



סדרת אתגרי Check Point CSA 2019

מאת Dvd848 ו-YaakovCohen88

מבוא

באמצע מאי 2019 פרסמה חברת Check Point סדרת אתגרים כחלק מקמפיין גיוס עבור ה-Check Point Security Academy. האתגרים חולקו לארבע קטגוריות: רברסינג, פיתוח, קריפטוגרפיה ושונות. במאמר זה נציג את הפתרונות שלנו לסדרת האתגרים.

אתגר #1: Pinball Cipher - (קטגוריה: קריפטוגרפיה. ניקוד: 30)

*Time is of the essence, so I shall skip the introductions.
My children have disappeared - I haven't heard from them in weeks. In my relentless searches, all I was able to obtain is this single short message they have left for me.
The message is encrypted with the Pinball Cipher, a toy system which I authored personally for our family use. Unfortunately, the message author used a key unknown to me.
I have provided you with the encrypted message, as well as the encryption software for this Pinball Cipher (for both Windows and Linux).
Yours Truly, Sr. Reginald Hargreeves*

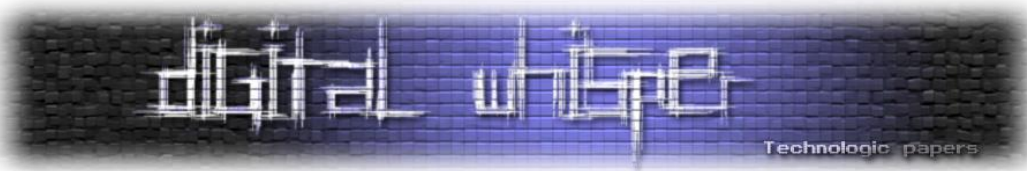
לאתגר צורף קובץ הרצה בשם pinball יחד עם קובץ מוצפן בשם msg.enc. נתחיל מהרצת התוכנה המצורפת:

```
root@kali:/media/sf_CTFs/checkpoint/Pinball_Cipher/pinball# ./pinball.elf
Pinball Encryptor 1.0.0
Sir Reginald Hargreeves
Encrypts and decrypts messages.

USAGE:
  pinball.elf <SUBCOMMAND>

FLAGS:
  -h, --help      Prints help information
  -V, --version    Prints version information

SUBCOMMANDS:
  help            Prints this message or the help of the given subcommand(s)
  table           Displays the encryption table.
  transform       Performs encryption/decryption (both are the same operation).
```



אם נבקש מהתוכנה להציג את טבלת ההצפנה, נקבל את הפלט הבא:

```
root@kali:/media/sf_CTFs/checkpoint/Pinball_Cipher/pinball# ./pinball.elf table
177 030 077 225 170 116 089
228 139 058 083 195 202 201
197 113 114 053 184 105 043
178 029 210 090 150 045 212
135 240 099 051 147 085 060
156 039 169 101 078 180 165
075 108 102 163 166 027 092
204 046 015 198 209 086 120
232 172 106 154 226 023 057
054 141 216 149 153 142 071
```

כמו כן, התוכנה מציעה תיעוד נרחב יותר של פעולת ההצפנה/פענוח (לפי התיאור, מדובר באותה פעולה):

```
root@kali:/media/sf_CTFs/checkpoint/Pinball_Cipher/pinball# ./pinball.elf help transform
pinball.elf-transform
Performs encryption/decryption (both are the same operation).

USAGE:
  pinball.elf transform <INPUT_FILE> <OUTPUT_FILE> <KEY_PY> <KEY_PX> <KEY_VY> <KEY_VX>

FLAGS:
  -h, --help          Prints help information
  -V, --version        Prints version information

ARGS:
  <INPUT_FILE>        Path of the plaintext file you want to encrypt.
  <OUTPUT_FILE>        Output will be written to this file.
  <KEY_PY>             Component PY of the encryption key. [possible values: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
  <KEY_PX>             Component PX of the encryption key. [possible values: 0, 1, 2, 3, 4, 5, 6]
  <KEY_VY>             Component VY of the encryption key. [possible values: +, -]
  <KEY_VX>             Component VX of the encryption key. [possible values: +, -]
```

ננסה להצפין קובץ לדוגמא:

```
root@kali:/media/sf_CTFs/checkpoint/Pinball_Cipher/pinball# echo aaaaaaaaaaaa > test.txt
root@kali:/media/sf_CTFs/checkpoint/Pinball_Cipher/pinball# ./pinball.elf transform test.txt test.enc 4 3 - -
XORing plaintext byte with 51
XORing plaintext byte with 210
XORing plaintext byte with 113
XORing plaintext byte with 228
Boing!
XORing plaintext byte with 30
Boing!
XORing plaintext byte with 58
XORing plaintext byte with 53
XORing plaintext byte with 150
XORing plaintext byte with 85
XORing plaintext byte with 165
Boing!
XORing plaintext byte with 27
XORing plaintext byte with 209
XORing plaintext byte with 154
XORing plaintext byte with 216
Boing!
XORing plaintext byte with 172
root@kali:/media/sf_CTFs/checkpoint/Pinball_Cipher/pinball# xxd -g 1 test.enc
00000000: 52 b3 10 85 7f 5b 54 f7 34 c4 7a b0 fb d2      R....[T.4.z...
```

מלבד השמות של קבצי הקלט והפלט, הכנסנו נקודות-ציון וכיווניות. לפי ההדפסות של התוכנה, ההצפנה מתבצעת באמצעות ביצוע פעולת XOR של התו הנוכחי של הטקסט עם תו שנלקח מטבלת ההצפנה לפי חוקיות של "כדור מקפץ": נקודות-הציון שהכנסנו משמשות בתור המיקום ההתחלתי של כדור דמיוני בתוך הטבלה, הכדור ממשיך לפי הכיווניות שהגדרנו וכאשר הוא פוגע ב"קיר" (קצה הטבלה) הוא ניתן ומשנה את כיוונו. את מהלך הכדור בדוגמא שלנו אפשר להציג באופן הבא:

177	030	077	225	170	116	089
228	139	058	083	195	202	201
197	113	114	053	184	105	043
178	029	210	090	150	045	212
135	240	099	051	147	085	060
156	039	169	101	078	180	165
075	108	102	163	166	027	092
204	046	015	198	209	086	120
232	172	106	154	226	023	057
054	141	216	149	153	142	071

פענוח ההצפנה נעשה בדיוק באותו אופן, מכיוון שפעולת XOR היא הופכית. כמובן שעלינו לדעת מהן נקודות-הציון ההתחלתיות ואיזו כיווניות נבחרה על מנת לשחזר את המסלול המדויק של ההצפנה.

מכיוון שאיננו יודעים את הנתונים ההתחלתיים של הקובץ msg.enc, ננסה לבצע Brute Force על מנת לגלות אותם. נוכל להניח שהקובץ מכיל טקסט בלבד, ולכן אם נקבל פענוח של תו מסוים שאינו בר-הדפסה, נדע שהנתונים ההתחלתיים שאנו מנסים כעת הינם שגויים ונוכל לעבור מיד לניסיון הבא.

הקוד הבא יבצע זאת:

```
import os
import mmap
import string

def memory_map(filename, access=mmap.ACCESS_READ):
    size = os.path.getsize(filename)
    fd = os.open(filename, os.O_RDONLY)
    return mmap.mmap(fd, size, access=access)

table_str = """
177 030 077 225 170 116 089
228 139 058 083 195 202 201
197 113 114 053 184 105 043
178 029 210 090 150 045 212
135 240 099 051 147 085 060
156 039 169 101 078 180 165
075 108 102 163 166 027 092
204 046 015 198 209 086 120
232 172 106 154 226 023 057
054 141 216 149 153 142 071
""".strip()

class PinballCipher(object):
    def __init__(self, table_str):
        self.table = [] # [Y] [X]
        for row in table_str.split("\n"):
            self.table.append([int(x) for x in row.split()])
        self.rows = len(self.table)
        self.columns = len(self.table[0])
```



```
def initialize(self, py, px, vy, vx):
    assert(0 <= px < self.columns)
    assert(0 <= py < self.rows)
    assert(vx == 1 or vx == -1)
    assert(vy == 1 or vy == -1)

    self.py = py
    self.px = px
    self.vy = vy
    self.vx = vx
    self.reset = True

def next(self):
    yield self.table[self.py][self.px]
    self.reset = False
    while not self.reset:
        expected_y = self.py + self.vy
        expected_x = self.px + self.vx

        if expected_y < 0 or expected_y >= self.rows:
            self.vy *= -1
        if expected_x < 0 or expected_x >= self.columns:
            self.vx *= -1

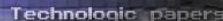
        self.py += self.vy
        self.px += self.vx
        yield self.table[self.py][self.px]

with memory_map("msg.enc") as ciphertext:
    ciphertext_len = len(ciphertext)
    pc = PinballCipher(table_str)
    for y in range(pc.rows):
        for x in range(pc.columns):
            for vy in [1, -1]:
                for vx in [1, -1]:
                    res = ""
                    pc.initialize(y, x, vy, vx)
                    for i, c in zip(range(ciphertext_len), pc.next()):
                        xored_c = chr(ciphertext[i] ^ c)
                        if xored_c not in string.printable:
                            break
                        res += xored_c
                    else:
                        print (res)
                        print ((y, x, vy, vx))
                        print ("\n")
```

נריץ את הקוד ונקבל:

```
root@kali:/media/sf_CTFs/checkpoint/Pinball_Cipher/pinball# python3 solve.py
Congrats! The flag for the pinball cipher is: CSA{wE_sh0uLd_hav3_BeEn_n1cer_t0_oUr_siST3r}
(4, 6, -1, 1)

Congrats! The flag for the pinball cipher is: CSA{wE_sh0uLd_hav3_BeEn_n1cer_t0_oUr_siST3r}
(4, 6, -1, -1)
```

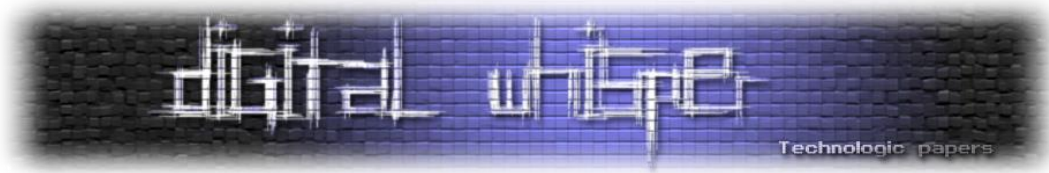



The king of Sheshach shall drink after them

[illegible]

```
root@kali:/media/sf_CTFs/checkpoint/BadaBase# cat ciphertext | base64 -d | xxd -g 1 | head
00000000: dd ab 97 9a df 94 96 91 98 df 90 99 df ac 97 9a .....
00000010: 8c 97 9e 9c 97 df 8c 97 9e 93 93 df 9b 8d 96 91 .....
00000020: 94 df 9e 99 8b 9a 8d df 8b 97 9a 92 dd df b6 df .....
00000030: 97 90 8f 9a df 86 90 8a df 9b 90 91 d8 8b df 8b .....
00000040: 8d 86 df 96 8b df 92 9e 91 8a 9e 93 93 86 df c4 .....
00000050: d6 df 9b 98 ce cc cc 99 8c 9b 98 95 9e cb ca cc .....
00000060: cc cb ca ca 8c cc cd ce cc cc cc 9b 9e 9b 9a 88 .....
00000070: 8c 9e 99 ca cb cc ca 9a 88 8b 97 98 99 97 98 86 .....
00000080: 8a 8b 9a 8d 9b 94 8c 95 9e 96 9b 9a 99 99 8c 9b .....
00000090: 8c 8f d7 de 9b 9b cc cc cb cd cb cd 99 9b cc cb .....
```

באופן קסמי למדי, התוצאה הזו מתקבלת כאשר מבצעים XOR עם 0xFF. למשל, אם נבצע $0xFF \oplus 0xFE$ נקבל 0x01. כמו שרצינו.



נשתמש בסקריפט הקצר הבא:

```
from Crypto.Util.strxor import strxor_c
import base64

with open("ciphertext") as f:
    text = f.read()
    print strxor_c(base64.b64decode(text), 0xFF)
```

התוצאה:

```
root@kali:/media/sf_CTFs/checkpoint/BadaBase# python decrypt.py
"The king of Sheshach shall drink after them" I hope you don't try it
manually ;) dg133fsdgja4533455s321333dadewsaf5435ewthgfhgyuterdksjai
deffsdsp(!dd334242fd342sjkfhsdjkf$adasdadasd32457sdfsfsfsdadw#$@$ada
sdadasd324576832wfes# ddsadddsaja4533455s545433dadewsaf5435ewthgfhgyu
ter@!sjaiAAAsdsp(!dd3342434342sjkfhsdjkf$adasdadasd32457sdfsfsfsdadw
d#$@sdadasd324576832wfedss# dg133fsdgja4533455s321333dadewsaf5435e
wthgfhgyuterdksjaideffsdsp(!dd334242fd342sjkfhsdjkf$adasdadasd32457sd
fsfsfsfsdadw#$@$adasdadasd324576832wfes# dg133fsdgja4533455s321333dade
wsaf5435ewthgfhgyuterdksjaideffsdsp(!dd334242fd342sjkfhsdjkf$adasdada
sd32457sdfsfsfsdadw#$@$adasdadasd324576832wfes# dtfgfIsdgja4533455s5
45433dadewsaf5435ewthgfhgyuterdksjaideffsdsp(!dd334242fd342sjkfhsdjkf
$adasdadasd32457sdfsfsfsdadw#$@$adasdadasd324576832wfes# "The king o
f Sheshach shall drink after them" ... ndgs3424effsdsp(!dd324dfdsf&2d
sf&fdssfs#$fd342sjkfhsdjkfsdfsfs543gfdgdfdsfsfsfsk9809dfsf90sfds fs
atdkui^%$;lkl;sdfs;l;arkdffb<fs>REW54c. ##### FLAG IN FOLLOWING QUOTES
:"C.S.A. (2019)-Rulz-{@!*~*!@>} A.S.C." ##### .hdsa434524345sdgffdsf
sfsdsg dg435fsdgja4533455s321333dadewsaf5435ewthgfhgyuterdksjaideffsd
sp(!dd334242fd342sjkfhsdjkf$adasdadasd32457sdfsfsfsdadw#$@$adasdadas
d324576832wfes# dg435fsdgja45dsadadadasd98d90sa809daskdlasd708a78d098
09sa;dd324dfdsf&2dsf&dsadsadsfdssfs#$fd342sjkfhsdjkfsdfsfsfsdadw#$@$
adasdadasd324576832wfes# ddsadddsaja4533455s545433dadewsaf5435ewthgf
gyuter@!sjaiBBBsdsp(!dd3342434342sjkfhsdjkf$adasdadasd32457sdfsfsfsd
adwd#$@sdadasd324576832wfedss# ddsadddsaja4533455s545433dadewsaf54
35ewthgfhgyuter@!sjaiAAAsdsp(!dd3342434342sjkfhsdjkf$adasdadasd32457s
dfsfsfsfsdadwd#$@sdadasd324576832wfedss# ddsadddsaja4533455s545433d
adewsaf5435ewthgfhgyuter@!sjaiAAAsdsp(!ddasdadadsdskfhsdjkf$adasdad
asd32457sdfsfsfsdadwd#$@sdadasd324576832wfedss# ddsadddsaja453345
5s545433dadewsaf52^5ewthgfhgyuter@!sjaiAAAsdsp(!dd3342434342sjkfhsdjk
f$adasdadasd32457sdfsfsfsdadwd#$@sdadasd324576832wfedss# ddsaddds
aja4533455s545433dadewsaf5435ewthgfhgyuter@!sjaiAAAsdsp(!dd31424cxzcz2
sjkfhsdjkf$adasdadasd32457sdfsfsfsdadwd#$@sdadasd324576832wfedss#
ddsadddsawQ4533455s545433dadewsaf5435ewthgfhgyuterdksjaideffsdsp(!dd
334242fd342sjkfhsdjkf$adasdadasd32457sdfsfsfsdadwd#$@sdadasd32457
6832wfes# dg133fsdgja4533455s321333dadewsaf5435ewthgfhgyuterdksjaidef
fsdsp(!dd334242fd342sjkfhsdjkf$adasdadasd32457sdfsfsfsdadw#$@$adasda
dasd324576832wfes# ddsadddsaja4533455s545433dadewsaf5435ewthgfhgyuter
@!sjaiAAAsdsp(!dd3342434342sjkfhsdjkf$adasdadasd32457sdfsfsfsdadwd#$
$adasdadasd324576832wfedss# "The king of Sheshach shall drink after t
hem" I hope you don't try it manually ;)
```

הדגל

```
C.S.A. (2019)-Rulz-{@!*~*!@>} A.S.C.
```



אתגר #3: Hunting Tinba (קטגוריה: שונות. ניקוד: 20)

The Cyber Law Enforcement Agency in Bolivia has raided a local cyber-crime group RnD center. They discovered that the cyber-crime group had breached dozens of organizations all over the world. All of the victims were infected with the Tinba malware.

(https://en.wikipedia.org/wiki/Tiny_Banker_Trojan) The Bolivian agency asked California State police assistance in shutting down the cyber-criminals' data center.

Acme inc. CISO was notified by the Bolivian agency that his organization is one of the victims that were breached.

Help Acme inc. CISO uncover the needle in the haystack by analyzing the attached Firewall log. The log contains a few million outbound connections. Your mission is to find the smoking gun of Tinba infection and discover the IP address of the CnC server.

Acme inc. CISO is specifically interested in the amount of unique infected machines in the time windows between 18:00 06:00, this answer is your flag!

FYI - YOU HAVE A 5 ATTEMPTS LIMIT ON THIS CHALLENGE

לאתגר צורף קובץ לוג ארוך מאוד בפורמט הבא:

```
root@kali:/media/sf_CTFs/checkpoint/Hunting_Tinba/fw# head fw.log
time_stamp      src_ip  dst_ip
2019-05-27 00:00:00 10.0.3.63 104.17.100.211
2019-05-27 00:00:00 10.0.3.47 107.22.248.119
2019-05-27 00:00:00 10.0.2.72 220.226.182.143
2019-05-27 00:00:00 10.0.2.252 146.243.122.35
2019-05-27 00:00:00 10.1.1.121 178.33.34.145
2019-05-27 00:00:00 10.0.0.2 104.27.162.194
2019-05-27 00:00:00 10.0.3.95 47.246.2.232
2019-05-27 00:00:00 10.0.1.218 45.60.124.77
2019-05-27 00:00:00 10.0.0.206 212.89.96.10
root@kali:/media/sf_CTFs/checkpoint/Hunting_Tinba/fw# wc -l fw.log
5000001 fw.log
```

עלינו למצוא את כתובת ה-IP של שרת השליטה והבקרה של הנוזקה, ומשם נוכל לספור כמה מחשבים נגועים יצרו קשר עם השרת בשעות שצוינו.

אפשר להפעיל המון יוריסטיקות שונות על מנת למצוא מועמדים מתאימים, אך במקרה הזה (עבור 20 נקודות) האסטרטגיה הפשוטה ביותר היא זו שעבדה:

- עבור כל כתובת יעד, נספור את כמות הפעמים שפנו אליה בסך הכל
- נמין את התוצאות לפי סדר יורד, כאשר ככל שפנו לכתובת מסוימת יותר, כך גדל הסיכוי שהיא השרת שאנו מחפשים



הסקריפט הבא יבצע זאת, ויספור גם את מספר הפעמים שפנו אל חמשת המועמדים המובילים בטווח שהשאלה הגדירה:

```
from collections import defaultdict
from datetime import datetime
import operator

ranges = [(datetime.strptime('2019-05-27 18:00:00', '%Y-%m-%d
%H:%M:%S'), datetime.strptime('2019-05-28 06:00:00', '%Y-%m-%d
%H:%M:%S') ),
          (datetime.strptime('2019-05-28 18:00:00', '%Y-%m-%d
%H:%M:%S'), datetime.strptime('2019-05-29 06:00:00', '%Y-%m-%d
%H:%M:%S') )]

common_ips = defaultdict(int)
ips_in_range = defaultdict(int)

print "Counting..."

with open("fw.log") as f:
    for line in f:
        try:
            date, time, src_ip, dest_ip = line.split()
            common_ips[dest_ip] += 1

            entry_time = datetime.strptime(date + ' ' + time, '%Y-%m-%d
%H:%M:%S')

            for (start_time, end_time) in ranges:
                if start_time <= entry_time < end_time:
                    ips_in_range[dest_ip] += 1

        except:
            pass

print "Sorting..."

sorted_common_ips = sorted(common_ips.items(),
key=operator.itemgetter(1), reverse = True)

print "Result:"

for ip, count in sorted_common_ips[:5]:
    print "IP: {} \t Total Count: {} \t Count in Range: {}".format(ip,
count, ips_in_range[ip])
```

התוצאה:

```
root@kali:/media/sf_CTFs/checkpoint/Hunting_Tinba/fw# python solve.py
Counting...
Sorting...
Result:
IP: 107.154.112.80      Total Count: 80 Count in Range: 34
IP: 192.85.6.161      Total Count: 80 Count in Range: 34
IP: 46.29.101.84      Total Count: 80 Count in Range: 30
IP: 205.186.165.29    Total Count: 79 Count in Range: 28
IP: 83.242.155.166    Total Count: 79 Count in Range: 32
```

המספר שהתקבל בתור תשובה הוא 32.



אתגר #4: Pretty Damn Funny - (קטגוריה: שונות. ניקוד: 70)

In the heat of the battle we managed to intercept a raven flying over the battlefield. The raven carried a USB that contained a file with a message for the enemy. Help us recover the message from the file!

Hint: You may be interested in reading about Incremental updates.

לאתגר צורף קובץ PDF.

כאשר פותחים את קובץ ה-PDF באמצעות קורא PDF ויזואלי, מקבלים דף ריק לחלוטין. עם כלי מבוסס שורת-פקודה, אנחנו מקבלים תוצאה שונה במקצת:

```
root@kali:/media/sf_CTFs/checkpoint/Pretty_Damn_Funny# pdftotext raven.pdf - read between the lines
```

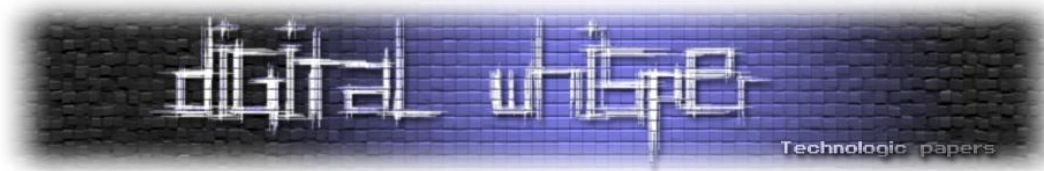
קובץ PDF הוא בבסיסו קובץ ASCII (למרות שלעיתים הוא יכול להכיל תוכן בינארי), מה שאומר שאפשר לפתוח אותו באמצעות עורך טקסט ולצפות במבנה שלו. למשל, הקובץ שלנו מתחיל כך:

```
root@kali:/media/sf_CTFs/checkpoint/Pretty_Damn_Funny# cat raven.pdf | head
%PDF-1.4
%
1 0 obj
<</Length 62/#46#69#6C#74#65#72/#46#6C#61#74#65#44#65#63#6F#64#65>>stream
xs
w3T02I5TH(JMLQHJ-)OMyR@d*d[x]C Ÿi
endstream
endobj
2 0 obj
<</Length 62/#46#69#6C#74#65#72/#46#6C#61#74#65#44#65#63#6F#64#65>>stream
```

ומסתיים כך:

```
root@kali:/media/sf_CTFs/checkpoint/Pretty_Damn_Funny# cat raven.pdf | tail && echo
xref
0 1
0000000000 65535 f
69 1
0000016958 00000 n
trailer
<</Size 71/Root 67 0 R/Info 70 0 R/Prev 16826>>
startxref
17042
%%EOF
```

כמובן שקוראי PDF יודעים לתרגם את התוכן הזה לדף שאנו רואים ויזואלית.



אחת התכונות של קובץ PDF היא התמיכה ב-Incremental Updates - כלומר, היכולת לעדכן מסמך על ידי הוספת הגרסה החדשה של המסמך **לאחר** הגרסה הנוכחית, ולא במקומה. למשל, על ידי שימוש ביכולת הזו, נוכל ליצור קובץ PDF כזה (בהפשטה):

Version 1
Version 2
Version 3

כשנפתח את הקובץ עם קורא PDF, נראה רק את הגרסה האחרונה. אולם, נוכל לשחזר כל אחת מהגרסאות הקודמות על ידי "מחיקת" הגרסה האחרונה. בפועל, ההפרדה בין גרסאות נעשית באמצעות המילה השמורה %%EOF, ו"מחיקת" גרסה היא בסך הכל התעלמות מכל מה שנמצא אחרי ה-EOF של הגרסה שנרצה להציג.

בקובץ שלנו, %%EOF חוזר 23 פעמים, מה שאומר שיש לנו 23 גרסאות שונות:

```
root@kali:/media/sf_CTFs/checkpoint/Pretty_Damn_Funny# cat raven.pdf | grep -c %%EOF
23
```

בעקרון אפשר להשתמש בתוכנה כמו [pdfresurrect](#) על מנת לחלץ את כל הגרסאות הקודמות של המסמך, אך בפועל יוצא שמדובר ב-23 מסמכים שעל פניו דומים ויזואלית וטקסטואלית לגרסה האחרונה שכבר ראינו. אם כך, נצטרך לבחון את הקובץ בצורה מדויקת יותר. קיימים כלים שמציגים את עץ האובייקטים של קובץ PDF, למשל:

```
root@kali:/media/sf_CTFs/checkpoint/Pretty_Damn_Funny# pdf-parser raven.pdf
PDF Comment '%PDF-1.4\n'

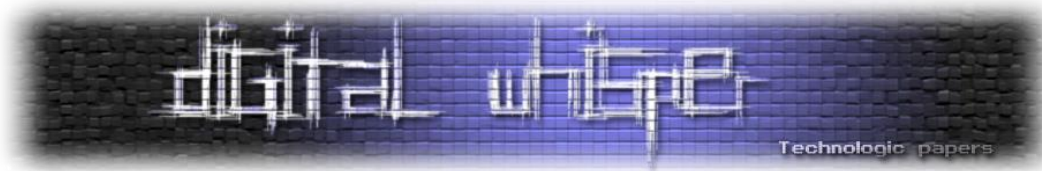
PDF Comment '%\xe2\xe3\xcf\xd3\n'

obj 1 0
Type:
Referencing:
Contains stream

  <<
    /Length 62
    /Filter /FlateDecode
  >>

obj 2 0
Type:
Referencing:
Contains stream

  <<
    /Length 62
    /Filter /FlateDecode
  >>
```



בחינה מדוקדקת של העץ העלתה שאובייקט 69 חוזר פעמים רבות עם הבדלים קלים בלבד:

```
root@kali:/media/sf_CTFs/checkpoint/Pretty_Damn_Funny# pdf-parser --object 69 raven.pdf
obj 69 0
Type: /Page
Referencing: 68 0 R, 20 0 R

<<
  /Type /Page
  /Parent 68 0 R
  /MediaBox [0 0 612 792]
  /Contents 20 0 R
>>

obj 69 0
Type: /Page
Referencing: 68 0 R, 60 0 R

<<
  /Type /Page
  /Parent 68 0 R
  /MediaBox [0 0 612 792]
  /Contents 60 0 R
>>

obj 69 0
Type: /Page
Referencing: 68 0 R, 48 0 R

<<
  /Type /Page
  /Parent 68 0 R
  /MediaBox [0 0 612 792]
  /Contents 48 0 R
>>
```

נסמן רק את השורה הרלוונטית:

```
root@kali:/media/sf_CTFs/checkpoint/Pretty_Damn_Funny# pdf-parser --object 69 raven.pdf | grep Referencing
Referencing: 68 0 R, 20 0 R
Referencing: 68 0 R, 60 0 R
Referencing: 68 0 R, 48 0 R
Referencing: 68 0 R, 46 0 R
Referencing: 68 0 R, 40 0 R
Referencing: 68 0 R, 42 0 R
Referencing: 68 0 R, 17 0 R
Referencing: 68 0 R, 44 0 R
Referencing: 68 0 R, 35 0 R
Referencing: 68 0 R, 56 0 R
Referencing: 68 0 R, 9 0 R
Referencing: 68 0 R, 45 0 R
Referencing: 68 0 R, 4 0 R
Referencing: 68 0 R, 35 0 R
Referencing: 68 0 R, 41 0 R
Referencing: 68 0 R, 19 0 R
Referencing: 68 0 R, 36 0 R
Referencing: 68 0 R, 35 0 R
Referencing: 68 0 R, 55 0 R
Referencing: 68 0 R, 21 0 R
Referencing: 68 0 R, 52 0 R
Referencing: 68 0 R, 53 0 R
Referencing: 68 0 R, 13 0 R
```



המספר המשתנה הוא הפנייה לאובייקט אחר, אותו אפשר לקרוא כך:

```
root@kali:/media/sf_CTFs/checkpoint/Pretty_Damn_Funny# peepdf -C "object 20" raven.pdf 2>&1

<< /Length 62
/Filter /FlateDecode >>
stream
BT
/F1 20 Tf
1 g
[(read between)] TJ%C
[( the lines)] TJ
ET
endstream
```

בדוגמא אנו רואים קריאה של אובייקט #20, כאשר תוכן האובייקט הכיל את הטקסט TJ%C. כל אובייקט הכיל תו אחר, וניתן היה לשלוף את הדגל במלואו על ידי הרצת הסקריפט הבא:

```
#!/bin/sh
pdf-parser --object 69 raven.pdf | grep -Po " Referencing: 68 0 R,
\\K(\\d+)" | while read -r line ; do
    new_char=$(peepdf -C "object $line" raven.pdf 2>&1 | grep -Po
"\\[\\(read between\\)\\]\\ TJ%\\K(\\.)";)
    echo -n $new_char
done
```

הדגל:

```
root@kali:/media/sf_CTFs/checkpoint/Pretty_Damn_Funny# ./solve.sh && echo
CSA{ke3P_caLm_4Nd_r7fM}
```




אתגר #5: Da Vinci - (קטגוריה: שונות. ניקוד: 70)

I ordered a reproduction of a famous renaissance painting, but the artist sent me this file. Can you help?

לאתגר צורף קובץ PCAP.

בהינתן קובץ PCAP, אנחנו רגילים למצוא בו תעבורת רשת, אך הוא יכול לשמש ללכידת כל סוג תעבורה. כל מה שצריך על מנת לצפות בתעבורה הוא שתוכנת הקצה (למשל: WireShark) תממש לוגיקה שיודעת להבין ולנתח את הפקטות שעוברות (dissector).

במקרה שלנו, קובץ ה-PCAP הכיל תעבורת USB:

No.	Time	Source	Destination	Protocol	Length	Leftover Capture Data	Info
1	0.000000	host	2.6.0	USB	36		GET_DESCRIPTOR Request DEVICE
2	0.000624	2.6.0	host	USB	46		GET_DESCRIPTOR Response DEVICE
3	0.000626	2.6.0	host	USB	28		GET_DESCRIPTOR Status
4	0.000673	host	2.6.0	USB	36		GET_DESCRIPTOR Request CONFIGURATION
5	0.001053	2.6.0	host	USB	37		GET_DESCRIPTOR Response CONFIGURATION
6	0.001055	2.6.0	host	USB	28		GET_DESCRIPTOR Status
7	0.001081	host	2.6.0	USB	36		GET_DESCRIPTOR Request CONFIGURATION
8	0.002097	2.6.0	host	USB	87		GET_DESCRIPTOR Response CONFIGURATION
9	0.002099	2.6.0	host	USB	28		GET_DESCRIPTOR Status
10	0.007123	host	2.6.0	USB	36		SET_CONFIGURATION Request
11	0.007125	host	2.6.0	USB	28		URB_CONTROL out
12	0.007951	2.6.0	host	USB	28		GET_STATUS Status
13	0.012449	host	2.6.0	USB	36		GET_DESCRIPTOR Request STRING
14	0.012750	2.6.0	host	USB	32		GET_DESCRIPTOR Response STRING

ראשית, נבחן בתור מה מכשיר ה-USB מזדהה:

>	Frame 2: 46 bytes on wire (368 bits), 46 bytes captured (368 bits)
>	USB URB
▼	DEVICE DESCRIPTOR
	bLength: 18
	bDescriptorType: 0x01 (DEVICE)
	bcdUSB: 0x0200
	bDeviceClass: Device (0x00)
	bDeviceSubClass: 0
	bDeviceProtocol: 0 (Use class code info from Interface Descriptors)
	bMaxPacketSize0: 64
	idVendor: Wacom Co., Ltd (0x056a)
	idProduct: CTL-471 [Bamboo Splash, One by Wacom (S)] (0x0300)
	bcdDevice: 0x0100
	iManufacturer: 1
	iProduct: 2
	iSerialNumber: 0
	bNumConfigurations: 1

מדובר במכשיר של חברת Wacom Co. Ltd מדגם CTL-471:



זהו בעקרון לוח כתיבה אלקטרוני, אך במקרה שלנו הוא דווקא מזדהה כעכבר:

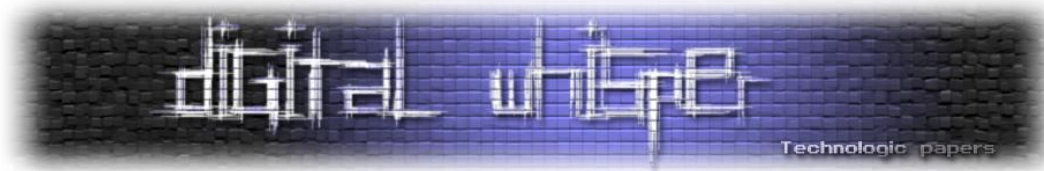
```

> Frame 8: 87 bytes on wire (696 bits), 87 bytes captured (696 bits)
> USB URB
> CONFIGURATION DESCRIPTOR
▼ INTERFACE DESCRIPTOR (0.0): class HID
    bLength: 9
    bDescriptorType: 0x04 (INTERFACE)
    bInterfaceNumber: 0
    bAlternateSetting: 0
    bNumEndpoints: 1
    bInterfaceClass: HID (0x03)
    bInterfaceSubClass: Boot Interface (0x01)
    bInterfaceProtocol: Mouse (0x02)
    iInterface: 0
    
```

מבחינת התעבורה, נראה שעיקר התעבורה נשלח בשדה שנקרא "Leftover Capture Data" וש-WireShark לא יודע לנתח אותו:

No.	Time	Source	Destination	Protocol	Length	Leftover Capture Data	Info
36	4.801067	2.6.1	host	USB	37	c000000000000000...	URB_INTERRUPT in
37	5.707076	2.6.1	host	USB	31	018000fd	URB_INTERRUPT in
38	5.712987	2.6.1	host	USB	31	01000000	URB_INTERRUPT in
39	5.751024	2.6.1	host	USB	31	01800000	URB_INTERRUPT in
40	5.767079	2.6.1	host	USB	31	018000fe	URB_INTERRUPT in
41	5.772995	2.6.1	host	USB	31	01000000	URB_INTERRUPT in
42	5.789073	2.6.1	host	USB	31	01800002	URB_INTERRUPT in
43	5.803024	2.6.1	host	USB	31	01800004	URB_INTERRUPT in
44	5.811021	2.6.1	host	USB	31	01800005	URB_INTERRUPT in
45	5.819031	2.6.1	host	USB	31	01800002	URB_INTERRUPT in
46	5.827024	2.6.1	host	USB	31	01800203	URB_INTERRUPT in
47	5.833023	2.6.1	host	USB	31	01800004	URB_INTERRUPT in
48	5.841049	2.6.1	host	USB	31	01800003	URB_INTERRUPT in
49	5.848993	2.6.1	host	USB	31	01800004	URB_INTERRUPT in
50	5.857011	2.6.1	host	USB	31	01800003	URB_INTERRUPT in
51	5.862985	2.6.1	host	USB	31	01800004	URB_INTERRUPT in
52	5.870997	2.6.1	host	USB	31	01800003	URB_INTERRUPT in
53	5.879000	2.6.1	host	USB	31	01800002	URB_INTERRUPT in
54	5.887021	2.6.1	host	USB	31	0180fe03	URB_INTERRUPT in

המטרה, אם כך, היא לנסות לשחזר את תנועות העכבר על מנת לקבל את הדגל.



ממחקר קצר באינטרנט (למשל [פה](#)) עלה כי עכברים נוהגים לשלוח פקטות של שלושה או ארבעה בתים שבהם מקודד המידע שמעיד על תזוזה, לחיצה וכד'. ואכן, אצלנו רוב הפקטות הן של ארבעה בתים. נתחיל מחילוצן מתוך קובץ התעבורה על מנת שיהיה קל יותר לעבוד איתן.

כך נראית הודעת USB המכילה פקטת מידע של ארבעה בתים:

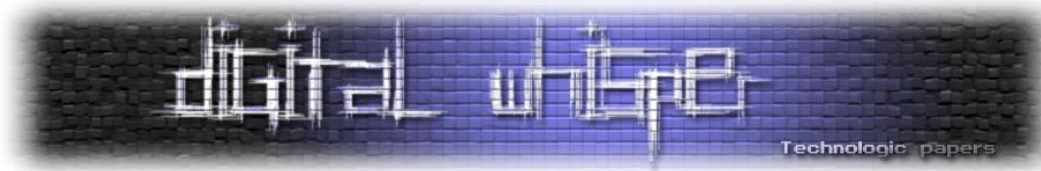
```
> Frame 37: 31 bytes on wire (248 bits), 31 bytes captured (248 bits) on 0
USB_URB
  [Source: 2.6.1]
  [Destination: host]
  USBPcap pseudoheader length: 27
  IRP ID: 0xffffe105279d4a60
  IRP USBD_STATUS: USBD_STATUS_SUCCESS (0x00000000)
  URB Function: URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER (0x0009)
  > IRP information: 0x01, Direction: PDO -> FDO
  URB bus id: 2
  Device address: 6
  > Endpoint: 0x81, Direction: IN
  URB transfer type: URB_INTERRUPT (0x01)
  Packet Data Length: 4
  [bInterfaceClass: HID (0x03)]
  Leftover Capture Data: 018000fd
```

נסמן את כל ההודעות עם Device Address ל-6 ונחלץ מהן את ה-Leftover Capture Data:

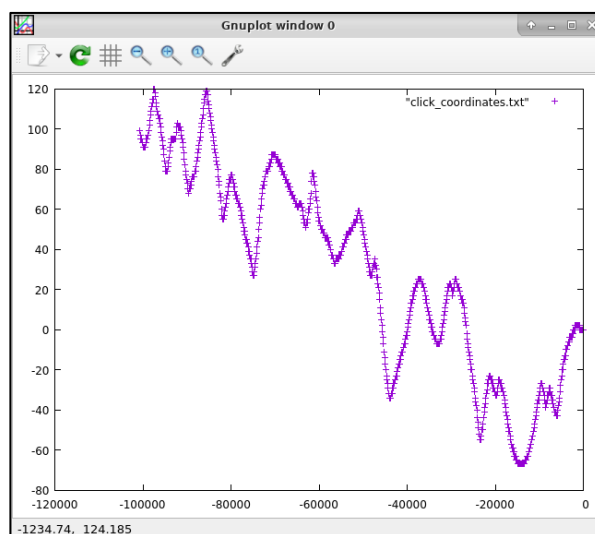
```
root@kali:/media/sf_CTFs/checkpoint/Da_Vinci# tshark -r davinci.pcap -T fields -e usb.capdata,usb.device_address==6 2> /dev/null > capture_data.txt
root@kali:/media/sf_CTFs/checkpoint/Da_Vinci# head capture_data.txt
c0:00:00:00:00:00:00:00:00
01:80:00:fd
01:00:00:00
01:80:00:00
01:80:00:fe
01:00:00:00
01:80:00:02
01:80:00:04
01:80:00:05
01:80:00:02
```

אם נחזור לקישור המידע הקודם, נגלה שהוא מפרט בדיוק את מבנה הפקטות הצפוי. מה שמעניין אותנו הוא כנראה:

- הביט הראשון בבית הראשון: האם הכפתור השמאלי לחוץ
- הבית השני: תזוזת ה-X
- הבית השלישי: תזוזת ה-Y



עם זאת כל ניסיון לפרש כך את המידע שברשותנו נכשל כשלון חרוץ. הנתונים פשוט לא מסתדרים ומציירים יצירות אבסטרקטיות בסגנון:



נראה שעלינו לפרש את המידע בצורה אחרת. ננסה להוציא סטטיסטיקה של הערכים השונים שניתן למצוא בכל אחד מארבעת הבתים:

```
root@kali:/media/sf_CTFs/checkpoint/Da_Vinci# awk -F: '{print$1}' capture_data.txt | sort | uniq -c
5490 01
51 c0
root@kali:/media/sf_CTFs/checkpoint/Da_Vinci# awk -F: '{print$2}' capture_data.txt | sort | uniq -c
160 00
3153 80
2228 81
root@kali:/media/sf_CTFs/checkpoint/Da_Vinci# awk -F: '{print$3}' capture_data.txt | sort | uniq -c
2700 00
1431 02
233 03
63 04
18 05
4 06
1 f9
4 fa
10 fb
39 fc
122 fd
916 fe
root@kali:/media/sf_CTFs/checkpoint/Da_Vinci# awk -F: '{print$4}' capture_data.txt | sort | uniq -c
2625 00
1442 02
69 03
12 04
2 05
2 06
2 fb
16 fc
188 fd
1183 fe
```

לפי הסטטיסטיקה הזו, ניתן לראות שהבית הראשון הוא כמעט תמיד 0x01 (ולמעשה, כאשר הוא 0xC0, אורך הפקטה הוא מעל 4 בתים), ולכן לא ממש מעניין כנראה.

הבית השני נע לרוב בין 0x80 ל-0x81 ולכן הוא דווקא מסתדר טוב עם לחיצה על הכפתור השמאלי. והנתונים של הבתים השלישי והרביעי יחסית דומים, ומעבר לכך - שניהם מאוד קרובים ל-0x00 או ל-



0xFF. זה מאוד מתאים לנקודות-הציון X ו-Y שמיוצגות על ידי המשלים ל-2, כמו שעכבר אמיתי אמור לייצג. לכן, נראה שבסך הכל ההבדל מול ה-spec שראינו קודם הוא סדר הבתים (לפחות בנוגע למידע שמעניין אותנו).

לא נשאר הרבה, פשוט נדמה את תזוזות העכבר ונצייר נקודה כאשר הלחצן השמאלי נלחץ באמצעות הסקריפט הבא:

```
from dataclasses import dataclass
from PIL import Image, ImageDraw

@dataclass
class Coordinate:
    x: int
    y: int

class UsbMouseData(object):
    def __init__(self, raw_string_data):
        arr = list(map(lambda x: int(x, 16),
            raw_string_data.split(":")))
        if len(arr) != 4:
            raise ValueError("Error: Incorrect format")
        self.is_clicked = arr[1] & 1
        self.x = self.twos_complement(arr[2])
        self.y = self.twos_complement(arr[3])

    @staticmethod
    def twos_complement(val, bits = 8):
        if (val & (1 << (bits - 1))) != 0:
            val = val - (1 << bits)
        return val

def draw_dot(draw_obj, x, y, radius = 1):
    draw_obj.ellipse((x - radius, y - radius, x + radius, y + radius),
        fill = 0)

def create_mouse_outline(input_path, output_path):
    with open(input_path) as f:
        img_size = 4000
        img = Image.new("L", (img_size, img_size), 255)
        draw_obj = ImageDraw.Draw(img)
        position = Coordinate(img_size / 2, img_size / 2)

        for line in f:
            try:
                line = line.rstrip()
                data = UsbMouseData(line)

                position.x += data.x
                position.y += data.y

                if data.is_clicked:
                    draw_dot(draw_obj, position.x, position.y, radius =
2)
            except:
                print ("Error with line: {}".format(line))
```

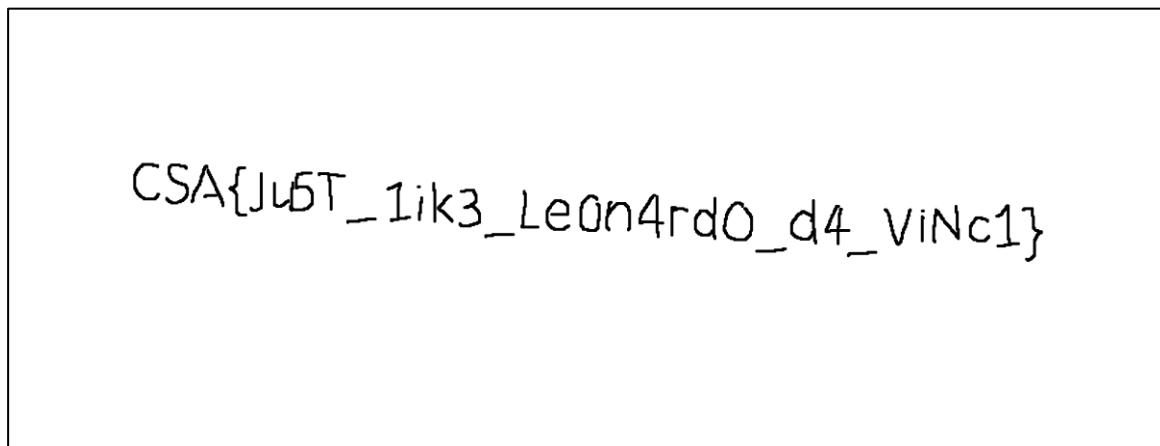


```
pass

img.save(output_path)

if __name__ == "__main__":
    create_mouse_outline("capture_data.txt", "res.png")
```

התוצאה היא:



כלומר, הדגל הוא:

```
CSA{JuST_1ik3_Le0n4rd0_d4_ViNc1}
```



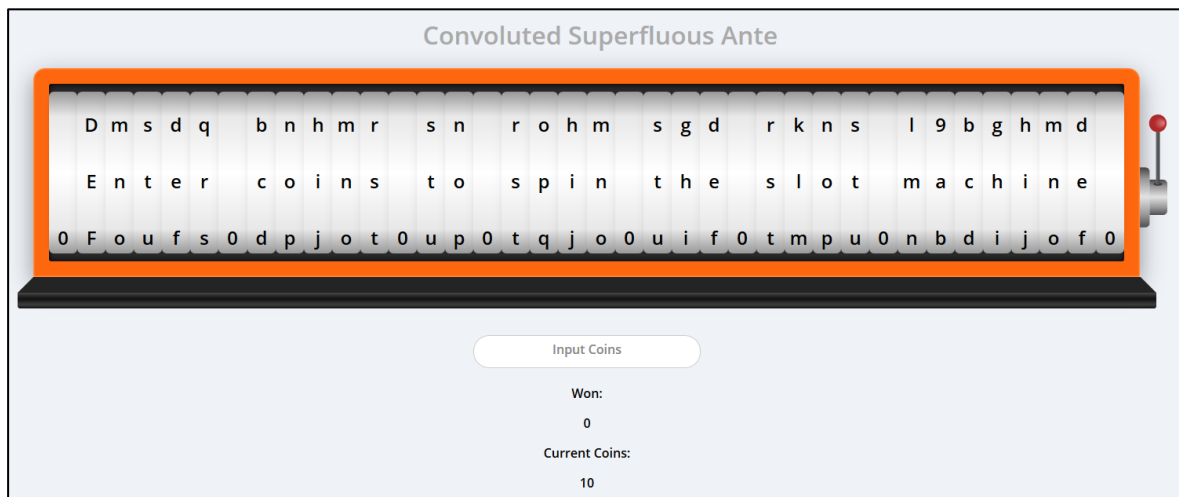
אתגר #6: Slot machine - (קטגוריה: רברסינג. ניקוד: 30)

Win big with our slot machine! But the real prize is a secret, can you take it all?

Slot machine is here: <http://csa.bet/>

Slot machine script running in the backend is in `slotmachine_dummy.py`

האתר המצורף הכיל מכונת מזל:



בתיאור האתגר נטען כי המימוש של מכונת המזל הוא:

```
import random
import collections

from .secret import flag

PRINTABLE =
"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!\"#$%&' (
)*+,-./:;<=>?@[\\]^_`{|}~"
flag_length = len(flag)
SLOT_LENGTH = 10
NO_COINS = "No more coins! Goodbye."
NOT_ENOUGH_COINS = "You don't have enough coins!"
INVALID_COIN_NUMBER = "Coin number can't be negative"
INITIAL_COINS = 10

class Slotmachine(object):
    def __init__(self):
        self.slots = [[i]+[random.choice(PRINTABLE) for i in
range(SLOT_LENGTH)] for i in flag]
        self.attempt_num = 0
        self.total_coins = INITIAL_COINS
        self.last_result = ""
        self.last_gamble = 0

    def get_prize(self):
        result = self.last_result
```



```
prize = sum([x for x in collections.Counter(result).values() if
x > 2])
prize *= self.last_gamble
self.total_coins += prize
return prize

def check_invalid_input(self, coins):
    if self.total_coins <= 0:
        self.last_result = ""
        return NO_COINS
    if self.total_coins < coins:
        self.last_result = ""
        return NOT_ENOUGH_COINS
    if coins < 0:
        self.last_result = ""
        return INVALID_COIN_NUMBER
    return None

def spin(self, coins):
    invalid_message = self.check_invalid_input(coins)
    if invalid_message:
        return invalid_message.center(flag_length, ' ')

    self.last_gamble = coins
    self.total_coins -= coins

    random.seed(coins + self.attempt_num)
    self.attempt_num += 1

    for i in self.slots:
        random.shuffle(i)

    result = ""
    for i in self.slots:
        result += random.choice(i)
    self.last_result = result
    return result

# This is used to run the slotmachine locally, the server doesn't use
this.
def main():
    slotmachine = Slotmachine()
    print("You have {} coins".format(slotmachine.total_coins))
    get_next_num = True
    while get_next_num:
        try:
            prize = 0
            coins = int(input("Enter number of coins:\n"))
            result = slotmachine.spin(coins)
            if result == NO_COINS:
                get_next_num = False
            elif result != NOT_ENOUGH_COINS:
                prize = slotmachine.get_prize()
            print(result)
            print("You won {} coins!".format(prize))
            print("{} coins left.".format(slotmachine.total_coins))

        except ValueError:
            get_next_num = False
```




```
except NameError:
    get_next_num = False

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

מכונת המזל הזו מבקשת מאיתנו להכניס מספר מטבעות ולסובב את הידית. סיבוב הידית גורר לוגיקה מסוימת שבסופה יוחלט בכמה מטבעות נזכה, אם בכלל. הלוגיקה הזו פחות מעניינת בפני עצמה, נתרכז רק בחלקים שמאפשרים לנו להדליף את הדגל.

בכלל, איפה הדגל בכל הסיפור הזה? הדגל מגיע ממודול אחר ומשמש לאתחול משתנה מחלקה בשם slots:

```
self.slots = [[i]+[random.choice(PRINTABLE) for i in range(SLOT_LENGTH)]
for i in flag]
```

כלומר, slots הוא רשימה (באורך הדגל) של רשימות (באורך 10 כל אחת). כל תת-רשימה מתחילה בתו ה-i של הדגל, ולאחריה תשעה תווים אקראיים ("ריפוד"). או לפחות, זהו המצב ההתחלתי של המשתנה, שהרי הוא מעורבב בכל פעם שמתבצעת קריאה ל-spin, כלומר כאשר אנו מסובבים את הידית. כאשר זה קורה, התוכנה מבצעת את הלוגיקה הבאה:

```
random.seed(coins + self.attempt_num)
self.attempt_num += 1

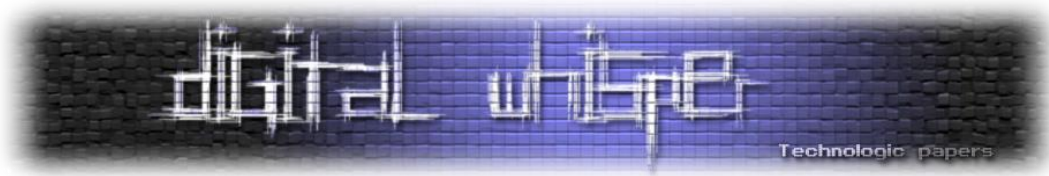
for i in self.slots:
    random.shuffle(i)

result = ""
for i in self.slots:
    result += random.choice(i)
self.last_result = result
```

אנחנו רואים פה אתחול של random seed באמצעות מספר המטבעות עליהן הימרנו (משתנה בשליטתנו) ומונה רץ של מספר הסיבובים שהיו עד עתה (משתנה ידוע לנו). כבר בנקודה הזאת, אמורה להידלק נורה אדומה לכל מי שמכיר את התיאוריה של מספרים אקראיים בעולם מדעי המחשב. הגרלת מספר אקראי לחלוטין היא משימה יחסית קשה (עוד על כך [פה](#)), והמימושים הבסיסיים הקשורים למספרים אקראיים לרוב מספקים תחליף זול בהרבה: מספרים [פסאודו-אקראיים](#). כלומר, הם רק נראים אקראיים, אבל למעשה הם ניתנים לחיזוי מדויק להפליא. זה כמובן בסדר במקרים מסוימים. במקרים אחרים, זה פתח [לצרות צרורות](#).

כדי לדעת לחזות (או לשחזר) רצף של מספרים שהגיעו ממחולל מספרים פסאודו-אקראיים, כל מה שאנחנו צריכים לדעת הוא ה-seed שאיתו בוצע האתחול. וזה בדיוק מה שיש לנו במקרה הזה.

מה קורה הלאה? ספריית random משמשת לערבוב slots (אנחנו יכולים לחזות את הערבוב) ואז משמשת לבחירת תו אחד מכל תת-מערך (את מיקומו אנו יכולת לחזות) ליצירת המשתנה result.



המשתנה result ייראה בתור מחרוזת אקראית באורך הדגל שכוללת לעיתים תווים מהדגל במיקום לא ידוע, אך למעשה נוכל לזהות מתוכה את התווים שהגיעו מהדגל עצמו ולסנן החוצה את ה"רעש".

אבל - לפני הכל, נתחיל מגילוי אורך הדגל, שמשתקף באורך המשתנה slots, שמשתקף באורך result:

```
root@kali:/media/sf_CTFs/checkpoint/Slot_machine# curl http://csa.bet/ -I
HTTP/1.1 200 OK
Server: nginx/1.14.0 (Ubuntu)
Date: Thu, 25 Jul 2019 19:56:08 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 2926
Connection: keep-alive
X-Frame-Options: SAMEORIGIN
Vary: Cookie
Set-Cookie: sessionId=hro6qe7fyhf58corhre8b34586o2wi99; expires=Thu, 08 Aug 2019 19:56:08 GMT; HttpOnly; Max-Age=1209600; Path=/; SameSite=Lax

root@kali:/media/sf_CTFs/checkpoint/Slot_machine# curl http://csa.bet/spin/?coins=1 -H "Cookie: sessionId=hro6qe7fyhf58corhre8b34586o2wi99;" && echo
{"result": "IgZaC;06FDJM[=lMd29:gA{c74Sp/,J_@??lXU", "prize": 0, "current_coins": 9}
```

קיבלנו חזרה מחרוזת באורך 38 תווים, וזהו אורך הדגל שלנו.

כעת אפשר להתחיל להדליף את הדגל. נעשה זאת על ידי הרצת שתי מכונות במקביל: מכונה מקומית (עם שינויים קלים שנפרט מיד) ומכונה מרוחקת (המכונה של השרת). נדאג שה-seed של שתי המכונות יהיה זהה, וכך נוכל להשתמש במידע שמתקבל מקומית על מנת להסיק מסקנות לגבי מה שקורה בשרת המרוחק.

נתחיל מאתחול מספר משתנים:

```
FLAG_LEN = 38
flag = "0" * FLAG_LEN
remote_flag = [None] * FLAG_LEN
```

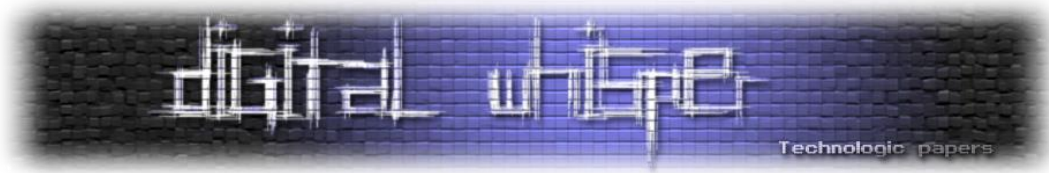
במקום לייבא את flag ממודול אחר (שאיננו בידינו), אנחנו מאתחלים אותו בתור מחרוזת אפסים. במקביל, אנחנו מורידים את "0" ממשתנה PRINTABLE על מנת לוודא שבכל פעם שנפגוש את התו "0", נדע שהוא הגיע מהדגל ולא בתור "ריפוד" אקראי.

נגדיר שתי פונקציות-עזר חדשות:

```
def setup_remote():
    s = requests.session()
    s.get("http://csa.bet")
    return s

def remote_spin(s, coins):
    r = s.get("http://csa.bet/spin/?coins={}".format(coins))
    j = json.loads(r.text)
    return j["result"]
```

הפונקציות הללו יסייעו לנו לסובב את המכונה המרוחקת יחד עם סיבוב המכונה המקומית. ל-`setup_remote()` נקרא בתחילת הפונקציה הראשית.



לבסוף, נשנה את הלולאה הראשית באופן הבא:

```
while get_next_num:
    prize = 0
    coins = 1
    result = slotmachine.spin(coins)
    remote_result = remote_spin(s, coins)
    if result == NO_COINS:
        get_next_num = False
    elif result != NOT_ENOUGH_COINS:
        prize = slotmachine.get_prize()
        if "0" in result:
            start = 0
            index = result.find("0", start)
            while index >= 0:
                remote_flag[index] = remote_result[index]
                print ("".join([c if c is not None else "?" for c in remote_flag ]))
                if all(remote_flag):
                    print ("".join(remote_flag))
                    return
            index = result.find("0", index + 1)
```

ניתן לראות פה שאנחנו מכניסים מטבע אחד בכל סיבוב, ומסובבים את המכונה המקומית ואת המכונה המרוחקת בדיוק באותו אופן, לקבלת שני משתני result - מקומי ומרוחק. לאחר מכן, אנחנו בודקים את התוצאה המקומית: אם קיים בה התו "0", סימן שבאותו המקום בתוצאה המרוחקת קיים תו מהדגל (שכן הערבוב הוא אותו ערבוב). ניקח את התו המתאים ונשבץ אותו ב-remote_flag. בתחילת הריצה הפלט נראה כך:

```
root@kali:/media/sf_CTFs/checkpoint/Slot_machine# python3 slotmachine.py
You have 10 coins
????????????????1????????????????p????????
????????????????1????????????????p????????
```

לאחר מספר לא רב של סיבובים, כבר יש לנו את רוב הדגל:

```
C????????????????1????????????????p????????
C????D????????????1????????????????p????????
C????D????_????1????????????????p????????
C????D????_????1????1????????p????????
C????D????_????1????1t????????p????????
C????D????_????1????1t????????pR????????
CS????D????_????1????1t????????pR????????
CS????D0????_????1????1t????????pR????????
CS????D0????_????1????1t_????pR????????
CS????D0????_????1????1t_????pR????e???
CS????D0????_????1????1t_????pR????eD??
CS????D0??t_????1????1t_????pR????eD??
CS????D0??t_????1????1t_???u???pR????eD??
CS????D0??t_????1????1t_???u???pR????eD?}
CSA?D0??t_????1????1t_???u???pR????eD?}
CSA?D0n?t_????1????1t_???u???pR????eD?}
CSA?D0n?t_????1????1t_???u???pR????eD?}
CSA?D0n?t_???b1?_1t_???u???pR????eD?}
CSA?D0n?t_???b1?_1t_???u???pR????eD?}
CSA?D0n?t_???b1?_1t_???u???pR????eD?}
CSA?D0n?t_???b1?_1t_???u???pRn????eD?}
CSA?D0n?t_???b1?_1t_???u???pRn????eD?}
CSA?D0n?t_???b1?_1t_???u???pRn????eD?}
```

בשביל התו האחרון צריך לסובב ולסובב, אבל בסוף הוא מתקבל:

```
CSA{D0n't_G?mb13_W1th_youR_pRnG_SeeD5}  
CSA{D0n't_G?mb13_W1th_youR_pRnG_SeeD5}  
CSA{D0n't_G?mb13_W1th_youR_pRnG_SeeD5}  
CSA{D0n't_G?mb13_W1th_youR_pRnG_SeeD5}  
CSA{D0n't_G?mb13_W1th_youR_pRnG_SeeD5}  
CSA{D0n't_G?mb13_W1th_youR_pRnG_SeeD5}  
CSA{D0n't_G?mb13_W1th_youR_pRnG_SeeD5}  
CSA{D0n't_G?mb13_W1th_youR_pRnG_SeeD5}  
CSA{D0n't_G?mb13_W1th_youR_pRnG_SeeD5}  
CSA{D0n't_G?mb13_W1th_youR_pRnG_SeeD5}  
CSA{D0n't_G?mb13_W1th_youR_pRnG_SeeD5}  
CSA{D0n't_G?mb13_W1th_youR_pRnG_SeeD5}  
CSA{D0n't_G?mb13_W1th_youR_pRnG_SeeD5}  
CSA{D0n't_G?mb13_W1th_youR_pRnG_SeeD5}  
CSA{D0n't_G?mb13_W1th_youR_pRnG_SeeD5}  
CSA{D0n't_G?mb13_W1th_youR_pRnG_SeeD5}  
CSA{D0n't_G?mb13_W1th_youR_pRnG_SeeD5}  
CSA{D0n't_G?mb13_W1th_youR_pRnG_SeeD5}  
CSA{D0n't_G?mb13_W1th_youR_pRnG_SeeD5}  
CSA{D0n't_G4mb13_W1th_youR_pRnG_SeeD5}  
CSA{D0n't_G4mb13_W1th_youR_pRnG_SeeD5}
```

הדגל:

```
CSA{D0n't_G4mb13_W1th_youR_pRnG_SeeD5}
```


אתגר #7: Prince of Persia (קטגוריה: רברסינג. ניקוד: 80)

In the Sultan's absence, the Grand Vizier JAFFAR rules with the iron fist of tyranny. Only one obstacle remains between Jaffar and the throne: the Sultan's beautiful young FLAG...

Marry Jaffar... or die within the hour. All the FLAG's hopes now rest on the brave youth it loves. Little does it know that he is already a prisoner in Jaffar's dungeons...

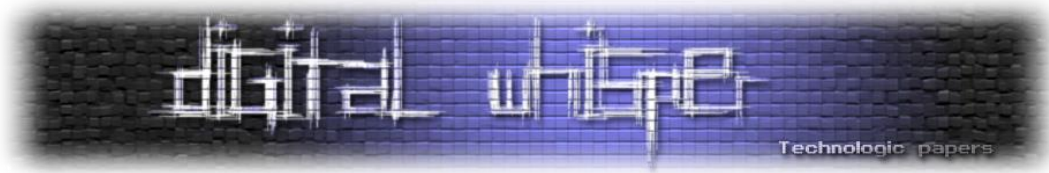
Reach the end and save the flag :)

לאתגר צורף קובץ ארכיון בשם SDLPoP-master.zip. נחלץ את הקובץ ונמצא שהוא מכיל מימוש מחודש למשחק הנוסטלגי "הנסיך הפרסי". לפי קובץ ה-Readme, מדובר בפרויקט קוד פתוח בשם [SDLPoP](#) שמנוהל ב-Github. הקובץ הכיל מה שנראה כמו snapshot של ה-repository, כולל קבצי המקור ושאר הקבצים הנלווים.



הצעד המתבקש הבא הוא להוריד עצמאית snapshot של ה-repository, ולהשוות את שתי התיקיות, על מנת לבדוק אם הוכנסו שינויים בגרסה שקיבלנו באתגר. ואכן, אפשר למצוא שינויים בקבצים הבאים:

DLPoP-master				SDLPoP-master-cp			
Name	Size	Modified		Name	Size	Modified	
src	1,092,721	18/05/2019 20:05:43		src	1,095,798	13/07/2019 20:41:56	
common.h	1,710	18/05/2019 20:05:43	✗	common.h	1,725	18/05/2019 10:05:00	
data.h	19,008	18/05/2019 20:05:43	✗	data.h	19,349	18/05/2019 10:05:00	
proto.h	30,060	18/05/2019 20:05:43	✗	proto.h	30,389	18/05/2019 10:05:00	
seg000.c	61,612	18/05/2019 20:05:43	✗	seg000.c	61,977	18/05/2019 10:05:00	
seg001.c	22,307	18/05/2019 20:05:43	✗	seg001.c	22,404	18/05/2019 10:05:00	
seg002.c	31,475	18/05/2019 20:05:43	✗	seg002.c	31,693	18/05/2019 10:05:00	
seg005.c	29,668	18/05/2019 20:05:43	✗	seg005.c	29,834	18/05/2019 10:05:00	
seg006.c	52,683	18/05/2019 20:05:43	✗	seg006.c	52,746	18/05/2019 10:05:00	
seg007.c	30,226	18/05/2019 20:05:43	✗	seg007.c	30,781	18/05/2019 10:05:00	
.editorconfig	65	18/05/2019 20:05:43					
.gitignore	848	18/05/2019 20:05:43					



השינוי המשמעותי ביותר נמצא בקובץ seg000.c, בפונקציה check_the_end:

```
// seg000:0FB0
void __pascal far check_the_end() {
    if (next_room != 0 && next_room != drawn_room) {
        drawn_room = next_room;
        load_room_links();
        if (current_level == /*14*/ custom->win_level && drawn_room == /*5*/ custom->win_room) {
            #ifdef USE_REPLAY
                if (recording) stop_recording();
                if (replaying) end_replay();
            #endif
            // Special event: end of game

            end_sequence();

            different_room = 1;
            loadkid();
            anim_tile_modif();
            start_chompers();
            check_fall_flo();
            check_shadow();
        }
    }
}
```

```
// seg000:0FB0
void __pascal far check_the_end() {
    char strcc[100] = { 0 };
    char arr[8] = { 0 };
    char finalstr[100] = { 0 };
    char str[57] = { 0 };

    if (next_room != 0 && next_room != drawn_room) {
        drawn_room = next_room;
        load_room_links();
        if (current_level == custom->win_level &&
            drawn_room == custom->win_room) {

            // Special event: end of game
            rooms[current_level - 1] = 20;

            for (int i = 0, j = 0; i < (current_level + 1) * 4; i++) {
                if (i % 4 != 3) {
                    str[j*3 + i % 4] = block1[i-j];
                }
                else {
                    j++;
                }
            }

            RC(rooms, str, (unsigned char*)strcc);
            decode(pre, arr, 5);
            sprintf(finalstr, "%s %s", arr, strcc);
            printf("%s\n", finalstr);
            show_dialog(finalstr);
            redraw_screen(0);
            end_sequence();

            different_room = 1;
            loadkid();
            anim_tile_modif();
            start_chompers();
            check_fall_flo();
            check_shadow();
        }
    }
}
```

מעבר להוספה של משתנים מקומיים והסרה של הערות, אנחנו רואים כאן הוספה של לוגיקה שאמורה לרוץ בסוף המשחק, במידה והשחקן ניצח: פענוח של מספר מחרוזות והדפסתן למסך.

המימוש של RC - הפונקציה שאחראית על פענוח הדגל - פחות חשוב עכשיו (נראה אותו בסוף). מה שחשוב בעיקר הוא הקלט ל-RC:

```
RC(rooms, str, (unsigned char*)strcc);
```

הפונקציה מקבלת כקלט מערך בשם rooms יחד עם מערך בשם str. המערך str נבנה כמה שורות קודם בהסתמך על מערך בשם block1, שהוא מערך שנוסף על ידי יוצרי האתגר בקובץ data.h ואותחל במקום, ולכן ניתן להחשיב אותו בתור קבוע ולהתעלם ממנו לעת עתה. מה שיותר מעניין הוא המערך rooms:

```
extern char rooms[15];
```

הוא מאוכלס בזמן ריצה.

האכלוס העיקרי שלו הוא בסוף כל שלב:

```
case SEQ_END_LEVEL: // end level
    ++next_level;
    if (enable copyprot) rooms[current_level - 1] = curr_room;
```

ההשמה אל המערך תבוצע רק במידה והמשחק הוגדר לכלול את ה-Copy Protection שלו (על מנת להילחם בתופעת הפיראטיות, משחקים ישנים מסוימים הגיעו בזמנו עם סט שאלות שהפנה לחוברת ההדרכה שחולקה עם המשחק, ורק מי שהחזיק בה היה מסוגל לענות על השאלות. במקרה של "הנסיך הפרסי", השאלות הופיעו בתור שלב נפרד במשחק. בחידוש, ניתן לבחור האם להציג שלב זה או לא). במקרה כזה, המערך מאוכלס במספר החדר הנוכחי (כלומר זה שהשחקן נמצא בו כשהוא מסיים את השלב, שהרי כל שלב מתפרס על פני מספר חדרים).



בשני מקרים, קיימת לוגיקה מיוחדת לאכלוס תאים מסוימים במערך. בשלב 12:

```
} else if(custom->tbl_seamless_exit[current_level] >= 0) {  
    // Special event: level 12 running exit  
    if (Kid.room == custom->tbl_seamless_exit[current_level]) {  
        rooms[current_level - 1] = custom-  
>tbl_seamless_exit[current_level];  
        ++next_level;
```

ההשמה נעשית בהתבסס על מערך אחר, קבוע מראש (שהוגדר ב-data.h):

```
.tbl_seamless_exit = {-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  
23, -1, -1, -1},
```

ובשלב האחרון, כמו שראינו כבר:

```
// Special event: end of game  
rooms[current_level - 1] = 20;
```

אם כך, המטרה היא לאכלס את שאר התאים במערך, כלומר, להבין באיזה חדר כל שלב מסתיים. ב-"נסיך הפרסי" השלבים מיוצגים על ידי קבצים בינאריים:

```
root@kali:/media/sf_CTFs/checkpoint/Prince_Of_Persia/SDLPoP-master-cp/data/LEVELS# ls  
res2000.bin res2003.bin res2006.bin res2009.bin res2012.bin res2015.bin  
res2001.bin res2004.bin res2007.bin res2010.bin res2013.bin  
res2002.bin res2005.bin res2008.bin res2011.bin res2014.bin
```

כל קובץ כזה מקודד את השלב במלואו - הקירות, הדלתות, האויבים, הכניסה, היציאה וכו'. אבל - ברסינג של הפורמט ייקח זמן. אפשר לחלופין לדבג את המשחק, אבל גם זה ייקח זמן. אנחנו עוסקים במשחק של שנות התשעים, ולכן נעשה מה שהיה נהוג לעשות אז - נרמה.

בשנות התשעים לא היה משחק שכיבד את עצמו שלא כלל צ'יטים - רצף מקשים שאפשר בקלות להגיע ליכולת כלשהי כמו חיים נוספים, ציוד נוסף, יכולת פיזית נוספת וכד' (בוחן פתע: מה זה IDDDQD?). גם "הנסיך הפרסי" כלל צ'יטים, כל מה שהיה צריך לעשות כדי להפעיל אותם זה להריץ את המשחק יחד עם הפרמטר megahit בשורת הפקודה. לאחר מכן, היה אפשר להוסיף חיים, להרוג את כל האויבים, להאריך את מגבלת הזמן וכמובן לקפוץ שלבים. הרימייק שלנו כלל את הצ'יטים הללו והוסיף גם כמה משלו. את הרשימה המלאה אפשר למצוא בקובץ ה-readme:

```
Cheats:  
* Shift-L: go to next level  
* c: show numbers of current and adjacent rooms  
* Shift-C: show numbers of diagonally adjacent rooms  
* -: less remaining time  
* +: more remaining time  
* r: resurrect kid  
* k: kill guard  
* Shift-I: flip screen upside-down  
* Shift-W: slow falling  
* h: look at room to the left  
* j: look at room to the right
```

- * *u*: look at room above
 - * *n*: look at room below
 - * *Shift-B*: toggle hiding of non-animated objects
 - * *Shift-S*: Restore lost hit-point. (Like a small red potion.)
 - * *Shift-T*: Give more hit-points. (Like a big red potion.)
 - * *Shift+F12*: Save a screenshot of the whole level to the screenshots folder, thus creating a level map.
 - * *Ctrl+Shift+F12*: Save a screenshot of the whole level with extras to the screenshots folder.
- You can find the meaning of each symbol in Map_Symbols.txt.*

ראו איזה פלא - באמצעות SHIFT+L אנחנו יכולים לקפוץ בין שלבים ובאמצעות CTRL+SHIFT+F12 אנחנו יכולים לקבל מפה מפורטת של השלב כולו.

כך, לדוגמא, נראה השלב הראשון:

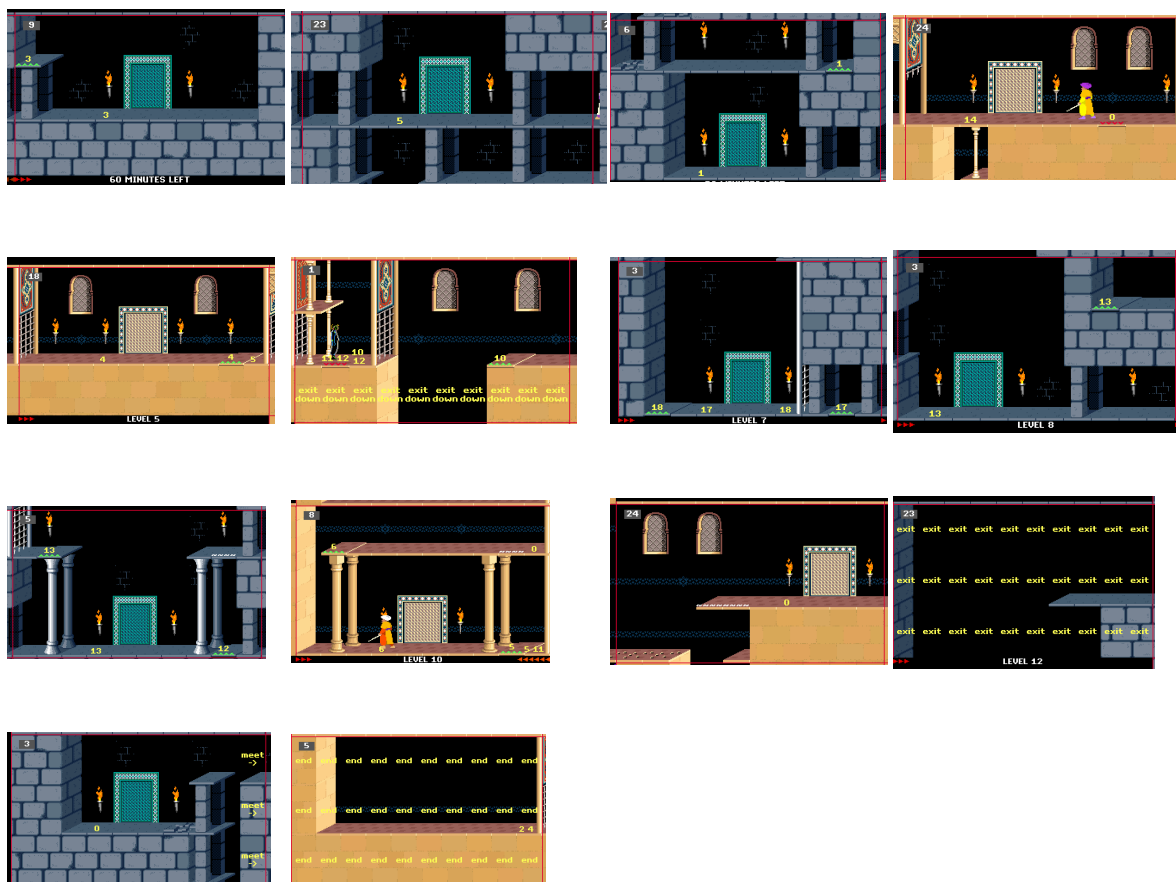


אם נתמקד בחדר שכולל את דלת היציאה, נראה שהוא ממוספר:



זהו חדר מספר 9.

נמשיך כן לפי השלבים השונים, ובסך הכל נקבל:



שימו לב שבמספר שלבים היציאה היא לא מהדלת אלא מחדר שעליו כתוב exit.

בתור מערך, נקבל:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Level	1	2	3	4	5	6	7	8	9	10	11	12	13	14	?
Value	9	23	6	24	18	1	3	3	5	8	24	23	3	5	?

נתבונן בתוצאה שקיבלנו:

1. ב"נסיך הפרסי" יש 14 שלבים, מדוע יש 15 תאים במערך?
2. שלב 12 אכן נראה כמו שלב מיוחד שבו היציאה אינה דרך דלת, והערך 23 שקיבלנו דרך תמונת המסך מסתדר עם הערך שראינו שמתייחס לחדר זה באופן מיוחד
3. עבור שלב 15, החדר המתקבל מהתמונה הינו 5, אך ישנו תנאי מיוחד שדורס את הערך ושם בו 20



לגבי השאלה הראשונה, עיון בקוד מגלה ששלב 15 הוא שלב ה-Copy Protection:

```
#ifdef USE_COPYPROT
    if (enable_copyprot && level_number == custom->copyprot_level) {
        level_number = 15;
    }
#endif
```

ולכן החדר המתאים הוא חדר 3:



אם כך, השלמנו את המערכ, וכעת נותר לאסוף את החלקים השונים של הפתרון שפוזרו לאורך הקבצים השונים: למשל, פונקציית RC, המערכ block1, פונקציית decode וכו'.

בסך הכל, נקבל את הקוד הבא:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define N1 256

char rooms[15] = { 9, 23, 6, 24, 18, 1, 3, 3, 5, 8, 24, 23, 3, 20, 3 };
const char block1[] = {214, 85, 173, 9, 13, 217, 126, 133, 241, 98, 37,
11, 50, 52, 8, 18, 230, 22, 122, 125, 160, 86, 8, 226, 17, 235, 234, 154, 238,
250, 210, 123, 171, 178, 43, 98, 237, 136, 68, 184, 17, 74, 113, 74, 138};
char pre[] = {0x26, 0x10, 0x06, 0x04, 0x12, 0x5d};

void swap(unsigned char *a, unsigned char *b) {
    unsigned char tmp = *a;
    *a = *b;
    *b = tmp;
}

int KSA(char *key, unsigned char *S) {
    int len = 15;
    int j = 0;
    for (int i = 0; i < N1; i++)
        S[i] = i;
    for (int i = 0; i < N1; i++) {
        j = (j + S[i] + key[i % len]) % N1;
        swap(&S[i], &S[j]);
    }
    return 0;
}
```



```
int PRGA(unsigned char *S, char *plaintext, unsigned char *ciphertext) {
    int i = 0;
    int j = 0;
    for (unsigned int n = 0, len = strlen(plaintext); n < len; n++) {
        i = (i + 1) % N1;
        j = (j + S[i]) % N1;
        swap(&S[i], &S[j]);
        int rnd = S[(S[i] + S[j]) % N1];
        ciphertext[n] = rnd ^ plaintext[n];
    }
    return 0;
}

int RC(char *key, char *plaintext, unsigned char *ciphertext) {
    unsigned char S[N1];
    KSA(key, S);

    PRGA(S, plaintext, ciphertext);

    return 0;
}

void decode(char* arr, char* res, int len) {
    res[0] = arr[0] ^ 0x61;
    res[1] = arr[1] ^ 0x62;
    res[2] = arr[2] ^ 0x63;
    res[3] = arr[3] ^ 0x65;
    res[4] = arr[4] ^ 0x66;
    res[5] = arr[5] ^ 0x67;
}

void check_the_end() {
    char strrc[100] = { 0 };
    char arr[8] = { 0 };
    char finalstr[100] = { 0 };
    char str[57] = {0};

    int current_level = 14;

    for (int i = 0, j = 0; i < (current_level + 1) * 4; i++) {
        if (i % 4 != 3) {
            str[j*3 + i % 4] = block1[i-j];
        }
        else {
            j++;
        }
    }

    RC(rooms, str, (unsigned char*)strrc);
    decode(pre, arr, 5);
    sprintf(finalstr, "%s %s", arr, strrc);
    printf("%s\n", finalstr);
}

void main(int argc, char** argv)
{
    check_the_end();
}
```



נריץ ונקבל:

```
root@kali:/media/sf_CTFs/checkpoint/Prince_Of_Persia# gcc solve.c -o solve && ./solve
Great: 0U0,3WpMΞQ<2,ER0F
@94
```

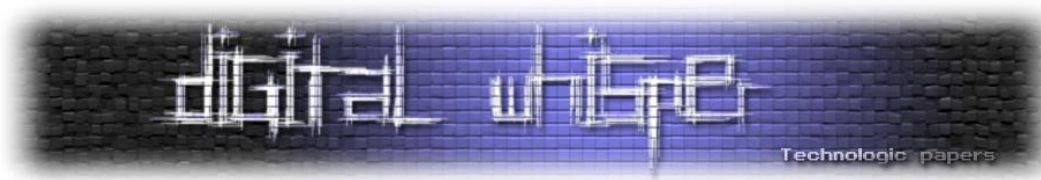
זה לא נראה כל כך טוב. אם להיות אופטימיים, אולי הייתה לנו שגיאה בודדת באחד החדרים. אפשר להריץ brute force על כל חדר בנפרד ולקוות לטוב. לשם כך נוסיף תוספת קטנה ל-main:

```
#define MAX_ROOM 30
#define FLAG_PREFIX "CSA"
void main(int argc, char** argv)
{
    int i, j;
    int limit = sizeof(rooms) / sizeof(rooms[0]);
    char strrc[100] = { 0 };
    int original_value;
    for (i = 0; i < limit; ++i)
    {
        original_value = rooms[i];
        for (j = 0; j < MAX_ROOM; ++j)
        {
            memset(strrc, 0, sizeof(strrc));
            rooms[i] = j;
            check_the_end(strrc); // Modify to accept parameter, remove
printf()
            if (memcmp(FLAG_PREFIX, strrc, sizeof(FLAG_PREFIX) - 1) ==
0)
            {
                printf("Room: %d, Value: %d, flag: %s\n", i + 1, j,
strrc);
                return;
            }
        }
        rooms[i] = original_value;
    }
}
```

התוצאה:

```
root@kali:/media/sf_CTFs/checkpoint/Prince_Of_Persia# gcc solve.c -o solve && ./solve
Room: 6, Value: 22, flag: CSA{<3_4nd_Th3y_L1VeD_HapPiLy_Ev3r_Af7eR_<3}
```

לפי המפה, בשלב 6 אין חדר 22, אבל עם הצלחה לא מתווכחים.



אתגר #8: MBA (קטגוריה: רברסינג. ניקוד: 120)

Our company had a spy up in corporate, apparently he's also some kind of math wiz. Who would have thought? He has an MBA!

These are the only files we managed to retrieve, can you reveal his secret to us?

לאתגר צורף ארכיון zip עם הקבצים הבאים:

```
root@kali:/media/sf_CTFs/checkpoint/MBA/fs# ls
bin_0aced962069e40051b6a293d163cc4b9c3d79d96 message_0aced962069e40051b6a293d163cc4b9c3d79d96
bin_15ef660b9d018a2b53be167fd97ba3ee718bfe53 message_15ef660b9d018a2b53be167fd97ba3ee718bfe53
bin_1db0c41b8af3255b06d6363e0ada55345d44ea19 message_1db0c41b8af3255b06d6363e0ada55345d44ea19
bin_2a0d1ad918f817fb386354bbbf3a9666fd7fa0b3 message_2a0d1ad918f817fb386354bbbf3a9666fd7fa0b3
bin_2bd2a2435be46f2ffb27ef18ebb3b1eef3f555ad message_2bd2a2435be46f2ffb27ef18ebb3b1eef3f555ad
bin_3a92fd21f105d56d7125bbb8b92e06f7636216cd message_3a92fd21f105d56d7125bbb8b92e06f7636216cd
bin_407ae8c2f9f70b68871fe259953533e05a1089bd message_407ae8c2f9f70b68871fe259953533e05a1089bd
bin_4d4dda776e7046f6f136fa50e7768583cabab58a message_4d4dda776e7046f6f136fa50e7768583cabab58a
bin_4e5000fe5c19840aa37fa3a876f7007c9419222f message_4e5000fe5c19840aa37fa3a876f7007c9419222f
bin_51fd182fb872f966a363225911b703bc35ebc5c9 message_51fd182fb872f966a363225911b703bc35ebc5c9
bin_5b7594ce1596d1175b46d96d217cd3a9bf3ded2f message_5b7594ce1596d1175b46d96d217cd3a9bf3ded2f
bin_6b93159d0fe72ebca36c5be17e5fbed6b520e5a message_6b93159d0fe72ebca36c5be17e5fbed6b520e5a
bin_758dfb510bc1b7e3adb5c9db9f1cca92182864eb8 message_758dfb510bc1b7e3adb5c9db9f1cca92182864eb8
bin_775a014d52d36dcf4722d9ce04adf7f191d158c message_775a014d52d36dcf4722d9ce04adf7f191d158c
bin_7ffaafacd7a9d4e13c6567010f1d179e10a8f121 message_7ffaafacd7a9d4e13c6567010f1d179e10a8f121
bin_93c7ba8dad617f29d492d001268787761e45acda message_93c7ba8dad617f29d492d001268787761e45acda
bin_9572f6771e168d90bcb519a8e9d71f6b5a5279a2 message_9572f6771e168d90bcb519a8e9d71f6b5a5279a2
bin_9a77ac1f2f854f64c7daf66905e4d52d4407fb6e message_9a77ac1f2f854f64c7daf66905e4d52d4407fb6e
bin_9ee481edb32a39158c5fe7275891a9a9f067fa75 message_9ee481edb32a39158c5fe7275891a9a9f067fa75
bin_a241d27534c7f21160c2c1eaf9f3a9d60de34961 message_a241d27534c7f21160c2c1eaf9f3a9d60de34961
bin_a2de858730acf04a751cd7441104f6593bd35061 message_a2de858730acf04a751cd7441104f6593bd35061
bin_b2eb960e38905a5040e503bfd36b6c8a4f143e2d message_b2eb960e38905a5040e503bfd36b6c8a4f143e2d
bin_c469bfd5f0be21f827d8ab73d91e62f387f99950 message_c469bfd5f0be21f827d8ab73d91e62f387f99950
bin_f524a0f22c6568420ad0d409e782d9f0f8fd3e78 message_f524a0f22c6568420ad0d409e782d9f0f8fd3e78
bin_f70ddd95e75211a0a35fd52350edc66548c47068 message_f70ddd95e75211a0a35fd52350edc66548c47068
bin_secret
```

יש לנו צמידים של קבצים: על כל קובץ bin יש לנו קובץ message (מלבד bin_secret, שמכיל את הדגל).

דוגמא לקובץ בינארי:

```
root@kali:/media/sf_CTFs/checkpoint/MBA/fs# xxd -g 1 bin_7ffaafacd7a9d4e13c6567010f1d179e10a8f121
00000000: 01 fe 01 fe 00 ff 00 8a 7b 00 00 00 00 00 00 00 00 .....{.....
00000010: 02 ff 02 fe 00 fd 01 8a ff 12 00 00 00 00 00 00 00 .....
00000020: 02 fe 02 fd 01 fd 00 8a ef b3 07 00 00 00 00 00 00 .....
00000030: 01 fe 03 ff 00 fd 00 8a 59 0b 00 00 00 00 00 00 .....Y.....
00000040: 01 fe 03 fe 01 fe 01 8a 5f 00 00 00 00 00 00 00 ....._.....
00000050: 02 ff 03 fd 02 fd 00 8a ac 87 07 00 00 00 00 00 00 .....
00000060: 00 fe 01 fd 01 fe 03 8a f7 1e 00 00 00 00 00 00 00 .....
00000070: 03 fd 02 fd 02 fd 02 8a 35 43 5a 06 00 00 00 00 00 .....5CZ.....
00000080: 01 fd 03 fe 00 fd 01 8a 7f 1e 00 00 00 00 00 00 00 .....
00000090: 00 ff 01 ff 02 fd 00 8a b5 16 00 00 00 00 00 00 00 .....
000000a0: 01 fd 01 fd 01 fe 02 8a eb c1 0a 00 00 00 00 00 00 .....
000000b0: 00 fd 00 fe 03 ff 01 8a f9 0a 00 00 00 00 00 00 00 .....
000000c0: 01 fd 03 ff 00 ff 02 8a df 1e 00 00 00 00 00 00 00 .....
000000d0: 2e 2e 2e 2e 2e 2e 2e 01 03 00 02 .....

```

דוגמא לקובץ ההודעה המתאים:

```
root@kali:/media/sf_CTFs/checkpoint/MBA/fs# xxd -g 1 message_7ffaafacd7a9d4e13c6567010f1d179e10a8f121
00000000: 59 57 35 6b
YWsk
```



דוגמא לקובץ בינארי נוסף:

```
root@kali:/media/sf_CTFs/checkpoint/MBA/fs# xxd -g 1 bin_0aced962069e40051b6a293d163cc4b9c3d79d96
00000000: 03 fd 02 fe 01 fe 02 8a 5b 00 00 00 00 00 00 00 00 .....[.....
00000010: 01 fe 00 fd 03 ff 03 8a f8 00 00 00 00 00 00 00 00 .....D.....
00000020: 00 ff 01 ff 03 fd 03 8a 44 01 00 00 00 00 00 00 00 .....4.....
00000030: 03 fd 00 fd 03 fd 01 8a 34 00 00 00 00 00 00 00 00 .....{.....
00000040: 01 ff 03 fd 02 ff 01 8a 13 00 00 00 00 00 00 00 00 .....
00000050: 02 ff 00 ff 03 ff 00 8a 7b 01 00 00 00 00 00 00 00 .....
00000060: 2e 2e 2e 2e 2e 2e 02 01 03 00 .....

```

דוגמא לקובץ ההודעה המתאים:

```
root@kali:/media/sf_CTFs/checkpoint/MBA/fs# xxd -g 1 message_0aced962069e40051b6a293d163cc4b9c3d79d96
00000000: 59 58 4a 6c YXJl

```

אין הרבה מה לעשות עם זה מלבד לנסות לחפש מאפיינים דומים לאורך כל הקבצים.

מאפיינים שניתן למצוא מעיון בכלל הקבצים:

1. באמצעות שינוי רוחב ההצגה של הקובץ, ניתן להגיע למצב שבו ישנה עמודה רציפה של 0x8a מהשורה הראשונה ועד השורה הלפני-אחרונה.
2. הבתים משמאל ל-0x8a מתאפיינים בערכים קרובים ל-0 ואז קרובים ל-0xff לסירוגין.
3. הבתים שמימין ל-0x8a מציגים שונות יחסית גבוהה, ולרוב הם מרופדים באפסים עד סוף השורה. לעיתים במקום אפסים הם מרופדים ב-0xff עד סוף השורה.
4. לקראת סוף הקובץ, תמיד ניתן למצוא רצף של שבעה ערכי 0x2e, ואחריהם X בתים עם ערכים קרובים ל-0. אורך ההודעה המפוענחת הוא תמיד אותו ה-X.

מספר קבצים נותנים לנו תובנה נוספת. ניקח לדוגמא את הקובץ הבא:

```
root@kali:/media/sf_CTFs/checkpoint/MBA/fs# xxd -g 1 message_9a77ac1f2f854f64c7daf66905e4d52d4407fb6e
00000000: 62 57 39 79 5a 51 3d 3d bw9yZQ==

```

מעבר לעובדה שההודעה מקודדת ב-base64 (נתון לא מעניין בשלב הזה), יש לנו שני תווים שחוזרים על עצמם בסוף ההודעה. הקובץ הבינארי המתאים נראה כך בסופו:

```
root@kali:/media/sf_CTFs/checkpoint/MBA/fs# xxd -g 1 -c 18 bin_9a77ac1f2f854f64c7daf66905e4d52d4407fb6e | tail -n 4
0000069c: 05 ff 02 ff 06 ff 00 ff 04 8a 00 00 00 00 00 00 .....
000006ae: 00 fd 05 fd 03 fd 05 fd 00 8a c9 01 00 00 00 00 .....
000006c0: 07 ff 05 fd 01 ff 06 fe 02 8a 00 00 00 00 00 00 .....
000006d2: 2e 2e 2e 2e 2e 2e 05 02 07 04 00 03 01 01 .....

```

אפשר לראות ששני הבתים האחרונים זהים, בדיוק כמו ב-text.plaintext. זוהי עדות חזקה לכך שמדובר פה בצופן החלפה כלשהו. ואם ניקח את התיאוריה הזו צעד נוסף קדימה, נסיק מהעובדה שכל הערכים לפני ההחלפה קרובים ל-0 שמדובר במעין "אינדקסים" למשהו בסגנון "טבלת החלפה" שכנראה נבנית מוקדם יותר.

ואכן, בבלוק העיקרי של הבתים אנחנו רואים שימוש גדול ב"אינדקסים" הללו. מכאן צריך קפיצה לוגית קטנה, מגובה בעובדה שהגיבור שלנו מתואר בתור אשף מתמטי.



כדי להגיע למסקנה שמדובר פה במערכת משוואות:

- כל שורה היא משוואה
- כל "אינדקס" הוא משתנה
- כל ערך בין שני משתנים (הבתיים שערכם קרוב ל-0xff) הוא אופרטור
- המשמעות של 0x8a היא "שווה"
- מימין ל-0x8a שוכנת התוצאה
- רצף ה-0x2e מסיים את מערכת המשוואות
- לאחר הרצף, מגיע מפתח הפענוח

מכיוון שעבור כל הדוגמאות כבר יש לנו קובץ עם התשובה, קל יחסית לאמת את התיאוריה הזו. פשוט צריך להציב את התשובה במערכת המשוואות ולנסות להנדס לאחור את האופרטורים. אם נצליח להגיע למערכת משוואות נכונה - אימתנו את התיאוריה.

נעבוד עם מערכת המשוואות הנוחה ביותר שקיימת בדוגמאות:

```
root@kali:/media/sf_CTFs/checkpoint/MBA/fs# xxd -g 1 -c 14 bin_f524a0f22c6568420ad0d409e782d9f0f8fd3e78
00000000: 00 fe 01 fd 00 8a e0 1a 00 00 00 00 00 00 00 .....
0000000e: 03 fd 01 fe 00 8a 38 e5 ff ff ff ff ff ff .....8.....
0000001c: 01 ff 02 ff 03 8a 00 00 00 00 00 00 00 00 .....&.....
0000002a: 03 ff 01 fe 00 8a 26 00 00 00 00 00 00 00 .....&.....
00000038: 00 fd 01 ff 02 8a 00 00 00 00 00 00 00 00 .....
00000046: 02 fd 02 ff 02 8a 00 00 00 00 00 00 00 00 .....
00000054: 02 fd 01 ff 02 8a 10 00 00 00 00 00 00 00 .....
00000062: 01 ff 01 fe 02 8a 10 00 00 00 00 00 00 00 .....
00000070: 02 fe 01 ff 02 8a 30 00 00 00 00 00 00 00 .....0.....
0000007e: 03 ff 03 fe 03 8a 44 00 00 00 00 00 00 00 .....D.....
0000008c: 01 ff 00 ff 02 8a 10 00 00 00 00 00 00 00 .....
0000009a: 02 ff 02 ff 02 8a 30 00 00 00 00 00 00 00 .....0.....
000000a8: 03 ff 02 fd 01 8a 4e 00 00 00 00 00 00 00 .....N.....
000000b6: 03 fe 00 fe 01 8a 34 b1 0b 00 00 00 00 00 .....4.....
000000c4: 03 ff 01 fd 01 8a 00 00 00 00 00 00 00 00 .....
000000d2: 2e 2e 2e 2e 2e 2e 01 03 00 02 .....
root@kali:/media/sf_CTFs/checkpoint/MBA/fs# xxd -g 1 message_f524a0f22c6568420ad0d409e782d9f0f8fd3e78
00000000: 51 6e 56 30 QnV0
```

הנוחות של מערכת המשוואות הזו נובעת מהעובדה שקיימים בה בסך הכל ארבעה משתנים, וכל משוואה כוללת שלושה משתנים בלבד. במערכות אחרות המשוואות היו ארוכות יותר וכך היה קשה יותר לנחש מהם האופרטורים.

מתוך מערכת המשוואות הזו, נבחר כמה משוואות שנראות קלות לפתרון. נשתמש בסימנים Δ , \blacksquare , \emptyset על מנת לייצג את הפעולות השונות. כמו כן, נניח שהבתיים מימין ל-0x8a מתפרשים בתור int64_t ב-little endian, כי זה מסתדר עם הריפוד באפסים או ב-0xff לכיוון ימין.

בשורה 0xb6 יש לנו משוואה שכוללת רק סימן אחד:

$$x_3 \Delta x_0 \Delta x_1 = 0xbb134$$

אם נצליב את סדר המשתנים (שמופיע אחרי ה-0x2e) עם התשובה, נקבל ש:

$$x_0 = 0x56, x_1 = 0x51, x_2 = 0x30, x_3 = 0x6e$$



נציב ונקבל:

$$0x6e \Delta 0x56 \Delta 0x51 = 0xbb134$$

ננסה פעולות שונות עד שנגיע למסקנה שפעולת הכפל מתאימה.

נעבור למשוואה בשורה 0xe שכבר כוללת סימן אחד שפיצחנו:

$$x_3 \blacksquare x_1 \Delta x_0 = -6856 = 0x6e \blacksquare 0x51 \times 0x56$$

ננסה פעולות שונות עד שנגיע למסקנה שפעולה החיסור מתאימה.

נעבור למשוואה בשורה 0x9a שכוללת רק פעולה אחת ומשתנה אחד:

$$x_2 \emptyset x_2 \emptyset x_2 = 0x30 = 0x30 \emptyset 0x30 \emptyset 0x30$$

במבט ראשון, זאת לא משוואה קלה לפתרון. אך במהרה מתגלה שפעולת & (bitwise and) מתאימה במדויק.

מצאנו את שלוש הפעולות (כמובן שמומלץ לוודא נכונות מול משוואות אחרות בקובץ הנוכחי).

השלב המתבקש הבא הוא בדיקת נכונות מול קבצים אחרים, אך למרבה הצער נראה שהפעולות שמצאנו לא מתאימות. נראה שכל קובץ מפרש את סט הפעולות בצורה אחרת.

מכיוון שאיננו מעוניינים לעבור את התהליך המפרך הזה פעם נוספת, נכתוב סקריפט שיבצע brute force על הפעולות במקומנו.

הסקריפט בוחר שלוש פעולות מתוך הקבוצה "^(\"|\", \"+\", \"&\", \"-\", \"*\")\", ואז משתמש במנוע Z3 על מנת למצוא פתרון למערכת המשוואות.

```
from itertools import permutations
from z3 import *

import os
import mmap
import glob
import struct

def memory_map(filename, access=mmap.ACCESS_READ):
    size = os.path.getsize(filename)
    fd = os.open(filename, os.O_RDONLY)
    return mmap.mmap(fd, size, access=access)

class MBASolver(object):

    EQUAL = '\x8A'
    SEPARATOR = bytes('\x2E'.encode("ascii") * 7)
    RESULT_LENGTH = 8

    def __init__(self, path):
        self.file = memory_map(path)
        self.variable_section_length = self.get_variable_section_length()
        self.separator_index = self.get_separator_index()
```



```
self.single_equation_length = self.variable_section_length +
len(self.EQUAL) + self.RESULT_LENGTH

def __del__(self):
    self.file.close()

def get_variable_section_length(self):
    for i in range(len(self.file)):
        if chr(self.file[i]) == self.EQUAL:
            assert (i % 2 == 1)
            return i
    raise Exception("Can't find variable section length")

def get_separator_index(self):
    return self.file.find(self.SEPARATOR)

def handle_single_equation(self, equation, operators, variables, solver):
    operators[ord(self.EQUAL)] = ""
    str_equation = ""
    for i in range(0, self.variable_section_length, 2):
        variable = "x{}".format(equation[i])
        if variable not in variables:
            variables[variable] = BitVec(variable, 32)
            solver.add(variables[variable] >= ord('!'))
            solver.add(variables[variable] <= ord('~'))

        operator = operators[equation[i+1]]

        str_equation += "{} {}".format("variables['"+variable+"'",
operator)

        result = struct.unpack_from('<q',
            equation[self.variable_section_length + 1:])[0]
        str_equation += "== {}".format(result)

        solver.add(eval(str_equation))

def Solve(self, verbose = False):
    for perm in permutations(["*", "-", "&", "+", "|", "^"], 3):
        solver = Solver()
        variables = {}
        operators = {}
        x = 0xff
        for i in range(3):
            operators[x] = perm[i]
            x -= 1

        for i in range(0, self.separator_index,
self.single_equation_length):
            self.handle_single_equation(
                self.file[i:i+self.single_equation_length],
                operators, variables, solver)

        if solver.check() == sat:
            res = ""
            model = solver.model()
            if verbose:
                print(model)
                print(operators)
            for c in self.file[self.separator_index + len(self.SEPARATOR):]:
                res += chr(model[variables["x{}".format(c)]]).as_long()
            return res

def test():
    msg_prefix = "message_"
    bin_prefix = "bin_"
```



```
for path in glob.glob(f'fs/{msg_prefix}*'):
    print ("Testing {}".format(path))
    with open(path) as f:
        expected = f.read()
        actual = MBASolver(path.replace(msg_prefix, bin_prefix)).Solve()
        if expected != actual:
            print (f"\t Mismatch: Expected: {expected}, actual {actual}")
    print ("All tests done")

if __name__ == "__main__":
    print(MBASolver("fs/bin_secret").Solve(verbose = True))
```

הפתרון:

```
root@kali:/media/sf_CTFs/checkpoint/MBA# python3 solve.py
[x11 = 95,
 x5 = 110,
 x4 = 97,
 x3 = 125,
 x29 = 95,
 x31 = 101,
 x33 = 123,
 x35 = 95,
 x16 = 73,
 x1 = 76,
 x6 = 114,
 x10 = 77,
 x13 = 105,
 x14 = 67,
 x8 = 53,
 x28 = 105,
 x34 = 116,
 x25 = 61,
 x26 = 66,
 x23 = 65,
 x32 = 104,
 x15 = 52,
 x27 = 101,
 x19 = 77,
 x12 = 116,
 x30 = 77,
 x18 = 48,
 x21 = 55,
 x24 = 95,
 x2 = 66,
 x22 = 111,
 x9 = 83,
 x17 = 48,
 x20 = 67,
 x7 = 101,
 x0 = 48]
{255: '&', 254: '|', 253: '+', 138: ''}
CSA{Bo0Lean_7iMe5_4rIthMetiC=_B00M}
```



אתגר 9: Lost inside my PPTX - (קטגוריה: פיתוח. ניקוד: 30)

Oh my, I'm locked outside my car! I mean, my car is right here but... I've lost my key!

My car doesn't require a physical key, though. It's a digital one. Luckily, I've once written it in an odd way... Please help me find it! I'm running late for dinner O_o

לאתגר צורף קובץ הסבר:

Oh my, I'm locked outside my car! I mean, my car is right here but... I've lost my key!

My car doesn't require a physical key, though. It's a digital one. Luckily, I've once written it in an odd way...

Please help me find it! I'm running late for dinner O_o

In the attached file "real_key.rar" you'll find a lot of power point files. My key hides in there, as I'll explain below.

Luckily, I've also created a "small_key.rar", so we can easily play around with it. Extract this one first.

The first relevant powerpoint file is called "START.pptx". Open it.

As you'll see, it contains some slides. Every slide has text in the following format:

<CHARACTER>, <NEXT_PRESENTATION_NAME>, <SLIDE_NUMBER>

Each relevant presentation holds a single character from my key, in a single slide. Not all presentations are relevant, though.

For "START.pptx", this is guaranteed to be the first slide. As you can see, it displays the following text:

`L, E26DWA33X6.pptx, 10`

You can ignore the rest of the slides - as only one slide is relevant in every (relevant) presentation.

This means that the first character for my key is `L`. The next relevant character can be found in slide 10 of presentation E26DWA33X6.pptx.

Let's go there:

`5, PRZ12DA3D8.pptx, 1`

So our next character is `5`. The next relevant character is in the first slide of PRZ12DA3D8.pptx, let's go there:



`M, A6LLDC9D18.pptx, 12`

So our character is `M`. The next one will be in slide 12 of A6LLDC9D18.pptx. Let's go there...

Wait, it doesn't exist! This means we're done with the key, and it is:

`L5M`.

Note that all the other presentations file were irrelevant for solving this.

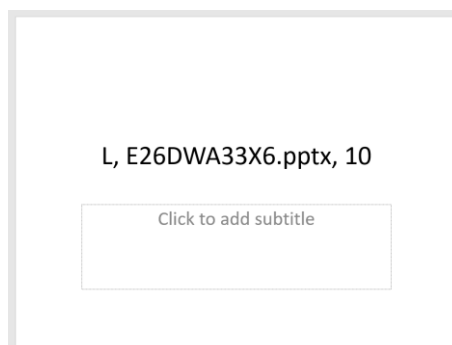
I could have done this without you, of course, but my real key is much longer. It's written in the presentations in "real_key.rar" in the same mechanism, though. So you know what you need to do.

My mom just called, dinner is getting cold. Please help me!

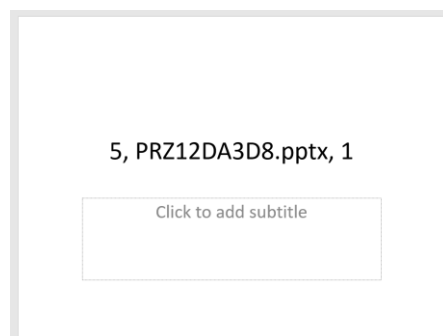
כמו כן, צורפו שני קבצי ארכיון: small_key.rar ו-real_key.rar.

```
root@kali:/media/sf_CTFs/checkpoint/Lost_inside_my_PPTX# ls small_key
1IBSY00ZVU.pptx  C33P4YFD71.pptx  E26DWA33X6.pptx  LY6D4PEZ2.pptx  PLD2PSRAXH.pptx  R958TF7J2.pptx  TZ03G719EP.pptx
9AP15SFDPQ.pptx  DY01M81EZV.pptx  HMUMDZYCSR.pptx  O0N5LT3D1V.pptx  PRZ12DA3D8.pptx  START.pptx
```

ב-real_key היו 1005 מצגות Power Point. נעקוב בקצרה אחרי תחילת הדוגמא. אם פותחים את start.pptx מקבלים את השקופית הבאה:



נמשיך למצגת אשר צויינה בטקסט ונקבל בשקופית 10 את:



וכן הלאה. בסך הכל היה מדובר במשימה יחסית פשוטה, במיוחד לאחר מציאת ספרייה שמסוגלת להתמודד עם קבצי Power Point.



הקוד:

```
from collections import namedtuple
import pptx
import re

TARGET_TEXT_REGEX = re.compile("(.),\s+([a-zA-Z0-9]+\..pptx),\s+(\d+)")
Result = namedtuple('Result', 'letter file_name slide_num')

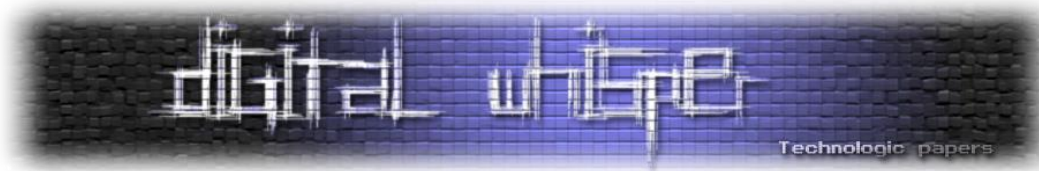
def get_text(prs, slide_num):
    shapes = prs.slides[slide_num - 1].shapes
    for shape in shapes:
        if not shape.has_text_frame:
            continue
        for paragraph in shape.text_frame.paragraphs:
            for run in paragraph.runs:
                match = TARGET_TEXT_REGEX.match(run.text)
                if match is not None:
                    return Result(match.group(1), match.group(2),
int(match.group(3)))
            return None

flag = ""
result = Result("", "START.pptx", 1)
while True:
    try:
        result =
get_text(pptx.Presentation("real_key/{}".format(result.file_name)),
result.slide_num)
        print result
        flag += result.letter
    except pptx.exc.PackageNotFoundError:
        break

print "Flag: {}".format(flag)
```

התוצאה:

```
root@kali:/media/sf_CTFs/checkpoint/Lost_inside_my_PPTX# python
solve.py
Result(letter=u'C', file_name=u'90VA981P16.pptx', slide_num=21)
Result(letter=u'S', file_name=u'1KKT508COY.pptx', slide_num=15)
Result(letter=u'A', file_name=u'NC7C0UG1M8.pptx', slide_num=19)
Result(letter=u'{', file_name=u'HLDQ0U6RGK.pptx', slide_num=3)
Result(letter=u'7', file_name=u'93T27YM76L.pptx', slide_num=21)
Result(letter=u'I', file_name=u'BTZTXSP4T0.pptx', slide_num=19)
Result(letter=u'Z', file_name=u'TZZJ1QI1DU.pptx', slide_num=11)
Result(letter=u'V', file_name=u'QVVT40B1U.pptx', slide_num=2)
Result(letter=u'Y', file_name=u'5Z4...31T...pptx', slide_num=1)
Result(letter=u'M', file_name=u'IT99D9EAXD.pptx', slide_num=25)
Result(letter=u'4', file_name=u'FWUWG7CVCA.pptx', slide_num=15)
Result(letter=u'V', file_name=u'MV17IX9P13.pptx', slide_num=15)
Result(letter=u'}', file_name=u'B7986MG33D.pptx', slide_num=14)
Flag: CSA{7IZUWIKWIEQZNS9F5TVVY7JR9ZAB3CR2L0VA01BCJ01W47IEHGNT1G
7QDHPJK5QVFH55JD45R4NF2TRF0VCMRGM3AFMDF2H8C YE1BMUQM50NFWB2WQNCYM
4V}
```



אתגר #10: Blockchain - (קטגוריה: פיתוח. ניקוד: 40)

Implement the attached blockchain specs and get the flag!

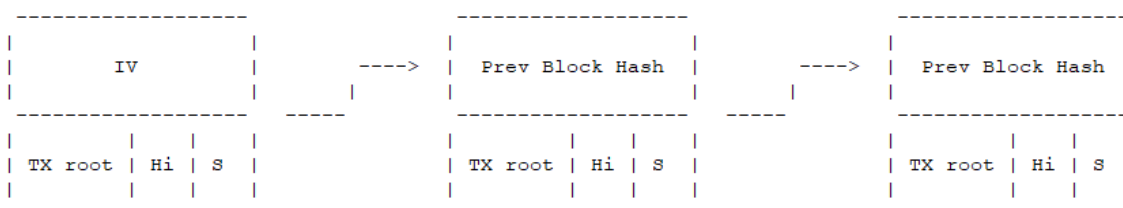
לאתגר צורף קובץ הוראות עם התוכן הבא:

The CSA blockchain is made out of blocks constructed as following:

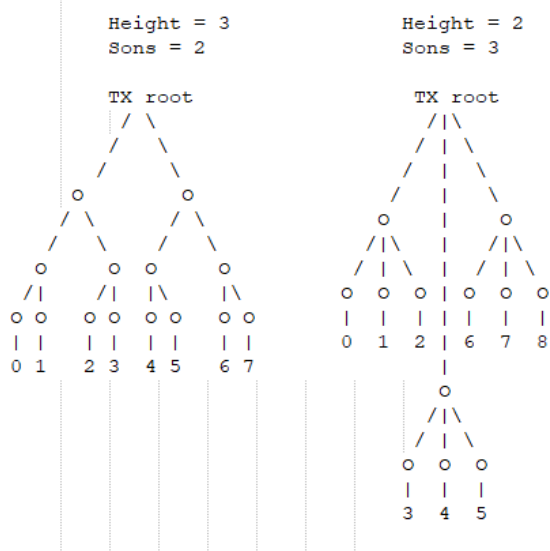
Each TX root is calculated by a balanced Merkle tree. The hash function used by the Merkle tree is md5. In each Merkle tree every non-leaf sons amount is fixed, but may change over different trees.

The first block hash is calculated by md5 of concatenation initialization vector with the first block TX root. The rest of the blocks hash is calculated by md5 of concatenation previous block hash with the current TX root.

In the attach zip are 16 blocks, your mission is to calculate the hash of the last block, given the IV a861f335d4d457a7c1d00640da380dc4.



Example balanced Merkle trees:





כמו כן, צורף קובץ blocks.7z עם מספר תיקיות:

```
root@kali:/media/sf_CTFs/checkpoint/Blockchain# ls blocks
block_0-height_4-sons_4  block_13-height_3-sons_4  block_2-height_2-sons_4  block_6-height_3-sons_3
block_10-height_3-sons_5  block_14-height_4-sons_2  block_3-height_3-sons_3  block_7-height_5-sons_2
block_11-height_4-sons_2  block_15-height_4-sons_3  block_4-height_2-sons_4  block_8-height_3-sons_5
block_12-height_2-sons_3  block_1-height_4-sons_3  block_5-height_5-sons_5  block_9-height_5-sons_4
```

כל תיקייה הכילה מספר קבצים, לדוגמא:

```
root@kali:/media/sf_CTFs/checkpoint/Blockchain# ls blocks/block_2-height_2-sons_4
tx_0 tx_1 tx_10 tx_11 tx_12 tx_13 tx_14 tx_15 tx_2 tx_3 tx_4 tx_5 tx_6 tx_7 tx_8 tx_9
```

וכל קובץ הכיל תוכן כלשהו:

```
root@kali:/media/sf_CTFs/checkpoint/Blockchain# cat blocks/block_2-height_2-sons_4/tx_0 && echo
DCBa24d5858f8BbF60Db1Eb3d8c40DFF3c      fc77CCDf0DAEFa10c282e2Ffe5b0c7E431      5.884893
```

המשימה דרשה לייצר [עץ מרקל](#):

עץ מרקל (Merkle tree) הוא עץ גיבוב בינארי שבו כל קודקוד מסומן בערך גיבוב של בניו (או ערכי העלים) והוא סוג של טבלת גיבוב בצורת רשימה היררכית. כלומר, קיים קשר בין ערכי כל הצמתים החל מהעלים ועד לשורש העץ.

אנחנו נממש עץ מרקל באמצעות שתי מחלקות: מחלקת קודקוד ומחלקת עלה. שתי המחלקות יידרשו לממש מתודה של get_hash (ועוד מתודות עזר-נוספת של get_height). לשם כך, נעזר באב משותף:

```
class MerkleObject(ABC):
    def __init__(self, hash_func):
        self.hash_func = hash_func

    def get_hash(self):
        raise NotImplementedError

    def get_height(self):
        raise NotImplementedError
```

ההגדרה של עלה היא יחסית פשוטה:

```
class MerkleLeaf(MerkleObject):
    def __init__(self, hash_func, data):
        super().__init__(hash_func)
        self.hash = hash_func(data).hexdigest()

    def get_hash(self):
        return self.hash

    def get_height(self):
        return 0
```

העלה הוא כמובן בגובה 0. הוא מקבל בקריאת האתחול שלו את פונקציית ה-hash (MD5 לפי דרישת התרגיל) ואת המידע שאותו הוא אמור לייצג, מחשב את ה-hash על המידע ושומר את התוצאה.



הגדרת הקודקוד לא הרבה יותר מסובכת:

```
class MerkleNode(MerkleObject):
    def __init__(self, hash_func, expected_num_children):
        super().__init__(hash_func)
        self.expected_num_children = expected_num_children
        self.children = []

    def add_child(self, child):
        if len(self.children) >= self.expected_num_children:
            raise ValueError("Error: Too many children added to current
node")
        if not isinstance(child, MerkleObject):
            raise ValueError("Error: Expecting a MerkleObject")
        if self.hash_func != child.hash_func:
            raise ValueError("Error: Hash function mismatch")

        self.children.append(child)

    def get_num_children(self):
        return len(self.children)

    def get_hash(self):
        if len(self.children) != self.expected_num_children:
            raise RuntimeError("Error: Missing children")

        res = ""
        for c in self.children:
            res += c.get_hash()
        return self.hash_func(res.encode("ascii")).hexdigest()

    def get_height(self):
        return self.children[0].get_height() + 1
```

מכיוון שמדובר בעץ שלם, על מנת לחשב את גובה הקודקוד ניתן לבחור את הגובה של כל אחד מבניו ולהוסיף אחד.

חישוב ה-hash הוא חיבור של כל ה-hash-ים של הבנים, וחישוב hash על התוצאה. שאר הלוגיקה קשורה בעיקר להוספת ילדים ובדיקת שגיאות. עלינו לקרוא את הקבצים מהדיסק ולבנות עצי מרקל מתאימים. נעשה זאת באופן הבא:

```
EXTRACT_HEIGHT_SONS_RE = re.compile(r"block_(\d+)-height_(\d+)-
sons_(\d+)$")

def create_tree(path, height, num_sons):
    queue = []

    file_id = 0
    num_parents = (num_sons ** height) // num_sons
    for i in range(num_parents):
        n = MerkleNode(hashlib.md5, num_sons)
        for j in range(num_sons):
            with open(os.path.join(path, "tx_{}".format(file_id)), "rb")
as f:
                n.add_child(MerkleLeaf(hashlib.md5, f.read()))
            file_id += 1
        queue.append(n)
```




```
while len(queue) != 1:
    num_parents = num_parents // num_sons
    for i in range(num_parents):
        n = MerkleNode(hashlib.md5, num_sons)
        for j in range(num_sons):
            n.add_child(queue.pop(0))
        queue.append(n)

assert(queue[0].get_height() == height)
return queue[0]

def build_tree(subdir):
    roots = {}
    for filename in glob.iglob(os.path.join(subdir, '*'),
recursive=False):
        if os.path.isdir(filename):
            match = EXTRACT_HEIGHT_SONS_RE.search(filename)
            if match is None:
                raise ValueError("Error: Unexpected path format:
{}".format(filename))
            block = int(match.group(1))
            height = int(match.group(2))
            num_sons = int(match.group(3))
            roots[block] = create_tree(filename, height, num_sons)
    return roots
```

שם התיקיה מכיל את מבנה העץ, למשל block_0-height_4-sons_4 או block_1-height_4-sons_3. נשתמש במידע זה על מנת לבנות עץ שלם מתאים מבחינת גובה ומספר בנים. הפונקציה create_tree מקבלת את הגובה ואת מספר הבנים, ומייצרת את העץ מלמטה למעלה בהתאם למספר הבנים הרצוי.

כל שנותר הוא לבצע את השרשור הנדרש, תחילה באמצעות ה-IV ואז באמצעות התוצאות הקודמות:

```
if __name__ == "__main__":
    roots = build_tree('blocks')

    iv = "a861f335d4d457a7c1d00640da380dc4"
    prev_hash = iv
    for i in range(len(roots)):
        current_data = prev_hash + roots[i].get_hash()
        prev_hash =
hashlib.md5(current_data.encode("ascii")).hexdigest()

    print (prev_hash)
```

התוצאה:

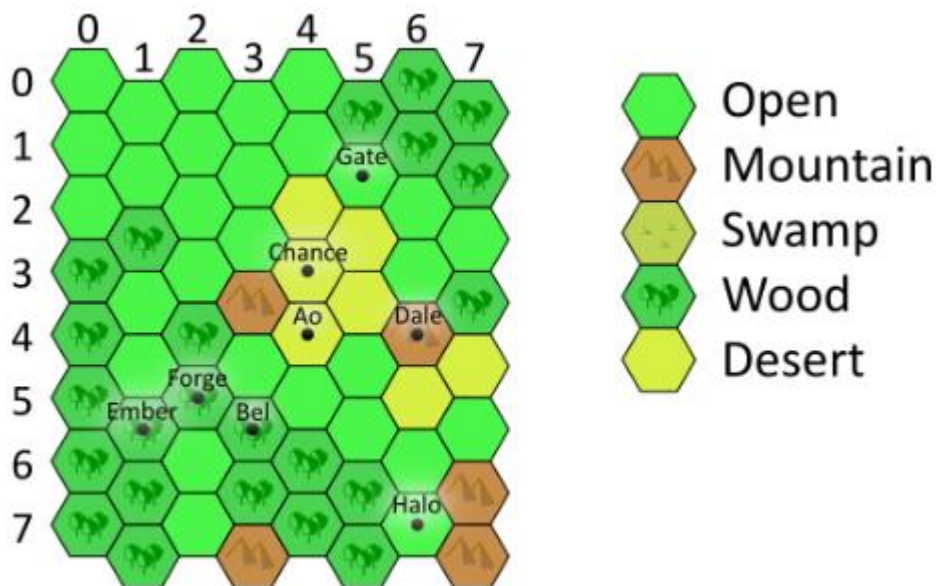
```
root@kali:/media/sf_CTFs/checkpoint/Blockchain# python3 solve.py
74e3f0fc6278b98fa14f1f199d518505
```

אתגר #11: Roads in the wilderness - (קטגוריה: פיתוח. ניקוד: 80)

Various cities populate the map, each has some resources missing others. Your mission, should you accept it, is to connect the cities with roads. Further details are in the readme. Happy road building!

לאתגר צורף קובץ עם תיאור מפורט:

You have a map of hexagonal tiles. Each tile has coordinates (column, row).
For example:



Each tile has a terrain type. There are five different types of terrain: mountain, wood, open, swamp, and desert. On some tiles there are cities. The cities in the map above are on tiles: Ao (4,4), Bel (3,5), Chance (4,3), Dale (6,4), Ember (1,5), Forge (2,5), Gate (5,1), Halo (6,7). A city has resources. There are six types of resources: produce, wood, stone, clay, ore, and textile. Each city is missing some resource or resources.

In this example:

- * Ao has stone
- * Bel has produce and stone
- * Chance has textile and clay
- * Dale has wood and clay
- * Ember has ore, textile and clay
- * Forge has ore and stone
- * Gate has ore

* Halo has produce and ore

Each terrain type has a movement cost:

* Mountain = 6

* Wood = 2

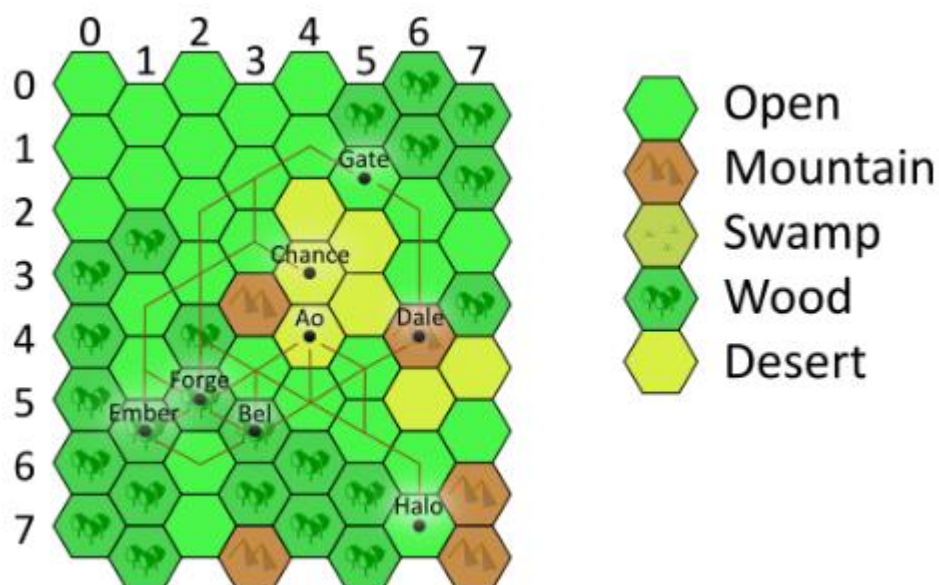
* Open = 1

* Swamp = 4

* Desert = 7

Your job is to find the minimal roads system with the minimal total cost, per city, that will enable all the cities access to all the resources.

Here is the solution for the map above:



The input is a description of map:

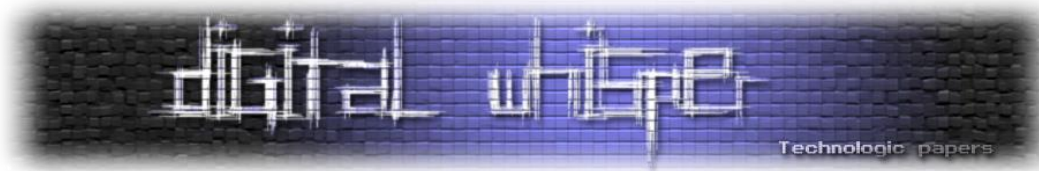
* a list of columns, each a list of tiles

* a list of cities (the city coordinates and the resources it has)

Example input (same as the map above):

Map terrain:

```
[open, open, open, wood, wood, wood, wood, wood],
[open, open, wood, open, open, wood, wood, wood],
[open, open, open, open, wood, wood, open, open],
[open, open, open, mountain, open, wood, wood, mountain],
[open, open, desert, desert, desert, open, wood, wood],
[wood, open, desert, desert, open, open, wood, wood],
```



[wood, wood, open, open, mountain, desert, open, open],
[wood, wood, open, wood, desert, open, mountain, mountain]

Cities:

(4, 4), Stone
(3, 5), Produce, Stone
(4, 3), Textile, Clay
(6, 4), Wood, Clay
(1, 5), Ore, Textile, Clay
(2, 5), Ore, Stone
(5, 1), Ore
(6, 7), Produce, Ore

The solution should be a text describing the roads, each line a road. A road is a list of tiles.

For example (part of the solution to the example):

(6, 7), (6, 6), (5, 5), (4, 5), (3, 5)
(6, 7), (6, 6), (5, 5), (4, 5), (3, 4), (2, 5), (1, 5)
(6, 7), (6, 6), (5, 5), (5, 4), (6, 4)
...

The input is at <http://3.122.27.254/map>

Post your solution at <http://3.122.27.254/solution>

עלינו למצוא עבור כל עיר את מערכת המסלולים בעלת העלות המינימלית, שתעניק לעיר גישה לכל ששת המשאבים.

נתחיל ממספר הבהרות:

- יש למזער את העלות ביחס לכלל המשאבים יחדיו, ולא ביחס לכל משאב בנפרד. לדוגמא: נניח שעיר א' זקוקה לשני משאבים (X ו-Y), כאשר משאב X נמצא בערים ב', ג' ומשאב Y נמצא בערים ב', ד'. נניח עוד שהמרחק לעיר ב' הוא 6 יחידות, והמרחק לכל אחת מהערים ג', ו-ד' הוא 4 יחידות. לכן, ניתן למלא צורך זה על ידי מסלול אחד עם עלות של 6 לעיר ב' או שני מסלולים, כל אחד עם עלות של 4 לערים ג' ו-ד'. במקרה כזה, עלינו לבחור במסלול לעיר ב', מכיוון שהעלות הכוללת של האלטרנטיבה תהיה 8.
- מסלול הוא אוסף של נקודות ציון מנקודה א' לנקודה ב'. העלות של מסלול היא סכום העלויות של כל נקודות הציון למעט נקודת הציון הראשונה במסלול.
- אם הפתרון שלנו כולל מסלול מעיר א' לעיר ב' ואת אותו המסלול מעיר ב' לעיר א', עלינו לכלול את שני המסלולים בתשובה.



כעת נעבור לפתרון. החלק הראשון של הפתרון אמור לקפוץ מיידית לכל מי שאי פעם למד קורס בסיסי באלגוריתמים. המעבר בין "נקודות ציון/אריחים" ו"כבישים" לפי נוסח השאלה לבין גרף עם קודקודים וקשתות צריך להיות אוטומטי. העלות של התנועה בין נקודות ציון נקראת בתורת הגרפים "משקולות אי-שליליות על הקשתות", והיא אמורה להקפיץ מיד את אחד האלגוריתמים המפורסמים ביותר בתורת הגרפים: [אלגוריתם דייקסטרה](#) (Dijkstra).

אלגוריתם דייקסטרה, פרי יצירתו של אדסחר דייקסטרה, פותר את בעיית מציאת המסלול הקצר ביותר מנקודה בגרף ליעד. מכיוון שניתן למצוא באמצעות אלגוריתם זה, בזמן זהה, את המסלולים המהירים לכל הנקודות בגרף, בעיה זאת נקראת לעיתים מציאת המסלולים הקצרים מנקודה.

האלגוריתם עובד על גרף נתון, מכוון או לא מכוון, בעל משקולות אי-שליליות על הקשתות. המשקולות בגרף מסמלות מרחק. משמעותו של המסלול הקצר ביותר בין שתי נקודות היא המסלול בעל סכום המשקולות הנמוך ביותר בין שתי הנקודות.

אלגוריתם דייקסטרה ייתן לנו את המרחק הקצר ביותר מעיר מסוימת לכל עיר אחרת על המפה (למעשה, לכל קודקוד אחר על המפה). לאחר מכן, נצטרך לבצע חישוב נוסף כדי לבנות את אוסף המסלולים שיענה על דרישות השאלה, אך כרגע נדחה שלב זה ונתחיל לבנות את התשובה שלב אחרי שלב. לשם כך, נחלק את הפתרון שלנו לשלושה חלקים לוגיים:

1. ייצוג של הגרף כאובייקט בפני עצמו
2. הפעלת אלגוריתם דייקסטרה ע"מ לחשב את המסלולים הקצרים ביותר בין הערים השונות
3. לוגיקה שמשתמשת בתוצאת החישוב של דייקסטרה על מנת למצוא את סט המסלולים בעל העלות הנמוכה ביותר, כפי שנדרש מאיתנו

נתחיל מייצוג הגרף. לשם כך הגדרנו מספר מחלקות ייצוג. המחלקה הבאה מייצגת נקודת ציון:

```
class Coordinate(namedtuple("Coordinate", ["x", "y"])):
    __slots__ = ()
    def __repr__(self):
        return f"({self.x}, {self.y})"

    def __eq__(self, other):
        if isinstance(other, Coordinate):
            return (self.x == other.x) and (self.y == other.y)
        return False

    def __hash__(self):
        return hash(self.x) * hash(self.y)
```

היא מממשת מתודות עזר על מנת לוודא ששתי נ"צ שוות אם ערכי ה-x וה-y שלהם שווים. לשם ייצוג המשאבים וסוגי השטח השונים, נגדיר שני Enums:

```
Terrain = Enum('Terrain', zip(['OPEN', 'WOOD', 'MOUNTAIN', 'DESERT', 'SWAMP'],
count(1)))

Product = Enum('Product', zip(['STONE', 'PRODUCE', 'TEXTILE', 'CLAY', 'WOOD',
'ORE'], count(1)))
```




ולבסוף, עיר כוללת נקודת ציון ואוסף של משאבים:

```
City = namedtuple('City', 'coordinate products')
```

הייצוג של הגרף, אם כך, ממומש באופן הבא:

```
class HexagonMap(object):
    NEIGHBORS = [
        [(1, -1), (-1, -1), (0, 1), (1, 0), (0, -1), (-1, 0)], # Even
        [(0, 1), (-1, 0), (1, 0), (0, -1), (-1, 1), (1, 1)]    # Odd
    ]

    def __init__(self, cost_map: typing.Dict):
        self.initialized = False
        self.map = []
        self.cities = []
        self.cost_map = cost_map

    def from_data_file(self, path: str):
        # See full implementation in appendix.
        # In short, creates a 2D array of terrains (self.map),
        # creates a list of cities (self.cities), and initializes
        # self.width and self.height

    def get_neighbors(self, coordinate: Coordinate) -> typing.List:
        """Return the neighbors of a given coordinate"""
        res = []
        for (dx, dy) in self.NEIGHBORS[coordinate.x % 2]:
            new_x = coordinate.x + dx
            new_y = coordinate.y + dy
            if (0 <= new_x < self.height) and (0 <= new_y < self.width):
                res.append(Coordinate(new_x, new_y))
        return res

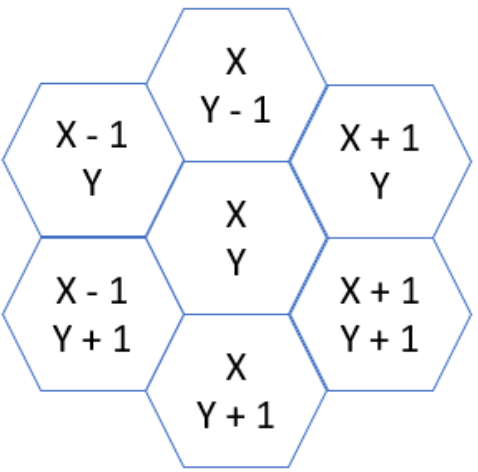
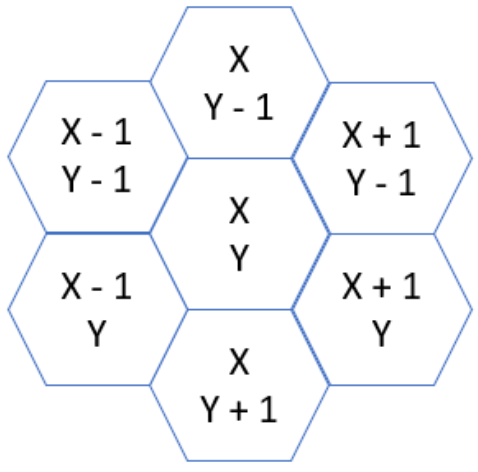
    def get_all_nodes(self) -> typing.Generator[Coordinate, None, None]:
        """Return all nodes in graph"""
        for x in range(self.width):
            for y in range(self.height):
                yield Coordinate(x, y)

    def get_cost(self, coordinate: Coordinate) -> int:
        """Return the cost of a given coordinate"""
        return self.cost_map[self.map[coordinate.x][coordinate.y]]
```

את המימוש המלא של המתודה שאחראית על קריאת הייצוג הטקסטואלי של הגרף השמטנו מכיוון שהוא ארוך ופחות מעניין עבור התמונה הגדולה. ניתן למצוא אותו בנספח א'.

מלבד המתודה הזו, ייצוג הגרף כולל מתודה להחזרת כל נקודות הציון, מתודה לקבלת העלות של נקודת ציון מסוימת (רשימת העלויות מתקבלת כפרמטר באתחול האובייקט) ומתודה להחזרת כל השכנים של נקודת ציון מסוימת.

נקודות הציון של השכנים תלויות בזוגיות של ערך ה-X, לפי החוקיות הבאה:

X אי-זוגי	X זוגי
	

נספח ב' כולל את המפה לדוגמא בצירוף נקודות הציון.

לאחר השלמת הייצוג של הגרף, נעבור לחישוב טבלת העלויות הבסיסית באמצעות אלגוריתם דייקסטרה. נשתמש במימוש של scipy לחישוב טבלת העלויות.

הקוד שאחראי לכך הוא:

```
class RoadOptimizerBase(object):
    def __init__(self, hm: HexagonMap):
        self.hm = hm

        self.city_indices = [self.coordinate_to_index(c.coordinate) for c in
self.hm.cities]

        self.costs, self.paths = shortest_path(self._create_cost_graph(),
method='auto', directed=True,
return_predecessors=True,
unweighted=False, overwrite=False,
indices = self.city_indices)

    def _create_cost_graph(self) -> csr_matrix:
graph = np.zeros((self.hm.width ** 2, self.hm.height ** 2), dtype = int)
for node in self.hm.get_all_nodes():
graph[self.coordinate_to_index(node)][self.coordinate_to_index(node)] = 0
for neighbor in self.hm.get_neighbors(node):
graph[self.coordinate_to_index(node)][self.coordinate_to_index(neighbor)]
= self.hm.get_cost(neighbor)
return csgraph_from_dense(graph)

    def coordinate_to_index(self, coord: Coordinate) -> int:
return coord.x * self.hm.width + coord.y

    def index_to_coordinate(self, index: int) -> Coordinate:
return Coordinate(index // self.hm.width, index % self.hm.width)

    def get_index_in_city_indices(self, city_index: int) -> int:
return self.city_indices.index(city_index)

    def get_path(self, origin_city_index: int, dst_city_index: int) -> typing.List:
origin_index_in_city_indices = self.get_index_in_city_indices(origin_city_index)
```

```
path = [dst_city_index]
dst = dst_city_index
while self.paths[origin_index_in_city_indices, dst] != -9999:
    path.append(self.paths[origin_index_in_city_indices, dst])
    dst = self.paths[origin_index_in_city_indices, dst]

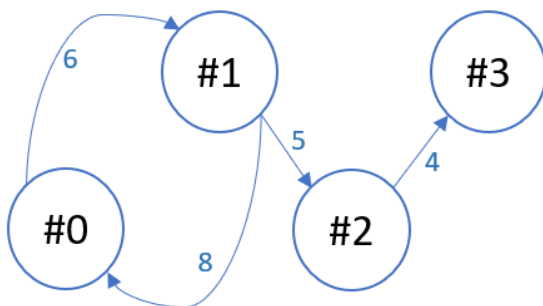
return path[:-1]

def get_roads_for_city(self, origin_city: City):
    raise NotImplementedError("Implement me")
```

הערות מימוש:

ה-API של scipy דורש ייצוג של גרף בתור טבלה, כאשר השורות והעמודות מייצגות נקודות ציון על הגרף. הערך של הטבלה במיקום j, i יהיה העלות של הקשת בין i ל- j . אם אין קשת בין שתי נקודות הציון הללו, הערך יהיה 0.

למשל, להלן ייצוג של גרף פשוט והטבלה המתאימה לו:



	#0	#1	#2	#3
#0	0	6	0	0
#1	8	0	5	0
#2	0	0	0	4
#3	0	0	0	0

במפה שקיבלנו, לא מסמנים קודקודים על הגרף באמצעות מספר סידורי רץ אלא באמצעות נקודות ציון עם ערכי (x, y) , אבל מכיוון שהגרף שלנו מייצג טבלה מלאה, במקרה הזה קל לעבור בין שני הייצוגים באופן הבא:

```
def coordinate_to_index(self, coord: Coordinate) -> int:
    return coord.x * self.hm.width + coord.y

def index_to_coordinate(self, index: int) -> Coordinate:
    return Coordinate(index // self.hm.width, index % self.hm.width)
```

עוד עניין שראוי לציון הוא שה-API של scipy מאפשר לצמצם את החישובים שיבוצעו ע"י אלגוריתם דייקסטרה באמצעות הגדרה מדויקת של נקודות המוצא. כלומר, במקום לחשב את העלות מכל קודקוד אל כל קודקוד אחר על הגרף, האלגוריתם יחשב את רק העלות מקודקודי המוצא שנגדיר לו. נקודות המוצא שלנו הן הערים, לכן נשלח בפרמטר indices רשימה של המספרים הסידוריים (כלומר האינדקסים) של הערים בלבד. בתור תוצאה נקבל חזרה טבלה שבה התא (i, j) מכיל את העלות הנמוכה ביותר עבור הגעה מ- i ל- j , כאשר i מסמל את המיקום ברשימת הערים ששלחנו ב-indices (ולא את המספר הסידורי של קודקוד המוצא) בעוד ש- j את המספר הסידורי של קודקוד היעד.

בנוסף, נקבל טבלה נוספת שתאפשר לנו לשחזר את המסלול שהביא אותנו אל העלות הנמוכה ביותר, כאשר כל תא בטבלה מצביע על הקודקוד הקודם, והמסלול מסתיים עם קבלת הערך 9999.

נותר לנו רק להשתמש בנתונים שאספנו על מנת למצוא את אוסף המסלולים המינימלי שעונה על ההגדרה.

הדרך הנאיבית לעשות זאת היא באמצעות מעבר על כל תתי-הקבוצות של הערים:

1. עבור כל עיר (להלן: "עיר מוצא"):

a. עבור כל תתי-קבוצה של ערים:

i. אם תתי-הקבוצה כוללת את כל המשאבים, נסכום את עלות המסלולים אל כל הערים בתתי-

הקבוצה

b. נבחר את תתי-הקבוצה עם הסכום המינימלי

הפתרון יכלול את אוסף תתי-הקבוצות המינימליות עבור כל עיר מוצא. נתחיל מפתרון נאיבי שכזה ובהמשך נראה כיצד ניתן לייעל אותו ולהקטין את מרחב החיפוש. לצורך כך, ניצור שש קבוצות - קבוצה לכל משאב. נוסיף לכל קבוצה את כל הערים שיש להן את המשאב המתאים. בדוגמא שלנו, הקבוצות ייראו כך:

Stone	Produce	Textile	Clay	Wood	Ore
Ao, Bel, Forge	Bel, Halo	Chance, Ember	Chance, Dale, Ember	Dale	Ember, Forge, Gate, Halo

כעת, עבור כל עיר-מוצא, נבחר עיר אחת מכל קבוצה ונקבל אוסף של ערים שמעניק לנו את כל המשאבים. לדוגמא, עבור עיר המוצא Ao, נתחיל מבחירה של:

Stone	Produce	Textile	Clay	Wood	Ore
Ao	Bel	Chance	Chance	Dale	Ember

נחשב את העלות הכוללת אל כל הערים הללו: 22. נמשיך עם ביצוע בחירה אחרת של עיר אחת מכל קבוצה:

Stone	Produce	Textile	Clay	Wood	Ore
Ao	Bel	Chance	Chance	Dale	Forge

נחשב את העלות הכוללת: 20. מכיוון שהתוצאה הזו טובה יותר מהתוצאה הקודמת, נשמור אותה בתור התוצאה הטובה ביותר. כאשר נסיים לעבור על כל הבחירות האפשריות, נקבל את תתי-קבוצת הערים האופטימלית עבור עיר המוצא Ao.



השיטה הזו אמנם תתן לנו את הפתרון שרצינו, אך זמן הריצה שלה אינו אופטימלי, מכיוון שעבור כל עיר מוצא, היא צריכה לעבור על $4*1*3*2*2*3$ תוצאות. זה אולי לא נורא כשיש לנו רק שש ערים, אך זוהי רק דוגמא. השאלה האמיתית כוללת 512 ערים ולכן עלינו למצוא דרך לצמצם את מרחב החיפוש.

נוכל לעשות זאת באופן הבא: אנחנו הרי עוברים קבוצה-קבוצה ובוחרים עיר מכל קבוצה. כעת, לפני שאנחנו בוחרים עיר מקבוצה מסוימת, נבדוק אם העלות הכוללת עד עתה של כל הערים שכבר בחרנו עולה על התוצאה האופטימלית שמצאנו עד עתה. אם זה המצב - אנחנו יכולים לוותר על בחירה מכל הקבוצות שנותרו - שהרי אין סיכוי שנצליח לשפר את התוצאה הקיימת. את הלוגיקה הזו קל מאוד לממש באמצעות רקורסיה, ובפרט [Backtracking](#):

```
class RoadOptimizerV1(RoadOptimizerBase):
    def __init__(self, hm: HexagonMap):
        super().__init__(hm)

        # Create a list of length len(Product). Each item is a list of all the
        # cities that have the matching product.
        product_to_index = {product:i for i, product in enumerate(Product)}
        self.cities_per_product = [[] for i in range(len(Product))]
        for city in hm.cities:
            for product in city.products:
                self.cities_per_product[product_to_index[product]].append(
                    city)

        self.coordinate_to_index = {city.coordinate: i for i, city in enumerate(hm.cities)}

    def _get_minimal_roads(self, product_index: int, current_cost: int, set_of_cities:
typing.Set):
        if product_index == len(self.cities_per_product):
            if self.best_answer is None or current_cost < self.best_cost:
                self.best_cost = current_cost
                self.best_answer = copy.deepcopy(set_of_cities)
            return

        for city in self.cities_per_product[product_index]:
            already_in_set = city in set_of_cities
            skip = False
            if not already_in_set:
                # We only add a city if it wasn't already in the set
                set_of_cities.add(city)
                updated_cost = current_cost +
int(self.costs[self.origin_index in city_indices][city])
                if updated_cost > self.best_cost and self.best_answer is not None:
                    # Prune the branch now since the current cost is already bad enough,
                    # so no point in continuing down this path
                    skip = True
            else:
                # We didn't add a city (it was already there), so cost hasn't changed
                updated_cost = current_cost

            if not skip:
                # Continue to the next product
                self._get_minimal_roads(product_index + 1, updated_cost, set_of_cities)

            if not already_in_set:
                # We only remove a city if we were the ones to add it
                set_of_cities.remove(city)

    def get_roads_for_city(self, origin_city: City) -> str:
        res = []
        self.best_cost = math.inf
        self.best_answer = None
        origin_city_index = self.coordinate_to_index(origin_city.coordinate)
        self.origin_index in city_indices =
self.get_index in city_indices(origin_city_index)

        self._get_minimal_roads(0, 0, set())
        for dest_city_index in self.best_answer:
```




```
if dest_city_index != origin_city_index:
    res.append(", ".join(str(self.index_to_coordinate(p)) for p in
self.get_path(origin_city_index, dest_city_index)))

return "\n".join(res)
```

המחלקה הזו תמצא את התוצאה האופטימלית עבור עיר מסוימת. באתחול שלה, היא מקבלת את הגרף ואז מחלקת את ערים לקבוצות של משאבים. לאחר מכן, יש לקרוא למתודת `get_roads_for_city` עם עיר-מוצא כלשהי. המתודה תקרא למתודה הרקורסיבית `_get_minimal_roads`. מתודה זו, בתורה, תבצע את הבחירה של ערים כפי שהסברנו קודם. אם עלות-הביניים כבר גבוהה מהתוצאה הטובה ביותר, המתודה תרים את דגל Skip ועל ידי כך תדלג על כל תת-המרחב.

הערת מימוש: בפועל אנו מוסיפים את העיר לאוסף הערים המתגבש רק אם היא לא הייתה שם קודם, ומסירים אותה רק אם הוספנו אותה ב-context הנוכחי. המימוש הזה מסייע לנו לתחזק את `current_distance` בצורה פשוטה (האלטרנטיבה היא לחשב אותו מאפס בכל קריאה רקורסיבית על ידי סכימת העלות של כל מרחקי הערים באוסף הנוכחי, אך אנחנו נמנעים מכך ופשוט מוסיפים את המרחק הנוכחי רק אם הוספנו את העיר אל האוסף באותה הזדמנות).

החלק האחרון של הסקריפט הוא זה שמחבר את כל החלקים יחד:

```
from collections import namedtuple, defaultdict
from scipy.sparse import csr_matrix
from scipy.sparse.csgraph import *
from itertools import count
from enum import Enum
import argparse
import numpy as np
import typing
import time
import copy
import math
import re

# See code from above

def solve(hm: HexagonMap, output_file: str):
    with open(output_file, "w") as o:
        ro = RoadOptimizerV1(hm)
        print("Calculated initial costs in {} seconds".format(time.time() - start))
        for i, city in enumerate(hm.cities):
            print("City {}/{}".format(i + 1, len(hm.cities)), end="\r")
            print(ro.get_roads_for_city(city), file = o)

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('-i', '--input_file', action = 'store',
                        help='Input file name', default = 'example.txt')
    parser.add_argument('-o', '--output_file', action = 'store',
                        help='Output file name')

    args = parser.parse_args()

    hm = HexagonMap({Terrain.OPEN: 1, Terrain.WOOD: 2, Terrain.MOUNTAIN: 6,
                     Terrain.DESERT: 7, Terrain.SWAMP: 4})

    output_file = args.output_file if args.output_file else "out_" + args.input_file

    start = time.time()
    hm.from_data_file(args.input_file)
    solve(hm, output_file)
    end = time.time()
    print("Parsed {} cities, Done in {} seconds".format(len(hm.cities), end - start))
```



כל זה טוב ויפה, וניתן לפתור את התרגיל באמצעות אלגוריתם זה בתוך פחות מדקה, אבל האם זה הפתרון הכי יעיל שניתן להציע? ובכן, התשובה היא שלא, וזה גם היופי באלגוריתמיקה - הסתכלות על הבעיה מנקודת מבט שונה יכולה פתאום להציע פתרון יעיל יותר בכמה סדרי גודל.

הניסיון השני שלנו יפעל באופן הבא: עבור כל עיר-מוצא, ראשית נזהה את רשימת המשאבים שחסרים לעיר זו. לאחר מכן, נעבור על כל החלוקות האפשריות של רשימת המשאבים החסרים לתתי-קבוצות.

כל חלוקה מורכבת למעשה ממספר תתי-קבוצות (לא ריקות) כך שכל משאב מופיע בדיוק בתת-קבוצה אחת. לכל תת-קבוצה כזו, ננסה למצוא את העיר עם העלות הנמוכה ביותר שמכילה את כל המשאבים בתת-הקבוצה (אם אין עיר כזו, החלוקה הזו לא טובה לנו ונדלג הישר לחלוקה הבאה).

בסופו של דבר, נישאר רק עם חלוקות שבהן הצלחנו לזהות עבור כל תת-קבוצה בחלוקה את העיר עם העלות הנמוכה ביותר. נותר רק לבחור מתוך רשימת החלוקות הזו את החלוקה הטובה ביותר, כלומר זו שסכום העלויות של הערים הוא הנמוך ביותר, וסיימנו.

נראה דוגמא קצרה. אם עיר-המוצא שלנו היא Ember, אז רשימת המשאבים שחסרים לעיר היא:
Produce, Wood, Stone

להלן כל החלוקות האפשריות של רשימת המשאבים החסרים לתת-קבוצות:

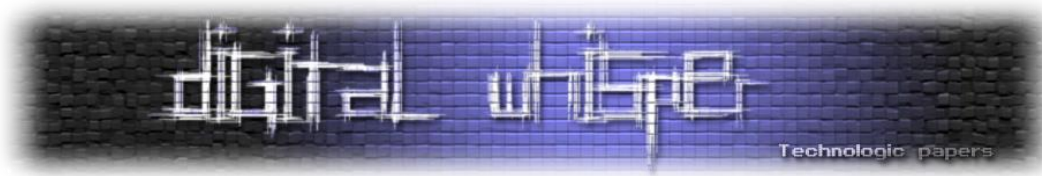
```
[[<Wood>, <Produce>, <Stone>]]  
[[<Wood>], [<Produce>, <Stone>]]  
[[<Wood>, <Produce>], [<Stone>]]  
[[<Produce>], [<Wood>, <Stone>]]  
[[<Wood>], [<Produce>], [<Stone>]]
```

החלוקה הראשונה מורכבת מתת-קבוצה יחידה. מכיוון שאין עיר שמכילה את כל המשאבים בתת-קבוצה זו, היא לא טובה לנו ונדלג עליה.

עבור החלוקה השנייה נגלה שהעיר Dale נותנת את התוצאה הטובה ביותר עבור תת-הקבוצה הראשונה (עלות של 11), והעיר Bel נותנת את התוצאה הטובה ביותר עבור תת-הקבוצה השנייה (עלות של 3). בסך הכל, העלות הכוללת של החלוקה הזו היא 14. נמשיך לבדוק את שאר החלוקות, ולבסוף נבחר בחלוקה עם העלות הנמוכה ביותר.

הקוד הבא יבצע זאת בפועל:

```
class RoadOptimizerV2(RoadOptimizerBase):  
    MAX_VAL = 0xFFFFFFFF  
  
    def __init__(self, hm: HexagonMap):  
        super().__init__(hm)  
  
        # Mapping of each city to the set of products it holds  
        self.cities_to_products = defaultdict(set)  
        for city in hm.cities:  
            for product in city.products:  
                self.cities_to_products[self.coordinate_to_index(city.coordinate)].add(product)  
  
        # The set of all products
```



```
self.all_products = set([p for p in Product])

@classmethod
def partition(cls, collection: typing.Collection) -> typing.List:
    """
    Generate all different partitions of a given collection.
    A partition of a set is a grouping of the set's elements into non-empty subsets,
    in such a way that every element is included in exactly one subset.
    """
    if len(collection) == 1:
        yield [ collection ]
        return

    first = collection[0]
    for smaller in cls.partition(collection[1:]):
        # Insert 'first' in each of the subpartitions' subsets
        for n, subset in enumerate(smaller):
            yield smaller[:n] + [[ first ] + subset] + smaller[n+1:]
        # Put 'first' in its own subset
        yield [ [ first ] ] + smaller

def get_roads_for_city(self, origin_city: City) -> str:
    Pair = namedtuple('Pair', 'cost city_index')
    paths = []

    # The list of products which the origin city does NOT have.
    missing_products = list(self.all_products -
self.cities_to_products[self.coordinate_to_index(origin_city.coordinate)])

    origin_city_index = self.coordinate_to_index(origin_city.coordinate)
    origin_index_in_city_indices = self.get_index_in_city_indices(origin_city_index)

    candidates = []
    for partition in self.partition(missing_products):
        # For every way to partition the missing product list:
        lst = []
        for subset in partition:
            # For each subset of the current way to partition the missing products:
            min_cost = self.MAX_VAL
            min_city = None
            for dest_city_index in self.cities_to_products:
                if set(subset).issubset(self.cities_to_products[dest_city_index]) and
self.costs[origin_index_in_city_indices, dest_city_index] < min_cost:
                    # If the current destination city has all the products in the current
subset, and the cost to the destination
                    # city is lower than the previous minimum, save the current result as the
new minimum
                    min_cost, min_city = int(self.costs[origin_index_in_city_indices,
dest_city_index]), dest_city_index

            if min_cost == self.MAX_VAL:
                # We couldn't find any city with all the missing products of the current
subset
                break

        lst.append(Pair(min_cost, min_city))

    if len(lst) == len(partition):
        # For the current partition of missing products, we were able to find for each
subset the minimal-cost city
        # which has all the resources of the subset.
        partition_cost = sum([pair.cost for pair in lst])
        candidates.append(Pair(partition_cost, [pair.city_index for pair in lst]))

    # Find the solution with the minimal cost among all solution candidates
    total_cost, cities = min(candidates, key=lambda x: x.cost)

    for dest_city_index in cities:
        lst = ", ".join(str(self.index_to_coordinate(p)) for p in
self.get_path(origin_city_index, dest_city_index))
        if lst not in paths:
            paths.append(lst)

    return "\n".join(paths)
```



הבעיה עם הפתרון הראשון היא שפתרון זה יעבור על אותו אוסף ערים מספר פעמים בסדר שונה. את הבעיה פתרנו עם הפתרון השני, שמבצע את המעבר על ידי שימוש בחלוקות (שבהן אין חשיבות לסדר).

עבור הדוגמא, הפלט הוא:

```
(4, 4), (3, 4), (3, 5)
(4, 4), (3, 4), (2, 5), (1, 5)
(4, 4), (5, 4), (6, 4)
(3, 5), (2, 6), (1, 5)
(3, 5), (4, 5), (5, 4), (6, 4)
(4, 3), (3, 2), (3, 1), (4, 1), (5, 1)
(4, 3), (3, 2), (2, 3), (2, 4), (3, 4), (3, 5)
(4, 3), (3, 2), (3, 1), (4, 1), (5, 1), (6, 2), (6, 3), (6, 4)
(6, 4), (5, 4), (4, 5), (3, 5), (2, 6), (1, 5)
(6, 4), (5, 4), (4, 5), (3, 5)
(1, 5), (2, 6), (3, 5)
(1, 5), (2, 6), (3, 5), (4, 5), (5, 4), (6, 4)
(2, 5), (1, 5)
(2, 5), (3, 5)
(2, 5), (3, 4), (4, 5), (5, 4), (6, 4)
(5, 1), (4, 1), (3, 1), (2, 2), (2, 3), (1, 3), (1, 4), (1, 5)
(5, 1), (4, 1), (3, 1), (2, 2), (2, 3), (2, 4), (3, 4), (3, 5)
(5, 1), (6, 2), (6, 3), (6, 4)
(6, 7), (6, 6), (5, 5), (4, 5), (3, 5)
(6, 7), (6, 6), (5, 5), (4, 5), (3, 4), (2, 5), (1, 5)
(6, 7), (6, 6), (5, 5), (5, 4), (6, 4)
```

שימו לב שקיימים פתרונות אחרים עבור הבעיה. למשל, בפתרון לדוגמא, ישנו מעבר בקודקוד (2, 2) בעוד שקיימת תשובה שקולה לחלוטין העוברת דרך קודקוד (2, 3).

הפלט עבור הקלט האמיתי ארוך מדי (1207 מסלולים), לכן נכלול רק את הדגל:

```
root@kali:/media/sf_CTFs/checkpoint/Roads_in_the_wilderness# curl --data-binary @out_map.txt -X
POST http://3.122.27.254/solution && echo
CSA{All_The_Roads_Lead_To_The_Importer}
```



סיכום

כמו בשנים קודמות, גם השנה חברת Check Point פרסמה CTF מוצלח ביותר.

באופן פרדוקסלי, דווקא האתגר עם מספר הנקודות המועט ביותר (Hunting Tinba) היה אחד הקשים, כנראה שאף פעם לא קל למצוא מחט בערמת שחת.

שני תרגילי התכנות הראשונים היו קלים למדי, אך התרגיל השלישי פיצה על כך עם קשר יפה לאלגוריתמים ולתורת הגרפים.

וכמובן, היה נחמד לשוב ולהיזכר בנסיך הפרסי שהיה אחד המשחקים הפופולריים בארץ בזמנו. מרשים מאוד לראות רימייק שנוצר מהדיסאסמבלי של גרסת הדוס.

פרסום הפתרון נעשה באישור היוצרים לאחר שהגיוס ל-CSA 2019 הסתיים, אך האתגר עצמו עדיין יישאר פתוח בתקופה הקרובה, ומי שלא הספיק מוזמן לנצל את ההזדמנות ולהתנסות.

כל הכבוד ליוצרים, מחכים ל-CTF של שנה הבאה!

Reversing	Slot machine ✓ 30	Prince Of Persia ✓ 80	MBA ✓ 120
Development	Lost inside my PPTX ✓ 30	Blockchain ✓ 40	Roads in the wilderness ✓ 80
Misc	Hunting Tinba ✓ 20	Pretty Damn Funny ✓ 70	Da Vinci ✓ 70
Crypto	Pinball Cipher ✓ 30	BadaBase ✓ 40	



נספח א': אתגר #11 - קריאת הייצוג הטקסטואלי של הגרף

המימוש המלא של הפונקציה:

```
def from_data_file(self, path: str):
    if self.initialized:
        raise RuntimeError("Map already initialized")
    try:
        with open(path) as f:
            if f.readline() != "Map terrain:\n":
                raise RuntimeError("Incorrect format: Expected map terrain")

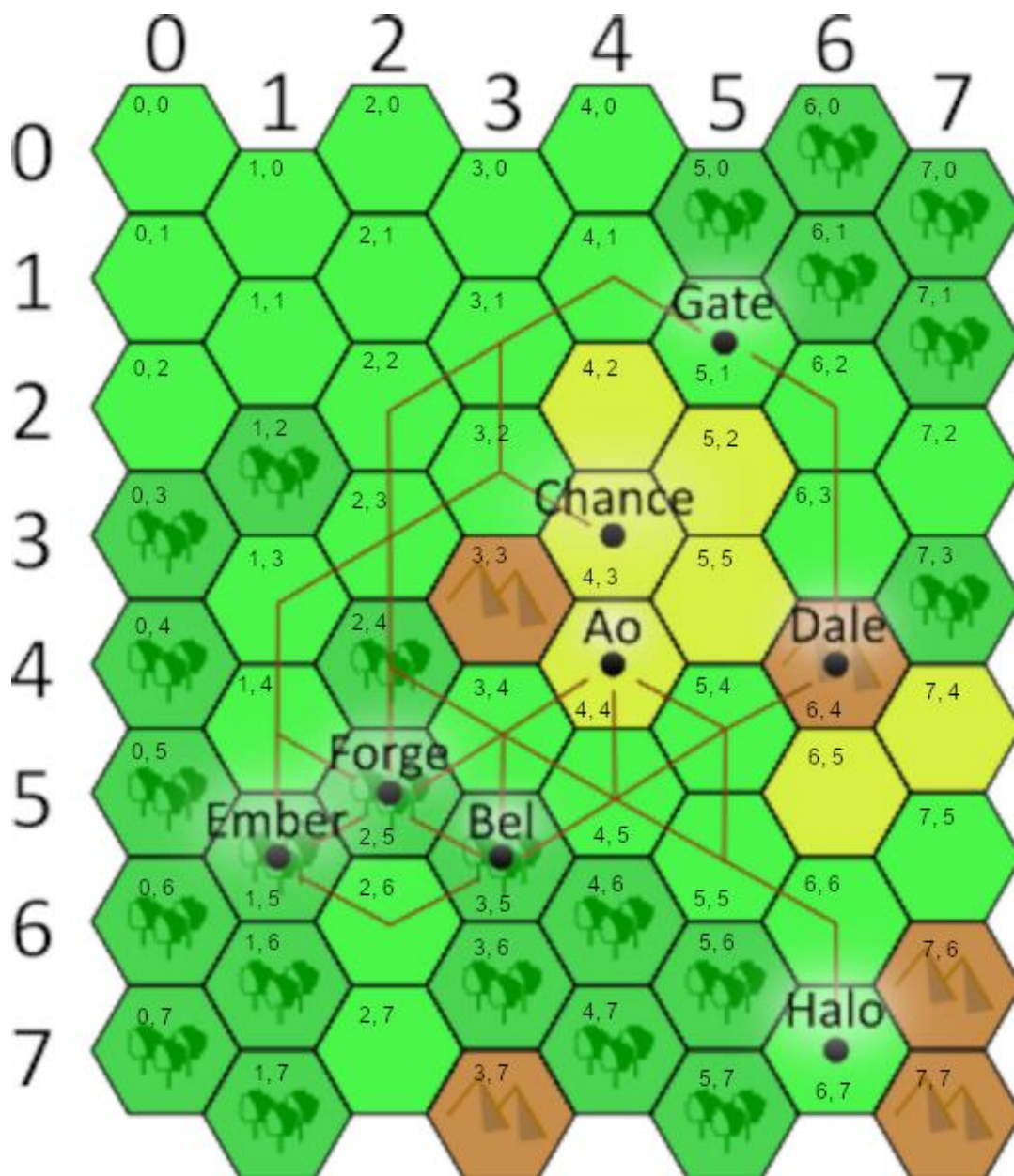
            line = f.readline()
            while line != "Cities:\n":
                if line != "\n":
                    line = line.strip("[],\n")
                    self.map.append(list(map(lambda x:
                        Terrain[x.upper()], line.split(", "))))
                    line = f.readline()

            assert(line == "Cities:\n")
            city_regex = re.compile(r'^\((\d+),\s(\d+)\),\s([\w, ]+)\$')

            while line != "":
                match = city_regex.match(line.rstrip())
                if match:
                    coord = Coordinate(int(match.group(1)),
                                        int(match.group(2)))
                    products = tuple(map(lambda x: Product[x.upper()],
                                        match.group(3).split(", ")))
                    city = City(coord, products)
                    self.cities.append(city)
                line = f.readline()

            self.width = len(self.map[0])
            self.height = len(self.map)
            self.initialized = True
    except RuntimeError as e:
        raise e
    except Exception as e:
        raise RuntimeError("Error parsing data file: {}".format(e))
```

נספח ב': אתגר #11 - המפה לדוגמא בצרוף נקודות ציון



- | | |
|---|----------|
| 1 | Open |
| 6 | Mountain |
| 4 | Swamp |
| 2 | Wood |
| 7 | Desert |