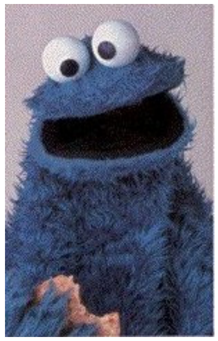
Class 31: Cookie Monsters and Semi-Secure Websites



CS150: Computer Science
University of Virginia
Computer Science

David Evans
http://www.cs.virginia.edu/evans

Why Care about Security?

Security

- Confidentiality – keeping secrets
  - Protect user's data
- Integrity – making data reliable
  - Preventing tampering
  - Only authorized people can insert/modify data
- Availability
  - Provide service (even when attacked)
  - Can't do much about this without resources

How do you authenticate?

- Something you know
  - Password
- Something you have
  - Physical key (email account?, transparency?)
- Something you are
  - Biometrics (voiceprint, fingerprint, etc.)

Serious authentication requires at least 2 kinds

Early Password Schemes

Login does direct password lookup and comparison.

UserID	Password
alyssa	fido
ben	schemer
dave	Lx.Ly.x

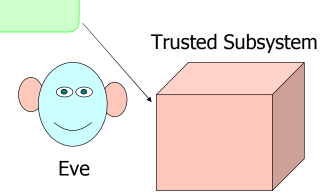
```
Login: alyssa
Password: spot
Failed login.  Guess again.
```

Login Process

Terminal

```
Login: alyssa
Password: fido
```

login sends
<"alyssa", "fido">



Password Problems

- Need to store the passwords
  - Dangerous to rely on database being secure

Solve this today

- Need to transmit password from user to host
  - Dangerous to rely on Internet being confidential

Solve this Wednesday

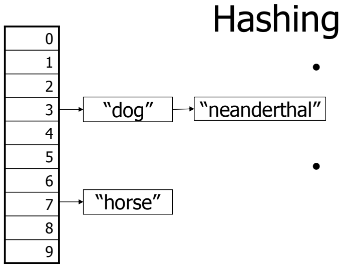
First Try: Encrypt Passwords

- Instead of storing password, store password encrypted with secret *K*.
- When user logs in, encrypt entered password and compare to stored encrypted password.

UserID	Password
alyssa	encrypt_K ("fido")
ben	encrypt_K ("schemer")
dave	encrypt_K ("Lx.Ly.x")

Problem if *K* isn't so secret: $\text{decrypt}_K(\text{encrypt}_K(P)) = P$

Hashing



$H(\text{char } s[]) = (s[0] - \text{'a'}) \bmod 10$

- Many-to-one: maps a large number of values to a small number of hash values
- Even distribution: for typical data sets, probability of $(H(x) = n) = 1/N$ where N is the number of hash values and $n = 0..N-1$.
- Efficient: $H(x)$ is easy to compute.

Cryptographic Hash Functions

One-way

Given h, it is hard to find x such that $H(x) = h$.

Collision resistance

Given x, it is hard to find $y \neq x$ such that $H(y) = H(x)$.

Example One-Way Function

Input: two 100 digit numbers, x and y

Output: the middle 100 digits of $x * y$

Given x and y, it is easy to calculate

$f(x, y)$ = select middle 100 digits $(x * y)$

Given $f(x, y)$ hard to find x and y.

A Better Hash Function?

- $H(x) = \text{encrypt}_x(0)$
- Weak collision resistance?
  - Given x, it should be hard to find $y \neq x$ such that $H(y) = H(x)$.
  - Yes – encryption is one-to-one. (There is no such y.)
- A good hash function?
  - No, its output is as big as the message!

Actual Hashing Algorithms

- Based on cipher block chaining
  - Start by encrypting 0 with the first block
  - Use the next block to encrypt the previous block
- SHA [NIST95] – 512 bit blocks, 160-bit hash
- MD5 [Rivest92] – 512 bit blocks, produces 128-bit hash
  - This is what we will use: built in to PHP

Hashed Passwords

UserID	Password
alyssa	*md5* ("fido")
ben	*md5* ("schemer")
dave	*md5* ("Lx.Ly.x")

Dictionary Attacks

- Try a list of common passwords
  - All 1-4 letter words
  - List of common (dog) names
  - Words from dictionary
  - Phone numbers, license plates
  - All of the above in reverse
- Simple dictionary attacks retrieve most user-selected passwords
- Precompute $H(x)$ for all dictionary entries

(at least) 86% of users are dumb and dumber

Single ASCII character	0.5%
Two characters	2%
Three characters	14%
Four alphabetic letters	14%
Five same-case letters	21%
Six lowercase letters	18%
Words in dictionaries or names	15%
Other (possibly good passwords)	**14%**

(Morris/Thompson 79)

Salt of the Earth

(This is the standard UNIX password scheme.)

Salt: 12 random bits

UserID	Salt	Password
alyssa	1125	DES+^{25} (0, "Lx.Ly.x", 1125)
ben	2437	DES+^{25} (0, "schemer", 2437)
dave	932	DES+^{25} (0, "Lx.Ly.x", 932)

DES+ (m, key, salt) is an encryption algorithm that encrypts in a way that depends on the salt.

How much harder is the off-line dictionary attack?

Python Code

```
// We use the username as a "salt" (since they must be unique)
encryptedpass = crypt.crypt (password, user)
```

user	password
alyssa	9928ef0d7a0e4759ffefbadb8bc84228
evans	bafd72c60f450ed665a6eadc92b3647f

Authenticating Users

- User proves they are a worthwhile person by having a legitimate email address
  - Not everyone who has an email address is worthwhile
  - Its not too hard to snoop (or intercept) someone's email
- But, provides much better authenticating than just the honor system

Registering for Account

- User enters email address
- Sent an email with a temporary password

```
rnd = str(random.randint (0, 9999999))
        + str(random.randint (0, 9999999))
encrnd = crypt.crypt (rnd, str(random.randint (0, 99999)))
users.userTable.createUser (user, email, firstnames, lastname, encrnd)
sendMail.send (email, "hoorides-bot@cs.virginia.edu", "Reset Password", \
              "Your " + constants.SiteName + \
              " account has been created.  To login use:\n   user: " + \
              user + "\n   password: " + encrnd + "\n")
...
```

From register-process.cgi

Users and Passwords

```
def createUser(self, user, email, firstnames, lastname, password) :
    c = self.db.cursor ()
    encpwd = crypt.crypt (password, user)
    query = "INSERT INTO users (user, email, firstnames, lastname, password) " \
          + "VALUES ('" + user + "', '" + email + "', '" \
          + firstnames + "', '" + lastname + "', '" + encpwd"')"
    c.execute (query)
    self.db.commit ()
def checkPassword(self, user, password):
    c = self.db.cursor ()
    query = "SELECT password FROM users WHERE user='" + user + "'"
    c.execute (query)
    pwd = c.fetchone ()[0]
    if not pwd:
      return False
    else:
      encpwd = crypt.crypt (password, user)
      return encpwd == pwd
```

From users.py (cookie processing and exception code removed)

Cookies

- HTTP is stateless: every request is independent
- Don't want user to keep having to enter password every time
- A cookie is data that is stored on the browser's machine, and sent to the web server when a matching page is visited

Using Cookies

- Look at the PS7 provided code (cookies.py)
- Cookie must be sent before any HTML is sent (util.printHeader does this)
- Be careful how you use cookies – anyone can generate any data they want in a cookie
  - Make sure they can't be tampered with: use md5 hash with secret to authenticate
  - Don't reuse cookies - easy to intercept them (or steal them from disks): use a counter than changes every time a cookie is used

Problems Left

- The database password is visible in plaintext in the Pythin code
  - No way around this (with UVa mysql server)
  - Anyone who can read UVa filesystem can access your database
- The password is transmitted unencrypted over the Internet (next class)
- Proving you can read an email account is not good enough to authenticate for important applications

Charge

- Authentication
  - At best, all this does is check someone can read mail sent to a .virginia.edu email address
- Find PS8 partners now!