Matematica binaria e esadecimale (For Totally Newbies)		
Data	by Spider	
ottobre 2001	***UIC's Home Page***	**Published by Quequero**
1+1=2? *Sbagliato!...*	*Qualche mio eventuale commento sul tutorial :)))*	*1+1=10!!*
....	**Home page:** http://bigspider.cjb.net **E-mail:** spider_xx87@hotmail.com **Server IRC: irc.azzurra.it** **Canale: #crack-it** **Nickname: ^Spider^**	....
Difficoltà	(X)**NewBies** ( )**Intermedio** ( )**Avanzato** ( )**Master**	

In questo tute faremo un po' di matematica, che serve moltissimo nel cracking quanto nella programmazione. Ho preferito fare un tute abbastanza breve. Questo non perché sono svogliato... cioè... anche per questo :) , ma il vero motivo è un altro. So che i tutorials introduttivi al cracking e alla programmazione possono spesso risultare noiosi, quindi ho preferito limitarmi all'essenziale. Il resto lo imparerete strada facendo.

Matematica binaria ed esadecimale

(For Totally Newbies)

Written by Spider

Introduzione

La matematica è un argomento molto importante e che non va assolutamente tralasciato se si vuole entrare nel mondo del cracking e/o della programmazione. A volte si tende a saltare questi argomenti per passare direttamente alla pratica... E invece no, la teoria è molto importante, questi preliminari non possono essere saltati, altrimenti vi ritroverete fregati in men che non si dica. Bene, dopo questa precisazione possiamo incominciare. :)

Tools usati

- Un cervello funzionante!
- Può essere utile la calcolatrice di Windows (bisogna utilizzare quella scientifica, che permette di fare calcoli in binario, esadecimale, ecc...)

Essay

Come forse saprete già (ma probabilmente lo scoprirete solo ora...) nel cracking e nella programmazione si usano moltissimo il sistema di numerazione binario (o a base 2) e quello esadecimale (a base 16). Cosa sono questi sistemi di numerazione? Semplice: nel linguaggio comune si usa il sistema di numerazione a base 10, o decimale, in cui i numeri sono rappresentati con cifre cha vanno da 0 fino a 9, per un totale di 10 simboli diversi. In base binaria invece abbiamo a disposizione solo 2 simboli, cioè lo zero e l'uno. In quella esadecimale abbiamo invece 16 simboli, e siccome i numeri sono solo 10 (da 0 a 9) dobbiamo ricorrere alle lettere, le prime 6 per l'esattezza. Quindi avremo questi simboli per il sistema binario:

0, 1

Questi per il sistema decimale:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

E questi per quello esadecimale:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Vi starete chiedendo: ma perché la gente si complica la vita in binario e in esadecimale? In realtà non ci si complica la vita: si semplifica molto il lavoro. Ricordatevi che quando si programma diamo istruzioni alla CPU. Ma la CPU può capire solo 2 valori, ovvero "0 = non passa corrente" e "1 = la corrente passa"... capito il nesso? Con la numerazione binaria "parliamo" la stessa matematica della CPU. La numerazione esadecimale invece è molto comoda perché i numeri rappresentati in esadecimale sono molto compatti. Facciamo qualche esempio:

Binario	Decimale	Esadecimale
1	1	1
10	2	2
1010	10	A
10000	16	10
11111111111111111111111111111111	4294967295	FFFFFFFF

Ehm... Forse ho un po' esagerato... Comunque lo scopo di questa tabella è soltanto quello di farvi capire le differenze tra i tre sistemi di numerazione. Adesso

faremo un po' di calcoli, ma solo quelli necessari per saper effettuare le conversioni tra binario-decimale e viceversa. In tutti i sistemi di numerazione la posizione delle cifre è fondamentale per stabilire il valore effettivo. Ad esempio, la cifra 33 è formata da due 3, ma il primo 3 non ha lo stesso valore del secondo, ma ha bensì un valore 10 volte superiore... e non a caso siamo in base 10! Lo stesso vale per il binario: 111 è formato da tre 1, ma il primo dei due (cioè quello a sinistra) vale il doppio del secondo e il quadruplo del terzo, mentre il secondo vale il doppio del terzo... temo di non essere stato molto chiaro. Traducendo in formule il valore di una cifra vale $n * b^p$, dove n è la cifra in questione, p è la posizione della cifra partendo da destra (la cifra più a destra vale 0, la penultima 1, la terzultima 2 e così via) e b è la base del sistema di numerazione (2 per il binario, 10 per il decimale, ecc.). Facciamo un esempio: troviamo il valore rappresentato dal secondo 1 (la terza cifra partendo da sinistra) nel numero $(10100101)_2$ (il 2 messo lì significa che la cifra è in base 2). Partendo da destra, l'1 in questione occupa la 6ª posizione, e quindi il suo valore sarà $1*2^5=32$ decimale. Per convertire un intero numero decimale si fa il medesimo procedimento con tutte le cifre. Supponiamo di dover convertire in decimale il numero $(01101011)_2$:

$0 * 2^7 + 1 * 2^6 + 1 * 2^5 + 0 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 = 0 + 64 + 32 + 0 + 8 + 0 + 2 + 1 = 107$

01101011 è quindi uguale a 107 in base decimale.

Adesso vediamo il procedimento inverso, cioè trasformare un numero decimale in base binaria. In questo caso è molto più semplice: si divide sempre per 2 e si scrivono i resti, e si smette appena si ottiene 1. Dopo si invertono i resti... ed eco il nostro numero binario :). Facciamo un esempio. Trasformiamo 118 in binario:

```
118 |
 59 |0
 29 |1
 14 |1
  7 |0
  3 |1
  1 |1
  0 |1
```

Invertiamo i resti: il risultato è 1110110

Adesso vediamo un altro tipo di conversione: da binario a decimale e viceversa. Per convertire tra questi due sistemi di numerazione basta tenere presente che ad ogni 4 cifre binarie corrisponde 1 cifra esadecimale. Quindi basta memorizzare questa tabella di conversione e il gioco è fatto... ma niente paura, questo lo imparerete col tempo e con la pratica, in ogni caso ecco la tabella:

Binario	Esadecimale
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Per fare un esempio convertiamo in bin il numero A12:

```
 A   1    2
1010 0001 0010
```

Come forse avrete già notato fino a questo momento abbiamo utilizzato soltanto numeri positivi. Ma se noi vogliamo utilizzare numeri negativi che facciamo? In questo caso si ricorre a un metodo chiamato complemento a 2. Il funzionamento è poco intuitivo, ma in realtà è molto semplice. Chiariamo direttamente con un esempio, rappresentando in binario il numero -7:

1. Per prima cosa rappresentiamo in bin il numero voluto (-7) senza il segno, e quindi 7 = 00000111.

2. Invertiamo tutti i bits del nostro numero. Questo procedimento è detto complementazione, da cui il nome 'complemento a 2':

00000111 = 11111000

3. Si aggiunge 1 al risultato:

11111000 + 00000001 = 11111001

La rappresentazione binaria di -7 è dunque 11111001. Con questo sistema, utilizzando n bit, possiamo rappresentare numeri nell'intervallo che va da $-2^{(n-1)}$ fino a $2^{(n-1)}$-1. Il primo bit ci serve per il segno.

Adesso faremo un pochino di logica binaria, che è anche una cosa utile da sapere. Accanto alle normali operazioni aritmetiche (addizione, sottrazione, moltiplicazione e divisione), che dovreste già conoscere :) , troviamo le operazioni logiche che lavorano bit a bit, tra cui le più usate sono l'AND, l'OR, lo XOR e il NOT. Ricordatevi che queste variabili prendono due valori binari e restituiscono un terzo valore. Per calcolare un'operazione logica basta tenere presenti le cosiddette "tavole di verità". Vediamo quella dell'AND:

0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

In pratica l'AND restituisce 1 solo se entrambi i valori in input sono 1.Adesso esaminiamo l'OR:

0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

L'OR è altrettanto semplice: restituisce 0 solo se entrambi i valori passati sono uguali a0, altrimenti restituisce 1.
Ora esaminiamo lo XOR:

0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

Lo XOR è leggermente più complesso: restituisce 0 se gli elementi sono uguali, 1 se sono diversi. La proprietà dello XOR è di essere un'operazione reversibile, nel senso che se xoriamo un numero due volte per lo stesso valore riotteniamo il valore iniziale, cioè:

(a XOR b) XOR b = a

Questa proprietà è molto importante, perché viene spesso utilizzata per criptare e decriptare dati, quali le password dei programmi che crackerete, e quindi è molto importante sapersi destreggiare con lo xoring. Ma di questo scenderemo più nei dettagli in futuro.
Adesso vediamo il NOT. Questo è l'unico operatore che richiede un solo valore, ed anche il più facile da ricordare. Vediamo la tavola:

0	**1**
1	**0**

Semplicemente inverte il valore che riceve. Anche questa è un'operazione reversibile, perché la negazione di una negazione si annulla:

NOT (NOT a) = a

Esistono anche delle operazioni derivate, che altro non sono che la combinazione di altre due operazioni. Ad esempio, NOR(a,b) equivale a NOT OR(a,b), e NAND è un NOT AND. Tuttavia queste operazioni sono meno importanti e molto meno usate.

Con questo abbiamo finito. Spero di non avervi annoiato troppo, ma un po' di matematica è indispensabile.

Spider

Note finali

Saluti a Quequero, AndreaGeddon, Yado, kill3xx, +Malattia, case, albe, TheMR, bubbo, deimos, dades, Blackdeath, phobos, DsE, Cieli Sereni, [cHr]... e basta credo. Spero di non aver dimenticato nessuno.

Disclaimer

Home