digital whisper Technologic Papers

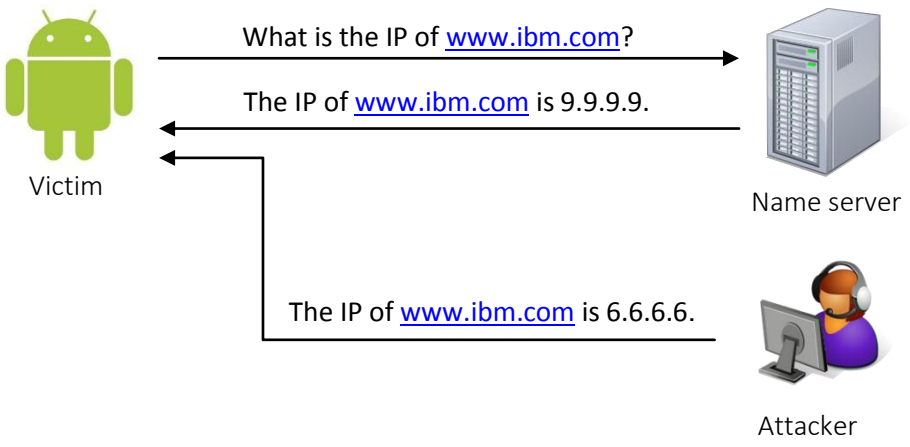
Android DNS poisoning: Randomness gone bad

מאת רועי חי

במאמר זה נציג פגיעות מעניינת שהתגלתה על ידנו ב-DNS Resolver באנדרואיד, הפגיעות מאפשרת לבצע מתקפת DNS poisoning על המכשיר. נתחיל מהקדמה קצרה על DNS ועל מתקפת DNS poisoning. נמשיך עם תיאור מפורט של מנגנון ה-DNS באנדרואיד, ונקנח עם הפגיעות עצמה והשלכותיה.

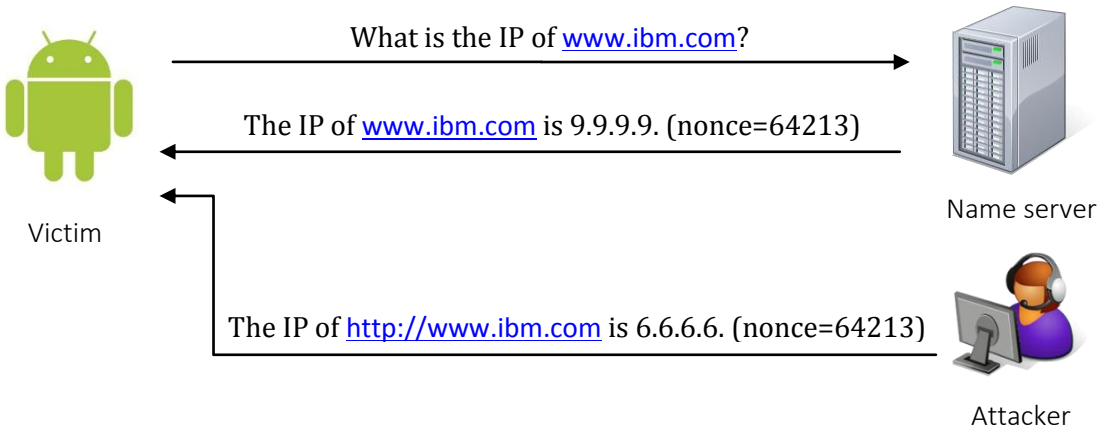
קצת על DNS ו- Blind DNS poisoning

DNS הוא פרוטוקול שעובד בדרך כלל מעל UDP. אופן הפעולה של התוקף ב-Blind DNS poisoning הוא לאלץ name server או Resolver מסויים לשלוח בקשה, ולהחזיר תשובה זדונית כ-name server אליו הוא פונה, לפני שמגיעה התשובה המקורית. כאשר אנחנו מדברים על Blind DNS poisoning אנו מניחים שהתוקף אינו רואה את המידע. אם המצב שונה - כללי המשחק משתנים, ופרוטוקול ה-DNS פשוט אינו מסוגל להתמודד עם מצב זה ללא DNSSEC.



המצב המתואר למעלה הוא טריוואלי מבחינת התוקף, כי הוא יכול לתזמן את המתקפה כך שתמיד או ברוב המקרים יינצח את השרת המקורי. כדי להתמודד עם מצב זה כל בקשת DNS מכילה מזהה ייחודי, nonce, שחייב להופיע גם בתשובה. התוקף חייב לנחש נכון את ה-nonce כדי להצליח במתקפה. הדבר אינו מונע מתקפה רפטטיבית: אם התוקף נכשל בנסיון מסויים בגלל טעות בניחוש, הוא יכול לנסות שוב (אך עם domain name שונה, עקב caching).

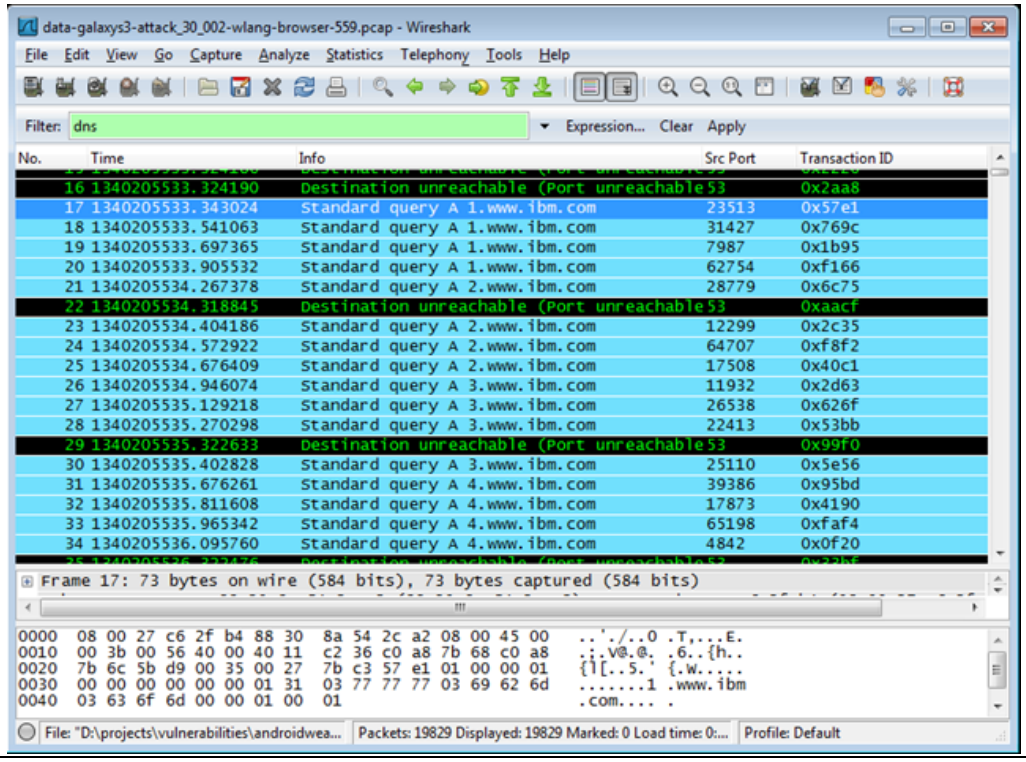
כלומר בפועל, התקיפה מבוצעת באופן הבא:



זמן התקיפה הממוצע עד ההצלחה הראשונה הוא ביחס ישר לגודל ה-nonce. ב-2008, דן קמינסקי הראה מתקפה פרקטית כאשר ה-nonce מורכב משדה ה-TXID בלבד (זהו הערך הראשון המופיע ב- DNS Header, מספר בגודל 16 ביט). בעקבות הגילוי, מימושי DNS פגיעים הוסיפו רנדומיזציה גם ב- UDP source port (16 ביט בקירוב, תלוי בעומס המערכת). לכן ה-nonce כיום הוא בסדר גודל של 32 ביט, מספר שהוא בלתי-פיזבילי לתקיפה.

הסנפה מוזרה

במסגרת העבודה יצא לנו להסניף תעבורת DNS שיצאה ממכשיר Galaxy S3, שהריץ אנדרואיד 4.0.4:



ניתן להבחין בקלות כי יש קשר חזק בין צמדי ה-TXID וה-UDP source port.
לדוגמא:

- 23513, 22496
- 28779, 27765
- 25110, 24150

במימוש תקין המספרים צריכים להיות ברוב המקרים שונים מהותית.

כמובן שהתוצאות עוררו בנו חשד כי יש בעיתיות במנגנון ה-DNS באנדרואיד, ולכן החלטנו לחקור אותו לעומק.

DNS Resolution באנדרואיד

אנדרואיד מכילה מימוש משלה ל-libc בשם Bionic. ספרייה זו מכילה את ה-DNS Stub resolver של המערכת תחת:

```
/libc/netbsd/resolv
```

מימוש ה-source port randomization מתבצע במעטפת לפונקציה bind, בשם random_bind:

```
static int
random_bind( int s, int family )
{
  ...
  /* first try to bind to a random source port a few times */
  for (j = 0; j < 10; j++) {
    /* find a random port between 1025 .. 65534 */
    int port = 1025 + (res_randomid() % (65535-1025));
    if (family == AF_INET)
      u.sin.sin_port = htons(port);
    else
      u.sin6.sin6_port = htons(port);

    if ( !bind( s, &u.sa, slen ) )
      return 0;
  }

  /* nothing after 10 tries, our network table is probably busy */
  /* let the system decide which port is best */
  if (family == AF_INET)
    u.sin.sin_port = 0;
  else
    u.sin6.sin6_port = 0;

  return bind( s, &u.sa, slen );
}
```

יצירת ערך ה-TXID היא בפונקציה res_nmkquery תחת res_mkquery.c:

```
int
res_nmkquery(res_state statp,
     int op,       /* opcode of query */
     const char *dname,    /* domain name */
     int class, int type,   /* class and type of query */
     const u_char *data,   /* resource record data */
     int datalen,     /* length of data */
     const u_char *newrr_in,   /* new rr for modify or append */
     u_char *buf,     /* buffer to put query */
     int buflen)     /* size of buffer */
{
  ...
  hp = (HEADER *)(void *)buf;
  hp->id = htons(res_randomid());
  ...
}
```

כפי שניתן לראות, שתי הפונקציות מייצרות את המספר האקראי ע"י קריאה ל-res_randomid תחת res_init.c. זהו בעצם ה-PRNG (Pseudo-random number generator) עליו ה-DNS Resolver באנדרואיד מתבסס.

```
u_int
res_randomid(void) {
  struct timeval now;

  gettimeofday(&now, NULL);
  return (0xffff & (now.tv_sec ^ now.tv_usec ^ getpid()));
}
```

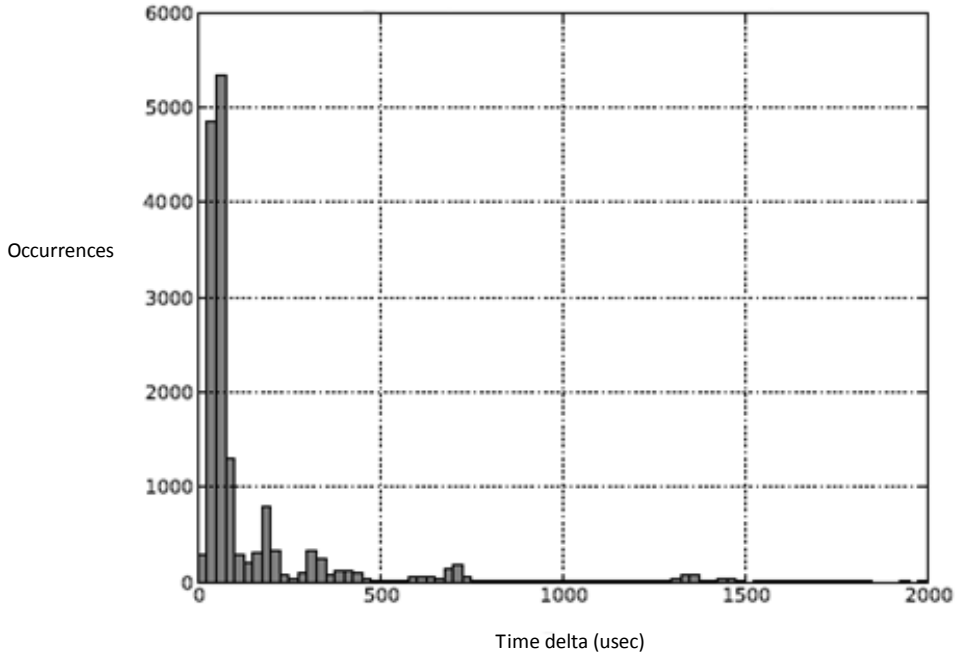
ניתן לראות מהקוד לעיל שהערך האקראי שה-PRNG מחזיר הוא פונקציה של הזמן ושל ה-PID:

$$WORD(time_{sec} \oplus time_{\mu frac} \oplus PID)$$

עבור תהליך מסויים, הערך הרנדומלי תלוי אך ורק בזמן (ברזולוציה של מיקרו-שניות), כאשר ה-TXID וה-Port הם פונקציה שלו.

הפגיעות

נשים לב ששני הערכים הרנדומליים עליהם ה-TXID וה-UDP source port מתבססים, נוצרים תוך פרק זמן קצר מאוד. על מנת לאשש את ההשערה דגמנו את המערכת אלפי פעמים, וקיבלנו את ההיסטוגרמיה הבאה עבור הפרש הזמנים בין הקריאות.



ניתן להבחין כי ברוב המקרים מדובר על מיקרו-שניות בודדות שמפרידות בין שתי הקריאות ל-PRNG. מכיוון שה-PRNG עבור תהליך מסויים תלוי בזמן בלבד (ברזולציה של מיקרו-שניות), הוא יחזיר עבור שתי הקריאות שני ערכים דומים מאוד!

מבחינת התוקף, בהינתן שהוא ניחש נכון ערך מסויים, הוא יכול לנחש נכון את הערך השני בהסתברות גבוהה מאוד. במאמר הראנו שבמערכות מסויימות מספר הביטים הרנדומליים הוא קרוב ל-20. באופן פרקטי הפגיעות מחזירה אותנו לעולם הישן של לפני 2008!

אז מה ניתן לעשות עם זה?

מתקפת DNS poisoning יכולה להשפיע על integrity ו-confidentiality של כל לקוח המשתמש ב-cache המורעל.

דבר מעניין שהתוקף יכול לעשות באנדרואיד הוא לתקוף את אפליקצית ה-Browser. אם התוקף מצליח לשכנע את הקורבן לגלוש לאתר בשליטתו, הוא יכול להריץ עליו קוד JavaScript שמייצר שאילתות DNS עם subdomains משתנים תחת דומיין לפי בחירתו (למשל: www.ibm.com).

כלומר הוא ינסה לתקוף עד ההצלחה הראשונה את ה-subdomains הבאים:

- 1.www.ibm.com
- 2.www.ibm.com
- 3.www.ibm.com
- ...

באותו זמן התוקף שולח תשובות DNS שיקריות עם nonce שמיוצר לפי הטכניקה המופיעה במאמר. לאחר ההצלחה, התוקף יכול למשל לראות wildcard cookies של www.ibm.com, ובכך לגנוב את הזהות שלו מול אותו אתר. התקיפה יותר חזקה מ-XSS בהיבט מסויים, מכיוון שלהבדיל מ-XSS, התקיפה מאפשרת לתוקף לראות גם HTTPOnly cookies.

גרסאות פגיעות

אנדרואיד 4.0.4, הפגיעות תוקנה באנדרואיד 4.1.1 ע"י גוגל (תוך שבועיים מרגע הדיווח!)

קישורים

- המאמר המקורי, הכולל ניתוח הסתברותי מפורט וטכניקה מלאה לתקיפה: http://bit.ly/MkteBx
- סרטון המדגים exploit לפגיעות: http://youtu.be/ffnF7Jej7l0

על המחבר

רועי מוביל את קבוצת מחקר הסקיורטי ב-IBM. הקבוצה חוקרת חולשות במגוון תחומים, מ- Application Security ועד Network Security. לאחרונה סיים B.Sc. במדעי המחשב בטכניון.