Web Security Issues

CSS 322 – Security and Cryptography

Contents

- A selection of issues (some historical, others still current) related to security of web applications

URLs and URIs

- URI = Uniform Resource Identifier; can be either:
  - URL = Uniform Resource Locator; or
  - URN = Uniform Resource Name
- URL structure:
  - Protocol://domainname.com/directory/file
  - E.g. http://ict.siit.tu.ac.th/sgordon/index.html
- URLs can include username and password:
  - ftp://siit:stevecourse@ict.siit.tu.ac.th/sgordon/lecture.pdf
  - HTTP does not use username/password but other protocols (e.g. FTP) may
- Port numbers can also be included:
  - http://ict.siit.tu.ac.th:8080/sgordon/index.html
  - HTTP defaults to port 80 if no port number given

HTTP

- HTTP = HyperText Transfer Protocol
    - Request/response protocol, with two main request types from client:
        - GET – request a web page from server
        - POST – send information to server
            - E.g. when you fill in a form, POST is used to send the form data to the web server
    - Response contains content/information from server and status code (e.g. 200 OK; 404 File Not Found; …)
    - Request may contain many fields, including:
        - FROM – email address of user; can lead to spam; no longer used by most browsers
        - AUTHORIZATION – username logged in; used for authentication
        - COOKIE – discussed soon …
        - REFERRER – URL of page from which the client came from; can be used to track users' activities

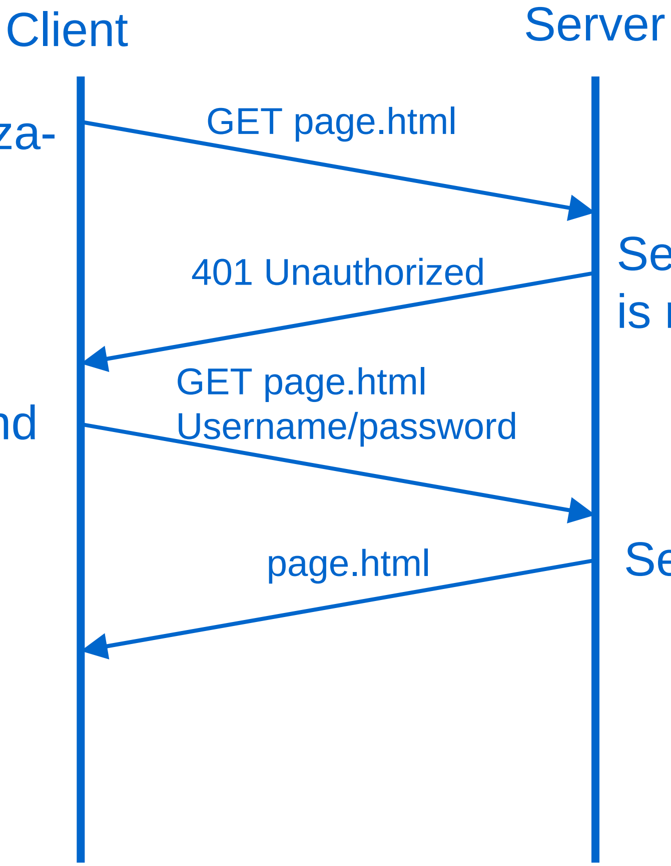
HTTP Authentication

- Securing web access:
  - HTTP over SSL (HTTPS): secure, but complex to implement
  - In-built HTTP authentication:
    - HTTP Basic Authentication
    - HTTP Digest Authentication
- Basic approach is:
  - Web browser requests a web page
  - Web server sends the web page (HTML) back to client

Challenges of HTTP Authentication

- HTTP is stateless
- User needs to be authentication, who may be connecting from machine with no user-specific information (e.g. Internet café)
  - Hence, we need to use passwords
- A lot of requests may be to same server; so we want to avoid extra authentication for each request
- Aim: If attacker steals server database, attacker cannot impersonate user on that server, or on other servers (even if use same password)

HTTP Basic Authentication



Client

Server

Client requests a page (does not know authorization is need)

GET page.html

Server tells client authorization is needed for a realm

401 Unauthorized

Client requests page and includes username and password

GET page.html
Username/password

Server sends page.html

page.html

Username and password are sent as plaintext – very insecure!

HTTP Digest Authentication

- Same as Basic Authentication, except MD5 hash of password is sent:
  - HA1 = MD5(username, realm, password)
  - HA2 = MD5(url)
  - Response = MD5 (HA1, server nonce, nonce count, client nonce, quality of protection, HA2)
- Browser will cache hash, URL and realm for the user
  - Subsequent requests do not need user input
- Server stores hash of password and associated realm
  - If attacker steals server database, can impersonate user in same realm (not in other realms or servers, even if same password)

HTTP Digest Authentication

- Server sends a nonce value to client
  - Client sends nonce back to server, and also increment the nonce count (nc) by 1 for each subsequent request
    - Avoids server always requesting authorization (saves 1 round trip time)
    - Allows server to identify replay attacks (server stores nonce and nonce count – if receive same value again, then replay)
- Quality of protection (qop)
  - Can specify authentication and/or integrity
    - auth = authentication only
    - auth-int = authentication and integrity
    - auth,auth-int = authentication is required, integrity is preferable

HTTP Authentication Example

- Initial Client Request

```
GET /dir/index.html HTTP/1.0
        Host: localhost
```

- Server 401 Response

```
HTTP/1.0 401 Unauthorised
Server: SokEvo/0.9
Date: Sun, 10 Apr 2005 20:26:47 GMT
WWW-Authenticate: Digest
        realm="testrealm@host.com",
        qop="auth,auth-int",
        nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
        opaque="5ccc069c403ebaf9f0171e9517f40e41"
Content-Type: text/html
Content-Length: 311
<<error web page included here>>
```

HTTP Authentication Example

- Client Authorized Request

```
GET /dir/index.html HTTP/1.0
          Host: localhost
Authorization: Digest
          username="Mufasa",
          realm="testrealm@host.com",
          nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
          uri="/dir/index.html",
          qop=auth,
          nc=00000001,
          cnonce="0a4f113b",
          response="6629fae49393a05397450978507c4ef1",
          opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

quality of protection

nonce count

client nonce

- Server Response
  - Sends the web page

Cookies

- HTTP is stateless
  - There is no information stored at server that connect multiple requests from clients
  - Many web applications want to know if a request is a follow-up from a previous request
- Cookies add state to web browsing (HTTP)
- A cookie is a data structure created by server and stored at client
- Cookies can be used to:
  - Create electronic shopping carts
  - Log in to web sites
  - Personalise web pages
  - Track browsing activities of users

Why are Cookies Needed?

- Alternatives?
  - Assume TCP session uniquely identifies user
    - Client IP, Client Port, Server IP, Server Port
    - Every request from same Client IP/Port to a web server is treated as from one unique user
    - But in practice, many users go through proxy (e.g. SIIT):
      - From a web servers point of view, all users on SIIT network are seen as coming from same Client IP/Port
  - Browser include username in every request (possible in HTTP)
    - But what if you want to browse anonymously
  - Browse includes random number X representing user in every request
    - Allows you to browse anonymously, but if attacker intercepts X, they can impersonate you

How do Cookies Work?

- Procedure:
  - Web server creates a cookie
    - E.g. when you first visit a web site
  - Web server sends cookie to client in HTTP response
    - In HTTP header:

      set-cookie: name=value
  - Client stores the cookie
  - When client visits the web server again, it sends the cookie, unchanged
    - In HTTP Get request:

      cookie: name=value
- Now the server can connect the current page you are visiting with the previous pages you visited

Cookie Rules

- Cookies have lifetimes
  - Cookie must be deleted from browser if past its expiry date or (if no persistent) when browser closes
- Cookies can only be sent to a domain:
  - If web server www.siit.tu.ac.th sends your browser a cookie, you can only send it back to any machine in tu.ac.th domain (e.g. siit.tu.ac.th or eng.tu.ac.th – your browser cannot send cookie to google.com)
    - This is simple way to prevent tracking specific users, however some tracking is still possible …

Tracking Users with Cookies

- What?
  - Identify which sites a user visited, but not identify the user
  - Identify that user X has visited a particular site several times
  - (With extra information) Identify a user and all the sites they visit
- Why?
  - Information about user’s behaviour is valuable
    - Target advertising
    - E.g. insurance company deny you medical insurance if they know you have visited sites about serious diseases
  - Many privacy issues arise (that we do not have time to cover!)
- How?
  - Sites collude (share information) or put information into REFERRER field
    - If user logs in to one site, can identify user across all sites
    - Correlating information across sites can be performed using:
      - Web server logs, proxy logs, HTTP redirects or embedded images, …

Site Spoofing

- A malicious user creating a website pretending to be another website:
  - As a result, the malicious user can obtain account details (PINs, passwords) as well as track interests
- If using SSL (HTTPS), this is impossible?
  - SSL: assured you are talking to correct site if:
    1. No CA you trust issued a certificate to verify BadBank's public key belongs to the name GoodBank
    2. Your list of trusted CAs (e.g. in browser) is not modified to include public key's of un-trusted CAs
    3. URL you are browsing to is for organisation you expect
  - Example:
    - BadBank has a domain: gg.tv
    - You click on a link: http://www.goodbank.com!secure_login_to_goodbank@gg.tv/
    - Takes you to gg.tv (but you think it is www.goodbank.com)
    - SSL will check certificate of gg.tv – will be successful if gg.tv has certificate signed by CA in your browser

User Impersonation

- HTTP requires username/password for each page
  - But web browsers cache username/password in cookies so easy for user
- If someone users browser after you, the username/password may still be cached (they can login as you)
  - Should only cache cookies for short time
  - Cookies should be deleted when browser closes (not all are!)
- Browsers now save usernames/passwords in stable storage (not in cookies)
  - Even harder to force browser to delete information; hence easier for malicious user to impersonate you
- Similar issues arise with browsers storing telephone numbers, addresses, email etc to make life easier for user
    - Many security and privacy issues arise from this