CS253 - Web Application Engineering

Unit 4: User Authentication and Access Control

Contents

Introduction

This unit is about authentication. We are going to be learning about hashing and we will see some of the clever uses of hashing.

We will talk about cookies, which are small pieces of data used to store data on the client, or in the browser, and which are the basis for login systems and registration. Then we will go on to see how you can store passwords so that you can build a complete login system for your blog.

This will be your homework task for this unit.

Cookies

Cookies are small pieces of data stored in the browser for a website. 'Small' in this context means less than 4 kilobytes, and in practice, typically only a hundred bytes. A cookie is really just a simple string. Conceptually, cookies take the format:

name = value

In practice, this might look something like this:

user_id = 12345

Cookies are really commonly used for the kinds of temporary information that needs to be stored by the browser. A good example of this type of information is whether you are logged into a particular website. Your browser will store a cookie that shows you are logged in as (for example), user 12345.

When the browser makes a request to a web server, the server may send back some cookie data in the form of an HTTP header in its response. The browser just stores the cookie, and each time it makes a request to that website in the future, the browser will send the cookie data back to the server.

You can generally save up to about 20 cookies per website. This is determined by the browser.

We have already said that cookies must be less than 4 kilobytes. In practice they should always be much less. There is a lot that can go wrong with longer cookies. Some browsers don't handle multi-line cookies properly. Indeed, some web-servers don't handle multi-line cookies properly!

Another limitation is that a cookie must be associated with a particular domain. A cookie for udacity.com is only sent to udacity.com, and udacity.com can only set cookies for udacity.com.

These are all enforced on the client-side by the browser.

Good Uses For Cookies Quiz

What are good uses of cookies?

- Storing login information.
- Storing small amounts of data to avoid hitting a db.
- Storing user preference information.
- Tracking you for ads.

Cookie Headers

As we said earlier, 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
```

The **Set-Cookie** header will set the cookie named user_id to the value 12345. If the server wants to send multiple cookies, it simply needs to send multiple Set-Cookie headers:

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

The server can send as many cookies as it wants. It is up to the browser to decide whether or not it stores them! For this reason, keep cookies to less than 20.

In future requests, the browser will then send its own header within the request:

```
Cookie: user_id = 12345
```

If the browser will be sending multiple cookies, they are separated by a semi-colon, thus:

```
Cookie: user_id = 12345; last_seen = Dec 25 1985
```

It is therefore a good idea to avoid using semi-colons in the values of your cookies! If you really need to incorporate semi-colons in your cookie values, use some encoding on the cookie values to ensure that you escape the semi-colons otherwise you will effectively corrupt the incoming cookie header.

Wikipedia page on HTTP Cookies

Cookie Examples

Steve used telnet to view the header information in the HTTP response from Google.com. The cookies below appear:

Set-Cookie:
PREF=ID=9dc1d7062ae5fd16:FF=0:TM=1336504404:LM=1336504404:S=KVV_FUsYL5CImBd4; expires=Thu, 08-May-2014 19:13:24 GMT; path=/; domain=.google.com
Set-Cookie: NID=59=poGaObJlWuDDoF4zB0fxk2gYic2IT66JXfwFVTYg0Yx1lV_BwI9G_-1NNAKFi9B1eqlAlxDAyJnAEFKCIkg0PtfrdKu9dcsKTzljNqgPIjuTYP3rGAuK6L5suIyp57y-; expires=Wed , 07-Nov-2012 19:13:24 GMT; path=/; domain=.google.com; HttpOnly

The first cookie is named PREF and is set to the value:

ID=9dc1d7062ae5fd16:FF=0:TM=1336504404:LM=1336504404:S=KVV_FUsYL5CImBd4

This is actually an example of Google storing multiple pieces of data in one cookie. The cookie value is terminated with a semi-colon, and is then followed by some additional parameters (each also separated by semi-colons). The first of these is the expires time:

> expires=Thu, 08-May-2014 19:13:24 GMT

After this time, the cookie will have expired and will no longer be sent.

The cookie is only valid for the path and domain set by the parameters:

> path=/; domain=.google.com

The next cookie is named NID, and includes similar parameters.

Steve then demonstrated that Linux/Unix or Mac users can use the curl –I command to view the HTTP headers instead of telnet.

One other way to view HTTP headers is to use the debug tools in your browser. Steve demonstrated these tools using Google Chrome.

Sending Cookies Quiz

Which header does a browser use to send a cookie to a server?

Setting Cookies Quiz

Which header does a server use to set a cookie?

Cookie Domains

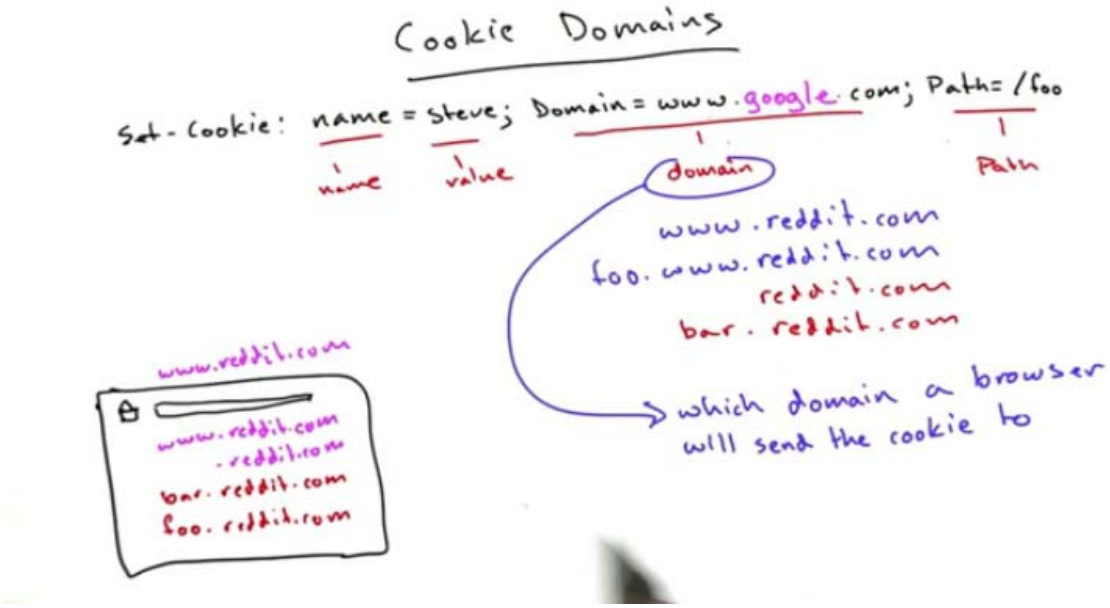
As we have seen, cookies can have additional parameters, not just the value. Consider the following Set-Cookie header:

Set-Cookie: name = Steve, Domain = www.redit.com; path = /;

In this case, the name parameter is '**name**', and its value is '**Steve**'. This is followed by a parameter named '**Domain**' with the value '**www.redit.com**' which is the domain that this cookie is relevant to. Lastly, there is the parameter named '**path**' which has the value '/' ('/' is always the default path).

Let's look at the Domain parameter. With this parameter specified, the cookie will not be sent to the server unless the server's domain is the same as the value of the Domain parameter or ends with the value of the Domain parameter, in this case, '**www.redit.com**'.

So, in this case, the cookie will be sent to **www.reddit.com** and to **foo.www.reddit.com**, but it will not be sent to **bar.reddit.com** or even to just **reddit.com**.



Valid Receivers Quiz

Which of these domains would receive this cookie?

Set-Cookie: user=123; Domain = ide.udacity.com

- udacity.com
- ide.udacity.com
- other.ide.udacity.com
- other.udacity.com

Valid Setters Quiz

Which of these domains could set this cookie?

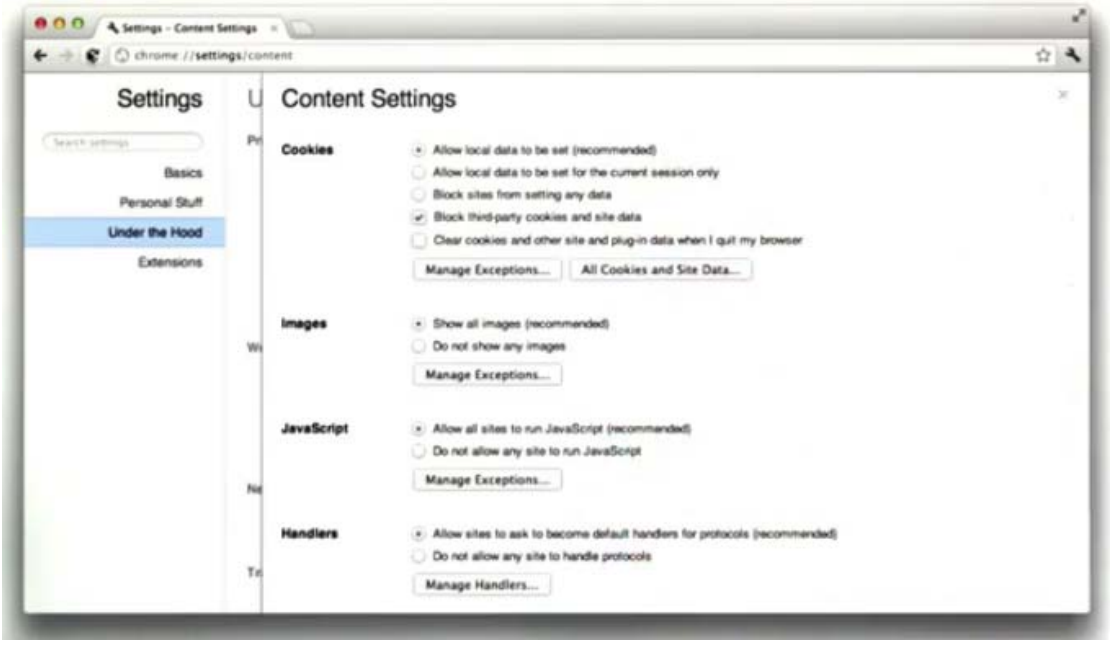
> Set-Cookie: user=123; Domain = ide.udacity.com

- udacity.com
- ide.udacity.com
- other.ide.udacity.com
- other.udacity.com

Cookie Settings

Browsers will normally all have a preferences page. These will allow you to set your personal preferences for how the browser uses cookies.

When designing web applications you should never assume that cookies will always be allowed, and that your cookies will always be there, since your users may have modified the cookie settings in their browsers.

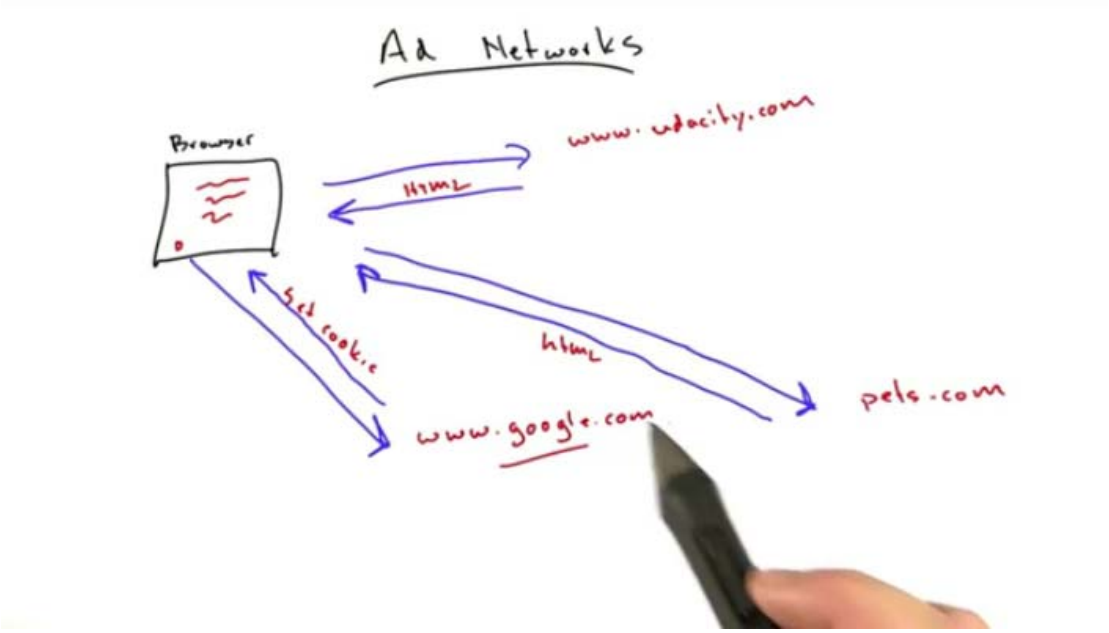


Ad Networks

Let's take a moment to consider Ad Networks. These aren't strictly relevant to this class, but they do provide an interesting illustration of how cookies can be used.

Let's say that your browser makes a request to some website, and that website responds with a page of HTML. One of the things in this HTML could be a little 1-pixel hidden image, that you do not even see, but which makes a request to another site, for example, www.google.com.

This does actually happen. Google provides an analytics package that a lot of websites use to track traffic and what users are doing. Google may respond with a cookie that assigns you an i.d. so that when you come back to the website again, Google can track that you are the same user returning to the site. This is an example of a "3rd-party-cookie".



This is a quite legitimate use of cookies. Google analytics is used by Udacity, Reddit and Hipmunk to monitor traffic and so forth. However, it does mean that Google is able to build up quite a lot of data about the sites that you are visiting.

Cookie Expiration

Let's consider the following cookie:

Set-Cookie: user = 123; Expires = Tue, 1 Jan 2005 00:00:00 GMT

The extra parameter '**Expires**' sets an expiration date for the cookie. The browser will hang onto the cookie until the given expiry date, in this case, **Tue, 1 Jan 2005**.

If no 'Expires' parameter is provided, the cookie will expire when the browser window is closed. In this case, the cookie is known as a **session-cookie**. A common example of this is the remember me check-box on a login box. If the box is checked, then an Expires parameter is set for the cookie. If not, the cookie is as session cookie.

Cookie Deletion Quiz

When does a cookie with no Expires parameter get deleted from your browser?

- Jan 1, 2025.
- Never.
- When you close your browser.
- In 1 day.

Cookies In App Engine

What we will do now is to build a little web application that uses cookies to keep track of how many times you have visited a web site. This will illustrate how to use cookies in App Engine and highlight a few things that you should bear in mind.

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)
```

The first thing that we want to do is to set the content type to text so that we don't have to deal with HTML at this stage, by adding the line:

```
self.response.headers['Content-Type'] = 'text/plain'
```

to the class MainPage(). The next thing we need to do is to add a line to get a cookie, called **visits**:

```
visits = self.request.cookies.get('visits', 0)
```

The way this works is to look at the **request** object, which is always on **self**. The request object will have a **cookies** object. This is a dictionary-like object where App Engine stores the cookies when it parses the HTTP headers. We call the dictionary function get on the **cookies** object to see whether the key '**visits**' is in the dictionary. If it is, we the get the value of the key, and if not then we get the default value which we have set to be 0.

Lastly, we will add a line to show the user how many times they have visited the site:

```
self.write("You've been here %s times!" % visits)
```

Our MainPage() class now looks like this:

```
class MainPage(Handler):
    def get(self):
        self.response.headers['Content-Type'] = 'text/plain'
        visits = self.request.cookies.get('visits', 0)
        self.write("You've been here %s times!" % visits)
```

So far so good. When a user visits our page they will see the text:

```
You've been here 0 times!
```

Now what we need to do is to update the value of visits each time that the user visits our site and then store the updated value in the cookie:

```
class MainPage(Handler):
    def get(self):
        self.response.headers['Content-Type'] = 'text/plain'
        visits = self.request.cookies.get('visits', 0)
        visits += 1
        self.response.headers.add_header('Set-Cookie', 'visits=%s' % visits)
```

So, we increment the value of visits and then call add_header to add the **Set-Cookie** header. We won't worry about domain or path for now.

When we re-load the page we get:

```
You've been here 1 times!
```

which is good, but when we refresh we get an error!

Internal Server Error

The server has either erred or is incapable of performing the requested operation.

```
Traceback (most recent call last):
  File "D:\Program Files (x86)\Google\google_appengine\lib\webapp2\webapp2.py", line
1536, in __call__
    rv = self.handle_exception(request, response, e)
  File "D:\Program Files (x86)\Google\google_appengine\lib\webapp2\webapp2.py", line
1530, in __call__
    rv = self.router.dispatch(request, response)
  File "D:\Program Files (x86)\Google\google_appengine\lib\webapp2\webapp2.py", line
1278, in default_dispatcher
    return route.handler_adapter(request, response)
  File "D:\Program Files (x86)\Google\google_appengine\lib\webapp2\webapp2.py", line
1102, in __call__
    return handler.dispatch()
  File "D:\Program Files (x86)\Google\google_appengine\lib\webapp2\webapp2.py", line
572, in dispatch
    return self.handle_exception(e, self.app.debug)
  File "D:\Program Files (x86)\Google\google_appengine\lib\webapp2\webapp2.py", line
570, in dispatch
    return method(*args, **kwargs)
  File "E:\Applications\cookies\main.py", line 25, in get
    visits += 1
TypeError: coercing to Unicode: need string or buffer, int found
```

We have a Type Error and, in fact, when we think about it, this makes sense.

The browser doesn't care what data-type a cookie is. However, we specified the value of visits to be the **string** %s: **visits=%s** and then we tried to use it as an **integer** in the expression **visits += 1**.

We can modify the MainHandler() class to manage the data type as follows:

```
class MainPage(Handler):
        def get(self):
                self.response.headers['Content-Type'] = 'text/plain'
                visits = self.request.cookies.get('visits', '0')
                #make sure visits is an int
                if visits.isdigit():
                        visits = int(visits) + 1
                else:
                        visits = 0

                self.response.headers.add_header('Set-Cookie', 'visits=%s' % visits)

                self.write("You've been here %s times!" % visits)
```

Now let’s do something with the cookie value. Let’s say that we want to reward users with a special message of thanks if they have visited our website more than 100 times. We can do this quite simply as follows:

```
if visits > 100:
        self.write("You are the best ever!")
else:
        self.write("You've been here %s times!" % visits)
```

Now, users who have visited more than 100 times will see the message “**You are the best ever!**”, and everyone else will see how many times they have visited the site as before.

Cheating Quiz

What can we do to get 10,000 visits?

- Reload the page 10,000 times.
- Send the link to 10,000 friends
- Edit the cookie in our browser.

How To Cheat

Cookies can be edited in most browsers, and their values changed. The exact method varies from browser to browser.

So how do we prevent users from cheating like this?

Hashing

We are going to talk about a technique called **hashing**. Hashing is a technique that we can use to verify the legitimacy of our data.

So what is a 'hash'?

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**:

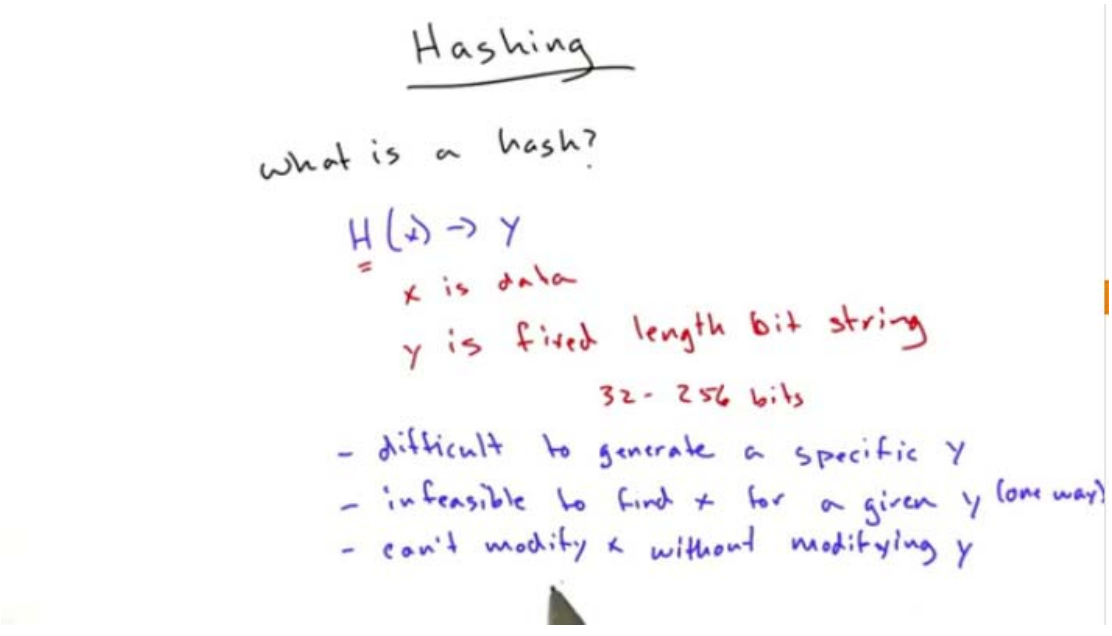
H(x) → y

x is data
y is fixed-length bit-string

x can be of any size. y can be of arbitrary, fixed length, but, depending on the algorithm used, y is usually on the order of 32 – 256 bits long (certainly with the common algorithms that we will be dealing with).

Let's consider some of the properties of the hash function, **H()**:

- It is generally very difficult to identify a piece of data that hashes to a specific value of y.
- It is infeasible to find a particular value of x for a given output value of y.
    - The hash function is a 'one-way' function.
- You can't modify x without modifying y.



Hash functions are covered in much more detail in the Udacity course CS387 – Applied Cryptography.

Hash Algorithms

There are many popular hash algorithms. In general, if you are going to use hashing for security purposes, **DON'T WRITE YOUR OWN HASH ALGORITHM!**

Some popular algorithms are

- CRC32 – Designed for checksums. Fast.
- MD5 – Relatively fast, but not secure. Still the most popular hashing algorithm.
- SHA-1 – Not as fast as MD5. Fairly secure. 2nd most popular algorithm.
- SHA256 – Pretty good security, but slower.

Collision: – when two things hash to the same value.

Hashing In Python

Let's start looking at how we can do hashing in Python, since we are going to be doing a lot of it in this class.

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

- md5()
- sha1()
- sha224()
- sha256()
- sha384()
- sha512()

to hash a string like the word "Hello!" in Python, we would enter:

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

To see the hashed value we would call the hexdigest() function on x:

```
>>> x.hexdigest()
'952d2c56d0485958336747bcdd98590d'
```

If we hash the phrase "Hello, World" we will get the hash shown below:

```
>>> hashlib.md5("Hello, World").hexdigest()
'82bb413746aee42f89dea2b59614f9ef'
```

But if we change a single letter and hash "Hello, world", with a lower case w, we now get a completely different result from the hash function:

```
>>> hashlib.md5("Hello, world").hexdigest()
'bc6e6f16b8a077ef5fbc8d59d0b931b9'
```

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

Hashlib and SHA256 Quiz

Use the hashlib library in Python to find the sha256 hash of the string udacity (lowercase).

Hashing Cookies

Now that we have learned how to hash data we can put our new-found knowledge to good use in order to prevent people from cheating with our cookie. 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, [hash]
```

Now, a would be cheat cannot forge a cookie without knowing what hashing algorithm we are using.

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…
```

Making Hashed Cookies Quiz

Implement the function make_secure_val, which takes a string and returns a string of the format: s,HASH

Verifying Hashed Cookies Quiz

Implement the function check_secure_val, which takes a string of the format s,HASH and returns s if hash_str(s) == HASH, otherwise it returns None.

Putting It 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
```

Note: we have replaced the comma, ‘,’ in the cookie value with the pipe,’|’. This is because of because of an issue that Google App Engine has with commas in cookies, where a comma in a cookie has a special meaning.

Next, we will change the variable used to store the value of visits held in the cookie to **visit_cookie_str**. The variable **visits** will now be the actual visit count, which will be zero by default:

```
visits = 0
visit_cookie_str = self.request.cookies.get('visits')
```

Now, we need to test if the cookie for visits exists (i.e. if the user has visited our site before). If it does, then we run the function **check_secure_val()** on visit_cookie_str. If the cookie is valid, we then update the **visits** variable accordingly:

```
if visit_cookie_str:
        cookie_val = check_secure_val(visit_cookie_str)
        if cookie_val:
                visits = int(cookie_val)
```

At this point, the value of **visits** will either be 0 (if this is the user's first visit or the cookie is invalid) or it will be the value stored in the cookie. Now we increment visits to count the current visit:

```
visits += 1
```

The next thing that we need to do is to update the cookie value. We need to use our new **make_secure_val()** function and then add the Set-Cookie header:

```
new_cookie_val = make_secure_val(str(visits))
self.response.headers.add_header('Set-Cookie', 'visits=%s' % new_cookie_val)
```

The complete MainPage() class now 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)
```

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

5|e4da3b7fbbce2345d7772b0674a318d5

Cookie Hashing

What we are doing with our application now is that we are storing a cookie that looks something like this:

visit = 1, [HASH]

On the server side, we are comparing the hashed value of 1 with the hash value stored in the cookie. If they match, we accept the value as valid, and if it doesn't then we discard it.

Does this actually solve our problem?

Well, it is certainly an improvement, but if they know we are using MD5 (which is pretty easy to guess), it is easy to forge a cookie by simply changing the cookie value, hashing that new value using MD5 and storing the new hash in the cookie.

What we need to do is to incorporate some secret knowledge that cheats do not know. Instead of hashing the value (in this case, 1) to produce the hash, we will hash a secret string plus the value:

H(SECRET + 1) = [HASH]

As long as the secret string remains secret, a would-be attacker won't be able to forge the cookie even if they know our algorithm.

Before we add this functionality to our application, let us introduce a new bit of Python.

Up until now, we have been using hashlib to run basic hashes. There is a second library, specifically for doing message authentication called HMAC:

Hash-based **M**essage **A**uthentication **C**ode

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").hexdigest()
'fd4c2d860910b3a7b65c576d247292e8'
```

HMAC Quiz

Implement the hash_str() function to use HMAC and our SECRET instead of md5.

Incorporating HMAC

OK, so let’s incorporate the new functions that we have written into our code:

```
SECRET = 'imsosecret'

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

Normally, SECRET would be held in another module that you don’t publish or share.

Password Hashing

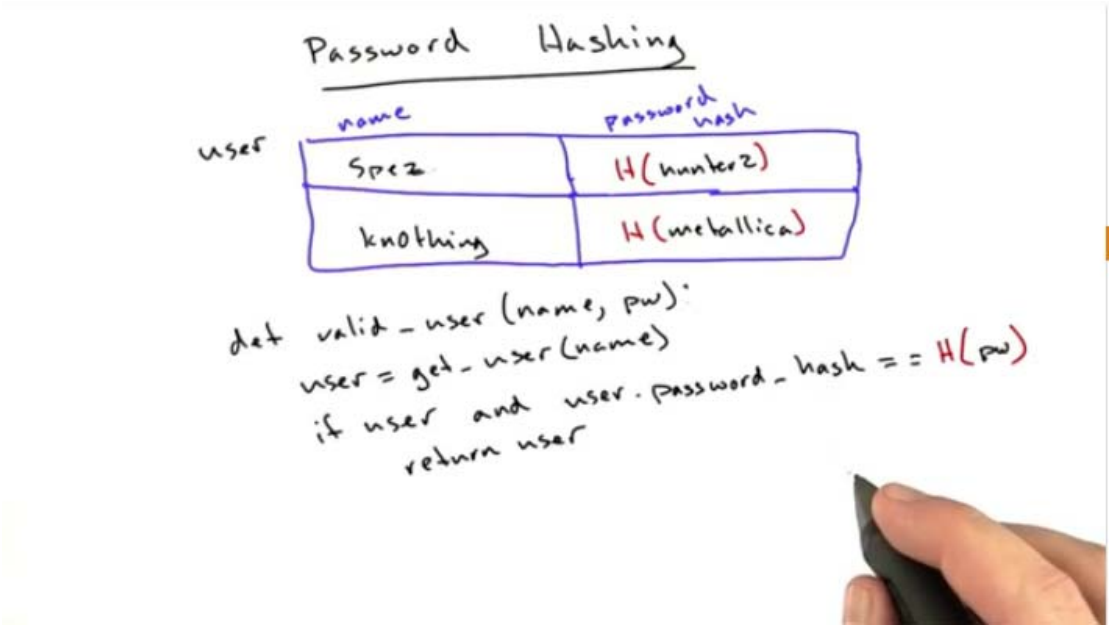
OK, we've spent a lot of time looking at how we can use hashing, and the HMAC variant of hashing, to make cookies that won't be tampered with. Now let's look at using hashing for passwords.

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:

name	password
Spez	hunter2
Kn0thing	metallica

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

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



The problem with this is that, if your database gets compromised you are in real trouble. You have 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:

name	Password hash
Spez	H(hunter2)
Kn0thing	H(metallica)

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
```

This actually takes very little work and makes your site much more secure.

So, this is a very important strategy that you should employ when you're building user registration systems (such as in this week's homework…).

Why Hash Passwords Quiz

Why do we hash passwords?

- To keep snooping Sys Admins from knowing everyone's passwords
- Because people often use the same password for many websites
- If the db is compromised, the passwords are reasonably safe.
- If you don't, you will regret it!

Storytime

Steve related the story of how, when he first started Reddit, he build a basic login system but didn't hash the passwords. He just stored them in plain text. Even though he knew at the time that this was wrong, he was in a rush and didn't realise just how big Reddit was going to become.

As the system grew, Steve knew that he should implement password hashing (password hashing has actually been around since the 1970s).

Another thing that happened as Reddit grew was that they were targeted by spammers. These spammers would create hundreds of accounts that would coordinate together to cheat the system and get their links up. One thing that almost all the spammers did was that they would create lots of accounts, but they would all have the same password. As the passwords were stored in plaintext, this made it really easy to identify the spammers.

One day, Steve's laptop was stolen. The laptop had a chunk of the Reddit database on it. He then had to go to Reddit, "with his tail between his legs" and admit that the database was compromised and that he hadn't hashed the passwords.

The lesson is clear. Never skip this step. Always hash your passwords. Even if you don't get it exactly right, it's better than not doing it at all.

Hash Your Passwords Quiz

Is it really, really embarrassing to have a database stolen with plaintext passwords?

Rainbow Tables

Now that you have your passwords hashed, are you completely safe?

Absolutely not!

The problem is that there are just a handful of good hashing algorithms that people use. If someone were to create a mapping of every word to the hash of that word for each of these algorithms that would create a problem. An attacker would just need to look up the password for any given hash! Tables of these mappings are known as rainbow tables.

These rainbow tables already exist and can be found and downloaded from the Internet.

There is a simple way to get around this problem. We have already seen from the way we secured our cookies that all you need to do is to add some secret to the password to defeat the rainbow tables. However, you cannot always use the same secret or your password database would become vulnerable to the same technique.

What we do instead is to use something called a salt.

A salt is just some random characters which are added to the password before it is hashed. This effectively defeats the quick lookup using rainbow tables from working. Our hash is now created by applying the hashing function to the password *plus the salt*:

$$h = H(pw + salt)$$

In the user table we now store the username and the hash. The salt can be stored in the clear along with the hash in the password field.

name	Password hash
Spez	H(hunter2 + salt), salt
Kn0thing	H(metallica+ salt), salt

Outside of this class, **you should think very hard before doing this yourself**. As with all cryptography, you should probably not be implementing these functions yourself. You should also think very carefully before using a third-party library since many of these also get it wrong!

Let's try implementing some naïve functions for hashing and salting a password to see how the flow works, and then we will discuss some of the things that you should look for when you are evaluating someone else's approach.

Making Salts Quiz

Implement the function make_salt() that returns a string of 5 random letters using python's random module.

Note: The string package might be useful here.

Hashing Salts Quiz

Implement the function make_pw_hash(name, pw) that returns a hashed password of the format:

HASH(name + pw + salt),salt

Use sha256

Validating Salts Quiz

We previously created the function make_pw_hash() that returned a salted version of the password.

Now you need to implement the function valid_pw() that returns True if a user's password matches its hash.

TIP: You will need to modify make_pw_hash() to make this work.

Bcrypt

What we have seen so far for password hashing is good to understand, and most modern computers has passwords in just the way that we have shown you.

The problem with most hashing functions is that they are designed to be fast. This is generally a good thing, and is really great for verifying cookies and so forth. However, for cases like passwords where it is much more likely that you will be subject to a brute-force attack, it would be really useful if we had a function that is both really good (like SHA256) but is also rather slow, so that as computers get faster and faster the hash function stays the same speed.

Fortunately, there is just such a function. It is called bcrypt.

bcrypt basically takes an extra parameter which is “how long do you want this to take”, giving us a function that will stay slow forever because we can make it slow.

A downside is that it is not built into Python and needs to be installed separately.

In future, outside this course, instead of using SHA256, you should be using bcrypt.

HTTPs

One last thing about passwords. Although we now know how to store our passwords securely on the server, we still have the problem of sending the password from the browser to your server.

When we looked at forms, we saw that when a password is entered into a form, and the form is submitted, the password is sent in clear text over the Internet. To protect the password while it is in transit, and prevent attackers from sniffing the password off the wire, we should use HTTPs ("HTTP secure").

HTTPs is just like HTTP, except it is encrypted over SSL. An attacker then shouldn't be able to read anything coming over the wire.

The Best Quiz

When given the choice, which is the best hashing algorithm to hash passwords?

- MD5
- SHA256
- HTTPs
- bcrypt

Answers

Good Uses For Cookies Quiz

- **Storing login information.**
- **Storing small amounts of data to avoid hitting a db.**
- Storing user preference information.
- **Tracking you for ads.**

Sending Cookies Quiz

Cookie

Setting Cookies Quiz

Set-Cookie

Valid Receivers Quiz

- udacity.com
- **ide.udacity.com**
- **other.ide.udacity.com**
- other.udacity.com

Valid Setters Quiz

- udacity.com
- **ide.udacity.com**
- **other.ide.udacity.com**
- other.udacity.com

Cookie Deletion Quiz

- Jan 1, 2025.
- Never.
- **When you close your browser.**
- In 1 day.

Cheating Quiz

What can we do to get 10,000 visits?

- **Reload the page 10,000 times.**
- Send the link to 10,000 friends
- **Edit the cookie in our browser.**

Hashlib and SHA256 Quiz

016d473857f1029884ec80ede8ae486f33d2fdad9411d63cd2aab11097ee997c

Making Hashed Cookies Quiz

```
import hashlib

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

def make_secure_val(s):
    return "%s,%s" %(s, hash_str(s))
```

Verifying Hashed Cookies Quiz

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

HMAC Quiz

```
import hmac

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

Why Hash Passwords Quiz

- **To keep snooping Sys Admins from knowing everyone's passwords**
- **Because people often use the same password for many websites**
- **If the db is compromised, the passwords are reasonably safe.**
- **If you don't, you will regret it!**

Hash Your Passwords Quiz

Yes!

Making Salts Quiz

```
import random
import string

def make_salt():
    return "".join(random.choice (string.letters) for x in range(5))
```

Hashing Salts Quiz

```
import random
import string
import hashlib

def make_salt():
    return ''.join(random.choice(string.letters) for x in xrange(5))

def make_pw_hash(name, pw):
    salt = make_salt()
    h = hashlib.sha256(name + pw + salt).hexdigest()
    return '%s,%s' % (h, salt)
```

Validating Salts Quiz

```
import random
import string
import hashlib

def make_salt():
    return ''.join(random.choice(string.letters) for x in xrange(5))

def make_pw_hash(name, pw, salt = None):
    if not salt:
        salt = make_salt()
    h = hashlib.sha256(name + pw + salt).hexdigest()
    return '%s,%s' % (h, salt)

def valid_pw(name, pw, h):
    salt = h.split(',')[1]
    return h == make_pw_hash(name, pw, salt)
```

The Best Quiz

- MD5
- SHA256
- HTTPs
- **bcrypt**