

## דו"ח אבטחת סיסמאות

מגישים : עומר רשף, ת.ז. 314651266  
ליאור צמח, ת.ז. 207381435

תוכן עניינים :

2	מבוא על מנגנוני אחסון סיסמאות .....
2	תיאור השרת וההגנות .....
3	תיעוד הקונפיגורציה .....
5	סימולציית תקיפות וניתוח נתונים .....
6	נסיונות כושלים לאורך תהליך הפיתוח .....
6	תוצאות הניסויים וניתוחם .....
8	תיעוד התקיפות בקבצי לוג .....
8	עמידה בכללי אתיקה .....
9	מסקנות והמלצות .....

## מבוא על מנגנוני אחסון סיסמאות

אחסון מאובטח של סיסמאות הוא מרכיב קריטי בהגנה על משתמשים ומערכות. המטרה היא לשמור סיסמאות באופן שאינו חושף את הסיסמה עצמה גם אם מאגר הנתונים נפרץ. אחסון סיסמאות בטקסט גלוי מהווה סיכון חמור, שכן דליפת בסיס נתונים כזה תחשוף מיד את סיסמאות המשתמשים. במקום זאת, נהוג לשמור ערכי גיבוב (hash) של הסיסמאות וכך לוודא שלא ניתן לשחזר מההash את הסיסמה המקורית.

בעת אימות סיסמה, המערכת מבצעת גיבוב לסיסמה שהוזנה ומשווה לערך השמור במאגר. גישה זו מונעת מאדם זדוני לקבל את הסיסמה עצמה גם אם הצליח להשיג את ה-hash.

עם זאת, גם שמירת hash בלבד אינה מספקת, וכדי לחזק את האבטחה נהוג לשלב טכניקות נוספות.

## תיאור השרת וההגנות

היישום שנבנה ב-Flask במסגרת הפרויקט מממש שרת אינטרנט לאימות משתמשים, עם מספר מנגנוני הגנה משולבים שנועדו להגן מפני פריצות ותקיפות על מנגנון ההתחברות. להלן המנגנונים העיקריים שהוטמעו בשרת, תפקידם ואופן השתלבותם:

### Salt

משמעותו הוספת ערך אקראי וייחודי לכל סיסמה לפני hash, כך שאפילו אם שני משתמשים בוחרים באותה סיסמה ערכי ההאש שהם מקבלים יהיו שונים לחלוטין.

ה-Salt נשמר לצד ההאש במסד הנתונים, ותפקידו העיקרי לסכל התקפות באמצעות Rainbow Tables ולהקשות על זיהוי סיסמאות זהות של משתמשים שונים.

### Pepper

לעיתים נעשה שימוש ב-pepper ערך סודי גלובלי הנוסף לסיסמה לפני hash. בניגוד ל-salt את pepper לא שומרים במסד הנתונים יחד עם ההאש, אלא שומרים אותו בנפרד כמשתנה גלובי (למשל בקובץ קונפיגורציה או כערך סביבה בסביבה מאובטחת).

ה-pepper המשותף לכל הסיסמאות מוסיף שכבת סודיות נוספת: תוקף שמצליח לקבל גישה למסד הנתונים ולערוך ההאש וה-salt עדיין יידרש לו לגלות את ערך pepper הסודי כדי לפענח את הסיסמאות ובכך שילוב של hash עם pepper salt מגן גם מפני התקפות מבוססות מאגרי האשים מוכנים מראש וגם מפני ניסיונות brute-force לאחר דליפת נתונים.

## TOTP – Time Based One-Time Password

מעבר לאופן האחסון של סיסמאות, קיים צורך להתמודד עם מצב שבו סיסמה עלולה להתגלות או להיגנב.

טכניקת TOTP סיסמא חד פעמית מתוזמנת – מספקת גורם שני לאימות משתמש. המשמעות היא שלא מספיק סיסמא סטטית כדי להיכנס אלא נדרשת גם קוד חד פעמי המשתנה מדי זמן קצוב (לרוב 30 שניות) באמצעות אפליקציה או התקן ייעודי (כמו למשל גוגל אוטנטיקייטור).

במקרה כזה גם אם הסיסמא הודלפה התוקף יידרש לספק גם קוד חד פעמי של הזמן הנוכחי, ששונה בכל ניסיון כניסה מה שמקשה על פריצת החשבון. השילוב של שמירת סיסמאות בצורה מאובטחת יחד עם אימות דו שלבי מספק מענה מקיף יותר להגנת חשבונות משתמשים.

## CAPTCHA

מנגנון המציג למשתמש אתגר שבני אדם יכולים לפתור יחסית בקלות אך תוכנות אוטומטיות יתקשו להתמודד איתו. בשרת שלנו, טופס ההתחברות ידרוש מהשתמש לעבור מבחן captcha (במציאות זה יכול להיות מבחן של בחירת תמונה מתאימה או פתרון תרגיל פשוט אך במימוש שלנו אנחנו רק מדמים captcha) לפני שיוכל לנסות להתחבר. במימוש שלנו captcha היא טוקן סודי שניתן לייצור בעזרת פניה לשרת Rest API עם נקודת הקצה admin/get\_captcha\_token/ שמייצר טוקן סודי ושומרת אותו בשרת. המשתמש בהתחברות יכניס את הטוקן הזה וכך השרת יאמת שהטוקן שהזין זהה לטוקן ששמור אצלו.

שילוב CAPTCHA מקשה על תוקף אוטומטי מפני שגם אם אינו נחסם לחלוטין, הוא נדרש למאמץ אנושי בכל ניסיון נוסף.

## Rate Limiting – הגבלת קצב הפניות

מנגנון שמגביל את כמות ניסיונות ההתחברות או הבקשות שניתן לבצע בפרק זמן נתון. מטרתו למנוע ניסיונות Brute force על ידי האטת קצב הניחושים כמו למשל, הגבלה של מספר ניסיונות התחברות כושלים לדקה לכל משתמש. בשרת שלנו, מנגנון זה משולב כך שאם מתקבלות יותר מדי בקשות התחברות ברצף תוך זמן קצר, השרת

מעכב או חוסם זמנית בקשות נוספות מאותו גורם. ללא הגבלת קצב כלל, יישום מזמין מתקפה כיוון שתוקף יכול לנסות סיסמאות ללא הפרעה. עם מנגנון זה התקפה אוטומטית מהירה תהפוך ללא ישימה משום שהשרת גביל את התדירות ומעלה את משך הזמן שנדרש לניחוש מספר רב של סיסמאות.

## Account Lockout – נעילת חשבון לאחר ניסיונות כושלים

מנגנון הנעילה חוסם חשבון משתמש לאחר מספר מוגדר של ניסיונות כניסה כושלים. תפקידו להגן על חשבונות ספציפיים מפני ניסיונות חוזרים לניחוש סיסמה נכונה.

בשרת שבנינו לאחר סף מוגדר (5 ניסיונות כושלים ברצף) החשבון יינעל למשך תקופה מסויימת (5 דקות). המימוש משולב בקוד ההתחברות כך שבכל ניסיון כושל עולה מונה כשלונות וכשמגיעים ל-5 לא ניתן להתחבר גם תוכנס הסיסמה הנכונה. רק לאחר 5 דקות ניתן לנסות שוב. מנגנון זה עוזר לוודא שתוקף לא יוכל לנסות כמויות גדולות של סיסמאות על חשבון ספציפי ברצף.

## TOTP – Time Based One-Time Password

כמוסבר במבוא המנגנון מספק גורם שני לאימות. בשרת שלנו מנגנון זה פועל לאחר השלושה הקודמים ולאחר שהוזנה סיסמה תקינה ואז המערכת דורשת גם את הקוד החד פעמי הנוצר באפיקציה ייעודית (במקרה שלנו בפייתון בעזרת ספריית pyotp). בעת ההרשמה לכל משתמש מוגדר מפתח סודי עבור ה-TOTP הנשמר במסד הנתונים ומשוך לאפליקצית האימות של המשתמש. בעת ניסיון הכניסה השרת מחשב את הקוד העדכני המתאים לשעה ולמפתח הסודי של המשתמש ומשווה אותו לקוד שסיפק המשתמש. רק אם שני הקודים זהים ובתוקף (בדרך כלל תקף 30 שניות) הכניסה תאושר.

המנגנון משולב בזרימת ההתחברות: לאחר אימות סיסמה מוצלח, משתמש שלא הגיש קוד נכון לא יקבל גישה מלאה. בכך, גם אם סיסמה דלפה, חשבון המשתמש נותר מוגן כל עוד התוקף אינו מחזיק גם במכשיר הפיזי או באפליקציה המכילה את המפתח הסודי ליצירת הקודים.

## Hash + Salt + PEPPER

מנגנון קריטי בבסיס השרת הוא אופן שמירת הסיסמאות עצמם במסד הנתונים. כפי שצויין במבוא, סיסמאות אינן נשמרות כטקסט גלוי אלא כערכי האש קריפטוגרפיים. במימוש שלנו עבור כל סיסמה חדשה נוצר salt בגודל 8 בתים אקראיים המשולב בסיסמה ובנוסף אליו משורשר pepper הסודי המוגדר כמשתנה גלובלי בשרת. בפועל בעת ההרשמה המשתמש בוחר סיסמה שעוברת שרשרת עם salt וה-pepper ואז נשלחת לפעולת ההאש – הפלט נשמר כסיסמה בבסיס הנתונים. בשרת שלנו אפשרנו להשתמש בכמה מנגנוני קריפטוגרפיה: Sha256, bcrypt, aragon2.

בשלב אימות הסיסמה השרת שולף את salt השמור עבור אותו משתמש, מוסיף אותו ואת ה-pepper לסיסמה שהוזנה, מבצע גיבוב ומשווה לערך ההאש שבמסד.

שילוב זה בשרת Flask הוטמע בשכבת השירות של ההרשמה/התחברות, באופן ששוקף למשתמש - המשתמש מספק סיסמה רגילה, והמערכת כבר מטפלת בהוספת ה-salt וה-pepper וגיבוב לפני כל השוואה או אחסון.

בסך הכל, חמשת רכיבי האבטחה הללו משתלבים יחד במערכת: הגבלת הקצב, הנעילה ו-CAPTCHA מטפלים בהגנה על ממשק ההתחברות מפני התקפות אוטומטיות ושימוש לרעה וה-hash עם salt+pepper ו-TOTP מגנים על המידע הרגיש עצמו ומוסיפים שכבת זיהוי נוספת במקרה בו סיסמה נפלה לידיים זרות. השילוב הנכון בין המנגנונים מספק עומק הגנתי עבור המערכת כולה.

## תיעוד הקונפיגורציה

המערכת שפותחה גמישה ומאפשרת שליטה בהפעלת מנגנוני האבטחה השונים דרך קובצי התצורה או משתני סביבה, ללא צורך בשינוי הקוד. בצורה זו, ניתן להתאים את רמת ההגנה ולקבוע אילו שכבות אבטחה פעילות בהתאם לצרכים או לסביבת ההרצה. להלן פירוט אפשרויות הקונפיגורציה, כולל דוגמה לערכי ברירת המחדל של כל מנגנון, כל המשתנים מוגדרים כמשתני סביבה של המערכת:

ENABLE\_PEPPER - מאפשר להדליק או לכבות את מנגנון pepper, ברירת המחדל – דלוק. כאשר דלוק מיוצג בערך 1 בספרה החמישית משמאל ב-protection\_flag שמודפס בקובץ הלוגים.

ENABLE\_SALT - מאפשר להדליק או לכבות את מנגנון salt, ברירת המחדל – דלוק. כאשר דלוק מיוצג בערך 1 בספרה השישית משמאל ב-protection\_flag שמודפס בקובץ הלוגים. במקרה של הצפנה עם שיטת ארגון 2 בשל צורת העבודה של API אין יכולת לכבות את השימוש ב-salt ולכן תמיד יהיה דלוק במקרה של שימוש בארגון 2. כתוצאה מכך במקרה שכזה הספרה השישית תהיה תמיד 1 ללא קשר למשתנה הסביבה ENABLE\_SALT.

ENABLE\_RATE\_LIMITING - מאפשר להדליק או לכבות את מנגנון rate limiting, ברירת המחדל – דלוק. כאשר דלוק מיוצג בערך 1 בספרה הראשונה משמאל ב-protection\_flag שמודפס בקובץ הלוגים.

ENABLE\_LOCKOUT - מאפשר להדליק או לכבות את מנגנון account lockout, ברירת המחדל – דלוק. כאשר דלוק מיוצג בערך 1 בספרה השנייה משמאל ב-protection\_flag שמודפס בקובץ הלוגים.

ENABLE\_CAPTCHA - מאפשר להדליק או לכבות את מנגנון captcha, ברירת המחדל – דלוק. כאשר דלוק מיוצג בערך 1 בספרה השלישית משמאל ב-protection\_flag שמודפס בקובץ הלוגים.

ENABLE\_TOTP - מאפשר להדליק או לכבות את מנגנון TOTP, ברירת המחדל – דלוק. כאשר דלוק מיוצג בערך 1 בספרה הרביעית משמאל ב-protection\_flag שמודפס בקובץ הלוגים. במידה ושם המשתמש לא נמצא בבסיס הנתונים ואינו מזוהה הערך ב-protection\_flag יהיה 0 ללא קשר לערך שיש ב-ENABLE\_TOPT.

## סימולציית תקיפות וניתוח נתונים

במסגרת חלק זה בפרויקט פותחה והופעלה מערכת לביצוע תקיפות אוטומטיות על שרת האימות שתואר. מטרת הניסוי הייתה לבחון כיצד מנגנוני אבטחה שונים כמו Salt, Pepper, Rate Limiting, Account Lockout ו-TOTP משפיעים על יכולת התוקף להצליח במתקפות נפוצות ועל הזמן שנדרש לכך.

יש לציין שהניסוי שביצענו מכיל מספר מגבלות שמונעות ממנו להיות מקיף ושלם יותר. ראשית, חומרת הסביבה בה אנו רצים לא משתווה לחומרה של שרת אמיתי או של תוקף אמיתי. כמו כן, השרת הנתקף לא מכיל פניות במקביל של משתמשים תמימים נוספים, אלא מטפל רק בבקשות התוקף. בנוסף, כמות התקיפות שאנחנו מבצעים וכמות המשתמשים בשרת קטנה מאוד ביחס לתקיפה בסביבה אמיתית. גם מנגנוני ההגנה של השרת כדוגמת Captcha אינם מלאים בשל מגבלות הניסוי. למרות כל המגבלות הללו, ניסינו להתבסס על עקרונות אבטחת מידע, לשלב אותם עם תוצאות הניסוי ולהגיע למסקנות והמלצות חשובות.

כמו כן, בניסוי יכלו להיות גורמים שונים שעלולים לפגוע בדיוק התוצאות. למשל סביבת ריצה לא מבודדת (בה רצים תהליכים שונים שיכולים לקחת זמן ריצה), באגים בניית התוצאות או במימוש קוד השרת / תוקף. במהלך העבודה שמנו דגש על דיוק התוצאות ובדידוד הסביבה עד כמה שניתן כדי לקבל מידע אמין שאפשר ממנו להגיע לתובנות ומסקנות.

המערכת שביצעה את התקיפות נבנתה כסקריפט Python, המייצר תרחישי התקפה שונים, מריץ אותם מול השרת תחת קונפיגורציות שונות, מודד ביצועים ומפיק לוגים ונתונים לניתוח.

בחלק מהבדיקות, התבצע חישוב טהור של זמן ביצוע הגיבוב ללא ניסיון מול שרת Flask, מכיוון שהתקורה של ניסיון החיבור אל מול שרת HTTP גורמת לעיכוב משמעותי בבדיקה שגורם לחוסר הבחנה בין אלגוריתמי הגיבוב השונים.

התקיפות שנבדקו מחולקות לשני סוגים מרכזיים:

1. **Brute-Force**: ניסיון לנחש סיסמה של משתמש יחיד.
2. **Password Spraying**: שימוש בסיסמה אחת או מספר סיסמאות על קבוצת משתמשים.

הסקריפט שנכתב מייצר התקפות בהתאם לקונפיגורציה שהתקבלה כפרמטר. לכל התקפה מוגדרים:

- **משתמשים לתקיפה**
- **רשימת סיסמאות לניחוש**
- **מנגנוני ההגנה המופעלים בשרת**

התקיפות רצות אוטומטית ברצף, כאשר כל תרחיש מבוצע על מופע מבודד של שרת עם סט הגנות שונה, בהתאם לקונפיגורציה.

אמנם בניסוי לא בוצעה מתקפת Brute-Force מלאה הכוללת מיליוני ניסיונות וספריות סיסמאות רחבות, אך הדימוי שבוצע כן משיג את מטרתו המחקרית. המטרה איננה בהכרח לבחון האם ניתן "לנחש" סיסמה מתוך מאגר עצום, אלא לבחון כיצד מנגנוני ההגנה של השרת מגיבים לדפוס התקפי של ניסיונות חוזרים על חשבון בודד. גם עם מספר קטן של סיסמאות, ניתן לראות את השפעת סוג Hash על קצב הניסיונות, את הבלימה המיידית של Rate Limiting ונעילת חשבון, ואת ההשפעה של pepper על עלות החישוב. כלומר, ההתנהגות של האטה, חסימה, או כישלון עקבי של התוקף משקפת את מה שהיה מתרחש גם במתקפה אמיתית בהיקפים גדולים. לכן, למרות שהמאגר קטן, הניסוי מדגים היטב את הרעיון המרכזי של Brute-Force ואת יעילות מנגנוני ההגנה.

גם מתקפת Password Spraying בפרויקט לא נדרשה למאגר גדול של סיסמאות, משום שעיקרון התקיפה הוא שימוש במספר מצומצם של סיסמאות נפוצות על כמות גדולה של משתמשים. למרות שהמאגר קטן, הניסוי מדמה בצורה מדויקת את התנהגות התקיפה במציאות: ניסיון יחיד נשלח לכלל המשתמשים, ההגנות המיועדות ל-Brute-Force כמו Rate Limiting או Lockout כמעט אינן משפיעות, והמערכת מאפשרת לתוקף להמשיך ולנסות לרוחב המשתמשים ללא חסימה מוקדמת. העובדה שהתקיפה מצליחה גם כאשר חלק מהמנגנונים פעילים ממחישה את נקודת החולשה עליה התקפה זו מתבססת. הגנות שמבוססות על "ניסיונות רבים על משתמש אחד" אינן יעילות מול התקפה רוחבית. לכן, גם בהיקף קטן, הניסוי משקף היטב את אופי התקיפה ואת הצורך במנגנוני הגנה ייעודיים לטיפול בה.

## נסיונות כושלים לאורך תהליך הפיתוח

בשלבי הפיתוח הראשונים הרצנו את כל סוגי התקיפות ישירות מול שרת ה-Flask. לבסוף גילינו שזמני התגובה שנמדדו לא שיקפו את זמן החישוב של פונקציית הגיבוב, אלא את התקורה הטבעית של השרת שגדולה בהרבה מזמן הגיבוב עצמו. התקורה הזו הייתה קבועה יחסית ולכן היה קשה להבחין בהבדלים בין אלגוריתמי גיבוב שונים (SHA-256 / bcrypt / Argon2) ונוצר קושי להפיק מסקנות משמעותיות. כדי לפתור זאת עברנו בחלק מהבדיקות למצב סימולציה מקומית (local verification) שבו פונקציית התקיפה מדלגת על שכבת ה-HTTP ומוודאת רק את הגיבוב עצמו. המעבר הזה אפשר לנו לקבל תוצאות מייצגות בהרבה.

## תוצאות הניסויים וניתוחם

### השוואת מהירות מנגנוני הגיבוב

בשלב הראשון נבחנו רמת העמידות של כל אחד ממנגנוני הגיבוב כאשר כל מנגנוני ההגנה כבויים :

מנגנון גיבוב	ניסיונות חיבור בשנייה
SHA-256	299593
Argon2	7.06
Bcrypt	3.63

תוצאות מבחני מהירות ה-Hash מציגות באופן ברור את הפער העצום בין מנגנוני הגיבוב המהירים לבין המנגנונים היעודיים שנועדו להאט תוקפים.

האלגוריתם SHA-256 ביצע מספר עצום ניסיונות חיבור בשנייה, נתון המשקף את העובדה שמדובר בפונקציית גיבוב קריפטוגרפית מהירה מאוד שאינה מותאמת לאחסון סיסמאות. לעומתו, Argon2 הצליח לבצע רק כ-7 ניסיונות בשנייה, וב-bcrypt הקצב ירד לכ-3.6 ניסיונות בלבד. פער זה איננו מקרי' הוא משקף את העיקרון הקריפטוגרפי עליו מתבססים מנגנוני גיבוב איטיים : להפוך כל ניסיון לנחש סיסמה לפעולה יקרה מבחינת זמן ומשאבי מחשב. בכך, כל מנגנון איטי מסוג bcrypt או Argon2 מעלה משמעותית את העלות של מתקפת Brute-Force ומספק שכבת הגנה שאלגוריתם SHA-256 לא מסוגל להעניק.

## תוצאות תקיפה מבוססת BruteForce

מנגנון גיבוב	מנגנוני הגנה דלוקים	ניסיונות חיבור בשנייה	תוצאה
Bcrypt	SALT, RATE LIMITING	5 לאחר הגעה למקסימום ניסיונות, כל ניסיון חיבור נדחה מיד ונענה בשלילה	כישלון – לאחר חמשת ניסיונות התחברות התוקף נכנס להשהייה שמונעת ממנו להמשיך את התקיפה
Bcrypt	SALT, LOCKOUT	5 לאחר הנעילה, כל ניסיון חיבור נדחה מיד ונענה בשלילה	כישלון – לאחר ניסיון ההתחברות החמישית התקיפה נחסמה
Argon2	SALT, PEPPER	3	הצלחה
sha256	SALT, TOTP	102	הצלחה
Argon2	SALT, PEPPER, RATE LIMITING, LOCKOUT, CAPTCHA	5 לאחר חמש ניסיונות חיבור, התקבלה נעילה ולא ניתן להתחבר יותר	כישלון – מנגנונים שונים מונעים יותר מ-5 תקיפות בפרק זמן קצר

יש לקחת בחשבון שהבדיקות הללו נעשות מעל שרת flask ולכן ניסיונות החיבור מושפעים מתקורת השרת.

## השפעת מנגנון Rate Limiting

בבדיקת Brute-Force עם Rate Limiting מעל פונקציית bcrypt המערכת הצליחה למנוע לחלוטין את הצלחת ההתקפה, למרות שהסיסמה נמצאת ברשימת הניחוש. קצב ההתחברות היה כעשרים ושישה ניסיונות לשנייה, אך ההתקפה הסתיימה ללא הצלחה מכיוון שהחל מניסיון ההתחברות השישי, הגענו למספר הניסיונות המקסימלי וכל בקשר ההתחברות נוספת נתקלה בסירוב מצד השרת. המשמעות היא שהמנגנון יוצר האטה משמעותית בקצב הניסיונות, אשר הופכת Brute-Force לפחות אפשרית. בתקיפה אמיתית עם מספר רב של סיסמאות, המנגנון יכול לגרום לעיכוב של שעות ואפילו ימים. המסקנה: Rate Limiting יעיל מאוד נגד Brute-Force.

## השפעת מנגנון Account Lockout

בדומה למנגנון Rate Limiting, גם כאן החשבון ננעל לאחר מספר כשלונות קטן ולכן התוקף אינו יכול לבצע מספיק ניסיונות כדי להגיע לסיסמה הנכונה. בבדיקה שביצענו, נעשה ניסיון לתקיפת bruteforce על משתמש מעל אלגוריתם גיבוב bcrypt. מכיוון שהגענו למספר המירבי של ניסיונות, כל ניסיון נוסף בתקיפה נחסם ולא הצלחנו למצוא את הסיסמה המתאימה. ההתקפה נחסמת כמעט מיד, ללא תלות בחוזק הסיסמה או באיכות רשימת הניחוש. לכן נעילת חשבון היא מנגנון אפקטיבי במיוחד בעצירת מתקפות Brute-Force, אך מצד שני יש לקחת בחשבון משתמש תמים יכול להינעל בקלות אם אדם זדוני מנסה לתקוף את משתמשו.

## השפעת מנגנון PEPPER

מנגנון זה אינו מבצע האטה מורגשת של זמן ההתחברות ולא יסייע בעיכוב זמני התקיפה. ניתן לראות כי תרומתו העיקרית אינה בזמן הריצה אלא באבטחה הכוללת של המערכת. במקרה של דליפת בסיס נתונים, התוקף יקבל את ה-Hash ואת ה-Salt אך עדיין יידרש לנחש גם את ערך ה-Pepper כדי להצליח לשחזר את הסיסמאות. לכן, גם אם השפעתו בזמן התקיפה עצמה זניחה, מומלץ לשלב Pepper כחלק מארכיטקטורת האבטחה הכוללת.

## השפעת מנגנון TOTP

מכיוון שמנגנון ה-TOTP שנמצא בשרת זה אינו אמיתי לחלוטין, השפעותיו לא באו לגמרי לידי ביטוי בפרויקט זה, אך מנגנון זה חשוב ומקנה שכבת הגנה נוספת שמונעת ביצוע bruteforce. בשרת ההתחברות אמיתי, לפני כל התחברות יש צורך בקבלת סיסמה זמנית והקשה שלה. הדבר הופך את התקיפה למורכבת הרבה יותר עם אלמנט נוסף שיש לבצע.

## השפעת מנגנון CAPTCHA

בדומה ל-TOTP, גם מנגנון זה אינו בא לידי ביטוי בצורה מלאה כמו בשרת אמיתי שנותן נקודת קצה של התחברות מלאה. עם זאת, מנגנון זה מאלץ את התוקף לממש אלמנט נוסף בתקיפה, שדורש סוג של אימות לאחר מספר ניסיונות התחברות כושלים. בשרת אמיתי, מנגנון זה חשוב מאוד ומאתגר עבור התוקף מכיוון שמדובר על אימות ויזואלי לרוב, שאינה ניתנת לפתרון באוטומציה פשוטה. לכן במימוש מנגנון זה ניתנת הגבלה משמעותית של כל תוקף המנסה לבצע ניסיונות כניסה בקצב גבוה.

## תוצאות תקיפה מבוססת Password Spraying

מנגנוני הגנה דלוקים	זמן תקיפת שלושים משתמשים	תוצאה
SALT	0.0046 שניות	הצלחה
SALT, LOCKOUT	0.0043 שניות	הצלחה
SALT, RATE LIMITING	0.0045 שניות	הצלחה
SALT, PEPPER, RATE LIMITING, LOCKOUT, CAPTCHA	1.6 שניות	הצלחה

התבצעו מגוון ניסיונות תקיפה בשיטת Password spraying, מעל סוגים שונים של הגנות שרת.

ההגנות אשר נבדקו : Rate limiting, Lockout, Captcha, Pepper, Salt. למעשה, הייתה תקיפה אל מול מופע של שרת שמיישם את כל אחת מההגנות ההלו במקביל, ואפילו אל מול שרת זה התגלה במהירות משתמש התואם את אחת הסיסמאות במאגר.

מכיוון שהמתקפה משתמשת בסיסמה אחת נגד מספר גדול של משתמשים, מספר הניסיונות הנדרש נמוך מאוד, למעשה בכל אחת מהבדיקות נמצאה סיסמה לאחר סבב אחד בלבד של ניסיונות חיבור.

לכן מנגנוני הגנה כמו Lockout ו-Rate-Limiting כמעט אינם באים לידי ביטוי. הם מופעלים רק לאחר מספר כישלונות רצופים על אותו משתמש, אך סוג מתקפה זה מחלק את הניסיונות בין משתמשים שונים, ולכן אנחנו לא מגיעים לספים הללו.

התוצאות מדגישות שההגנות הקלאסיות נגד Brute-Force אינן אפקטיביות נגד Spraying. גורם ההכרעה המרכזי בהצלחה הוא קיומן של סיסמאות חלשות במאגר. כמו כן, מנגנון Captcha אמיתי היה מקשה משמעותית על התוקף. מנגנון הגנה נוסף שאפשר ליישם כנגד מתקפה כזאת הוא חסימה של כתובת IP שמנסה לבצע מספר חריג של התחברויות, גם אם כל ניסיון חיבור נעשה למשתמש שונה.

## תיעוד התקיפות בקבצי לוג

בפרויקט אשר פיתחנו, קיימת תיקייה בשם logs שמכילה פירוט של תוצאות הבדיקה. הקובץ attempts.log מכיל את כל ניסיונות החיבור שתועדו בשרת Flask. הקובץ attack\_summary\_results.json מכיל סיכום של כל תקיפה עם מגוון נתונים שעוזרים לנתח אותה כמו מספר חיבורים בדקה, זמן עד למציאת הסיסמה, ההגנות הפעילות בשרת, מנגנון הגיבוב בשרת ועוד.

## עמידה בכללי אתיקה

במהלך פיתוח הפרויקט, ביצוע התקיפות הממוחשבות וניתוח התוצאות, הקפדנו לפעול בהתאם לכללי האתיקה המקובלים בתחום אבטחת המידע. כל התקיפות שבוצעו במסגרת העבודה בוצעו במערכת פנימית שפותחה במיוחד לצורך המחקר, אותה הצגנו בדו"ח המחקר והקוד שלה גלוי לכולם. המחקר אינו מערב שרתים, משתמשים או נתונים אמיתיים.

כל פרטי ההרשאות, הסיסמאות והמשתמשים הינם מלאכותיים ונוצרו אך ורק למטרות המחקר. לא בוצעה בשום שלב גישה או ניסיון גישה למשאבים חיצוניים ולא נלקח מידע אמיתי על משתמשים. בנוסף, כל פעולות התקיפה נועדו אך ורק לצורך בחינת מנגנוני הגנה ומתבצעות תוך הבנה כי השימוש בהן מחוץ לגבולות הקורס והלימודים הינו אסור ועלול להוות עבירה פלילית.



## מסקנות והמלצות

לאחר פיתוח השרת, מימוש התקיפות השונות וניתוח התוצאות, גיבשנו מספר המלצות חשובות למימוש בשרת מרוחק:

- יש לחייב את המשתמש לבחור סיסמה חזקה, כזאת שלא נמצאת במאגר סיסמאות נפוצות ושקשה לנחש בשיטת Brute force.
- מימוש מלא של הגנות TOTP ו/או Captcha מהווה מרכיב קריטי בהגנה בפני מתקפות וחשוב ליישם אותו היטב.
- מעבר להגנות הבסיסיות, יש לשקול ניתוח של ניסיונות התחברות בהקשר רחב. הדבר שהכי בלט לנו הוא שמרבית ההגנות מתייחסות לניסיון התחברות של משתמש נקודתי, והדבר לא מונע מתקפת Spraying. יש לבצע ניתוח חכם יותר של כל ניסיון התחברות, ולזהות מצב שבו אדם זדוני מנסה להתחבר מספר פעמים גם אם למשתמשים שונים.
- מעבר להגנה על ניסיון פריצה, חשוב גם להגן על משתמשים מפני ניסיון Dos שיגביל אותם מהאפשרות להתחבר.
- ניטור בזמן אמת לאירועי פריצה יכול לעזור באיתור התוקף, והצלת חשבונות לפני פגיעה רחבה. תחקור בדיעבד יכול לגלות מה קרה, אך יכול להיות מאוחר מידי מכיוון שהמידע של המשתמשים כבר נגנב.

## מקורות

- תיעוד רשמי של Flask לטובת שרת הסיסמאות - <https://flask.palletsprojects.com/en/stable>
- תקיפת Brute force בויקפדיה - [https://en.wikipedia.org/wiki/Brute-force\\_attack](https://en.wikipedia.org/wiki/Brute-force_attack)
- pyotp תיעוד ודוגמאות - <https://pyauth.github.io/pyotp/>
- מבוא לאבטחת המרחב המקוון – מדריך הלמידה