CS2021 Week 12

Web Development – Building Login System

Solution: Cookies and Hashing in Python

https://www.udacity.com/wiki/cs253/unit-4

Using Cookies for Authentication

- Cookies are small pieces of data stored in the browser--each piece of data is for a particular website.
- Cookies are used to determine whether a user is logged into a

particular website

After a request to a web server, the server may send back some cookie data in the form of an HTTP header in its response.

Each time a user makes a request to that website in the future, the browser will send the cookie data back to the server.

Client browsers control what cookies are sent to which websites – browsers have rules that associate cookie data with a particular domain.

Users can *hack* cookies -- how

bad is that?

More About Cookies

Cookies are sent in HTTP headers. When a server wants to send a cookie to your browser, it sends an HTTP response header that looks something like this:

Set-Cookie: user_id = 12345
Set-Cookie: last_seen = Dec 25 1985

When client sends multiple cookies to server they are separated by a semi-colon as

follows:

Cookie: user_id = 12345;
last_seen = Dec 25 1985

Cookies in App Engine: Counting visits

```
#Let's begin with the basic app template that we are all familiar with:
import os
import webapp2
import jinja2
from google.appengine.ext import db

template_dir = os.path.join(os.path.dirname(__file__), 'templates')
jinja_env = jinja2.Environment(loader = jinja2.FileSystemLoader(template_dir),
                               autoescape=True)
class Handler(webapp2.RequestHandler):
    def write(self, *a, **kw):
        self.response.out.write(*a, **kw)

    def render_str(self, template, **params):
        t = jinja_env.get_template(template)
        return t.render(params)

    def render(self, template, **kw):
        self.write(self.render_str(template, **kw))

class MainPage(Handler):
    def get(self):
        self.write('test')

app = webapp2.WSGIApplication([('/', MainPage)], debug=True)
```

Hashing Cookies Can Protect a Website

Hashing is a technique used to verify the legitimacy of data.
Our website should know when users try to cheat – want to prevent users from modifying cookies.

A hash is a function, let's call it H(), which when applied to a piece of data, x, returns a fixed-length bit-string, y.
H(x) → y x is data, y is fixed-length bit-string
x can be of any size data, but y is of fixed length and depending on the algorithm used is usually on the order of 32 - 1024 bits long.

Important Properties of Hash function H():

Given an cookie that is a hashed value y:

Make it easy to test whether a given data x hashes to y: that is H(x) =y

Make it impossible to identify an inverse – that is, hard to locate a piece of data that hashes to a specific value of y.

The hash function should intuitively be a 'one-way' function server controls.

Hashing In Python

To perform hashing in Python we can use the hashlib library. This library incorporates a number of hashing functions:

md5() 128 bits
sha1() 160 bits
Sha2 family of functions with various digest sizes

Example: hash a string "Hello World!" in Python, we would enter:

```
import hashlib
x = hashlib.md5("Hello World!")
```

Hashing Examples

```
>>> import hashlib
>>> x= hashlib.md5("Hello World!")
>>> type(x)
<type '_hashlib.HASH'>
>>> y=hashlib.sha1("Hello World!")
>>> x.hexdigest()
'ed076287532e86365e841e92bfc50d8c'
>>> y.hexdigest()
'2ef7bde608ce5404e97d5f042f95f89f1c232871’
```

But if we change a single letter, we now get a completely different result from the hash function:

```
>>> z=hashlib.sha1("Hello World")
>>> z.hexdigest()
'0a4d55a8d778e5022fab701977c5d840bbc486d0'
```

A nice thing about MD5 and SHA1 is that they are available on every system, and if I hash the phrase "Hello, World" on any system I will get the same result.

Hashing Cookies to prevent cheats

Now we use hashing to prevent people from cheating with our cookies.
The algorithm will look something like this:

Instead of simply saying:
Set-Cookie: visits=5

we will add a hash of the value, something like:
Set-Cookie: visits=5|
    e4da3b7fbbce2345d7772b0674a318d5
That is setting visits to 5|[our special hash of “5”]

When we receive the cookie, we just hash the value and compare it with the hash to ensure that the value hasn't been tampered with:

```
if H(value) == hash:
    valid
else:
    discard...
```

Putting It All Together

We can now restructure our program to use our new secure functions. First, we add the functions themselves to our program:

```
import hashlib

def hash_str(s):
        return
hashlib.md5(s).hexdigest()

def make_secure_val(s):
```

```
        return "%s|%s" % (s,
hash_str(s))

def check_secure_val(h):
        val = h.split('|')[0]
        if h ==
make_secure_val(val):
                return val
        else:
                return None
```

MainPage() looks like this:

```
class MainPage(Handler):
        def get(self):
                self.response.headers['Content-Type'] =
'text/plain'
```

```
        visits = 0
        visit_cookie_str =
self.request.cookies.get('visits')
        if visit_cookie_str:
            cookie_val =
check_secure_val(visit_cookie_str)
            if cookie_val:
                visits = int(cookie_val)
        visits += 1
        new_cookie_val =
make_secure_val(str(visits))
        self.response.headers.add_header('Set-
Cookie', 'visits=%s' % new_cookie_val)
        if visits > 100:
            self.write("You are the best ever!")
        else:
            self.write("You've been here %s times!"
% visits)
```

Inspect your hashed cookies

When we run this in the browser a few times and check the cookie we see the value is now:

5|
e4da3b7fbbce2345d7772b0674a
318d5

Hash-based Message Authentication Code

This is basically a special algorithm, built into Python, for when you want to combine a key with your value to create a hash. HMAC looks something like this:

hmac(secret, key, H) = [HASH]

H is the hashing function.

To see how this works in the Python interpreter, we can hash the message "udacity" with the keyword "secret" as follows:

```
>>> import hmac
>>> hmac.new("secret",
"udacity”,
hashlib.md5 ).hexdigest(
)
'fd4c2d860910b3a7b65c576
d247292e8’
```

```
SECRET =
'IAMAserversecret'
import hmac
def hash_str(s):
    return
hmac.new(SECRET,
s).hexdigest()
```

Passwords

We can use hashing, and the HMAC variant of hashing, to make cookies that won't be tampered with.
Let's say that our database has a table for users. This table has one column for the username and the other column for the password:

In order to check that a user is valid, we have a function something like this:

```
def valid_user(name, pw):
    user = get_user(name)
    if user and user.password == pw:
        return user
```

Password Hashing because Databases gets compromised!

Have you given away all your user's passwords! This means that, not only are your users going to be angry because you have compromised their privacy, your website is also in

trouble because you have bad-guys logging in and messing around with people's accounts because they now know everybody's passwords.

To protect the passwords, instead of storing the actual plaintext passwords in our database, we can store a password hash in the database:

Now, if our database gets compromised, all the attacker has is a bunch of password hashes, and we have already seen that it is very, very difficult, if not impossible, to turn the hash of a string back into the original string.

Our user validation function also changes to compare the hash of the password entered by the user with the password hash stored in the database:

```
def valid_user(name, pw):
    user = get_user(name)
    if user and
user.password_hash == H(pw):
        return user
```

HW: Add User Registration System to your Blog