עוגיית ביש המזל

(איך למדתי להפסיק לחשוש ולהתחיל לאהוב מחקר קושחות)

מאת ליאור אופנהיים

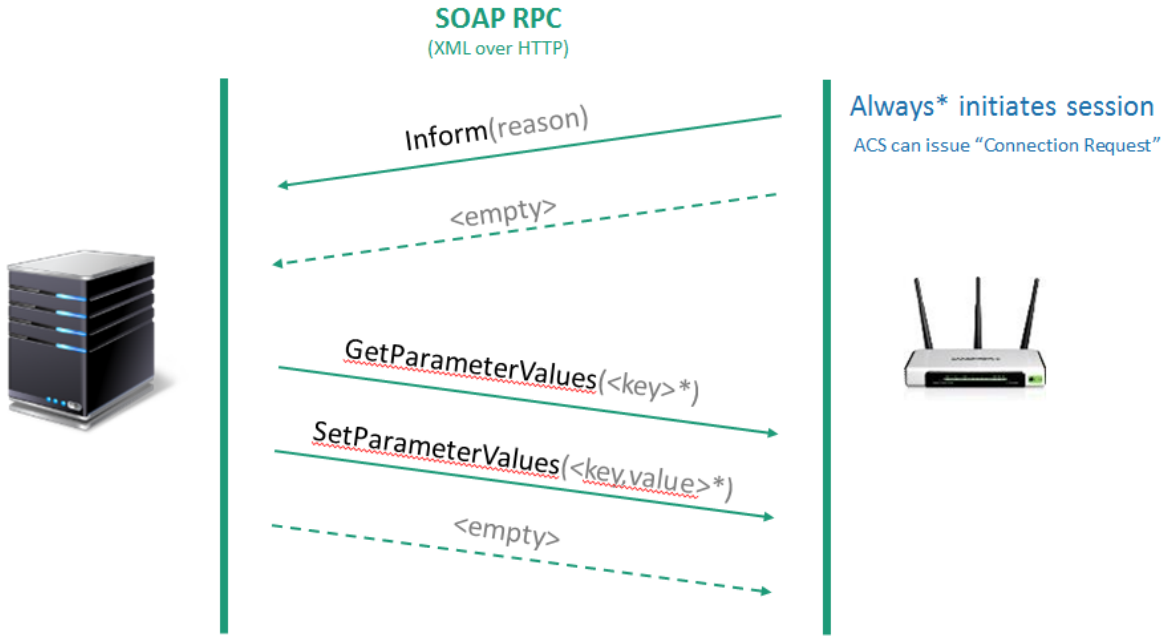
הקדמה

מאמר זה מתאר מחקר שערכנו בקבוצת ה- Malware & Vulnerability Research בצ'קפוינט. מחקר זה הוביל לחשיפת חולשת Misfortune Cookie שפורסמה בחודש שעבר. ניצול של חולשה זו מאפשר השתלטות מרחוק על מליוני ראוטרים בכל רחבי העולם. החולשה התגלתה כחלק מפרוייקט רחב יותר על בעיות אבטחה ב-TR-069. אי לכך, בחלקו הראשון של המאמר אסקור את פרוטוקול TR-069 בקצרה. אף על פי שהחולשה איננה שוכנת בפרוטוקול כשלעצמו, הבנה בסיסית של הפרוטוקול הכרחית להבנת העוצמה הגלומה בחולשה בפרט, ובמשטח התקיפה ככלל.

TR-069, או CWMP (CPE WAN Management Protocol), הוא פרוטוקול המאפשר לספקית תשתית או אינטרנט שליטה מרחוק בציוד קצה אשר מסופק על ידה ללקוחותיה, או בז'רגון המקצועי - CPE (Customer Premise Equipment). הוא שוחרר ב-2004 על ידי ה-Broadband Forum, שהוא ארגון המאגד כמה חברות תקשורת גדולות. כיום הוא בשימוש נרחב בקרב ספקיות אינטרנט, המשתמשות בו על מנת לשלוט בצורה נוחה בכל צי הראוטרים הביתיים שלהן. למרות שהפרוטוקול תוקנן אך לפני שנים מספר, כמעט כל ראוטר ביתי תומך בו, ולפי ההערכות, ישנם יותר ממאה מליון מכשירים שנשלטים מרחוק באמצעותו.

מבחינה טכנית, הפרוטוקול הוא למעשה בקשות SOAP RPC, קרי, XML מעל HTTP. הצד היוזם לשיחה הוא תמיד ה-CPE, אשר פותח session מול צד השרת שנקרא לפי התקן "ACS" (Auto Configuration Server), אשר נמצא ברשות הספקית. ה-CPE יפתח חיבור שכזה כל פרק זמן מוגדר, או לאחר שקרה אירוע מיוחד, לדוגמא כאשר המכשיר הופעל מחדש או כאשר בוצע שינוי קונפיגורציה. הסיבה לכך שהצד היוזם הוא תמיד הלקוח היא מטעמי אבטחה. כלומר, למנוע מצד שלישי לנסות ולהתחזות ל-ACS מול הלקוח. אם תהיתם איך ה-CPE יודע את הכתובת של ה-ACS, אז לרב הכתובת מוטמעת בו בזמן התקנת

הקושחה במעבדות ספקית התקשורת (בהנחה שהמכשיר סופק על ידי הספקית), אך אפשרי לקנפג על ידי בקשות DHCP מסויימות, וכמובן גם באופן ידני.



בתרשים שלהלן ניתן לראות שיחה סטנדרטית בין ה-CPE ל-ACS. ה-CPE שולח בקשת Inform לשרת בה הוא מציין את הסיבה לבקשה, ולאחר מכן בקשת HTTP ריקה, המצביעה על כך שלצד המסוים אין עוד מידע לשלוח. לאחר מכן, יכול ה-ACS להגיב בחזרה על הבקשה הריקה ולשלוח הוראות ל-CPE. הוראה יכולה להיות קריאה או כתיבה של פרמטרים של המכשיר, הורדה והעלאה של קבצים, ואפילו עדכון firmware מרחוק.

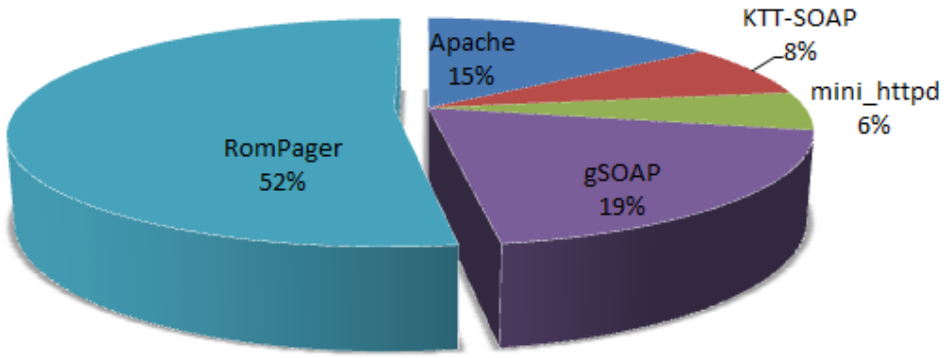
לעיתים, הספקית תרצה גישה מיידית למכשירי הקצה, לדוגמא במקרה ולקוח מתקשר לתמיכה הטכנית של הספקית. במקרה שכזה, הספקית תרצה ליזום session באופן מיידי מול המכשיר שלי, ולראות מה מקור התקלה, במקום לחכות עד לפעם הבאה שהמכשיר ייזום שיחה. עבור תרחישים אלו נוצר מנגנון בפרוטוקול נוסף הנקרא Connection Request. מנגנון זה מאפשר לספקית לשלוח בקשת HTTP ספציפית ל-CPE (ל-URL ספציפי + שם משתמש וסיסמא). במידה והבקשה מאומתת על ידי הלקוח, הוא ייזום באופן מיידי שיחה אל מול ה-ACS המוגדר לו (ולא למי ששלח את הבקשה!). מנגנון זה "מאלץ" למעשה את הלקוח להפעיל שרת HTTP על המכשיר שלו, ולהאזין לפורט ב-WAN באופן קבוע, המחכה ל- Connection Requests. לפי הפרוטוקול, מספר הפורט המומלץ הוא 7547, אך הדבר ניתן לשינוי על ידי הספקית.

זהו, עד כאן בענייני TR-069. התכלית של חלק זה הייתה להסביר מדוע ישנם ראטורים בעולם אשר מאזינים ל-WAN בפורט 7574 או בפורט אחר. השאלה המתבקשת כעת היא כמה ראוטרים באמת מקשיבים בפורט הזה, ואיזה מין שרתי HTTP עומדים מאחוריו.

בכדי לבצע סריקת אינטרנט רחבה השתמשנו בכלי קוד פתוח הנקרא ZMAP. על פי החוקרים שכתבו את הכלי, ZMAP מאפשר לסרוק את כל האינטרנט ב-42 דקות בפורט ספציפי בהינתן חיבור אינטרנט של 1 ג'יגה ביט לשנייה. עם חיבור אינטרנט רגיל, הצלחנו ,תוך ימים ספורים, לבדוק את הזמינות בפורט 7547 של כל ה-IP-ים ב-IPv4. מסתבר שישנם 46,063,733 IP-ים שמגיבים בפורט הזה (או 1.18% מהאינטרנט). אם זה נשמע לכם מעט, דעו כי הנתונים מצביעים על כך שמדובר בפורט השני הכי פופולארי בעולם - אחרי HTTP (פורט 80) עם 1.77% מהאינטרנט. שוב אזכיר שמדובר רק בפורט הדיפולטי, וספקיות אחרות מקנפגות במכשירים שלהן פורט אחר (fun fact: הפורט Connection Requestשל בזק הוא 30005).

אחרי שנקשנו על כל דלתות ה-IPv4 בעולם, דגמנו חלק מה-IP-ים שהגיבו בחזרה בפורט 7547 באופן רנדומלי, ושלחנו להם בקשת HTTP סטנדרטית. המטרה היא לעשות fingerprinting לשרתי HTTP שמגיבים לנו באמצעות שדה ה-"Server" ב-HTTP headers.
להלן התוצאות.

מי אתה ROMPAGER?



כפי שאתם רואים, יותר מחצי מהשרתים המזוהים (וקצת יותר מרבע מכלל ה-IP-ים) מזוהים כשרתי RomPager. אם אתם לא יודעים מה זה RomPager, אתם לא לבד, גם לעבדכם הנאמן לא היה שמץ של מושג. מסתבר שמדובר ב-embedded HTTP server, כלומר שרת המיועד לשימוש במערכות מחשב ייעודיות (שהן לרב חלשות יותר ממחשב רגיל). RomPager שוחרר לראשונה ב-1996, ופעיל עד היום (גרסא אחרונה - 5.4).

מה שלא סיפרתי לכם זה ש-98% משרתי ה-RomPager החזירו את המחרוזת הזו - "RomPager 4.07/UPnP 1.0", ובמילים אחרות הרב המכריע של שרתי ה-RomPager מריצים בדיוק את אותה הגרסא. כדי להוסיף חטא על פשע, בדיקה מול החברה המייצרת את RomPager העלתה כי הגרסא הנ"ל שוחררה ב-2002. בכדי לסבר את האוזן, מדובר ב-11.3 מליון מכשירים שמריצים את הגרסא הזו בפורט 7547 המאזינים לכלל האינטרנט... בבדיקה מקבילה על פורט 80 נמצאו מעל 2 מליון מכשירים. הסיבה לכך היא כמובן שפתיחת פורט 80 ל-WAN הינה פרצת אבטחה ידועה, ולכן הרבה לקוחות פרטיים וספקיות יודעים כבר לחסום אותה, בעוד שפתיחת פורט 7547 היא הכרחית למימוש הפרוטוקול CWMP.

נשאלת השאלה "מי אלו המכשירים שמריצים את הגרסא הארכאית הזו?". כדי לבדוק את זה, שוב שלחנו בקשות HTTP, אך הפעם בפורט 80, למדגם של ה-IPים שהגיבו לנו. בדרך כלל, פנייה ל-"/" בשרת RomPager מחזירה תשובת 401 Authorization Required כאשר בשדה ה-realm ב-header מופיע שם הראוטר (שוב, בד"כ), דבר המאפשר לנו זיהוי די ודאי של דגם. בסריקה נצפו למעלה מ-200 דגמים שונים מעשרות יצרניות ראוטרים כגון: TP-LINK, D-Link, Huawei, ZTE, Edimax, ZyXEL...

בהמשך המאמר נעלה השערה מה גרם לגרסא הכה ספציפית כזו להגיע לכל כך הרבה דגמים.

מחקר RomPager 4.07:

לצורך המחקר קניתי ראוטר של TP-LINK מדגם W8961ND בחנות המחשבים הקרובה לביתי. במקביל, הורדתי את ה-firmware המתאים למודל זה, אשר מכיל קובץ אחד בשם ras. הקובץ לא נראה כפורמט מוכר, ורב המידע בו היה פסאודו-אקראי, מה שמצביע על דחיסה או הצפנה של המידע.

עצה #1 - כשאתם מקבלים firmware לא מזוהה, ראשית בדקו אותו ב-Binwalk.
הרצה ב-Binwalk (כלי שימושי לניתוח קבצים בינארים) זיהתה את ה-firmware והצליחה לחלץ ממנו שני קבצים בינארים ושתי תמונות. אחד מהקבצים הבינארים התברר בדיעבד כ-bootloader - הקוד האחראי לאתחול של הראוטר. הקובץ בינארי השני הכיל את החלק הארי של ה-firmware (שהיה דחוס). התמונות היו פשוט resource-ים שנועדו לממשק ניהול (לוגואים למינהם). הפורמט של הקבצים הבינאריים היה מקושר, לפי Binwalk, ל-ZynOS. מדובר במערכת הפעלה למכשירי embedded שפותחה על ידי ZyXEL (אותה ZyXEL שמופיע ברשימת היצרניות שמשתמשות בגרסא המדוברת של RomPager). אם כן, ככל הנראה המערכת הפעלה של הראוטר שלנו היא ZynOS.

```
DECIMAL       HEX           DESCRIPTION
--------------------------------------------------------------------------------
84992         0x14C00       ZynOS header, header size: 48 bytes, rom image type: ROMBIN, uncompressed size: 66696, compress
ags: 0xE0, uncompressed checksum is valid, the binary is compressed, compressed checksum is valid, memory map table address: 0x
85043         0x14C33       LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed size: 6669
128002        0x1F402       GIF image data, version 8"9a", 200 x 50
136194        0x21402       GIF image data, version 8"9a", 560 x 50
350208        0x55800       ZynOS header, header size: 48 bytes, rom image type: ROMBIN, uncompressed size: 5068696, compre
, flags: 0xE0, uncompressed checksum is valid, the binary is compressed, compressed checksum is valid, memory map table address
350259        0x55833       LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed size: 5068
```

עוד קוריוז אחרון לפני שנתחיל להנדס לאחור את הקושחה. מסתבר שלא רק הראוטר שלנו מריץ ZynOS, אלא כל אחד ואחד מהראוטרים שמצאנו שמכילים את RomPager 4.07. בשלב הזה אנחנו עדיין לא יודעים מה החוט שמקשר בין כל הנתונים האלו.

ראשית, נפתח את הקובץ הבינארי המרכזי שלנו ב-IDA. מכיוון שהקובץ הוא רק בינארי (ה-ZynOS header קולף ע"י Binwalk), IDA לא מצליחה לזהות את הפורמט, וצריך להכריח אותה לפרסר את הקובץ כ-mipsb32 (הארכיטקטורה נמצאה ע"י חיפוש הספסיפיקציות של הראוטר באינטרנט). נראה שאנחנו מקבלים opcode-ים הגיוניים שמצביעים שאנחנו בכיוון הנכון, אבל IDA עדיין לא מצליחה לבצע ניתוח מלא של הקובץ. הסיבה לכך היא שאנחנו לא הזנו את ה-base address הנכון של הקושחה, ולכן IDA לא מצליחה לחבר בין פונקציות שונות ובין קוד ל-data. למרות שה-ZynOS header אמור להכיל את כתובת הבסיס עבור כל קובץ בינארי באחד השדות, במקרה שלנו השדה הוא 0. מציאת כתובת הבסיס נעשתה לבסוף על ידי התאמה בין מצביעים אבסולטיים בקוד, בין הזכרון המתאים עבורם.

עכשיו אפשר להתחיל לעבוד. מכיוון שלא היו לי כלל סימבולים החלטתי להתחיל להבין את הפונקציות הכי מקושרות בחומרה (כלומר, אלו שנקראות מהכי הרבה נקודות שונות בקוד). התכלית לכך היא יצירת בסיס נוח לעבוד מעליו כאשר מנסים להבין פונקציות בעלות היקף רחב יותר. באמצעות הטכניקה הזו גיליתי פונקציות libc רבות הקשורות למחרוזות (strcpy, strcat..) ולזכרון (memmove, memcopy).

עצה #2 - אל תנסו לעשות רברסינג ל-libc בעצמכם. תשתמשו ב-binary diffing מול libc מקומפל לארכיטקטורה הרלוונטית שיעשה עבורכם את העבודה. מכיוון ש-RomPager מקומפל סטטית עם כל מערכת ההפעלה, קשה להפריד בין קוד של השרת לבין שאר הקוד של הראוטר.

נקודת האחיזה הייתה מחרוזות אינדיקטיביות שקושרו לפונקציונאליות של השרת. באמצעותן הגעתי לפונקצייה מעניינת במיוחד שאחראית לאתחול מערך של struct-ים מהסוג הזה:

```
Struct HttpHeader
{
      Void * funcPtr
      char * headerName
      int size
}
```

ה-struct-ים האלו אחראים לקריאה לפונקציית הפרסור הרלוונטית לכל header של HTTP. מכיוון שהפרסור נעשה בשלב הראשוני ביותר של הבקשה, פונקציות הפרסור האלו מהוות משטח תקיפה מצוין לבדיקה.

רב הפונקציות משתמשות ב-strncpy בכדי לבצע העתקה של המידע לתוך מבנה הזכרון שמאכסן את הבקשה. באמצעות שימוש בפונקציה זו, ניתן לוודא שהקלט לא יחרוג מהבאפר שהוקצה לו. אולם מסיבה לא ידועה, כל הפונקציות שמפרסרות sub-headers של digest authorization (דרך התאמתות מול שרתים מעל HTTP) משתמשות דווקא ב-strcpy ואינן בודקות את אורך הקלט. כלומר, אנחנו יכולים להכניס לתוך שדות אלו ערכים הארוכים כאוות נפשנו, ולחרוג מגבולות הבאפר שהוקצה לנו (למעשה, יש מגבלה אחרת, שחוסמת מלמעלה את האורך של כל שורה ב-HTTP headers, מה שמאפשר לנו לכתוב עד 600 בתים בערך).

מכיוון שאנחנו עדיין לא ממש יודעים איך נראים המבנים של הבקשות כפי שמאוחסנות בזכרון, ננסה לשלוח "על עיוור" בקשות עם שדות digest ארוכים במיוחד ונראה אם אנחנו מצליחים לגרום להתנהגות לא נורמאלית. לבסוף, הגיע ההקרסה המיוחלת עבור שדה ה-username כאשר אורך הקלט גדול מ-581 בתים. ניסיתי לחפש את ההיסט הרלוונטי במבנה הבקשה בתוך הקוד, אך לא מצאתי אף התאמה.

עצה #3 - השקיעו בהרמת סביבת דיבוג דינאמי:

בשלב זה הבנתי שהניתוח הסטטי לא מספיק לצרכי המחקר, ואני חייב להשיג יכולת ניתוח דינאמי של המכשיר. אפשרות אחת היא לנסות לטעון את הקושחה לתוך qemu ולבצע אמולציה של המכשיר. אם הדבר אפשרי, במיוחד אם מדובר במכשיר מבוסס linux, זו האופציה המומלצת, לפי דעתי. במקרה שלי, בגלל הייחודיות של מערכת ההפעלה, לא הצלחתי לבצע את האמולציה. בצר לי, החלטתי לנסות להתחבר למכשיר באופן פיזי על ידי ממשק JTAG. למי שלא בקיא ברזי הנדסת מעגלים אלקטרוניים, מדובר בפורט פיזי על גבי המכשיר שמאפשר לנו לבצע דיבוג חיצוני של המכשיר (במקור הפרוטוקול נועד כדי לבצע בדיקות איכות למעגלים בצורה יעילה).

אף על פי שהחיפושים אחר חיבור JTAG עלו בתוהו, נמצא על גבי המכשיר חיבור סיריאלי המתקשר מעל פרוטוקול U-ART, שהוא פרוטוקול הרבה יותר פשוט מ-JTAG, המאפשר תקשורת סיריאלית גנרית מול המעגל. התוכן של התקשורת תלוי במימוש הספציפי של מערכת ההפעלה. בכדי לחבר את הפורט הסיריאלי למחשב, יש צורך להשתמש במכשיר נוסף שיבצע תרגום של האות מהמעגל לממשק USB (לא חובה, אבל כנראה מפשט את ההתממשקות). אני השתמשתי ב-BusPirate לצורך העניין.

הממשק הסריאלי של RomPager מאפשר לנו שלוש פונקציות מרכזיות:

- הדפסת debug strings, כולל dump בעת קריסות
- ממשק פקודות שנתמך על ידי ה-bootloader לפני עליית הבינארי הראשי. הממשק מאפשר לקרוא ולכתוב לזכרון
- ממשק telnet

הניצול של ממשק הפקודות של ה-bootloader לצורך עריכה של הבינארי הראשי נעשתה בעזרת הפוסט הזה.

עכשיו נוכל להקריס את המכשיר באמצעות החולשה שמצאנו ולראות מה מתקבל בפלט מהפורט הסיראלי.

```
TLB refill exception occured!
EPC= 0x61616161
SR= 0x10000003
CR= 0x50801808
$RA= 0x00000000
Bad Virtual Address = 0x61616160
UTLB_TLBL ..\core\sys_isr.c:267 sysreset()


        $r0= 0x00000000 $at= 0x80350000 $v0= 0x00000000 $v1= 0x00000001
        $a0= 0x00000001 $a1= 0x805D7AF8 $a2= 0xFFFFFFFF $a3= 0x00000000
        $t0= 0x8001FF80 $t1= 0xFFFFFFFE $t2= 0x804A8F38 $t3= 0x804A9E47
        $t4= 0x804A9460 $t5= 0x804A8A60 $t6= 0x804A9D00 $t7= 0x00000040
        $s0= 0x804A8A60 $s1= 0x8040C114 $s2= 0x805E2BF8 $s3= 0x80042A70
        $s4= 0x00000001 $s5= 0x8000007C $s6= 0x8040E5FC $s7= 0x00000000
        $t8= 0x804A9E48 $t9= 0x00000000 $k0= 0x61616160 $k1= 0x8000007C
        $gp= 0x8040F004 $sp= 0x805E2B90 $fp= 0x805E2BF8 $ra= 0x8003A3D0


          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

805e2bf8: 80 5e 2c 28 80 04 2a 70 80 40 f8 ac 80 40 f3 e0     .^,(..*p.@...@..
805e2c08: 80 40 e5 fc 00 00 00 00 80 40 e6 0c 80 48 4e 29     .@.......@...HN)
805e2c18: 00 55 54 4c 42 5f 54 4c 42 4c 00 ac 00 00 00 00     .UTLB_TLBL......
805e2c28: 80 5e 2c 40 80 10 16 d0 80 40 f3 e0 00 00 00 00     .^,@.....@......
805e2c38: 80 40 f8 ac 00 00 00 00 80 5e 2c 58 80 10 1a 00     .@.......^,X....
```

כפי שניתן לראות בתמונה לעיל, מסתבר שההקרסה נובעת מכך שהדריסה אפשרה לנו להשתלט על ה-instruction pointer של המערכת (ב-mips-ית - EPC). ניתוח של ה-stack dump ומעט עריכות זכרון לצורך הדפסת לוגים אינפורמטיבים (למעשה, שימוש ב-API של הפורט הסיריאלי כדי להדפיס ערכים של רגיסטרים ואזורי זכרון) אפשרה לזהות את מקור החולשה, שנבע מקריאה ל-callback שנעשית לאחר

סיום פרסור הבקשה. משום מה, הפונקציה שקוראת ל-callback לא מקושרת לאף פונקציה אחרת באופן ישיר, ולכן IDA לא הצליח לזהות אותה, מה שמסביר מדוע לא מצאתי את ה-offset בקוד.

עצה #4 - תשקיעו הרבה ב-code exploration מלא לפרוייקט שלכם, כלומר זהו את כל אזורי הקוד והפונקציות בקובץ הבינארי. בניגוד ל-elf או exe, בפורמטים בינארים לא מוכרים IDA מתקשה בניתוח אוטומאטי של הבינארי, ולעיתים לא תזהה את כל אזורי הקוד (או תזהה אזורי מידע כאזורי קוד בטעות).

בכל המקרים שיצא לי לבדוק, הערך הרגיל של ה-callback הוא 0, ולכן הוא כלל לא מורץ. אינני יודע מה המשמעות שלו.

במבט ראשון, נראה שהגשמנו את חלומו הרטוב של כל מנצל חולשה. שליטה ב-instruction pointer הינה כפסע מהרצת קוד על המערכת. כל שנותר לנו הוא לגלות מקום בזכרון שבשליטתנו, לשים שם את ה-shellcode שלנו, ולהשתמש בחולשה על מנת "לקפוץ" לשם. אלא שכאן אנו נתקלים בבעיה משמעותית. בהסתכלות על מגוון רחב של קושחות של ראוטרים המריצים את RomPager בגרסא 4.07 אנחנו רואים דמיון רב. אך לצערנו, מכיוון שכל קושחה לכל מודל קומפלה בנפרד, מבנה הזכרון שונה ממודל למודל (ואף בין גרסאות שונות של אותו המודל), כך שאין לנו שום נקודת אחיזה גנרית המשותפת לכלל הקושחות. כמובן שעבור דגם וגרסת קושחה ספציפיים - הבעיה פתורה. דרך נוספת לפתור את הבעיה היא למצוא חולשה נוספת אשר תסגיר לנו את מבנה הזכרון של הראוטר, ובאמצעותה ניתן יהיה לכתוב השמשה גנרית לחולשה. בשלב זה החלטתי להפסיק את נסיונות ההשמשה, ולהמשיך לחפש בכיוונים אחרים.

בקצרה אזכיר שבמהלך המחקר נמצאה חולשה נוספה שבאמצעותה ניתן להשתלט על ה-EPC, הנובעת מניצול הקלט בפונקיות נוספות האחראיות לפרסור sub-headers נוספים ב-digest authorization (כי כולן משתמשות ב-strcpy), באמצעות שליחה של מספר בקשות HTTP מסויימות ברצף. אולם, גם מפני שאנחנו מגיעים למצב דומה (יחסית) מבחינת השמשה, וגם מפני שהחולשה הזו עובדת רק בפורט 80, לא ארחיב עליה במאמר זה.

עוגית ביש המזל:

הממצא המעניין ביותר במחקר זה הוא חולשת Misfortune Cookie, שקשורה לפונקצייה האחראית לפרסור העוגיות בבקשת ה-HTTP. מכיוון שהחולשה לא נגרמת כתוצאה מהעתקת זכרון ללא בדיקה אורך, דילגתי בתחילה על התעמקות במימוש שלה, אך כפי שנראה תכף, הוא טומן בחובו כמה בעיות קריטיות.

מכיוון שRomPager הינה מערכת המותאמת למערכות embedded, אין בה הקצאות זכרון דינאמיות, ולכן העוגיות נשמרות באופן סטטי במערך של עוגיות (בגודל 10) בתוך המבנה המאכסן את הבקשה. בנוסף, שמות העוגיות הם בפורמט קבוע: C0,C1…C9, כאשר העוגייה בשם C0, תשמר במקום הראשון, C1 - בשני, וכן הלאה עד C9. הגודל המוקצה לכל עוגיה הוא 40 בתים.

מי שיודע קצת mips מוזמן לבחון את קטע הקוד הבא ולנסות למצוא את הבעיה בכוחות עצמו (להתרכז ב-branch השמאלי של הקוד):

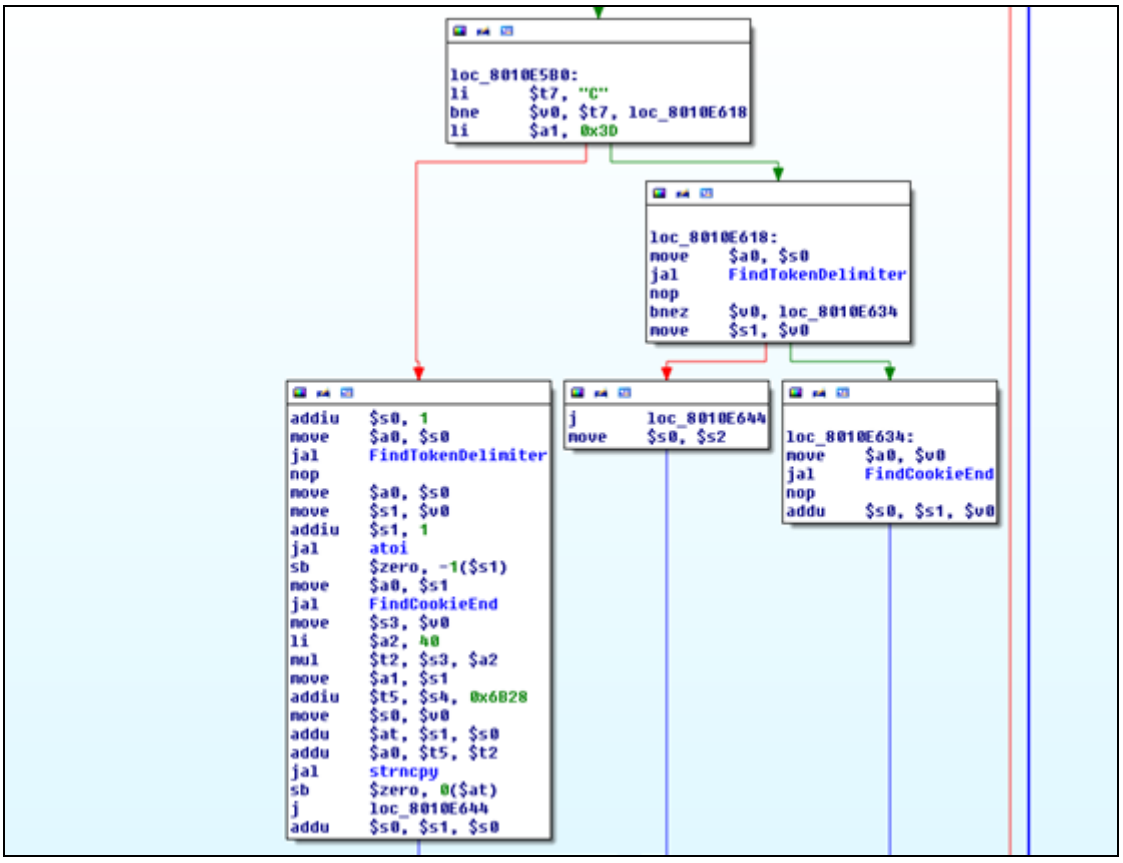
s0 - מצביע לשדה ה-Cookie ב-HTTP header (לדוגמא "C3=abcd\r\n")

v0 - מכיל את הבית הראשון של שם העוגיה

s4 - this (מצביע למבנה הבקשה, אשר מכיל את מערך העוגיות בהיסט 0x6b28)

FindTokenDelimiter מקבל מצביע למחרוזת ומחזיר מצביע למיקום הראשון של "=" במחרוזת

FindCookieEnd - מקבל מצביע למחרוזת ומחזיר מצביע למיקום הראשון של ירידת שורה



לאלו שלא מתמצאים ב-mips, המרתי את הפונקציה לקוד C קצר (ומעט מופשט):

```
void ParseCookie(Request * request, char * name, char * value)
{
    if (name[0] == 'C')
    {
        int index = atoi(name+1);
        strncpy(request->CookieArray[index],value,40);
    }
}
```

הבעיה טמונה בכך שבשום שלב לא נבדק האם העוגיה חורגת מגבול מערך העוגיות. כך שלמעשה, הכנסת העוגיה C1234 בבקשת ה-HTTP תאפשר לנו לכתוב לזכרון של הראוטר במיקום:

```
this->CookieArray + (1234* 40)
```

בצורה זו אנחנו יכולים לכתוב לכל הזכרון שנמצא אחרי מערך העוגיות. בנוסף, הכנסת עוגיה "שלילית" (לדוגמא "C-300") תאפשר לנו לכתוב גם באזורי הזכרון שנמצאים לפני המערך. במילים אחרות, החולשה מאפשרת כתיבה לכל הזכרון של הראוטר.

היתרון המשמעותי של החולשה הזו על פני החולשות שהוזכרו לעיל הוא שהכתיבה היא יחסית למיקום של מערך העוגיות, כך שבאופן מובנה, אנחנו מתגברים על בעיית מבני הזכרון השונים בקרב מודלים שונים, שנתקלנו בה בעת ניסיון השמשת החולשה הקודמת. כלומר, אם ננסה לדרוס שדה אחר בתוך מבנה הבקשה של RomPager, נשתמש באותו "index עוגייה" בדיוק עבור כל דגמי הראוטרים שמריצים את RomPager 4.07.

מה יכולת כתיבה כזו מאפשרת? ובכן, מחקר נוסף של מבני הזכרון של RomPager, אפשר לי ליצור בקשה הכוללת עוגיות אשר תשכתבנה ערכים מסויימים, שבסופו של דבר אפשרה לי להכנס ל-admin panel של השרת מכל פורט שפתוח (כולל 7547). מכיוון שבשלב זה, רב הראוטרים הפגיעים עדיין לא עודכנו עם קושחה אשר סוגרת את הפרצה, החלטנו שלא לחשוף את פרטי הניצול של החולשה.

סיכום

נחזור לאחת משאלות המחקר, מדוע כל כך הרבה דגמים משתמשים באותה גרסא של RomPager? מסתבר כי התשובה לכך טמונה בחומרה. כל הדגמים הללו משתמשים ב-chipset-ים של אותה החברה - Trendchip (כיום MediaTek). Trendchip קנתה מ-Allegro רשיון ל-RomPager ב-2002, ורשיון ל-ZynOS מ-Zyxel, ושילבה אותם ב-SDK אשר צורף ל-chipset. כך שלמרות שכל יצרנית שינתה במקצת את הקושחה, בעיקר לצרכים קוסמטיים, מתחת למכסה המנוע כל הראוטרים השתמשו באותה מערכת הפעלה.

חשוב לציין כי אמנם החולשה נמצאה ב-RomPager, אך AllegroSoft אינה האשמה במצב זה. החולשה עצמה תוקנה כבר ב-2005 (החל מגרסה 4.34) כחלק משדרוג חבילת הקוד, בלא ידיעה על ההשלכות של ניצול חולשה זו. הדבר אשר מנע את פעפוע העדכון לראוטרים הנמכרים כיום הינו שרשרת ארוכה וסבוכה הכרוכה בייצור הראוטרים אשר מתחילה ביצרנית ה-chipset-ים, עוברת דרך יצרנית הראוטרים (TP-Link, Dlink ודומיהן), ונגמרת בספקיות האינטרנט אשר בסופו של דבר מוכרות את הראוטר ללקוחותיהן. מכיוון שכל השרשרת הזו כרוכה בהסכמים שחלקם כבר פגו, ולא קיימת אחריות ריכוזית על המוצר הסופי בידי גורם אחד, אין זה מפליא כי גם עשור לאחר שתוקנה החולשה, עדיין כל המכשירים המיוצרים כיום באופן זה עודם פגיעים.

גם אם נדמה שפרצות אבטחה בראוטרים ביתיים אינן מסוכנות כמו פירצות המאפשרות גישה למחשב, קמפיינים מאסיבים שתקפו מאות אלפי ראוטרים הוכיחו כי ביכולת לשלוט בצי של ראוטרים טמון כוח עצום: תקיפות DDOS, שבירת מודל האבטחה של ה-NAT והתפשטות למחשבים בתוך ה-LAN. מכיוון שעולם הראוטרים הביתיים עדיין איננו מוכוון אבטחה כפי שהיינו רוצים שיהיה, החולשה הזו לא תעלם בזמן הקרוב. התקווה היא שפרסום חולשה זו ואחרות יפעיל לחץ על הספקיות ויצרניות הראוטרים לעבור למודלי אבטחה מודרניים יותר, שימנעו מתופעות שכאלו להשנות. פרטים נוספים ורשימת הראוטרים הפגיעים נמצאים באתר הזה.

אחרית דבר ו\או עצה #5:

בשלב כלשהו במחקר מצאתי גרסא חדשה יותר של RomPager (4.34), כחלק מקושחה מבוססת לינוקס. היתרון בגרסא זו היא שהיא הייתה מקומפלת כקובץ elf, שכלל private symbols. במילים אחרות, הקובץ הכיל שמות של חלק גדול מהפונקציות כפי שהופיע בקוד המקור. ביצוע binary diffing, מול הגרסא שחקרתי אפשר לי לייבא הרבה מהסימבולים לפרוייקט שלי, משום שחלק לא קטן מהפונקציונאליות הכללית של RomPager לא השתנה במעבר בין הגרסאות. בדיעבד, גילוי מוקדם של הגרסא הזו היה חוסך לי הרבה זמן של פענוח פונקציונאליות סבוכות של RomPager.

על הכותב

הכותב הינו חוקר אבטחה בקבוצת ה-Malware & Vulnerability של צ'קפוינט. לשאלת, טענות, השגות וביקורות בונות או הורסות - lioro@checkpoint.com

תודות

תודה לשחר טל, ראש קבוצת Malware & Vulnerability בצ'קפוינט, שהיה שותף במחקר זה.