CS1320
Creating Modern Web and Mobile Applications

Lecture 30

Security III

Review

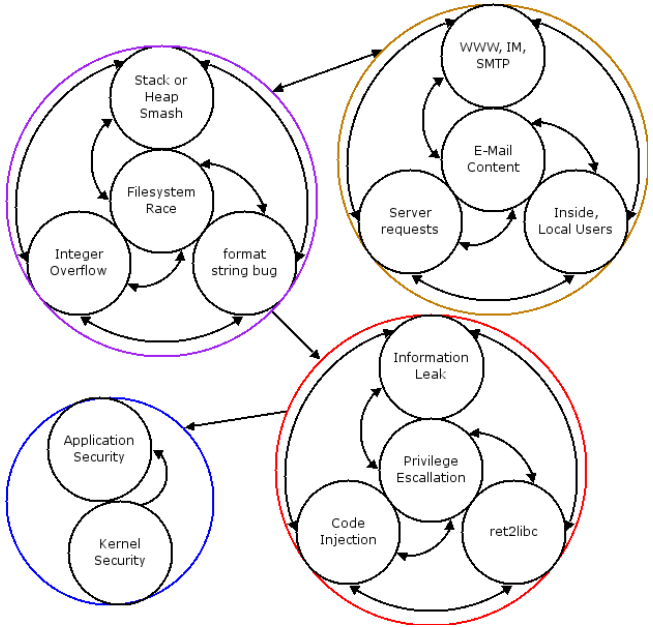
- Security is a major concern
- Lots of obvious problems
- Lots of non-obvious problems
  - SQL injection attacks are the most prevalent
  - Cross-site scripting (XSS)
  - Cross-site request forgery (XSRF)
  - File attacks …
- And there are others …
  - These get more obscure, complex, difficult to address

Logging In

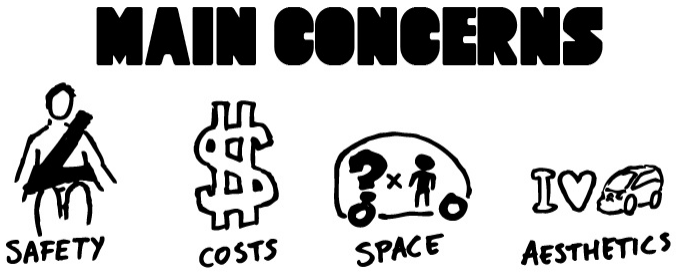
- Common operation
  - Should be easy
  - What are the problems?
- What are the operations
  - Registration (initial name & password)
  - Log in (provide name & password to validate)
  - Access while logged in

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 attempt to login
- Loss of database (SQL injection attack; stolen laptop)
- Guessing passwords
- Finding duplicate passwords

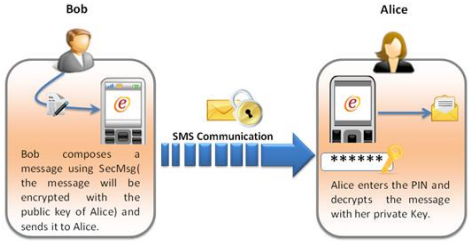


Considerations



- Passwords need to be protected
  - Difficult to spoof
  - Not susceptible to man-in-the-middle attacks
  - Getting database doesn't reveal passwords
  - Getting database doesn't reveal similar passwords
  - Guessing passwords is expensive
- Not sensitive to snooping
  - Seeing what is sent one time, won't help the attacker 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

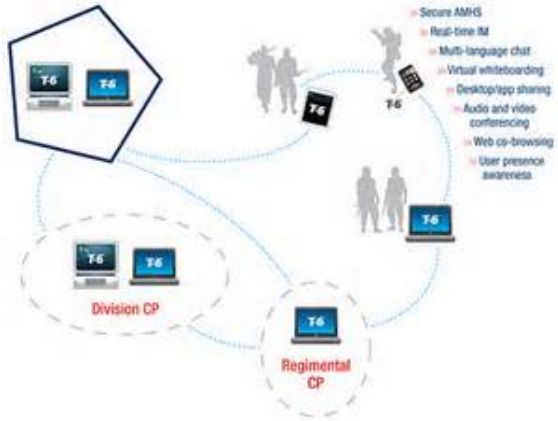
Secure Communication



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

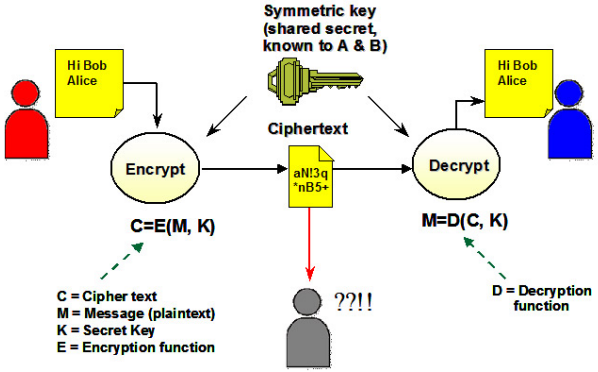
- Encrypt all communication
  - Simpler solution than trying to encrypt password
  - Between the browser and the server
  - Handles some of the issues raised with passwords
    - Still need to handle loss of database, guessing, …
  - Handles other problems as well
    - Credit card numbers and other private information



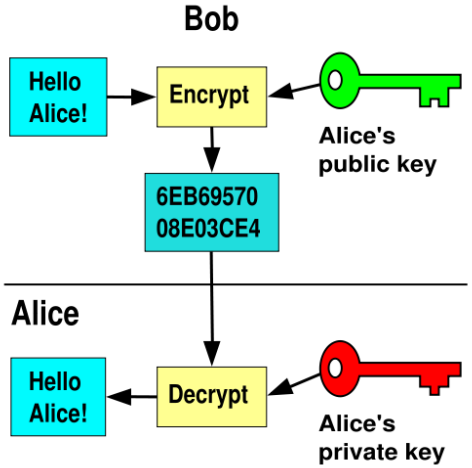
- Encrypted communications are relatively standard
  - Clients need to agree on how to encode/decode
    - Agreeing on an algorithm for encoding/decoding
    - Agreeing on a key for that algorithm

Standard Encryption

- Both parties agree on a key K
- $F_K(m)$ and $F_K^{-1}(m')$ 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 back-ends; JavaScript front end
- Problem: agreeing on K



Public Key Cryptosystems

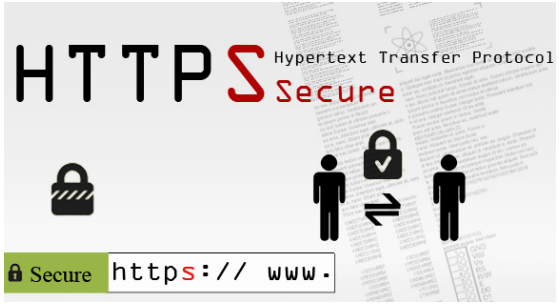


- Public Key Cryptosystems
  - Originator has two pieces of information X and Y
  - F(string,X) = encoded string
  - F^{-1}(string,X) is difficult to compute
  - F^{-1}(string,X,Y) is easy to compute
- Examples
  - Y,Z are 200 digit primes, X is Y*Z
  - Create a string using X such that string can only be decoded knowing the factors of X
  - Other examples are possible
- This is often used to agree on a key K for encryption
  - Too complex for continuous encryption

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
- Many applications are HTTPS only today
  - Browsers are starting to enforce this
- SSL is not specific to web applications
  - Can be used by mobile apps, etc.

HTTPS Connections

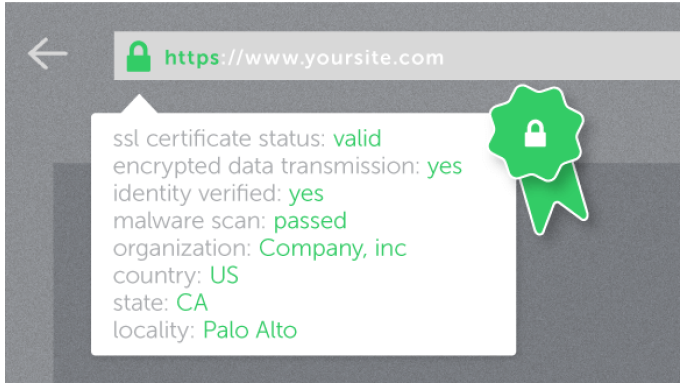


- Browser makes a connection to the server
- SSL handshake protocol
  - Browser sends and requests a certificate
    - Certificates are effectively public keys
    - Associated and verified as authentic to a particular URL
    - This is one way public key systems are used
  - Server replies with a certificate 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**
  - **Better yet, 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
  - Libraries available for Java, Swift, JavaScript, …

HTTPS Certificates



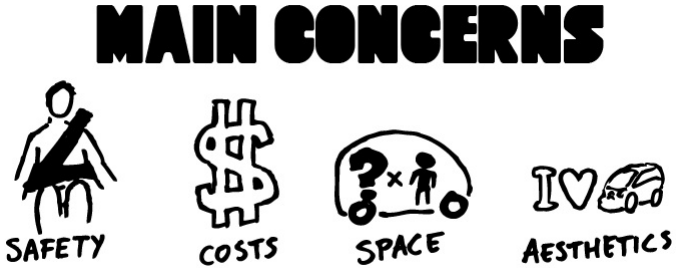
- A certificate
  - Contains the public key
  - Validates that you are the owner of the given address
- Certificates can now be obtained for free
  - AWS provides them for you (as do other cloud services)
    - AWS certificate manager
  - letsencrypt.org is another source

HTTPS Isn't Perfect

- Bugs in SSL implementations
- Subject to sophisticated man-in-the-middle attacks
  - If not done correctly
  - And many (up to 95%) of the servers aren't
- Subject to spoofing attacks
- So we still need to design login protocols
  - That don't rely solely on https
  - That aren't broken if the database is exposed
  - That don't leak information about passwords

Main Concerns

- Not sensitive to snooping
  - Seeing what is sent one time, won't help the attacker 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

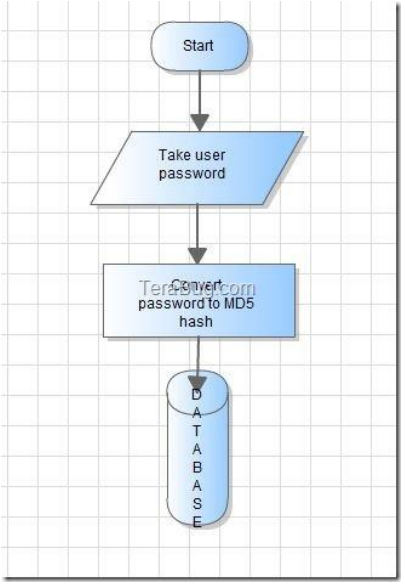


Sending Passwords

- How might you hide a password from the Internet?
  - /login?uid=spr&pwd=password
  - Or the POST equivalent
- Server sends Hash(password) to client to check
  - Does this work?
- Send Hash(password) to server
  - Secure hash functions
    - **F(X) = Y** easy to compute
    - **$F^{-1}(Y)$ can't be computed easily**
    - SH1(), SH256(), …
  - Does this work?

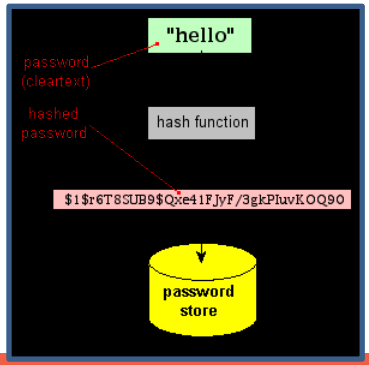
Saving Passwords

- How to save passwords on your website?
  - Why is this a problem? Does it matter?
  - What if your site or database 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
  - SH1, SH256, …
- Is this sufficient?

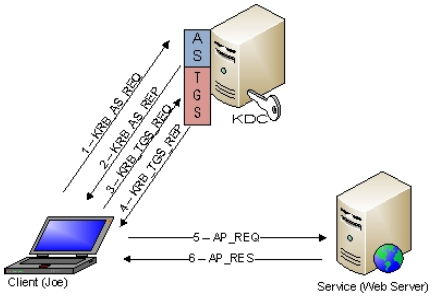


Secure Password Hashing

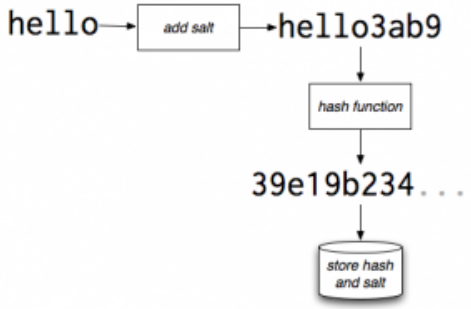
- What happens if two users have the same password?
  - What will the stored password hashes look like?
- What happens if users use ‘common’ passwords?
- Solution: “salt” the hashes.
  - You generate a random string which is stored alongside the hashed password for each user.
    - Can just be the user id
  - Compute and store hash(<salt> + <password>)
- Is this sufficient?



Sending Passwords



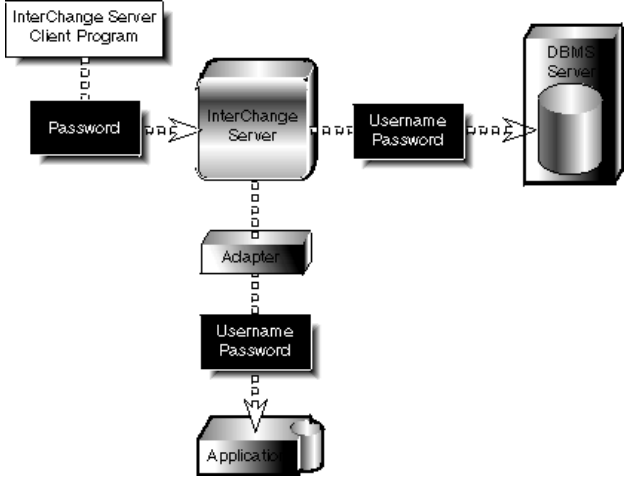
- More complex protocols
  - Server sends salt (random string, session id) to client
  - Client sends Hash(string + password) to server
  - Client sends Hash(string + Hash(password)) to server
  - Client sends Hash(string + userid + Hash(password)) to server
  - Client sends Hash(string + Hash(userid + password)) to server
- Do these work?
  - Can they be checked by the server?
  - What does server need to store to check these?



Secure Password Hashing

- Brute force attack on stored passwords
  - Compute SHA256($salt . $password) for 1M common 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.

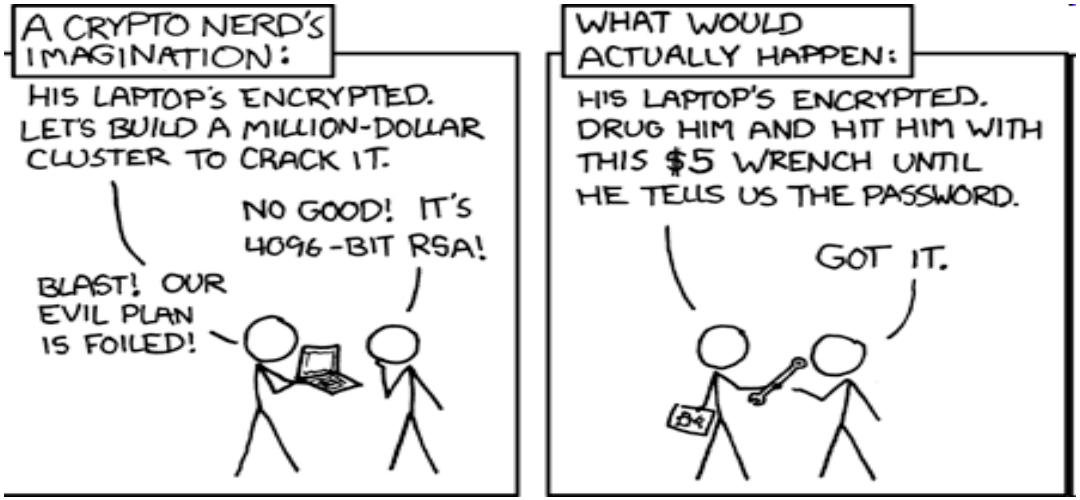
Managing Login



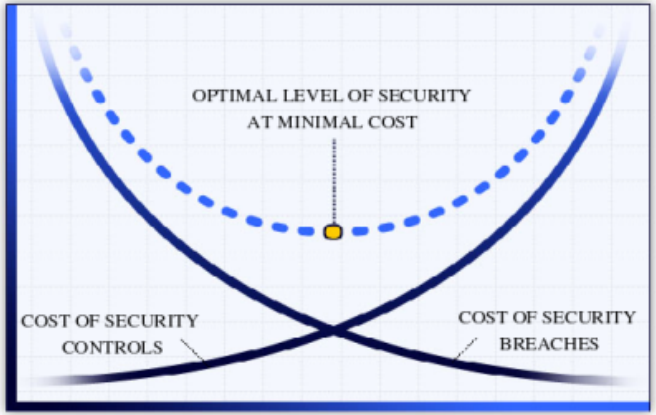
- Doing it yourself
  - Login page sends
    - Hash(sessionid + Hash(userid + Hash(password))
  - Server stores
    - Hash(userid + Hash(password))
  - Registration page sends
    - userid, Hash(userid + Hash(password))
    - Or just userid, Hash(password)
- Better yet – let someone else do it
  - **Passport**, bcrypt, oauth, … packages exist to support this
  - Support using other credentials as well (Facebook, Google, …)

Secure Password Hashing

- Your thread model should look like this:



Remember Security is Relative



- No application is totally secure
  - Any app or system can be broken
  - But you can control the cost to break it
- Make your application as secure as necessary
  - Cost to break much greater than value of breaking it

Next Time

- Privacy

Project Design Presentations

- A week from Wednesday
- Design Documentation
  - 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, testing
- Alternatives
  - 10-15 Minute presentation of your overall design
  - A 3-10 page document describing the above

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.

Homework

- Most of you looked this up
- Most of the ideas are valid, some high-level
- As with any security problem
  - Start with understanding the threat model
  - What are you worried about
  - What do you think can go wrong