Lecture 10

Friday, 16 December 2022 15:34

prepare description of practical exercise solutions

---

prepared statements fix non-data parts of queries

could also be implemented on the server side
i.e. 'template function' which is stored on server;
client provides missing data when performing query

might also be faster than providing statements at runtime

HTTP is stateless

advantage: slightly more resilient to DoS attacks

disadvantage: problematic w.r.t. authentication

ephemeral state e.g. cookie must be sent to server in every HTTP request from client

alternative: hidden form fields in HTML
<input type="hidden" name="key" value="value">
problem: data is lost when browser (tab) is closed

generally: beneficial for server to store data at client
but at cost of more communication
and does not work well for data that should not be manipulated by client
unless we store handle at client (i.e. session ID (large, so difficult to guess)) and data at server

cookies: small files that represent handle to trusted state stored at server

- domain name
- expiry date → could be: session end (i.e. browser close)
- subdirectory which may access cookie

stored at client: privacy/legal issues

can be used for tracking → personal information

attackers attempt to steal cookies which can be used to impersonate the cookie owner

---

CSRF: make authenticated user access a link which performs an unintended action → modifies state
cross site request forgery

abuse trust relationships between client and server
if attacker cannot perform an action themselves, make the victim perform the action instead

major reason for not using GET requests to modify state

note that clicking a link might not be necessary
since e.g. images are loaded automatically

Javascript & same-origin policy
↓
Client-side, so can be much faster

Javascript can be embedded in HTML files
or loaded from separate remote file

JS can:
- track browser events e.g. mouse click
- communicate over HTTP
- store + read cookies

We do **not** want:
- no disk read/write
- access to other windows/tabs
- access to other domains → same origin policy

scripts included in page A can only access data in page B if they have the same origin

same origin → combination of
- protocol
- domain
- port

some services need sharing that is disallowed by same origin policy
e.g. SSO (single sign on)

XSS: attempt to circumvent same origin policy
cross site scripting

trick browser into believing script comes from trusted source
- send cookie to attacker
- make client perform unintended request

persistent XSS: store script on legitimate website e.g. comment form
countermeasures:
- two factor authentication
- check uploaded files for code
- set cookie with HttpOnly flag

reflected XSS: make user send script to server, which then echoes it back
e.g. search function which echoes back search term in result
countermeasures:
- avoid echoing of input
- perform input validation and make sure it does not contain scripts

server-side XSS: make server access website

e.g. bypass firewall, access server itself (localhost), access back-end systems which are not publicly accessible