Web application security

- Web server technologies and security models
- HTTP protocol and security
- Input validation
- Untrusted client
- Secret data
- Securing server
- Some simple attacks
Web server technologies

- CGI — *Common Gateway Interface*, FastCGI
- SSI — *Server-Side Includes*
- ISAPI
- ASP, ASP.NET
- PHP
- EmbPerl
- JavaScript, Node.JS
- Java Servlets, EJB
- Framework-specific and specialized web servers
Web server security models

- One process, one server-side user account, static pages
- One process, one server-side user account, scripts interpreted in the same process
- One process for the web server, another for backend (possibly running as another user)
- Subprocesses/threads running as different users (suexec family), possibly interpreting scripts
HTTP protocol and security

- Designed for static page serving
- Modified for dynamic web pages and caching
- HTTP has no notion of session
  - Carried by cookies, URL parameters, hidden form fields etc
  - Each application or framework needs to implement its own sessions
  - Must create strong session identifiers
  - Session idle timeout and maximum lifetime
  - Bind sessions to client IP + maybe other data
  - Begin a new session on authentication
HTTP protocol and security

• POST vs GET:
  – URLs in logs and caches
  – Redirect erroneous GET to error instead of processing as POST

• Fail securely
  – HTTP status codes — 200 OK + only one error code to not reveal additional details
  – Do not reveal internal info in error messages (backtraces, line numbers, paths etc) — only log them

• Another minor data leak — avoid detailed server signature in error messages and auto-generated pages
Input

- All aspects of an HTTP request must be validated, regardless of what a "normal" request is supposed to look like
- Assume attackers will change cookies, hidden form fields and POST parameters
- Assume attackers will POST the URLs in the wrong order, at the wrong time, and for the wrong reasons
- **Example:** User-Agent: ‘ OR 1=1 --

```java
String userAgent = request.getHeader("user-agent");
...
String sQuery =
    "DELETE FROM UP_USER_UA_MAP WHERE USER_ID=" +
    userId + " AND USER_AGENT='" + userAgent + "'";
...
stmt.executeUpdate(sQuery);
```
Untrusted client

- Expect that the browser has been subverted — attacker can use his own program for HTTP
  - What would you use to submit requests without browser?
- Don’t trust any client-side input validation — it is useful for providing feedback to the user (JavaScript, Flash, Java applets) but validation must be done on server side
- Don’t expect any form submits be consistent with your forms
- Hidden variables are neither hidden nor immutable
- Client can send any headers — Referer, User-Agent, Cookie etc

wget --referer http://www.example.com
   --header 'User-Agent: VillaZilla/1.0'
   --post-data 'user=foo&password=bar' www.ut.ee
Secret data

- Browser is an open book — nothing you send to it stays secret
- Don’t expect to restrict access with CSS & Javascript by downloading but not showing data by default
- Examples
  - tcpflow port 80
  - STRACE/Detours (inline hooking)
  - Tamper Data browser extension
Practice at home

- http://www.try2hack.nl/

- Try to get as far as you can but at least level 5.
Secret data

- Attackers will look for patterns in URLs, URL parameters, hidden fields, and cookie values.
- They will read any comments in your HTML.
- They will contemplate the meaning of your naming conventions, and they will reverse engineer your JavaScript.
- They will use search engines to dig up information from other users of the application and to find stale pages you no longer recall you deployed.
- If you are using Asynchronous JavaScript and XML (AJAX), expect that the attacker will decipher the data transfer protocol and message formats that the client is trading with the server.
What files are accessible?

- Make available ONLY the files that are meant to be accessed by the clients. This does not include
  - Additional program files (.inc)
  - Data files
  - Authentication data (.htaccess)
  - Temporary files
  - Source code (and its backups)
  - Session files
  - Uploaded files (maybe)
More good security practice

- What files are considered for execution/interpretation on the server?
- Under which credentials the code runs?
- Only minimal set of users must be able to modify the files under web root
- Gather executable pages into one place on server?
- Names and data files of external programs should contain full path if possible
- Web server pseudo-user should only be allowed to access minimal set of files
- File upload dangers
- Be wary of `eval` and other code-interpreting constructs
Some simple attacks

- Double decoding attacks (URL encoding, UTF-8, base64, ...)
- HTTP Response splitting — attacker gets newline into headers returned by the server
- Open redirect (phishing can use it to fool users)

```java
String nextPage = request.getParameter("next");
if (nextPage.matches("[a-zA-Z0-9/:?&_\\]+") ) {
    response.sendRedirect(nextPage);
}
```
Open redirect fixes

String nextPage =
    pageLookup.get(request.getParameter("next"));
response.sendRedirect(nextPage);

OR

String nextPage = request.getParameter("next");
if (nextPage.matches("[a-zA-Z0-9/]+") {
    response.sendRedirect("http://example.com/" + nextPage);