Network Services

Intermediate UNIX System Administration DeCal

Lab 7 — 29 March 2010

by Jordan Salter, with thanks to Joshua Kwan

Introduction

This week's lecture gave an overview of Internet services — how you access them, and how remote servers provide them; this lab is a continuation of that overview. You should do these exercises on your virtual servers, except where otherwise noted; answers should be mailed to `mgasidlo+decal@OCF.Berkeley.EDU`.

1 Interacting with Mail Locally

Let's start by poking around with some standard UNIX mail commands. You should do these exercises on one of the OCF's Solaris servers (`apocalypse` or `conquest` — I recommend the latter), since we've already configured `sendmail` for you.

1. Use the `mail` command to send yourself a message. How do you specify a subject? How do you read your mail using this command?

2. Play around with `pine` and `mutt` and figure out how to read and send mail messages in both. Which do you prefer, and why?

3. "Real men" use `ed`, the standard text editor, over `emacs` and `vi`. The mail client war is a bit like the editor war, and "real men" send mail with `netcat` (`telnet` works in a pinch). See what you think — try using `sendmail -t` or `sendmail -bs` (what's the difference?) to manually send a message to your neighbor. Include a transcript of your session in your submission.

2 DNS queries

Before you begin, install the `host` and `dnsutils` packages.

1. Use `host` to look up the MX record for Berkeley.EDU. Show the syntax you used.

2. There are *many* different types of DNS records. What's an NS record, and what are Berkeley.EDU's NS records? Where are the servers Berkeley's NS records point to actually located, and why might IST have set things up that way?

3. Use `dig` to perform the same queries as in questions (1) and (2), and show the syntax you used. Then run `dig` with no arguments — what do you see in the "ANSWER SECTION"? What are these servers?

3 HTTP

(You'll want to install `netcat` on your virtual server before continuing.) In this part, you'll be writing a simple shell script that downloads the contents of an arbitrary URL, saving the result to the file "`output`." Assume that the URL your script is passed is in the format

```
http://www.example.com/path/to/document
```

(don't worry about escapes or special characters). Here's what your script should do:

1. Use `cut` to extract the domain name (`www.example.com`) and the path (`/path/to/…`). Bear in mind that a valid path can contain an arbitrary number of slashes.

2. Use `nc` to connect to the server on port 80. (You can pipe input into netcat's stdin.) An HTTP request looks like this:

   ```
   GET /path/to/document HTTP/1.1
   Host: www.example.com
   (newline)
   ```

   Use `echo` and the "enable interpretation of backslash escapes" option — your HTTP request, on one line, will look something like

   ```
   GET /path/to/document HTTP/1.1\nHost: www.example.com\n\n
   ```

   (note the *two* concluding newlines).

3. Pipe this input into a read loop (like you did with `rename-tv`). We're not interested in the HTTP headers, which are terminated by a blank line, so you should discard lines until you find an empty one. (To check if you've found an empty line, test whether it's equal to the output of `echo -e '\r'` — try comparing it to `$'\r'`.) Once you've found a blank line, `echo` every remaining line.

4. Save the output of this read loop to a file called "`output`."

Before you begin, try out `nc` and `echo -e` on their own and make sure you understand how they work.

Extra for Experts™!

Once you're done, if you're feeling adventurous, you might try…

- The filename "`output`" is not very descriptive. Make your script save its output somewhere more useful.

- Look at the Wikipedia article on HTTP status codes. The first HTTP header your script receives includes a status code — common ones include "`200 OK`," "`301 Moved Permanently`," "`404 Not Found`," and so on. Build in support for a few of these status codes — maybe follow redirects (look at the `Location:` header), or print a warning when you encounter a `404`.
- If you're a shell-scripting ninja, you finished the entire lab in ten minutes flat, and you're bored, read the appropriate RFCs and write a `bash`-based IRC client.