World Wide Web

COSC301 Laboratory Manual

Today we will configure the most common web server on the planet: Apache. This lab will in no way give you enough experience to manage an Internet-facing webserver with dynamic content, and we don't have enough time to cover related issues such as web development. Indeed, there are multiple papers dedicated to precisely that. This is the first step down a very long road; an introduction to one aspect of a potent career path.

1. DNS Alterations

Web servers are commonly addressed using any of two common notations, either http://www.`domain`/ or http://`domain`/. We've configured this in the DNS lab already, but then our web server (which didn't actually exist) was known to DNS as goliah.

Change the DNS so that both `www.localdomain.` and `localdomain.` resolve to our server's addresses for both A and AAAA records[1]. You should *not* add an entry to the reverse zone, as we are defining aliases, and only canonical names go into the reverse zone. Remove the old entry for the non-existant Goliah. Use the following to test what you have done.

The `localdomain.` case is just a little bit tricky; remember that `@` is shorthand for the `$ORIGIN`, which in this particular file (as defined in `named.conf*`), is `localdomain.`, so either use a `@` or leading whitespace, just as we did when specifying the nameservers (`NS` record) for our zone.

```
Move to Server
# rndc reload
Check your logs to ensure it worked, then begin testing
$ dig -t A +short www.localdomain
192.168.1.1
$ dig -t AAAA +short www.localdomain
fd6b:4104:35ce::1
$ dig +short -x 192.168.1.1
server1.localdomain.
$ dig +short -x fd6b:4104:35ce::1
server1.localdomain.
$ dig -t A +short localdomain
192.168.1.1
$ dig -t AAAA +short localdomain
fd6b:4104:35ce::1
```

2. Install and Configure Apache

On the server, you will need to install the packages for Apache and PHP for this lab:

```
# apt-get install apache2 php links curl libapache2-mod-php
…
```

Configuring Apache includes giving it a few details on how it should run, and enabling the use of PHP. To make it easier to add drop-in functionality (via packages) to the web-server without requiring an administrator to manually edit configuration files, Debian based systems

[1] Alternatively, we could have made it so www has a CNAME of server1, but having duplicate A and AAAA records means we have greater flexibility, either to add other servers in round-robin DNS or to disable IPv6 by removing the AAAA record when accessing the web service.

modularise the Apache configuration directory to a greater extent than other distributions. **Read all of `/usr/share/doc/apache2/README.Debian.gz` before you proceed**, so you learn about how Apache's configuration is managed under a Debian-based distribution. Use **zless** to read the file.

Now, here are the configuration elements we want you to insert into the appropriate places ("appropriate" locations as per the `README.Debian.gz` file). In the instructions below, we tell you where to insert the elements. However, in case you can't find them, there is a general approach using **grep** to find them (see detail below).

1. Insert `ServerName server1.localdomain` into `/etc/apache2/apache2.conf` under the section of *global configurations*.

   This tells the server the canonical name of the host. It will be seen in error messages such as `404 File Not Found` responses when a requested web-page does not exist. It will try to automatically determine the value of this by looking up DNS, but you may find startup more reliable to specify it here, in the event that DNS is temporarily unavailable.

2. A web-server can often have multiple "virtual" sites, each site having a different name. The web-server offers different pages depending on which name it was called by. The default site is the site that is used if the name is not recognised as the name of a virtual site. The site configuration for the "default" site can be found in the file `/etc/apache2/sites-available/000-default`.

   Change `ServerAdmin` to `webmaster@localdomain` for the default site. This is a standard address that your email server should accept and should be redirected to a real person; we'll do that in a later lab.

3. It can be challenging to find where particular configuration elements are set because of the split-out management of the various files. The splitting of configuration elements helps the package managament system to manage Apache's configuration — facilitating drop-in configuration — but does make it harder to browse the configuration. To help with this, we can use **grep -r** to search recursively under `/etc/apache2/` which will tell us where we can find information.

   What is the value of `DirectoryIndex`?

```
$ grep -r DirectoryIndex /etc/apache2/
```

   Note that files under `/etc/apache2/mods-enabled/` are actually symbolic links to the real files under `/etc/apache2/mods-available/`. The same goes for the `sites-enabled` and `sites-available`.

   Assuming the value of `DirectoryIndex` were `index.php index.html`, when the server is asked for a directory (eg. `http://www.cs.otago.ac.nz/cosc301/`), this would cause the server to *first* look for `index.php` *then*, if `index.php` were not found, try `index.html` when it has been asked for a directory without a filename. There are often multiple types of files that might be tried, only the first found file is used. Thus, order matters.

4. Using the appropriate tool (ie. mentioned in `README.Debian.gz`), enable the `php7.2` module if it is not enabled already. You should not use a tool such as **ln** to do this manually, but use the tool designed for the job. You will find an example of using the tool in the section of *Virtual Hosts*.

5. Using the appropriate tool, disable the `cgi` module if it is not already disabled. What does this module do? Where can you find authoritative documentation about this module?

6. Using the appropriate tool, enable the `auth_digest` module, which we shall use later for authorisation. What command did you use?

Now check the syntax, restart the server and ensure that its listening.

```
# apache2ctl configtest
Syntax OK
# apache2ctl graceful
```

Check your logs in `/var/log/apache2/error.log` to make sure it started without complaint.

We better check out what it is listening for on the network, as we also want to have IPv6 connectivity as well.

```
# lsof -Pni
COMMAND … USER     … TYPE … NODE NAME
…
apache2 … root     … IPv6 … TCP  *:80 (LISTEN)   one of these
apache2 … www-data … IPv6 … TCP  *:80 (LISTEN)   and five of these
…
```

What do you notice about this? There are a few things we want you to pay particular attention to here. The first is that the server generally doesn't run as root, except at the beginning (the single entry is for starting other helper processes, it needs root permissions to bind to port 80). The helper processes, that actually serve the content, run not as the user "root", but instead run as the user "www-data". Why is that? (You'll need to write this down for the assessment).

The second major thing to notice is that all of **apache2**'s sockets are IPv6 sockets, there are no IPv4 sockets. You may be wondering why that is; surely we want to be able to reach our server over IPv4. This is a common feature of "dual-stack" servers ("dual-stack" means that a server runs both IPv4 and IPv6). The basic rule is this: if something goes to a port (say, port 80) on IPv4, but there is no IPv4 socket on that port, but there is an IPv6 socket on that port, then the connection goes passed to the IPv6 socket. It is important to realise that it is still IPv4 across the network.

You may think that's a bit odd. What's an IPv6 server going to do with an IPv4 connection? Well, the kernel translates it. The IPv6 server sees an IPv6 connection from the address `::ffff:192.168.1.113`. This special type of address notation, with both colons and dots, is called an "IPv4-mapped IPv6 address". The IPv4 address, in decimal is embedded in the lowest 32-bits of the otherwise hexadecimal address. This is done to make this sort of address instantly recognisable; this example would be otherwise written as `::ffff:c0a8:171`.

Some systems turn off IPv4-mapped IPv6 addresses by default

> Applications can explicitly enable or disable this feature, and the default is generally to have it enabled, although some systems will default to having this turned off. Debian, for example, leaves the Linux kernel default alone (ie. enabled), while Redhat turns it off via a sysctl.

This dual-stack transitioning support makes it easier to create network services because you only have to care about IPv6, and not necessarily IPv4 (although at some point you generally *do* need to care, but mostly when dealing with the addresses themselves).

On the client, try to connect to the web server using a web browser.

Note

We've sort of shot ourselves in the foot by only having one component in the domain name (`localdomain`). This is because, if we want to go to http://localdomain/, the browser sees that there is only one domain component, and so believes that it must surely be a non-fully-qualified domain name, and so the resolver on the client adds `localdomain` to the name. Thus, it looks up `localdomain.localdomain`, which fails. In the case of many web-browsers, it may then assume you really meant something like `www.localdomain.com` and then tries that.

To work around this, we can ask it to lookup `localdomain.` (note the trailing dot), which says the name is arleady fully qualified. Not all tools will accept that though.

You should see *Apache2 Ubuntu Default Page* shown in the browser. If so, that indicates that you can reach your web server. Let's do a few more tests, but this time, we'll watch the access logs in real-time as we do that. On the server, **tail -f /var/log/apache2/access.log**, and connect to the following with your browser on the client:

- http://www.localdomain/
- http://192.168.1.1/
- http://192.168.1.1:80/
- http://[fd6b:4104:35ce::1]:80/
- http://[fe80::a00:27ff:fe50:e093]/

  As expected, this may not work with Firefox (or any browser), as it would require a scope identifier, which is not allowed in URI syntax. Remember that Link-local addresses should not generally be used by applications, but as an administrator, you should not ignore them.

- http://server1.localdomain/
- http://server1.ipv4.localdomain/
- http://server1.ipv6.localdomain/

Okay, so we've verified basic operation and reachability. Let's go onto making some content and test the PHP component of our server.

3. Write Some Content

We won't assume that you can write HTML, or know any PHP, so copy the following example, just to show that PHP pages are indeed being processed.

Write this into `index.php` (note the `.php` suffix) in your document root (`/var/www/html`). Remove or rename the existing `index.html`.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>Bob's Construction Company</title>
```

```
</head>

<body>

<h1>Can we fix it?</h1>

<p><?php echo "Yes we can!"; ?></p>

</body>
</html>
```

Exercise

As an exercise view the site again. You need to click the reload button to force the refresh of the page. You should get a page titled “Bob’s Construction Company”. If you get a paragraph that says “Yes we can!”, then that means PHP is working. You may like to head on ever to www.php.net [http://www.php.net/] and follow a tutorial later.

4. Virtual Hosts

Virtual hosting is a common practice amongst web servers. In essence, it allows one server to host many different web-sites. The server must know which site the client is visiting. There are two different mechanisms for this: the first requires each site to have a different IP address (IP-based virtual hosts), and the server will be assigned many different IP addresses. This is a waste of valuable address space, and there is little reason for using this in todays environments, unless you are using HTTPS.

It is too early to see whether it will make much of a comeback with IPv6. In IPv6, each virtual site could easily have its own address (or even a whole bunch of them).

The way we do virtual hosting these days is to use name-based virtual hosts. This means the client sends along a `Host` header, which specifies the name of the server it believes it is talking to (it gets this from the hostname portion of the `URL`.

The biggest disadvantage of name-based virtual hosting is that it isn’t widely available yet for encrypted web-traffic. It’s only now becoming widely spread in modern browsers[2], so it will take a while for all the older devices to be replaced. This is because the SSL certificate is locked to the fully-qualified host-name the client is connecting to. The SSL session is established before HTTP is spoken, thus no `Host` header will have been sent and the server won’t know which certificate it should use.

We’re about to use **telnet** to interact with the web server using HTTP. However, by default, the version of Apache that ships with Ubuntu has a module enabled which imposes a limit on how long it will wait until it receives commands. This can be annoying for us slow humans, so we shall disable it.

```
# a2dismod reqtimeout
…
# /etc/init.d/apache2 restart
…
```

The bare minimum for a HTTP/1.1 request looks like the following. Try this using `telnet` to your web-server. Here we are talking the HTTP version 1.1 protocol. The 1.0 protocol is pretty much the same, but the `Host` header is not required.

[2] It is an extension to TLS called Server Name Identification. The Wikipedia has a good writeup about client and server coverage.

```
$ telnet localdomain. 80
Trying fd6b:4104:35ce::1...
Connected to localdomain.
Escape character is '^]'.
Request type, request path & arguments, HTTP version.
GET / HTTP/1.1
This let's the server know which virtual host to use.
Host: localdomain
Blank line to signify end-of-headers.

HTTP/1.1 200 OK
Date: …
Server: …
…
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>Bob's Construction Company</title>
</head>
…
The server will keep the connection open briefly…
… this is to allow the client to make another request.
Connection closed by foreign host.
```

Let's review what's going on here.

`telnet localdomain. 80`
: We connect to port 80 on the web server.

`GET / HTTP/1.1`
: We use the `GET` command to retrieve the document corresponding to `/`, which will be `index.php` in our case. We specify a version of the HTTP protocol, specifying 1.1.

`Host: localdomain`
: We send a `Host` header, which is required for HTTP/1.1. This is what the server can use to determine the *site* being visited, as the client puts the name of the server it thinks it is trying to connect to here.

Blank line
: In HTTP a blank line signifies that we are done sending headers, and it is the servers turn to send data.

`HTTP/1.1 200 OK` … blank line
: The server sends us back a set of headers, followed by a document. The `200 OK` part tells the client software the request was processed without any problems. One other well-known return code is 404: page not found.

Timeout at end
: This server is doing pipelining, which means we can make multiple requests per connection, making the transfer much faster. The server will keep the connection open to allow the client to send through another request, preventing another lengthy TCP connection setup and teardown[3]. After a while with no input, it will close the connection.

Let's create a virtual host for the new domain `boogaloo.nz`. For a public internet, you would need to purchase the domain name first, and have somewhere to house the server, or buy some web-hosting services. We won't be doing any of that in our private networks.

[3] Not to mention the TCP Slowstart algorithm for flow control.

1. Create a new domain called `boogaloo.nz` by adding a forward zone to `named.conf.local` on the DNS server, and copying the `db.localdomain` file to `db.boogaloo`, modifying it to suit the following requirements:

   - We want `http://boogaloo.nz` and `http://www.boogaloo.nz` to be valid points of access to the web server, which should point to our server's IPv4 and IPv6 addresses.
   - We don't need to touch the reverse zones. They will remain pointing to entries in localdomain. This is because all the data in `boogaloo.nz` is alias data, and as such doesn't need to go into the reverse zone[4].

2. Test the new DNS entries.

```
# /etc/init.d/bind9 restart
Starting BIND:  /usr/sbin/named
Check logs, ensure it is still running...
$ dig -t A +short boogaloo.nz
192.168.1.1
$ dig -t AAAA +short boogaloo.nz
fd6b:4104:35ce::1
$ dig -t A +short www.boogaloo.nz
192.168.1.1
$ dig -t AAAA +short www.boogaloo.nz
fd6b:4104:35ce::1
```

3. Create a file `/etc/apache2/sites-available/boogaloo.conf`. Inside it, write the following.

```
<VirtualHost *:80>
    ServerName boogaloo.nz
    ServerAlias www.boogaloo.nz
    ServerAdmin webmaster@boogaloo.nz
    DocumentRoot /var/www/boogaloo
    ErrorLog /var/log/apache2/boogaloo-error.log
    CustomLog  /var/log/apache2/boogaloo-access.log combined
</VirtualHost>
```

4. Create the document root for the virtual host.

```
# mkdir /var/www/boogaloo
```

5. Enable the site in the Apache configuration.

```
# a2ensite boogaloo
```

6. Test the configuration file syntax, and restart the server. If it fails to start, you should check the error log for Apache which can be found in `/var/log/apache2/error.log`.

```
# apache2ctl configtest
Syntax OK
# apache2ctl graceful
# tail /var/log/apache2/error.log
```

7. Create a simple `index.html` file in the docroot for boogaloo.nz.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
```

[4] While it is technically possible to have multiple PTR records, no software will expect that and you would be likely to get a round-robin result, which is not useful.

```
<html>
<head><title>Boogaloo</title></head>
<body>
  <h1>Funky!</h1>
</body>
</html>
```

8. **Exercise**
   As an exercise point your web-browser to your new site, to both points of access in your new domain, as well as ensuring that your old domain still works as expected.

5. Self-assessment

1. Why does the web server run as the user www-data? What rights does that user have to the web page content?

2. In addition, do some research and find some best practices for securing Apache. Write down at least five of them. Make sure you understand what they do.

6. Last Words

Web services, and especially applications that run on top of them, are some of the most often attacked services. **Do not** deploy them on the Internet without being very familiar with how they work and the real world issues. Consulting the Apache Documentation [http://httpd.apache.org/docs/] is a great place to start becoming more familiar with the Apache webserver. Start reading some of the material at www.sans.org and all web-developers should at least be familiar with the OWASP Top 10 [http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project] web vulnerabilities.

7. [Optional] Authentication and Authorisation

Running short on time?

The remainder of this lab is optional, so if you don't have enough time, you can just stop right here.

Often, you want to limit access to material on your web-server using machanisms such as IP address ranges, or username and password. For this section, we'll have a quick look at username authentication, using the Digest authentication mechanism we enabled earlier.

There are two authentication mechanisms in common use, Basic and Digest. Basic doesn't encrypt the password, it just encodes the username and password values in Base64, which is easily decoded by anyone. Digest authentication makes a MD5 digest (hash) of the username, password and other things that the server decides. The full details are available in RFC 2617[5].

You have to remember however, that the data is still transmitted in the clear. If you need to have strong security, you should be using SSL/TLS. Check the Apache website for more information, particularly the Authentication, Authorisation and Access Control document.

[5] http://ftp.ics.uci.edu/pub/ietf/http/rfc2617.txt

We'll proceed by adding a private directory to the `boogaloo.nz` domain we added in the previous section. We'll then configure Apache to give access only based on user-name and password, using Digest authentication.

1. Create a password file `/etc/apache2/boogaloo-digest` with a username and password for yourself. This can be done using the following command. Notice that the password file is outside the document root of the webserver. You should briefly check the manual page for this command to see what the arguments and options are.

```
# htdigest -c \
> /etc/apache2/boogaloo-digest \
> "Boogaloo Employees Only" theauthor
Adding password for theauthor in realm Boogaloo Employees Only.
New password: Not echoed
Re-type new password:
```

2. Have a look at the generated password file to see what is inside.

3. Create a directory where we will put our protected material.

```
# mkdir /var/www/boogaloo/private/
```

5. In that directory, create a file called `.htaccess`, and place in it the following contents. The `AuthName` must match the realm you used above. The `Require` directive states that a username and correct password is required. Note that we can either specify that we require any valid user, or specify that only certain users may enter.

   On the server we have configured, `.htaccess` files are used by default. These are files that change Apache's configuration for particular directories, often to limit access. What such a file is allowed to reconfigure is listed in the main server configuration.

```
AuthType Digest
AuthName "Boogaloo Employees Only"
AuthUserFile /etc/apache2/boogaloo-digest
Require valid-user
Alternatively, specify a set of users
# Require user asuka sanjay jamilah michael
We could also set up groups of people
```

6. In order to allow use of `.htaccess` files to configure authentication, you need to add some lines to the `VirtualHost` stanza in `sites-available/boogaloo`.

```
<VirtualHost *>
    ServerName boogaloo.nz
    ServerAlias www.boogaloo.nz
    ServerAdmin webmaster@boogaloo.nz
    DocumentRoot /var/www/boogaloo
    ErrorLog /var/log/apache2/boogaloo-error.log
    CustomLog /var/log/apache2/boogaloo-access.log common
    <Directory /var/www/boogaloo>
        AllowOverride AuthConfig Limit
    </Directory>
</VirtualHost>
```

   It is important to note that normally we would prefer to put things under `/etc/apache2/` files rather than in `.htaccess` files. Here, I have used `.htaccess` files to demonstrate the concept. They can be useful in certain situations.

7. Optionally, you can also create groups by specifying a group file to use, which means you can limit access to a group of users. These are not system groups however, but rather

specified in a seperate file, in much the same way the users and their encrypted passwords are.

8. Create a file `index.html` in the private directory, and point your browser to the private area of the Boogaloo website.

   Make sure you get the output above, and not an HTML error page.