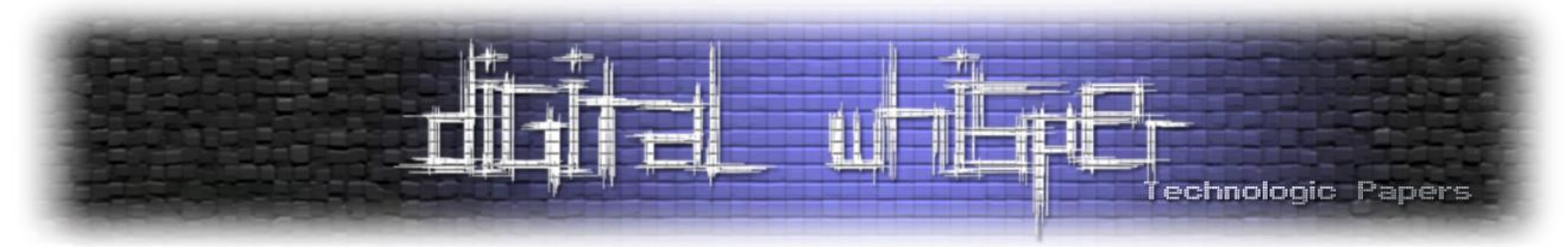


All Your WiFi Repeater are Belong to Us

מחקר חולשות על רכיב תקשורת מסוג מגדיל טווח - חלק ב'

מאת עומר כספי

הקדמה

במאמר זה אציג את ההמשך של מחקר החולשות שביצעתי על רכיב התקשורת מסוג Repeater בחלק א' שפורסם בגיליון 86 של המגזין.

במאמר זה אציג איך הגעתי לקבלת **שליטה מלאה על רכיב זה** בשל היכולת להחליף את ה-Firmware[1] של הרכיב.

תזכורת עד היכן הגענו במאמר הקודם:

במאמר הקודם הגענו למצב בו יש לנו יכולת Remote Code Execution על הרכיב.

ליכולת היו מספר מגבלות:

1. הבינאריים שהעלנו ויכלנו להריץ ישבו במערכת קבצים **זמנית** כלומר לאחר כיבוי המכשיר מה שהעלנו אבד ואיבדנו את השליטה על הרכיב
2. ה-Buffer שהשתמשנו בו להזרקת פקודות היה יחסית קטן (25 תווים) ולכן היינו מוגבלים בהזרקות שיכלנו לבצע
3. לא יכלנו לשלוט על החלק התקשורתי של הרכיב (כלומר להפיל פאקטות או לשנות את תוכנן).

בחלק זה של המאמר הציג את כלל השלבים שביצעתי כדי לעקוף את המגבלות הללו.

[1] Firmware - החלק התוכנתי הקשור בתפעול החומרה של המכשיר, במכשירי Embedded זו יכולה להיות מערכת ההפעלה

הגדלת ה-Buffer

בפעם שעברה מצאנו את ההזרקה בקובץ CGI בשם webupg נמשיך להסתכל במחרוזות מעניינות:

```
.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
```

נסתכל במחרוזת של ה-TFTP:

```
loc_40505C:
lw      $v0, 0x148+var_128($sp)  # Load Word
addiu   $s1, $sp, 0x148+var_110  # Add Immediate Unsigned
lw      $a3, 0x148+var_118($sp)  # Load Word
lui     $a2, 0x40        # Load Upper Immediate
move    $a0, $s1         # s
sw      $v0, 0x148+var_138($sp)  # Store Word
la      $a2, aUsrBinTftpGRSL  # "/usr/bin/tftp -g -r %s -l /var/image.im"...
lw      $v0, 0x148+var_11C($sp)  # Load Word
li      $a1, 0x100       # maxlen
la      $t9, snprintf    # Load Address
jalr    $t9 ; snprintf   # Jump And Link Register
sw      $v0, 0x148+var_134($sp)  # Store Word
lui     $a0, 0x40        # Load Upper Immediate
lw      $gp, 0x148+var_130($sp)  # Load Word
la      $t9, system      # Load Address
jalr    $t9 ; system     # Jump And Link Register
la      $a0, aRmVarImageImgR  # "rm /var/image.img -rf"
lui     $a0, 0x40        # Load Upper Immediate
lw      $gp, 0x148+var_130($sp)  # Load Word
la      $t9, system      # Load Address
jalr    $t9 ; system     # Jump And Link Register
la      $a0, aRmVarTftpLogRf  # "rm /var/tftp.log -rf"
lui     $a0, 0x40        # Load Upper Immediate
move    $a1, $s1
jal     UPG_CreateScript  # Jump And Link
la      $a0, aBinShS     # "#!/bin/sh\n%s"
li      $v0, 1           # Load Immediate
lw      $gp, 0x148+var_130($sp)  # Load Word
lui     $a1, 0x40        # Load Upper Immediate
sw      $v0, 0x148+var_138($sp)  # Store Word
lui     $v0, 0x41        # Load Upper Immediate
la      $t9, PC_StateMachine  # Load Address
la      $v0, s_ucTftpStateMachine  # Load Address
lui     $a3, 0x40        # Load Upper Immediate
la      $a1, aTftp       # "tftp"
la      $a3, UPG_ProcCtrl  # Load Address
move    $a0, $zero
sw      $v0, 0x148+var_134($sp)  # Store Word
jalr    $t9 ; PC_StateMachine  # Jump And Link Register
li      $a2, 1           # Load Immediate
lw      $gp, 0x148+var_130($sp)  # Load Word
bnez    $v0, loc_4051C8  # Branch on Not Zero
lui     $v0, 0x44D       # Load Upper Immediate
```

נראה שהפעם משתמשים ב-Buffer בגודל 256 תווים על סטרינג שממנו נשאר לנו 188 תווים להזרקה, עם Buffer כזה נוכל להשתמש בהזרקה בהרבה יותר קלות.

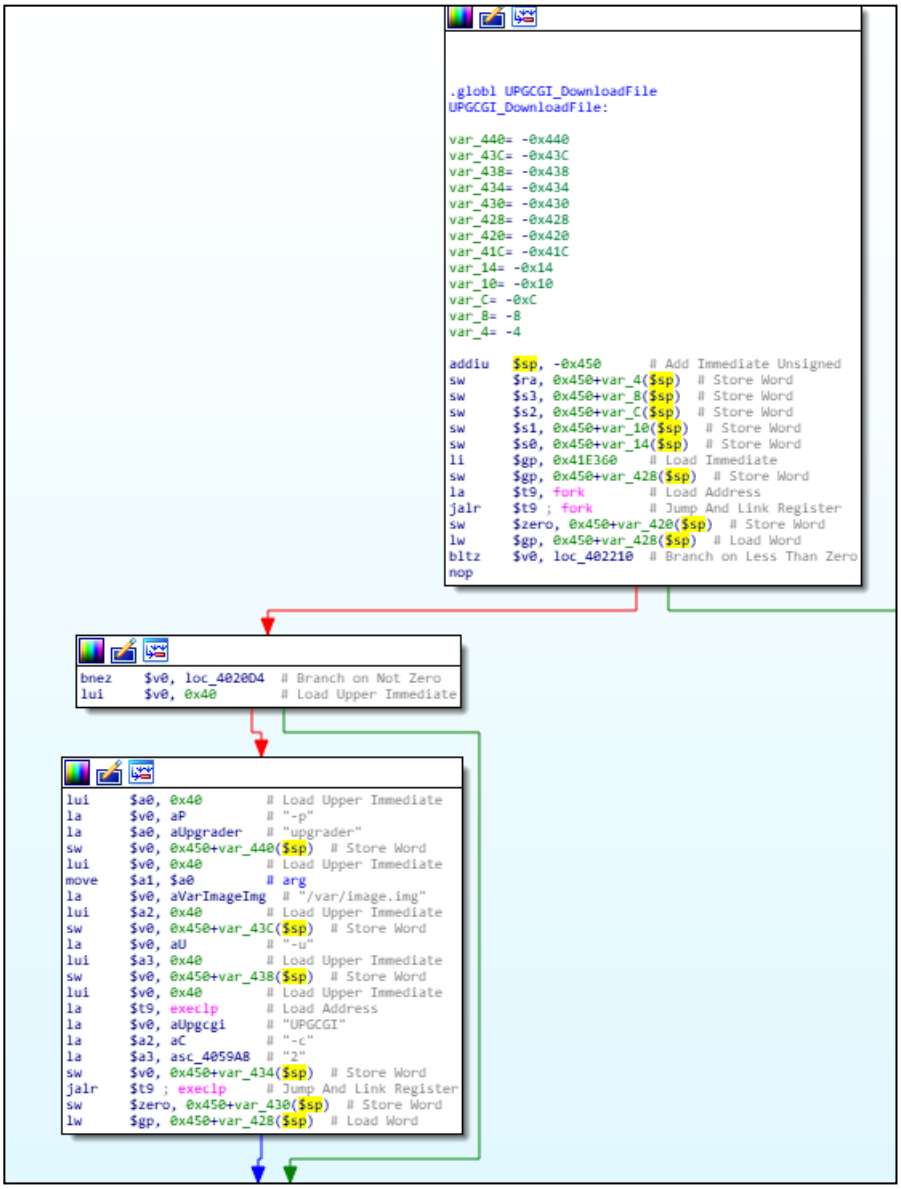
השגת Persistency על הרכיב: שלב התכנון

הכיוון הראשון שלי היה לבדוק את השלב בו הרכיב שומר את הקונפיגורציה. נראה שאם הרכיב מסוגל לשמור את סיסמאת ה-WiFi שלי אז כנראה יש לו מקום כלשהו לשמור את הקונפיגורציה שלו ואולי אני אוכל להשתמש במקום הזה כדי להשיג persistency.

התחלתי לחפש בקובץ webupg ומצאתי כמה פונקציות מעניינות בשם:

- UPGCGI_DoConfig_Upgrade
- UPGCGI_DoFirmware_Upgrade
- UPGCGI_DownloadFile

נרצה להתחקות אחרי תהליך השדרוג של הקונפיגורציה וה-Firmware כדי להבין איך המכשיר שומר דברים באופן קבוע. נסתכל על הפונקציה UPGCGI_DownloadFile:



נראה שהפונקציה יוצרת process חדש אשר קורא לכלי בשם upgrader:

```
loc_40351C:
la      $t9, strcmp        # Load Address
move    $a0, $s0           # s1
jalr    $t9 ; strcmp       # Jump And Link Register
addiu   $a1, (aDownloadconfig - 0x400000)  # "downloadConfig"
lw      $gp, 0x38+var_28($sp)  # Load Word
bnez    $v0, loc_403548    # Branch on Not Zero
lui     $a1, 0x40          # Load Upper Immediate
```

נראה שאנחנו צריכים להעביר בפרמטר name (שקובע איזה פעולה סקריפט ה-CGI יעשה) את הערך downloadConfig כדי לקרוא לפונקציה. כאשר פניתי לרכיב עם הפרמטר downloadConfig שיוביל אותי לפונקציה, ירד אל המחשב שלי קובץ XML אשר מכיל את הקונפיגורציה:

```
<X_TWSZ-COM_UsrDNSServers t="s"></X_TWSZ-COM_UsrDNS
<DNSServers t="s"></DNSServers>
<DNSOverrideAllowed t="b">0</DNSOverrideAllowed>
<DNSEnabled t="b">1</DNSEnabled>
```

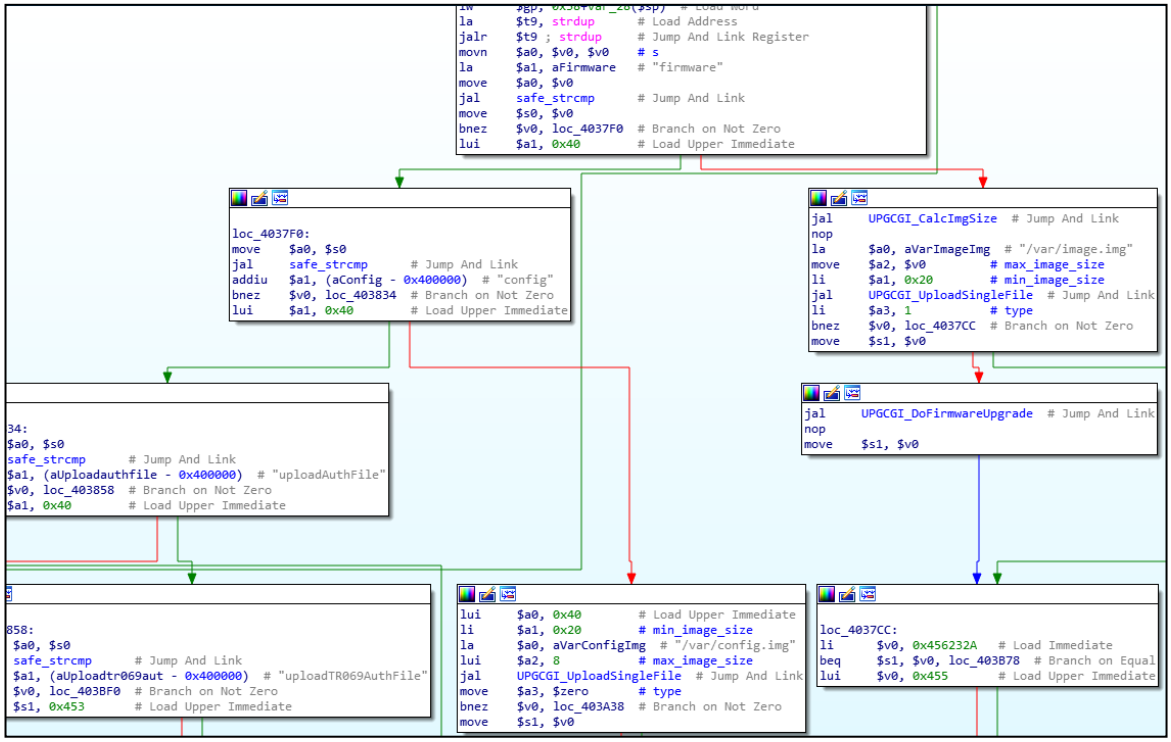
כמובן לא אעבור על כל הקובץ אבל נשים לב שהוא שומר קונפיגורציה של דברים כגון שרתי DNS ושאר הגדרות תקשורתיות מעניינות. נסתכל על הפונקציה UPGCGI_DoConfig_Upgrade:

```
addiu   $sp, -0x50         # Add Immediate Unsigned
sw      $ra, 0x50+var_4($sp)  # Store Word
li      $gp, 0x41E360      # Load Immediate
sw      $gp, 0x50+var_20($sp)  # Store Word
la      $t9, fork          # Load Address
jalr    $t9 ; fork         # Jump And Link Register
sw      $zero, 0x50+var_18($sp)  # Store Word
lw      $gp, 0x50+var_20($sp)  # Load Word
bltz    $v0, loc_402024    # Branch on Less Than Zero
nop

bnez    $v0, loc_401FE0    # Branch on Not Zero
lui     $v0, 0x40          # Load Upper Immediate

lui     $a0, 0x40          # Load Upper Immediate
la      $v0, aP            # "-p"
la      $a0, aUpgrader     # "upgrader"
sw      $v0, 0x50+var_40($sp)  # Store Word
lui     $v0, 0x40          # Load Upper Immediate
move    $a1, $a0           # arg
la      $v0, aVarConfigImg  # "/var/config.img"
lui     $a2, 0x40          # Load Upper Immediate
sw      $v0, 0x50+var_3C($sp)  # Store Word
la      $v0, aU            # "-u"
lui     $a3, 0x40          # Load Upper Immediate
sw      $v0, 0x50+var_38($sp)  # Store Word
la      $v0, aUpgcgi       # "UPGCGI"
la      $a2, aC            # "-c"
sw      $v0, 0x50+var_34($sp)  # Store Word
la      $v0, aM            # "-m"
la      $a3, asc_405980    # "3"
sw      $v0, 0x50+var_30($sp)  # Store Word
lui     $v0, 0x42          # Load Upper Immediate
la      $t9, execlp        # Load Address
la      $v0, s_DeviceMode  # Load Address
sw      $v0, 0x50+var_2C($sp)  # Store Word
jalr    $t9 ; execlp       # Jump And Link Register
sw      $zero, 0x50+var_28($sp)  # Store Word
lw      $gp, 0x50+var_20($sp)  # Load Word
```

נראה שגם היא קוראת לכלי השדרוג כדי לשדרג את הקונפיגורציה. כלומר, עכשיו שיש לנו קובץ קונפיגורציה תקין נוכל לערוך אותו, ולשים (לדוגמא) שרת DNS משלנו כך שכל מי שמחובר דרך ה-Repeater יגיע לאתרים שאני אכוון עליהם, זה יכול להיות ווקטור תקיפה להמשך השתלטות על מכשירים נוספים של הקורבן.

אך מכיוון שאנו רוצים להשיג persistence על המכשיר נתמקד מעכשיו בתהליך שדרוג ה-firmware כי אם נוכל להחליף את מערכת הקבצים או הקרנל - נוכל לגרום למכשיר לעשות מה שנרצה.

נסתכל האם יש בדיקות תקינות כלשהן שקובץ ה-firmware או קונפיגורציה עוברים:



כמו שניתן לראות, גם במקרה של ה-Firmware וגם בקונפיגורציה (במקרה של קונפיגורציה הקריאה לא נכנסה לתמונה) נקראת הפונקציה UPGCGI_UploadSingle ורק במידה והיא מחזירה 0 נקרא השדרוג.

נפרט בפונקציה רק את החלקים הרלוונטים:

```
loc_402C9C:
la      $t9, remove_CRLF_and_save_buffer  # Load Address
lui     $s0, 0x42          # Load Upper Immediate
addiu   $a0, $s0, (sLineBuffer - 0x420000)  # original_buffer
addiu   $a1, $sp, 0x108+n  # size
jalr    $t9 ; remove_CRLF_and_save_buffer  # Jump And Link Register
addiu   $a2, $sp, 0x108+var_EC  # buffer
addiu   $a0, $s0, (sLineBuffer - 0x420000)  # ptr
move    $a3, $s1           # s
lw      $gp, 0x108+var_F8($sp)  # Load Word
li      $a1, 1             # size
lw      $a2, 0x108+n($sp)  # n
la      $t9, fwrite        # Load Address
jalr    $t9 ; fwrite       # Jump And Link Register
la      $s0, sLineBuffer   # Load Address
move    $v0, $zero
lw      $gp, 0x108+var_F8($sp)  # Load Word
addiu   $a0, $sp, 0x108+var_E0  # s1
li      $v1, 0xB           # Load Immediate

loc_402CE4:
addu    $a2, $s0, $v0      # Add Unsigned
addu    $a1, $a0, $v0      # Add Unsigned
lbu     $a2, 0x20($a2)     # Load Byte Unsigned
addiu   $v0, 1             # Add Immediate Unsigned
bne     $v0, $v1, loc_402CE4  # Branch on Not Equal
sb      $a2, 0($a1)        # Store Byte

la      $t9, strncasecmp   # Load Address
lui     $a1, 0x40          # Load Upper Immediate
sb      $zero, 0x108+var_D5($sp)  # Store Byte
la      $a1, aDracoDbAp    # "DRACO_DB_AP"
jalr    $t9 ; strncasecmp  # Jump And Link Register
li      $a2, 0xB           # n
lw      $gp, 0x108+var_F8($sp)  # Load Word
bnez    $v0, loc_402D70    # Branch on Not Zero
nop
```

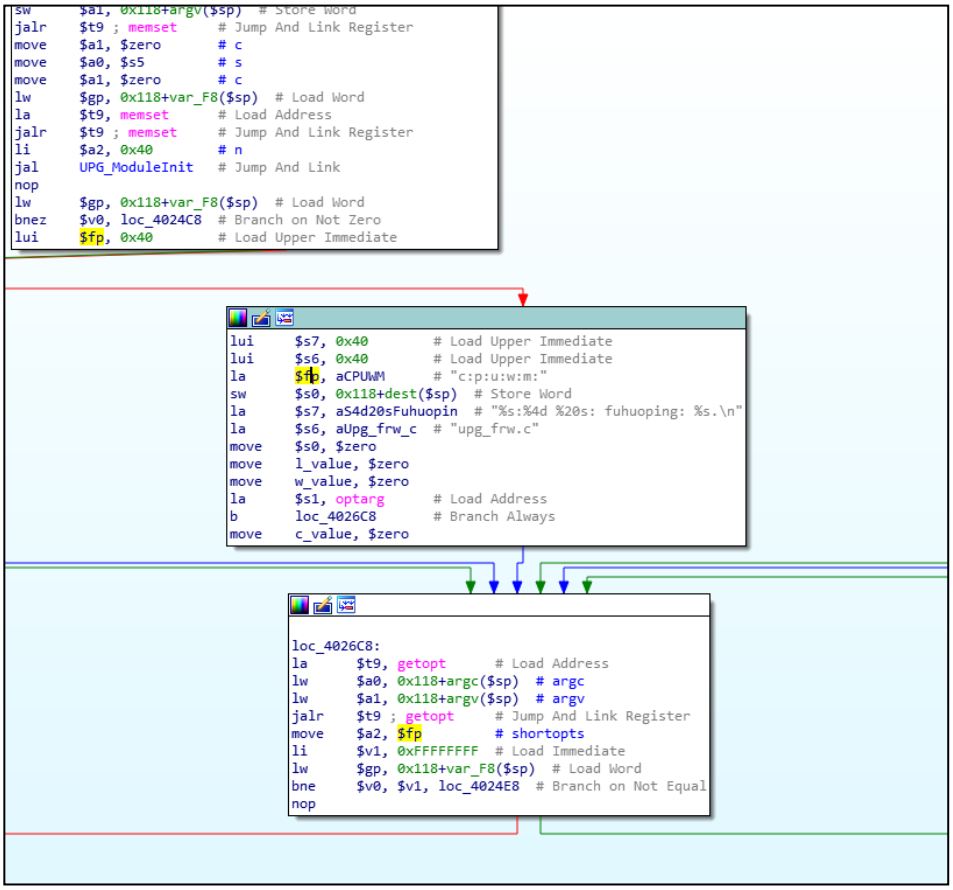
במידה ומדובר בשדרוג firmware נקראים 112 בתים מהקובץ שמועלה (בהמשך נגלה שזה הגודל של ה-Header של קובץ השדרוג) ולאחר העתקה ל-buffer בודקים אם יש מחרוזת עם הערך "DRACO_DB_AP" בהיסט 32 מתחילת הקובץ, מעבר לכך אין שום בדיקות מעניינות חוץ מבדיקות גודל של הקובץ.

נשים לב כי גם הפונקציה UPGCGI_DoFirmware_Upgrade משתמשת באותו בינארי בשם upgrader כדי לשדרג את ה-firmware לכן כדי להמשיך להתחקות אחר תהליך השדרוג נסתכל על קובץ זה.

יש לציין שניסיתי לחפש קובץ שדרוג באתר של היצרן כדי לחסוך את עבודת ההנדסה לאחור של תהליך השדרוג אך היצרן לא פירסם קובץ כזה.

Down the rabbit hole we go

כאשר אנחנו פותחים את הקובץ upgrader ב-Ida נראה שמעבר לקריאת לפונקציית האתחול שמאתחלת IPC API, התוכנית רצה בלולאה ומפרסרת את הפרמטרים שהביאו לה:



ה-IPC API משמש לצורך החזרת ערך אשר יציין אם הריצה של הכלי הצליחה או לא.

יש לציין שהנדסתי לאחור גם את ה-API הנ"ל אך הוא לא רלוונטי למטרה שלנו לכן לא אתמקד בו. ניתן לראות שהפונקציה משתמשת בפונקציה getopt על מנת לפרסר ארגומנטים. אחד הפרמטרים שמעבירים לפונקציה היא רשימה של כל הדגלים שהיא מקבלת מופרדים ב-":", כלומר - האופציות שהכלי יודע לקבל הם: c, p, u, w, m

הפונקציה רצה ובודקת כל פעם איזה דגל getopt החזיר הפעם (ערך ההחזרה של getopt הוא התו של הדגל שכרגע מפרסרים).

אסביר בקצרה על כל אופציה:

- **אופציה c:** ערך מספרי בין 1 ל-3 אשר מציין את הפעולה המבוקשת. כאשר הערך 1 מציין שדרוג firmware, הערך 2 מציין קריאת קובץ קונפיגורציה והערך מציין 3 כתיבת קובץ קונפיגורציה.

- **אופציה w:** ערך מספרי 1 או 0. במידה והערך הוא 0 בשדרוג firmware - הקונפיגורציה תמחק, אחרת הקונפיגורציה נשארת.

- **אופציה p:** שם הקובץ שישמש לפעולה, כלומר אם בחרנו לקרוא את הקונפיגורציה זה יהיה השם של הקובץ שיקרא, אם בחרנו בכתיבת הקונפיגורציה או שדרוג - זה הקובץ שישמש למטרה זו:

```
# int __cdecl UPG_GetUserMID(char *caller_name)
.globl UPG_GetUserMID
UPG_GetUserMID:

var_10= -0x10
var_8= -8
var_4= -4

addiu   $sp, -0x20        # Add Immediate Unsigned
sw      $ra, 0x20+var_4($sp)  # Store Word
sw      $s0, 0x20+var_8($sp)  # Store Word
li      $gp, 0x41C170     # Load Immediate
sw      $gp, 0x20+var_10($sp)  # Store Word
lui     $a1, 0x40         # Load Upper Immediate
la      $t9, strcmp       # Load Address
la      $a1, aUpgcgi      # "UPGCGI"
jalr    $t9 ; strcmp      # Jump And Link Register
move    $s0, $a0
lw      $gp, 0x20+var_10($sp)  # Load Word
beqz    $v0, loc_400F78   # Branch on Zero
li      $v1, 0xD01        # Load Immediate
```

```
la      $t9, strcmp       # Load Address
lui     $a1, 0x40         # Load Upper Immediate
move    $a0, $s0          # s1
jalr    $t9 ; strcmp      # Jump And Link Register
la      $a1, aTr069       # "TR069"
lw      $gp, 0x20+var_10($sp)  # Load Word
beqz    $v0, loc_400F78   # Branch on Zero
li      $v1, 0x10B        # Load Immediate
```

```
la      $t9, strcmp       # Load Address
lui     $a1, 0x40         # Load Upper Immediate
move    $a0, $s0          # s1
jalr    $t9 ; strcmp      # Jump And Link Register
la      $a1, aFtpupg      # "FTPUPG"
lw      $gp, 0x20+var_10($sp)  # Load Word
beqz    $v0, loc_400F78   # Branch on Zero
li      $v1, 0xE01        # Load Immediate
```

- **אפוציה u:** מחרוזת אשר מציינת איזה process קרא לכלי, כאשר הכלי מעבד את המחרוזת שניתנה לו. באופציה זו הוא בודק את המחרוזת מול רשימה ידועה מראש של process-ים וממיר אותם ל-ID שלהם ב-IPC API. במידה והוא לא מוצא מוחזר 1- ומודפסת הודעת שגיאה מתאימה.

- **אופציה m:** מחרוזת אשר מציינת את ה-MID, שהוא ה-ID של ה-process שקרא לכלי.

- **אופציה l:** ערך בין 0 ל-1. במידה והועבר 0 נמחק ה-Log של המערכת.

בקובץ יש פונקציה מעניינות בשם UPG_UpdateImg.

```
var_120= -0x120
var_11E= -0x11E
var_20= -0x20
var_1C= -0x1C
var_18= -0x18
var_14= -0x14
var_10= -0x10
var_C= -0xC
var_8= -8
var_4= -4

addiu   $sp, -0x238      # Add Immediate Unsigned
sw      $ra, 0x238+var_4($sp)  # Store Word
sw      $s6, 0x238+var_8($sp)  # Store Word
sw      $s5, 0x238+var_C($sp)  # Store Word
sw      $s4, 0x238+var_10($sp)  # Store Word
sw      $s3, 0x238+var_14($sp)  # Store Word
sw      $s2, 0x238+var_18($sp)  # Store Word
sw      $s1, 0x238+var_1C($sp)  # Store Word
sw      $s0, 0x238+var_20($sp)  # Store Word
li      $gp, 0x41C170    # Load Immediate
sw      $gp, 0x238+var_228($sp)  # Store Word
la      $t9, access      # Load Address
move    $s0, $a0
lui     $a0, 0x40        # Load Upper Immediate
move    $s5, $a1
la      $a0, aVarUpdate_img  # "/var/update_img"
move    $a1, $zero       # type
move    $s1, $a2
jalr    $t9 ; access     # Jump And Link Register
andi    $s2, $a3, 0xFFFF  # AND Immediate
lw      $gp, 0x238+var_228($sp)  # Load Word
beqz    $v0, loc_402340  # Branch on Zero
lui     $a0, 0x40        # Load Upper Immediate
```

```
la      $t9, access      # Load Address
la      $a0, aVarUpdate_bin  # "/var/update_bin"
jalr    $t9 ; access     # Jump And Link Register
move    $a1, $zero       # type
lw      $gp, 0x238+var_228($sp)  # Load Word
beqz    $v0, loc_402340  # Branch on Zero
nop
```

```
jal     UPG_CheckValid   # Jump And Link
move    $a0, $s0
beqz    $v0, loc_4021B4  # Branch on Zero
move    $s0, $v0
```

בחלק זה של הקוד נבדק האם קיימים שני קבצים בשמות: update_img ו-update_bin. לאחר מכן הפונקציה קוראת לפונקציה בשם UPG_CheckValid, במידה ופונקציה זו תחזיר 0, נמחקים הקונפיגורציות וה-log-ים של המערכת אם הקריאה לקובץ הכילה את האופציות הרלוונטיות. נסתכל לעומק על הפונקציה UPG_CheckValid ונסתכל אילו דברים הפונקציה בודקת:

```
.globl UPG_CheckValid
UPG_CheckValid:

var_E0= -0xE0
flash_size_string= -0xD8
thirty_two_bytes_from_end_of_file= -0xB8
img_or_bin_string_in_file= -0x98
var_94= -0x94
kernel_and_squashfs_size= -0x80
var_7C= -0x7C
logging_var= -0x20
var_14= -0x14
var_10= -0x10
var_C= -0xC
var_8= -8
var_4= -4

addiu   $sp, -0xF0       # Add Immediate Unsigned
sw      $ra, 0xF0+var_4($sp)  # Store Word
sw      $s3, 0xF0+var_8($sp)  # Store Word
sw      $s2, 0xF0+var_C($sp)  # Store Word
sw      $s1, 0xF0+var_10($sp)  # Store Word
sw      $s0, 0xF0+var_14($sp)  # Store Word
li      $gp, 0x41C170    # Load Immediate
sw      $gp, 0xF0+var_E0($sp)  # Store Word
jal     UPG_CheckSum     # Jump And Link
move    $s0, $a0
lw      $gp, 0xF0+var_E0($sp)  # Load Word
beqz    $v0, loc_401104  # Branch on Zero
lui     $a0, 0x40        # Load Upper Immediate
```

הפונקציה קוראת לפונקציה נוספת אשר מחשבת את ה-checksum של הקובץ ומשווה ל-4 בתים האחרונים של הקובץ (לכן נוכל להסיק שב-4 בתים האלה אמור להיות ה-checksum המחושב של הקובץ):

```
loc_40119C:
la      $t9, fread          # Load Address
addiu   $a0, $sp, 0xF0+thirty_two_bytes_from_end_of_file  # ptr
move    $a3, $s0            # stream
li      $a1, 0x24           # size
jalr    $t9 ; fread         # Jump And Link Register
li      $a2, 1              # n
move    $s1, $v0
li      $v0, 1              # Load Immediate
lw      $gp, 0xF0+var_E0($sp)  # Load Word
beq     $s1, $v0, loc_4011E4  # Branch on Equal
lui     $a0, 0x40           # Load Upper Immediate
```

```
loc_4011E4:
la      $t9, strcmp         # Load Address
la      $a1, (aVarUpdate_bin+0xC)  # s2
jalr    $t9 ; strcmp        # Jump And Link Register
addiu   $a0, $sp, 0xF0+img_or_bin_string_in_file  # s1
lw      $gp, 0xF0+var_E0($sp)  # Load Word
bnez    $v0, loc_4013B8  # Branch on Not Zero
move    $a0, $s0            # stream
```

הפונקציה מזיזה את הסמן של הקובץ ל-40 בתים לפני הסוף וקוראת 36 בתים (המשתנה שנקרא אליו, נקרא 32 בתים לפני הסוף כי לא החשבתי את ה-checksum), לאחר מכאן ערך שנקרא משווה למחרוזת "bin" (בהמשך נראה שיש שני סוגים שדרוג bin ו-image):

```
la      $t9, open           # Load Address
la      $a0, aProcLlconfigFl  # "/proc/llconfig/flash_size"
jalr    $t9 ; open          # Jump And Link Register
move    $a1, $zero          # oflag
lw      $gp, 0xF0+var_E0($sp)  # Load Word
bgez    $v0, loc_401244  # Branch on Greater Than or Equal to Zero
move    $s1, $v0
```

```
loc_401488:
la      $t9, open           # Load Address
la      $a0, aProcLlconfigIm  # "/proc/llconfig/img_space"
jalr    $t9 ; open          # Jump And Link Register
move    $a1, $zero          # oflag
lw      $gp, 0xF0+var_E0($sp)  # Load Word
bgez    $v0, loc_401504  # Branch on Greater Than or Equal to Zero
move    $s0, $v0
```

```
loc_401244:
addiu   $s2, $sp, 0xF0+flash_size_string  # Add Immediate Unsigned
move    $a1, $zero          # c
la      $t9, memset         # Load Address
move    $a0, $s2            # s
jalr    $t9 ; memset        # Jump And Link Register
li      $a2, 0x20           # n
move    $a1, $s2            # buf
li      $a2, 0xA            # nbytes
lw      $gp, 0xF0+var_E0($sp)  # Load Word
la      $t9, read           # Load Address
jalr    $t9 ; read          # Jump And Link Register
move    $a0, $s1            # fd
move    $a0, $s2            # nptr
move    $a1, $zero          # endptr
lw      $gp, 0xF0+var_E0($sp)  # Load Word
li      $a2, 0x10           # base
la      $t9, strtol         # Load Address
jalr    $t9 ; strtol        # Jump And Link Register
move    $s3, $v0
move    $s2, $v0
li      $v0, 0xFFFFFFFF  # Load Immediate
lw      $gp, 0xF0+var_E0($sp)  # Load Word
bne     $s3, $v0, loc_4012CC  # Branch on Not Equal
lui     $a1, 0x40           # Load Upper Immediate
```

במידה והמחרוזות שוות נקרא קובץ בשם flash_size (ערכו שווה ל-0x8000000, ערך זה יעזור לנו אחר כך) ומשווה לגודל של הקובץ.

במידה והקובץ קטן או שווה, נוצר קובץ ריק בשם "update_bin" בתיקייה var ומוחזר הערך 0:

```
loc_4013B8:
la      $t9, fseek          # Load Address
move    $a1, $zero          # off
jalr    $t9 ; fseek         # Jump And Link Register
move    $a2, $zero          # whence
lw      $gp, 0xF0+var_E0($sp)  # Load Word
bgez    $v0, loc_40140C     # Branch on Greater Than or Equal to Zero
move    $a3, $s0            # stream

loc_40140C:
la      $t9, fread          # Load Address
addiu   $a0, $sp, 0xF0+var_94  # ptr
li      $a1, 0x70           # size
jalr    $t9 ; fread         # Jump And Link Register
li      $a2, 1              # n
lw      $gp, 0xF0+var_E0($sp)  # Load Word
beq     $v0, $s1, loc_401464  # Branch on Equal
lui     $a0, 0x40           # Load Upper Immediate

loc_401464:
la      $t9, fclose         # Load Address
jalr    $t9 ; fclose        # Jump And Link Register
move    $a0, $s0            # stream
lui     $a1, 0x40           # Load Upper Immediate
lw      $gp, 0xF0+var_E0($sp)  # Load Word
la      $a1, aImgs          # "imgs"
la      $t9, strcmp         # Load Address
jalr    $t9 ; strcmp        # Jump And Link Register
addiu   $a0, $sp, 0xF0+var_7C  # s1
lw      $gp, 0xF0+var_E0($sp)  # Load Word
beqz    $v0, loc_4014B8     # Branch on Zero
lui     $a0, 0x40           # Load Upper Immediate
```

במידה והערך של המחרוזות לא שווה, הסמן של הקובץ מוזז לתחילת הקובץ ונקראים 112 בתים ובהם נבדק שבהיסט 24 בתים בקובץ כתובה המחרוזת "imgs":

```
loc_4014B8:
la      $t9, open           # Load Address
la      $a0, aProcIlconfigIm  # "/proc/llconfig/img_space"
jalr    $t9 ; open          # Jump And Link Register
move    $a1, $zero          # oflag
lw      $gp, 0xF0+var_E0($sp)  # Load Word
bgez    $v0, loc_401504     # Branch on Greater Than or Equal to Zero
move    $s0, $v0

loc_401244:
addiu   $s2, $sp, 0xF0+flash_size_string  # Add Immediate Unsigned
move    $a1, $zero          # c
la      $t9, memset         # Load Address
move    $a0, $s2            # s
jalr    $t9 ; memset        # Jump And Link Register
li      $a2, 0x20           # n
move    $a1, $s2            # buf
li      $a2, 0xA            # nbytes
lw      $gp, 0xF0+var_E0($sp)  # Load Word
la      $t9, read           # Load Address
jalr    $t9 ; read          # Jump And Link Register
move    $a0, $s1            # fd
move    $a0, $s2            # nptr
move    $a1, $zero          # endptr
lw      $gp, 0xF0+var_E0($sp)  # Load Word
li      $a2, 0x10           # base
la      $t9, strtol         # Load Address
jalr    $t9 ; strtol        # Jump And Link Register
move    $s3, $v0
move    $s2, $v0
li      $v0, 0xFFFFFFFF     # Load Immediate
lw      $gp, 0xF0+var_E0($sp)  # Load Word
bne     $s3, $v0, loc_4012CC  # Branch on Not Equal

loc_401504:
addiu   $s1, $sp, 0xF0+flash_size_string  # Add Immediate Unsigned
move    $a1, $zero          # c
la      $t9, memset         # Load Address
move    $a0, $s1            # s
jalr    $t9 ; memset        # Jump And Link Register
li      $a2, 0x20           # n
move    $a1, $s1            # buf
li      $a2, 0xA            # nbytes
lw      $gp, 0xF0+var_E0($sp)  # Load Word
la      $t9, read           # Load Address
jalr    $t9 ; read          # Jump And Link Register
move    $a0, $s0            # fd
move    $a0, $s1            # nptr
move    $a1, $zero          # endptr
lw      $gp, 0xF0+var_E0($sp)  # Load Word
li      $a2, 0x10           # base
la      $t9, strtol         # Load Address
jalr    $t9 ; strtol        # Jump And Link Register
move    $s1, $v0
li      $v1, 0xFFFFFFFF     # Load Immediate
lw      $gp, 0xF0+var_E0($sp)  # Load Word
bne     $s1, $v1, loc_401598  # Branch on Not Equal
lui     $a0, 0x40           # Load Upper Immediate
```

לאחר בדיקה זו נבדק הערך של 4 בתים בהיסט 20 מתחילת הקובץ מול הקובץ img_space (ערכו שווה ל-0x7d0000), במידה והגודל שנקרא מתחילת קובץ השדרוג קטן או שווה - נוצר קובץ ריק בשם update_img בתיקייה var ומוחזר הערך 0.

נחזור לפונקציה UPG_UpdateImg. במידה ועברנו את כל הבדיקות הנדרשות המערכת מדפיסה ללוג:

"update image file on flash by calling kernel !"

ואז מתבצעת קריאה ל-system עם המחרוזת reboot.

רגע, מה? למה שהתהליך הזה יגיד שהוא קורא לקרנל ומיד לאחר מכן יבקש לרסט את המכשיר? ההשערה שלי הייתה שהשדרוג מתרחש בעת עליית המערכת על ידי אחד משני גורמים: הקרנל עצמו (מאוד לא סביר כאשר משדרגים firmware) או ה-bootloader[2] עצמו.

על מנת להבין יותר טוב את תהליך השדרוג החלטתי לבצע dump ל-flash כדי לראות אם נוכל להשיג את ה-bootloader, ועל ידי הרצה או הנדסה לאחור שלו להבין את תהליך השדרוג.

Dumping the flash

נשתמש בפקודה cat על הקובץ /proc/mtd כדי לראות את מבנה המחיצות ב-flash של הרכיב:

```
# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00030000 00010000 "boot"
mtd1: 000fca00 00010000 "kernel"
mtd2: 006d3600 00010000 "rootfs"
```

נראה שהרכיב בנוי מ-3 מחיצות: אחת נקראת boot (כנראה מכילה את ה-bootloader והגדרות), אחת מכילה את הקרנל ואחת מכילה את מערכת הקבצים. חדי העין ישימו לב שהגודל של הקרנל ומערכת הקבצים ביחד הם 0x7d0000. הגדול המקסימלי של שדרוג מסוג image, לכן ניתן להניח ששדרוג מסוג bin כותב על כל ה-flash כולל ה-bootloader ואילו שדרוג מסוג image ישדרג רק את הקרנל ומערכת הקבצים.

מכיוון ששדרוג של ה-bootloader הוא עסק מסוכן (כיוון שאם משהו ידפק בשדרוג שלנו אנחנו עלולים לעשות brick למכשיר) לכן נתמקד בשדרוג מסוג image בלבד. עשיתי dump ל-flash על ידי השימוש ב-shell שכבר יש לנו על הרכיב, נסתכל עליו ב-binwalk:

```
# binwalk entire_repeater_flash.raw

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
6488          0x1958          LZMA compressed data, properties: 0x5D, dictionary
 size: 8388608 bytes, uncompressed size: 107408 bytes
69890         0x11102         Zlib compressed data, default compression
81409         0x13E01         Zlib compressed data, default compression
135426        0x21102         Zlib compressed data, default compression
196608        0x30000         LZMA compressed data, properties: 0x5D, dictionary
 size: 8388608 bytes, uncompressed size: 3374168 bytes
1231360       0x12CA00        Squashfs filesystem, little endian, version 4.0, c
ompression:lzma, size: 2141851 bytes, 751 inodes, blocksize: 65536 bytes, create
d: 2015-02-09 14:01:04

#
```

[2] Bootloader - החלק התוכנתי אשר האחראי לאתחל את בקרי החומרה ולהעלות את מערכת ההפעלה

נראה שקיים קובץ בהיסט 0x30000, זהו סופה של מחיצת ה-bootloader והוא נדחס באלגוריתם LZMA, כנראה שזהו הקרנל וניתן לראות לאחריו את מערכת הקבצים.

מעבר לקרנל ולמערכת הקבצים, נסתכל על שאר הקבצים ש-binwalk חילץ. שלושת הקבצים אשר נדחסו על ידי האלגוריתם Zlib הם עותקים שונים של קובץ הקונפיגורציה, כנראה חלקם משמשים לגיבוי.

נריץ binwalk על קובץ ה-LZMA בתחילת ה-Flash:

```
root@kali:~/Desktop/_entire_repeater_flash.raw.extracted# binwalk 1958

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
86608         0x15250         CRC32 polynomial table, big endian
99996         0x1869C         HTML document header
100720        0x18970         HTML document footer
100772        0x189A4         HTML document header
101132        0x18B0C         HTML document footer
101184        0x18B40         HTML document header
101523        0x18C93         HTML document footer
101576        0x18CC8         HTML document header
101824        0x18DC0         HTML document footer
101876        0x18DF4         HTML document header
102226        0x18F52         HTML document footer

root@kali:~/Desktop/_entire_repeater_flash.raw.extracted#
```

נראה ש-binwalk לא מזהה שום דבר יותר מדי מעניין (הדפי HTML הם דפי ווב לממשק שדרוג), נריץ strings:

```
TBS bootloader V1.0 Build65129 for RTL8197D_DRACO_DB_AP(Oct 27 2014-17:28:17)
abortboot
sysdata_get
eth_init
! B0c@
2s"RR
$b4C
6S&r
P:3*
@%pF`g
bwrV
tfdGT$D
fWvvF
XDHex
JuZTj7z
\dLE<
~6NU^t.
gqr<
$/o|
f-=v
mj>zjZ
l6qnk
IiGM>nw
rt8196d
4prepare_tags
setup_commandline_tag
0### ERROR ### Please RESET the board ###
Flash:
DRAM:
```

המחרוזות שאנחנו רואים מאוד מעניינות! זה נראה כמו ה-bootloader, אני מניח שזה גירסא של U-boot שעברה שינוי של היצרן בגלל שהרבה מהפקודות (המחרוזות גדולות מדי לתמונה) היו לי מוכרות, כמו גם המחרוזת: "Please RESET the board" - היא מחרוזת נפוצה ש-U-boot[3] מדפיס כאשר הוא נכנס ל-panic.

כעת, משאנחנו יודעים שה-bootloader הוא U-boot היינו רוצים לבחון את החומרה ולחפש חיבורי UART ו-JTAG על מנת שנוכל לדבג טוב יותר את תהליך השדרוג,לכן נפתח את המארז ונתחיל לחפש רכיבים.

מי שיתעסק בתוכנה יענש בחומרה

אסביר קודם כל למה אנחנו מחפשים UART ו-JTAG ומה הם בקצרה.

UART הוא פרוטוקול נפוץ לתקשורת סיריאלית, הרבה bootloader-ים מאפשרים ממשק ניהול על החומרה עם פקודות פשוטות יחסית.

JTAG הוא סטנדרט בתעשיית החומרה לבדיקת ה-PCB[4] שיוצר במפעל (בניגוד לתוכנה יכולות להיות תקלות בייצור הלוחות לכן נדרשים כלים לבדיקם) כיום הרבה יצרנים משתמשים בסטנדרט זה גם כדי לדבג את רכיבי החומרה (מעבד, זיכרון Flash וכו') ישירות (הסטנדרט מדבר רק על פינים במחבר JTAG ועל מכונת המצבים של הסטנדרט, אבל לא על איזה מידע עובר ברגלי המידע לכן כל יצרן מממש יכולות debug באופן שונה).

לרוב מפתחים אשר יעבדו ברמת baremetal יעבדו עם JTAG programmer[5] כדי להוריד את התוכנית שלהם ל-RAM ולדבג את המעבד.

ל-UART מחבר נפוץ של 4 פינים שכוללים ערוץ שידור (TX), ערוץ קליטה (RX), מתח (VCC) ואדמה (GND). ל-JTAG יש מספר רב של מחברים, תלוי ביצרן של המעבד. למעבדי MIPS יש תקן בשם EJTAG שמדבר על 3 סוגי מחברים: מחבר בעל 6 רגליים, מחבר בעל 12 רגליים (הנפוץ ביותר) ומחבר בעל 14 רגליים.

ננסה לזהות רכיבים על הלוח ואת המחברים המדוברים.

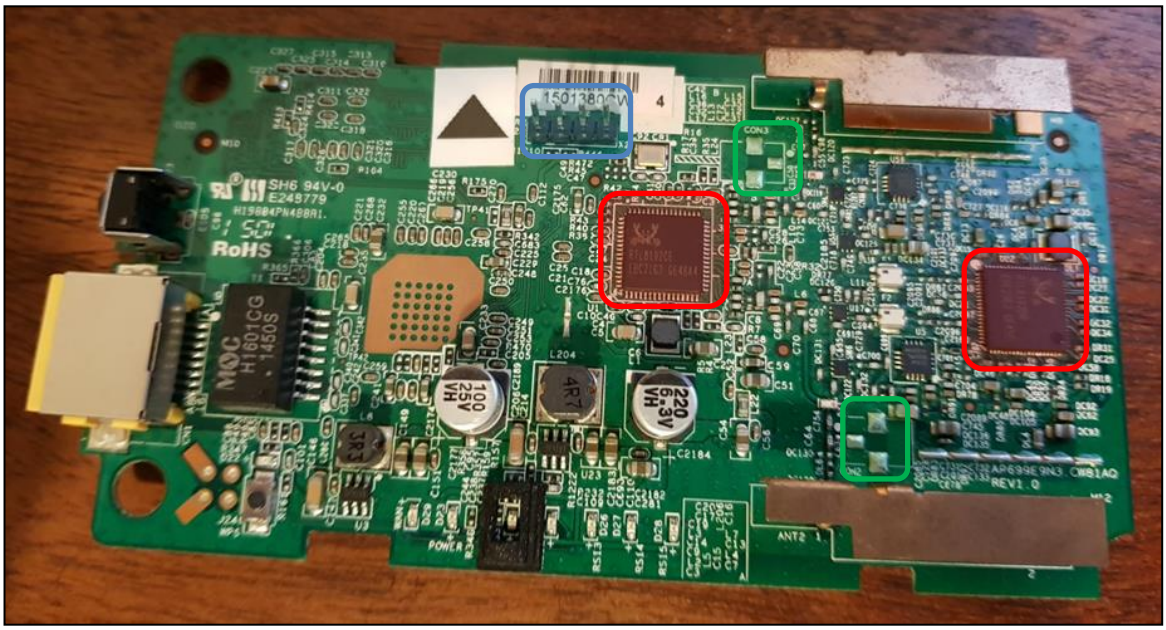
---

[3] Bootloader - U-boot נפוץ למערכות Embedded

[4] PCB - printed circuit board הוא הלוח שעליו מונחים הרכיבים האלקטרוניים שיוצרים את המעגל החשמלי

[5] JTAG Programmer - מכשיר אשר מאפשר שליטה על החומרה דרך ה-JTAG יאפשר שליטה על רכיבים כגון המעבד, ה-RAM זיכרון ה-Flash וכו'

קדמת הלוח:

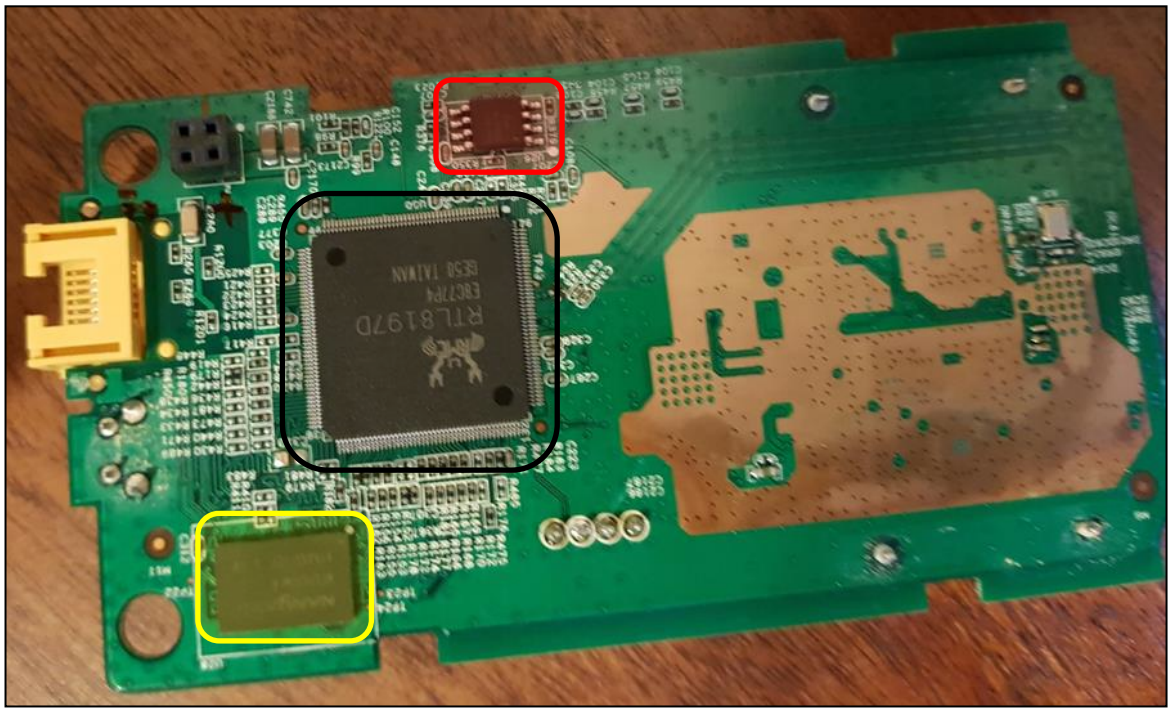


סימנתי את הרכיבים המעניינים בצבעים:

- **אדום**: אלו משדרי WIFI.
- **כחול**: פינים ל-UART למזלנו היה רשום על ה-PCB את שמות הרגליים שציינתי קודם, יש להדגיש שלא היו מולחמות רגליים לפינים במקור אך כדי שאוכל להתחבר לסיריאל ביקשתי מחבר שילחים את הרגליים לשם.
- **ירוק**: פינים שחשבתי שהם אולי JTAG כיוון שאין עוד פינים על הלוח (גם לא מאחורה) אך לאחר שמדדתי את החיבורים עם וולטמטר ו-oscilloscope[6] ראיתי שהרגליים לא מעבירות שום מידע או זרם, לכן הן לא יכולות להיות חיבור JTAG.

[6] Oscilloscope - מכשיר אלקטרוני אשר משקף תנודות גלים אלקטרונים

הלוח מאחורה:



הרכיבים המעניינים גם פה מסומנים בצבעים:

- **שחור**: המעבד עצמו
- **אדום**: ה-flash של הרכיב מסוג SPI NOR
- **צהוב**: ה-RAM של הרכיב

עוד כיוון שניסיתי ולא צלח, הוא בגלל שלא מצאנו JTAG על ה-PCB (יכול להיגרם משלל סיבות לדוגמא שה-flash של הרכיב נצרב במפעל ואז מולחם ללוח), ניסיתי לחפש את רגלי ה-JTAG של המעבד עצמו כיוון שהוא תומך בסטנדרט EJTAG.

על מנת למצוא את רגלי ה-JTAG של המעבד חיפשתי את ה-datasheet[7] שלו אך ללא הצלחה. במידה והייתי מוצא אותן הייתי יכול להלחים אליהן מחבר מתאים ולדבג את המעבד ישירות.

לעומת זאת מצאתי דברים מעניינים אחרים כגון: לוח הפיתוח שעליו החומרה שלנו בוססה, מצאתי גם שאם היה מחבר JTAG על ה-PCB הוא היה אמור להיות מחבר בעל 12 פינים.

כדי לעבוד בצורה נוחה יותר, נרצה לעבוד עם הרכיב ללא הקופסא שלו, ביקשתי עזרה מחבר שמדד את המתח שמספק הספק כוח שיש בקופסא והוא הכין ספק מתח עם מחבר מתאים ללוח, ספק שגם מתחבר לשקע. כדי שנוכל לעבוד עם הסיריאל עלינו למצוא את ה-baud rate[8] שבו הרכיב עובד.

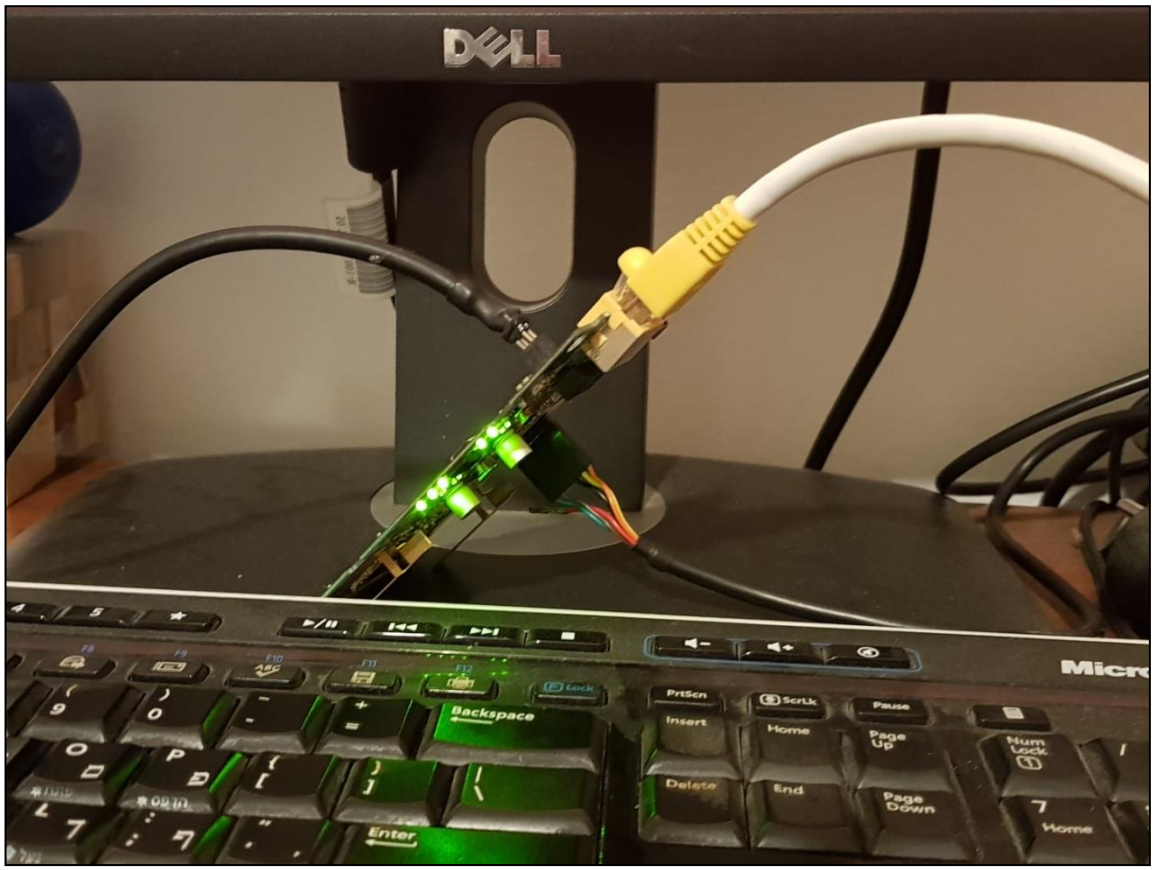
---

[7] Datasheet - מפרט טכני הנותן פרטים על רכיב אלקטרוני או מכני
[8] Baud rate - קצב העברת הנתונים בו התקשורת הסיריאלית עובדת

על מנת שנוכל להתחבר אליו ניקח ממיר USB ל-UART סטנדרטי ונחווט אותו בצורה כזו שתפקידי הרגליים על הרכיב (התפקיד של כל רגל היה רשום על ה-PCB) יתאימו לתפקידי החוטים בממיר.

לרוב, רכיבים עובדים באחד מתוך כמה baud rate-ים סטנדרטים, ניסיתי את כל הערכים עד שמצאתי שהרכיב מדבר בקצב 115200. יכלתי באותה מידה להשתמש בכלים כגון JTAGulator כדי למצוא את הערך הזה.

כעת סביבת העבודה החומרתית שלי נראתה כך:



כעת, משמצאנו חיבור סיריאלי ללוח ואנחנו יודעים את הקצב שלו נוכל להסתכל על תהליך השדרוג בזמן שהוא מתבצע ב-U-boot.

הנדסה לאחור של ה-U-boot

עד שלב זה ניסיתי לשדרג את ה-U-boot עם שימוש בפלט הסיריאלי אך ללא הועיל, קובץ השדרוג לא עבר את הבדיקות (שחלקן הצלחתי לגלות על ידי הדפסות שהתהליך עשה לסיריאל). כאשר הצלחתי לשדרג, הרכיב לא עלה כלל מעבר ל-U-boot כיוון שבשדרוג נכתבים שני ערכים לקונפיגורציה של U-boot שב-flash שנמצאים בהיסט 4 ו-8 בתים בהתאמה בקונפיגורציה.

ערכים אלה הם ההיסט ב-flash שבו נמצא הקרנל וההיסט שבו נמצא מערכת הקבצים, בכל שדרוג ההיסט של מערכת הקבצים לא השתנה ונכתב לשם ערך לא חוקי.

על מנת להבין באופן מלא את תהליך השדרוג אנחנו נדרשים להנדס לאחור את ה-U-boot. ל-U-boot יש שני חלקים אשר נקראים first stage bootloader ו-second stage bootloader.

כאשר המעבד נדלק, אפשר להשתמש רק ב-ram שיש על המעבד עצמו. וכמו שאתם יכולים לתאר - גודל זה הוא מאוד מאוד קטן. לכן ה-first stage bootloader נטען לשם (או רץ מה-flash ישירות) מקנפג רכיבים התחלתיים ואת הבקר של ה-ram ומעתיק את ה-second stage bootloader שתפקידו לטעון את מערכת ההפעלה ולקפוץ אליה.

כאשר מקמפלים את U-boot, ה-second stage שלו מגיע במגוון פורמטים (כגון elf וכו') ולרוב מגיע עם header שאומר מה הוא ה-entry point שלו. יש עוד פורמט אחד בשם bin שהוא הבינארי המקומפל ללא שום header שאומר איפה הוא מתחיל, לכן נאלץ להנדס לאחור את ה-first stage כדי לדעת איפה ה-entry point.

כדי לדעת איפה מתחיל ה-first stage אסביר קצת מה קורה כאשר מעבד MIPS מתחיל לפעול.

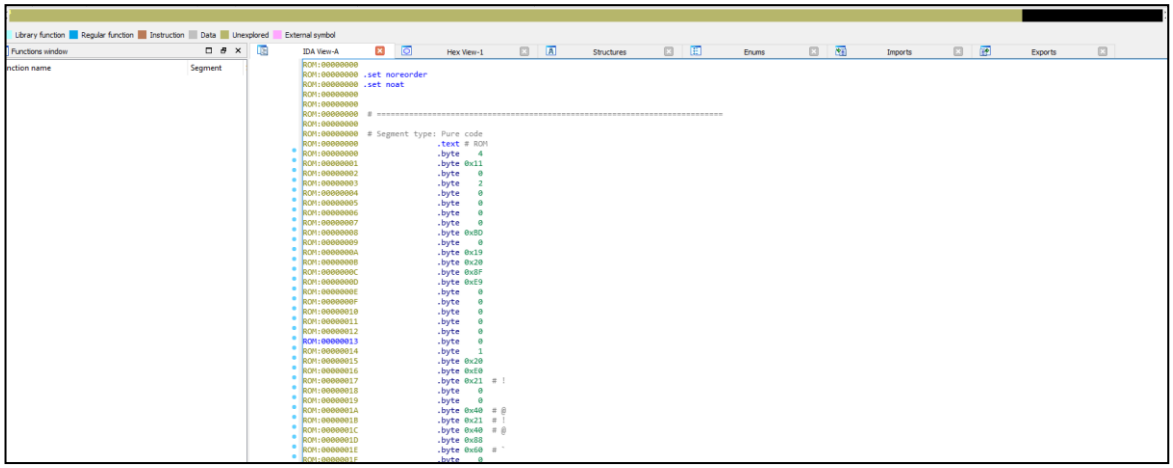
בארכיטקטורת MIPS כאשר המעבד נדלק, מופעל אוטומטית ה-MMU עם מפת כתובות ידועה חלקית. המעבד מנסה לטעון את ה-boot vector[9] שבכתובת הוירטואלית 0xBFC0000. הכתובת הפיזית שמתורגמת יכולה להשתנות על פי היצרן של החומרה. הניחוש שלי היה שמכיוון שחלק מה-flash-ים מסוג NOR תומכים ב-XIP[10] אז המעבד ניגש להיסט 0 ב-flash ומתחיל לרוץ משם.

יש להדגיש שגם אם הניחוש שלי נכון, יהיה קשה לדעת אם הקוד שאנחנו קוראים הוא תקין או לא מכיוון שה-first stage bootloader הוא ממש ספציפי למעבד וללוח שעליו הוא רץ ולכן קיימים איתחולים של כל מיני בקרים, שלרוב יהיו כתיבת ערכים ישירות לכתובות קבועות (כאשר דרייברים וקוד low level מתקשר עם חומרה הוא לרוב קורא או כותב לכתובות שהם הבקר חומרה עצמו), ואין לנו שום פרטים עליהן כיוון שאין לנו שום מידע על החומרה. דברים כגון מפת זיכרון של הבקרים יכלה מאוד לעזור.

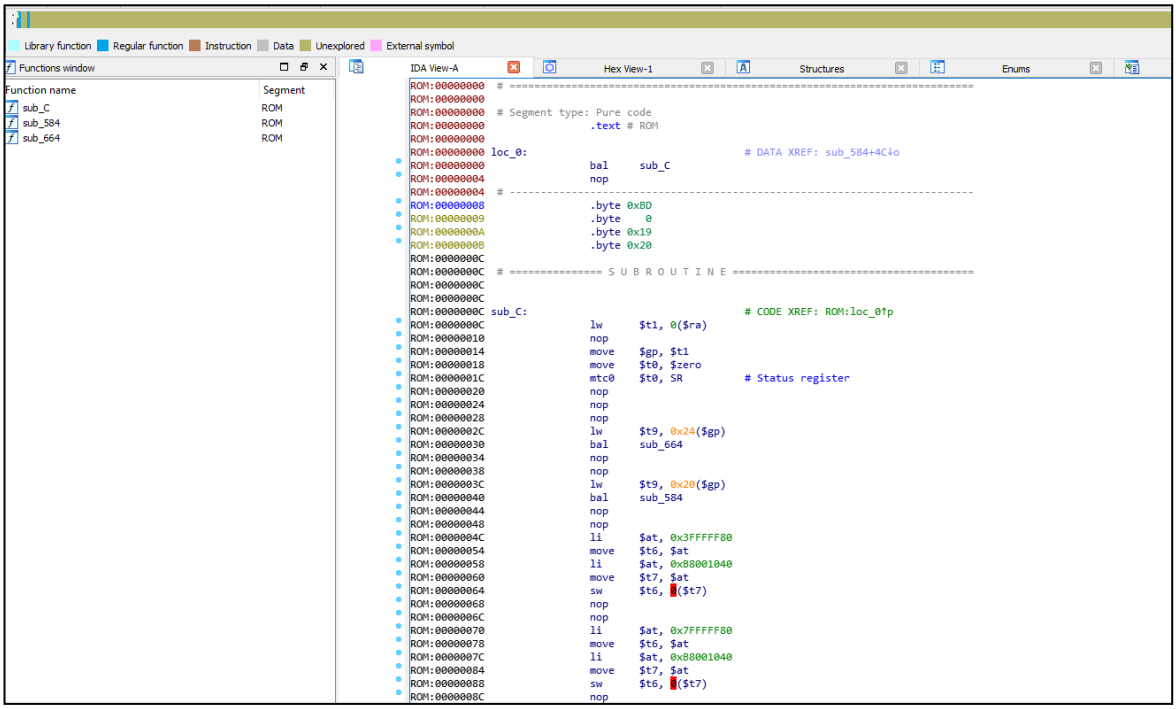
---

[9] Boot vector - הקוד שמתבצע כאשר המעבד נדלק(יוצא ממצב reset)
[10] XIP - execute in place היא טכנולוגיה שמאפשרת ביצוע של קוד ישירות מה-flash ללא העתקה ל-RAM

כמו כן, ה-first stage bootloader בשלבי הריצה הראשונים שלו נכתב באסמבלי טהור מה שעוד יותר יקשה על Ida. נפתח ב-Ida את ה-flash בהיסט 0:



ניתן לראות ש-Ida לא מזהה כלום, נתחיל "להכריח" את Ida לתרגם את החלקים הרלוונטים לקוד:



נראה ש-Ida מתחיל לזהות קצת, אך יש פה דוגמא למה שהזכרתי קודם - אנחנו לא יודעים אם הקוד באמת תקין. אם תשימו לב, בהיסט 8, בהתחלה יש data וזה כי Ida תירגם את החלק לקוד בהתחלה, אך ראיתי שבהיסט 12 נקרא הערך של כתובת זו בזיכרון לאוגר gp שמשומש לאורך כל התוכנית בשביל חישוב מיקום פונקציות וקריאת מידע מהחלקים שונים ב-first stage.

כעת, משאנחנו יודעים את הערך של האוגר gp נוכל לשים את הערך ב-Ida. וכמו כן נגדיר ל-Ida שהקובץ טעון מכתובת 0xBD000000 שלאחר קצת חיטוט בממשק הניהול של ה-second stage גיליתי שזה הכתובת הוירטואלית של ה-flash:

```
ROM:BD000000 loc_BD000000:                                   # DATA XREF: ROM:off_BD001928↓o
ROM:BD000000                 bal     sub_BD00000C
ROM:BD000004                 nop
ROM:BD000004  # ---------------------------------------------------------------------------
ROM:BD000008                 .byte 0xBD
ROM:BD000009                 .byte    0
ROM:BD00000A                 .byte 0x19
ROM:BD00000B                 .byte 0x20
ROM:BD00000C
ROM:BD00000C  # =============== S U B R O U T I N E =======================================
ROM:BD00000C
ROM:BD00000C
ROM:BD00000C sub_BD00000C:                                   # CODE XREF: ROM:loc_BD000000↑p
ROM:BD00000C                 lw      $t1, 0($ra)
ROM:BD000010                 nop
ROM:BD000014                 move    $gp, $t1
ROM:BD000018                 move    $t0, $zero
ROM:BD00001C                 mtc0    $t0, SR          # Status register
ROM:BD000020                 nop
ROM:BD000024                 nop
ROM:BD000028                 nop
ROM:BD00002C                 lw      $t9, off_BD001944
ROM:BD000030                 bal     sub_BD000664
ROM:BD000034                 nop
ROM:BD000038                 nop
ROM:BD00003C                 lw      $t9, off_BD001940
ROM:BD000040                 bal     sub_BD000584
ROM:BD000044                 nop
ROM:BD000048                 nop
ROM:BD00004C                 li      $at, 0x3FFFFF80
ROM:BD000054                 move    $t6, $at
ROM:BD000058                 li      $at, 0xB8001040
ROM:BD000060                 move    $t7, $at
ROM:BD000064                 sw      $t6, 0($t7)
ROM:BD000068                 nop
ROM:BD00006C                 nop
ROM:BD000070                 li      $at, 0x7FFFFF80
ROM:BD000078                 move    $t6, $at
ROM:BD00007C                 li      $at, 0xB8001040
ROM:BD000084                 move    $t7, $at
ROM:BD000088                 sw      $t6, 0($t7)
ROM:BD00008C                 nop
ROM:BD000090                 nop
```

כעת Ida יודעת לחשב את הערכים שנקראים על ידי gp, זה יוכל לעזור לנו רבות. נמשיך להנדס לאחור ו"להכריח" את Ida שחלקים מסוימים הם קוד כי עדיין היא לא זיהתה את רוב החלקים. אחת הפונקציות ש-Ida כן זיהה היא פונקציה שמדפיסה לסיריאל את המחרוזת "...Booting", על ידי כתיבה של המחרוזת ישירות לכתובת בזיכרון (כנראה שכתובת זו היא buffer השידור של הבקר הסיריאלי).

לאחר כל מיני תהליכים שה-first stage עושה (כגון העתקת 47KB של ה-flash ל-ram, מעבר לביצוע ב-ram, ומעבר לקוד שנראה שקומפל מקוד C) אנחנו מגיעים לחלק מעניין:

```
sub_BD000820:

var_10= -0x10
var_4= -4

li      $gp, 0x1100        # Load Immediate
addu    $gp, $t9           # Add Unsigned
addiu   $sp, -0x20         # Add Immediate Unsigned
sw      $ra, 0x20+var_4($sp)  # Store Word
sw      $gp, 0x20+var_10($sp)  # Store Word
lw      $t9, off_BD00193C  # Load Word
addiu   $a1, $a0, 0xC      # Add Immediate Unsigned
jalr    $t9                # Jump And Link Register
lui     $a0, 0x81D0        # Load Upper Immediate
lw      $gp, 0x20+var_10($sp)  # Load Word
bnez    $v0, loc_BD000860  # Branch on Not Zero
lui     $t9, 0x81D0        # Load Upper Immediate
```

```
lw      $ra, 0x20+var_4($sp)  # Load Word
jr      $t9                # Jump Register
addiu   $sp, 0x20          # Add Immediate Unsigned
```

```
loc_BD000860:
b       loc_BD000860       # Branch Always
nop
 # End of function sub_BD000820
```

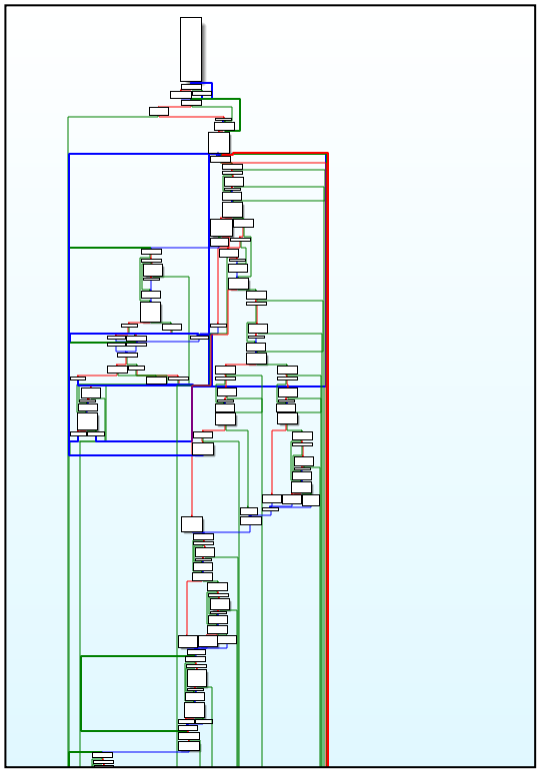
אנחנו רואים שיש פה קריאה לפונקציה עם פרמטר שהוא היסט לחלק של ה-flash שהועתק ל-ram. במידה והיא מצליחה - הקוד קופץ לכתובת הקבועה 0x81d00000, שעל פי מפת הכתובות הקבועה של MIPS זהו חלק מה-ram. כמו כן נשים לב שכתובת זו מועברת כפרמטר לפונקציה שנקראת לפני הקפיצה, לכן נסתכל בקצרה על חלקים רלוונטים בפונקציה זו:

```
lw      $t9, off_BD001948  # Load Word
sw      $a2, 0x60+var_C($sp)  # Store Word
move    $a0, $s0
addiu   $a1, $sp, 0x60+var_34  # Add Immediate Unsigned
jalr    $t9                # Jump And Link Register
sw      $a3, 0x60+var_10($sp)  # Store Word
lw      $gp, 0x60+var_48($sp)  # Load Word
lw      $a2, 0x60+var_C($sp)  # Load Word
lw      $a3, 0x60+var_10($sp)  # Load Word
bnez    $v0, loc_BD0018E4  # Branch on Not Zero
lui     $v0, 0x8354        # Load Upper Immediate
```

```
move    $a0, $s0
addiu   $a1, $sp, 0x60+var_34  # Add Immediate Unsigned
sw      $v0, 0x60+var_20($sp)  # Store Word
lw      $v0, off_BD001938  # Load Word
lw      $t9, off_BD001934  # Load Word
sw      $v0, 0x60+var_34($sp)  # Store Word
addiu   $v0, $sp, 0x60+var_40  # Add Immediate Unsigned
jalr    $t9                # Jump And Link Register
sw      $v0, 0x60+var_50($sp)  # Store Word
li      $v1, 0xFFFFFFFD  # Load Immediate
movz    $v1, $zero, $v0  # Move Conditional on Zero
b       loc_BD0018E8     # Branch Always
move    $v0, $v1
```

```
loc_BD0018E4:
li      $v0, 0xFFFFFFFD  # Load Immedia
```

הפונקציה קוראת לשתי הפונקציות שסימנתי בצהוב: נראה שהראשונה מאפסת חלק מסוים בזיכרון והשניה נראת מאוד מסובכת, אציג חלק מה-Proximity graph שלה כדי שתוכלו להבין:



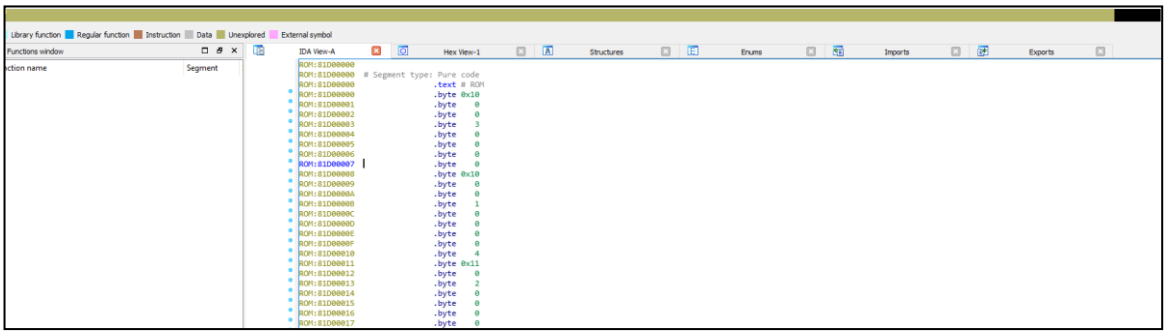
זה בערך רק חצי מה-proximity graph[11] שלה, ההחלטה שלי הייתה לא להתעכב עליה כי זה יקח המון זמן (כיוון שאנחנו יכולים לעשות רק אנליזה סטטית והנחתי שזו הפונקציה פתיחת הדחיסה של ה-second stage בגלל שאחד הפרמטרים שמועבר לה זה כתובת בחלק של ה-flash שהועתק ל-ram).

כעת יש לנו את הכתובת שאנחנו חושבים שמתחיל בו ב-second stage. איך נדע באיזה היסט זה מההתחלה? יש אופציה בממשק ניהול של ה-second stage להציג כתובות מסוימות בזיכרון לכן אזין פקודה שתציג את הכתובת הזו ואנסה לעשות התאמה של 20 הבתים הראשונים מול 20 בתים כלשהם ב-second stage הלא דחוס שכבר יש לנו:

```
Booting...


TBS bootloader V1.0 Build65129 for RTL8197D_DRACO_DB_AP(Oct 27 2014-17:28:17)

DRAM:  64 MB
Flash:  8 MB
kernel_offset=0x30000,rootfs_offset=0x12ca00
system supports single image and version is R2
out sysdata_get
------>>>arg_test:0x81d21240
is sysdata...
init ehternet...
IP: 192.168.1.1 MAC: 78:d9:9f:f5:4f:e3
IN: eth_init...
P0phymode=01, embedded phy
Hit Space or Enter key to stop autoboot:  0
out : abortboot
RTL8197D# md 81d00000
81d00000: 10000003 00000000 10000001 00000000    ................
81d00010: 04110002 00000000 81d19130 8fe90000    ...........0....
81d00020: 00000000 0120e021 3c088334 00000000    ..... .!<..4....
81d00030: 0100e821 00000000 8f880008 25080054    ...!........%..T
81d00040: 01000008 00000000 81d1a390 81d21240    ...............@
```

נראה שזהו היסט 0 בקובץ, יש לציין שניסיתי לפתוח את ה-second stage עוד בשלב שהבנתי שזה גירסא של U-boot אך בגלל חלק מהקשיים שפירטתי בהנדסה לאחור ב-first stage מופיעים ב-second stage לא יכלתי לדעת אם זה קוד תקין.

מצאנו את כתובת ההתחלה של ה-second stage. נפתח אותו ב-Ida:

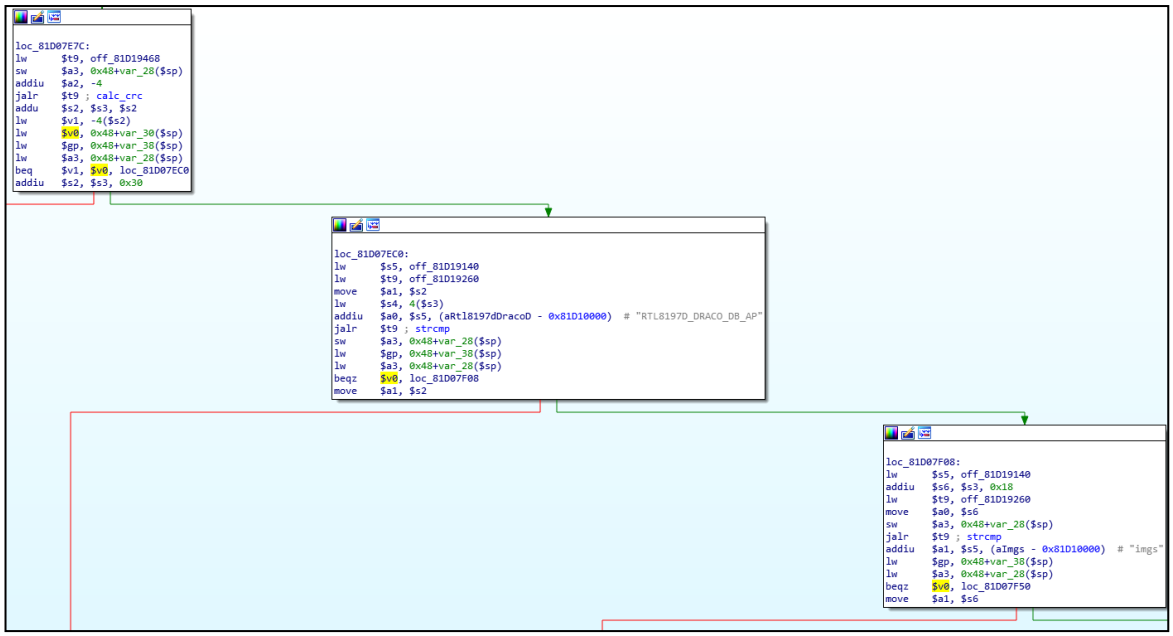


נראה שגם פה ida לא זיהתה כלום.

[11] Proximity graph - הדרך הצגה בה ida מייצגת חלקי קוד כריבועים ואת יחסי הגומלין בהם על ידי חצים

לאחר שנעשה את אותו תהליך שעשינו ב-first stage של "הכרחה" של Ida שחלקים מסוימים הם קוד, טעינה לכתובת שאותה מצאנו וקביעת האוגר gp נראה ש-Ida זיהתה את כל הקוד והצליחה למצוא התייחסויות להרבה מה-data segment.

התחלתי בתהליך הזה לזהות כל מיני פונקציות מוכרות כגון strcpy או strcmp כדי שתהליך ההנדסה האחורית יהיה קל יותר. נמצא בחלון ה-strings מחרוזת מעניינת עם הערך "System update completely! Restarting system" ויש פונקציה שאפילו משתמשת בה, ככל הנראה זו פונקציית השדרוג, אתייחס לחלקים הרלוונטים בה:



נראה שה-crc נבדק, נבדק שבהיסט 48 מתחילת הקובץ (זהו עדיין חלק מה-header של השדרוג) כתוב המחרוזת "RTL8197D_DRACO_DB_AP" ונבדק שבהיסט 24 מתחילת הקובץ כתוב המחרוזת "imgs".

לאחר בדיקות אלו נלקחים הערכים בהיסטים 0 ו-4 ב-header השדרוג, ונכתבים להיסטים 4 ו-8 בקונפיגורציה ב-flash וקובץ השדרוג נכתב ל-flash ללא שינוי התוכן.

יוריקה! זה החלק האחרון שהיה חסר לנו.

כיוון שהקוד שביצע את בדיקות השדרוג עד עכשיו לא בדק את חלק זה חלק (ועוד חלקים נוספים שלא אפרט כגון השם של הגירסא וכו'), לא יכלתי לדעת על קיומם של שדות אלה ב-header. כעת, כל מה שנשאר לעשות כדי לבדוק הוא להכין קובץ שדרוג עם קרנל שהגיע עם הרכיב ומערכת קבצים שהגיעה עם המכשיר אך עם שינוי - כדי שנוכל לראות אם הצלחנו.

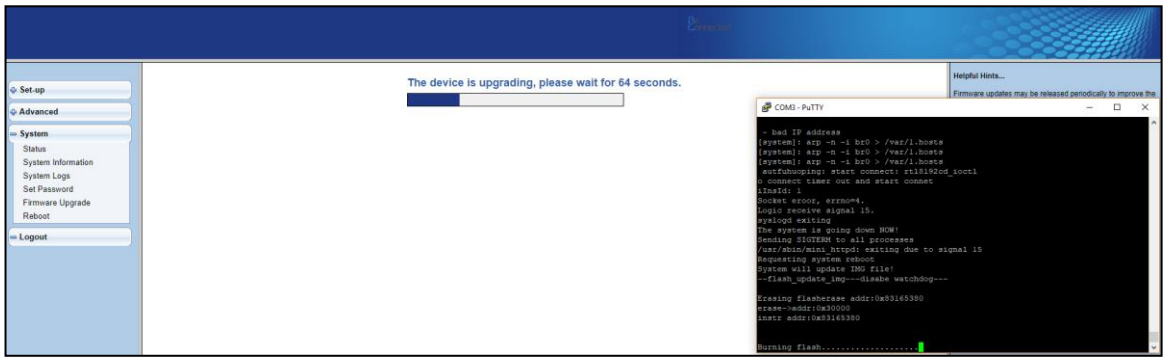
נפתח את מערכת הקבצים שיש לנו מהרכיב על ידי הכלי unsquashfs. נוסיף בתיקייה bin קובץ לבחירתנו (סתם קובץ בשם hacked או את הכלי netcat ונערוך את הקובץ /etc/rc.local אשר יריץ בכל הדלקה את netcat שיחכה לחיבורים בפורט 1337 ויעביר אותם לטרמינל). נכין ממערכת הקבצים ששינינו squashfs

חדש באותה גירסא ואותו אלגוריתם דחיסה (יש לנו את ערכים אלה מה-binwalk שביצענו dump ל-flash) על ידי הכלי mksquashfs ונראה שאפשר לצאת לדרך ☺

לסיכום, קובץ שדרוג תקין מסוג image מורכב מ:

1) Header שדרוג המכיל את הדברים הבאים:
   a) בהיסט 0 את ההיסט שבו ימצא הקרנל ב-flash
   b) בהיסט 4 את ההיסט בו תימצא מערכת הקבצים ב-flash
   c) בהיסט 20 את הגודל של הקרנל ומערכת הקבצים דחוסים
   d) בהיסט 24 את המחרוזת "imgs"
   e) בהיסט 32 את המחרוזת "DRACO_DB_AP"
   f) בהיסט 48 את המחרוזת "RTL8197D_DRACO_DB_AP"
2) kernel דחוס לפי אלגוריתם LZMA (כיוון שבשדרוג התוכן של הקובץ נכתב ללא שינוי)
3) מערכת קבצים מסוג squashfs מגירסא 4.0 דחוסה לפי אלגוריתם LZMA
4) 4 בתים אחרונים בקובץ הם crc של שאר הקובץ

ננסה לשדרג דרך ממשק ה-Web:



נראה שהמכשיר נכנס למצב שדרוג, נבדוק את האם הקובץ שהוספנו מופיע לאחר שהרכיב סיים את השדרוג ועלה הלינוקס (הפעם, לאור כל מה שגילינו על ה-header הלינוקס עלה באופן תקין):

```
#
# pwd
/bin
# ls -l hacked
-rw-r--r--    1 544      500             7 Dec  1  2017 hacked
#  autfuhuoping: start connect: rtl8192cd_ioctl
```

[בתמונה ניתן לראות שהוספנו קובץ בשם hacked אך מכאן ועד לבצע כמעט כל שנרצה בעת עליית מערכת ההפעלה - המרחק קצר]

נראה שאכן הצלחנו לשדרג firmware! כלומר - יש לנו שליטה מלאה על הרכיב כיוון שאנחנו יכולים להחליף את ה-frimware כולו כולל הוספת קבצים או החלפת קרנל או אפילו החלפת ה-bootloader.

מה היה אפשר לעשות אחרת?

הייתי רוצה לנצל את מאמר זה גם כדי לדבר על נושא הצג המגן: מה היצרן יכל לעשות אחרת לגבי כל הפריצות שמצאנו:

1. שירות ה-telnet שפתוח על הרכיב בברירת מחדל עם שם משתמש וסיסמא חלשים היה צריך להיסגר
2. בשרת ה-Web ניתן היה להוסיף בדיקות קלט לפרמטרים שמועברים בבקשות ה-HTTP, במידה ואכן היו בדיקות כאלה - הן היו מונעת את ה-LFI ואת ה-command injection שמצאנו בחלק הראשון.
3. תהליך השידרוג של הרכיב היה צריך לאמת שאכן השדרוג בא מהיצרן בעזרת חתימה דיגיטלית. הפירצה שבה שידרוג הרכיב לא מאומת ומאפשר לנו יכול להשתלט על הרכיב לחלוטין נחתם במספר ה-CVE הבא: CVE-2018-9232.

סיכום

הצלחנו להתגבר בחלק זה של המאמר על כל ההגבלות שהיו לנו בתחילת חלק זה. בסוף המחקר הגענו למצב בו אנו יכולים להשיג אחיזה מוחלטת על הרכיב ולהישאר ברכיב גם לאחר כיבוי. נוכל להשתמש במכשיר הזה (שלא מנותר ומאובטח כמו מחשבים ביתים או טלפונים) בהמשך להשתלטות על הקורבן וניסיון פריצה למכשירים אחרים כגון טלפון או מחשב.

בנוסף, מאוד נהנתי לבצע את החלק הזה במחקר ולמדתי המון בעיקר בתחום של הנדסה לאחור. אני מקווה שנהניתם מהקריאה לפחות כמו שאני נהנתי לבצע את חלק הזה של המחקר.

על המחבר

עומר כספי, מפתח Low level שבזמנו הפנוי מתעסק במחקר חולשות, בבדיקות חדירות והנדסה לאחור.

לכל שאלה, הערה / הארה, מוזמנים לפנות אליי בכתובת:

komerk0@gmail.com

תודות

תודה מיוחדת ל**חי מזרחי** שותפי לחלק הקודם של המחקר על ייעוץ, סיפוק החומרה, הערות ותיקונים לטיוטה זו של המאמר.

תודה ל**דימה נורטון**, **בן אגאי**, **ליאור שרון** ו**עידן נדב** שנתנו ייעוץ, עזרה בהתעסקות עם החומרה, הערות ותיקונים לטיוטה זו של המאמר.

ביבליוגרפיה ומקורות נוספים לקריאה

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

https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-9232