Knäcka WEP krypteringen i 802.11 nätverk med hjälp av Airsnort

Författare: Alexander Isacson aleis959@student.liu.se
Sam Hadiani samha240@student.liu.se

1 Inledning

Denna rapport är skriven för kursen TDDC03 Informationssäkerhet vid Linköpings Tekniska Högskola. Den behandlar de svagheter som undergräver säkerheten i Wired Equivalent Privacy (WEP) protokollet som är väldigt populärt i trådlösa 802.11 nätverk.

Vi har valt att dela upp rapporten i två delar. En teoridel, där vi beskriver de svagheter i WEP-protokollet som vi utnyttjar, samt en praktisk del där vi utför en attack mot vårt testnätverk. Attacken körs från linuxdistributionen Gentoo. Alla kommandon är specifika för den distributionen, men kan av en van användare lätt överföras till någon annan linuxdistribution.

2 Teori

Här ska vi beskriva att det är möjligt och hur man schematiskt går till väga för att knäcka WEP-krypteringen.

2.1 Bakgrund

De trådlösa nätverken, som bygger på någon av 802.11 standarderna, börjar bli allt mer populära. Säkerheten i dessa nätverk är dock minimal. Den säkerhetslösning som togs fram i samband med standarden heter WEP. Denna är, som vi ska visa här, nästan helt värdelös.

Om man väljer att inte använda WEP är det väldigt enkelt att avlyssna trafiken. Man behöver i stort sett bara gå inom räckvidd för accesspunkten med sin dator och köra ett program som t.ex. *Ethereal*.

En dator associerar i standardkonfigurationen till den accesspunkt som har starkast signalstyrka. Faran i detta är att någon kan sätta upp en accesspunkt i närheten ett nätverk och då kan felkonfigurerade datorer associera dit istället. Eftersom trafiken då går genom det andra nätet förloras all kontroll och ägaren av den nya accesspunkten har total kontroll över de paket som skickas.

2.2 RC4

WEP-kryptering använder sig av en hemlig nyckel, *k*. Denna nyckel används både av accesspunkten och av de mobila användarna. WEP använder sig av ett *stream-cipher* som kallas *RC4* för att generera en lång slumpmässig sekvens med hjälp av *k*. För att bilda en WEP-frame sammanfogas meddelandet, M, med dess checksumma, c(M), för att bilda M · c(M). Senare läggs initierings vektorn, IV, på den hemliga nyckeln, vilket används för att initiera RC4-algoritmen. Genom att addera utsekvensen från RC4 och meddelandet bitvis modulo 2 (XOR) erhålls tillslut kryptogrammet enligt nedan:

$$C = (M \cdot c(M)) \ (XOR)\ RC4(IV \cdot k)$$

RC4 är uppbyggd av två huvuddelar. KSA som står för *key scheduling algorithm*, beskrivs i algoritm 1.

```
KSA(K)
Initialization
        For i = 0 ... N -1
                S[i] = i
        j = 0
Scramblig
        For i = 0...N-1
                j=j+S[i]+K[i mod l]
                Swap(S[i],S[j])
```

Algoritm 1 KSA algoritmen.

KSA använder sig antingen av en 64- eller 128-bitars-nyckel och det är den som bygger upp RC4:ans tillståndsfält, S. Tillståndsfältet består av en permutation av alla tal mellan 0 och 255. Den andra beståndsdelen i RC4 är PRGA, *Pseudo Random Generation Algorithm*. Den använder sig av S för att skapa en pseudoslumpsekvens. Den fungerar enligt nedan:

```
PRGA(K)
Initialization
        i = 0
        j = 0
Generation loop
        i = i + 1
        j = j + S[i]
        Swap(S[i],S[j])
        Output z = S[S[i] + S[j]]
```

Algoritm 2 PRGA algoritmen.

2.3 Resolved-tillstånd

Attacken fokuserar på den första byten av sekvensen eftersom man vet hur den okrypterade 802.11 headern ser ut. Man har då information om meddelandet, kryptogrammet och kan där ur härleda nyckeln.

Första byten skapas enligt PRGA: S[S[1] + S[S[1]]]. Första byten är således endast beroende av tillstånden S[1], S[S[1]] och S[S[1] + S[S[1]]]. Det går att visa att det finns en chans >5% att inget av de värdena kommer att genomgå någon ytterligare SWAP-operation. Inträffar detta kallas det ett *resolved*-tillstånd. För att man ska kunna säga något om nyckeln i ett *resolved*-tillstånd måste man veta att det är en chans <95% att någon av dessa värden kommer att användas in någon SWAP och att resultatet då blir helt slumpmässigt.

2.4 Slutsatser vi kan dra

Vi tänker oss fallet då vi har en IV som är I ord lång. Den har lagts till i början av den hemliga nyckeln (K[0], K[1],...,K[*l*-1]). Vi ska försöka få ut information om ett visst ord B ur nycklen K[B], genom att söka bland de IV:n som efter I steg uppfyller $S_I[1] < I$ och $S_I[1] + S_I[S_I[1]] = I + B$. Det är då sannolikt att vi befinner oss i ett *resolved*-tillstånd efter steg I + B och det mest sannolika utdata är:

$$out = S_{I+B-1}[j_{I+B}] = S_{I+B-1}[j_{I+B-1}] + K[B] + S_{I+B-1}[I+B]]$$

Nu går det lätt att beräkna K[B] när värden på j_{I+B}, S_{I+B-1} och *out* är kända vilket de i detta fallet alltid är.

$$K[B] = S^{-1}_{I+B-1}[out] - j_{I+B-1} - S_{I+B-1}[I+B]$$

$S^{-1}_t[X]$ beskriver positionen inom permutationen S_t där man kan hitta värdet X.

2.5 Attacken

Attacken går ut på att statistiskt analysera de paket som man tror är i *resolved*-tillståndet. Eftersom vi har en chans som är >5% att vi kan dra slutsatser om nyckeln, och att det i annat fall är helt slumpmässiga utfall, kan vi försöka dra våra slutsatser på samtliga paket. Genom att sedan kontrollera vilka svar som är mest frekventa bland de beräknade nycklarna erhålls en trolig nyckel till *RC4*-algoritmen. Den formella algoritmen ser ut som nedan:

```
RecoverWEPKey()
  Key[0...KeySize] = 0
  for KeyByte = 0...KeySize
    Counts[0...255] = 0
    foreach packet   P
      if P.IV  {(KeyByte+3,0xFF,N) N  0x000..0xFF}
        Counts[SimulateResolved(P,Key)] += 1
    Key[KeyByte] = IndexOfMaximumElement(Counts)
  return Key
```

Algoritm 3 Algoritmen för att ta fram WEP nyckeln.

3 Praktiskt

Målet med det praktiska experimentet är att få fram WEP-nyckeln. Detta genom att avlyssna trafiken på det trådlösa nätverket.

3.1 Hårdvara och mjukvara

Vi använder oss av två bärbara datorer och en Netgear MR314 router. På en av de bärbara datorerna har vi installerat Gentoo Linux den har ett Netgear MA401 PCMCIA trådlöst nätverkskort. Den andra bärbara datorn kör Microsoft Windows XP och det är kommunikationen mellan windowsdatorn och routern som vi vill avlyssna och ta fram WEP-nyckeln för. I rapporten nämner vi följande mjukvara:

Mjukvara	***Version***	***URL***
Gentoo Linux	1.4_rc4	www.gentoo.org
Microsoft Windows XP	Home Edition	www.microsoft.com
Airsnort	0.2.1b	airsnort.shmoo.com
Linux-WLAN-ng	0.2.1_pre3	linux-wlan.org
HostAP	0.0.2	hostap.epitest.fi
Orinico drivrutiner	0.13b	airsnort.shmoo.com/orinocoinfo.html
Ethereal	0.9.12	www.ethereal.com
Kismet	2.8.1-r1	www.kismetwireless.net

Tabell 1 Mjukvara som använts i projektet.

3.2 Installation

Vi förutsätter att båda operativsystemen är korrekt installerade. För information gällande hur man installerar Gentoo hänvisas till http://www.gentoo.org/doc/en/gentoo-x86-install.xmlFör Windows hänvisas till den medföljande dokumentationen. Det är viktigt att linuxkärnan kompileras med stöd för PCMCIA, dessutom att *USE* flaggan[1] *pcmcia* är satt.

För Linux finns det tre olika drivrutiner till det trådlösa nätverkskortet: HostAP, Linux-WLAN-ng samt Orinoco. Olika resultat erhölls med de olika drivrutinerna. Vi rekommenderar enbart Orinoco-drivrutinen och därför har vi bara tagit med anvisningar för hur den installeras. Vi kommer dock beskriva de problem vi stött på med de andra drivrutinerna.

3.2.1 HostAP

Fördelen med HostAP gentemot de två andra drivrutinerna är att den kan agera som accesspunkt för nätverket. Även om det inte är något av intressen i detta projekte, var det denna drivrutinen vi testade först.

Med denna drivrutin lyckades vi inte att sniffa några paket. Vi hade problem vid kompileringen och bestämde oss för att prova en av de andra drivrutinerna.

3.2.2 Linux-WLAN-ng

Även med denna drivrutin stötte vi på problem. Den laddades inte korrekt när nätverkskortet sattes i. Efter att ha genomsökt otaliga epost-listor fann vi att det var andra som hade haft liknande problem med den senaste versionen. Därför installerade vi istället version 0.2.1_pre1. Nu kunde vi för första gången börja sniffa paket. Vi märkte dock snabbt att det bara var vart tionde paket som sniffades och att det tog fruktansvärt lång tid att hitta intressanta paket. Dessvärre insåg vi sent att det var drivrutinerna som var problemet, vi trodde först att det var en begränsning i *Airsnort* programmet.

1 USE flaggor används i Gentoo för att välja vilka funktioner som ska kompileras in vid installation av paket.

3.2.3 Orinoco

På pappret såg denna drivrutin ut att vara den minst kraftfulla av de tre. Trots detta visade det sig att denna var den bäst lämpade för uppgiften och därför den vi rekommenderar. Den måste patchas för att kunna användas för att sniffa. Det görs som beskrivet i kodstycke 1:

```
# cd or # cd /root/
# tar -zxf orinoco-0.13b.tar.gz
# patch -p0 < orinoco-0.13b-patched.diff
# cd orinoco-0.13b
# make
# make install
```

Kod 1 Patchning och installation av Orinoco drivrutinen.

Nu är det bara att starta om PCMCIA-tjänsten så den nya drivrutinen läses in.

```
# /etc/init.d/pcmcia restart
```

Kod 2 Starta om PCMCIA tjänsterna.

Verifiera att installationen är korrekt genom att kontrollera att kommandot *dmesg* ger utdata liknade den nedan.

```
# dmesg

hermes.c: 4 Dec 2002 David Gibson <hermes@gibson.
dropbear.id.au>
orinoco.c 0.13b (David Gibson <hermes@gibson.
dropbear.id.au> and others)
orinoco_cs.c 0.13b (David Gibson <hermes@gibson.
dropbear.id.au> and others)
eth1: Station identity 001f:0001:0008:000a
eth1: Looks like a Lucent/Agere firmware version
8.72
eth1: Ad-hoc demo mode supported
eth1: IEEE standard IBSS ad-hoc mode supported
eth1: WEP supported, 104-bit key
eth1: MAC address 00:02:DE:AD:BE:EF
eth1: Station name "HERMES I"
eth1: ready
eth1: index 0x01: Vcc 5.0, irq 3, io 0x0100-0x013f
eth1: New link status: Connected (0001)
```

Kod 3 Utdata för att verifiera att installationen är korrekt.

För att försäkra sig om att det är den patchade versionen av drivrutinen, och inte den som följer med linuxkärnan, som installerats kör följande kommando och kontrollera att det finns en rad som börjar med ordet *monitor*. Se kodstycke 4.

```
# iwpriv eth1

eth1 Available private ioctl :
...
monitor             (8BE8) : set 2 int & get 0
...
```

Kod 4 Verifiering av patchning.

Om inte denna raden finns är det fel version av drivrutinen som laddas. Ta i så fall bort alla *hermes.o* och *orinoco** under strukturen */lib/modules*, installera därefter om.

3.3 Påbörja sniffningen

Nu ska kortet vara installerat så att det går att sniffa med det. För att att kunna lyssna av trafiken måste kortet sättas i *Monitor Mode*. Det klarar *Airsnort* av att göra. Installera det genom att skriva nedanstående.

```
# emerge airsnort
```

Kod 5 Installation av Airsnort

När installationen är klar ska programmet startas och rätt kanal och nätverks-*interface* väljas. Ställ även in *orinoco_cs* som drivrutin. Om det är oklart vilken kanal trafiken går på ska sniffningen startas i *Scan* läge. Så fort kanalen är känd ställs den in under *Channel* för att effektivisera sniffningen så mycket som möjligt.

Det har visat sig svårt att få sniffningen att starta stundtals. Om *Kismet* installeras går det att kringgå det problemet genom att köra nedanstående kommandon.

```
# kismet_monitor -c 1
Using /etc/kismet/kismet.conf sources...
Enabling monitor mode for an orinoco card on eth1
channel 1

# kismet_unmonitor
Using /etc/kismet/kismet.conf sources...
Disabling monitor mode for an orinoco card on eth1
You will likely need to restart your PCMCIA
services to reconfigure your card for the correct
channel and SSID.

# airsnort
```

Kod 6 Alternativ uppstart av Airsnort.

Konfigurationen av Kistmet tar vi inte upp här utan hänvisar till projektets hemsida, www.kismetwireless.net.

3.4 Resurser som Airsnort kräver

Under tiden som *Airsnort* körs ligger CPU användningen på ca 30%, om man skickar data, och på ca 10% om man genererar trafik med en ping-flood. Detta på en PIII 600MHz. Det är egentligen drivrutinerna för kortet som behöver den beräkningskraften för att analysera alla de paket som ligger på nätverket och inte Airsnort i sig. När nätet är fullt belastat har det en genomströmning på ca 500KB/s. Eftersom *Airsnort* sorterar alla inkommande paket efter hur sannolika de är att ge relevant information om nyckeln så går CPU användningen upp på 100% allt längre stunder allteftersom fler och fler intressanta paket upptäcks.

Airsnort processen kräver mellan 3MB och 4MB RAM minne så det går att köra på en förhållandevis billig dator. (Eftersom det är ett grafiskt program så kräver fönstersystem och dylikt givetvis mer minne.)

Den största resursen som programmet kräver är tid för att samla ihop paketdata. Det tar i storleksordningen fem till tio timmar beroende på hur mycket tur man har, och då är nätet under maximal belastning hela tiden. Vi har märkt att det ofta går snabbare att hitta intressanta paket i början av sniffningen.

3.5 Resultat

Det första försöket att attackera nätverket misslyckades. Efter att ha funnit 6075 intressanta paket, överfört mer än 40GB data, och skickat fler än 45 miljoner paket totalt avbröt vi attacken. Vi hade då spenderat kring tre dagar med att försöka finna WEP-nyckeln.

Att vi inte lyckades tror vi beror på att vi sniffade data med olika drivrutiner, sniffade med flera sniffers parallellt, (*Airsnort*, *Ethereal*, *Kismet*) samt på det faktum att datorn hängde sig vid några tillfällen. Detta kan ha resulterat i korrupterad data.

40-bitars-nyckeln knäcktes på 7.5 timmar då vi hela tiden kopierade data till windowsdatorn. När nyckeln knäcktes hade vi undersökt 5 421 872 paket av 2 174 var intressanta, dvs att de troligtvis var i *resolved*-tillstånd.

128-bitars-nyckeln gick faktiskt snabbare då vi istället för att skicka data använde oss av en ping-flood för att på skynda paketuppsamlandet. Eftersom ping-pakten är små går det många paket på lite bandbredd. Det tog sex timmar att samla in 9 439 247 paket där 3 587 var intressanta.

Vi presenterar resultaten för de två attackerna vi gjort. Tidsåtgång och antal paket är bara riktmärken, vilka kan variera mycket från attack till attack. För att se hur attacken skalar, måste så många attacker göras att ett pålitligt medelvärde kan räknas ut. Pga tidsbrist hade vi dessvärre inte möjlighet att göra detta. Att värden från enskilda attacker är ointressanta ut skalningssynpunkt, beror på att antalet intressanta paket inte är jämt spridda över mängden av möjliga IV. Hur många paket som går åt och hur lång tid det tar beror på hur nätverkskorten i det nätverket som attackeras genererar sina IV samt hur mycket trafik som finns i nätverket.

Egentligen ska man inte själv framkalla någon trafik i det trådösa nätet. Detta är en passiv attack som är helt omöjlig att detektera eftersom ingen vet att trafiken avlyssnas. Om man inte själv genererar trafik kan det dock ta väldigt lång tid att samla in intressanta paketet. Från Airsnorts hemsida:

> To get an idea [about how long it will take], assume that your business (it's not very big yet) has four employees, all using the same password. These employees surf the net pretty continuously throughout the day (they're not very good employees.) These employees will generate about a million packets a day. These employees will generate approximately a hundred and twenty interesting packets every day, so after sixteen days, the network will almost certainly be cracked.

3.6 Verifiera att attacken lyckats

För att verifiera att attacken verkligen har fungerat så konfigurerade vi kortet på linuxdatorn med den funna WEP-nyckeln. Då används kommandot *iwconfig*.

```
# iwconfig eth1 key 23ab53ef692ba2ff3dee10s2d6
```

Kod 7 Ställa in WEP nyckeln.

Nu fick vi ett IP-nummer av DHCP-servern och vi kan komma ut på internet via routern. Attacken har lyckats. I loggfilen står att läsa koden i ruta 8.

Att den ansluter och sen kopplar ifrån för att sedan ansluta igen beror på att *iwconfig* kommandot kördes där emellan. Den nya anslutningen använder sig av krypteringen.

```
[cardmgr] socket 0: Netgear MA401RA Wireless
Adapter
[cardmgr] executing: 'modprobe orinoco_cs'
[cardmgr] executing: './network start eth1'
[kernel] eth1: New link status: Connected (0001)
[kernel] eth1: New link status: Disconnected
(0002)
[kernel] eth1: New link status: Connected (0001)
[cardmgr] +  * Bringing eth1 up...  [ ok ]
```

Kod 8 Utdrag ur logfilen vid anslutning med WEP.

4 Avslutning

Vi säger som de flesta andra som har synat WEP i sömmarna, ”Använd WEP, men lita inte på det”. Betrakta ditt WEP-nät som helt öppet, använd protokoll som SSH och SSL och sätt upp Virtuella Privata Nätverk. Alla accesspunkter ska vara utanför brandväggarna.

WEP visar i alla fall tydligt att du inte vill ha användare som lyssnar av din trafik. Det ger ett juridiskt skydd ifall du kommer på någon som har ”stulit” din nyckel.

5 Källor

[1] A. Stubbelfield, J. Ioannidis and A.D. Rubon, "*Using the Fluhrer, Mantin and Shamir Attack to Break WEP"*, AT&T Labs, 2001.

[2] S. Fluhrer, I. Mantin, and A. Shamir, “*Weakness in the Key Scheduling Algorithm of RC4”*

[3] Airsnort FAQ, airsnort.schmoo.com