CSCI 1320
Creating Modern Web Applications
Lecture 28: Security II
Project Design Presentations

• 10-15 Minute presentation of your overall design
  • Concentrating on back end and implementation
    • Describe and justify implementation decisions
    • Show the overall architecture
  • What are the primary tasks (stories)
  • How are the tasks handled
    • What is done in the front end, server, database, ...
  • Strategies for handling
    • Security, scalability

• Alternative
  • A 3-10 page document describing the above
Review

• Security is a major concern
• Lots of obvious problems
• Lots of non-obvious problems
  • SQL injection attacks
  • Code insertion attacks
  • Cross-Site Scripting attacks (XSS)
• And others ...
Cross-Site Request Forgery

• Suppose user logs into banking application
  • Cookies used to validate user/session
  • Suppose there is a URL that transfers money
    • transfer?from=checking&to=43434&amt=1000000.00

• Agent puts an ad on another page
  • Clicking on the add, generates that URL

• What happens if user clicks on the ad
  • While logged into banking application
Cross-Site Request Forgery

- Use a random value for each request
  - Set on previous request, kept in session
  - Passed back as part of any request
  - Validated by server before the action
  - Effective URL will be different each time
    - Can’t be spoofed by another client (easily)

- Can be passed to client in various ways
  - Sent as part of html or XMLHttpRequest
  - Included in a hidden form field
  - Can also be put into header by JavaScript
  - Packages exist to support this

- What can go wrong?
Server Attacks

• Inputs from web page attack the server directly
• Outside login to server
  • Weak passwords, user ids
  • Access from other machines
• Server becomes compromised
Buffer Overflow Attack

• Code:
  
  ```c
  void function(char* text) {
    char buf[1000];
    strcpy(buf, text);
    // do some editing of buf
    // save result
  }
  ```

• Stack (high to low)
  
  8888: <ptr to text>
  8884: <return address>
  8880: <old stack ptr>
  7880: buf[0 .. 999]
Preventing Buffer Overflow

• Use “safe” languages (Java, C#, JavaScript, ...)
• Check sizes of data before putting in array
  • Reads, copies, inputs
• Randomize code locations between runs
• Don’t let data pages be executable
Server Attacks

- Can buffer overflow happen in Java? JavaScript? Php?
- Even safe languages can have problems
  - Out of memory, out of file space
  - Run arbitrary code (malicious servlets)
  - Tie up server for long time
  - Java/PHP/... security problems
    - File access, exec, eval, ...
What Else Can Go Wrong

- People
- Denial of service
- Timing attacks...
Question

Which is not true about password management in an application?

A. Passwords should never be sent in clear text.
B. Passwords can be checked by the client so that no passwords need to be stored on the server.
C. Passwords should be saved in the server database by using a hash function such as SHA256 over the password concatenated with a user-specific seed.
D. With proper encodings it is not easy for an attacker to determine if two users share the same password.
E. Login with passwords is difficult to get right so you should use a tested, third-party solution to handle login.
Logging In

• Common operation
  • Should be easy
  • What are the problems?

• What are the operations to be concerned with?
  • Registration (initial name & password)
  • Log in (provide name & password to validate)
  • Access while logged in
Homework

• What requirements did you come up with for registration and login?
Logging In: Threat Model

- Spoofing URLs
- Sending lots of requests
- Wi-Fi snooping
- Internet snooping
- Reading logs
- Man-in-the-middle attacks
- Phishing attacks
- Brute force
- Loss of database (SQL injection attack; stolen laptop)
- Guessing passwords
- Finding duplicate passwords
Main Concerns

• Not sensitive to snooping
  • Seeing what is sent one time, won’t help the user log in
• Loss of database doesn’t lose information
  • Difficult to get passwords from database
  • Difficult to even know if 2 users have the same password
Sending Passwords

• How might you hide a password from the Internet?
  • /login?uid=spr&pwd=password
• Send Hash(password) to server
  • MD5(), SH1(), SH256()
  • Does this work?
• Server sends Hash(password) to client
  • Does this work?
Sending Passwords

• More complex protocols
  • Server sends random string (i.e. session id) to client
  • Client sends Hash(string + password) to server
  • Client sends Hash(string + hash(password)) to server
  • Client sends Hash(string + userid + hash(password)) to server

• Do these work?
  • Can they be checked by the server?
Saving Passwords

How to save passwords on your website?
Why is this a problem? Does it matter?
What if your site is compromised
Users use same password for multiple sites
This is something many applications do wrong

**Never store user passwords in plaintext**

Store the hash of the passwords
Cryptographically secure hash function
MD5, SH1, SH256, ...

Is this sufficient?
Secure Password Hashing

What happens if two users have the same password?
What will the stored password hashes look like?
Solution: “salt” the hashes.
You generate a random string which is stored alongside the hashed password for each user.
Can just be the user id
Compute and store hash($salt . $password)
Is this sufficient?
Secure Password Hashing

Brute force attack on stored passwords
Compute SHA256($salt . $password) for possible passwords
This is relatively fast
Solution: “stretch” the hashes.
Instead of calling SHA256 once, call it thousands of times.
Makes it more expensive to mount a brute-force attack

Do you really want to write all that code?
Crypto code is notoriously tricky, the bugs are subtle, and the consequences of doing it wrong are dire.
Solution: Use Passport, bcrypt, or similar for Node.js. Solutions exists for other frameworks as well.
These solve of the problems we’ve mentioned, can become more computationally difficult as computing power increases, and are available in most web libraries
Secure Password Hashing

Your threat model should look like this:

![Comic strip](image)
Managing Login

• **Doing it yourself**
  • Web page sends
    • Hash(sessionid + Hash(userid + Hash(password)))
  • Server stores
    • Hash(userid + Hash(password))
  • Registration sends
    • userid, Hash(userid + Hash(password))
    • Or just userid, Hash(password)

• **Better yet – let someone else do it**
  • Passport, bcrypt, oauth, ... packages exist to support this
  • Support using other credentials as well (Facebook, Google, ...)

Secure Communication

- Want to make it so what is sent is unreadable
  - To anyone who can see all Internet traffic
  - How can this be done?

Encryption

- Recall hash functions $H(X) = \langle$random number$\rangle$
  - These are not encryption functions
- What is an encryption function
  - $F(X) = Y$ easy to compute
  - $F^{-1}(Y) = X$ difficult to compute (without additional knowledge)
- Examples of encryption functions
Encrypted Connections

• Encrypt all communication
  • Simpler solution than trying to encrypt password
  • Between the browser and the server
  • Handles some of the issues raised with passwords
  • Handles other problems as well
    • Credit card numbers and other private information

• Encrypted communications are relatively standard
  • Clients needs to agree on how to encode/decode
    • Agreeing on an algorithm for encoding/decoding
    • Agreeing on a key for that algorithm
Standard Encryption

• Both parties agree on a key K
• $F(K)$ and $F^{-1}(K)$ are easy to compute
  • If you know K
  • But are difficult if you don’t know K
  • May even be done in hardware
• Standard encryption functions available
  • DES is probably the most common
  • encryption/decryption libraries available
    • Most backends; JavaScript front end
• Problem: agreeing on K
Public Key Cryptosystems

• Public Key Cryptosystems
  • Originator has two pieces of information X and Y
  • $F(\text{string},X) = \text{encoded string}$
  • $F^{-1}(\text{string},X)$ is difficult to compute
  • $F^{-1}(\text{string},X,Y)$ is easy to compute

• Examples
  • Y,Z are 200 digit primes, X is Y*Z
  • Create a string using X such that string can only be decoded knowing the factors of X
  • Other examples are possible

• This is often used as part of a secure protocol
  • Agreeing on a key for a more secure encoding
Browser-Server Communication

• Can use encrypted communication in a web app
  • HTTPS represents an encrypted (secure) connection

• HTTPS is just like HTTP
  • Except that all data passed back and forth is encrypted
  • Browser and server agree on a key
  • Encryption is then done based on this key
  • This is handled by the Secure Sockets Layer (SSL)
  • Input on a different port

• SSL is not specific to web applications
HTTPS Connections

• Browser makes a connection to the server
• SSL handshake protocol
  • Browser sends and requests a certificate
    • Certificates are effectively keys that can be verified as authentic
    • This is one way public key systems are used
  • Server replies with a certificate of its own
• SSL change cipher protocol
  • Browser and server use their certificates to agree on a key
    • Again using a variant of public key systems
• Communication is done securely using that key
  • Key is only used for this particular session
HTTPS Usage

• If you are sending confidential information
  • Even just passwords
  • Especially credit card numbers, etc.
  • You should use HTTPS
  • Alternatively, use a separate service (e.g. stripe, paypal, ...)

• OPENSSL and other implementations exist
  • Typically built into server and browser
  • Different port used for secure communication
  • Integrated into Apache using Mod_SSL for example

• Problem: Obtaining a certificate
Next Time

• Security LAB
• Prelab
  • Form teams
  • Get and install a copy of the insecure server, study it
  • Make a list of vulnerabilities
• Teams decide whether to attack/defend (or both)
  • Attackers: develop a set of attacks to try on Monday
  • Defenders: fix the most obvious holes in the server for Monday
• Monday
  • We will post the URLs for defended servers
  • Attackers will mark the ones that are successfully attacked