

```

1 import random
2 import string
3 import hashlib
4
5 def make_salt():
6     return ''.join(random.choice(string.letters) for x in xrange(5))
7
8 # Implement the function valid_pw() that returns true if a user's password
9 # matches its hash. You may need to modify make_pw_hash.
10
11 def make_pw_hash(name, pw, salt = None):
12     if not salt:
13         salt = make_salt()
14     h = hashlib.sha256(name + pw + salt).hexdigest()
15     return '%s,%s' % (h, salt)
16
17 def valid_pw(name, pw, h):
18     salt = h.split(',')[1]
19     return h == make_pw_hash(name, pw, salt)
20
21 h = make_pw_hash('spez', 'hunter2')
22 print valid_pw('spez', 'hunter2', h)
23

```

RUN

True

Rainbow Tables

password hashing

passwords are hashed. are you completely safe?

rainbow table

NO

$a \leftarrow H(a, \text{salt})$

password $\leftarrow H(\text{password})$

Think very hard
about using
3rd.
party libraries

Salt

user	name	hash
	spez	H, salt

$h = H(\text{pw} + \text{salt}), \text{salt}$
random characters

Quiz

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

☐ md5

☐ sha256

☐ HTTPS

☒ bcrypt

Quiz

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

☒ yes ☐ no

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

Quiz

what can we do to get to 10,000 visits?

- ☒ reload the page 10,000 times
- ☐ send the link to 10,000 friends
- ☒ edit the cookie in our browser

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

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

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

Quiz

which header does a server use to set a cookie?

Set-Cookie

Quiz

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

Cookie

"shouldn't"

Quiz

what are ~~good~~ uses of cookies? *appropriate*

- ☒ Storing login information
- ☒ Storing small amounts of data to avoid hitting a db
- ☒ Storing user preference info
- ☒ tracking you for ads

want data to survive →

Password Hashing

Password Hashing

user	name	password hash
	Specz	H(hunter2)
	kn0thing	H(meballica)

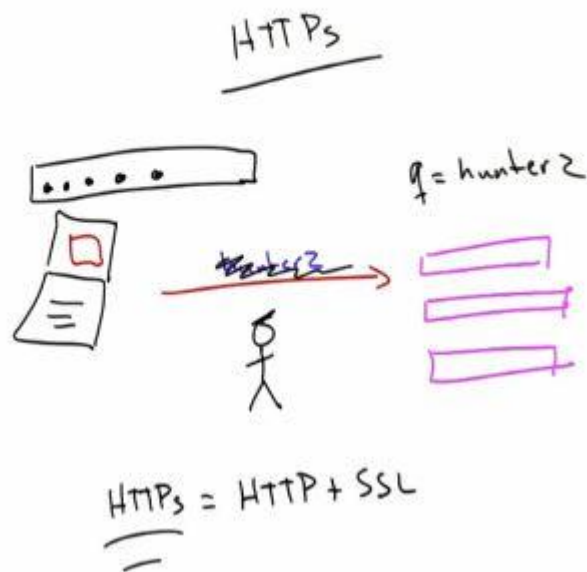
```
def valid_user(name, pw):  
    user = get_user(name)  
    if user and user.password_hash == H(pw):  
        return user
```

```
1 import random  
2 import string  
3  
4 # implement the function make_salt() that returns a string of 5 random  
5 # characters use python's random module  
6  
7 def make_salt():  
8     return ''.join(random.choice(string.letters) for x in xrange(5))  
9  
10 print make_salt()
```

Run

Ctrl+F

HTTPS



Incorporating HMAC

cookie hashing

visits = 1

visits = 1 | md5(1)

visits = 1 | HMAC(secret, 1)

```
3 # Implement the hash_str function to use HMAC and our SECRET instead of md5
4 SECRET = 'imsossecret'
5 def hash_str(s):
6     return hmac.new(SECRET, s).hexdigest()
7
8 def make_secure_val(s):
9     return "%s:%s" % (s, hash_str(s))
10
11 def check_secure_val(h):
12     val = h.split(':')[0]
13     if h == make_secure_val(val):
14         return val
15
16 print make_secure_val("test!!")
17
```

UN

test!96fc783b50f770c1a648f068c0fd5fee

```

1 import random
2 import string
3 import hashlib
4
5 def make_salt():
6     return ''.join(random.choice(string.letters) for x in xrange(1))
7
8 # implement the function make_pw_hash(name, pw) that returns a hashed password
9 # of the format:
10 # HASH(name + pw + salt),salt
11 # use sha256
12
13 def make_pw_hash(name, pw):
14     salt = make_salt()
15     h = hashlib.sha256(name + pw + salt).hexdigest()
16     return '%s,%s' % (h, salt)
17
18 print make_pw_hash('spez', 'hunter2')
19
20

```

RUN

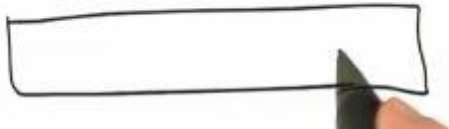
c7bc319229a56c0a1a694ece2740e35fcf82f808ca2caaeecc6f2e6938e81b0c,L3WY0

In [2]: hashlib.sha256("udacity").hexdigest()

Out[2]: '016d473857f1029884ec80ede8ae486f33d2fdad9411d63cd2aab11097ee997c'

Quiz

use the **hashlib** library in python
to find the **sha256** hash of
the string **udacity**
(lowercase)



Hashing

what is a hash?

$$H(x) \rightarrow y$$

=
x is data

y is fixed length bit string

32 - 256 bits

- difficult to generate a specific y
- infeasible to find x for a given y (one way)
- can't modify x without modifying y

Hashing Cookies

hashing cookies

set-cookie: visits=5, [hash] \rightarrow to browser
abc123

\rightarrow from browser

5, abc123
| |
val hash
if $H(val) == hash$
 valid!
else:
 invalid

Hash Algorithms

don't write your own!

(unless you're making a hashtable)

collision -
two thing hash
to the same value
Hard to find

fast
↓
slow

crc32 - checksums, fast
md5 - fast, ~~secure~~
 $H(x) \rightarrow y$
sha1 - secure-ish
sha256 - pretty good

Cookie Headers

HTTP Response

~20k

Set-Cookie: user_id=12345

name value (url)
Set-Cookie: last-seen = Dec 25 1985

HTTP Request

Cookie: user_id=12345; last-seen=Dec 2

don't

Cookie Hashing

Cookie Hashing

visit = 1, [HASH]

$H(1) =$

$H(\text{SECRET} + 1) = [\text{HASH}]$

hashlib

HMAC

Hash-based Message
Authentication Code

$\text{hmac}(\text{secret}, \text{key}, H) \rightarrow [\text{HASH}]$

Cookie Expiration

Set-Cookie: user=123; Expires = Tue, 1 Jan 2025 00:00:0 GMT

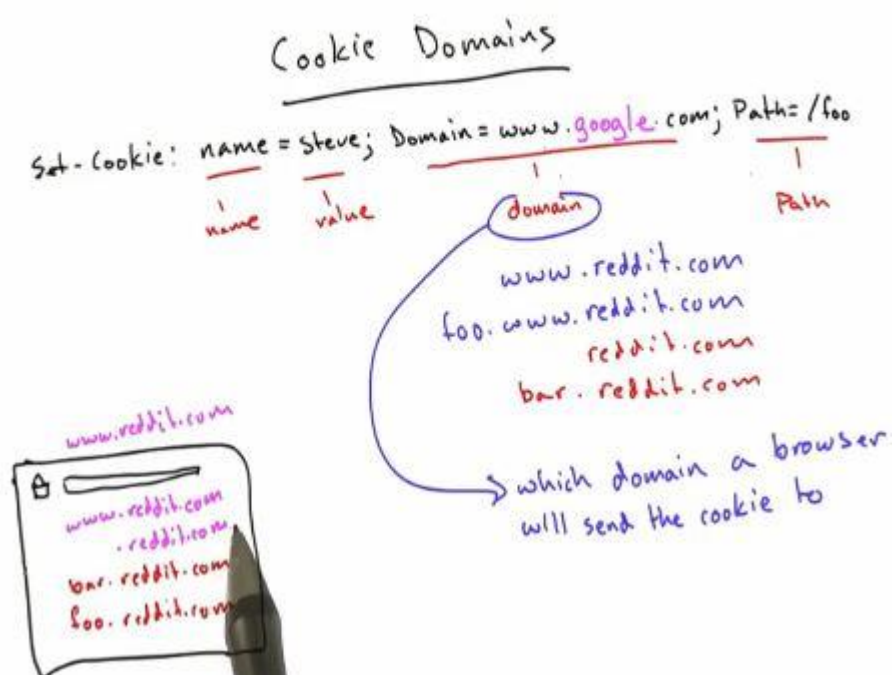
'session' cookie = no Expires

user

pass

☒ remember me

Cookie domains



Bcrypt

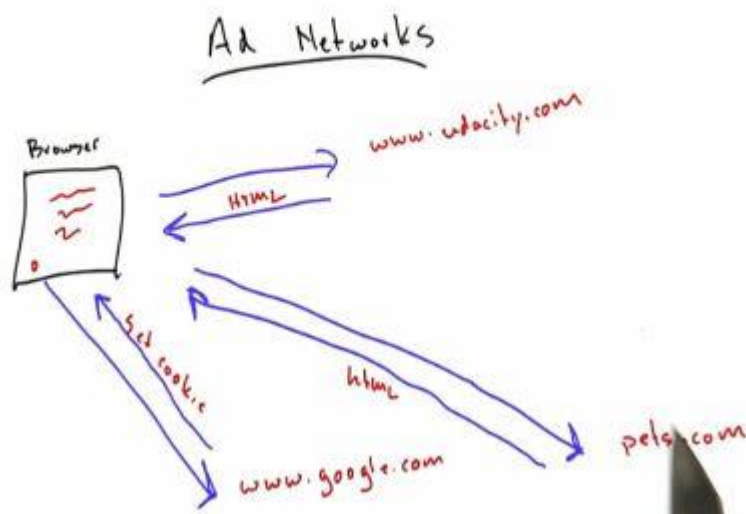
Password Hashing

problem with most hashing functions

they're designed to be fast

bcrypt

Ad Networks



Cookies

Cookies

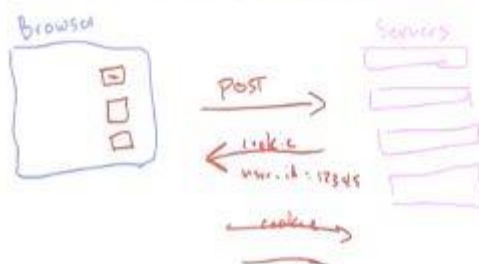
a ^($< 4kb$) small piece of data stored
in the browser for a website

name = value

user.id = 12345

Browser Limit
- 20 cookies /
website

- $< 4kb$



- only for website

```
5  
6 # Implement the function make_secure_val, which takes a string and returns a  
7 # string of the format:  
8 # s,HASH  
9  
10 def make_secure_val(s):  
11     return "%s,%s" % (s, hash_str(s))  
12  
13 print make_secure_val("cool")
```

RUN

cool,b1f4f9a523e36fd969f4573e25af4540