Monitoring

Monitoring info and scripts

- Nick Abbots Script
- One-liners
- Bots/Crawlers & Control/Mitigation
- Resource usage and performance
- xmlrpc & wp-login
- ANS IPs
- Useful Online Tools

Nick Abbots Script

Nick Abbots Script:

```
vi ukfastmon;chmod u+x ukfastmon; ./ukfastmon; rm -f ukfastmon
```

```
#!/bin/bash
if [ "$1" = "w" ]; then
        locate wp-includes/version.php | xargs grep -H '^$wp_version' | grep -v virtfs
echo ""
echo ""
echo -n "The latest stable release of Wordpress is: "
curl -s https://wordpress.org/download/ | grep "Version" | awk '{print $9}' | cut -d ')' -f1
exit
fi

echo ""
echo ""
if [ ! -e /usr/bin/geoiplookup ]; then
        if [ -e /etc/network/interfaces ]; then
                apt-get install libgeoip1 geoip-bin
        else
                yum install GeoIP --enablerepo=epel -y
        fi
fi

if [ "$1" = "q" ]; then
        querydate=$2
elif [ "$1" = "d" ]; then
        querydate=$2
elif [ "$1" = "t" ]; then
        querydate="`date +%d/%b/%Y`:$2"
else
        querydate="`date +%d/%b/%Y`"
fi

if [ -e /dev/shm ]; then
        logs="$(TMPDIR=/dev/shm mktemp)"
```

```
else
        logs="$(mktemp)"
fi
if [ "$1" = "l" ]; then
        if [ "$3" = "d" ]; then
                querydate=$4
        fi
        if [ "$5" = "r" ]; then
                zgrep $querydate $2 | grep -v $6 > $logs
        else
                zgrep $querydate $2 > $logs
        fi
else

echo "Searching all access logs for $querydate"
echo ""

if [ -e /usr/local/cpanel/cpanel ]; then
        if [ "$1" = "q" ]; then
                find /usr/local/apache/domlogs/* -maxdepth 1 -type f | grep -v ssl | grep -v
ftp | grep -v byte | grep -v siteupda | xargs zgrep "`date +%d/%b/%Y`" | grep $querydate
2>/dev/null > $logs
        else
                find /usr/local/apache/domlogs/* -maxdepth 1 -type f | grep -v ssl | grep -v
ftp | grep -v byte | grep -v siteupda | xargs grep $querydate 2>/dev/null > $logs
                find /home/*/logs -type f | grep -v ftp | xargs zgrep $querydate 2>/dev/null
>> $logs
                find /home/*/access-logs -type f | grep -v ftp | xargs zgrep $querydate
2>/dev/null >> $logs
        fi
elif [ -e /usr/local/psa/version ]; then
        if [ "$1" = "d" ]; then
                if [ -e /var/www/vhosts/system ]; then
                        find /var/www/vhosts/system -name access_log* -o -name *proxy_access*
| xargs zgrep $querydate 2>/dev/null > $logs
                else
                        find /var/www/vhosts -name access_log* | xargs zgrep $querydate
2>/dev/null > $logs
                fi
        else
```

```
                if [ -e /var/www/vhosts/system ]; then
                        find /var/www/vhosts/system -name "*access_log" -o -name
*proxy_access* | xargs grep $querydate 2>/dev/null  > $logs
                else
                        find /var/www/vhosts -name "*access_log" -o -name *proxy_access* |
xargs grep $querydate 2>/dev/null > $logs
                fi
        fi
else
        find /var/log -name "*access_log*" -o -name "*access.log*" -o -name "*access_ssl_log*"
-o -name "varnishncsa.log" | xargs zgrep $querydate 2>/dev/null >> $logs
        find /home -name "*access_log*" -o -name "*access.log*" | xargs zgrep $querydate
2>/dev/null >> $logs
        find /var/www/vhosts -name "*access_log*" -o -name "*access.log*" | xargs zgrep
$querydate 2>/dev/null >> $logs
        find /var/www -name "*requests.log*" | xargs zgrep $querydate 2>/dev/null >> $logs
        find /root -name log | xargs zgrep $querydate 2>/dev/null >> $logs
        find /root -name varnishncsa.log | xargs zgrep $querydate 2>/dev/null >> $logs
fi
fi

if [ "$1" = "l" ]; then
echo "Searching specific log file $2 for $querydate"
echo ""
echo "Total number of requests"
echo ""
wc -l $logs | awk '{print $1}'
echo ""
else
echo "Total number of requests"
echo ""
wc -l $logs | awk '{print $1}'
echo ""
echo "Top 20 access log:IPs being hit"
echo ""
awk '{print $1}' $logs | sort | uniq -c | sort -gr | head -n 20
echo ""
fi
echo "Top 20 IPs"
echo ""
```

```
awk '{print $1}' $logs | cut -d: -f2 | sort | uniq -c | sort -gr | head -20 | awk '{
printf("%5d\t%-15s\t", $1, $2); system("geoiplookup " $2 " | head -1 | cut -d \\: -f2 ") }'
echo ""
echo "Top 10 /24s - treat country code with caution"
echo ""
awk '{print $1}' $logs | cut -d: -f2 | cut -d. -f1-3 | awk -F'[ .]' '{print $1 "." $2 "." $3
".0"}' | sort | uniq -c | sort -gr | head -10 | awk '{ printf("%5d\t%-15s\t", $1, $2);
system("geoiplookup " $2 " | head -1 | cut -d \\: -f2 ") }'
echo ""
echo "Top 20 requests"
echo ""
awk  '{print $6 " " $7 " " $9}' $logs | sort | uniq -c | sort -gr | head -n 20
echo ""
echo "Top 20 referrers"
echo ""
awk '{print $11}' $logs | sort | uniq -c | sort -gr | head -n 20
echo ""
echo "Top 20 user agents"
echo ""
cut -d\" -f6 $logs | sort | uniq -c | sort -gr | head -n 20
```

One-liners

==========

Common Error/Limit identification

Apache MaxRequestWorkers

```
grep -i 'MaxRequestWorkers' $(locate apache || locate httpd | grep -iE "error.log$")
```

PHP Max_Children

Check for all logged instances of max_children limit being reached - prints each occurrence. Remember, max_children being reached is usually the symptom of a larger issue - probably traffic.

```
grep -i max_children $(locate fpm | grep -iE "error.log$")
```

Check for all instances of max_children from today's date in the logs, and output as a count for each domains number of occurrences. (untested)

```
grep -i max_children $(locate fpm | grep -iE "error.log$") | grep -i "$(date +%d-%b)" | cut -d' ' -f5 | tr -d ']' | sort | uniq -c | sort -nr | awk 'BEGIN {print "Total PHP max_children errors on " strftime("%d-%b-%Y")}; {print $0}'
```

Check specific date's occurrences of PHP max_children being reached (date needs updating to desired date.)

```
grep -i max_children $(locate fpm | grep -iE "error.log$") | grep -i '30-May' | cut -d' ' -f5 | tr -d ']' | sort | uniq -c | sort -nr | awk 'BEGIN {print "Total PHP max_children errors today"}; {print $0}'
```

PHP Memory Limit

Search logs for instances of PHP memory_limit being reached: (untested)

```
grep -i 'Fatal error: Allowed memory size of' $(find / -iname "*error.log")
```

or

```
grep -i  'Fatal error: Allowed memory size of' $(locate "*.ini")
```

Check domain PHP memory_limit values:

```
grep -i 'memory_limit' $(find / -iname "*.ini")
```

or

```
grep -i 'memory_limit' $(locate "*.ini")
```

==========

Traffic Analysis

(File paths will need updating for these commands)

---

wp-login requests

```
grep -i 'wp-login' /path/to/access/log | cut -d ':' -f1 | sort | uniq -c
```

XML-RPC requests

```
grep -i 'xmlrpc' /path/to/access/log | cut -d ':' -f1 | sort | uniq -c
```

---

bot traffic user agents

```
 awk -F'"' '{print $6}' /var/www/vhosts/domain/logs/access_ssl_log | sort | uniq -c | sort -nr
| head -20 | grep -iE 'bot|crawler'
```

==========

MySQL Tuner

```
wget -O mysqltuner.pl mysqltuner.pl && perl mysqltuner.pl
```

==========

Bots/Crawlers & Control/Mitigation

==========

Robots.txt?

Robots.txt is a file that can be used on a server to decide which bots/crawlers can access the site as well as how often they can make requests. It's important to know that this file is essentially a guideline published to and recognised by bots/crawlers, outlined in this file are the 'rules' for crawling that domain - this file will NOT forcibly block bots/crawlers from making requests.

Below is an annotated example of how the robots.txt file could be configured:

```
# The below line defines the 'site map' for bots to reference. This is essentially a map of a domain, useful for boosting SEO also.
Sitemap: https://www.example.com/sitemap.xml #specifies the 'site map' for bots to discover all pages on a domain

# The below block specifies that all user-agents can crawl all pages, except from /login/.
User-Agent: *
Disallow: /login/


# This below line allows all search engines to crawl all pages except /login/
User-Agent: *
Disallow: /login/

# Alternatively, you can specify a specific search engine by user agent
User-Agent: Googlebot
Disallow: /admin/

#For sites with extreme crawling rates, it may be worth adding a 'crawl-delay' directive, as below:
User-Agent: Googlebot
crawl-delay: 10
```

==========

Identifying Bot Traffic

There are various sorts of server setups you'll encounter while attempting to decipher traffic logs. The below outlines the general methodology I use, however, these commands and approaches will likely need to be adjusted depending on your use-case.

Counting bot requests:

```
grep -iE 'bot|crawler' /path/to/access/log | wc -l
```

Identification of user-agents

```
awk -F'"' '{print $6}' /path/to/access/log | sort | uniq -c | sort -nr | head -20 | grep -iE 'bot|crawler'
```

==========

Bad Bots

Not all bots listen to rules defined within the robots.txt file, in cases where required, a server-wide block on 'bad bots' can be implemented.

---

cPanel

WHM > Apache Configuration > Includes Editor > Pre-main Includes > All Versions

```
<Directory "/home">
SetEnvIfNoCase User-Agent "petalbot" bad_bots
SetEnvIfNoCase User-Agent "baidu" bad_bots
SetEnvIfNoCase User-Agent "360Spider" bad_bots
SetEnvIfNoCase User-Agent "dotbot" bad_bots
SetEnvIfNoCase User-Agent "megaindex\.ru" bad_bots
SetEnvIfNoCase User-Agent "yelpspider" bad_bots
SetEnvIfNoCase User-Agent "fatbot" bad_bots
SetEnvIfNoCase User-Agent "ascribebot" bad_bots
SetEnvIfNoCase User-Agent "ia_archiver" bad_bots
SetEnvIfNoCase User-Agent "moatbot" bad_bots
```

```
SetEnvIfNoCase User-Agent "mj12bot" bad_bots
SetEnvIfNoCase User-Agent "mixrankbot" bad_bots
SetEnvIfNoCase User-Agent "orangebot" bad_bots
SetEnvIfNoCase User-Agent "yoozbot" bad_bots
SetEnvIfNoCase User-Agent "paperlibot" bad_bots
SetEnvIfNoCase User-Agent "showyoubot" bad_bots
SetEnvIfNoCase User-Agent "grapeshot" bad_bots
SetEnvIfNoCase User-Agent "WeSee" bad_bots
SetEnvIfNoCase User-Agent "haosouspider" bad_bots
SetEnvIfNoCase User-Agent "spider" bad_bots
SetEnvIfNoCase User-Agent "lexxebot" bad_bots
SetEnvIfNoCase User-Agent "nutch" bad_bots
SetEnvIfNoCase User-Agent "sogou" bad_bots
SetEnvIfNoCase User-Agent "Cincraw" bad_bots
<RequireAll>
Require all granted
Require not env bad_bots
</RequireAll>
</Directory>
```

Plesk

Apache

Nginx

==========

Alternative bot traffic control measures

In cases where bot traffic continues to rampage requests to a domain/s, it's worth advising the use of CloudFlares 'Bot Fight Mode'. Other CDN providers may well have their own offerings - worth advising the client of if they utilise an alternative network.

Resource usage and performance

==========

ATOP

==========

Nice value

In Linux, the nice value is a user-space and process-level mechanism that influences the scheduling priority of a process. It helps determine how much CPU time a process receives relative to other processes. The concept of "niceness" ranges from -20 (most favorable to the process) to 19 (least favorable to the process), with the default being 0.

Setting Nice Value for a new process

Nice value can be set when initially running the command

```
nice -n 15 command
```

Setting Nice value for an existing process

1234 needs to be replaced with the PID

```
renice -n 5 -p 1234
```

==========

xmlrpc & wp-login

==========

XMLRPC

Identifying the total number of xmlrpc requests:

```
grep -i 'xmlrpc' /path/to/access/log | cut -d ':' -f1 | sort | uniq -c | wc -l
```

Blocking/Disabling XMLRPC

Apache - .htaccess:

```
<Files ~ xmlrpc>
Order deny,allow
Deny from all
Allow from IP_IP_IP_IP #if required
</Files>
```

nginx -

==========

WP-LOGIN

Identifying the total number of wp-login requests:

```
grep -i 'wp-login' /path/to/access/log | cut -d ':' -f1 | sort | uniq -c | wc -l
```

Blocking/Disabling WP-LOGIN Access

Firstly, I personally recommend that all clients do the following:

1. Change the wp-login admin URL
2. Limit access to wp-login by IP

Changing wp-login URL:

Plugin: Changing the wp-login URL can be achieved using a plugin - such as 'Admin login URL Change'

Changing the wp-login.php file directly:

Edit the wp-login.php file and locate the string beginning with 'site_url':

```
action="<?php echo esc_url( site_url( 'wp-login.php?action=confirm_admin_email', 'login_post'
) ); ?>" method="post">
```

Following the site_url string in the example above, you'll see that wp-login.php is specified. Changing the string here will update the admin URL for that WordPress instance.

Limiting wp-login access by IP

Apache - .htaccess

```
<Files ~ wp-login>
Order deny,allow
Deny from all
Allow from IP_IP_IP_IP #if required
</Files>
```

nginx -

ANS IPs

Monitoring IPs

(Zabbix):

MAN4: 81.201.136.209
MAN5: 46.37.163.173

MAN5-Proxy: 46.37.163.133
81.201.136.192/27
94.229.162.0/27
81.201.136.168/29
46.37.163.128/26

LogicMonitor

46.37.179.0/28
81.201.138.160/28

Support Firewalls

80.244.179.100
185.197.63.204
46.37.163.116

Useful Online Tools

https://ping.pe/