Digital Whisper

גליון 45, אוקטובר 2013

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	שילה ספרה מלר, ניר אדר, אפיק קסטיאל
כתבים:	יניב מרקס, אפיק קסטיאל (cp77fk4r), ניר גלאון, רונן שוסטין (Antartic), פלג הדר (P), עו"ד יהונתן קלינגר ורועי חי.

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

דבר העורכים

ברוכים הבאים לגיליון ה-45 של Digital Whisper!

אחרי הפסקה של חודש (בטח לא שמתם לב שחודש שעבר לא הוצאנו גיליון, אה? ☺), הנה אנחנו כאן עם הגליון החדש.

כמו תמיד היינו רוצים להגיד תודה רבה לכל מי שהתאמץ, השקיע, נתן מהזמן שלו (גם החודש הזה וגם חודש שעבר!), ובזכותו המגזין שלנו ממשיך להתפרסם ולפרוח: תודה רבה ל**יניב מרקס**! תודה רבה ל**ניר גלאון**! תודה רבה ל**רונן שוסטין (Antartic)**! תודה רבה ל**פלג הדר (P)**! תודה רבה ל**עו"ד יהונתן קלינגר**! ותודה רבה ל**רועי חי**!

וכמובן - תודה רבה ל**שילה ספרה מילר**, העורכת שלנו, על כל העזרה עם עריכת המגזין!

בנוסף, היינו רוצה להגיד תודה רבה ל**שחר מאור גייגר** על מאמר מעולה שכתב, ובסופו של דבר לא התפרסם, שלא באשמתו.

קריאה מהנה!

ניר אדר ואפיק קסטיאל.

תוכן עניינים

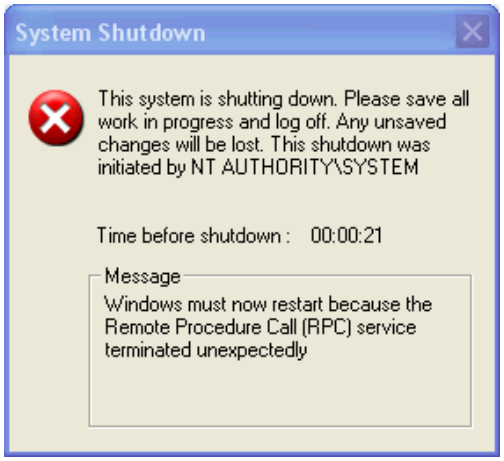
חדשות

מאת אפיק קסטיאל (cp77fk4r)

Blaster

חודש שעבר, לפני עשר שנים, הופיע הדיווח הראשון על התולעת Win32.Lovesan.a, שהיום ידועה יותר בכינוים MSBlast / Blaster. התולעת ניצלה חולשה במנגנון ה-DCOM RPC במערכת ההפעלה Windows (MS03-026), שלפי מיקרוסופט הוגדרה כך: " Buffer Overrun In RPC Interface Could Allow Code Execution". החולשה הייתה קיימת במערכת הפעלה Window XP 32bit וב-Windows Server 2000. שבאותה העת (שנת 2003) היו שיא הטכנולוגיה והיוו את רב השוק המיקרוסופטי. ניצול החולשה לא דרש אינטרקצייה של המשתמש, מה שהפך אותה להיות קטלנית ביותר.

במקרים בהם התולעת הייתה מנסה לתקוף מערכות Windows Server 2003 היא ברב המקרים הייתה גורמת לשירות ה-RPC שלהן לקרוס ולמערכת ההפעלה להתחיל תהליך כיבוי לאחר הצגת ההודעה:



את החולשה אותה ניצלה התולעת, גילו האקרים מהקבוצה Last Stage of Delirium הפולנית כחלק מפרויקט מיפוי של מנגנון ה-RPC של מיקרוסופט. את המצגת על הפרויקט, בשם " Microsoft Windows RPC Security Vulnerabilities" הם הציגו בכנס Hack In The Box 2003 (ניתן להוריד אותה מכאן).

Blaster הייתה התולעת הראשונה (הידועה לפחות) שניצלה חולשה מרוחקת במנגנון ה-RPC, לאחריה פורסמו עוד תולעים נוספות שניצלו כשלי אבטחה שונים באותו המנגנון, תולעים כגון Welchia ו-Sasser.

התולעת, קיבלה את שמה בעקבות טקסט שחוקרי וירוסים מצאו בא:

```
"I just want to say LOVE YOU SAN!! soo much"
```

ובעקבות שם הקובץ הבינארי אותו הורידה התולעת:

```
msblast.exe
```

הטען של התולעת שקל 11kb, ולאחר כיווץ ב-UPX הוא שקל 6kb, ובמהלך ריצתו הוא היה מוריד את הקובץ msblast.exe לתיקיית מערכת ההפעלה ומריץ אותו משם.

הודעה נוספת שנמצאה בקוד של התולעת הייתה:

```
Billy Gates why do you make this possible ? Stop making
money and fix your software!!
```

מבחינת דרכי טעינה או שרידות, הוירוס לא היה מתוחכם מדי, הוספת ערך אחד תחת המפתח:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion
\Run
```

היה כל מנגנון השרידות שלו.

לאחר שהתולעת הייתה מורצת על המחשב היא הייתה מתחילה לסרוק טווחי IP (לפעמים מקומיים ולפעמיים אינטרנטים) ומנסה להדביק אותם. במידה והיא הייתה מוצאת קורבן שניתן היה לנצל בו את MS03-026, היא הייתה שולחת לו Payload להרצת Shellcode שפותח סוקט להאזנה בפורט 4444 ואז באמצעותו היא הייתה שולחת את עצמה ומריצה את עצמה על המחשב החדש.

באותה התקופה, התולעת היוותה סיכון לא קטן, שכן היא ניצלה חולשה מאוד קריטית, בדו"ח שהגישו F-Secure נכתב כך:

```
Q:What makes this worm special?
A:It spreads using the MS03-026 DCOM/RPC hole, "Buffer
Overrun In RPC Interface" - which is one of the most common
security holes in the world right now.
```

המטרה של התולעת הייתה בסופו של דבר ליצור ב-16/08/2013 מתקפת DDoS מסוג SYN_flood בפורט 80 כנגד אתר העדכונים של מיקרוסופט. אך במקום הכתובת www.update.microsoft.com, כותבי התולעת הכניסו את הכתובת: windowsupdate.com, כך שלא נגרם שום נזק לאתר העדכונים.

בעניין הזה קמו לא מעט תיאוריות. כותבי וירוס כל כך רציני פספסו ב-URL שמהווה את המטרה של התולעת? הרבה חוקרי אבטחה הסכימו כי הטענה מטופשת. אך עם זאת, שום חוקר אבטחה לא פרסם כי מצא מטרה נוספת לשמה נכתבה התולעת.

מקורות וקישורים לקריאה נוספת:

- http://www.securelist.com/en/descriptions/old24773
- http://technet.microsoft.com/en-us/security/bulletin/ms03-026
- http://www.f-secure.com/weblog/archives/Blaster_press_release.pdf
- http://nsrg.eecs.umich.edu/publications/IEEE_Security_Privacy_Blaster_Final.pdf
- http://www.computerworld.com/s/article/84519/Blaster_Worm_Linked_to_Severity_of_Blackout
- http://www.tik.ee.ethz.ch/~ddosvax/publications/papers/dimva2005-duebendorfer-et-al-blaster_sobig.pdf
- http://www.bme.jhu.edu/resources/IT/MSBlast/w32.blaster.worm.info.pdf

OpenX BackDoor

OpenX הינה מערכת לניהול פרסומות באתרי אינטרנט. חודש שעבר שוחררה הגרסא מספר 2.8.11 של OpenX, וביחד עם הגרסה, פורסם פוסט בבלוג של OpenX על ידי ניק סוראקו. בפוסט מספר ניק, שמספר חוקרי אבטחה גרמנים מ-heise Security, פנו אליהם וסיפרו להם על כך שהם מצאו כי שני קבצים ב-OpenX הוחלפו.

אותם חוקרי אבטחה גילו כי שני קבצים בחבילה OpenX 2.8.11 הולפו בקבצים הנראים זהים לקבצים המקורים, אך כוללים בתוכם קוד המאפשר למי שיודע אודותיו להריץ קוד מרחוק על השרת. נראה שהקוד הזדוני נמצא במוצר עוד מנובמבר 2012. את הגרסה החדשה (הכוללת תיקונים וכמובן - אינה כוללת את הקוד הזדוני), ניתן להוריד מהקישור הבא:

http://download.openx.org/openx-2.8.11.zip

בפוסט, ניק מוסיף ואומר שהקוד הזדוני נמצא רק במוצר החינמי שמציעה החברה ולא בכלל החבילות.

בפוסט שפרסמו החוקרים מ-heise Security הם מפרטים אודות הממצא ומספרים כי הקוד שהוחלף נשתל באמצע הקוד המטפל ביצירת סקריפט jQuery:

```
this.each(function(){l=flashembed(this,k,j)}<?php /*if(e)
{jQuery.tools=jQuery.tools||{version:
{}};jQuery.tools.version.flashembed='1.0.2';
*/$j='ex'./**/'plode'; /* if(this.className ...
```

הקוד מאפשר לתוקפים לשנות את התוכן של /www/images/debugs.php ולהחליפו בקוד המקנה להם גישה נוחה לשרת (WebShell) ומשם להשתלט עליו לגמרי.

על מנת לבדוק האם בגרסא שלכם מותקנת אותה דלת אחורית, חוקרי האבטחה מ-heise Security ממליצים לחפש בכלל קבצי ה-js על השרת את המחרוזת "<?php". על שרתי לינוקס:

```
find . -name \*.js -exec grep -l '<?php;\{}'
```

על שרתי Windows:

```
Findstr /s "<?php" *.js
```

אם אתם מוצאים כאלה - תהיו בטוחים שהם לא אמורים להופיע שם.

זאת לא הפעם הראשונה שאנחנו רואים מקרים כאלה, לא מזמן ראינו (וגם הזכרנו במסגרת המגזין) את הדלת האחורית שהאקרים הכניסו ב-UnrealIRCd, ב-ProFTPD וב-phpMyAdmin ובעוד תוכנות נוספות...

SSL BREACH

בכנס Black Hat האחרון, שלושת חוקרי האבטחה יואל גלוק, ניל האריס ו-אנג'לו פרדו פרסמו פרטים טכנים אודות מתקפה קריפטוגרפית שהם פיתחו. המתקפה מאפשרת לתוקפים לדלות פרטים רגישים העוברים תחת פרוטוקול ב-SSL/TLS.

למתקפה קוראים **BREACH** (קיצור של Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext), וכאמור, היא מאפשרת להאקר, הנמצא על מחשב שלישי המאזין לתווך (כגון במקרים של Man in the middle תחת התקפות Routing) לדלות נתונים מתווך התקשורת המוצפן. הרעיון העיקרי שעומד מאחורי המתקפה מזכיר מאוד את הרעיון מאחורי המתקפה CRIME - העובדה שבמקרים רבים מתבצע כיווץ למידע העובר בין שני צדדי התקשורת.

הרעיון של CRIME היה לנצל את הפיצ'ר הקיים בגרסאות השונות של הפרוטוקול TLS שתפקידו לכווץ את המידע בעת הצפנתו ברמת ה-TLS. הבעיה העיקרית ב-CRIME הייתה שברב המקרים, האופציה של כיווץ ברמת ההצפנה הייתה מבוטלת כברירת מחדל. חוקרי האבטחה שפיתחו את BREACH, לקחו את הריעון צעד אחד קדימה והרכיבו אותו על שלב הכיווץ ברמת ה-HTTP, ובכך הרחיבו את שטח הפגיעה של ההתקפה.

חוקרי האבטחה יוצאים מנקודת הנחה כי לתוקף קיימת גישה לתעבורת המידע המוצפנת בזמן אמת (לדוגמא, כאשר גורם זדוני ברשת המקומית, מבצע ARP Poisoning למחשב אחר ברשת). במקרה הזה, לאותו גורם זדוני קיימת גישה לתווך התקשורת בין היעד הגלישה של הקורבן לבין הקורבן עצמו, אך הוא אינו מסוגל לשלוף את הנתונים הנמצאים בתוך התקשורת מפני שהם מוגנים באמצעות SSL/TLS.

המתקפה מנצלת את העובדה כי בהרבה מקרים, פרוטוקול ה-HTTP "משקף" ללקוח מחרוזות שונות, כדוגמת נתונים שונים בכותרי הפרוטוקול (לדוגמא - ערכי Cookies הנשלחים מהמשתמש לשרת ובחזרה), או כדוגמת משתנים ב-GET. במקרים כאלה, כאשר קיימת מחרוזת המוחזרת מהשרת ללקוח מופיעה גם בבקשה עצמה, חבילת המידע המוחזרת מהשרת תהיה קטנה בהרבה מחבילה שבה קיימים נתונים חדשים.

המתקפה BREACH מנצלת את העובדה שהתוקף אומנם לא יכול לדעת מה התוכן של כל חבילת מידע, אך כן את גודלה, ובכך לנסות לנחש את הנתונים. אם אנו יודעים כי התוקף גולש למערכת מסוימת, ואנו יודעים בדיוק איזה פרמטר מעניין אותנו, ומה שמעניין אותנו זה ערכו, נוכל לנסות לנחש תו-אחר-תו את ערכו ולבדוק את גודל החבילה המוחזרת.

במאמר הטכני שפרסמו החוקרים, הם נתנו כדוגמא את המערכת OWA (קיצור של Outlook Web Access) של מיקרוסופט.

אם אנו מעוניינים לשלוף את ה-CSRF Token של משתמש ב-OWA על מנת לבצע פעולות בשמו, "כל" שאנו זקוקים לו זה הערך הקיים ב-"canary". הערך נשלח ב-URL והוא נראה כך:

```
GET /owa/?ae=Item&t=IPM.Note&a=New&id=canary=<guess>
```

אין לנו שום בעיה להקים מערכת OWA על שרת ה-Exchange שלנו ולראות את מבנה החבילה. אופן המתקפה שמבוצעת ב-BREACH תהיה לנסות לנחש את הערך הקיים ב-canary. לנסות לנחש את כלל הערך יהיה בלתי אפשרי (אורכו הוא 32 בתים), אך בזכות מודל הכיווץ של הפרוטוקול, ובזכות זה שאנו יכולים לדעת "מבחוץ" מה הגודל של חבילת המידע, נוכל לנסות לנחש את התו הראשון של המשתנה. במידה ונצליח - גודל החבילה יקטן ביחס לחבילה שבה לא הצלחנו לנחש את אותו התו.

הרעיון מזכיר מאוד את המשחק "בול-פגיעה", אנו משחקים את המשחק מול השרת והוא רק עונה לנו בכן ולא האם פגענו, במידה וננסה לנחש את כלל המחרוזת - לא נוכל לדעת היכן פגענו. אך מפני שמדובר במתקפה שמבצעת On-line, נוכל למקסם את הסיכויים שלנו ולנסות לנחש תו תו, ובכך להקביל את המתקפה למשחק "בול-פגיעה" כך שנחייב את השרת לענות לנו תמיד ב-"בול" עם פגענו (מפני שאנו מנחשים תו-תו, נוכל תמיד לדעת מה המיקום של התו שפגענו - מפני שאנו מנסים לנחש תמיד רק תו אחד בכל איטרציה).

החוקרים מאחורי BREACH מימשו את המתקפה כך שהיא תנסה לנחש שני תווים בכל בקשה, תו בכל אזור שונה במשתנה אותו אנו מעוניינים לשלוף. הרעיון הוא שהם מבצעים משחקי סטטיסטיקה וחילוף בין צמדי תווים עם התוצאות שחוזרות על מנת לחסוך בבקשות, ובכך לקצר משמעותית את זמן התקיפה.

בסוף המאמר, החוקרים הכניסו פרק קצר ובו הם מציעים דרכים להתמודד עם המתקפה ולהתגונן מפניה. להלן חלק מהפתרונות שהוצגו:

- המתקפה הנ"ל מתבססת על היכולת של התוקף לדעת מה גודל החבילה שמוחזרת מהשרת לאחר הכיווץ, חוקרי האבטחה הציעו שבכל חבילה שנשלחת תשלח כמות אקראית של מידע זבל ובכך הגודל המקורי של החבילה ישתנה והתוקף לא יוכל להתבסס עליו כאל תשובה האם בוצע כיווץ מוצלח לחבילה או לא.
- הפתרון הראשון והפשוט הוא לבטל את האופציה של כיווץ המידע ברמת ה-HTTP, וכך, לא משנה האם התוקף הצליח לנחש תו בודד מערכו של המשתנה או את כולו - לא יופעל שום מנגנון כיווץ והחבילה תשמר בגודלה המקורי.

- פתרון נוסף, שדורש שינוי בארכיטקטורת הפרוטוקול הוא הכנסת האזור בו המשתמש יכול להשפיע (כל מה שמוגדר כ-"User Input" לערוץ כיווץ נפרד משאר החלקים של חבילת המידע. ובכך להפריד לחלוטין את השפעת ההצלחה או הכישלון של ניחוש תו בודד או מחרוזת שלמה מכלל פרטי החבילה.

- שינוי המידע המשוקף מהשרת ללקוח. המתקפה מתבססת על כך שתוקף יודע למה הוא מצפה שיוחזר מהשרת - התוקף יודע שאם הוא שולח את התו "a", והשרת מתכוון להחזיר "a" - אז גודל החבילה יוקטן, והוא יודע לצפות לזה. אך אם התוקף לא יוכל לדעת מה השרת אמור להחזיר כ"תשובה נכונה" כאשר הוא מקבל את הערך "a" - הוא לא ידע מתי הוא פגע ומתי לא.

- ביצוע בקרה לכמות הבקשות הנשלחות מצד-הלקוח לצד-השרת, אין שום סיבה בעולם שמשתמש "רגיל" לא ישלח אלפי בקשות בשניות בודדות לשרת, ובמידה והשרת מזהה זאת - הוא יכול להגיד בוודאות שמשהו אינו כשורה.

מקורות וקישורים לקריאה נוספת:

- https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU/edit?pli=1
- http://en.wikipedia.org/wiki/CRIME_(security_exploit)
- http://breachattack.com/
- http://breachattack.com/resources/BREACH%20-%20SSL,%20gone%20in%2030%20seconds.pdf
- http://breachattack.com/resources/BREACH%20-%20BH%202013%20-%20PRESENTATION.pdf
- http://www.theregister.co.uk/2013/08/02/breach_crypto_attack/
- https://www.blackhat.com/us-13/briefings.html#Prado

הדלת האחורית של ה-NSA באלגוריתם של NIST

בזמן האחרון, עם כל הפרסומים והמסמכים של אדוארד סנודן, אודות כל מני יכולות של כל מני ארגונים לקרוא אימיילים של כל מני אנשים, יש די הרבה בלאגן. אני לא מעוניין (ולא יכול) לעשות סדר. אבל כן הייתי רוצה להתייחס לכתבה הבאה, שפורסמה ברב אתרי החדשות והעיתונים הגדולים: "ה-NSA יכולים לפצח תשדורות SSL".

ולמי שרק הרגע נחת ממאדים:

- http://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security
- http://www.newyorker.com/online/blogs/elements/2013/09/the-nsa-versus-encryption.html
- http://www.zdnet.com/has-the-nsa-broken-ssl-tls-aes-7000020312/
- http://www.theregister.co.uk/2013/09/06/nsa_cryptobreaking_bullrun_analysis/
- http://gizmodo.com/the-nsa-can-crack-almost-any-type-of-encryption-1258954266
- http://www.digitaltrends.com/web/nsa-has-cracked-the-encryption-protecting-your-bank-account-gmail-and-more/

אני לא יודע אם זה נכון, ולמען האמת אפילו לא הצלחתי לקרוא כתבה אחת עד הסוף. אבל כן הייתי מעוניין לפרט אודות הסיפור הבא:

הסיפור שלנו מתחיל, לפני מרץ 2007. אבל למען הסדר. ב-2007, פרסם המכון הלאומי לסטנדרטים וטכנולוגיות (NIST), מסמך בן 130 עמודים בשם: NIST Special Publication 800-90. המסמך נשא את הכותרת הבאה: " Recommendation for Random Number Generation Using Deterministic Random Bit Generators". תוכן המסמך כלל פירוט טכני אודות ארבעה טכניקות שונות למימוש PRNG.

לפני שנמשיך, נפרט קצת אודות PRNG והחשיבות של הנושא:
כמעט כל אלגוריתמי הצפנה המרכזיים כיום, משתמשים באלמנטים אקראיים (או לייתר דיוק פסאודו-אקראיים, נגיע לזה בהמשך) על מנת ליצור פרמטרים אקראיים אותם יהיה ניתן להכניס לאותן המשוואות המרכיבות את האלגוריתמים וכך לשמור על חוזקו מפני כל מני מתקפות שונות.

PRNG (קיצור של PseudoRandom Number Generator או בעברית: "מחולל מספרים פסאודו-אקראיים") הינה משפחה של אלגוריתמים ש"כל" תפקידם הוא לדמות אלמטנים אקראיים, המילה "כל" נמצאת בגרשיים, כי למרות שזה נשמע כמו משימה פשוטה - היא הינה כזאת. במחשבים הכל מתבצע בעזרת בעזרת חישובים קבועים, ולכל נתון יש מקור. אי אפשר להגיד למחשב "שלוף מספר אקראי משום מקום". מחשבים כיום לא מסוגלים להמציא מספר על דעת עצמם. ולכן, כיום, משתמשים באלגוריתמים פסאודו-אקראיים, שתפקידם לדמות אלמנטים אקראיים עד כמה שניתן.

אלגוריתמים שתפקידם לחולל מספרים פסאודו-אקריים לטובת משימות קריפטוגרפיות מכונים: Cryptographically Secure PseudoRandom Number Generator, (או בקיצור CSPRNG, או בעברית מחולל מספרים פסאודו-אקראיים קריפטוגרפי).

כמו שאפשר לנחש, לא פשוט ליצור אלגוריתמים מסוג זה, ועל האלגוריתמים הללו מבצעים בדיקות רבות ועליהם לעמוד בלא מעט מבחנים על מנת להחשב מספיק בטוחים על מנת שיהיה ניתן להשתמש בהם. במקרים רבים, חוסנו של האלמנט האקראי שקול כמעט לחוסנו של אלגוריתם ההצפנה עצמו.

אז בחזרה לסיפור שלנו, המסמך שפורסם ב-2007 על ידי המכון הלאומי לסטנדרטים וטכנולוגיות כלל ארבעה אלגוריתמים שונים, האלגוריתם הראשון מבוסס על פונקציות תמצות (Hashing). האלגוריתם השני מתבסס על פונקציות HMAC (או בעברית: קוד אימות מסרים", משפחה של פונקציות תמצות העושות שימוש במפתח הצפנה סודי). האלגוריתם השלישי מתבסס על מספר צפני בלוקים, והאלגוריתם הרביעי מתבסס על גופים מהגאומטריה האלגברית המכונים "עקומים אליפטים". בסיפור שלנו, אנו נתמקד באלגוריתם הרביעי.

האלגוריתם הרביעי, המכונה "Dual_EC_DRBG" (או בשמו המלא: " Dual Elliptic Curve Deterministic Random Bit Generator"), מתבסס על בעיה מהאלגברה החישובית המוכרת כ-"בעיית הלוגריתם הדיסקרטי", לא נכנס לפירוט הבעיה, אך בכלליות ניתן לומר שקיימים לה פתרונות, אך לא פתרונות יעילים, ועל כן משתמשים בה לא מעט בעולם ההצפנות. מהעמודים הראשונים במסמך של NIST ניתן לראות כי האלגוריתם הנ"ל (כמו השלושה הנוספים) **אושרו, נבדקו והומלצו על ידי ה-NSA**.

כבר ב-2006 עלה הספק כי באלגוריתם הנ"ל הוכנסה דלת-אחורית כך של-NSA יהיה קל לתקוף כל מערכת אשר עושה בו שימוש.

הספקות עלו על ממספר כיוונים, על ידי מספר חוקרים, כגון Berry Schoenmakers, Andrey Sidorenko, Kristian Gjøsteen-וDaniel R. L. Brown, אודות עבודתם ניתן לקרוא בקישורים הבאים:

- http://eprint.iacr.org/2006/190.pdf
- http://eprint.iacr.org/2007/048.pdf

ההרגשה הכללית של אותם חוקרים הייתה כי למרות שמדובר בבעיה קשה מתמטית, השימוש בה לטובת ייצור מספרים אקראיים לא מספיק בטוח. או כמו שברוס שנייר ניסח זאת:

```
The math is complicated, but the general point is that the random
numbers it produces have a small bias. The problem isn't large enough to
make the algorithm unusable.
```

[https://www.schneier.com/blog/archives/2007/11/the_strange_sto.html]

בוסף לכך, באותו מסמך של NIST הוצעה דרך מימוש שנועדה "לכסות על אותה הרגשה".

לאחר פרסום המסמך, באותה השנה, בכנס ההצפנה "CRYPTO 2007" שנערך בקליפורניה, חוקרי האבטחה Dan Shumow ו-Niels Ferguson ממיקרוסופט פרסמו הרצאה בשם: " On the Possibility of a Back Door in the NIST SP800-90 Dual Ec PRNG" (את המצגת ניתן למצוא כאן). במצגת, הסבירו אותם חוקרים על דרך פוטנציאלית שבה ניתן לתקוף מערכת אשר עושה שימוש באותו אלגוריתם:

הנוסחה המרכזית באלגוריתם מתבססת על שני עקומים אליפטיים: P ו-Q שעליהם מתבצעים רב החישובים. P ו-Q הם שני עקומים שלתוקף לא אמור להיות שום מידע אודות היחס ביניהם. גם Dan Shumow וגם Niels Ferguson לא יודעים האם יש יחס בין אותם מספרים, אך במחקר שלהם, הם גילו שבמידה ותוקף יוכל להצביע על יחס בין אותם מספרים, ובעזרת ניתוח של הפלט הראשוני של האלגוריתם - ניתן יהיה לנחש בקלות את יתר הבלוקים המרכיבים את הפלט, ובכך לדעת לחשב את המספר האקראי שהאלגוריתם אמור לספר למערכת המשתמשת בו.

במסמך שפורסם על יד ה-NIST, מצורפים נספחים, בנספח A, ניתן לראות כי הם ממליצים על שימוש במספר עקומים (P ו-Q) **ספציפיים**, לדוגמא:

A.1.1 Curve P-256

p = 115792089210356248762697446949407573530086143415290314195533631308867097853951

n = 115792089210356248762697446949407573529996955224135760342422259061068512044369

b = 5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b

Px = 6b17d1f2 e12c4247 f8bce6e5 63a440f2 77037d81 2deb33a0 f4a13945 d898c296

Py = 4fe342e2 fe1a7f9b 8ee7eb4a 7c0f9e16 2bce3357 6b315ece cbb64068 37bf51f5

Qx = c97445f4 5cdef9f0 d3e05e1e 585fc297 235b82b5 be8ff3ef ca67c598 52018192

Qy = b28ef557 ba31dfcb dd21ac46 e2a91e3c 304f44cb 87058ada 2cb81515 1e610046

A.1.2 Curve P-384

p = 39402006196394479212279040100143613805079739270465446667948293404245721771496870329047266088258938001861606973112319

n = 39402006196394479212279040100143613805079739270465446667946905279627659399113263569398956308152294913554433653942643

b = b3312fa7 e23ee7e4 988e056b e3f82d19 181d9c6e fe814112 0314088f 5013875a c656398d 8a2ed19d 2a85c8ed d3ec2aef

Px = aa87ca22 be8b0537 8eb1c71e f320ad74 6e1d3b62 8ba79b98 59f741e0 82542a38 5502f25d bf55296c 3a545e38 72760ab7

Py = 3617de4a 96262c6f 5d9e98bf 9292dc29 f8f41dbd 289a147c e9da3113 b5f0b8c0 0a60b1ce 1d7e819d 7a431d7c 90ea0e5f

Qx = 8e722de3 125bddb0 5580164b fe20b8b4 32216a62 926c5750 2ceede31 c47816ed d1e89769 124179d0 b6951064 28815065

Qy = 023b1660 dd701d08 39fd45ee c36f9ee7 b32e13b3 15dc0261 0aa1b636 e346df67 1f790f84 c5e09b05 674dbb7e 45c803dd

ידיעת אותו מספר האקראי המוחזר מהמערכת מחליש בצורה ניכרת את מערכת ההצפנה המבוססת עליו, מפני שמספר זה מהווה חלק נכבד מהחוסן של רכיבים נוספים באותן מערכות. מתיו גרין, מסביר בבלוג שלו "A Few Thoughts on Cryptographic Engineering", ולשם כך הוא מביא דוגמא:

אם נשתמש ב-Dual-EC DRBG על מנת ליצור את הערך של "ClientRandom" בשלב ה-ClientHello של בתחילת תהליך ה-HandShake של פרוטוקול ה-SSL/TLS, ואכן ה-NSA שתלו במכוון את אותם עקומים ובעזרת מידע שיש להם אודותם הם יכולים לצפות את הערך שחזר מה-Dual-EC DRNG, הם יוכלו לחשב את ערך ה-Pre-Master המוחזר כחלק משלב ה-Key Exchange. במידה והם אכן יצליחו לעשות זאת - יש בידיהם בדיוק את אותו מידע שיש ברשות ה-Client וה-Server, ובכך בעצם הם יוכלו לבצע את אותם השלבים שהלקוח ביצע על מנת לפענח את המידע המגיע מהשרת - ולבצע את אותם השלבים שאותם ביצע השרת, על מנת לפענח את התשדורת מהלקוח. בשלב הזה, לא משנה מה חוזק המפתח של ה-SSL, מבחינת הצופה מהצד - מדובר בתקשורת HTTP רגילה לחלוטין.

לאף אחד אין מושג מה מיוחד כל כך באותם עקומים, **ולאף אחד אין מושג למה ה-NIST בחרו להשתמש בוודא בהם.** והשאלה המתבקשת כאן היא: האם יש מישהו שיודע מה מיוחד בהם כל כך? האם יש מישהו שיודע משהו אודות היחסים ביניהם? והאם המישהו הזה הוא ה-NSA? לשאלות האלה אין תשובות.

מה היקף השימוש באותו אלגוריתם? אז מסתבר שלא מעט, בקישור הבא ניתן למצוא את הרשימה:
http://csrc.nist.gov/groups/STM/cavp/documents/drbg/drbgval.html

מהרשימה ניתן לראות כי האלגוריתם הנ"ל קיים, בין היתר במוצרים של החברות הבאות:

- RSA
- Cisco
- Juniper
- BlacBerry
- OpenSSL
- McAfee
- Samsung
- Symantec
- Microsoft

שום דבר לא בטוח במאה אחוז, אבל תארו לעצמכם עולם שבו לגוף מסוים, יש גישה... נניח, רק לחצי מהמוצרים של חצי מהחברות הרשומות ברשימה הנ"ל.

עכשיו, תתארו לכם שזה העולם שבו אנו חיים, כיום, מעניין, אה?

מקורות וקישורים לקריאה נוספת:

טכני:

- http://rump2007.cr.yp.to/15-shumow.pdf
- http://crypto.stackexchange.com/questions/10417/explaining-weakness-of-dual-ec-drbg-to-wider-audience
- http://blog.cryptographyengineering.com/2013/09/rsa-warns-developers-against-its-own.html
- http://www.bbc.co.uk/news/technology-24173977
- http://threatpost.com/in-wake-of-latest-crypto-revelations-everything-is-suspect/102377
- http://en.wikipedia.org/wiki/Dual_EC_DRBG
- http://crypto.stackexchange.com/questions/10189/who-uses-dual-ec-drbg
- http://www.nytimes.com/2013/09/06/us/nsa-foils-much-internet-encryption.html?hp&pagewanted=all&_r=0
- https://www.schneier.com/blog/archives/2007/11/the_strange_sto.html
- http://www.dslreports.com/shownews/Canada-Helped-NSA-Compromise-Dual-EC-DRBG-125763
- http://csrc.nist.gov/groups/STM/cavp/documents/drbg/drbgval.html
- http://www.ietf.org/rfc/rfc2246.txt

חדשות:

- http://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security
- http://www.newyorker.com/online/blogs/elements/2013/09/the-nsa-versus-encryption.html
- http://www.zdnet.com/has-the-nsa-broken-ssl-tls-aes-7000020312/
- http://www.theregister.co.uk/2013/09/06/nsa_cryptobreaking_bullrun_analysis/
- http://gizmodo.com/the-nsa-can-crack-almost-any-type-of-encryption-1258954266
- http://www.digitaltrends.com/web/nsa-has-cracked-the-encryption-protecting-your-bank-account-gmail-and-more/

Memory Analysis על קצה המזלג

מאת יניב מרקס ואפיק קסטיאל (cp77fk4r)

הקדמה

במאמרים הקודמים שפורסמו באתר, נותחו שיטות שונות של החדרת malwares למערכת ההפעלה והסתרתם מתוכנות האבטחה הקיימות (user/kernel mode rootkit). מאחר ומערכי ההגנה הסטנדרטיים מתבססים רבות על חתימות ומאחר וההתקפות מבוצעות תוך שימוש בשיטות חדשות וקשות לגילוי, ישנו פער ההולך וגדל של אותם מערכי הגנה.

אחת השיטות הקיימות היום בזיהוי תוכנה זדונית הינה ניתוח זיכרון חי (live memory analysis) של המחשב אותו אנו מעוניינים לבדוק. אנו נרצה לבצע בדיקה זו, למשל, כתוצאה מחשד שעולה לגבי מחשב מסוים או כחלק מתהליך בדיקה שגרתי של מחשבים ברשת ארגונית.

ישנם בשוק כלים מסחריים ותוכנות open-source המאפשרים לבצע בדיקה זו אך אנו במאמר זה נשתמש בתוכנת ה-volatility - open source (code.google.com/p/volatility/) על מנת להסביר ולהדגים מה ניתן לנתח וללמוד מהנתונים שנשלפו מהזיכרון.

שלב א' - יצירת Memory Dump

על מנת שניתן יהיה לנתח אתהזיכרון, מומלץ לבצע שמירה של הזיכרון על ממת שניתן יהיה לבצע את הניתוח במחשב אחר ללא חשש שהתוכנה הזדונית תופעל גם על המחשב הבודק. לצורך כך, קיימות מגוון תוכנות המאפשרות לנו לבצע זאת, כדוגמת mdd_1.3.exe, ניתן להוריד אותה מהקישור הבא:

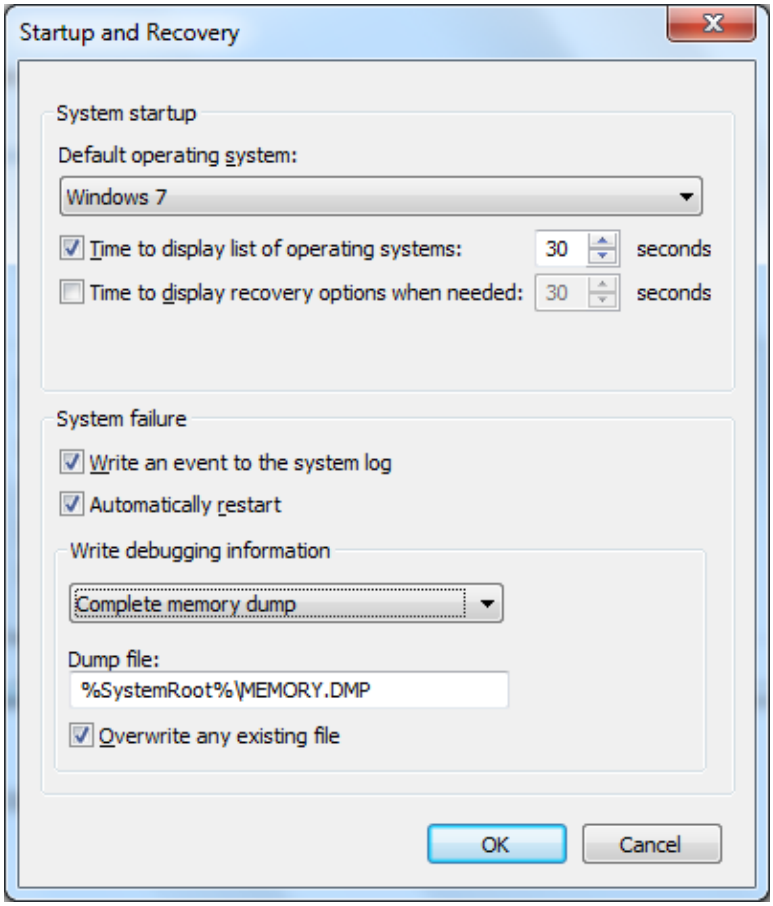
http://sourceforge.net/projects/mdd/files/mdd/mdd-1.3/mdd_1.3.exe/download

או להשתמש בפיצ'רים של מערכת ההפעלה.

על מנת להשיג "Full Memory Dump" של מערכת ההפעלה ברגע נתון, יש ראשית כל לאפשר את האופציה של יצירת Full Memory Dump בעת קריסת המערכת. נבצע זאת בעזרת השלבים הבאים:

- יש להכנס ל-Control Panel ושם להכנס ל-System.
- יש לבחור ב-Advanced ושם ל-Settings ואז ל-Startup and Recovery.

בשלב זה, נגיע לחלון הבא:



תלת "Write Debugging Information" יש לבחור ב-"Complete memory dump".

לאחר מכן נרצה את האפשרות של ליזום קריסה של המערכת על ידי טריגר מהמקלדת - בכל זאת שנרצה, נאפשר זאת בעזרת הצעדים הבאים:

- אם מדובר במקלדת המחוברת באמצעות PS/2, יש להכנס ל-Registry, תחת המפתח הבא:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\i8042prt\Parameters
```

ולהוסיף שם מפתח מסוג REG_DWORD, בשם CrashOnCtrlScroll עם הערך: 0x01.

- במידה ואנו משתמשים במקלדת המחוברת באמצעות USB, יש לבצע את אותו הדבר תחת הערך הבא:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\kbdhid\Parameters
```

לאחר שנבצע Reset למערכת ההפעלה, נוכל ליזום את קריסת המערכת באופן יזום ע"י לחיצת המקשת Right Ctrl ובאותו הזמן לחיצה על המקש SCROLL-LOCK פעמיים. הפעלת הטריגר הנ"ל תשלח למערכת קריאת KeBugCheck עם הערך 0xE2 ("MANUALLY_INITIATED_CRASH") שתגרום לקריסת המערכת ויצירת קובץ ה-Dump.

במידה ותרצו, קיימת אפשרות לשנות את הכפתורים האחראים על יצירת הטריגר, על מנת לעשות זאת, יש להכנס ב-Registry למפתח הבא:

- לבעלי מקלדות PS/2:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\i8042prt\crashdump
```

- ולבעלי מקלדות USB:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\kbdhid\crashdump
```

וליצור מפתח מסוג REG_DWORD בשם Dump1Keys עם אחד מהערכים הבאים:

Value	First key used in the keyboard shortcut sequence
0x01	Rightmost SHIFT key
0x02	Rightmost CTRL key
0x04	Rightmost ALT key
0x10	Leftmost SHIFT key
0x20	Leftmost CTRL key
0x40	Leftmost ALT key

[במקור: http://msdn.microsoft.com/en-us/library/ff545499.aspx]

שינוי לאחד מהערכים הנ"ל יחליף את כפתור ב-Ctrl, אך ישאיר את כפתור ה-SCROLL-LOCK, ניתן להחליף גם אותו, אך על זה תוכלו לקרוא בקישור הבא:

http://msdn.microsoft.com/en-us/library/ff545499.aspx

מעכשיו, נוכל לגרום לקריסה של המערכת בכל רגע שנרצה, לאחר עליית המערכת מחדש, יופיע קובץ Full Dump במיקום:

```
%systemRoot%\MEMORY.DMP
```

לפעמים נרצה לחקור תוכנה מסוימת החשודה כתוכנה זדונית ע"ג VM (VirtualMachine) ואז נרצה לחקור את הזיכרון של המכונה הוירטואלית.

ישנן כל מיני דרכים ליצור memory image גם עבור VM, השיטה שהוצגה בשורות האחרונות תעבוד גם במקרה של VM. ניתן לקרוא על כך גם באתר הבא:

http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1001624

בנוסף, במאמר זה נשתמש בכלי Volatility שתומך גם בפורטמים vmss (קובץ ה-"suspended state" של המכונה הוירטואלית) ו-vmx (קובץ הזיכרון של המכונה הוירטואלית), כך שלקיחת Snapshot יכולה להספיק בהחלט.

שלב ב' - ניתוח המידע

לצורך משימה זו קיימות מספר תוכנות, אך במאמר זה נשתמש בתוכנה volatility. לפני שניגש לאופן השימוש בה, נספר עליה מעט.

Volatility הינה אחת התוכנות החופשיות החזקות כיום המאפשרות לנו לבצע Memory analysis באופן פשוט יחסית. תוכנה זו נכתבה ב-Python ולכן נדרש גם להתקין Python interpreter על המחשב, או לחילופין ניתן להוריד מהאתר גם גרסת standalone המאפשרת לנו לנתח את הזיכרון ללא צורך בהתקנת Python interpreter בכלל.

העובדה כי התוכנה נכתבה ב-Python מקנה לה גמישות רבה ונתון זה מתבטא בכך כי חוקרי זכרון רבים (וביחוד חוקרי וירוסים, סוסים טרויאנים, תולעים ושאר נוזקות) כותבים "פלאגינים" לתוכנה. ומאותו הצד - החברה משתפת פעולה עם הקהילה ומוסיפה פלאגינים אלו ל-"Main Core" של המערכת.

התוכנה נכתבה ע"י החברה Volatile Systems LLC, ומופצת תחת הרישיון GNU. נכון לכתיבת שורות אלו, התוכנה תומכת במערכת הפעלה הבאות:

- 32-bit Windows XP Service Pack 2 and 3
- 32-bit Windows 2003 Server Service Pack 0, 1, 2
- 32-bit Windows Vista Service Pack 0, 1, 2
- 32-bit Windows 2008 Server Service Pack 1, 2
- 32-bit Windows 7 Service Pack 0, 1
- 64-bit Windows XP Service Pack 1 and 2
- 64-bit Windows 2003 Server Service Pack 1 and 2
- 64-bit Windows Vista Service Pack 0, 1, 2
- 64-bit Windows 2008 Server Service Pack 1 and 2
- 64-bit Windows 2008 R2 Server Service Pack 0 and 1
- 64-bit Windows 7 Service Pack 0 and 1
- 32-bit Linux kernels 2.6.11 to 3.5
- 64-bit Linux kernels 2.6.11 to 3.5

החברה מקיימת סדנאות וקורסים רבים בנושא, הן באופן וירטואלי והן בכנסים (כגון DefCon / BlackHat וכו'). קיימים לא מעט בלוגים עם מידע רב אודות התוכנה, אך הבלוג הרשמי של החברה הינו:

http://volatility.tumblr.com

הגרסא האחרונה של Volatility הינה 2.2, היא יצאה באוקטובר 2012, וניתן להוריד אותה בחינם מהקישור הבא:

https://code.google.com/p/volatility/wiki/Release22

התמיכה בקרנל של לינוקס (מ-2.6.11 עד 3.5) קיימת רק הגרסא האחרונה.

הפלאגינים העיקריים של Volatility מתחלקים לנושאים הבאים:

- **Image Identification** - פלאגינים שתפקידם לאתר את גרסאת המערכת ממנה נלקחה דגימת הזכרון.

- **Process and DLLs** - פלאגינים שתפקידם לבצעת את העבודה העיקרית עבור ניתוח תהליכי מערכת וקבצי DLL שנטענו לזיכרון. בין הפלאגינים ניתן למצוא פלאגינים שתפקידם להחזיר את רשימת התהליכים הרצים בזכרון ומידע אודותם, פלאגינים שתפקידם להחזיר את רשימת קבצי ה-DLL שנטענו לתהליך מסוים, פלאגינים שתפקידם לחלץ מידע ממבני נתונים הקיימים בתהליכים ספציפיים במערכת, פלאגינים המחזירים את משתני הסביבה של כל תהליך ותהליך ועוד.

- **Process Memory** - פלאגינים שתפקידם לבצע מיפוי והקלה של העבודה מול זיכריון של תהליך / תהליכים ספציפיים.

- **Kernel Memory and Objects** - משפחה של פלאגינים שתפקידם להקל בעת עבודה מול זיכרון של רכיבים ואובייקטים ברמת ה-Kernel של מערכת ההפעלה.

- **Win32k / GUI Memory** - מדובר במשפחה חדשה של פלאגינים, תפקידה להקל בעת העבודה מול רכיבים הקשורים בעיקר לשולחן העבודה ולממשקי משתמש, ניתן למצוא בה פלאגינים אשר תפקידם לאתר את רשימת ה-Session-ים המחוברים כעת למערכת, רשימת ה-Handles הפתוחים, החזרת ה-Z-Order של החלונות בשולחן העבודה, החזרת רשימת החלונות הפתוחים וכו'.

- **Networking** - קבוצה של פלאגינים שתפקידם לעזור בעת ניתוח אירועי מערכת ההפעלה הקשורים לרשת התקשורת, ניתן למצוא בה פלאגינים המחזירים את ה-Socket-ים הפתוחים, רשימת ה-Connection-ים הפתוחים ומצבם, רשימת ה-TCPObjects ועוד.

- **Registry** - משפחה של פלאגינים שתפקידם להקל בעת העבודה מול רכיב ה-Registry של מערכת ההפעלה, ניתן למצוא פלאגינים המבצעים חיפוש אחר מחרוזות ב-Registry, פלאגינים לשליפת מפתחות שלמים או Hive-ים שלמים, פלאגינים לפענוח מבני נתונים הקיימים ב-Registry ועוד.

- **File Formats -** משפחה של פלאגינים שתפקידה לעזור בעת העבודה עם קובץ הזיכרון, ניתן למצוא בה פלאגינים כדוגמת **crashinfo** - פלאגינים המחזיר מידע אודות קובץ ה-Dump. **Hibinfo** - פלאגין המחלץ מידע מקובץ ה-hibernation של מערכת ההפעלה ועוד.

- **Malware -** משפחה של פלאגינים לטובת מחקר וירוסים וקבצים זדוניים במערכת, ניתן למצוא בה פלאגינים המחפשים Hook-ים על תהליכים, פלאגינים המחפשים קוד או קבצי DLL המוזרקים לתוך תהליכים, פלאגינים המאתרים תהליכים "בלתי-נראים" ועוד.

- **Miscellaneous -** שונות, פלאגינים "יעודיים" הקשורים לנושאים שונים.

כאמור, הגרסא האחרונה תומכת גם בביצוע ניתוח זכרון של מערכות ההפעלה המבוססות על הקרנל של Linux (מ-2.6.11 עד 3.5), רב הפלאגינים דומים לפלאגינים הקיימים עבור מערכות ההפעלה מבוססות Windows, והם מחולקים לקטגוריות הבאות:

- **Processes**
- **Process Memory**
- **Networking**
- **Malware/Rootkits**
- **System Information**

לא נרחיב עליהם במאמר זה.

את רשימת הפלאגינים המלאה בה תומכת Volatility ופירוט אודותם, ניתן למצוא בקישור הבא:
https://code.google.com/p/volatility/wiki/Release22

באופן כללי, השיטה להפעלת volatility הינה (כאשר עובדים עם גרסת ה-standalone):

```
Volatility.exe -f mem.dump command
```

ועבור גרסת ה-Python:

```
python vol.py -f mem.dump command
```

- Mem.dump - הינו קובץ שנוצר בשלב א'.
- Command - הנה הבדיקה אותה אנו מעוניינים לבצע.

לדוגמא, סוגי הבדיקות / פלאגינים הקיימים במערכת (ישנן לא מעט פקודות שהתוכנה מאפשרת אך אנו נסקור רק חלק מהן).

דוגמאות

Imageinfo - חילוץ מידע אודות קובץ הזכרון שברשותנו.

כאשר אנו עובדים עם זכרון של מערכת ההפעלה - עלינו לדעת מול איזה מערכת הפעלה אנו עובדים, במידה ואנחנו לקחנו את ה-Dump אין שום סיבה שלא נדע את גרסאת מערכת ההפעלה. אך לא תמיד אנו לקחנו אותו, ולא פעם נוצר מצב שיש ברשותנו קובץ Dump שאין לנו מושג אודותיו.

Volatility מציעה לנו פתרון המאפשר לנו לדעת מאיזו מערכת הפעלה נלקח ה-Dump ע"י ניתוחו וחיפוש אחר מידע הקיים ב-Header של הקובץ וחיפוש אחר מבנים מסוימים הקיימים בקובץ Dump הנוצר במערכות הפעלה ספציפיות וע"י כך לאתר את גרסאתה.

הפקודה:

```
python vol.py -f mem.dump Imageinfo
```

איתור גרסאת מערכת ההפעלה חשובה בעת עבודה עם Volatility, מפני שיש אפשרות להפעיל פקודה מסוימת על קובץ הזכרון תחת פרופיל של מערכת הפעלה ספציפית ולקבל את הפלט באופן מדוייק יותר.

פלט לדוגמא:

```
python vol.py -f mem.dump imageinfo
Volatile Systems Volatility Framework 2.0
Determining profile based on KDBG search...
             Suggested Profile : Win7SP1x86, Win7SP0x86
                    AS Layer1 : JKIA32PagedMemory (Kernel AS)
                    AS Layer2 : FileAddressSpace (/Users/M/Desktop/win7.dmp)
                     PAE type : No PAE
                          DTB : 0x185000
                         KDBG : 0x8296cbe8
                         KPCR : 0x8296dc00
            KUSER_SHARED_DATA : 0xffdf0000
          Image date and time : 2010-07-06 22:40:28
     Image local date and time : 2010-07-06 22:40:28
          Number of Processors : 2
                    Image Type :
```

[במקור: https://code.google.com/p/volatility/wiki/CommandReference#imageinfo]

כחלק מהרצת הפקודה, חוזר הפלט "Suggested Profile", הפלט הנ"ל ממליץ לנו תחת איזה פרופיל יש להריץ את ניתוח הזכרון מערכת. כעת, לכל פקודת Volatility שנריץ את הקובץ הנ"ל נוסיף את המתג:

```
--profile= Win7SP1x86
```

או:

```
--profile= Win7SP0x86
```

על מנת להורות Volatility בעת ניתוח הזיכרון תחת איזה מערכת הפעלה נלקח ה-Dump ובכך להקל עליה.

לאחר שהבנו תחת איזו מערכת הפעלה נלקח ה-Dump, נוכל להתחיל לעבוד עליו. נוכל להריץ לדוגמא:

Connscan - בדיקת רשימת חיבורי רשת (מקביל ל-Netstat)

הפקודה:

```
python vol.py -f mem.dump --profile=Win7SP0x86 connscan
```

תוצאה לדוגמא:

```
Offset(P)  Local Address             Remote Address            Pid
---------- ------------------------- ------------------------- ---
0x02214988 172.16.176.143:1054       193.104.41.75:80          856
0x06015ab0 0.0.0.0:1056              193.104.41.75:80          856
```

[מקור: http://behindthefirewalls.blogspot.co.il/2013/07/zeus-trojan-memory-forensics-with.html]

על ידי הפעלת הפקודה הבאה, ניתן לראות אילו תהליכים יצרו תקשורת IP ומהן כתובות היעד, המקור ומספרי הפורט. בדוגמא הנ"ל, ניתן לראות כי תהליך שמספרו 856 ניסה לפנות לכתובת IP - 193.104.41.75 בפורט 80 (HTTP), ניתן גם לראות את מיקום התהליך בזיכרון.

Pstree - רשימת התהליכים (מקביל ל-Tasklist).

הפקודה:

```
python vol.py --profile=Win7SP0x86 -f mem.dump pstree
```

תחזיר תוצאה כגון:

```
Volatile Systems Volatility Framework 2.0
Name                                 Pid    PPid   Thds   Hnds   Time
 0x84E6E3D8:wininit.exe              384    340      3     73 2010-07-06 22:28:53
. 0x8D4CC030:services.exe            492    384     12    216 2010-07-06 22:28:54
.. 0x84E19030:svchost.exe           1920    492      8    115 2010-07-06 22:33:17
.. 0x8D4E5BB0:schtasks.exe          2512    492      2     60 2010-07-06 22:39:09
.. 0x8D7E9030:wsqmcons.exe          2576    492      1      3 2010-07-06 22:39:11
.. 0x8D5B18A8:dllhost.exe           1944    492     16    187 2010-07-06 22:31:21
.. 0x8D7EE030:taskhost.exe          1156    492     10    155 2010-07-06 22:37:54
.. 0x84D79D40:msdtc.exe              284    492     15    152 2010-07-06 22:31:24
.. 0x8D6781D8:svchost.exe           1056    492     16    589 2010-07-06 22:29:31
.. 0x8D777D40:taskhost.exe          2520    492     11    224 2010-07-06 22:39:10
.. 0x8D759470:sdclt.exe             2504    492      1      4 2010-07-06 22:39:09
.. 0x8D5574D8:rundll32.exe          2484    492      1      5 2010-07-06 22:39:08
.. 0x84D82C08:SearchIndexer.        1464    492     18    624 2010-07-06 22:33:20
... 0x8D759760:SearchFilterHo       1724   1464      6     82 2010-07-06 22:37:36
... 0x8D55E678:SearchProtocol       2680   1464      8    231 2010-07-06 22:39:27
.. 0x8D5CC030:svchost.exe           1140    492     17    375 2010-07-06 22:29:51
...
```

[מקור: https://code.google.com/p/volatility/wiki/CommandReference#pstree]

על ידי הפעלת הפקודה הבאה, ניתן לראות רשימת תהליכים הרצים במערכת, כולל את מספרי התהליכים שייצרו אותם (PPid), ניתן לראות כי למשל SearchIndexer נוצר על ידי תהליך מספר 492, בחיפוש נוסף בטבלה נוכל לראות כי התהליך 492 הוא services.exe.

Printkey - הצגת ערכי מפתחות ב-Registry:

הפקודה:

```
python vol.py -f mem.dump dump --profile=Win7SP0x86 printkey -K
"Software\Microsoft\Windows NT\CurrentVersion\Winlogon"
```

תחזיר פלט כגון:

```
Volatile Systems Volatility Framework 2.0
Legend: (S) = Stable (V) = Volatile

----------------------------
Registry: \Device\HarddiskVolume1\Documents and Settings\NetworkService\NTUSER.DAT
Key name: Winlogon (S)
Last updated: 2008-11-26 07:38:23

Subkeys:

Values:
REG_SZ ParseAutoexec : (S) 1
REG_SZ ExcludeProfileDirs : (S) Local Settings;Temporary Internet Files;History;Temp
REG_DWORD BuildNumber : (S) 2600
----------------------------
Registry: \Device\HarddiskVolume1\WINDOWS\system32\config\default
Key name: Winlogon (S)
Last updated: 2008-11-26 07:39:40

Subkeys:

Values:
REG_SZ ParseAutoexec : (S) 1
REG_SZ ExcludeProfileDirs : (S) Local Settings;Temporary Internet Files;History;Temp
REG_DWORD BuildNumber : (S) 2600
...
```

[מקור: http://behttps://code.google.com/p/volatility/wiki/CommandReference#printkey]

על ידי הפעלת הפקודה הבאה, ניתן לבדוק מה מוגדר מפתח ספציפי ב-Registry ומה עכשיו.

Svcscan - הצגת רשימת שירותי המערכת (Services) ומצב הריצה שלהם:

הפקודה:

```
python vol.py -f mem.dump dump --profile=Win7SP0x86 svcscan
```

תחזיר תוצאה לדוגמא:

```
Volatile Systems Volatility Framework 2.0
Record   Order Pid Name       DisplayName                    Type                      State          Path

[snip]

0x6ea738 0xf5 1148 WebClient  WebClient                      SERVICE_WIN32_SHARE_PROCESS SERVICE_RUNNING C:\WINDOWS\system32\svchost.exe -k LocalService
0x6ea7c8 0xf6 1028 winmgmt    Windows Management Instrumentation SERVICE_WIN32_SHARE_PROCESS SERVICE_RUNNING C:\WINDOWS\System32\svchost.exe -k netsvcs
0x6ea858 0xf7 ---- WmdmPmSN   Portable Media Serial Number Service SERVICE_WIN32_SHARE_PROCESS SERVICE_STOPPED --------
0x6ea8e8 0xf8 ---- Wmi        Windows Management Instrumentation... SERVICE_WIN32_SHARE_PROCESS SERVICE_STOPPED --------
0x6ea970 0xf9 ---- WmiApSrv   WMI Performance Adapter        SERVICE_WIN32_OWN_PROCESS   SERVICE_STOPPED --------
0x6eaa00 0xfa ---- WS2IFSL    Windows Socket 2.0 Non-IFS Service... SERVICE_KERNEL_DRIVER       SERVICE_RUNNING \Driver\WS2IFSL
0x6eaa90 0xfb 1028 wscsvc     Security Center                SERVICE_WIN32_SHARE_PROCESS SERVICE_RUNNING C:\WINDOWS\System32\svchost.exe -k netsvcs
0x6eab20 0xfc 1028 wuauserv   Automatic Updates              SERVICE_WIN32_SHARE_PROCESS SERVICE_RUNNING C:\WINDOWS\System32\svchost.exe -k netsvcs
0x6eabb0 0xfd 1028 WZCSVC     Wireless Zero Configuration    SERVICE_WIN32_SHARE_PROCESS SERVICE_RUNNING C:\WINDOWS\System32\svchost.exe -k netsvcs
0x6eac40 0xfe ---- xmlprov    Network Provisioning Service   SERVICE_WIN32_SHARE_PROCESS SERVICE_STOPPED --------
0x6eacd0 0xff ---- lanmandrv  lanmandrv                      SERVICE_KERNEL_DRIVER       SERVICE_RUNNING \Driver\lanmandrv
```

[מקור: https://code.google.com/p/volatility/wiki/CommandReference#svcscan]

על ידי הפעלת הפקודה הבאה, ניתן לבדוק אילו שירותי מערכת (Services) מופעלים ומהו נתיב ההפעלה של כל Service.

כאמור, מלבד ניתוח נתונים אודות מערכת ההפעלה, ל-Volatility יש מספר פלאגינים לטובת איתור קודים זדוניים הקיימים במערכת, נציג מספר דוגמאות:

Apihooks - איתור Hook-ים ברמת User/Kernel Mode.

הפקודה:

```
python vol.py -f mem.dump --profile=Win7SP0x86 -p PID apihooks
```

דוגמא לפלט:

```
Name                              Type      Target                          Value
IEXPLORE.EXE[2044]@winspool.drv   iat       KERNEL32.dll!GetProcAddress     0x0 0x7ff82360 (UNKNOWN)
IEXPLORE.EXE[2044]@winspool.drv   iat       KERNEL32.dll!LoadLibraryW       0x0 0x7ff82ac0 (UNKNOWN)
IEXPLORE.EXE[2044]@winspool.drv   iat       KERNEL32.dll!CreateFileW        0x0 0x7ff82240 (UNKNOWN)
IEXPLORE.EXE[2044]@winspool.drv   iat       KERNEL32.dll!LoadLibraryA       0x0 0x7ff82a50 (UNKNOWN)
IEXPLORE.EXE[2044]@winspool.drv   iat       ADVAPI32.dll!RegSetValueExW     0x0 0x7ff82080 (UNKNOWN)
```

[מקור: https://code.google.com/p/volatility/wiki/CommandReference22]

על ידי הפעלת הפקודה הבאה, אנו בודקים האם ישנם תהליכים במערכת אשר בוצע להם apihook, מה שיכול להצביע על תוכנה זדונית במערכת. בדוגמא הנ"ל, ניתן לראות כי ב-IAT (Import Address Table) של התהליך IEXPLORE, בוצעו Hook-ים למספר כתובות של פונקציות.

Malfind - איתור DLL וקוד מוזרק בזכירון של תהליכים ברמת ה-User Mode ע"י חיפוש תווים באזורים חשודים.

הפקודה:

```
python vol.py -f mem.dump malfind -p PID
```

תוצאה לדוגמא:

```
Volatile Systems Volatility Framework 2.0
Name                 Pid    Start      End        Tag    Hits Protect
explorer.exe           1724 0x01600000 0x01600FFF VadS      0      6
(MM_EXECUTE_READWRITE)
Dumped to: hidden_dumps/explorer.exe.4a065d0.01600000-01600fff.dmp
0x01600000   b8 35 00 00 00 e9 cd d7 30 7b b8 91 00 00 00 e9    .5......0{......
0x01600010   4f df 30 7b 8b ff 55 8b ec e9 ef 17 c1 75 8b ff    O.0{..U......u..
0x01600020   55 8b ec e9 95 76 bc 75 8b ff 55 8b ec e9 be 53    U....v.u..U....S
0x01600030   bd 75 8b ff 55 8b ec e9 d6 18 c1 75 8b ff 55 8b    .u..U......u..U.
0x01600040   ec e9 14 95 bc 75 8b ff 55 8b ec e9 4f 7e bf 75    .....u..U...O~.u
0x01600050   8b ff 55 8b ec e9 0a 32 bd 75 8b ff 55 8b ec e9    ..U....2.u..U...
0x01600060   7d 61 bc 75 6a 2c 68 b8 8d 1c 77 e9 01 8c bc 75    }a.uj,h...w....u
0x01600070   8b ff 55 8b ec e9 c4 95 4b 70 8b ff 55 8b ec e9    ..U.....Kp..U...
```

```
Disassembly:
01600000: b835000000                    MOV EAX, 0x35
01600005: e9cdd7307b                    JMP 0x7c90d7d7
0160000a: b891000000                    MOV EAX, 0x91
0160000f: e94fdf307b                    JMP 0x7c90df63
01600014: 8bff                          MOV EDI, EDI
01600016: 55                            PUSH EBP
01600017: 8bec                          MOV EBP, ESP
01600019: e9ef17c175                    JMP 0x7721180d
0160001e: 8bff                          MOV EDI, EDI
01600020: 55                            PUSH EBP
explorer.exe        1724 0x015D0000 0x015F5FFF VadS      0      6
(MM_EXECUTE_READWRITE)
Dumped to: hidden_dumps/explorer.exe.4a065d0.015d0000-015f5fff.dmp
0x015d0000   4d 5a 90 00 03 00 00 00 04 00 00 00 ff ff 00 00    MZ..............
0x015d0010   b8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00    ........@.......
0x015d0020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
0x015d0030   00 00 00 00 00 00 00 00 00 00 00 00 d0 00 00 00    ................
0x015d0040   0e 1f ba 0e 00 b4 09 cd 21 b8 01 4c cd 21 54 68    ........!..L.!Th
0x015d0050   69 73 20 70 72 6f 67 72 61 6d 20 63 61 6e 6e 6f    is program canno
0x015d0060   74 20 62 65 20 72 75 6e 20 69 6e 20 44 4f 53 20    t be run in DOS
0x015d0070   6d 6f 64 65 2e 0d 0d 0a 24 00 00 00 00 00 00 00    mode....$.......
```

לא נעבור על הפלט לעומק, אך ניתן לראות כי תחת התהליך Explorer.exe, בכתובת 0x015d0000 (אזור עם הרשאות MM_EXECUTE_READWRITE), קיים קוד בינארי (על פי ה-Header) שככל הנראה הוזרק עליו מתוך קוד זדוני המנסה לבצע פעולות תחת הרשאותיו.

Psxview - איתור תהליכים "בלתי נראים" במערכת:

הפקודה:

```
python vol.py -f mem.dump psxview
```

דוגמא לפלט:

Offset	Name	Pid	pslist	psscan	thrdproc	pspcid	csr_hnds	csr_list
0xff1b8b28	vmtoolsd.exe	1668	1	1	1	1	1	0
0x80ff88d8	svchost.exe	856	1	1	1	1	1	0
0xff1d7da0	spoolsv.exe	1432	1	1	1	1	1	0
0x810b1660	System	4	1	1	1	1	0	0
0x80fbf910	svchost.exe	1028	1	1	1	1	1	0
0xff2ab020	smss.exe	544	1	1	1	1	0	0
0xff3667e8	VMwareTray.exe	432	1	1	1	1	1	0
0xff247020	services.exe	676	1	1	1	1	1	0
0xff217560	svchost.exe	936	1	1	1	1	1	0
0xff143b28	TPAutoConnSvc.e	1968	1	1	1	1	1	0
0x80fdc648	1_doc_RCData_61	1336	0	1	1	1	1	0

[מקור: https://code.google.com/p/volatility/wiki/CommandReference#psxview]

על ידי הפעלת הפקודה, ניתן לבדוק האם ישנם תהליכים מוסתרים במערכת. הבדיקה נעשית על ידי השוואה בין מספר מקורות, כגון: PsActiveProcessHead (מצביע לרשימה מקושרת של התהליכים המוגדרים במערכת), רשימה מקושרת של csrss וכו'. בדוגמא הנ"ל, ניתן לראות כי התהליך

1_doc_RCData_61 קיבל את הערך 0 בעמודה pslist (הרשימה המקושרת הכוללת את רשימת התהליכים במערכת אינה כוללת תהליך זה, כלומר אין הצבעה ל-EPROCESS של התהליך) ולכן התהליך חשוד כתוכנה זדונית.

Userassist - איתור התהליכים האחרונים שרצו במערכת:

הפקודה:

```
python vol.py -f mem.dump userassist
```

דוגמא לפלט:

```
REG_BINARY    %windir%\system32\displayswitch.exe :
Count:           13
Focus Count:     19
Time Focused:    0:06:20.500000
Last updated:    2010-03-09 19:49:20

0000   00 00 00 00 0D 00 00 00 13 00 00 00 60 CC 05 00    ............`...
0010   00 00 80 BF 00 00 80 BF 00 00 80 BF 00 00 80 BF    ................
0020   00 00 80 BF 00 00 80 BF 00 00 80 BF 00 00 80 BF    ................
0030   00 00 80 BF 00 00 80 BF FF FF FF FF EC FE 7B 9C    ..............{.
0040   C1 BF CA 01 00 00 00 00                            ........

REG_BINARY    %windir%\system32\calc.exe :
Count:           12
Focus Count:     17
Time Focused:    0:05:40.500000
Last updated:    2010-03-09 19:49:20

0000   00 00 00 00 0C 00 00 00 11 00 00 00 20 30 05 00    ............ 0..
0010   00 00 80 BF 00 00 80 BF 00 00 80 BF 00 00 80 BF    ................
0020   00 00 80 BF 00 00 80 BF 00 00 80 BF 00 00 80 BF    ................
0030   00 00 80 BF 00 00 80 BF FF FF FF FF EC FE 7B 9C    ..............{.
0040   C1 BF CA 01 00 00 00 00                            ........
                        ........
```

[במקור: http://gleeda.blogspot.co.il/2011/04/volatility-14-userassist-plugin.html]

על ידי הפעלת הפקודה הבאה, ניתן לראות אילו תהליכים הורצו על ידי המשתמש במחשב, כאשר למעשה המידע הזה מאוכסן ב-registry. בדוגמא הנ"ל, ניתן לראות כי המשתמש הפעיל את התוכנות calc.exe ו-displayswitch.exe. הרחבה מעניינת על הנושא, ניתן לקרוא בבלוג של חוקר אבטחת המידע Didier Stevens, בפוסט:

http://blog.didierstevens.com/programs/userassist/

Bioskbd - קריאת ה-Keyboard buffer של ה-BIOS.

הפקודה:

```
python vol.py -f mem.dump bioskbd
```

על ידי הפעלת הפקודה הבאה, ניתן לראות (במקרים מסוימים) מקשים שהוקלדו ונשמרו באזור ה-BIOS הנמצא בזיכרון. עובד רק כאשר משתמשים ב-BIOS שאכן שומר את המידע ב-Buffer הנ"ל. לפי הבלוג של המפתחת, הפקודה תעבור על הביוסים של HP, Intel ו-Lenovo.

השגת סיסמאות ה-Login של המשתמשים במערכת:

במידה ובידינו קיים Snapshop של VM, או ממש זיכרון של מכונה שעלינו לחקור, נרצה להתחבר אליה על מנת לאמת את ממצאינו, או בכדי לקחת דגימות חיות מהמערכת (לדוגמא - כאשר נרצה לחקור תולעת שמתנהגת באופן מסוים רק כאשר מפעילים טריגר כזה או אחר). לא תמיד נוכל להשיג את הסיסמאות של אותה מערכת. Volatility מאפשרת לנו להגיע מאוד קרוב לכך ובמקרים לא מעטים אף להכניס אותנו פנימה.

יש בידינו Snapshop של מכונה וירטואלית במצב Lock, ראשית - נרצה לבדוק את סוג מערכת ההפעלה (את זאת אגב, נוכל לעשות בקלות ע"י ניסיון הפעלת ה-Snapshop בעזרת VMware ובדיקה כיצד נראה מסך ה-Logon...). כמו שראינו בתחילת המאמר, בעזרת Volatility נוכל לעשות זאת בעזרת הפקודה Imageinfo. לאחר מכן, נשלוף את רשימת ה-Hive-ים הקיימים ב-Registry במערכת:

```
Python vol.py -f mem.dump --profile=OS-Version hivelist
```

פלט לדוגמא, יראה כך:

```
Volatile Systems Volatility Framework 2.1_alpha
Virtual            Physical           Name
------------------ ------------------ ----
0xfffff8a001053010 0x000000000b1a9010 \??\C:\System Volume Information\Syscache.hve
0xfffff8a0016a7420 0x0000000012329420 \REGISTRY\MACHINE\SAM
0xfffff8a0017462a0 0x00000000101822a0
\??\C:\Windows\ServiceProfiles\NetworkService\NTUSER.DAT
0xfffff8a001abe420 0x000000000eae0420
\??\C:\Windows\ServiceProfiles\LocalService\NTUSER.DAT
0xfffff8a002ccf010 0x0000000014659010
\??\C:\Users\testing\AppData\Local\Microsoft\Windows\UsrClass.dat
0xfffff80002b53b10 0x000000000a441b10 [no name]
0xfffff8a00000d010 0x000000000ddc6010 [no name]
0xfffff8a000022010 0x000000000da51010 \REGISTRY\MACHINE\SYSTEM
0xfffff8a00005c010 0x000000000dacd010 \REGISTRY\MACHINE\HARDWARE
0xfffff8a00021d010 0x000000000cd20010 \SystemRoot\System32\Config\SECURITY
0xfffff8a0009f1010 0x000000000aa1a010 \Device\HarddiskVolume1\Boot\BCD
0xfffff8a000a15010 0x000000000acf9010 \SystemRoot\System32\Config\SOFTWARE
0xfffff8a000ce5010 0x0000000008c95010 \SystemRoot\System32\Config\DEFAULT
0xfffff8a000f95010 0x000000000c2b4010 \??\C:\Users\testing\ntuser.dat
```

[מקור: https://code.google.com/p/volatility/wiki/CommandReference22#hivelist]

המטרה שלנו היא להשיג את המיקום בזיכרון של ה-Hive-ים של ה-SYSTEM ושל ה-SAM. במקרה שלנו:

ה-Hive של ה-SYSTEM נמצא בכתובת:

```
0xfffff8a000022010
```

וה-Hive של ה-SAM נמצא:

```
0xfffff8a0016a7420
```

לא נרחיב על זה יותר מדי, אך למי שלא מכיר, ה-Registry במערכת מחולק ל-Hive-ים שונים, כל Hive תפקידו לשמור מידע מסוג שונה. יש Hive לכל משתמש, ויש Hive כללי לכלל המשתמשים. ב-SAM (קיצור של Security Accounts Manager) מערכת ההפעלה שומרת בין היתר HASH (מסוג LM ו-NTLM) של סיסמאת המשתמשים.

לאחר שהשגנו את הפרטים הנ"ל, נריץ את הפקודה Hashdump באופן הבא:

```
python vol.py hashdump -f image.dd -y [SYSTEM Hive Addr] -s [SAM Hive
Addr]
```

פלט לדוגמא:

```
Administrator:500:08f3a52bdd35f179c81667e9d738c5d9:ed88cccbc08d1c18bcded
317112555f4:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c08
9c0:::
HelpAssistant:1000:ddd4c9c883a8ecb2078f88d729ba2e67:e78d693bc40f92a53419
7dc1d3a6d34f:::
SUPPORT_388945a0:1002:aad3b435b51404eeaad3b435b51404ee:8bfd47482583168a0
ae5ab020e1186a9:::
ASPNET:1004:2b5f618079400df84f9346ce3e830467:aef73a8bb65a0f01d9470fadc55
a411c:::
```

[מקור: https://code.google.com/p/volatility/wiki/CommandReference23#hashdump]

זה לא נושא המאמר, ולכן לא נרחיב על כך, אך ניתן לראות כי המבנה של הפלט הינו שם המשתמש, לאחר מכן הגדרות הקבוצה אליה הוא שייך, ואז שתי מחרוזות בנות 32bit כל אחת. הראשונה הינה סיסמאת המשתמש שמורה כ-LM והשניה היא אותה הסיסמה שמורה כ-NTLM. קיימים כיום כלים רבים ולא מעט טכניקות המאפשרות את שבירת ה-Hash-ים הנ"ל בעזרת Rainbow Tables וכו'.

סיכום

ניתוח זיכרון חי (live memory analysis) מאפשר לקבל מידע רב על הנעשה במחשב שלנו ועל ידי כך לזהות האם ישנן תוכנות זדוניות שמערכות ההגנה הסטנדרטיות לא זיהו.

במאמר זה, נעשה שימוש ב-Volatility (https://code.google.com/p/volatility) על מנת להדגים מה ניתן לחקור תוך שימוש במידע הקיים בזיכרון. יש לציין כי ישנן פקודות רבות שלא הוזכרו במאמר אך יכולות לעזור רבות בניתוח.

מקורות ומאמרים לקריאה נוספת

- https://code.google.com/p/volatility/wiki/CommandReference23
- http://www.aldeid.com/wiki/Volatility
- http://volatility-labs.blogspot.com
- http://gleeda.blogspot.com
- http://www.behindthefirewalls.com/2013/07/zeus-trojan-memory-forensics-with.html
- https://blogs.sans.org/computer-forensics/files/2012/04/Memory-Forensics-Cheat-Sheet-v1_2.pdf

The Fun Part of Android

מאת ניר גלאון

הקדמה

מאמר זה יסביר בצורה עמוקה יחסית על המערכת אנדרואיד, כיצד היא עובדת, איך קבצי Apk בנויים, (תמונות, מחרוזות, וחלקי smali), איך ניתן לשנות קבצי Apk ולבסוף, איך הם נראים מנקודת המבט של המפתח.

מטרת המאמר אינה להסביר כיצד לפתוח קבצי Apk, וכדי לא להאריך את המאמר יתר על המידה, ולכן, ברשותכם, אפנה אתכם לפוסט שכתבתי המסביר בפירוט כיצד לבצע את זה, בעזרת התוכנה Apktool.
לחצו כאן לכניסה לפוסט.

בשביל שנהיה באותו קו, אציין כי החבילה (האפליקציה) שעליה אעבוד בהמשך היא של החייגן, תוכלו להוריד את שלי מכאן.

נתחיל מהבסיס, איך אנדרואיד באמת עובדת?

ראשית נבדיל בין המונחים. כולם מכירים את התרשים המפורסם שבו מפורטים השכבות השונות (הספריות) של המערכת, אך זו לא הדרך שבה היא עובדת (לא בדיוק בכל אופן), אלא הדרך שבה היא בנויה. אם נעלה ונסתכל על עבודת המערכת ממבט הציפור, נבין שהיא עובדת בצורה שונה ממערכות אחרות (לא ממציאה את הגלגל, אך יש פה כמה הברקות).

כדי להבין כיצד היא עובדת יש צורך להבין כיצד קבצי APK עובדים.

למה הכוונה? אנדרואיד כמערכת הינה חבילה של שורות קוד המריצות אפליקציות (או קבצי Apk).
החייגן שלנו הוא אפליקציה (קובץ Apk) שבמקרה הזה, מצורף באופן מובנה לקוד המקור של אנדרואיד. כאפליקציה אנחנו יכולים גם להסיר אותו (כדי לא ליצור בלאגן גוגל הגדירה את האפליקציות המובנות כ'לא ניתנות למחיקה', אך אין זה אומר שהן באמת לא ניתנות למחיקה, עם ההרשאות המתאימות-תוכלו לבצע הכל). כתוצאה מכך ניתן לבצע שינויים בכל דבר במערכת, לדוגמה: ניתן להסיר את החייגן המובנה ולהתקין במקומו אחד אחר מה-Play (לדוג' Skype).

איך קבצי APK בנויים?

אני לא אתווכח אם המושג אפליקציה הוא נכון או לא נכון, אך בשביל להבין עדיף לקרוא לאפליקציות קבצי APK, כי מה שהם בפועל זה חבילות (Packages), והחבילות האלו יכולות להכיל 4 דברים:

1. Activity - אפליקציה מורכבת מהמון מסכים (Activities) שמשתמשים עוברים ביניהם (ולרוב כל מסך מבצע פעולה שונה) המסך הזה נקרא Activity (ברבים: Activities).

כותב האפליקציה בעצם כותב כמה וכמה Activity ועל ידי כפתורים שונים המשתמש מבצע מעבר ביניהם ובמעבר מאפליקציה לאפליקציה אתם עוברים מ-Activity ל-Activity.

דוגמה להבנה: נכנסתם לג'ימייל (נכנסתם ל-Activity בתוך ה-Package שנקרא gmail), משם עברתם לדואר הנכנס (עברתם ל-Activity חדש בתוך ה-Package שנקרא gmail), משם נכנסתם לאיזשהו מייל (עברתם לעוד Activity בתוך ה-Package של gmail), ובתוך המייל שכתבו לכם יש כתובת ולחצתם עליה (ומשם עברתם ל-Activity בתוך ה-Package שנקרא maps), וראיתם שבכתובת שבמפה יש מידע נוסף, לחצתם עליו והוא קישר אתכם לויקיפדיה (אז עברתם ל- Activity בתוך ה-Package שנקרא chrome).

- ה-Activity (המסך) מורכב (לרוב) על ידי כפתורים, תיבות טקסט, תמונות וכד', את ה-Activity מעצבים באמצעות מסמך XML שמאפשר לנו לתאר כיצד המסך יראה והיכן יוצב כל רכיב. בעת מעבר מ-Activity אחד למשנהו (לדוג' בתוך אפליקציה-חבילה) ה-Activity הראשון נעצר וה-Activity השני מתחיל (אבל המערכת שומרת את ה-Activity הקודם במחסנית FIFO - האחרון שנכנס הוא הראשון שיוצא).

2. Service - ה-Service עובד ברקע, ולהבדיל מה-Activity אין לו ממשק משתמש (הוא מתואר על ידי שורות קוד בלבד ושקוף למשתמש). ה-Service עוזר לנו לתת שירות לאפליקציה שלנו באה לתת מאחורי הקלעים.

דוגמה להבנה: הדוגמה הכי קלה היא חבילת (אפליקציית) המוסיקה, חבילת המוסיקה מורכבת מכמה Activity, אנחנו עוברים בין השירים ובוחרים שיר להשמעה, כנראה שלאחר מכן גם עוברים ל-Activity חדש שבו ניתן להריץ את השיר אחורה-קדימה, להפסיק, להפעיל וכד'. אבל מה קורה כשאנחנו יוצאים מה-Activity (מהמסך של המוסיקה, ועוברים לדוגמא ל-Activity של משלוח הודעה)

אך רוצים שהמוסיקה תמשיך לנגן ברקע. על המצב הזה ה-Service עונה. ה- Service הינו רכיב הפועל ברקע ובא לתת לנו עזרה כדי לספק שירות כלשהו למשתמש, כמו להמשיך לנגן מוסיקה.

3. Content provider - ה-Content provider מנהל גישה למערך הנתונים של החבילה (האפליקציה) בין תהליך אחד לשני. כאשר מפתח רוצה לגשת לנתונים של אפליקציה אחרת הוא משתמש באובייקט שנקרא ContentResolver, הוא שולח בקשות ל-Content provider שמנגד מקבל את הבקשה, מבצע את הפעולה ומחזיר את התוצאה.

דוגמה להבנה: אנשי הקשר, אנו יכולים לפתח יישום שניגש ל-Content provider של אנשי הקשר על מנת לגשת לנתונים של החבילה (במקרה הזה, רשימת אנשי הקשר שלנו) ולבקש ממנו לקרוא את הרשימה, לשנות אותה וכד'.

4. Broadcast receiver - הרכיב הזה מאפשר למפתח להגיב לאירועי מערכת שונים. מערכת האנדרואיד משחררת הכרזות שונות במהלך פעולתה, לפעמים ההכרזות האלו מיועדות ל-Filter Intent מסוים ולפעמים הן פשוט "נזרקות לאוויר". הרכיב הנ"ל מאפשר לנו "להקשיב" להכרזות הנזרקות במערכת ולבצע פעולות שונות בהתאם להכרזות.

דוגמה להבנה: אני מניח שרובכם מכירים את האפליקציה שמקפיצה חלון Popup קטן ברגע שאנו מקבלים הודעת SMS, האפליקציה הזאת משתמשת ב-Broadcast receiver, היא מאזינה כל הזמן למערכת וברגע שהמערכת מודיע שקיבלה הודעת SMS האפליקציה נכנסת לפעולה ומבצעת כמה פעולות על מנת להקפיץ לכם הודעה על המסך שתציג את תוכן ה-SMS.

איזה אפליקציה נפתחת, מתי, למה ואיך?

כדי להבין קצת יותר לעומק איך אנדרואיד עובדת, אנחנו צריכים להבין למה אפליקציה אחת נפתחת ולא השניה וכיצד עובד המעבר בין Activity של חבילה-א לחבילה-ב.

<u>אז קודם כל, למושגים:</u>

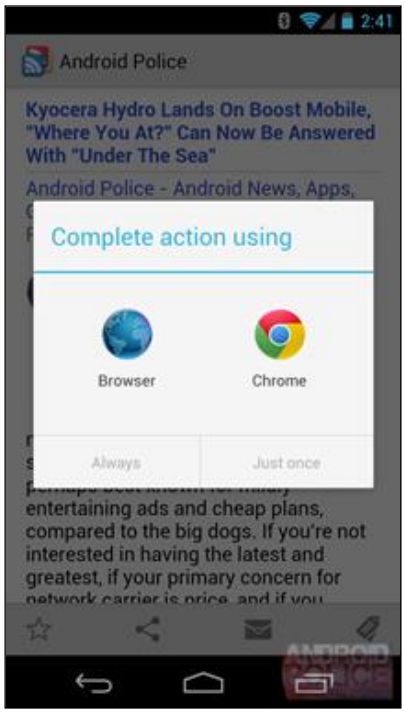
Intent ו-Intent Filter - כל חבילה (אפליקציה) גדולה יחסית תכיל לרוב Activity, Service ו- Broadcast receiver, אלה שלושת מרכיבי הליבה של כל חבילה. שלושת אלה מופעלים באמצעות הודעות הנקראות intents. אובייקט מסוג Intent מייצג כוונה של המשתמש או של מערכת ההפעלה. קיימים 2 סוגים של Intent, הסוג הראשון הוא Intent שמכוון באופן ברור ל-Activity או ל-Component (רכיב חומרתי) ספציפי ונקרא Explicit Intent.

הסוג השני הוא Intent שלא מכוון באופן ברור ואפשר להגיד שהוא פשוט "נזרק לחלל האוויר". במקרה כזה המערכת (Intent Filter) תאתר את ה-Activity (או את הרכיב החומרתי) אליו הוא מכוון בעזרת המאפיינים של ה Intent (שהינם Action, Category ו-Data).

- השימוש הנפוץ ביותר של Intent הינו מעבר בין Activitys (מסכי UI בתוך החבילה).

<u>להלן סיטואציה:</u> במכשיר שלנו יש 2 חייגנים (החייגן המובנה, ו-Skype) ואנו נמצאים ב-Activity של אנשי הקשר, ורוצים להתקשר לחבר. בלחיצה על איש הקשר ולאחר מכן על המספר שלו, עלה לנו חלון קטן שמבקש מאיתנו לבחור דרך איזה אפליקציה אנו רוצים לחייג לאיש הקשר (דרך החייגן המובנה או דרך ה-Skype), למה זה קורה?

כשאנו לוחצים על המספר של איש הקשר אנו שולחים Intent "לחלל האוויר" (ה-Intent לא מיועד באופן ספציפי לאף Activity או רכיב חומרתי. ה-Intent Filter מאתר את ה-Activity (או הרכיב החומרתי) אליו הוא מיועד לפי המאפיינים שהזכרנו לעיל (Action, Category ו-Data).

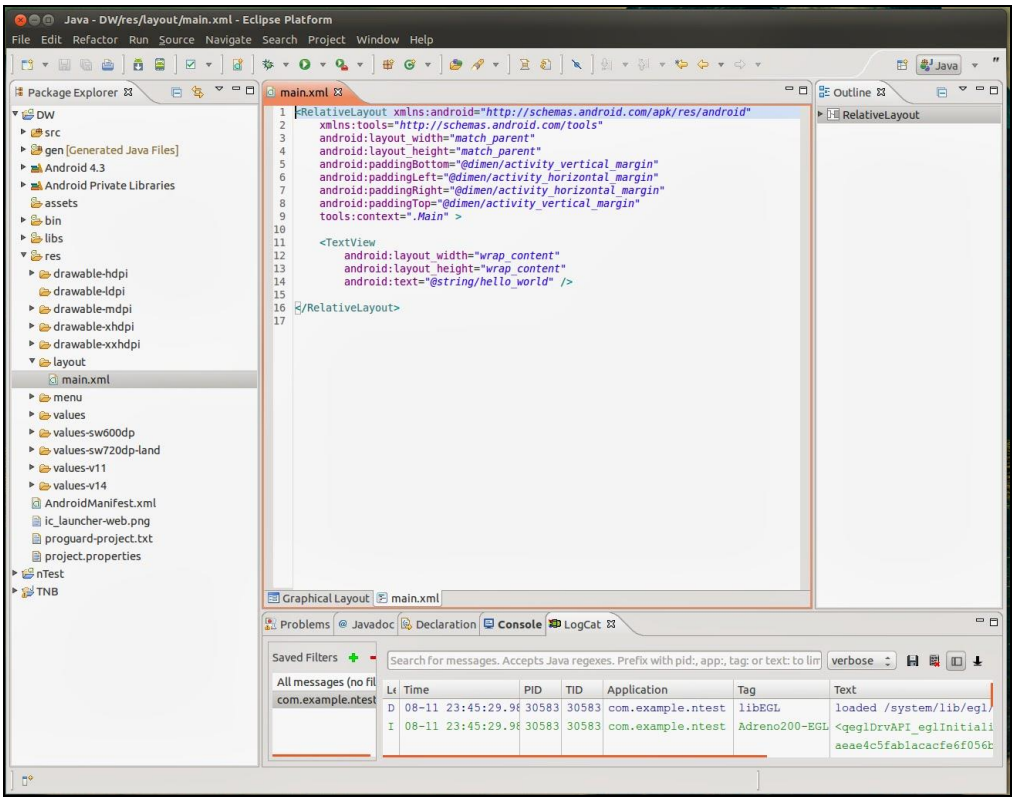


במקרה שלנו: הפעולה (Action) הינה ביצוע חיוג / התקשרות. הקטגוריה (Category) הינה חייגנים. והמידע (Data) הינו המספר שאליו אנו נחייג.

ה-Intent Filter מזהה 2 אפליקציות (חבילות) שעונות על המאפיינים האלו ויכולות להתמודד עם המשימה (החייגן המובנה וה-Skype), והוא נמצא בבעיה, יש 2 אפשרויות, במי לבחור? למערכת אין העדפה, ולכן קופצת לנו הודעת Popup קטנה שמבקשת מאיתנו לבחור כיצד לחייג.

מנקודת המבט של המפתח

בשביל להבין איך החבילות האלו בנויות מנקודת המבט של המפתחים, בניתי אפליקציה פשוטה שמורכבת מ-Activity (מסך אחד). בצד שמאל יש לנו את האקספלורר ובו כל התיקיות והקבצים.



- **התיקייה src:** התיקייה מכילה את כל הקוד וקבצי המקור שפותח בחבילה, בשפת JAVA.

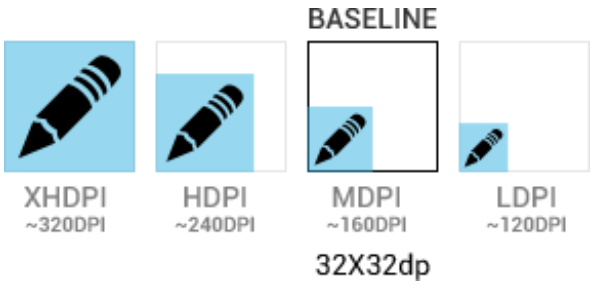
- **התיקייה gen:** המחלקה R היא מחלקה שנוצרת באופן אוטמטי ע"י הפלאגין ADT הכתובה בשפת JAVA (מג'ונרטת באופן עצמאי, אוטומטי). אם נפתח אותה, נראה המון תת מחלקות לפי הקבצים שייצרנו בפרויקט / חבילה שלנו.

- **התיקייה android 4.2:** התיקייה הזאת מכילה את כל החבילות (האפליקציות) המובנות שיש באנדרואיד ואיתן כל המחלקות והשיטות שלהן. כך המפתח לא צריך ליצור כל דבר, אלא מבצע שימוש במחלקות והשיטות הקיימות.

- **התיקייה Assets:** בתיקיית ה'נכסים' משתמשים כדי לאחסן משאבים כדוגמת פונטים (במידת הצורך), קבצי וידיאו, קבצי קול וכו'.

Resources:

- **התיקיות "drawable-hdpi / ldpi / mdpi / xhdpi"** הינן תיקיות שיכילו את כל התמונות שהאפליקציה צריכה. מדובר על אותן תמונות אך ברזולוציות שונות. ולמרות שקיימים עוד המון סוגי רזולוציות באנדרואיד, אלה הבסיסיות ובמידה ולא יצרנו תיקייה לרזולוציה ספציפית, כשהמשתמש יפעיל את האפליקציה המערכת תידע לבחור את התמונה בה הרזולוציה המתאימה ביותר ותתאים את התמונה למסך של המשתמש.



- **בתיקיית "layout"** נמצא קובץ ה-Activity (המסך) שיצרתי, ומתוארים על ידי קבצי XML. קבצי ה-XML שנמצאים ב-layout בעצם מגדירים את סידור המסך.

- **תיקיית ה-"Menu"** מכילה קבצי XML של התפריטים. מי שמשתמש באנדרואיד מכיר את כפתור השלוש נקודות המסמל את מקש האופציה / אפשרויות. אז בדיוק על זה מדובר, פה המפתח יגדיר את המקש הזה והאופציות שלו.

- **תיקיית ה"values"** מכילה קבצי XML עם ערכי מחרוזות. בעצם כל מחרוזות הטקסט שבאפליקציה, ובכך עבודת התרגום לשפות השונות הופכת לקלה יותר מכיוון שהשפות לא צמודות לUI (במילים אחרות: לא Hard coded).

- **AndroidManifest.xml:** הקובץ "AndroidManifest.xml" מכיל את כל המידע הרוולנטי על החבילה: החל משם החבילה (חבילת ה-JAVA) של האפליקציה (המשמש כמזהה יחודי לאפליקציה ב-Play), תיאורים של הרכיבים מהם האפליקציה בנויה (ה-activities, services, broadcast receivers, content providers), הסברים על המחלקות והמתודות השונות שהאפליקציה יכולה לבצע (לדוגמה ה-Intent שאיתם האפליקציה יכולה להתמודד), הצהרה על גרסת אנדרואיד מינימאלית שעליה האפליקציה יכולה לרוץ, פירוט של הספריות בהם האפליקציה עושה שימוש, ולבסוף הצהרות של ההרשאות שהאפליקציה (חבילה) צריכה (ובמידת הצורך גם ההרשאות שאפליקציות אחרות צריכות על מנת לעבוד מולה).

פתיחה ובניה של Apk

Take a look inside

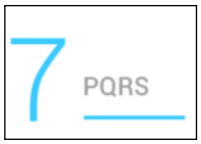
אחרי שפתחנו את הקובץ נקבל תיקייה בשם Phone (בחרתי לקרוא לתיקייה בשם המקורי של החבילה), אך פה נראה שהקובץ שבנוי בצורה קצת שונה ממה שראינו לעיל (מהזווית של המפתח), עכשיו אנו רואים את הפרויקט / חבילה בתצורה הסופית שלה (לאחר שעברה קימפול ע"י הקומפיילר של ה-Dalvik מה-bytecode, שהגיע כתוצאה מהקימפול של Java code).

בואו נתחיל לשחק:

נפתח את ה-Activity של החייגן, אז כמו שאמרנו ה-Activity יימצא בנתיב res/layout, ונפתח את הקובץ dialpad.xml. להלן השורה של הלחצן 7:

```
<ImageButton android:id="@id/seven"
android:src="@drawable/dial_num_7_wht"
android:contentDescription="@string/description_image_button_seven"
style="@style/DialpadButtonStyle" />
```

ואת התמונה של הכפתור לקחתי מהתיקייה drawable-hdpi וניתן לראות (לפי ההפניה ב XML שלעיל) שהשם שלה הוא dial_num_7_wht (להלן התמונה של הקובץ).



כמו שאמרנו, מחרוזות הטקסט באפליקציה נכתבות במסמך שונה, בקובץ XML בשם strings, וקובץ זה יהיה בתיקייה values (לכל שפה ישנה תיקיית values משלה, בצירוף קוד המדינה ב-2 תווים). ה- strings.xml בשפה העברית יהיה בתיקייה values-iw.

להלן חלק מתוכן הקובץ (בירוק מסומן התיאור של הלחצן 7).

```
<string name="description_image_button_five">חמש</string>
<string name="description_image_button_six">שש</string>
<string name="description_image_button_seven">שבע</string>
<string name="description_image_button_eight">שמונה</string>
<string name="description_image_button_nine">תשע</string>
<string name="description_image_button_star">כוכבית</string>
<string name="description_image_button_zero">אפס</string>
<string name="description_image_button_pound">סולמית</string>
<string name="description_dial_button">חייג</string>
```

ובדרך זו אנו יכולים לבצע עוד מגוון שינויים, כגון: לשנות את הרקע של האפליקציה, לשנות את האייקונים והתמונות השונות, אם האפליקציה אינה בעברית - לתרגם או לשנות את התרגום הקיים וכד'.

בנוסף, יש לציין כי במערכת עצמה ניתן לשנות מגוון נוסף של דברים:

- להוסיף / לשנות פונטים, הנמצאים בנתיב: /system/fonts .
- התראות מכשיר, רינגטונים, צלילי מערכת (הקליק של המצלמה, סוללה עומדת להיגמר, חיבור לדוק וכד') בנתיב: /system/media/audio .
- אנימציית ההפעלה של המערכת, נמצאת בנתיב: /system/media והעריכה שלה הינה פעולה פשוטה ביותר (להסבר מפורט יותר: לחצו כאן).
- אם רוצים שאפליקציה מסוימת לא תהיה ניתנת למחיקה (כמו אפליקציות מערכת), פשוט תעבירו את הקובץ Apk שלה לנתיב: /system/app. (אפליקציות רגילית מותקנות בתיקייה DATA/app בעלת ההרשאה rwxrwx—x, בעוד לתיקייה בנתיב system/app יש הרשאות rwxr-xr-x).
- אם רוצים לשנות את כפתורי המגע (במכשירי הנקסוס), אלה הן תמונות הנמצאות ב-SystemUI.apk. (להסבר מפורט יותר: לחצו כאן).

The interesting part

כשפתחנו את ה-Apk היו לנו בתיקייה כמה קבצים ותיקיות, אחת מהתיקיות המעניינות היא תיקיית ה-smali, שם נמצא כל הקסם. Smali אלה קבצי bytecode (לא bytecode של JAVA, אלא bytecode של dalvik). בשפת הסף המתאימים ל-Dalvik (המכונה הוירטואלית של אנדרואיד), (ניתן להגיד שהם המקבילים ל-assembly).

מכיוון שאפליקציית החייגן גדולה ומורכבת, בואו נפריד את הקובץ DW.apk שבניתי בשביל ההדגמה מהסעיף הקודם ("מנקודות המבט של המפתח"), זהו קובץ פשוט שלא עושה כלום חוץ מלהציג Activity אחד ובו רשום Hello world!, את הקובץ ניתן להוריד מכאן.

ניתן לראות שהספרייה smali מכילה תיקיות משנה המגדירות את את המזהה היחודי, אצלי הוא com.example.dw. ניכנס לתיקייה smali>com>example>dw.

בתיקייה dw ניתן לראות שני סוגים של קבצים, כאלו עם הסימן "$" בשם וכאלו בלי הסימן:

- הקבצים ללא סימן ה-$ הינם class רגיל בשפת JAVA.

- סימן ה-$ בשם הקובץ מסמן כי זוהי מחלקת JAVA **פנימית** בקובץ שבאה לפני ה-$ (כלומר הקובץ R$id.smali הינו מחלקה פנימית, class, בקובץ R בשם id).

הקבצים ללא סימן ה-$ הם הקובץ R.smali והקובץ Main.smali.

הקובץ Main.smali זהו ה-Activity שלנו (שמעוצב באמצעות קובץ XML בנתיב res/layout), והקובץ R.smali זהו קובץ שנוצר באופן אוטומטי וממפה את המשאבים של האפליקציה, ז"א כשהמפתח רוצה לקרוא לדוגמא לכפתור, ל-string, ל-layout, או לתמונה מסוימת (מהתקייה drawable), הוא קורא לו מתוך המחלקה R.

כעת, נסתכל בתוך הקובץ Maim.smali:

```
class public Lcom/example/dw/Main;
.super Landroid/app/Activity;
.source "Main.java"

# direct methods
.method public constructor <init>()V
  .locals 0

  .prologue
  .line 7
  invoke-direct {p0}, Landroid/app/Activity;-><init>()V

  return-void
.end method

# virtual methods
.method protected onCreate(Landroid/os/Bundle;)V
  .locals 2
  .parameter "savedInstanceState"

  .prologue
  .line 11
  invoke-super {p0, p1}, Landroid/app/Activity;->onCreate(Landroid/os/Bundle;)V

  .line 12
  const/high16 v0, 0x7f03

  invoke-virtual {p0, v0}, Lcom/example/dw/Main;->setContentView(I)V

  .line 13
  return-void
.end method

.method public onCreateOptionsMenu(Landroid/view/Menu;)Z
  .locals 2
  .parameter "menu"

  .prologue
  .line 19
```

```
  invoke-virtual {p0}, Lcom/example/dw/Main;->
getMenuInflater()Landroid/view/MenuInflater;

  move-result-object v0

  const/high16 v1, 0x7f07

  invoke-virtual {v0, v1, p1}, Landroid/view/MenuInflater;->
  inflate(ILandroid/view/Menu;)V

  .line 20
  const/4 v0, 0x1

  return v0
.end method
```

נחלק לחלקים את הקוד:

בשלוש השורות הראשונות ישנם הצהרות של ה-class.

משורה 5 עד 14 אנו רואים את המתודה של ה-constructor (הבנאי).

משורה 16 עד 32 רואים את המתודה onCreate (אחראית למה שקורה בעת הפעלת ה-Activity).

משורה 34 עד הסוף רואים את המתודה onCreateOptionMenu (שאחראית על השורה העליונה ב-Activity, הפנאל).

הבנאי מופיע כביכול משום מקום, אך זה בגלל שהמחלקה תמיד תהיה מורחבת (extends) מהמחלקה Activity (ניתן לראות זאת גם בשורה השניה, לפי ההפניה של super ל-Activity), ולכן תירש את הבנאי של Activity. אם נסתכל על המתודה onCreate, נוכל לראות כי היא אינה שונה מהקבילה ב-Java.

המתודה protected, השם שלה הוא onCreate, והיא מקבלת פרמטר בשם sacedinstanceState מסוג Bundle (במידה וישנם כמה פרמטרים, ההפרדה ביניהם תתבצע באמצעות נקודה פסיק), ובסוף המתודה מחזירה void.

קל לזהות כי סוגי האוייבקטים המוחזרים מתחילים ב-L וכתובים במרחב המלא. המשתנים לעומת זאת, מופיעים בפורמט הבא:

- **V** - מציין void.
- **I** - מציין int.
- **Z** - מציין boolean.
- **J** - מציין long (64 ביט).
- **B** - מציין byte.
- **F** - מציין float.
- **S** - מציין short.
- **D** - מציין double (64 ביט).
- **C** - מציין char.

אלה המשתנים הבסיסיים, ומי שמתעניין יותר יכול להמשיך לקרוא כאן.

שורה לאחר מכן, אנו רואים ".locals" ומספר, שורה זאת נותנת הוראה ל-Dalvik VM בכמה רשומות להשתמש. בנוסף, ה-smali משתמש ב"v" וב"p" עבור רשומות מקומיות או רשומות של פרמטרים (בהתאמה).

ה-opcode של Dalvik די ברורים, אך יש המון כאלה. לכן אציין פה את החושבים ומי שרוצה לראות את הרשימה המלאה יכול להיכנס לכאן.

- nvoke-super {vx, vy, ...} - מפעיל את המתודה ב class של ההוראה באובייקט vx ומעביר את הפרמטר/ים vy.
- nvoke-virtual {vx, vy, ...} - מפעיל את המתודה הווירטואלית באובייקט vx ומעביר את הפרמטר/ים vy.

נכניס הודעה קופצת (Toast) עם הטקסט Hack, אך נצטרך להכניס אותה ב-smali, והאמת שאין לי מושג איך לכתוב ב-smali, אז להלן השורה ב-Java:

```
Toast.makeText(getApplicationContext(), "Hack",
Toast.LENGTH_SHORT).show();
```

נכניס אותה לאפליקיה, נקמפל, נייצא, נפתח בעזרת apktool, ונראה את הקוד smali שקיבלנו, הוא אמור להיות כזה:

```
  invoke-virtual {p0}, Lcom/example/dw/Main;-
>getApplicationContext()Landroid/content/Context;

  move-result-object v1

  const-string v2, "Hacked!"

  const/4 v3, 0x0

  invoke-static {v1, v2, v3}, Landroid/widget/Toast;-
>makeText(Landroid/content/Context;Ljava/lang/CharSequence;I)Landroid/wi
dget/Toast;

  move-result-object v1

  invoke-virtual {v1}, Landroid/widget/Toast;->show()V
```

ונכניס אותו בשורה 33, לאחר ה-"line 13." ולפני ה-"return-void".

(אנו מכניסים את ה-Toast תחת המתודה onCreate, כי אנו רוצים שההודעה תקפוץ ישר כשנפעיל את האפליקציה). **לא לשכוח להעלות את המספר שליד ה-".locals" שכן אנו נשתמש בעוד רשומה**. נשמור, נבנה את האפליקציה מחדש, ונחתום אותה (שוב, חזרו למדריך הזה).

והעבירו את האפליקציה למכשיר, התקינו אותה, ובדקו שזה עובד.

סיכום

לסיום, למדנו איך מערכת ה-Android עובדת מאחורי הקלעים, ראינו איך קבצי Apk בנויים מנקודת המבט של המפתח, פתחנו קבצי Apk, שיחקנו עם קבצי XML, וביצענו קצת הנדסה לאחור (Reverse Engineering) על ידי התעסקות בקבצי smali.

אני מקווה שהצלחתי לחדש, להעשיר בידע, והכי חשוב - שנהנתם!

I am Very Good - Stage3 Pwned!

מאת רונן שוסטין (Antartic) ופלג הדר (P)

הקדמה

בשנת 2009, במסגרת הכנס "IL-Hack 2009", פרסם השב"כ ארבעה אתגרים בתחום ה-Reverse-Engineering מסוג CrackMe. מטרת פרסום אתגרים אלו הייתה מציאת אנשים בעלי כשרונות בתחום ה-RE. לאחר פתירת כל אתגר הופיעה על המסך הודעה המבשרת על הצלחה ואליה מצורפת כתובת דואר אלקטרוני לשליחת קו"ח. האתגרים ממוספרים מ-0 עד 3 על פי רמות הקושי. במאמר זה נתמקד באתגר האחרון מתוך הארבעה (stage3.exe).

במאמר זה נסקור טכניקות אנטי-דיבאגינג מסוימות. כרקע אנו ממליצים לקרוא את מאמרו המצויין של Zerithבנושא מתוך DigitalWhisper:

http://www.digitalwhisper.co.il/files/Zines/0x04/DW4-3-Anti-Anti-Debugging.pdf

בנוסף כדי להבין לעומק את הנושאים עליהם נדבר במאמר אנו ממליצים להיעזר במקורות אלו:

Intel x86 Instruction Set:

http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.htm

MSDN:

http://msdn.microsoft.com/en-us/library/windows/desktop/ff818516%28v=vs.85%29.aspx

כלים נדרשים:

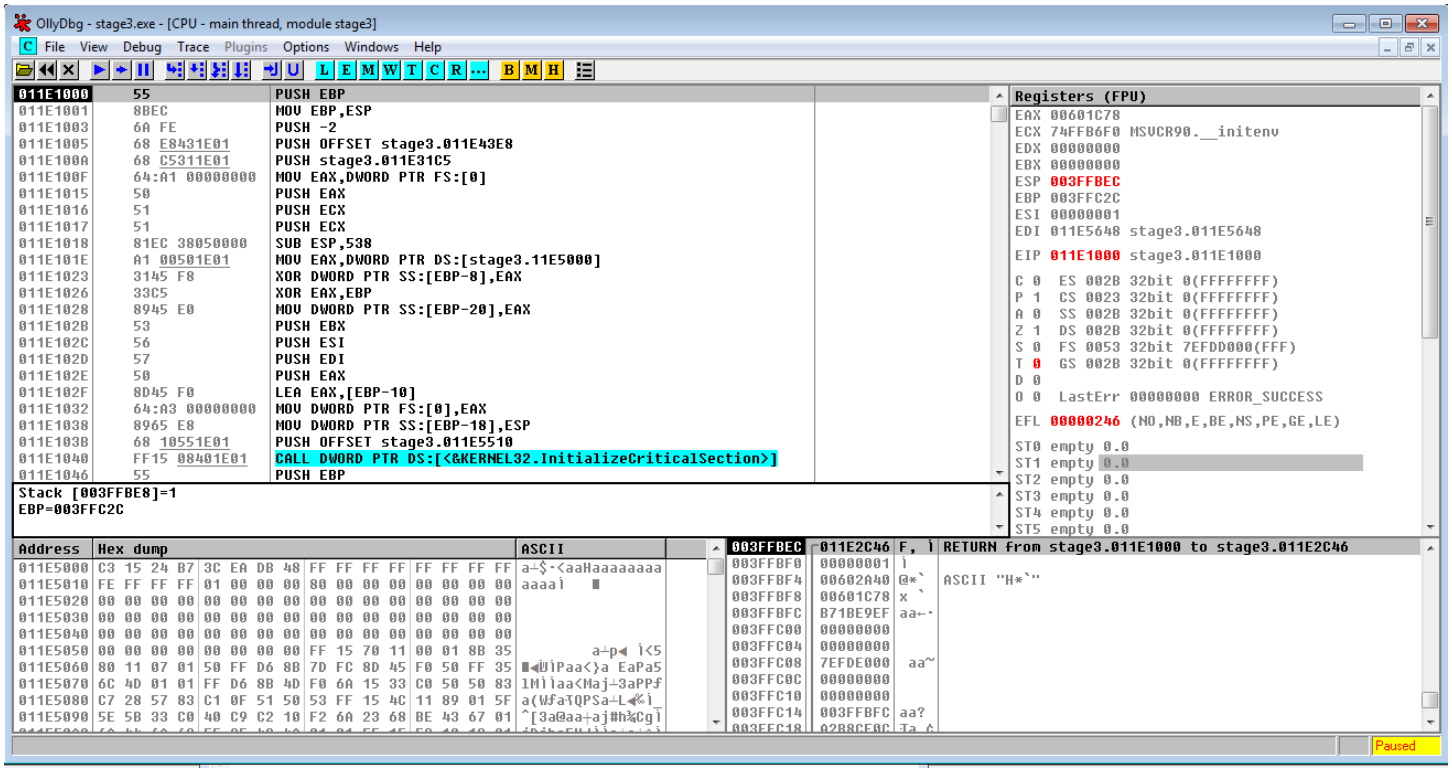
Ollydbg:

http://www.ollydbg.de

אין לייחס חשיבות לכתובות הזיכרון המופיעות במאמר, הן עלולות להשתנות מהרצה להרצה עקב מנגנון ה-ASLR הגורם לשינוי כתובת הבסיס בכל הרצה.

ניתוח התוכנית

נפתח את Ollydbg ונטען את התוכנית stage3.exe. נתחיל לצעוד (Step) בקוד התוכנית ולהתחיל לנתח אותו.



בתחילת התוכנית ניתן להבחין בהוראה INT 3 המשמשת לרוב דיבאגרים בתור נקודת עצירה:

```
5D              POP EBP
8365 FC 00      AND DWORD PTR SS:[EBP-4],00000000
CC              INT3
C745 FC FEFFFFFF MOV DWORD PTR SS:[EBP-4],-2
EB 26           JMP SHORT 00401088
33C0            XOR EAX,EAX
40              INC EAX
C3              RETN
```

כאשר ההוראה מתבצעת, השליטה מועברת אל ה-Exception Handler ומשם נקבעת זרימת התוכנית. מפתחים נוהגים להשתמש בהוראה זו כטכניקת אנטי-דיבאגינג, משום שהרצה של הוראה זו בסביבת דיבאגר תגרום לו לזהות כי זו אחת מנקודות העצירה שהמשתמש קבע לו, ולכן לא יעביר את השליטה אל ה-Exception Handler.

המפתח יוכל לנצל עובדה זאת, ובעזרת משתנה שיתנהג כדגל (flag) שישתנה אך ורק אם השליטה הועברה אל ה-Exception Handler ניתן יהיה לזהות אם דיבאגר מצורף לתוכנית ולשלוט בזרימת התוכנית בהתאם.

בדיקה דומה תבוצע בכל פעם שהשליטה תעבור אל ה-Exception Handler, ונזכיר זאת עוד בהמשך.

נצרף דוגמא הממחישה את הבדיקה המבוצעת:

```
int debugger_flag = 1; /* Assume a debugger is attached. */
__try {
	__asm {
		int 3; /* INT3 trap for the debugger. */
	}
}
__except (EXCEPTION_EXECUTE_HANDLER) {
	debugger_flag = 0; /* Debugger flag set if the control was passed to the exception handler. */
}

if (1 == debugger_flag)
		MessageBoxA(NULL, "Debugger Detected", "Debugger Detected", MB_OK);
```

כעת, נמשיך לעקוב אחר קוד התוכנית ולנתחו:

```
59              POP ECX
C745 FC 0100(   MOV DWORD PTR SS:[EBP-4],1
CD 01           INT 1
C745 FC FEFF(   MOV DWORD PTR SS:[EBP-4],-2
EB 26           JMP SHORT 00041123
33C0            XOR EAX,EAX
40              INC EAX
C3              RETN
```

ניתן לראות בקוד הוראות מסוג INT 1, תפקיד הוראה זו הינו לייצר חריגה שונה מזו ש-INT 3 מייצר, אך מימוש האנטי-דיבאגינג בתוכנית זו זהה לגבי שתי ההוראות.

נמשיך בניתוח התוכנית:

```
68 0C552501     PUSH OFFSET 0125550C
68 00552501     PUSH OFFSET 01255500
68 28562501     PUSH OFFSET 01255628
68 08552501     PUSH OFFSET 01255508
68 A4412501     PUSH OFFSET 012541A4            ASCII "%x %x %x %x"
FF15 BC40250(   CALL DWORD PTR DS:[<&MSVCR90.scanf>]
83C4 14         ADD ESP,14
83F8 04         CMP EAX,4
7D 05           JGE SHORT 01251154
E8 80110000     CALL 012522D4
```

ניתן לראות כי ישנו קלט לארבעה משתנים בעזרת הפונקציה scanf. הפורמט המצופה על-ידי הפונקציה הינו ארבעה ערכים הקסדצימליים.

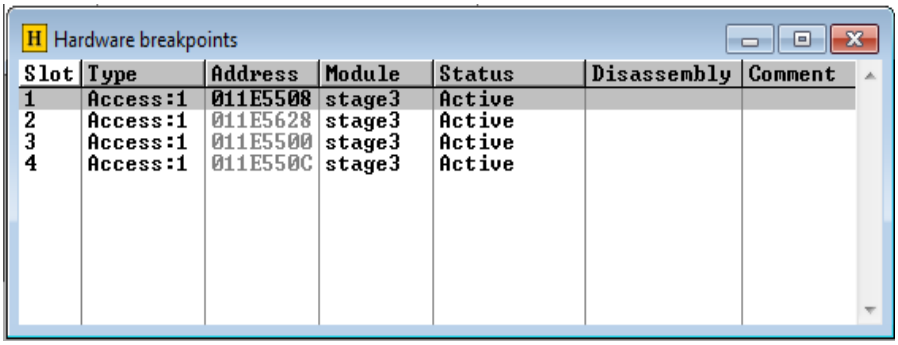
בתרגום חופשי לשפת C, הקריאה לפונקציה scanf נראית כך:

```
scanf("%x %x %x %x", &pass0, &pass1, &pass2, &pass3);
```

כאשר ניתן לראות את כתובות המשתנים שנדחפים בסדר הפוך למחסנית:

```
&pass0 = 0x01245508;
&pass1 = 0x01245628;
&pass2 = 0x01245500;
&pass3 = 0x0124550C;
```

נקבע נקודת עצירה מסוג חומרה (Hardware Breakpoint). כאשר תתבצע קריאה מכל אחת מכתובות המשתנים הדיבאגר יעצור, וכך נוכל לעקוב בקלות ולראות אילו פעולות התוכנית מבצעת על הקלט שהכנסנו.



כאמור, הפונקציה scanf מחזירה את מספר הארגומנטים אליהם המידע נקלט בהצלחה. ואכן ניתן לוודא זאת בהמשך הקוד, כאשר ישנה השוואה עם המספר ארבע. במידה והתנאי לא התקיים, נקפוץ לפונקציה אשר תודיע לנו כי הסיסמא שגויה, ולבסוף תזמן את יציאת התהליך (ExitProcess).

אם הקלט שהוכנס תאם ל-Format String, כלומר נקלטו ארבעה ערכים הקסדצימליים, התוכנית תמשיך לפעול. מכאן ואילך, נתייחס לסדר הסיסמאות בהתאם לסדר הקליטה מן המשתמש, כפי שניתן לראות בקריאה לפונקציה scanf.

בהמשך ריצת התוכנית, ניתקל בהוראה הבאה:

E5 02	IN EAX,2	I/O command

ההוראה IN הינה הוראה מיוחסת (Privileged Instruction) מסוג I/O. במעבדי ה-x86 כל הוראות ה-I/O ניתנות להרצה ב-Ring 0. כאשר ננסה להריץ הוראה מסוג זו תחת רמה מיוחסת פחות (Ring 3), פעולה זו תגרום לזריקת חריגה (Exception) מסוג General Protection או #GP, מכאן תעבור השליטה אל ה-Exception Handler ותבוצע בדיקה דומה לזו שהדגמנו בתחילת המאמר.

לקריאה נוספת על הנושא, אנו ממליצים לעיין בפרק 15.5.1 I/O Privilege Level בספר Intel Manuals.

כאנשים המתעסקים בתחום ה-RE, השימוש בנקודות עצירה הינו חלק בלתי נפרד מתהליך הדיבאגינג, באתגר זה נתקלנו בקושי לעשות זאת ונסביר כעת ממה נבע קושי זה.

```
6A 00           PUSH 0
6A 00           PUSH 0
6A 00           PUSH 0
68 4A210C01     PUSH 010C214A
6A 00           PUSH 0
6A 00           PUSH 0
FF15 20400C01   CALL DWORD PTR DS:[<&KERNEL32.CreateThread>]
```

ניתן להבחין בקריאה לפונקציה CreateThread מתוך ה-API של Windows:

```
CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE) 0x010C214A, NULL, 0,
NULL);
```

הקריאה לפונקציה תיצור Thread חדש אשר יריץ את הפונקציה הנמצאת בכתובת 0x10C214A במרחב הכתובות הוירטואלי של התהליך שקרא לפונקציה (Calling process):

```
C74424 08 0123456  MOV DWORD PTR SS:[ESP+8],67452301
C74424 0C 89ABCDE  MOV DWORD PTR SS:[ESP+0C],EFCDAB89
C74424 10 FEDCBA9  MOV DWORD PTR SS:[ESP+10],98BADCFE
C74424 14 7654321  MOV DWORD PTR SS:[ESP+14],10325476
```

אם נתעמק קצת בקוד הפונקציה, נגלה ארבעה קבועים בגודל ארבעה בתים כל אחד. חיפוש קצר בגוגל יביא אותנו למסקנה כי ישנו מימוש של אלגוריתם ה-MD5 בפונקציה. בהמשך הקוד מתבצעת השוואה בין שני ה-Hash-ים (Checksum), במידה והם אכן שווים, התוכנית תמשיך כהלכה, אחרת, נקבל את ההודעה המפורסמת:

"Debugger Detected :)"

ולאחר מכן תזומן הפונקציה ExitProcess.

אז מה בעצם עומד מאחורי האלגוריתם הזה ומדוע אנחנו לא יכולים להשתמש בנקודות עצירה?

ההשוואה מתבצעת על Checksum של הקוד המקורי (כלומר, ללא שינויים) אל מול ה-Checksum של הקוד במצבו הנוכחי (ה-Code Section). כאשר נשתמש בנקודת עצירה על הוראה מסוימת, הבית הראשון של האופקוד יתחלף ל-0xCC (האופקוד של ההוראה INT 3). בעקבות זאת, כאשר יתבצע ה-Checksum הבא, הוא יהיה שונה מה-Checksum המקורי, דבר שיוביל לזיהוי הדיבאגר.

```
6A 04       PUSH 4
59          POP ECX
BF 5C510C01 MOV EDI,OFFSET 010C515C
8D75 D4     LEA ESI,[EBP-2C]
33C0        XOR EAX,EAX
F3:A7       REPE CMPS DWORD PTR DS:[ESI],DWORD PTR ES:[EDI]
75 1A       JNE SHORT 010C2223
```

הסיסמאות

קודם לכן קבענו נקודות עצירה מסוג Hardware BP על כל אחת מכתובות המשתנים אליהם התבצע קלט הסיסמאות. העצירה הראשונה התבצעה בעת ניסיון קריאה מכתובת המשתנה אליו נקלטה הסיסמא השלישית.

```
E8 190E0000   CALL 002E20C7
FF35 00552E00 PUSH DWORD PTR DS:[2E5500]
68 BC412E00   PUSH OFFSET 002E41BC                ASCII "(:!!!%x!!!:)"
8D85 C0FCFFFF LEA EAX,[EBP-340]
50            PUSH EAX
FF15 C0402E00 CALL DWORD PTR DS:[<&MSVCR90.sprintf>]
83C4 0C       ADD ESP,0C
```

נראה כי ישנה קריאה לפונקציה sprintf עם המשתנה אליו קלטנו את הסיסמא השלישית:

```
sprintf(buffer, "(:!!!%s!!!:)", pass2);
```

בהינתן סיסמא "ABCDEFFE", המחרוזת הסופית תיראה כך:

```
(:!!!ABCDEFFE!!!:)
```

ניתקל בקריאה הבאה לפונקציה:

```
E8 150D0000   CALL stage3.0095204C
```

הפונקצייה הבאה הינה המימוש לאלגוריתם ה-MD5 עליו דיברנו קודם, היא מקבלת את הסיסמא השלישית ה'מקודדת' על-ידי sprintf והופכת אותו ל-Hash (פונקציית גיבוב קריפטוגרפית). עקב השימוש ב-MD5 ניתן להסיק כי ה-Format String בפונקציית ה-sprintf הינו Salt שמטרתו היא למנוע את גילוי הסיסמא על-ידי שימוש ב-Rainbow Tables ו-Dictionary Attack. קריאה לפונקציה נוספת תחזיר מחרוזת המייצגת את ה-Hash שנוצר בפורמט הקסדצימלי:

```
B8 28554000   MOV EAX,OFFSET 00405528   ASCII "acc0a69629dd1831c5980cb95ea0d181"
E8 9A0D0000   CALL 004020E4
```

בהמשך הקוד ניתן למצוא תבנית קוד מוכרת, היוצרת מחרוזת המייצגת Hash בפורמט הקסדצימלי.

```
68 2CA729F8          PUSH F829A72C
68 DC414000          PUSH OFFSET 004041DC                ASCII "aa472df4"
68 DB638E76          PUSH 768E63DB
6A 39                PUSH 39
6A 65                PUSH 65
6A 33                PUSH 33
6A 63                PUSH 63
6A 66                PUSH 66
6A 36                PUSH 36
6A 38                PUSH 38
6A 65                PUSH 65
68 E8414000          PUSH OFFSET 004041E8                ASCII "%c%c%c%c%c%c%c%x%s%x"
8D85 E0FEFFFF        LEA EAX,[EBP-120]
50                   PUSH EAX
FF15 C0404000        CALL DWORD PTR DS:[<&MSVCR90.sprintf>]
```

כך נראית המחרוזת שמחזירה הפונקציה (ה-Hash):

```
0012F8E4  Σ°↕  ASCII "e86fc3e9768e63dbaa472df4f829a72c"
004041E8  §A@  ASCII "%c%c%c%c%c%c%c%x%s%x"
```

נראה כי לאחר מכן מתבצעת השוואה בין ה-Hash הנוכחי שהוחזר ע"י הפונקציה לבין ה-Hash שנוצר מקלט הסיסמא השלישית:

```
5D                   POP EBP
8B85 CCFEFFFF        MOV EAX,DWORD PTR SS:[EBP-134]
0FBE8405 E0FEFFFF    MOVSX EAX,BYTE PTR SS:[EAX+EBP-120]
8B8D CCFEFFFF        MOV ECX,DWORD PTR SS:[EBP-134]
0FBE89 28554000      MOVSX ECX,BYTE PTR DS:[ECX+405528]
3BC1                 CMP EAX,ECX
74 13                JE SHORT 00401C4A
BE 00424000          MOV ESI,OFFSET 00404200             ASCII "Not Good !!!!!"
8B7D 08              MOV EDI,DWORD PTR SS:[EBP+8]
A5                   MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
A5                   MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
A5                   MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]
66:A5                MOVS WORD PTR ES:[EDI],WORD PTR DS:[ESI]
A4                   MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]
E9 95020000          JMP 00401EDF
```

ניתן לראות כי מתבצעת השוואה בין תוכן הבתים של כל אחת מן המחרוזות (המייצגות את ה-Hash-ים) כדי לבדוק האם הסיסמא שהקליד המשתמש נכונה. במידה והן אינן שוות, תופיע המחרוזת " Not Good !!!!" על המסך עם Dump של הזיכרון, אחרת, התוכנית תמשיך לרוץ כמתוכנן.

בכדי למצוא את הסיסמא, נצטרך להשתמש ב-Brute Force מפני שהיא מיוצגת כ-Hash.

מה אנחנו יודעים על הסיסמא?

- יש בידינו את ה-Hash של הסיסמא המקורית.
- הסיסמא היא מספר הקסדצימלי בגודל ארבעה בתים, כלומר, הערך המירבי יהיה 0xFFFFFFFF.
- ה-Salt בו השתמשו בכדי להצפין את הסיסמא הינו: "(:!!!%x!!!:)"

נוכל לכתוב כלי שיבצע Brute Froce בהתחשב בנתונים שרשמנו לעיל. אנו בחרנו ב-Python למטרה זו, ולפניכם הקוד:

```
"""
  Written by Ronen Shustin and Peleg Hadar
  For Digital Whisper
"""

import hashlib

i = 0 # Loop iterator. Also the hex number used for hashing.
enc = "e86fc3e9768e63dbaa472df4f829a72c" # Hashed password to BF.
buf = "" # Buffer.

while (0xFFFFFFFF > i) : # The password is hexadecimal number, 4 bytes sized.
    pass4 = hex(i).rstrip("L").lstrip("0x") # Strip the '0x' from the hex number (iterator) and the 'L' if
long.
    buf = hashlib.md5("(:!!!" + pass4 + "!!!:)").hexdigest() # Generates the MD5 hash with the
salt.

    if (0 == (i % 15000000)) : # Indicator - print loop iterator every 15 million iterations.
        print hex(i)

    if (enc ==  buf) : # Checks if the generated hash is equal to the hashed password.
        print pass4
        break
    i += 1 # Loop iterator increment.
```

כאשר הרצנו את התוכנית על מחשב עם מעבד Core i5 בעל ארבע ליבות, התוכנית פיצחה את ה-Hash תוך 02:19:21 שעות ועצרה ב-0xABCDDCBA, וכך מצאנו את הסיסמא השלישית.

נמשיך לעקוב אחרי התוכנית ונראה כי הדיבאגר יעצור בעת ניסיון קריאה מכתובת המשתנה אליו נקלטה הסיסמא הרביעית:

```
A1 0C551A00      MOV EAX,DWORD PTR DS:[stage3.1A550C]
35 6F6F6F6F      XOR EAX,6F6F6F6F
A3 0C551A00      MOV DWORD PTR DS:[stage3.1A550C],EAX
A1 0C551A00      MOV EAX,DWORD PTR DS:[stage3.1A550C]
33D2             XOR EDX,EDX
83C9 FF          OR ECX,FFFFFFFF
F7F1             DIV ECX
69C0 A32F6760    IMUL EAX,EAX,60672FA3
B9 33C0F7F0      MOV ECX,F0F7C033
2BC8             SUB ECX,EAX
894D E4          MOV DWORD PTR SS:[EBP-1C],ECX
```

נתרגם את האלגוריתם הבא לשפת C:

```
int pass =  *(0x01a550c); //User 4th password.
int final = 0xffffffff; //Final result.

pass ^= 0x6f6f6f6f;
final /= pass;
pass *= 0x60672fa3;
final = (0xf0f7c033 - pass);
```

במידה ומנת החילוק שווה לאפס, פעולת הכפל באלגוריתם לא תשפיע על התוצאה הסופית, והיא תהיה שווה ל-0xf0f7c033.

במידה ומנת החילוק שווה לאחד, התוצאה תוכפל ב-0x60672fa3 ותושם ב-pass. לאחר מכן ערכו של pass יחוסר מערכו של final, כלומר:

```
final  =  (0xf0f7c033 - pass)  =  0x90909090
```

איך נוכל להגיע למנת חילוק ששווה לאחד?

כפי שראינו קודם, במקרה שלנו מנת החילוק תהיה שונה רק כאשר המחלק שווה למחולק, זאת מפני ש-0xffffffff הינו המספר הגדול ביותר בטווח הנתון לנו (מספר הקסדצימלי בגודל ארבעה בתים), אך איך נוכל להגיע למצב בו המחלק שווה למחולק? נסתכל על ההוראה הבאה מתוך האלגוריתם:

```
pass ^= 0x6f6f6f6f
```

נצטרך קלט שלאחר הוראת ה-XOR שלעיל יהיה שווה ל-0xffffffff.

כיוון שהשימוש ב-XOR הינו סימטרי, נוכל למצוא את הקלט הנ"ל:

```
0xffffffff ^ 0x6f6f6f6f = 0x90909090
```

בסוף האלגוריתם, נדחפת תוצאת החישוב אל המחסנית והשימוש בה נעשה בהמשך הקוד.

```
68 84514000     PUSH OFFSET 00405184
C3              RETN
```

ניתן לראות כי כתובת הזיכרון של תוצאת החישוב נדחפת למחסנית, ומיד אחריה ישנה קריאה להוראת RETN, כלומר, זוהי הכתובת אותה המעבד יריץ (EIP) לאחר שיחזור מהפונקציה. הקלט של המשתמש לתוך כתובת החזרה, מאפשרת לו לשלוט על קוד התוכנית משום שהערכים שיקלטו יתורגמו ל-Opcodes.

במקרה שלנו הערך 0xf0f7c033 יתורגם ל:

```
33C0            XOR EAX,EAX
F7F0            DIV EAX
C606 00         MOV BYTE PTR DS:[ESI],0
C3              RETN
```

ערכו של האוגר EAX שווה לאפס כתוצאה מ-XOR עם עצמו. לאחר מכן מתבצעת חלוקה באפס אשר תגרום לחריגה (Exception) מסוג Divide Error או #DE. הערך 0x90909090 יתורגם כמובן ל:

```
90              NOP
90              NOP
90              NOP
90              NOP
```

דבר שימשיך את פעילות התוכנית בצורה תקינה, מכאן ניתן להסיק שהסיסמא היא 90909090.

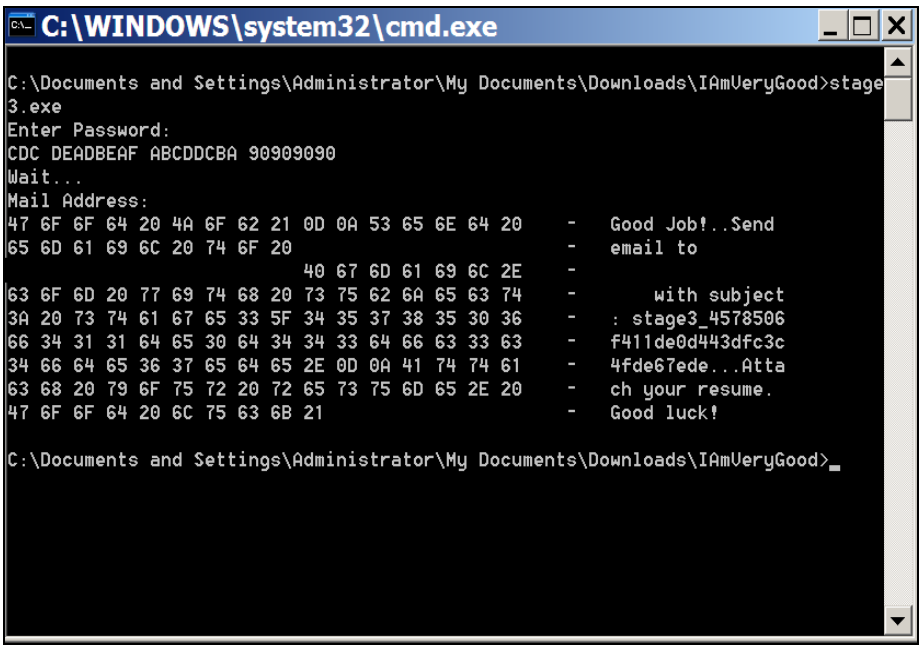
```
813D 28564000 AFB   CMP DWORD PTR DS:[405628],DEADBEAF
74 05               JE SHORT 00401504
E8 D00D0000         CALL 004022D4
C745 FC 07000000    MOV DWORD PTR SS:[EBP-4],7
```

בהמשך התוכנית הדיבאגר יעצור בעת ניסיון קריאה מכתובת המשתנה אליו קלטנו את הסיסמא השנייה. ניתן להבחין בהשוואה ברורה של ערך המשתנה אליו קלטנו את הסיסמא השנייה עם הערך 0xDEADBEAF. במידה וההשוואה נכשלת, ישנה קריאה לפונקציה שבסופה תופיע ההודעה Wrong "Password", אחרת התוכנית תמשיך לפעול כמתוכנן. מכאן ניתן להסיק שהסיסמא השנייה הינה DEADBEAF.

```
5D              POP EBP
FF15 1C404000   CALL DWORD PTR DS:[<&KERNEL32.GetCurrentProcessId>]
3905 08554000   CMP DWORD PTR DS:[405508],EAX
74 05           JE SHORT 00401613
E8 C10C0000     CALL 004022D4
```

נמשיך לעקוב אחר קוד התוכנית, ונראה כי הדיבאגר יעצור בעת ניסיון קריאה מכתובת המשתנה אליו קלטנו את הסיסמא הראשונה. ניתן לראות כי ישנה קריאה לפונקצייה GetCurrentProcessID מתוך ה-API של Windows, המחזירה את ה-PID של התהליך הקורא לפונקצייה (Calling process) בפורמט הקסדצימלי. נבחין מיד לאחר מכן בהשוואה של ערך המשתנה אליו קלטנו את הסיסמא הראשונה עם הערך שהחזירה הפונקציה, כלומר הסיסמא הראשונה אינה קבועה ומשתנה מהרצה להרצה. בפשטות, עלינו לבדוק מהו ה-PID של stage3.exe, להמיר אותו למספר הקסדצימלי, וכך מצאנו את הסיסמא הראשונה.

לאחר הרצת התוכנית והרצת הקלטים הדרושים, נקבל את התוצאה הבאה:



מזל טוב! ☺

סיכום

במאמר ניתחנו את האתגר הרביעי תוך כדי סקירה של שיטות ה-Anti-Debugging אשר מומשו בתוכנית. עקבנו אחר זרימת הקוד, קבענו נקודות עצירה על כתובות משתני הקלט של הסיסמאות, שהובילו אותנו לאלגוריתמים אשר ביצעו בדיקות או מניפולציות על קלט הסיסמא, חקרנו את האלגוריתמים וכך לבסוף הגענו לארבע הסיסמאות.

קישורים לקריאה נוספת

לקוראים המעוניינים להרחיב בנושאים עליהם כתבנו במאמר, מצורפת רשימה של קישורים שימושיים:

Intel x86 Instruction Set:

http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.htm

MSDN:

- http://msdn.microsoft.com/en-us/library/windows/desktop/ff818516%28v=vs.85%29.aspx

המאמר של זריף "Anti Anti-Debugging" מתוך Digital-Whisper:

- http://www.digitalwhisper.co.il/files/Zines/0x04/DW4-3-Anti-Anti-Debugging.pdf

סדרת המדריכים של lena151 למעוניינים להתחיל בReverse Engineering:

- http://www.tuts4you.com/download.php?list.17

הבלוג של R4ndom, כולל המון מדריכים:

- http://www.thelegendofrandom.com/blog

מאגר קראקמי (Crackme) בכל מיני רמות:

- http://www.crackmes.de

כמה עולה לוותר על הפרטיות שלך?

מאת עו"ד יהונתן י. קלינגר

רקע

מהי עלות הפרטיות שלך? על שאלה זו רבים ניסו לענות, ועסקים אחדים עדיין מנסים לברר. נייר קצר זה יבחן את העלות של הפרטיות שלך על ידי נסיון לנתח את שוק האפליקציות הסלולריות כשוק כמעט יעיל, על ידי השוואת הרשאות שניתנות בהתקנת אפלקיציות חינמיות או בתשלום, ועל ידי בחינה האם אפליקציות חינמיות עולות יותר למשתמש או מביאות יותר הכנסות למפתח.

בהמשך, אני אתייחס לאפליקציות ופעולות ספציפיות, ואשאל האם עלות הפרטיות שלך מרשה לך להוריד אפליקציות בחינם, או אפילו להורידן בצורה פיראטית מאתרי שיתוף קבצים.

המסקנה הראשונית שלי, אם בכלל אפשר להסיק דבר, היא שאם אתה מעדיף שלא יעקבו אחרייך, או אתה מעדיף להתחמק מהאפשרות שתהיה תחת מעקב, שימוש באפליקציות בתשלום יפחית את דאגותייך, אך לא יותיר אותך חופשי מדאגות. בדרך הכלל, אפליקציות חינמיות נוטות לבקש יותר מידע על המיקום שלך והיכן אתה, ונוטות להיות יותר פולשניות. עוד ממצא הוא כי מאז 2010 ועד היום, מספר ההרשאות שהתבקשו לכל אפליקציה ככל הנראה גדל בצורה משמעותית.

מהי העלות של אפליקציות בחינם?

בשנים האחרונות, מחירי האפליקציות ירדו מממוצע של כ-10$ למחיר אפקטיבי של 1.5$ (לאפליקציה בתשלום), כאשר 80% מהאפליקציות הן חינמיות להורדה, בהתאם לדו"ח של גרטנר מ-2013. הדו"ח מציין כי 102,000,000,000 (מאה ושתיים מיליארד) אפליקציות יותקנו בשנת 2013, כאשר 26,683,000,000$ יושקעו ברכישתן, בין אם על ידי רכישה ובין אם על ידי רכישה בתוך האפליקציה (In App Payment) או פרסום. הדבר אומר שבפועל המחיר הממוצע לאפליקציה (בכלל, תשלום וחינם) הוא בין 0.26$ ל-0.19$ (על פי המחקר של Flurry); מידע נוסף מראה שהאפליקציה הממוצעת לאנדרואיד עולה יותר מאשר אפליקציה בחנות של אפל.

ביוני 2013, Flurry, חברה שמספקת שירותי מדידה לאפליקצטיות, שחררה דו"ח שטוען כי האפליקציה הממוצעת לאנדרואיד (בתשלום וחינם) עולה 0.06$; כאשר 90% מכלל האפליקציות (אפל ואנדרואיד) ניתנות ללא תשלום; אנו יכולים להניח, עקב כך, שמודל האפליקציה ב-0.99$ פשוט לא עובד.

כמו בשוק התוכנות למחשבים האישיים, השנים האחרונות הראו נדידה מאפליקציות בתשלום לאפליקציות חינמיות. לדוגמא (ונגיע לדוגמא זו מאוחר יותר), מתוך האפליקציות המכניסות ביותר בחנות האפליקציות Google Play בישראל, האפליקציה הראשונה בתשלום היא Asphalt 8, במקום 16. רוב האפליקציות הפופולריות כוללות משחקי הימורים ומזל, כמו Zynga Poker, ומשחקים אשר חינמיים למשחק אך המשתמשים צריכים לרכוש נכסים וירטואליים כדי לשחק, או כדי לקבל חיי משחק נוספים, כמו Candy Crush Saga אשר נמצא כעת בראש הטבלה.

אם אתה מפתח תוכנה, אזי קיימים חופן של אמצעים כדי לייצר הכנסות מהאפליקציה שלך; אבל אם אתה מחשיב את ראש הטבלה של האפליקציות המכניסות ורואה שהן כולן ניתנות בחינם, אז אתה צריך לשקול מחדש את האפשרות למכור את האפלקציה שלך.

שוק האנדרואיד ייבא כמה מהמודלים המסורתיים שראינו במחשב האישי, שהיא מבוססים על תוספי דפדפן, החלפה של מנוע החיפוש ודף הבית, והצגת פרסומות קופצות והתראות שמעודדות משתמשים לרכוש מוצרים. מודלים עסקיים כאלה הם אלו שהעבירו את חברת בבילון מיצרן של תוכנות תרגום לחברה להפצת סרגלי כלים השווה מיליארדי דולרים.

מתווכי פרסומות סלולריות וחברות אחרות מציעות למפתחים דרכים לייצר הכנסות מהאפליקציות שלהם (בעגה המקצועית: מונטיזציה) על ידי התקנת תוספי פיתוח (SDK) שכוללים את קוד התוכנה שלהם. לדוגמא, החברה הישראלית StartApp מציעה למפתחים פתרון מוניטיזציה שכולל הצגת פרסומות למשתמשי הקצה על כל המסך, באנרים, מודעות ביציאה מהאפליקציה, תיבות חיפוש ועמוד שמוצג למשתמש לאחר שיחת טלפון. באזור חודש דצמבר 2012, StartApp הציעה $0.05 עבור כל התקנה בארצות הברית, $0.01 עבור כל התקנה ממדינה אחרת. על אינטגרציה חלקית, שולמו כ-$80 מהסכומים האלו.

אולם, אפליקציות אחרות מוצאות עוד דרכים לשנות את ממשק המשתמש שלך. אפליקציות שמשתמשות במערכת של Conduit, לדוגמא, מחליפות את מסך הנעילה של הטלפון שלך בצורה כזו שמודעות שמוצגת על מסך הנעילה יוצרות הכנסה נוספת למפתח. אחרות משנות את ההעדפות הראשוניות שלך כמו ספק החיפוש שלך, דפדפן ברירת המחדש, הגדרות ה-DNS והעדפות דומות. שירותי מודעות אחרים פשוט ממליצים לך להתקין עוד אפלקיציות מחנויות צד שלישי (שאינן כמו Google Play עם תנאים יחסית מחמירים).

ההגדרה של תוכנה זדונית, אבל, משתנה כל יום. לכן, זה נדמה שכל יום יש דו"ח אחר על כמות התוכנות הזדוניות בחנות האפליקציות של גוגל, כאשר רובם סופרות את כמות התוכנות או מתייחסות להן בצורה ספורדית.

התעריפים שמוצעים על ידי חברות כמו StartApp הולמים באמת את ממציאה של Forbes לגבי ההכנסה הפוטנציאלית מאפליקציות בתשלום. אבל, כאשר TechCrunch הציגו את המספרים לעומק, הם מצאו שרק "20% מהאפליקציות בתשלום מורדות יותר מ-100 פעמים ורק 0.2% מהאפליקציות בתשלום מורדות יותר מ-10,000 פעמים. מנגד, 20% מהאפליקציות בחינם מורדות 10,000 פעמים או יותר" (על פי הדו"ח הזה).

מנגד, AppBrain, שירות המלצת אפליקציות, גובה $0.2 עבור התקנה ממפתחים שרוצים שימליצו על האפליקציה שלהם; זה אומר שכדי שהשירות יהיה כדאי, המפתח צריך להרוויח בדרך אחרת. זה אומר שאפיקי מוניטיזציה קיימים, ומנגד, שישנם תמריצים לוותר על הפרטיות של המשתמשים.

המרווח בין $0.2 להתקנה, לבין הממוצע של $0.26 אומר שישנם דרכים לייצר כסף מהתקנת אפליקציות והפעלת משתמשים. חלק מהן היו פחות אתיות, כאשר נעשו בצורה פולשנית ופחות מודעת לפרטיות. אולם, לאחרונה גוגל שינתה את מדיניות המפתחים כדי להוציא מחוץ לשוק כמה פעולות שהיו לא רצויות, כגון שליחת הודעות Push פרסומיות. אולם, זה עדיין משחק של חתול ועכבר.

באיזה מידע בדיוק משתמשים?

ובכן, הדרך הראשונה בה הפרטיות שלך נפגעת היא על ידי הצגת פרסומות אשר מותאמות לך ספציפית או על ידי מאפיין התנהגותי כלשהוא. אפילו שירות Google AdMob לפרסום סלולרי מאפשר פרסום "מותאם אישית" (כלומר מוכן עבורך). אולם, כמה אפליקציות פשוט שמות "Offer Walls" שמציקות לך להוריד אפליקציות ספציפיות, ואחרות משלבות בתוך האפליקציה שלהן אפליקציות אחרות אשר מציגות לך תוכן לא רצוי.

דוגמא מעניינת היא האתר הפופולרי Ynet, אשר שילב בתוכו את עמוד הנעילה של Conduit, ודרש הרשאות בעייתיות ביותר. אבל זו, כמובן, לא הדוגמא היחידה.

שלב ראשון: האם אפליקציות בחינם דורשות יותר הרשאות?

אפליקציות מסוימות דורשות הרשאות קיצוניות לצורך הפעלתן, כמה מהן עושות זאת כדי לשלב בתוכן אפליקציות נוספות, כאשר אחרות שעושות זאת מהוות סיכון אבטחה. בבדיקה הנוכחית שלי, אני לא יכלתי להפריד בין השתיים, אבל התייחסתי אליהן כאל קבוצות נפרדות.

הגישה הראשונה שלי, שכשלה בצורה אומללה, היתה לבדוק בצורה ידנית את ההרשאות של 30 האפליקציות המובילות בתשלום לעומת 30 האפליקציות המובילות בחינם, ולסמן הרשאות בעייתיות שהיו דרושות (כמו "בטל את נעילת המסך" או "קבל גישה למיקום"). כפי שניתן לראות מהטבלה, בממוצע גם אפליקציות בתשלום וגם אפליקציות בחינם דורשות 3 הרשאות בעייתיות בכל אפליקציה.

התוצאות האלו המשיכו להיות עקביות כאשר בחנתי את 30 האפליקציות המובילות בתחום הבידור.

כאשר בחנתי את 30 האפליקציות המכניסות ביותר, קיבלתי ממוצע גבוה מעט יותר של 3.83. זה אומר שאפליקציות יותר רווחיות, כאשר הן בדרך כלל בחינם, דורשות יותר הרשאות. אולם, המספרים לא מובהקים כפי שרציתי שהם יהיו.

זה אומר שהייתי צריך לחפור עמוק יותר; ההשערה הראשונה שלי היתה שלא נמצא את הבעיות באפליקציות המובילות, כיוון שהן נתונות ליותר ביקורת; לכן, הלכתי ובדקתי גרסאות יותר קשוחות, ועם הרבה יותר כח ואלימות.

בדיקה אמיתית: השוואה בין אפליקציות פיראטיות.

לצורך הבדיקה, הורדתי שלוש חבילות של אפליקציות פיראטיות מאתר The Pirate Bay: הראשונה היא ערכה של 156 אפליקציות חינמיות (חינם כמו בירה, חלק מהן אמורות להיות בתשלום); השניה היא חבילה של 378 משחקים בתשלום; השלישית היא אוסף של 79 ערכות ואפליקציות בתשלום.

בנוסף, הורדתי את Top Android Games Pack מאתר Kat.Ph שמעודכן לספטמבר 2013; כאשר מתוכו, השתמשתי ב-31 משחקים שלא היו בקבצי RAR (בסך הכל 1GB של משחקים מתוך 4GB).

קחו בחשבון שערכת 378 המשחקים היא מ-2010, ולכן יכולה להראות את האבולוציה של דרישת הרשאות נוספות, אבל לא יכולה לייצג אפליקציות בתשלום היום.

ההשוואה שלי, כ"קבוצת בקרה" לכאורית, היו העתקים של האפלקיצות המובילות שהורדתי מאתר APKDrawer, כיוון שלא רציתי לעקוף את מנגנון האבטחה של Google Play.

לצורך קבלת "קבוצת בקרה אולטימטיבית", השתמשתי ב-Humble Bundle.

כלומר, היו לנו 6 קבוצות שונות: Kat.Ph, Pirated-156, Paid-70, Top-30, 31, 2010-378 ו-Humble-9. לכל אחת מהקבוצות היו מאפיינים שונים ותכונות שונות.

לאחר ההורדה, הלכתי ובדקתי את ההרשאות האנדרואיד כדי לברר מי מהן בעייתיות. התסריט שכתבתי בוחן כל אחת מחבילות הAPK ומייצר פלט עבור כל אפליקציה עם מספר ההרשאות שהיא דורשת ועם מספר ההרשאות הבעייתיות. לצורך כך השתמשתי ב-APK-Tool, כלי להנדסה לאחור שניתן ברשיון חופשי. באמצעות "aapt -d permissions foo.apk" אתה יכול לקבל את הפלט הרצוי לבד.

כשהסקריפט רץ, הוא מייצר קבצי טקסט, ומעבד אותן לצורך ספירת הנתונים הרלוונטים. אתם יכולים לתפוס את הנתונים והסקריפט כאן, וגם לעיין בטבלה שהוזכרה קודם כדי לקבל את הנתונים מעובדים.

לאחר מכן, השתמשתי בנתונים כדי לבדוק האם אפליקציות בתשלום דורשות הרשאות פחות בעייתיות, ולראות האם האפליקציות המובילות (קבוצת הבקרה שלנו) דורשות פחות הרשאות.

הסקריפט מיועד לבדוק את כמות ההרשאות הבעיתיות, והוא בהחלט לא אופטימאלי. זה כמו לתת לעורך דין לכתוב קוד (היי, זה בדיוק מה שזה). לפני שאתם מריצים, ודאו שאין רווחים בשמות הקבצים, ואם יש, תשתמשו ב* *'name 'y/ /_/* כדי להחליף אותם. אני גם לא אחראי לכל תוצאה מהרצת הסקריפט, לרבות אם זומבים יפלשו לבית שלכם ויאכלו את המח של בת הזוג שלכם. תעשו את זה באחריותכם בלבד.

ממצאים

אם תסתכלו לאורך ציר הזמן, ולא רק חינם מול תשלום, אפליקציות בתשלום מ-2010 דורשות הרבה פחות הרשאות בממוצע. מממוצע של 5.4 הרשאות בעייתיות בקבוצת הביקורת שלנו, ל-0.53 מהמם (שימו לב, שאלה לא בדיוק היחסים שהייתי רגיל אליהם כשהשוותי את האפליקציות קודם). קבוצת הביקורת גם קיבלה תוצאה גבוהה יותר מאשר קבוצת ה-70 בתשלום (שכללה גם משחקים וגם אפליקציות), ומאשר קבוצת ה-156 הפיראטית, שקיבלו 4.67 ו-3.39 בהתאמה.

כאשר השוותי את הקובץ מ-Kat.ph, שהכיל 31 משחקים בתשלום, מספר ההרשאות הבעייתיות ירד ל-1.48, כמעט רבע מאשר ההרשאות הדרושות באפליקציות הפופולריות, וכשליש מהממוצע של האפליקציות הפיראטיות, וחצי מהתוצאה שהתקבלה עבור החבילה הפיראטית.

כקבוצת ביקורת עילאית, השתמשתי ב-Humble Bundle; זו חבילה שכוללת תשעה משחקים, מאוד פופולארים כמו Plants vs. Zombies ו-The Room. החבילה עצמה נמכרת כאשר ההכנסות הולכות לצדקה. מתוך היכרות עם המטרה הטובה, המשחקים ניתנים ללא הגנות נז"ק (DRM). כפי שציפיתי, הממוצע בחבילה היה 0.88 הרשאות לאפליקציה, כאשר רוב ההרשאות (למעט אחת שדרשה לדעת מהן האפליקציות האחרות שרצות) היו הרבה פחות בעייתיות מאשר אותה אפליקציה שניתנת בתשלום בקבוצת הביקורת.

עוד דבר מעניין, הוא שככל שהזמן עבר, רמת ההרשאות התגברה (מ-2010 ועד היום). אנו יכולים לראות שאפליקציות פופולריות דורשות יותר הרשאות בעייתיות; זה עשוי להיות עקב הכללה של יותר אפליקציות (כמו מצלמות, אפליקציות למעקב אחרי הטלפון וכדומה), ביחס למשחקים, אבל יכול להיות גם שהאפליקציות הפחות פופולריות (ובתשלום) דורשות פחות הרשאות.

דאגה משמעותית היא הגישה לנתוני מיקום. נתוני מיקום התבקשו בחלק משמעותי מדי מהאפליקציות. אם נסתכל הטבלה הבאה, נבין עד כמה:

Pirated-156	Top-70	Popular	Access Type
15%	19%	43%	Coarse GPS
16%	27%	23%	Fine GPS

כמו כן, בין 13% ל-17% מהאפליקציות מבקשות לייצר שיחות מטעמך. האפליקציות הפופולריות גם מבקשות יותר גישה לרשימת האפליקציות הרצות ברקע (שליש, לעומת רבע או שמינית, בהתאם לקבוצה) ודורשות יותר גישה לאנשי הקשר שלך (30%, לעומת 27% או 20%, בהתאם לקבוצה).

העניין הוא שהמשחקים יכולים להיות מפותחים ללא אותן הרשאות בעייתיות. במשחק Plants vs. Zombies שהופיע ב-Humble Bundle, הפסיקו לעדכן את האפליקציה בגלל שינוי במשחק שכלל רכישות בתוך האפליקציה. מארגני Humble Bundle האמינו שהוא עשוי להיות לא הולם מעתה לחבילה, ולכן עצרו את חבילת העדכונים בצורה זמנית.

מספר רב של אפליקציות דורש הרשאות רק כדי לספק פרסום, וזו עשויה להיות הסיבה מדוע יותר הרשאות נדרשות באפליקציות הפופולריות. כפי שראינו, פרסום אגרסיבי דורש יותר הרשאות ודורש יותר משאבי מערכת.

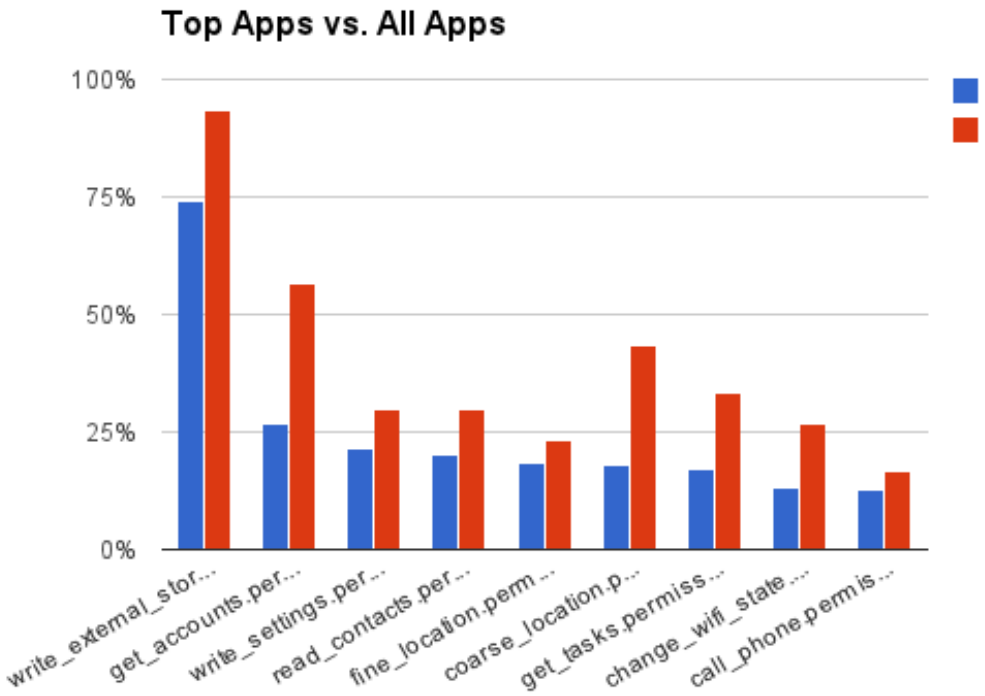
אם אנחנו מסתכלים על המספרים בגדול (בניכוי חבילת 2010) אנו רואים ממוצע של 3.625 הרשאות בעייתיות לאפליקציה.

ההרשאות הבעייתיות ביותר הן: (1) כתיבה לאחסון חיצוני (75% מהאפליקציות), אשר באמת דרוש לשמירה של ניקוד והגדרות; (2) קבלת רשימה של החשבונות על המכשיר, שעשוי היה להיות בסדר כדי לאמת את זהות רוכש האפליקציה, אבל זה גם נותן גישה לזהות חשבון ה-Google שלך למפתח האפליקציה. (27% מהאפליקציות); (3) שינוי של מאפייני המערכת. (26% מהאפליקציות). זה אומר שבעצם האפליקציה יכולה לשנות את עמוד הבית שלך, רקע של שולחן העבודה, מנוע החיפוש , או לשנות הגדרות אחרות כדי להציג פרסומות; (4) גישה לאנשי הקשר שלך (20% מהאפליקציות). זה אומר

שחמישית מהאפליקציות רוצות גישה לאנשי הקשר שלך; זה יכול להיות כדי לשתף איתם תמונות מצחיקות, אבל גם כדי לשלוח את הפרטים שלהם לאנשים אחרים; (5) לגשת למיקום הגס שלך (18% מהאפליקציות), שזה עשוי לעזור לך בניווט, אבל גם להציג לך פרסומות מבוססות מיקום.

אם תשוו את הנתונים לאלה שהגיעו מה-Humble Bundle או האפליקציות הפיראטיות מ-Kat.ph, תראו שהגישה למיקום לא כל כך פופולרית באפליקציות בתשלום (13% בחבילה של Kat.ph ואפס ב- Humble Bundle). ושאף אחת מהאפליקציות האלה לא רוצה לשנות את מאפייני המערכת שלכם, ורק אחת רוצה גישה לאנשי הקשר (בהשוואה ל-9 בקבוצת 30 האפליקציות הפופולריות).

בטבלה הזו אפשר לראות השוואה של קבוצת "כל האפליקציות" מול 30 המובילות בהרשאות שנדרשות:



עוד דבר שיש לקחת בחשבון הוא שההרשאה הבעייתית הבאה אחרי גישה לנתוני מיקום היא גם גישה למיקום מקורב (18%, 53 אפליקציות); זה אומר שיש קצת חפיפה, ובטח שלא חפיפה מלאה, כך שככל הנראה הגישה למיקום שלך הוא בטח פופולרי לפחות כמו גישה לזהות החשבון. זה גם אומר שהרבה זרים מקבלים את נתוני המיקום שלך.

מסקנות

המסקנה העיקרית היא שאם יש שתי חלופות, ואחת דורשת יותר הרשאות מהשניה, אנו צריכים לשקול מחדש את המחיר של "חינם". כאשר המפתח יכול לקבל 0.26$ עבור אפליקציה בחינם, ואנחנו יכולים לקנות את הגרסא בתשלום ב-1$, צריך לשאול אם לא שווה לשלם 0.74$ רק כדי לא לתת לרשת פרסום ולמפתח מידע על מי אתה, להחליף את מסך הנעילה שלך, לשלוח הודעות SMS בשמך לשירותי פרמיום, לרוקן את הסוללה שלך, או להחליף את הגדרות הגלישה באינטרנט.

אנחנו יכולים לראות שבין האפליקציות הפופולריות ביותר, והמכניסות ביותר, יש קבוצה מובהקת של "חינם", אבל הן דורשות יותר הרשאות. ההרשאות האלו הן המחיר של "חינם".

Subverting BIND's SRTT Algorithm: Derandomizing NS Selection

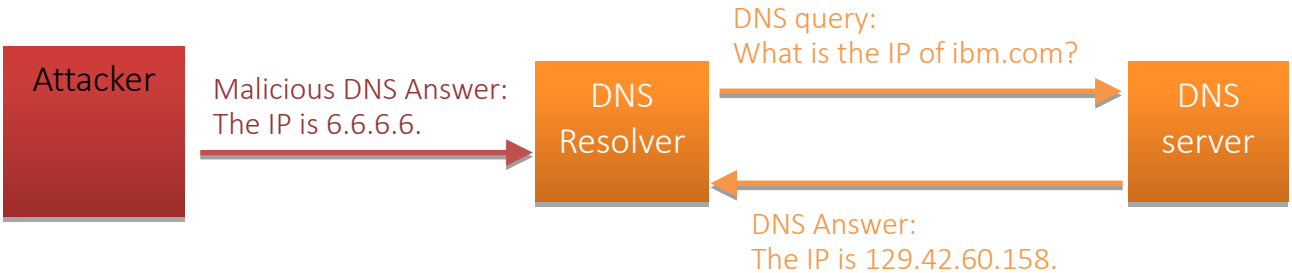
מאת רועי חי

הקדמה

במסמך זה אתאר מתקפה חדשה על שרת ה-DNS הנפוץ בעולם, BIND. המתקפה הוצגה בכנס USENIX WOOT '13. פרסום המתקפה נעשה בתיאום מלא עם ISC, הארגון שאחראי על BIND.

מבוא

DNS הוא פרוטוקול שעובד בדרך-כלל מעל UDP. אופן-הפעולה של התוקף ב- Off-path (Blind) DNS attack poisoning הוא לאלץ Name Server (NS) או DNS Resolver מסויים לשלוח בקשה, ולהחזיר תשובה זדונית כ-NS אליו הוא פונה, לפני שמגיעה התשובה המקורית. אנו מניחים שהתוקף אינו רואה את המידע. אחרת כללי המשחק משתנים, ופרוטוקול ה-DNS פשוט אינו מסוגל להתמודד עם מצב זה ללא DNSSEC.

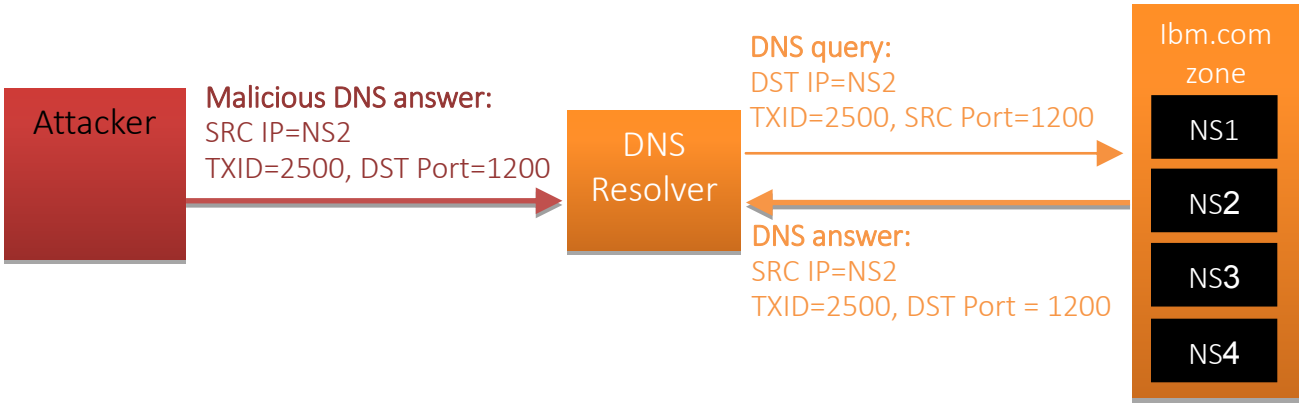


המצב המתואר למעלה הוא טריוויאלי מבחינת התוקף, כי הוא יכול לתזמן את המתקפה כך שתמיד או ברוב המקרים ינצח את השרת המקורי. לכן, כל בקשת DNS מכילה מזהה ייחודי, nonce, המורכב משלושה פרמטרים:

1. שדה ה-TXID ב-DNS header (16 ביטים רנדומליים).
2. UDP source port (16 ביטים רנדומליים בקירוב).
3. IP destination address.

מידת הרנדומליות של הפרמטר השלישי תלויה במספר שרתי ה-DNS האחראים על הדומיין המתושאל ובאלגוריתם הבחירה. למשל, אם אלגוריתם הבחירה הוא פשוט להגריל שרת באופן אחיד (לא המצב ב-BIND, כפי שנראה בפירוט בהמשך), ויש 8 שרתי DNS האחראים על הדומיין, אז נוספים 3 ביטים רנדומליים ל-nonce.

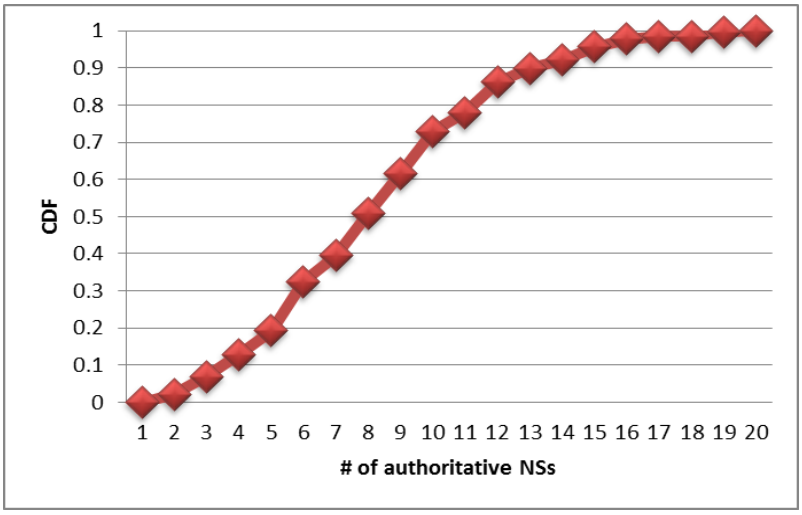
כדי להצליח בתקיפה התוקף חייב לנחש נכון את ה-nonce, ולכן מידת הרנדומליות שלו משפיעה באופן ישיר על זמן התקיפה. בפועל, תקיפת DNS (מוצלחת) נראית באופן הבא:



מה נשאר לחקור?

הנחת העבודה שלנו הייתה שמידת הרנדומליות של שדה ה-TXID וה-UDP Source Port במימושים נפוצים נחקרה רבות בעבר. לכן החלטנו להתמקד בשדה ה-IP destination address.

על מנת להסיק את הפוטנציאל של שדה זה (כלומר, באופן אופטימלי, כמה ביטים רנדומליים הוא יכול להוסיף ל-nonce), הורדנו את ה-root zone file (http://www.internic.net/domain/root.zone), ויצרנו את ה-CDF graph הבא:



גרף זה מלמד כי בערך למחצית מה-Top-Level Domains (TLDs) יש למעלה מ-8 שרתי DNS אחראיים ולכל ה-TLDs יש פחות מ-20 שרתי DNS אחראיים. עובדה זו מלמדת כי אפילו אם אלגוריתם הבחירה הוא רנדומלי לחלוטין (כלומר הוא מגריל את שרת ה-DNS באופן יוניפורמי), אז מספר הביטים הרנדומליים הנוספים הוא די נמוך.

לכן, אם נצליח לחזות את שרת ה-DNS הנבחר לא תתאפשר מתקפת DNS חדשה, אך כן נוכל לקצר מתקפות קיימות או חדשות אם תמצאנה. בנוסף לכך, אם התוקף הוא Man-in-the-Middle בין ה-Resolver לשרת DNS מסוים, והוא מסוגל לכפות תשאול מול אותו שרת, אז מתקפת Off-path DNS poisoning הופכת ל-On-path.

אלגוריתם בחירת שרת ה-DNS ב-BIND.

כאמור מבחינת Security, הדבר הכי טוב שה-Resolver יכול לעשות הוא להגריל באופן אחיד את השרת המתושאל. אולם, מבחינת ביצועים הדבר אינו אופטימלי: המוטיבציה של ה-Resolver היא לבחור את השרת אשר יענה לו הכי מהר, כלומר עם ה-Round-Trip Time (RTT) הנמוך ביותר. מכיוון שה-RTT משתנה לעתים תכופות (תלוי בעומסי הרשת, עומס השרת וכד'), *BIND* ממצע את ערכי ה-RTT. הממוצע נקרא Smoothed Round-Trip Time (SRTT).

באופן מפורט יותר, ערך ה-SRTT לכל שרת DNS הוא במיקרו-שניות, ונקבע באופן הבא:

1. **אתחול (Initialize):** הערך ההתחלתי מוגרל בין 1 ל-32.
2. **עדכון (Update):** כאשר מתקבלת תשובה חדשה משרת DNS מסויים, אז ל-BIND יש מידע אודות ה-RTT שלו. לכן האחרון נלקח ומשתקלל עם ה-SRTT הישן. כאשר משקל 0.3 ניתן ל-RTT ומשקל 0.7 ניתן ל-SRTT הישן.
3. **דעיכה (Decay):** על-מנת למנוע הרעבה, כאשר יש מספר שרתי DNS אופציונאליים, אז ערך ה-SRTT של שרתי ה-DNS הבלתי-מתושאלים יורד ב-2% לאחר כל שאילתה.
4. **שגיאה (Error):** כאשר שרת ה-DNS המתושאל מחזיר שגיאה או שלא עונה, ערך ה-SRTT שלו נענש ב-200ms (עם גבול עליון של שנייה).

כל ערכי ה-SRTT נשמרים ע"י BIND ב-cache. הכניסות ל-cache הן כתובות ה-IP של שרתי ה-DNS.

מה נעשה בעבר ומה הוא החידוש שלנו?

באופן כללי אם התוקף יכול להשפיע על ערכי ה-SRTT של שרתים שרירותיים הוא יכול לחזות לאיזה שרת BIND יפנה, כלומר הוא מבצע דרנדומזיציה (derandomization) לאלגוריתם הבחירה :

1. אם התוקף מצליח להוריד את ערך ה-SRTT של שרת מסויים לערך נמוך מערכי ה-SRTT של שאר השרתים האחראים על דומיין כלשהו, BIND יפנה אליו בבקשה הבאה.
2. אם התוקף מצליח להעלות את ערך ה-SRTT של כל השרתים האחראים על דומיין מסויים, למעט שרת בודד, כך שערכי ה-SRTT יהיו גבוהים מזה של האחרון, BIND יפנה לאותו שרת בבקשה הבאה.

בעבר הוצעו שתי מתקפות עיקריות על מנגנון בחירת ה-NS:

1. המתקפה הראשונה [Petr 2009], מנצלת את פעולת ה-Update באלגוריתם ה-SRTT. התוקף פשוט מתחזה לשרת אשר הוא רוצה להוריד את ה-SRTT שלו, ושולח תשובות מהירות בשמו.
2. המתקפה השניה [Herzberg & Shulman, 2012], מנצלת את פעולת ה-Error באלגוריתם ה-SRTT. התוקף מעלה את ה-SRTT של כל שרת אחראי למעט אחד. הוא מנצל IP fragmentation כדי לייצר תשובות שגויות.

נשים לב ששתי המתקפות הן הסתברותיות: התוקף חייב לנחש ערכים מסויימים. לעומת זאת, המתקפה שלנו היא דטרמיניסטית, כלומר היא תמיד עובדת. בנוסף לכך במתקפה שלנו השרת אשר אנו משפיעים על ערך ה-SRTT שלו בכלל לא מודע לתקיפה.

המתקפה

במאמר זה אתאר וריאציה פשוטה של המתקפה שלנו. במסמך המלא ניתן למצוא וריאציות נוספות.

במתקפה זו אנו מנצלים שרתי DNS סגורים, כלומר כאלה שלא עונים על שאילתות על דומיינים שהם אינם אחראיים עליהם. אלו הם **רוב** שרתי ה-DNS בעולם, כאשר התוקף רק צריך להכיר את כתובת ה-IP שלהם. אנחנו נראה כיצד התוקף יכול לנצל אותם על מנת להוריד את ערך ה-SRTT של שרת DNS שרירותי לערך שרירותי כרצונו, וכפי שצויין לעיל, כך הוא יוכל לדעת בוודאות ש-BIND יפנה אליו בשאילתה הבאה.

נניח כי הדומיין הנתקף הוא ibm.com, ויש שני שרתי DNS האחראים עליו: ns1.ibm.com ו-ns2.ibm.com. כמו כן נניח ששני שרתים אלו הם כבר ב-SRTT cache של ה-DNS Resolver הנתקף, וה-

SRTT של ns1.ibm.com גבוה יותר. נראה כיצד ניתן להוריד את הערך של ns2.ibm.com, כדי שבשאילתה הבאה ה-DNS Resolver יפנה בוודאות ל-ns2.ibm.com.

התוקף מצויד בארסנל הבא:

1. רשימה של שרתי DNS סגורים.
2. שרת DNS עליו הוא שולט, לדוגמא .ns.malicious.foo.
3. דומיין אשר השרת הנשלט אחראי עליו, למשל malicious.foo.

המתקפה מתחילה בתשאול ה-DNS Resolver על דומיין אשר השרת הנשלט אחראי עליו, למשל 666.malicious.foo. בסופו של יום ה-Resolver יפנה לשרת ה-DNS עליו התוקף שולט: ns.malicious.foo. אשר יחזיר Delegation, כלומר לא יענה על השאילתה בעצמו אלא יפנה את ה-Resolver לשרתים אחרים. הוא מחזיר את ה-Delegation הבא:

1. רשימה גדולה מאוד של שרתי DNS סגורים אשר לא ב-SRTT cache של ה-Resolver.
2. ns2.ibm.com.

כאשר ה-Resolver יקבל את ה-delegation הוא יכניס את הרשימה של שרתי ה-DNS הסגורים ל-SRTT cache שלו עם ערך נמוך מאוד, בין 1 ל-32, לפי פעולת האתחול באלגוריתם ה-SRTT. לכן הוא יפנה אליהם לפני שהוא יפנה ל-ns2.ibm.com. בשלב הבא הוא יפנה אליהם באופן סדרתי, והם יחזירו לו Query refused (כי הם סגורים).

בכל תשאול של שרת DNS סגור, תתבצע פעולת הדעיכה של האלגוריתם, כלומר ה-SRTT של ns2.ibm.com מוכפל בפקטור 0.98 (יורד ב-2%) לאחר 30 שניות ה-DNS Resolver יבצע timeout לבקשה. לכן בסוף התהליך ערך ה-SRTT של ns2.ibm.com יהיה 0.98^n מערכו המקורי, כאשר n הם מספר שרתי ה-DNS הסגורים שתושאלו עד ה-timeout. לפיכך הצלחנו להוריד את ערך ה-SRTT של ns2.ibm.com באופן דטרמיניסטי לערך שרירותי!

כיצד ניתן לתקן את הפגיעות?

נשים לב שהמתקפה מנצלת את העובדה ששרת DNS זדוני יכול להשפיע על ערך ה-SRTT של שרת DNS שרירותי. שורש הבעיה הוא שה-SRTT cache הוא גלובאלי. ניתן לתקן את הבעיה ע"י חלוקה של ה- SRTT cache, למשל לפי ה-zone המתושאל.

קישורים לקריאה נוספת

- המאמר המקורי מ-'13 USENIX WOOT: http://bit.ly/19Gy734
- המצגת מ-'13 USENIX WOOT: http://bit.ly/15AdyZ4
- הדיווח של ISC על הפגיעות: http://bit.ly/1beaBfJ

דברי סיום

בזאת אנחנו סוגרים את הגליון ה-45 של Digital Whisper. אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

אנחנו מחפשים כתבים, מאיירים, עורכים ואנשים המעוניינים לעזור ולתרום לגליונות הבאים. אם אתם רוצים לעזור לנו ולהשתתף במגזין Digital Whisper - צרו קשר! בנוסף, אנחנו עדיין מוסרים חתול מדהים בשם צ'ייסר, מי שמעוניין - שישלח מייל!

ניתן לשלוח כתבות וכל פניה אחרת דרך עמוד "צור קשר" באתר שלנו, או לשלוח אותן לדואר האלקטרוני שלנו, בכתובת editor@digitalwhisper.co.il.

על מנת לקרוא גליונות נוספים, ליצור עימנו קשר ולהצטרף לקהילה שלנו, אנא בקרו באתר המגזין:

www.DigitalWhisper.co.il

"Talkin' bout a revolution sounds like a whisper"

הגליון הבא ייצא ביום האחרון של חודש אוקטובר.

אפיק קסטיאל,

ניר אדר,

30.09.2013