

דו"ח אבטחת סיסמאות

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

תוכן עניינים :

מבוא על מנגנוני אחסון סיסמאות	2
תיאור השרת וההגנות	2
תיעוד הקונפיגורציה	3
סימולציה התקיפות וניתוח נתונים	5
ניסיונות כושלים לאורך תהליך הפיתוח	6
תוצאות הניסויים וניתוחם	6
תיעוד התקיפות בקבצי לוג	8
עמידה בכלי ATIKA	8
מסקנות והמלצות	9

מבוא על מנגנון אחסון סיסמות

אחסון מאובטח של סיסמות הוא מרכיב קריטי בהגנה על משתמשים ומערכות. המטרה היא לשמר סיסמות באופן שאינו חשוף את הסיסמה עצמה גם אם מאגר הנתונים נפרץ. אחסון סיסמות בטקסט גליי מהוות סיוכן חמור, שכן דליפת בסיס נתונים כזה תחשוף מיד את סיסמות המשתמשים. במקרה זאת, נהוג לשומר ערכى גיבוב (hash) של הסיסמות וכך לוודא שלא ניתן לשחרר מהhash את הסיסמה המקורית.

בעת אימות סיסמה, המערכת מבצעת גיבוב לסיסמה שהזונה ומשווה לערך השמור במאגר. גישה זו מונעת מאדם זודני לקבל את הסיסמה עצמה גם אם הצליח להשיג את ה hash. עם זאת, גם שמירת hash בלבד אינה מספקת, וכיום לחזק את האבטחה נהוג לשלב טכניקות נוספות.

תיאור השירות וההגנות

הישום שנבנה Flask במסגרת הפROYKT מימוש שרת אינטראנט לאימות משתמשים, עם מספר מנגנון הגנה מסוולבים שנועדו להגן מפני פריצות ותקיפות על מנגנון ההתחברות. להלן המנגנונים העיקריים שהוטמעו בשרת, תפקדים ואופן השתלבותם:

Salt

משמעותו הוספת ערך אקראי וייחודי לכל סיסמה לפני hash, כך שאפילו אם שני משתמשים בוחרים באותה סיסמה ערכיה החאים שהם מקבלים יהיו שונים לחלוון.

Salt נשמר לצד החאש במסד הנתונים, ותפקידו העיקרי לסכל התקפות באמצעות Rainbow Tables ולהקשות על זיהוי סיסמות זהות של משתמשים שונים.

Pepper

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

הערך pepper המשותף לכל הסיסמות מօסיף שכבת סודיות נוספת: תוקף שמצலח לקבלת גישה למסד הנתונים ולערכי החאש והsalt ערך ידרש לו לגלות את pepper הסודי כדי לעמע את הסיסמות ובכך שיLOB של hash עם salt pepper מון 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 סיסמות, קיים צורך להתמודד עם מצב שבו סיסמה עלולה להתגלות או להיגנב. תכנית TOTP סיסמה חד פעמיות מותזנת – מספקת גורם שני לאימות משתמש. המשמעות היא שלא מספיק סיסמא סטטית כדי להיכנס אלא נדרש גם קוד חד פעמי המשתנה מדי זמן קצר (לרוב 30 שניות) באמצעות אפליקציה או התקן ייעודי (כמו למשל גולגל אוטנטיקייטור).

במקרה כזה גם אם הסיסמא הזולפה התוקף ידרש לספק גם קוד חד פעמי של הזמן הנוכחי, שונות בכל ניסיון כניסה מה שמקשה על פריצת החשבון. השימוש לשמירת סיסמות 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 מהמשתמש לעבור מבחן captcha (במציאות זה יכול להיות מבחן של בחירות תמונה מתאימה או פתרון תרגילים פשוטים אך במימוש שלנו אנחנו רק מודמים captcha) לפני שיוכל לנסתות להתחבר. במימוש שלנו captcha היא טוקן סודי שניתן לייצור בעורת פניה לשרת ב-Rest API עם נקודת הקצה admin/get_captcha_token/`token` שמייצר טוקן סודי ושומרת אותו בשרת. המשתמש בהתחברות יכנס את הטוקן הזה וכן הרשת יאמת שהטוקן זהה לטוקן ששמור אצלו.

שילוב CAPTCHA מנסה על תוקף אוטומטי מפני שגם אם אין נחסים לחלוון, הוא נדרש למאיץ אנושי בכל ניסיון נוסף.

הגבלת קצב הפניות – Rate Limiting

מנגנון ש מגביל את כמות ניסיונות ההתחברות או הבקשות שניתן לביצוע בפרק זמן נתון. מטרתו למנוע ניסיונות Brute force על ידי האת קצב הניחושים כמו למשל, הגבלה של מספר ניסיונות ההתחברות כושלים לדקה לכל משתמש. בשרת שלנו, מנגנון זה מושלב כך שאם מתאפשרות יותר מדי בקשות ההתחברות ברצף תוך זמן קצר, השרת

מעכב או חוסם זמני בקשנות נוספת גורם. ללא הגבלת קצב כלל, יישום מזמין מתקפה כיון שתוקף יכול לנשוט סיסמאות ללא הפרעה. עם מנגנון זה התקפה אוטומטית מהירה תהופך ללא ישימה ממשום שהשתתת גביל את התדיירות ומעלה את משך הזמן שנדרש לניחוש מספר רב של סיסמאות.

נעילת חשבון לאחר ניסיונות כושלים – Account Lockout

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

בשרות שבינו לאחר סוף מוגדר (5 ניסיונות כושלים ברגע) החשבון יונען למשך תקופה מסוימת (5 דקות).

השימוש משולב בקוד החתחבות כך שבכל ניסיון כושל מונה כשלונות וכש망יעים ל5 לא ניתן להתחבר גם תוכנס הסיסמה הנכונה. רק לאחר 5 דקות ניתן לנסות שוב.

מנגנון זה עוזר לוודא שתוקף לא יוכל לנסות כמוניות גדולות של סיסמאות על חשבון ספציפי ברגע.

TOTP – Time Based One-Time Password

כמו שבעה במובוא המנגנון מספק גורם שני לאימוט. בשרת שלנו מנגנון זה פועל לאחר השליטה הקודמים ולאחר שהזונה סיסמא תקינה ואז המערכת דורשת גם את הקוד החד פעמי הנוצר באפקציה ייoudiy (במקורה שלנו בפייתו בעזרת ספריית `totpdk`). בעת ההרשמה לכל משתמש מוגדר מפתח סודי עבור הTOTP הנשמר במסד הנתנים ומושיק לאפליקציית האימוט של המשתמש. בעת כניסה הכניסה הרשות מחשב את הקוד העדכני המתאים לשעה ולמפתח הסודי של המשתמש ומשווה אותו לקוד שספק המשתמש. רק אם שני הקודים זהים ובתוקף (בדרכ כל תקף ל30 שניות) הכניסה תואר.

המנגנון משולב בזרימות החתחבות: לאחר אימוט סיסמה מוצלח, משתמש שלא הגיע קוד נכון לא יוכל גישה מלאה. בכך, גם אם סיסמה דלפה, חשבון המשתמש יותר מזמן כל עוד התוקף אינו מחזיק גם במק Shir הфизי או באפליקציה המכילה את המפתח הסודי לייצירת הקודים.

Hash + Salt + PEPPER

מנגנון קריטי בבסיס השרת הוא אופן שמירת הסיסמאות עצמן במסד הנתונים. כפי שצוין במובוא, סיסמאות אינן נשמרות כטקסט גלויל אלא כערבי האש קרייפטוגרפיים. במימוש שלנו עבר כל סיסמה חדשה נוצר salt בגודל 8 בתים אקראיים המשולב בסיסמא ובנוסף אליו משורשר הsalt pepper הсолדי המוגדר כמשתנה גלובלי בשרת. בפועל בעת ההרשמה המשמש בוחר סיסמא שעוברת שרשור עם `hash` ו-`pepper` ואז נשלחת לפעולות האש – הפלט נשמר כסיסמא בסיס הנתונים. בשרת שלנו אפשרנו להשתמש בכמה מנגנוני קרייפטוגרפיה: `SHA256`, `bcrypt`, `argon2`.

בשלב אימוט הסיסמא הרשות שולף את `salt` השמור עבור אותו משתמש, מוסיף אותו ואת `pepper` לסיסמה שהזונה, מבצע גיבוב ומשווה לערך האש שבמסגר.

שילוב זה בשרת Flask הוטמע בשכבת השירות של ההרשמה/התחבות, באופן ששקוף למשתמש - המשמש מספק סיסמה וגילה, והמערכת כבר מטפלת בהוספת `-salt` ו-`pepper` וגיבובה לפני כל השוואה או אחסון.

בכך הכל, חממת רכיבי האבטחה הללו משתלבים יחד במערכת: הגבלת הקצב, הנעילה ו-CAPTCHA מטפלים בהגנה על ממתק החתחבות מפני התקפות אוטומטיות ושימוש לרעה וה-`hash` ו-`salt+pepper` ו-TOTP מגנים על המידע הרגישי עצמו ומוסיפים שכבת זהה נוספת בסיסמה נפלה לידיים זרות. השימוש הנכון בין המנגנונים מספק עומק הגנתיע עבור המערכת כולה.

תיעוד הקונפיגורציה

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

`ENABLE_PEPPER` - מאפשר להציג או לכבות את מנגנון `pepper`, ברירת המחדל – דלוק. כאשר דלוק מיוצג ערך 1 בספרה החמישית ממשמאל `protection_flag` שמודפס בקובץ הלוגים.

1. `ENABLE_SALT` - מאפשר להציג או לכבות את מנגנון `salt`, ברירת המחדל – דלוק. כאשר דלוק מיוצג ערך 1 בספרה השישית ממשמאל `protection_flag` שמודפס בקובץ הלוגים. במקרה של הצפנה עם שיטת ארגון 2 בשל צורת העבודה של API אין יכולת לכבות את השימוש `salt` ולכן תמיד יהיה דלוק במקרה של שימוש בארגון 2. כתוצאה לכך במקרה שכזה הספרה השישית תהיה תמיד 1 ולא קשר למשתנה הסביבה `ENABLE_SALT`.

כasher dolok myozg beurk 1 basperha hareshona meshmaal bag protection_flag shmodaf bokvez hlogim.

dolok myozg beurk 1 basperha hashniya meshmaal bag protection_flag shmodaf bokvez hlogim.

myozg beurk 1 basperha hashlishit meshmaal bag protection_flag shmodaf bokvez hlogim.

ENABLE_TOTP - מאפשר להציג או לכבות את מגנון TOTP, בירית המחדל – Dolok. כאשר Dolok מוצג בערך 1 בספירה הרביעית משמאלי protection_flag שמודפס בקובץ הלוגים. Dolok מוצג בערך 0 ללא קשר לשאלה הדרישה.

סימולציה ותקיפות וניתוח נתונים

במסגרת חלק זה בפרויקט פותחה והופעלה מערכת לביצוע התקיפות אוטומטיות על שרת האימוט שתואר. מטרת הניסוי הייתה לבדוק כיצד מנגנוני אבטחה שונים כמו TOTP, Salt, Pepper, Rate Limiting, Account Lockout ו-**Captcha** מושפעים על יכולת התקוף להצלחה במתפקות נפוצות ועל הזמן שנדרש לכך.

יש לציין שהניסוי שביצענו מכיל מספר מגבלות שימושו להיות מكيف ושלם יותר. ראשית, חומרת הסביבה בה אנו רצים לא משתוויה לחומרה של שרת אמיתי או של תוקף אמיתי. כמו כן, השרת הנתקף לא מכיל פניות במקביל של משתמשים תמיימים נוספים, אלא מטפל רק בבקשתות התקוף. בנוסף, כמה התקיפות שאחננו מבצעים וכמות המשמשים בשרת קטנה מאוד ביחס לתקיפה בסביבה אמיתי. גם מנגנוני ההגנה של השרת כדוגמת **Captcha** אינם מלאים בשל מגבלות הניסוי. למרות כל המוגבלות הללו, ניסינו להתבסס על עקרונות אבטחת מידע, לשלב אותם עם תוצאות הניסוי ולהגיע למסקנות והמלצות חשובות.

כמו כן, ניסויינו יכול היה להיות גורמים שונים שעלו לפוגע בדיקת התוצאות. למשל סביבת ריצה לא מבודדת (בה רצים תחילcisים שונים שיכולים לחתן זמן ריצה), באגים בניטוח התוצאות או במימוש קוד הרשות / תוקף. במהלך העבודה שלנו דוש על דיקוק התוצאות ובידוד הסביבה עד כמה שניתן כדי לקבל מידע אמין שאפשר ממנו להציג לתובנות ומסקנות.

המערכת שביצעה את התקיפות נבנתה כסקריפט Python, המיציר תרחישי התקפה שונים, מרייצ' אותם מול השרת תחת קונפיגורציות שונות, מודד ביצועים ומפיק לוגים ונתונים ניתוח.

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

התKİפות 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 התקיפות בהתאם לكونפיגורציה שהתקבלה כפרמטר. לכל התקפה מוגדרים :

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

התKİפות רצות אוטומטית ברגע, כאשר כל תרחיש מבוצע על מופיע מבודד של שרת עם סט הנקודות שונה, בהתאם לكونפיגורציה.

אנו בניסוי לא בוצעה מתකפת Brute-Force מלאה הכוללת מיליון ניסיונות ומספרות סיסמאות רחבות, אך הדימוי שבוצע לנו מטרתו המקורית. המטרה איננה בהכרח לבחון האם ניתן "לנחש" סיסמה מותך מאור 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 ה-Hash על קצב הניסיונות, את הבילמה המיידית של Rate Limiting ונעילת חשבון, ואת הרשפעה של pepper על עלות החישוב. ככלומר, ההנחהות של האטה, חסימה, או כישלון עקבי של התקוף משקפת את מה שהוא מתרחש גם במתකפה אמיתי 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 נדרש למאגר גדול של סיסמאות, משום שעיקרון התקיפה הוא שימוש במספר מצומצם של סיסמאות נפוצות על כמה גודלה של משתמשים. למרות שהמאגר קטן, הניסוי מדמה בצוරה מדוקיקת את הנקודות התקיפה במציאות: ניסיון ייחד נשלח לכל המשתמשים, ההגנות המיעודות ל--**Rate Limiting** או **Lockout** כמעט אינן מושפעות, והמערכת מאפשרת לתקוף להמשיך ולנסות לרוחב המשתמשים ללא חסימה מוקדמת. העובדה שהתקיפה מצליחה גם כאשר חלק מהמנגנוני פעילים ממחישה את נקודת החולשה עליה התקפה זו מתבססת. הנקודות ש商量ססות על "ניסיונות רבים על משתמש אחד" אינן ייעילות מול התקפה רוחבית. לכן, גם בהיקף קטן, הניסוי משקף היטב את אופי התקיפה ואת הצורך במנגנון הגנה ייעודיים לטיפול בה.

ניסיונות כושלים לאורך תהליך הפיתוח

בשלבי הפיתוח הראשוניים הרצינו את כל סוגי התקיפות יישירות מול שרת Flask. לבסוף גילינו שזמן התגובה שנדדו לא שיקפו את זמן החישוב של פונקציית הגיבוב, אלא את התקורתה הטבעית של השרת שגדולה בהרבה מזמן הגיבוב עצמו. התקורת זו הייתה קיימת רק במקרה 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 מدلגת על שכבת HTTP ומודדת רק את הגיבוב עצמו. המעבר הזה אפשר לנו לקבל תוצאות מיצוגות בהרבה.

תוצאות הניסויים וניתוחם

השוואת מהירות מנגנוני הגיבוב

בשלב הראשון נבנה רמת העמידות של כל אחד מנגנוני הגיבוב כאשר כל מנגנון ההגנה קבוע :

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

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

האלגוריתם SHA-256 ביצע מספר עצום ניסיונות חיבור בשניה, נתון המשקף את העבודה שמדובר בפונקציית גיבוב קריפטוגרפית מהירה מאוד שאינה מותאמת לאחסון סיסמות. לעומת זאת Argon2 הצליח לבצע רק כ-7 ניסיונות בשניה, ובכך הקצב ירד לכ-3. ניסיונות בלבד. פער זה איננו מקרי' הוא משקף את העובדה הקrifptוגרפי עליון מותבסטים מנגנוני גיבוב איטיים : להפוך כל ניסיון לנחש סיסמה פעולה יקרה מביכות זמן ומשאבי מחשב. בכך, כל מנגנון איטי מסוג bcrypt או Argon2 מעלה משמעותית את הצלות של מתקפת Brute-Force ומספק שכבת ההגנה שאלגוריתם SHA-256 לא מסוגל להעניק.

תוצאות התקיפה מבוססת Bruteforce

מנגנון גיבוב	מנגנוני ההגנה דלוקים	ניסיונות חיבור בשניה	תוצאה
Bcrypt	SALT, RATE LIMITING	5 לאחר הגעה למקסימום ניסיונות, כל ניסיון חיבור נדחה מיד ונענה בשלילה	כישלון – לאחר חמישה ניסיונות התחברות התקוף נכנס להשהייה שמנועת ממנו להמשיך את התקיפה
Bcrypt	SALT, LOCKOUT	5 לאחר הנעליה, כל ניסיון חיבור נדחה מיד ונענה בשלילה	כישלון – לאחר ניסיון ההתחברות החמשית התקיפה נחסמה
Argon2	SALT, PEPPER	3	הצלחה
sha256	SALT, TOTP	102	הצלחה
Argon2	SALT, PEPPER, RATE LIMITING, LOCKOUT, CAPTCHA	5 לאחר חמישת ניסיונות מונעים יותר מ-5 התקיפות בפרק זמן קצר התקבלה נעילה ולא ניתן להתחבר יותר	כישלון – מנגנונים שונים מונעים יותר מ-5 התקיפות בפרק זמן קצר התקבלה נעילה ולא ניתן להתחבר יותר

יש לנקות בחשבונו שהבדיקות הללו נעשות מעל שרת flask ולכון ניסיונות החיבור מושפעים מתקורת השרת.

השפעת מנגנון Rate Limiting

בבדיקה Brute-Force עם פונקציית bcrypt המרכיבת הצלחה למneau לחלוטין את הצלחת ההתקפה, למרות שהטיסומה נמצאת ברשימת הניחושים. קצב התחרבות היה כעירים ושישה ניסיונות לשנייה, אך התקפה הסתיימה ללא הצלחה מכיוון שהחלה מניסיון התחרבות השישי, הגענו למספר הניסיונות המקסימלי וכל קשר התחרבות נוספת נתקלה בסירוב מצד השירות. המשמעות היא שהמנגנון יוצרআאֶתְּהַמִּזְמָרָה משמעותית בקצב הניסיונות, אשר הופכת נספת נתקלה בסירוב מצד השירות. בתקיפה אמיתי עם מספר רב של סיסמאות, המנגנון יכול לגרום לעיכוב של שניות ואף יותר ימים. המסקנה: Brute-Force יעיל מאוד נגד Rate Limiting.

השפעת מנגנון Account Lockout

בדומה למנגנון Rate Limiting, גם כאן החשבון נגעל לאחר מספר כשלונות קטן ולכון התוקף אינו יכול לבצע מספיק ניסיונות כדי להגיע לסיסמה הנכונה. בבדיקה שביצענו, נעשה נסיף לתקיפה bruteforce על משתמש בעל אלגוריתם גיבוב bcrypt. מכיוון שהגענו למספר המירבי של ניסיונות, כל ניסיון נוסף בתקיפה נחסם ולא הצליחנו למצוא את הסיסמה המתאימה. ההתקפה נחסמת כמעט מיד, ללא תלות בחזקה הסיסמה או באיכות רישימת הניחושים. لكن נעלית החשבון היא המנגנון אפקטיבי במיוחד בעקבות מתקפות Brute-Force, אך מכך שני יש לנקות בחשבון משתמש תמים יכול להינעל בקלות אם אדם זודני מנסה לתקוף את משתמשו.

השפעת מנגנון PEPPER

מנגנון זה אינו מבצע האטה מוגשת של זמן ההתקבות ולא יסייע בעיקוב זמני התקיפה. ניתן לראות כי תרומותיו העיקרית אינה בזמן הריצה אלא באבטחה הכוללת של המערכת. במקרה של דליפת בסיס נתונים, התוקף קיבל את-hash ואות-h-salt אך עדין יידרש לנחש גם את ערך-pepper כדי להצליח לשוחזר את הסיסמאות. וכן, גם אם השפטו בזמן התקיפה עצמה זניחה, מומלץ לשלב pepper כחלק מארכיטקטורת האבטחה הכוללת.

השפעת מנגנון TOTP

מכיוון שמנגנון TOTP שנמצא בשורת זה אינו אמיתי לחלוטין, השפעותיו לא באו לגמרי ידי ביתוי בפרויקט זה, אך המנגנון הזה חשוב ומקנה שכבת הגנה נוספת שמונעת ביצוע bruteforce. בשורת התקבות אמיתי, לפני כל התקבורות יש צורך בקבלת סיסמה זמנית והקשה שלה. הדבר הופך את התקיפה למורכבת הרבה יותר עם אלמנט נוסף שיש לבצע.

השפעת מנגנון CAPTCHA

בדומה לTOTP, גם המנגנון זה אינו בא לידי ביתוי בצורה מלאה כמו בשורת אמיתי שנוטן נקודת קצה של התקבות מלאה. עם זאת, המנגנון הזה מאלץ את התוקף למש אלמנט נוסף בתקיפה, שדורש סוג של אימוט לאחר מספר ניסיונות התקבות כושלים. בשורת אמיתי, המנגנון הזה חשוב מאוד ומאתגר עבור התוקף מכיוון שהוא מדבר על אימוט ויזואלי לרוב, שאינה ניתנת לפתרון באופן ממוציה פשוטה. לכן במימוש המנגנון זה ניתנת הגבלה משמעותית של כל תוקף המנסה לבצע ניסיונות כניסה בקצב גבוה.

פתרונות התקיפה מבוססת Password Spraying

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

התבצעו מגוון ניסיונות התקיפה בשיטת Password spraying, מעל סוגי שונים של הגנות שרת.

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

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

לכן מנגנוני הגנה כמו Lockout ו-Rate-Limiting כמעט אינסайдי ביתיים. הם מופעלים רק לאחר מספר כישלונות רצופים על אותו משתמש, אך סוג מתתקפה זה מחלק את הניסיונות בין משתמשים שונים, וכך אכן לא מגיעים למספרים הללו .

התוצאות מדגימות שההגנות הקלasicות נגד Brute-Force אינן אפקטיביות נגד Spraying . גורם ההכרעה המרכזית בהצלחה הוא קיומן של סיסמאות חלשות במאגר. כמו כן, מנגנון Captcha אמייתי היה מקשה משמעותית על התקוף. מנגנון הגנה נוסף שאפשר ליחסם כנגד מתתקפה כזו הוא חסימה של כתובות IP שמנסה לבצע מספר חריג של התחברויות, גם אם כל ניסיון חיבור נעשה למשתמש שונה .

тиיעוד התקיפות בקבצי log

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 התקיפה 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, ביצעו התקיפות הממוחשבות וניתוח התוצאות, הקפדו לפעול בהתאם לכללי האתיקה המקובלים בתחום אבטחת המידע. כל התקיפות שבוצעו במסגרת העבודה בוצעו במערכת פנימית שפותחה במיוחד לצורך המחקר, אותה הצגנו בדוח' מחקר והקוד שלו גלוי לכלום. המחקר אינו מעורב שרתנים, משתמשים או נתונים אמייטיים .

כל פרטיה הרשאות, הסיסמאות והמשתמשים הינם מלאכותיים ונוצרו אך ורק למטרות המחקר. לא בוצעה בשום שלב גישה או ניסיון גישה למשתמשים חיצוניים ולא נלקח מידע אמיתי על משתמשים. בנוסף, כל פעולות התקיפה נעשו אך ורק לצורך בדיקת מנגנוני הגנה ומתקבצות תוך הבנה כי השימוש בהן מוחז לגבולות הקורס והלימודים הינו אסור ועלול להוות עבירה פלילית.

מסקנות והמלצות

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

- ישחייב את המשמש לבחור סיסמה חזקה, כזו שהיא נמצאת במאגר סיסמאות נפוצות וקשה לנחש בשיטת Brute force.
- מיימוש מלא של הגנות TOTP ו/או Captcha מהוות מרכיב קריטי בהגנה מפני מתקפות וחשוב ליישם אותו היבר.
- מעבר להגנות הבסיסיות, יש לשקל ניתוח של ניסיונות התחברות בהקשר רחב. הדבר שהכי בלט לנו הוא שמרבית ההגנות מתייחסות לניסיון התחברות של משתמש נזודי, והדבר לא מונע מתקפת Spraying. יש לבצע ניתוח חכם יותר של כל ניסיון התחברות, ולזהות מצב שבו אדם זמני מנשה להתחבר מספר פעמים גם אם למשתמשים שונים.
- מעבר להגנה על ניסיון פריצה, חשוב גם להגן על משתמשים מפני ניסיון Dos שיגביל אותם מהאפשרות להתחבר.
- ניתור בזמן אמת לאירועי פריצה יכול לעוזר באיתור התוקף, והצלת חשבונות לפני פגיעה רחבה. תחקור בדיעבד יכול לגלוות מה קרה, אך יכול להיות מאוחר מדי מכיוון שהמידע של המשתמשים כבר נגמר.

מקורות

- תיעוד رسمي של Flask לטובת שירות הסיסמאות - <https://flask.palletsprojects.com/en/stable/>
- תקיפת Brute-force - https://en.wikipedia.org/wiki/Brute-force_attack
- תיעוד pyotp ודוגמאות - <https://pyauth.github.io/pyotp>
- מבוא לאבטחת המרחב המקומי – מדריך הלמידה