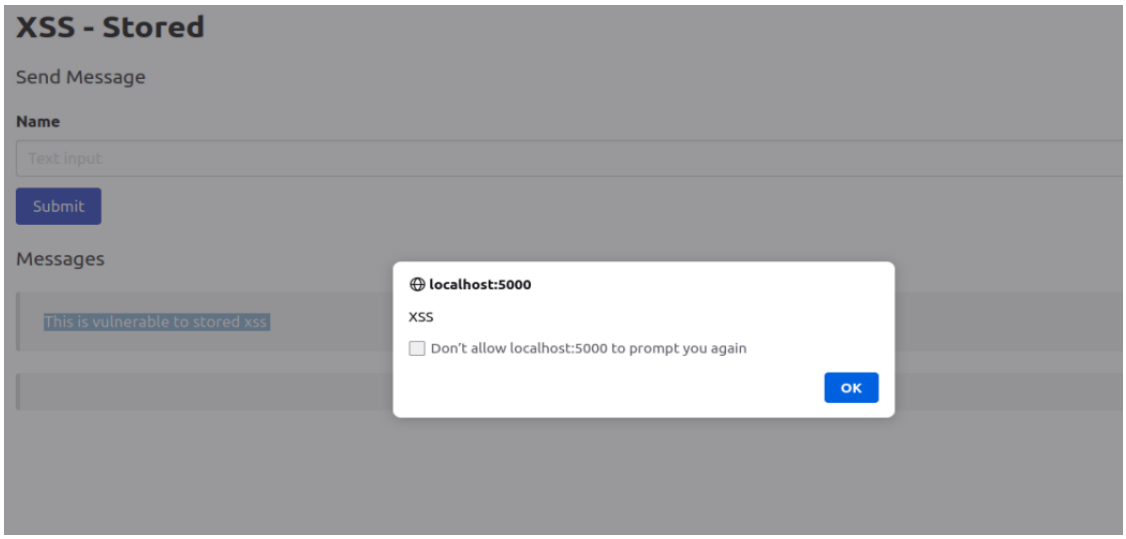



צילום מסך של התוצאה לאחר לחיצה על כפתור "Submit" והופעת ההודעה בכל טעינה של העמוד:




ואם אלחץ רענן זה יופי שוב:



## המלצות לתיקון:

- ברמת הקוד (תיקון לקוד):
  אימות וסינון קפדניים של הקלטים. ניקוי הקלט לפני הצגתו למשתמשים אחרים.שימוש בפונקציות מובנות לניקוי קלט.
  דוגמת קוד מתוקן:

```
from markupsafe import escape

@app.route('/search')
def search():
    query = request.args.get('product')
    safe_query = escape(query)
#    Use safe_query in the HTML template
```

- ברמת מערכות הגנה:
  שימוש ב-Content Security Policy (CSP) כדי להגביל הרצת סקריפטים לא מאושרים.
  תיאור: שימוש בכותרות HTTP כמו X-XSS-Protection.

- Content Security Policy (CSP) היא מנגנון אבטחה מבוסס HTTP headers שמטרתו להפחית את הסיכון של התקפות מבוססות Cross-Site Scripting (XSS) והזרקות נתונים אחרות. CSP מאפשר למנהל האתר להגדיר מדיניות שמגבילה את המקורות שממנו ניתן לטעון סקריפטים, סגנונות, תמונות וכדומה

## מה הסיכונים הגלויים לכל אחת מהפגיעויות:

- גניבת עוגיות ופרטי הזדהות של משתמשים.
- ביצוע פעולות לא מורשות בשם המשתמשים.
- הפצת תוכן זדוני למשתמשים אחרים.

# פגיעות 5 - Path Traversal

## תיאור כללי של הפגיעות:

Path Traversal היא פגיעות שמאפשרת לתוקף לגשת לקבצים ומידע רגיש על השרת דרך מניפולציה של נתיבי קבצים בקלט המשתמש. בכך ניתן לגשת למידע שלא אמור להיות נגיש מבחוץ, כמו קבצי קונפיגורציה או קבצים המכילים מידע אישי.

## הביטוי של הפגיעות בשרת:

בטופס ה-Path Traversal, השדה שבו מתבצעת ההזנה מאפשר גישה ישירה לנתיבי קבצים.
כתוצאה מכך, ניתן להזין נתיבים זדוניים ולגשת לקבצים רגישים בשרת.

## מה אפשר לפגיעות להתקיים:

הכשל בקוד הוא חוסר אימות או סינון של נתיבי קבצים לפני השימוש בהם. זה מאפשר לתוקף להזין נתיבי קבצים זדוניים ולהשיג גישה לקבצים רגישים.

## דרך הניצול:

**Payload המשומש בטופס Path Traversal:**

ה - URL המקורי:

http://localhost:5000/path-traversal-img?img=84721189311536093217.jpg

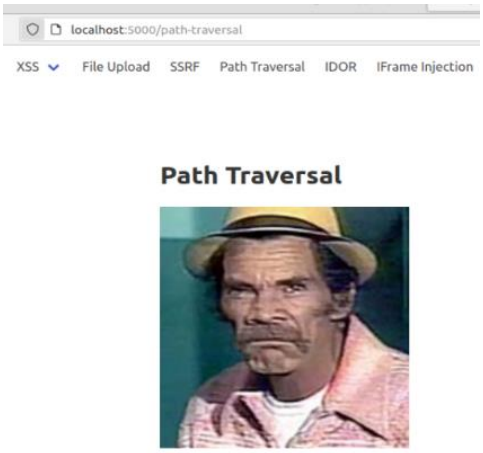
ה - URL הזדוני:

http://localhost:5000/path-traversal-img?img=../../../../../../etc/passwd

תוצאה בטופס Path Traversal:

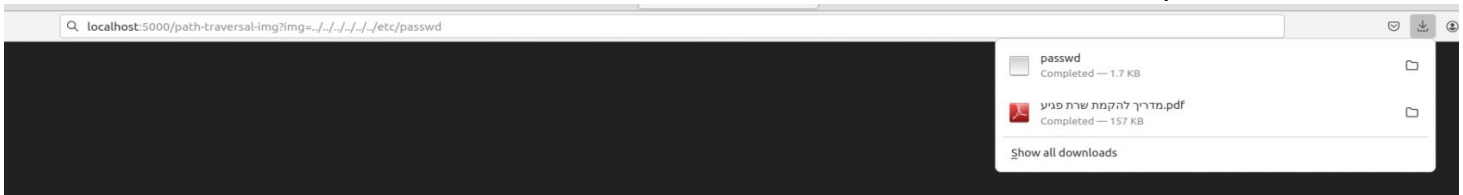
- קבלת גישה לקובץ המערכת `passwd` המציג את תוכן הקובץ, כולל שמות משתמשים ומידע נוסף על המערכת.

צילום מסך של טופס Path Traversal לפני הזנת הקלט הזדוני:




צילום מסך של הזנת הקלט הזדוני:


צילום מסך של התוצאה לאחר לחיצה על כפתור שליחה:



הקובץ שקיבלנו passwd.txt :

```
File  Edit  Search  Options  Help
oot:x:0:0:root:/root:/bin/bash
laemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
in:x:2:2:bin:/bin:/usr/sbin/nologin
ys:x:3:3:sys:/dev:/usr/sbin/nologin
ync:x:4:65534:sync:/bin:/bin/sync
ames:x:5:60:games:/usr/games:/usr/sbin/nologin
an:x:6:12:man:/var/cache/man:/usr/sbin/nologin
p:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
ail:x:8:8:mail:/var/mail:/usr/sbin/nologin
ews:x:9:9:news:/var/spool/news:/usr/sbin/nologin
ucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
roxy:x:13:13:proxy:/bin:/usr/sbin/nologin
ww-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
ackup:x:34:34:backup:/var/backups:/usr/sbin/nologin
st:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
rc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
nats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
obody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
ystemd-network:x:100:102:systemd Network Management,,,:/run/systemd/netif:/usr/sbin/nologin
ystemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd/resolve:/usr/sbin/nologin
yslog:x:102:106::/home/syslog:/usr/sbin/nologin
essagebus:x:103:107::/nonexistent:/usr/sbin/nologin
apt:x:104:65534::/nonexistent:/usr/sbin/nologin
uidd:x:105:110::/run/uuidd:/usr/sbin/nologin
ightdm:x:106:113:Light Display Manager:/var/lib/lightdm:/bin/false
whoopsie:x:107:117::/nonexistent:/bin/false
ernoops:x:108:65534:Kernel Oops Tracking Daemon,,,:/:/usr/sbin/nologin
ulse:x:109:119:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
vahi:x:110:121:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/usr/sbin/nologin
plip:x:111:7:HPLIP system user,,,:/var/run/hplip:/bin/false
aspl202:x:1000:1000:caspl202,,,:/home/caspl202:/bin/bash
boxadd:x:999:1::/var/run/vboxadd:/bin/false
```

## המלצות לתיקון:

- ברמת הקוד (תיקון לקוד):
  אימות וסינון קפדניים של נתיבי קבצים לפני השימוש בהם.
  שימוש בפונקציות מובנות לניהול נתיבי קבצים שמונעות מניפולציה של נתיבים.
  דוגמת קוד מתוקן:

```
import os

def secure_open_file(file_path):

    base_dir = "/secure/directory"

    secure_path = os.path.join(base_dir, os.path.normpath(file_path))

    if os.path.commonprefix([secure_path, base_dir]) == base_dir:

        with open(secure_path, 'r') as file:

            return file.read()

    else:

        return "Invalid file path"
```

- ברמת מערכות הגנה:
  שימוש ב-Web Application Firewall (WAF) כדי לסנן בקשות זדוניות ולמנוע גישה לנתיבים לא מורשים.
  הגבלת גישה לנתיבי קבצים רגישים בעזרת הגדרות אבטחה במערכת ההפעלה ובשרת.

## מה הסיכונים הגלויים לכל אחת מהפגיעויות:

- תוקף יכול לגשת למידע רגיש כמו קבצים פנימיים.
- תוקף יכול להשתמש בפגיעות כדי לבצע בקשות לא מורשות.
- תוקף יכול להשתמש בפגיעות כדי לחדור למערכות נוספות בארגון.

# פגיעות 6 - IFrame Injection

## תיאור כללי של הפגיעות:

IFrame Injection היא פגיעות שמאפשרת לתוקף להכניס תוכן זדוני לאתר על ידי החדרת URL לתוך אלמנט IFrame . זה מאפשר לתוקף להציג תוכן זדוני מהאתר שלו בתוך אתר לגיטימי.

## הביטוי של הפגיעות בשרת:

בטופס ה-IFrame Injection, השדה שבו מתבצעת ההזנה מאפשר הכנסה של כתובת URL.
כתוצאה מכך, ניתן להזין כתובות זדוניות ולהציג תוכן זדוני בתוך האתר.

## מה אפשר לפגיעות להתקיים:

הכשל בקוד הוא חוסר אימות או סינון של כתובת ה-URL לפני השימוש בה. זה מאפשר לתוקף להזין כתובת URL זדונית ולהציג תוכן זדוני באתר.

## דרך הניצול:

Payload המשומש בטופס IFrame Injection:

ה - URL המקורי:

http://localhost:5000/iframe-injection?page=/static/pages/about.html

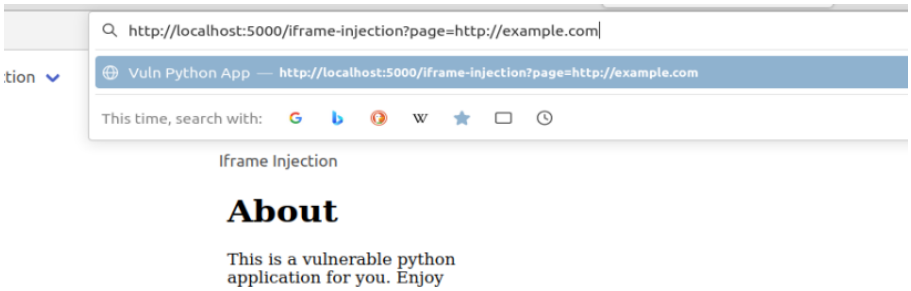
ה - URL הזדוני:

http://localhost:5000/iframe-injection?page=http://example.com

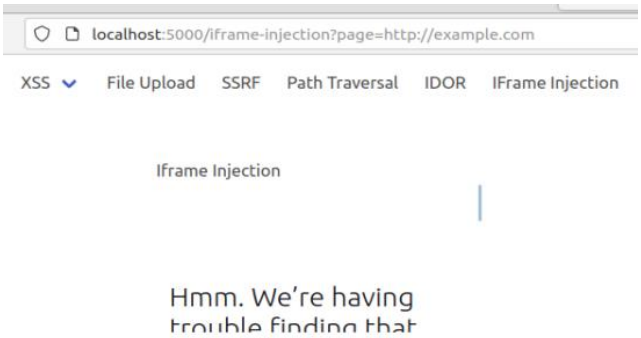
- תוצאה בטופס IFrame Injection:
  הכנסת תוכן מדומיין example.com בתוך מסגרת iframe.

צילום מסך של טופס IFrame Injection לפני הזנת הקלט הזדוני:


צילום מסך של הזנת הקלט הזדוני:



צילום מסך של התוצאה לאחר לחיצה על כפתור שליחה:




## המלצות לתיקון:

- ברמת הקוד (תיקון לקוד):
  אימות וסינון קפדניים של כתובת ה-URL לפני השימוש בה.
  הגבלת הכתובות שניתן להזין לאלמנט ה-iframe לכתובות פנימיות בלבד.
  דוגמת קוד מתוקן:

```
from urllib.parse import urlparse

def is_valid_url(url):

    parsed_url = urlparse(url)

    return parsed_url.netloc in allowed_domains

def iframe_injection_page(request, app):

    iframe_url = request.args.get('page')

    if not is_valid_url(iframe_url):

        return "Invalid URL", 400

    return render_template("iframe_injection.html", iframe_url=iframe_url)
```

- ברמת מערכות הגנה:
  שימוש ב-Web Application Firewall (WAF) כדי לסנן בקשות זדוניות ולמנוע הכנסה של כתובות URL זדוניות.
  שימוש בכותרות אבטחה (Content Security Policy) כדי להגביל את התוכן שניתן להכניס לאתר.

## מה הסיכונים הגלויים לכל אחת מהפגיעויות:

- תוקף יכול להציג תוכן זדוני בתוך האתר הלגיטימי.
- תוקף יכול להשתמש בפגיעות כדי להטעין את המשתמשים בפישינג או קוד זדוני.

# סיכום

בדוח זה חקרתי שש פגיעויות אבטחה שונות ביישום אינטרנט מבוסס Flask. פגיעויות אלו כוללות SQL Injection, SSRF, File Upload, XSS, Path Traversal ו-IFrame Injection. כל פגיעות נותחה לפרטי פרטים, כולל התיאור שלה, הביטוי שלה בשרת, דרך הניצול שלה והמלצות לתיקון.

הממצאים מראים את החשיבות של אימות וסינון קלטים ביישומי אינטרנט, שימוש בכלי אבטחה כמו WAF ו-CSP, והגבלת גישה למשאבים רגישים. הדוח מדגיש את הצורך בהקשחת אבטחת היישומים כדי למנוע ניצול פגיעויות ופגיעה בנתונים רגישים ובמערכות קריטיות.