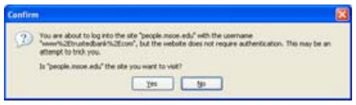
SE-4920: Lecture 18
Web issues…

- …including
  - SQL injection
  - OWASP (Open Web Application Security Project)
- Reading
  - Chapter 25
- Today's Outcomes
  - Describe the basic structure of URLs, HTTP requests, and HTTP digest authentication as they relate to security
  - Explain the use of HTTP cookies
  - Define cross-site scripting
  - Explain an SQL injection attack and various methods of remediation
  - Be familiar with OWASP and the OWASP Top 10 list

1

URL structure

- protocol://[user[:password]@]site[:port]/[infoToSite]
  - protocol = http | https | ftp | …
    - https = HTTP using SSL/TLS, default port 443
  - site = DNS name or IP address
  - User/password illegal for HTTP
    - Most browsers ignore (vulnerability?)
    - http://www.trustedbank.com:viewAccount@people.msoe.edu/~durant/somethingBad/



You are about to log into the site "people.msoe.edu/" with the username "www%2Etrustedbank%2Ecom", but the website does not require authentication. This may be an attempt to trick you.

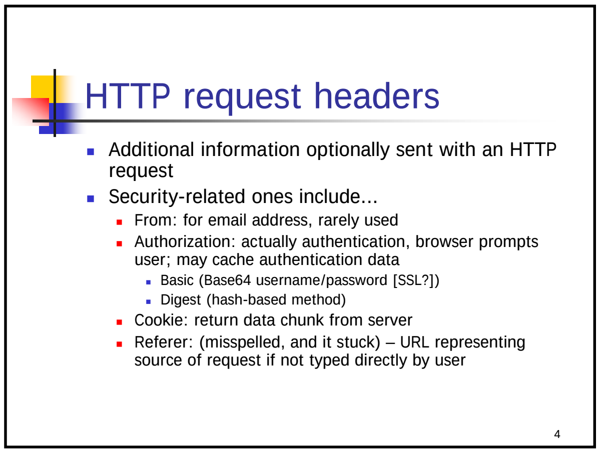
Is "people.msoe.edu/" the site you want to visit?

2

HTTP requests

- GET
  - Retrieve data
  - Any arguments must be in URL itself
- POST
  - Submit data (message body beyond URL)
  - Also receives reply
- HEAD, PUT, DELETE, TRACE, OPTIONS, CONNECT

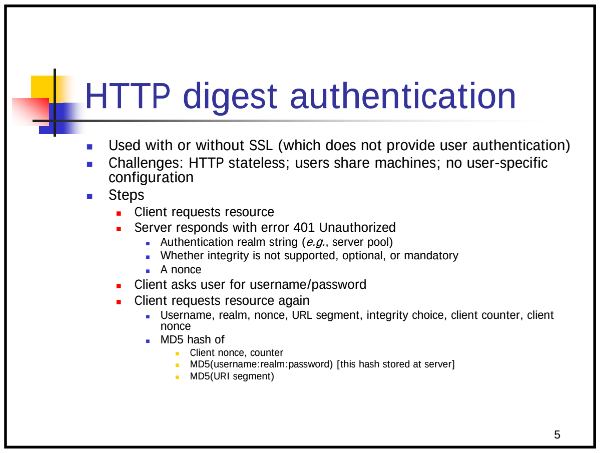
3



HTTP request headers

- Additional information optionally sent with an HTTP request
- Security-related ones include…
  - From: for email address, rarely used
  - Authorization: actually authentication, browser prompts user; may cache authentication data
    - Basic (Base64 username/password [SSL?])
    - Digest (hash-based method)
  - Cookie: return data chunk from server
  - Referer: (misspelled, and it stuck) – URL representing source of request if not typed directly by user

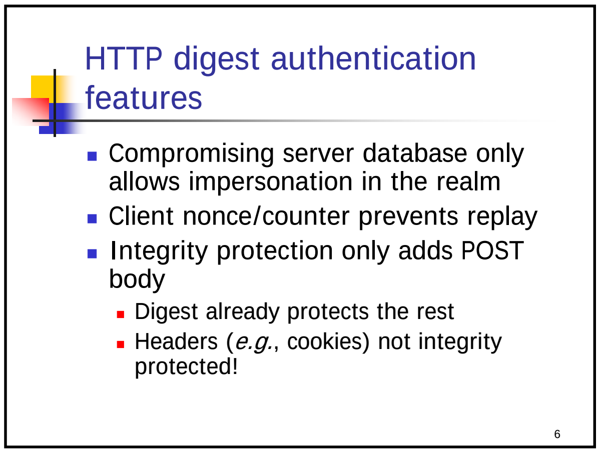
4



HTTP digest authentication

- Used with or without SSL (which does not provide user authentication)
- Challenges: HTTP stateless; users share machines; no user-specific configuration
- Steps
  - Client requests resource
  - Server responds with error 401 Unauthorized
    - Authentication realm string (*e.g.*, server pool)
    - Whether integrity is not supported, optional, or mandatory
    - A nonce
  - Client asks user for username/password
  - Client requests resource again
    - Username, realm, nonce, URL segment, integrity choice, client counter, client nonce
    - MD5 hash of
      - Client nonce, counter
      - MD5(username:realm:password) [this hash stored at server]
      - MD5(URI segment)

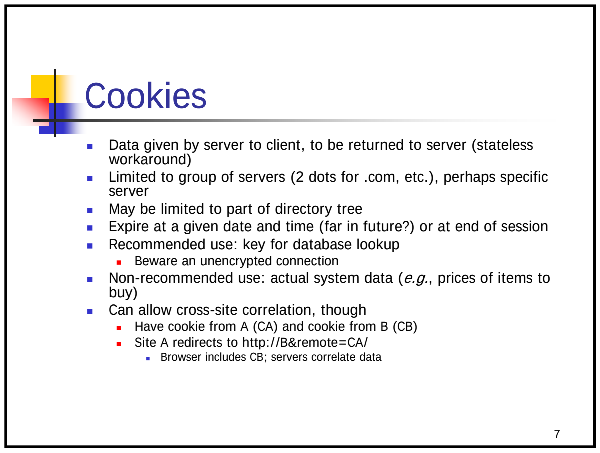
5



HTTP digest authentication features

- Compromising server database only allows impersonation in the realm
- Client nonce/counter prevents replay
- Integrity protection only adds POST body
  - Digest already protects the rest
  - Headers (*e.g.*, cookies) not integrity protected!

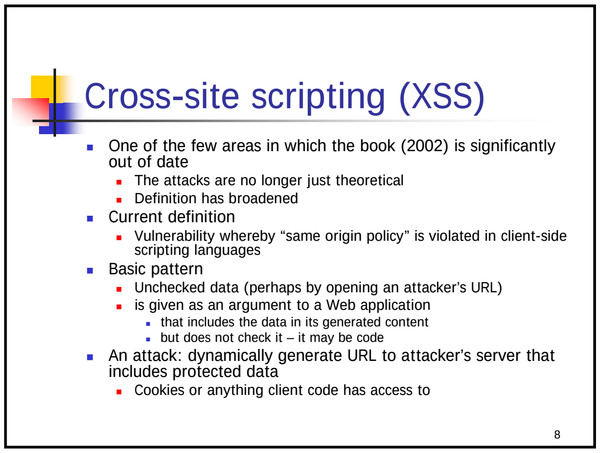
6



Cookies

- Data given by server to client, to be returned to server (stateless workaround)
- Limited to group of servers (2 dots for .com, etc.), perhaps specific server
- May be limited to part of directory tree
- Expire at a given date and time (far in future?) or at end of session
- Recommended use: key for database lookup
  - Beware an unencrypted connection
- Non-recommended use: actual system data (*e.g.*, prices of items to buy)
- Can allow cross-site correlation, though
  - Have cookie from A (CA) and cookie from B (CB)
  - Site A redirects to http://B&remote=CA/
    - Browser includes CB; servers correlate data

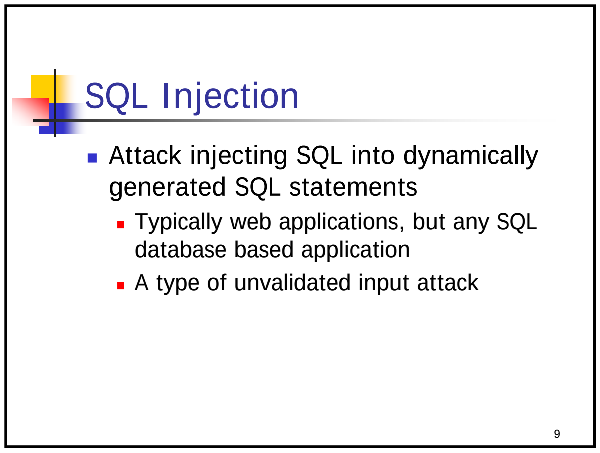
7



Cross-site scripting (XSS)

- One of the few areas in which the book (2002) is significantly out of date
  - The attacks are no longer just theoretical
  - Definition has broadened
- Current definition
  - Vulnerability whereby “same origin policy” is violated in client-side scripting languages
- Basic pattern
  - Unchecked data (perhaps by opening an attacker’s URL)
  - is given as an argument to a Web application
    - that includes the data in its generated content
    - but does not check it – it may be code
- An attack: dynamically generate URL to attacker’s server that includes protected data
  - Cookies or anything client code has access to

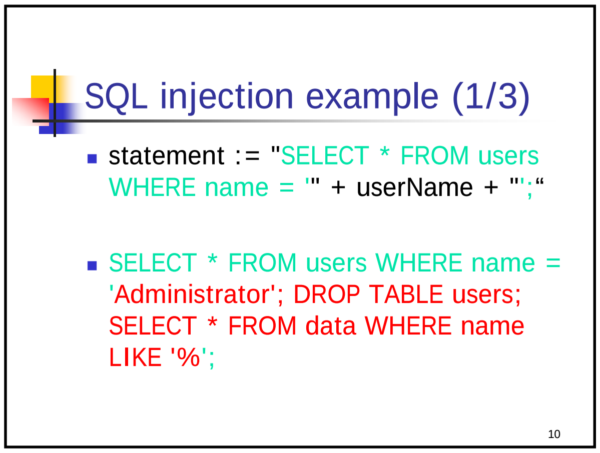
8



SQL Injection

- Attack injecting SQL into dynamically generated SQL statements
  - Typically web applications, but any SQL database based application
  - A type of unvalidated input attack

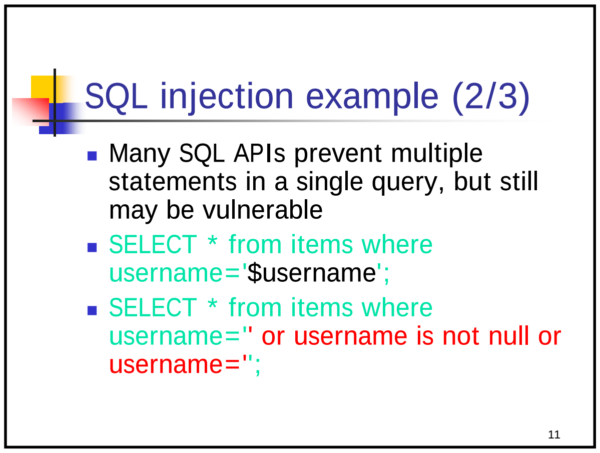
9



SQL injection example (1/3)

- statement := "SELECT * FROM users WHERE name = '" + userName + "';“
- SELECT * FROM users WHERE name = 'Administrator'; DROP TABLE users; SELECT * FROM data WHERE name LIKE '%';

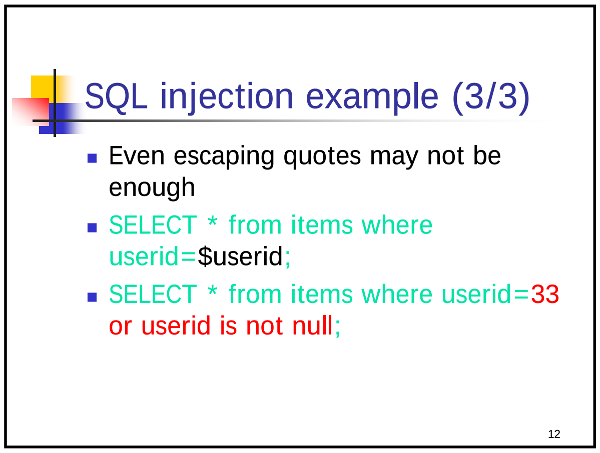
10



SQL injection example (2/3)

- Many SQL APIs prevent multiple statements in a single query, but still may be vulnerable
- SELECT * from items where username='$username';
- SELECT * from items where username='' or username is not null or username='';

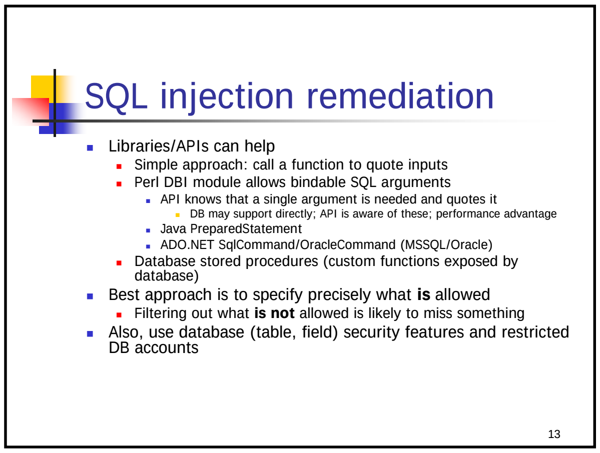
11



SQL injection example (3/3)

- Even escaping quotes may not be enough
- SELECT * from items where userid=$userid;
- SELECT * from items where userid=33 or userid is not null;

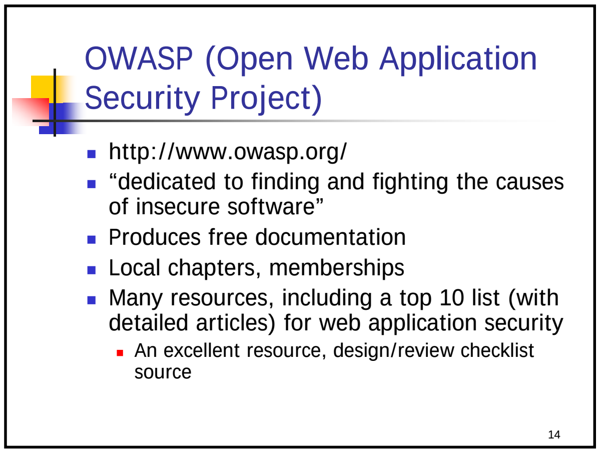
12



SQL injection remediation

- Libraries/APIs can help
  - Simple approach: call a function to quote inputs
  - Perl DBI module allows bindable SQL arguments
    - API knows that a single argument is needed and quotes it
      - DB may support directly; API is aware of these; performance advantage
    - Java PreparedStatement
    - ADO.NET SqlCommand/OracleCommand (MSSQL/Oracle)
  - Database stored procedures (custom functions exposed by database)
- Best approach is to specify precisely what **is** allowed
  - Filtering out what **is not** allowed is likely to miss something
- Also, use database (table, field) security features and restricted DB accounts

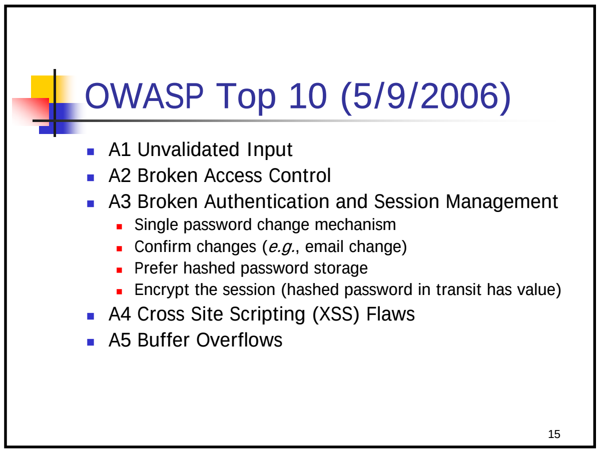
13



OWASP (Open Web Application Security Project)

- http://www.owasp.org/
- “dedicated to finding and fighting the causes of insecure software”
- Produces free documentation
- Local chapters, memberships
- Many resources, including a top 10 list (with detailed articles) for web application security
  - An excellent resource, design/review checklist source

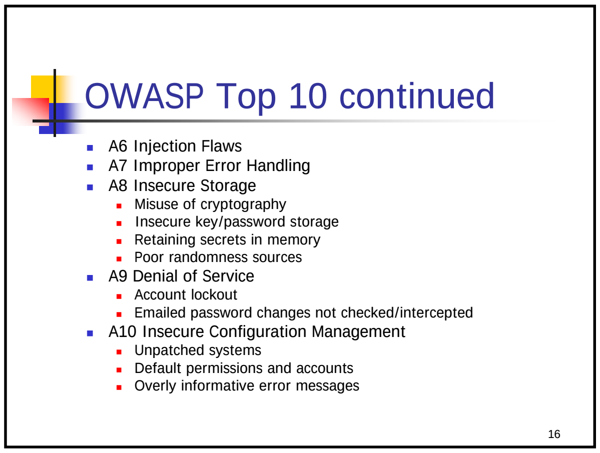
14



OWASP Top 10 (5/9/2006)

- A1 Unvalidated Input
- A2 Broken Access Control
- A3 Broken Authentication and Session Management
  - Single password change mechanism
  - Confirm changes (*e.g.*, email change)
  - Prefer hashed password storage
  - Encrypt the session (hashed password in transit has value)
- A4 Cross Site Scripting (XSS) Flaws
- A5 Buffer Overflows

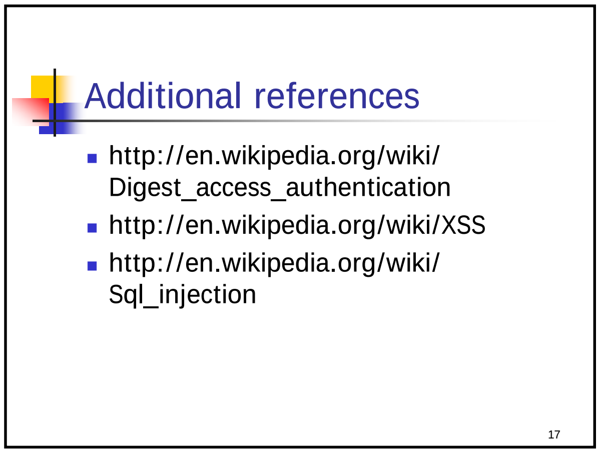
15



OWASP Top 10 continued

- A6 Injection Flaws
- A7 Improper Error Handling
- A8 Insecure Storage
  - Misuse of cryptography
  - Insecure key/password storage
  - Retaining secrets in memory
  - Poor randomness sources
- A9 Denial of Service
  - Account lockout
  - Emailed password changes not checked/intercepted
- A10 Insecure Configuration Management
  - Unpatched systems
  - Default permissions and accounts
  - Overly informative error messages

16



Additional references

- http://en.wikipedia.org/wiki/Digest_access_authentication
- http://en.wikipedia.org/wiki/XSS
- http://en.wikipedia.org/wiki/Sql_injection

17