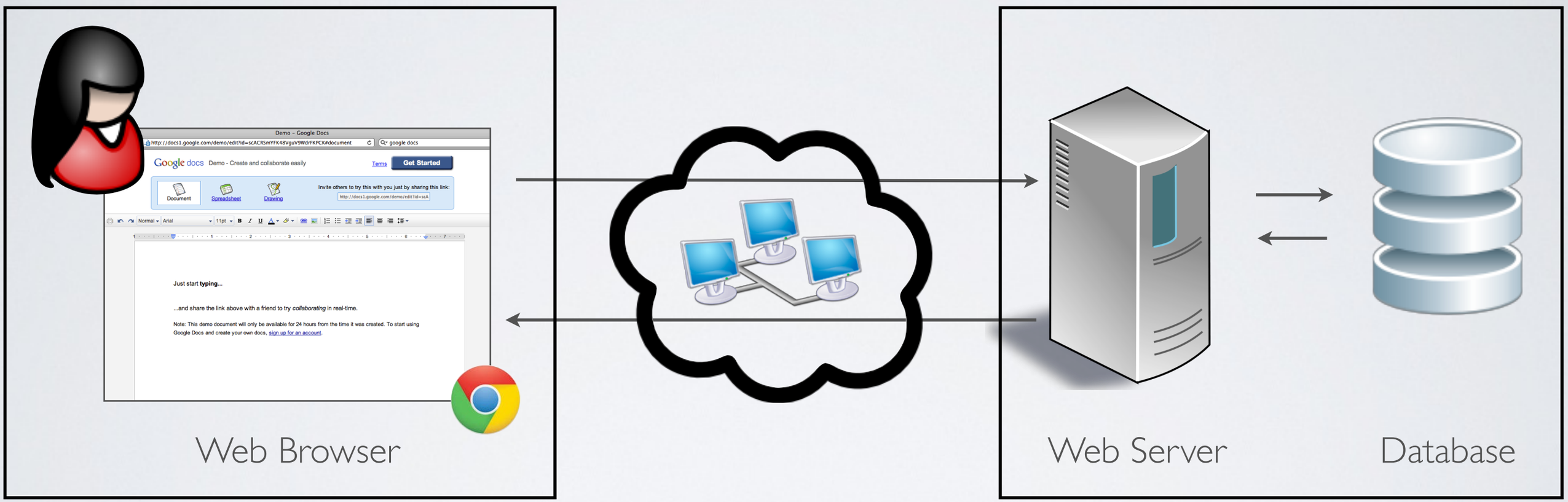
Web Security

Kc Udonsi

The Big Picture

The web architecture

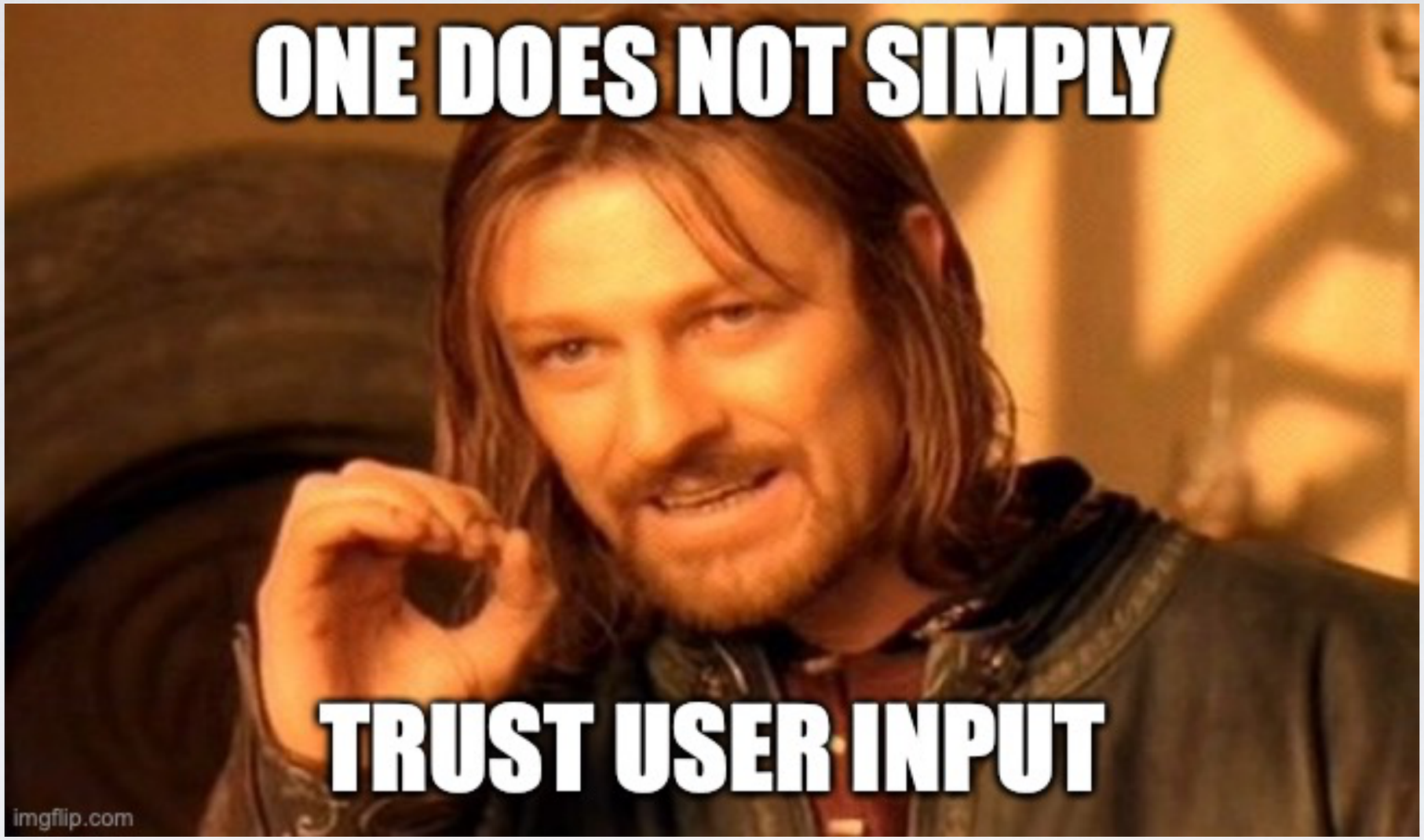


Securing the web architecture means securing ...

- The network
- The DNS (Domain Name System)
- The web server operating system
- The web server application (*Apache* for instance)
- The database application (*Oracle* for instance)
- The web user
- The web application

Our focus here!

Challenges of Web Security - TLDR;



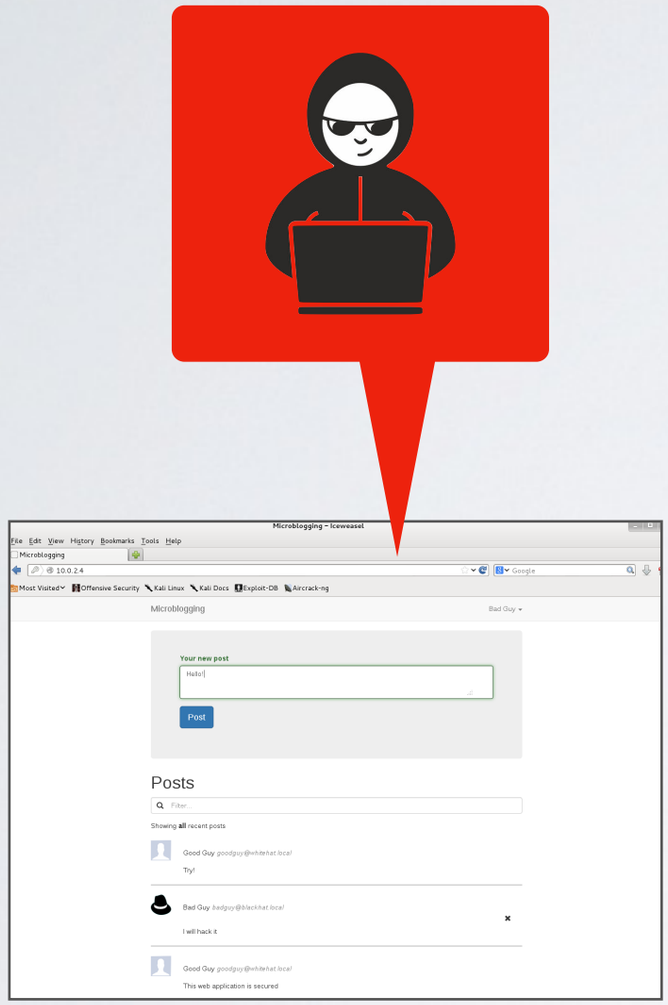
- All user input is untrusted

Challenges of Web Security

- Security across all layers of technology
- Very expressive (markup) languages
- Weakly verifiable state variables e.g cookies
- Loosely defined protocols and schemes
- URL parsing complexities
- Weakly verifiable content types e.g Polyglot files
- Use of middleware, routing, caches, proxies etc

Client Side

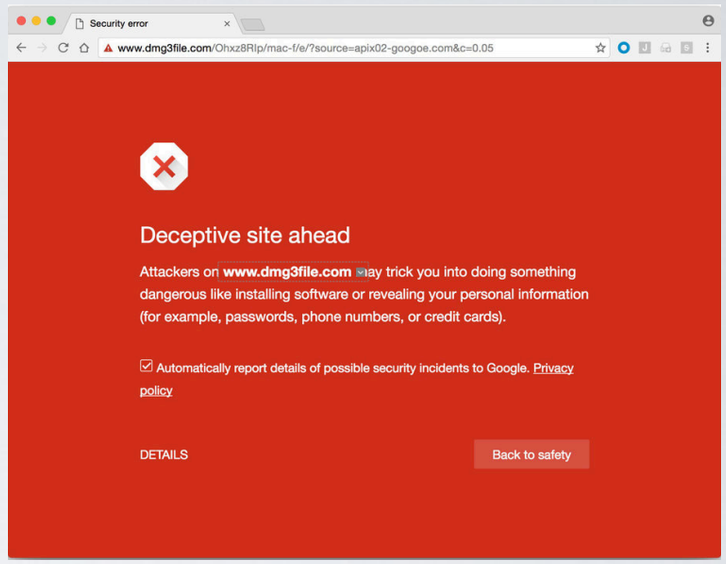
Mitigating Client Side Threats

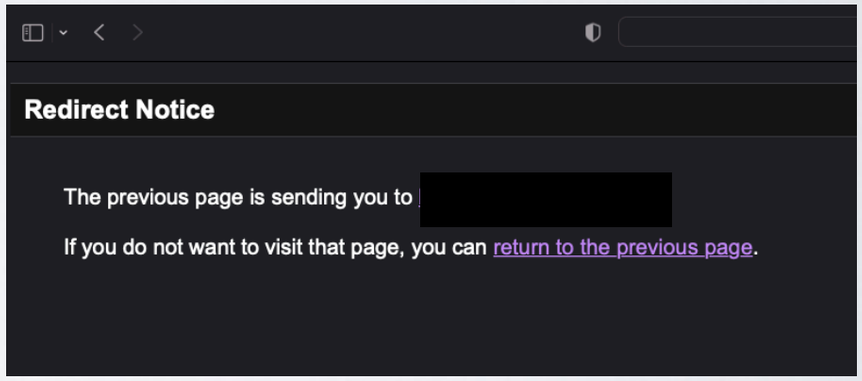


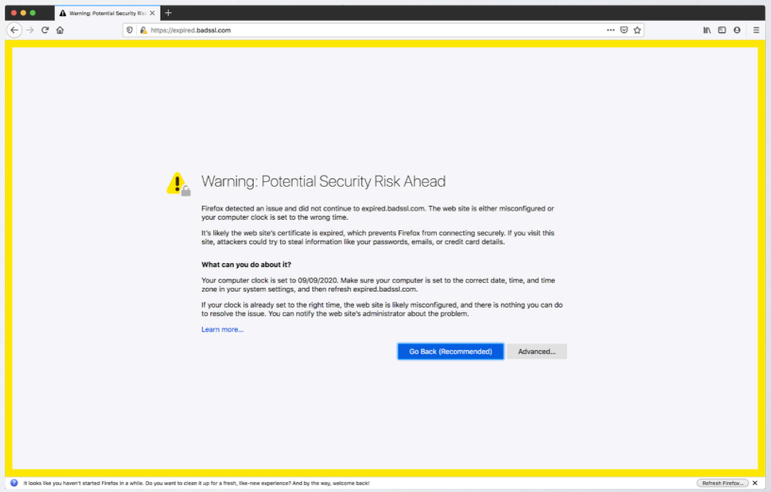
- Malicious URL detection / navigation warning
- Same Origin Policy
- Content Security Policies
- Privilege Separation

Malicious URL detection / Navigation warning

- Heuristics or Blacklist to prevent navigation
- Navigation awareness
- Certificate validation
- HTTS Strict Transport Security Enforcement
- HTTPS Everywhere
- Disable Javascript







Same-Origin Policy

- Same-Origin Policy (SOP) restricts how resources loaded from a domain may interact with resources from another domain

- **Domain:** (Scheme, host, port) tuple

- `http://utsc.utoronto.ca:80` =/= `https://utsc.utoronto.ca:443`

- E.g Malicious JS in one domain cannot access resources of other sites the user is visiting

Same-Origin Policy - CORS

- Very strict - often relaxed with Cross Origin Resource Sharing (CORS)
- Additional HTTP headers that specify permitted/trusted origins and access control e.g **Access-Control-Allow-Origin**

https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

Content Security Policy (CSP)

- Intended to mitigate XSS and other injection attacks
- Server returns **Content-Security-Policy** HTTP header or <meta> tag
- Controls what resources can be loaded, how they must be loaded and what can be executed.

```
<meta
  http-equiv="Content-Security-Policy"
  content="default-src 'self'; img-src https://*; child-src 'none';" />
```

All content must come from origin, can include images from https origins, no inline scripts, no nested contexts like iframes are permitted

https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

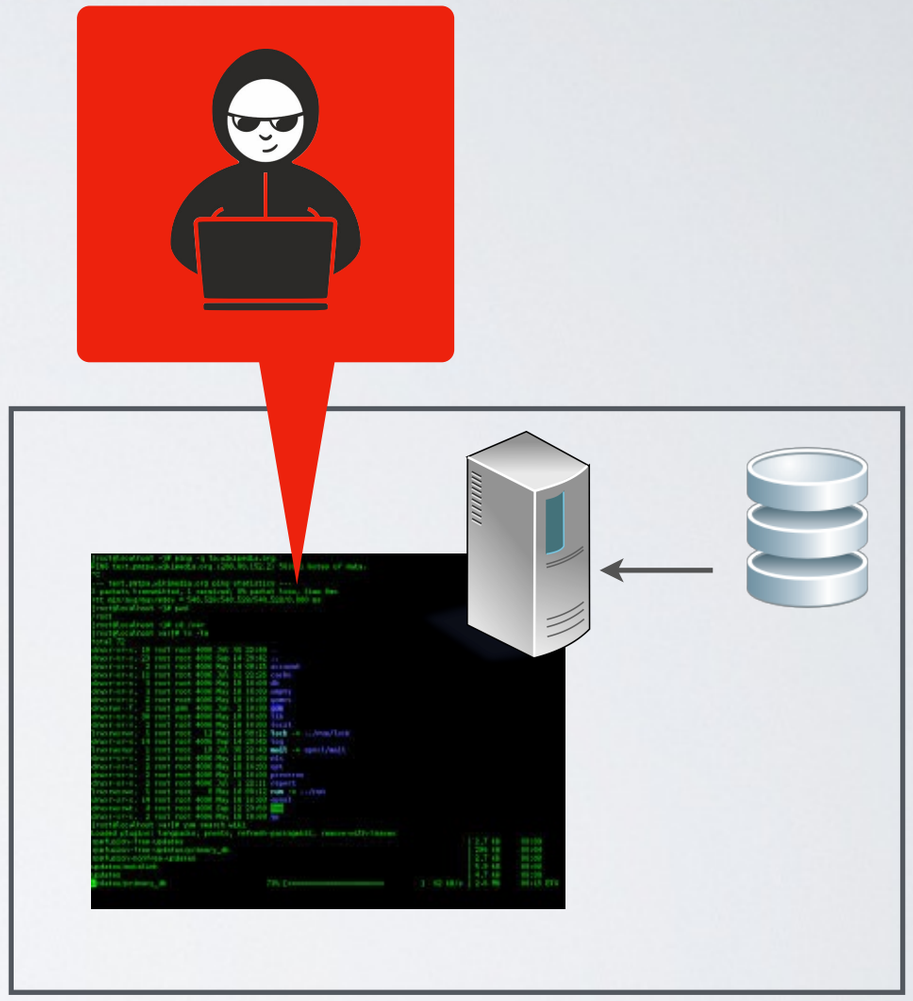
Privilege Separation

- Intended to impact of vulnerabilities exploited
- Divide client application into modules were each is limited to specific privileges required
- Helps to fine tune security controls and enforce boundaries
- E.g Admin area, authenticated area, guest/anonymous area etc

Server Side

Mitigating Server Side Threats

- Securing Authentication, Sessions and Access control
- Safe-handling of input data across boundaries
- Errors and Exception management
- Logging and alerting
- Web Application Firewalls



Securing Authentication, Sessions and Access

- **Authentication:** account creation/recovery/password change/credential validation
- Safe authentication error reporting
- Safe recovery and password change
- Avoid password requirements that weaken security
- Support for Multi-Factor-Auth

https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

Securing Authentication, Sessions and Access Cont.

- **Sessions:** Managing user activities across application and devices
- Set of data structures tracking user's interaction with application server-side
- **Unique unforgeable secret** sent automatically after receipt back to server by client for identification
- May employ cryptography for enhanced security
- May be stored in Cookies or browser local storage

https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

Securing Authentication, Sessions and Access Cont.

- **Access Control:** Fine-grained logic to determine R/W permissions per user or object.
  - Determines if a user is authorized to perform an action
  - Often implemented as roles
  - Avoid assumptions as to how users might interact with the application

https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

Safe-handling input data across boundaries

- Stored procedures, parameterized queries for SQL Injection
- Validation at each boundary / input point
- Whitelist/blacklist
- Sanitization
- Challenges of expressive languages on the web

Snippets of Doom: An Exercise

```
def path = System.console().readLine 'Enter file path:'
if (path.startsWith("/safe_dir/"))
{
    File f = new File(path);
    f.delete()
}
```

```
// Get username from parameters
String username = request.getParameter("username");
// Create a statement from database connection
Statement statement = connection.createStatement();
// Create unsafe query by concatenating user defined data with query string
String query = "SELECT secret FROM Users WHERE (username = '" + username + "' AND NOT role = 'admin')";
// ... OR ...
// Insecurely format the query string using user defined data
String query = String.format("SELECT secret FROM Users WHERE (username = '%s' AND NOT role = 'admin')",
username);
// Execute query and return the results
ResultSet result = statement.executeQuery(query);
```

```
@RestController
public class XSSController {

    @GetMapping("/hello")
    ResponseEntity<String> hello(@RequestParam(value = "name", defaultValue = "World") String name) {
        return new ResponseEntity<>("Hello World!" + name, HttpStatus.OK);
    }

}
```

https://github.com/snoopysecurity/Vulnerable-Code-Snippets

Errors and Exception management

- Avoid overly verbose external errors. Use uninformative messages publicly
- Log sanitized exceptions and errors
- Handle errors gracefully
- Poor handling may inform attacker of application internals even without displaying content

Logging and Alerting

- Audit logs are valuable for intrusion investigation
- Inform on application or attacker behaviour during / post breach
- E.g Authentication audits, critical transactions, successful attack mitigation
- Can be enabled on application and all intermediaries
- Rules or alerts triggers to inform administrators on anomalous behaviour

Web Application Firewalls

- Internal or external component performing intrusion prevention or detection in the capacity of a reverse-proxy
- Often leverages payload signatures or customized rules or policies for detection
- Often parses HTTP requests and response therefore could also be vulnerable to bypasses
- May protect against CSRF, XSS, SQLi etc

Good Code Security Hygiene

- Dev Sec Ops
- Web Application threat modelling and evaluation
- External assessments e.g Vulnerability Reward Programs
- Continuous education and awareness
- Defense-in-Depth; layered security to localize and mitigate threat impact
- Safe logging hygiene

Resources

- https://developer.mozilla.org/en-US/docs/Web/HTTP
- https://portswigger.net/web-security/cors
- https://portswigger.net/research/bypassing-csp-with-policy-injection
- https://portswigger.net/research/bypassing-csp-using-polyglot-jpegs
- https://portswigger.net/research/exploiting-cors-misconfigurations-for-bitcoins-and-bounties
- See corresponding Resource section on course website