CommonMisconfigurations

Common Apache Misconfigurations

This page will describe common misconfigurations as seen in #apache as well as describe why these are wrong.

Name Based Virtual Host

Not matching the value of NameVirtualHost with a corresponding <VirtualHost> block.

Example:

```
NameVirtualHost *:80

# This is wrong. No matching NameVirtualHost some.domain.com line.
<VirtualHost some.domain.com>
  # Options and stuff defined here.
</VirtualHost>

# This would be correct.
<VirtualHost *:80>
  ServerName some.domain.com
  # Options and stuff defined here.
</VirtualHost>
```

Why is the first virtual host wrong? It's wrong on a couple levels. The most obvious is that some.domain.com, used in the first <VirtualHost> block, doesn't match *:80 used in NameVirtualHost. The other is that NameVirtualHost refers to an interface, not a domain. For instance, using *:80 means all interfaces on port 80. NameVirtualHost 1.1.1.1:80 means address 1.1.1.1 on port 80. While you can use a "NameVirtualHost some.domain.com/<VirtualHost some.domain.com>" combination, it doesn't make much sense and is not generally used... at least not used by anyone experienced with Apache administration.

Reports in #httpd suggest that Webmin 1.510 (at least) may cause this issue.

Not setting a ServerName in a virtual host.

Example:

```
NameVirtualHost *:80

# This would be correct.
<VirtualHost *:80>
  ServerName some.domain.com
  # Options and stuff defined here.
</VirtualHost>

# This is wrong.
<VirtualHost *:80>
  # Options and stuff defined here, but no ServerName
</VirtualHost>
```

The second virtual host is wrong because when using name based virtual hosts, the ServerName is used by Apache to determine which virtual host configuration to use. Without it, Apache will never use the second virtual host configuration and will use the default virtual host. The default virtual host when using name based virtual hosts is the first defined virtual host.

Mixing non-port and port name based virtual hosts.

Example:

```
NameVirtualHost *
NameVirtualHost *:80

<VirtualHost *>
  ServerName some.domain.com
  # Options and stuff defined here.
</VirtualHost>

<VirtualHost *:80>
  ServerName some.domain2.com
  # Options and stuff defined here.
</VirtualHost>
```

Because NameVirtualHost * means catch all interfaces on all ports, the *:80 virtual host will never be caught. Every request to Apache will result in the some.domain.com virtual host being used.

Using the same Listen and/or NameVirtualHost multiple times.

Example:

```
# Can happen when using multiple config files.
# In one config file:
Listen 80
# In another config file:
Listen 80

# Like above, can happen when using multiple config files.
# In one config file:
NameVirtualHost *:80
# In another config file:
NameVirtualHost *:80
```

In the case of multiple Listen directives, Apache will bind to port 80 the first time and then try to bind to port 80 a second time. This yields a nice "Could not bind to port" error on start up. This seems to happen with newbies and Debian based distros, where Debian based distros have Listen 80 defined in ports.conf. Newbies don't realize this and create another Listen 80 line in apache2.conf.

Multiple NameVirtualHost lines will yield a "NameVirtualHost *:80 has no VirtualHosts" warning. Apache will ignore the second directive and use the first defined NameVirtualHost line, though. This seems to happen when one is using multiple virtual host configuration files and doesn't understand that you only need to define a particular NameVirtualHost line once. As above, this can occur in the debian ports.conf file, especially after an upgrade.

Multiple SSL name based virtual hosts on the same interface.

Example:

```
NameVirtualHost *:443

<VirtualHost *:443>
  ServerName some.domain.com
  # SSL options, other options, and stuff defined here.
</VirtualHost>

<VirtualHost *:443>
  ServerName some.domain2.com
  # SSL options, other options, and stuff defined here.
</VirtualHost>
```

See NameBasedSSLVHostsWithSNI for a detailed discussion, but in general most web browsers will work correctly with the above setup, historically Windows XP was the major operating system it would cause issues with.

When clients without SNI attempt to connect host information isn't used so Apache will always use the certificate of the default virtual host, which is the first defined virtual host for name-based virtual hosts. This means your users will get a certificate mismatch warning when trying to access some.domain2.com. Read more about this at http://httpd.apache.org/docs/2.2/ssl/ssl_faq.html#vhosts2

Scope

Adding/Restricting access and options in <Directory />

Example:

```
<Directory />
  # This was changed from the default of AllowOverride None.
  AllowOverride FileInfo Indexes
  # Default directives defined below.
</Directory>
```

<Directory /> is not a URL path. It is a filesystem path. Making changes in this <Directory> block will have no effect on your website DocumentRoot. In the example above, what might have been attempted was being able to use htaccess in the DocumentRoot. The problem being that the htaccess file will still be ignored because the AllowOverride is set in the wrong <Directory> block.

Changing the DocumentRoot value without updating the old DocumentRoot's <Directory> block

Example:

```
# Your old DocumentRoot value was /usr/local/apache2/htdocs
DocumentRoot /var/www/html
#
# This should be changed to whatever you set DocumentRoot to.
#
<Directory /usr/local/apache2/htdocs>
  # Options and access set here.
</Directory>
```

Access and options in Apache must be expressly given. Since there is no <Directory> block for the new document root that grants any access or options, you will get a permission error when you try to access your site.

Trying to set directory and index options in a script aliased directory.

Example:

```
ScriptAlias /cgi-bin/ /var/www/cgi-bin/
<Directory /var/www/cgi-bin>
  AllowOverride None
  Options Indexes ExecCGI
  DirectoryIndex index.cgi
  # Other options defined.
</Directory>
```

Script aliased directories do not allow directory listings specified with Options Indexes – this is a security feature. Also, script aliased directories automatically try to execute everything in them, so Options ExecCGI is unnecessary. The DirectoryIndex directive also does not work in a script aliased directory. The workaround, if you really need directory listings or other directory indexing options, is to use Alias instead of ScriptAlias.

Example:

```
Alias /cgi-bin/ /var/www/cgi-bin/
<Directory /var/www/cgi-bin>
  AllowOverride None
  Options Indexes ExecCGI
  AddHandler cgi-script .cgi
  DirectoryIndex index.cgi
  # Other options defined.
</Directory>
```

The options above will now work.

💡 TODO 💡

Add some more commonly seen stuff