שבירת האנונימיות באפליקציית Blindspot

מאת ינון שקדי

הקדמה

אפליקציית Blindspot הינה אפליקצייה לשליחת מסרים אנונימיים. Blindspot החלה את דרכה בסוף 2015, בליווי מסע פרסום אגרסיבי הכלל שלטי חוצות באיילון, וזכתה ליותר מחצי מיליון הורדות.

האפליקצייה עוררה הרבה רעש, וספגה ביקורות רבות בתקשורת ובמדיה החברתית, מתוך פחד שהיא תגרום לאלימות מילולית בקרב ילדים ונוער - משתמשיה העיקריים.

ביצעתי את בדיקת החדירות על מנת להבין עד כמה האפליקצייה מאובטחת ושומרת על פרטיות לקוחותיה, ומתוך רצון להשתפשף בתחום ה-PT ל-Android.

לאחר בדיקה קצרה, הצלחתי למעשה לבטל את אנונימיות האפליקצייה, ולהיות מסוגל לגלות את זהות המשתמשים אשר שלחו לי הודעות.

המאמר מיועד לאנשים עם רקע טכני בבדיקות חדירות, עם ידע בסיסי ב-Android. כדאי להכיר את המושגים: Java Bytecode, smali, decompilation, tunneling, websocket.

<u>אתגרים שהאפליקצייה מציבה בפנינו</u>:

- אובפסקציה חזקה של הקוד.
- האפליקצייה עושה שימוש בפרוטוקול WebSocket שלא עובר דרך ה-Proxy שמוגדר ברמת ה-Android.
- חתימה של כל הודעת HTTP/s באמצעות Oauth.

<u>מבנה כללי</u>:

האפליקצייה מאפשר ליצור צ'אט עם אדם מרשימת אנשי הקשר, אשר גם הוא משתמש רשום. ביצירת שיחה עם משתמש, נפתח צ'אט חדש, המאפשר לשלוח הודעות טקסט ומדיה.

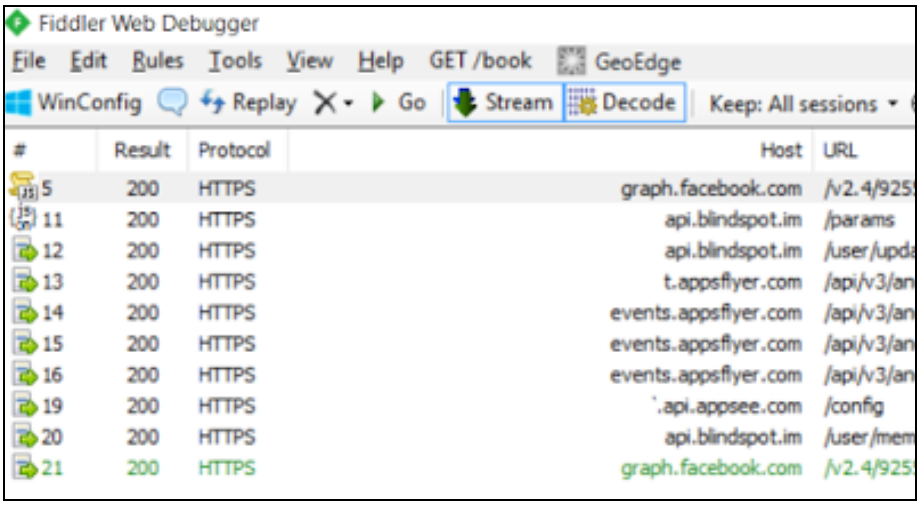
ניתוח ראשוני - תעבורה

אחד הכלים שיעזרו לנו בתור בודקי חדירות לאפליקציות Android, הוא היכולת לצפות ולשנות את התעבורה של האפליקצייה. באמצעות כלי זה, ניתן לחפש חולשות אפליקטיביות על השרת, ולהבין מה קורה בצד הלקוח בלי להתעסק עם הקוד יותר מדי.

בתור שלב ראשון, נוכל לצפות בתעבורה מסוג:

- **שאילתות DNS** - בכדי להבין עם איזה שרתים האפליקצייה מתקשרת. ב-Android אין אפשרות לבצע flush dns, ולכן לא בכל הפעלה של האפליקצייה יהיה אפשר לראות שאילתות dns חדשות. העניין עלול להיות מעיק, ולכן כדאי להדליק Wireshark לפני ההפעלה הראשונית. (ה- DNS Cache מתנקה כל 10 דקות ולפעמים גם Reset למכשיר יעזור).
- **תעבורת HTTP/s** - בדרך כלל צפייה בתעבורת HTTP ו-HTTPS זה עניין פשוט, מכיוון שניתן להגדיר Proxyברמת ה-Android לתעבורה זו.

על ידי לחיצה ארוכה על שם ה-Wifi שאליו אתם מחוברים ואז "Modify Network" תוכלו להגדיר HTTP Proxy, ולהשתמש ב-Web Proxy המועדף עליכם על מנת לצפות בתעבורה. אני אוהב להשתמש ב-Fiddler.



כדאי לשחק קצת עם האפליקצייה, לשלוח סוגים שנים של הודעות ולראות מה קורה מאחורי הקלעים. המאמר לא נוגע בזה, אבל שימו לב שבאפליקציית Blindspot, במהלך תהליך ההזדהות, הדף בכתובת:

```
api.Blindspot.im/join/activate
```

מחזיר json המכיל מפתח ו-token שבאמצעותם האפליקצייה חותמת כל הודעה שנשלחת לשרת. החתימה נעשית באמצעות OAuth ומתווספת ל-authorization header בבקשות עתידיות. ניתן לחתום הודעות עצמאית על ידי שירותי Online או להיות Leet-ים ולכתוב תוסף ל-Fiddler שעושה זאת לבד.

פרוטוקולים נוספים - לאחר קצת משחק עם האפליקציה, ניתן להבין כי :

- ב-Fiddler לא ניתן לראות את ההודעות ששולחים / מקבלים ממשתמשים אחרים באפליקצייה.
- מתבצעת בקשת DNS עבור chat.blindspot.im ואנחנו לא רואים את הכתובת הזו ב-Fiddler.

ניתן להניח שההודעות למשתמשים נשלחות לכתובת chat.Blindspot.im בפרוטוקול שאינו HTTP/s ולכן אנחנו לא רואים אותן ב-Fiddler. מכיוון שהתקשורת מול אותו שרת נעשית בפורט 443, הגיוני שהפרוטוקול שנעשה בו שימוש הוא WebSocket. חיפוש בקוד java של המחרוזת 'websocket' מאמת את החששות.

ניתוח קוד האפליקצייה

לאחר שקיבלנו רושם ראשוני על ידי צפייה בתעבורת האפליקצייה, נרצה להבין את הקוד ברמה בסיסית. הדרך הנוחה ביותר, היא לעשות De-compilation לקובץ ה-DEX. תהליך ה-Decompliation: בתוך קובץ ה-apk ניתן למצוא קובץ dex אשר מכיל את קוד האפליקצייה. את הקובץ הזה ניתן להמיר לקובץ jar, באמצעות הכלי 'dex2jar'. את קובץ ה-jar נפתח עם ה-Java Decompiler המועדף עלינו.

לצערי, אין JAVA Decompiler אחד שעושה את העבודה בצורה מושלמת (בניגוד ל-Reflector ב-.NET), ולכן, בהרבה מקרים הדרך הנכונה לעבוד, תהיה באמצעות מספר Decompiler-ים, ועם קוד ה-SMALI במקביל. התוכנה 'Bytecode Viewer' יכולה להקל עליכם בביצוע המשימה.

כמו-כן, ניתן להעזר בקובץ ה-Manifest על מנת לראות את שמות ה-Packag-ים שמוגדרים עבור כל Service,Activity ו-Broadcast Receiver.

מכיוון שהקוד עבר אובפוסקציה, קשה לעקוב אחרי ה-Flow. אפשר למצוא מחלקות מעניינות גם לפי חיפושים של מחרוזות. לדוגמא:

- מחלקה המטפלת בתקשורת תכיל אובייקט מסוג socket.
- מחלקה המטפלת בהצפנה ופענוח תכיל פונקציית decrypt.

כפי שאתם וודאי יודעים, בדרך כלל אין טכניקת קסם בשביל להבין קוד שעבר אובפוסקציה. לפני שמתחילים לשוטט בקוד, כדאי להבין מה המטרה (אם רוצים לבדוק חולשות על השרת, לא כדאי להיכנס לקוד של מחלקות גרפיות).

כאמור, אפליקציית Blindspot משתמשת בפרוטוקול WebSocket שלא עובר דרך ה-Proxy שהגדרנו ברמת מערכת ההפעלה, ולכן מה שנרצה לעשות זה לבנות פאטץ' שיגרום לתקשורת לעבור דרך Web Proxy אשר תומך ב-WebSocket.

כמה מילים על WebSocket, Proxies, HTTP Tunneling ומשמעות החיים:

WebSocket:

- פרוטוקול חדש יחסית, ליצירת Socket פשוט מעל דפדפן.
- חוסך הרבה משאבים, כמו HTTP Headers וריבוי TCP Handshakes ב-HTTP.
- יעיל מאד כשרוצים לראות נתונים בזמן אמת.
- התקשורת הינה דו-כיוונית, וה-TCP Connection נשאר פתוח כל עוד המשתמש גולש באתר.
- סיומות wss:// לחיבור מוצפן ו-ws:// לחיבור רגיל
- Handshake: הקשר היחידי בין WebSocket לבין HTTP הוא ה-Handshake.

זה קצת טריקי. אנסה להסביר את התהליך בצורה ברורה:

1. הדפדפן יצור socket רגיל מול השרת, אשר הבקשה הראשונה בו תהיה תואמת HTTP
2. השרת יחזיר תשובה תואמת HTTP
3. ה-socket נהיה Raw socket לכל דבר ועניין.

בקשה:

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbI
Sec-WebSocket-Protocol: cha
Sec-WebSocket-Version: 13
Origin: http://example.com
```

תשובה:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUk
Sec-WebSocket-Protocol: chat
```

שימו לב ל-Header ה-Upgrade.

חשוב להבין כי מכיוון ומדובר בסופו של דבר ב-Raw Socket, פרוטוקול ה-WebSocket אינו מודע כלל ל-HTTP או HTTP Proxy, ולכן זה לא טריוויאלי שהוא יעבור בקלות דרכו.

HTTPTunneling + Proxies Servers:

HTTP Tunneling הינה טכניקה ל-Tunneling של פרוטוקולים שונים תחת HTTP. מרבית שרתי ה-Proxy כיום תומכים בה, כמו גם ה-Web Proxy שלנו (Fiddler, Burp).

תהליך יצירת ה-Tunnel הולך כך: הדפדפן שולח בקשת HTTP עם פקודה מסוג CONNECT לשרת ה-Proxy. הבקשה תכיל את הכתובת של שרת היעד. שרת ה-Proxy יחזיר תשובה מסוג 200, ומשלב זה יעביר את כל התקשורת מהלקוח ישירות לשרת. זו למעשה דרך לממש TCP Proxy רגיל מעל HTTP Proxy.

עם ההבנה של שני הנושאים הללו, המסקנה המתבקשת היא שהדרך לבצע Proxy של WebSocket, היא באמצעות יצירת HTTP Tunnel מול שרת ה-Proxy באמצעות פקודת CONNECT, ולאחר מכן להתחיל את ה-WebSocket Handshake דרך אותו ה-Tunnel.

כמו-כן, דפדפנים חדשים עושים את זה לבד אם הם מזהים HTTP Proxy מוגדר.

לאחר שיטוט בקוד האפליקציה, ניתן לראות שנעשה דבר שבעיניי נראה קצת מוזר: נעשה שימוש ב-Socket רגיל על מנת ליצור תקשורת של WebSocket, במקום להשתמש בספריות java שמציעות מעטפת לפרוטוקול.

קוד יצירת ה-Socket במחלקה dbq:

```
try {
    Socket a;
    int a2 = ckh.a(port, z);
    if (ckh.b.d != null) {
        a = ckh.a(host, a2, z, i);
    } else {
        a = ckh.a.a(z).createSocket();
        a.connect(new InetSocketAddress(host, a2), i);
    }
    if (port >= 0) {
        host = host + ":" + port;
    }
    if (rawQuery != null) {
        rawPath = rawPath + "?" + rawQuery;
    }
    dbp.c = new ckb(ckh, z, userInfo, host, rawPath, a, i);
```

קוד שליחת ה-Handshake תואם ה-HTTP במחלקה ckb:

```
StringBuilder append = new StringBuilder("GET ").append(cjl.c)
.append(" HTTP/1.1\r\nHost: ")
.append(cjl.b)
.append("\r\nConnection: Upgrade\r\nUpgrade: websocket\r\nSec-WebSocket-Version:
.append(cjl.d)
.append("\r\n");
cjl.a(append, "Sec-WebSocket-Protocol", cjl.e);
cjl.a(append, "Sec-WebSocket-Extensions", cjl.f);
cjl.a(append, cjl.g);
if (!(cjl.a == null || cjl.a.length() == 0)) {
    append.append("Authorization: Basic ").append(cjk.a(cjl.a)).append("\r\n");
}
b.write(cjn.a(append.append("\r\n").toString()));
```

מימוש זה עושה לנו חיים קצת קשים, מכיוון שאין לנו פונקציה מוכנה שמבצעת את התהליך ה-Tunneling שהוסבר קודם, ונצטרך לממש את זה בעצמנו. אך למזלכם, במקרה מצאתי מימוש זה כבר בתוך קוד האפליקציה (כנראה שאריות Debugging של המפתחים), תחת המחלקה ckh.

כמו-כן, ניתן לראות בבירור שבמחלקה dbq קיים תנאי if שמוביל את האפליקצייה ל-Flow בו הפונקציה הזו נקראת:

```
try {
    Socket a;
    int a2 = ckh.a(port, z);
    if (ckh.b.d != null) {
        a = ckh.a(host, a2, z, i);
    } else {
```

בהפעלה רגילה של האפליקצייה התנאי לא מתקיים.

מתחילים לפצ'פץ'!

לפני קריאת פרק זה, מומלץ לעיין במדריך ה-SMALI שכתבתי:

http://inonsec.blogspot.co.il/2016/11/v-behaviorurldefaultvmlo_34.html

הדפסה ללוג:

בתור התחלה, נרצה לבנות פאטץ' שכל מה שהוא יעשה, זה להדפיס ללוג הודעות plain text, לפני שהן מוצפנות ומעוברות בתקשורת.

דוגמא לפונקציה שמטפלת בהודעות מסוג מסוים, היא הפונקציה a במחלקה cki:

```
public final cki a(byte[] arrby) {
    byte[] arrby2 = arrby;
    if (arrby != null) {
        arrby2 = arrby;
        if (arrby.length == 0) {
            arrby2 = null;
        }
    }
    this.g = arrby2;
    return this;
}
```

היא אינה מטפלת בכל הסוגים של ההודעות, אך בשביל הדוגמא היא מספיק טובה.

כעת נרצה לבנות טלאי, שידפיס ללוג את הערך של arrby בעת קריאה לפונקציה זו. מכיוון ש-arrby הוא מערך של בתים, נצטרך לעשות לו המרה ל-String לפני ההדפסה ללוג. שני עקרונות מרכזיים בכתיבת טלאים:

- כדאי לשנות כמה שפחות קוד קיים - אם מתאפשר, מומלץ להוסיף פונקציה חדשה, ולהוסיף לקוד המקורי רק קריאה לאותה פונקציה.
- אם אתם לא חזקים ב-SMALI, כדאי להסתמך כמה שיותר על קוד SMALI אשר קומפל מ-Java שאתם כתבתם. ניתן להמיר קוד Java ל-SAMLI באמצעות Plugin ייעודי ל-Intellij.

הפונקציה שנכתוב תראה ככה:

```
public void lala(byte[] arrby){
    String s = new String(arrby);
    Log.e("Output message : ",s);
}
```

ולאחר המרה ל-Smali:

```
.method public lala([B)V
    .registers 4

    .prologue
    .line 13
    new-instance v0, Ljava/lang/String;

    invoke-direct {v0, p1}, Ljava/lang/String;-><init>([B)V

    .line 14
    const-string v1, "Output message : "

    invoke-static {v1, v0}, Landroid/util/Log;->e(Ljava/lang/String;Ljava/lang/String;)I

    .line 15
    return-void
.end method
```

את קוד ה-SMALI נוסיף לקוד המחלקה cki.class, בתוך הסגמנט של virtual methods. לאחר מכן, בתחילת הפונקציה cki.a(byte[]) נוסיף קריאה לפונקציה שלנו:

```
.method public final a([B)Lcki;
    .locals 1

    .prologue
    .line 487
    if-eqz p1, :cond_0

    invoke-virtual {p0, p1}, Lcki;->lala([B)V
```

נבנה מחדש את קובץ ה-APK, נחתום עליו ונתקין ב-android.

בשלב הבא, נרצה לצפות ב-log. מומלץ למחוק את ה-log הישן לפני, באמצעות הפקודה:

```
adb logcat -c
```

לאחר מכן נשתמש בפקודה adb logcat על מנת לצפות ב-log בזמן אמת. כמו-כן, ניתן לסנן סוגים של אירועים. לדוגמא, סינון של אירועים מתחת לרמת חומרה של Error:

```
adb logcat E:*
```

```
E/Output message : ( 3962): ▯▯local_messa
E/Output message : ( 3962): ▯nmid▯▯▯ 761d
E/Output message : ( 3962): ▯▯mt▯▯rmid▯▯▯
```

שינוי אובייקט ה-Socket:

לאחר הניתוח של הקוד בשלב 2, הבנו שאנחנו רוצים לעשות שני שינויים בכדי לגרום לפרוטוקול ה-WebSocket לעבור דרך ה-Proxy שלנו:

יצירת Proxy Socket:

נגרום למחלקה dbq ליצור את ה-socket באמצעות הפונקציה a (string,int,Boolean,int) במחלקה ckh שצוינה קודם, ועושה בשבילנו את ה-HTTP Tunneling מול שרת ה-Proxy. הדרך הפשוטה לעשות זאת, היא לשנות את התנאי שקובע האם יווצר socket רגיל או עם פרוקסי.

```
if-eqz v0, :cond_16

.line 4573
invoke-virtual {v1, v4, v13, v2, v7}, Lckh;->a(Ljava/lang/String;IZI)Ljava/net/Socket;

move-result-object v6
```

```
:cond_16
iget-object v0, v1, Lckh;->a:Lcjx;

invoke-virtual {v0, v2}, Lcjx;->a(Z)Ljavax/net/SocketFactory;

move-result-object v0
```

ניתן לראות שמתבצעת פקודת if-eqz שבודקת האם הרג'יסטר v0 שווה לאפס. במידה וכן, תתבצע קפיצה בקוד ל-label שנקרא cond_16 שם יווצר socket רגיל. במידה ולא, תתבצע קריאה ל-ckh.a היוצרת socket עם Proxy. השינוי יהיה מזערי, ורק נשנה את הפקודה if-eqz לפקודה if-nez, וכך נדאג שבכל הרצה של האפליקצייה תתבצע קריאה ל-ckh.a.

<u>הגדרת כתובת IP ופורט ל-Proxy:</u>

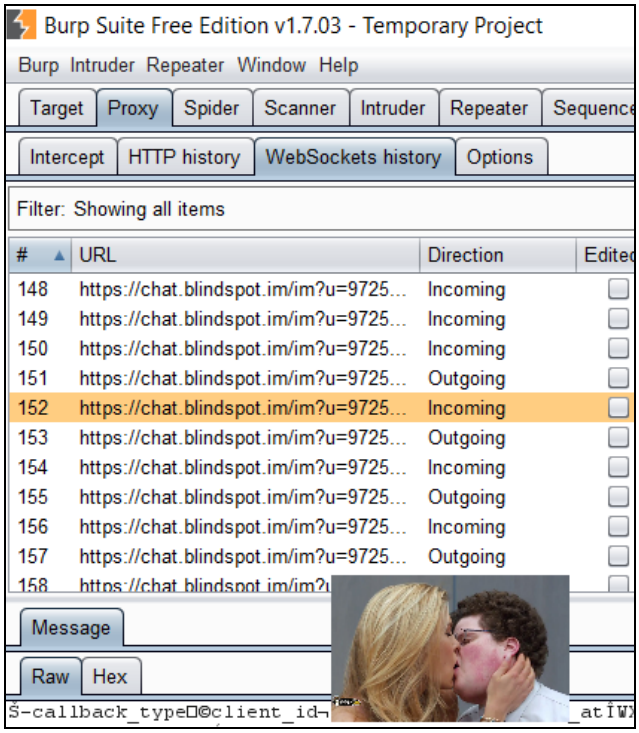
נגדיר במחלקה ckh את הכתובת IP והפורט של ה-Web Proxy שלנו. הדרך הפשוטה לבצע זאת, היא לשנות את הקוד לפני שמתבצעת קריאה לפונקצית ה-Constructor, InetsocketAddress(string,int), לדרוס את הערכים שנמצאים ברג'יסטרים הנשלחים אליה ולהחליף אותם בערכים משלנו:

```
118     const-string v4, "10.0.3.2"
119
120     const/16 v1, 0x22b8
121
122     invoke-direct {v2, v4, v1}, Ljava/net/InetSocketAddress;-><init>(Ljava/lang/String;I)V
```

הכתובת 10.0.3.2 מייצגת את ה-HOST ב-Genymotion, ו-0x22b8 מייצג את המספר 8888 בתקן IEEE 754 (הפורט של Fiddler).

עם כל אהבתי ל-Fiddler, לצערי Burp עובד בצורה טובה יותר עם WebSocket, ולכן באופן חד פעמי אני נאלץ להשתמש בו. כאשר נפעיל את האפליקציה עם ה-Patch החדש, ונפעיל במקביל Burp שמאזין על הפורט המתאים, נוכל לראות את תעבורת ה-WebSocket תחת History --> Websockets Proxy.

כמובן שההודעה הראשונה שמתקבלת מהשרת ברגע שמשתמש אחר מתחיל צ'אט אנונימי, מכילה את מספר הטלפון שלו בשדה client_id:



סיכום

מבחינה אבטחתית, החולשה עצמה אינה מאד מעניינת - השרת שולח את מספר הטלפון של לקוח A, אל לקוח B, בעת תהליך יצירת שיחה "אנונימית". במהלך בדיקת החדירות נתקלנו במספר גורמים אשר הקשו עלינו, אך הם הפכו את התהליך למעניין יותר.

ישנן דרכים יותר נוחות וגנריות להגיע לתוצאות אליהן הגעתי, אך מטרת המאמר הינה ללמד קונספטים כלליים הנוגעים לבדיקת חדירות ב-Android, ולא לבצע את התהליך בצורה היעילה והמהירה ביותר.

- החולשה דווחה בתאריך 21/11/2016
- החולשה תוקנה בין התאריכים 23-29/11/2016