Security Audit Checklist: Code Perspective

General tips

- Whitelist over blacklist
- Deny by default
- Least privilege principle
- Limit resource consumption (DoS)
- Judicious use of shell calls, eval functions

Admin strategies

- Examine log files for unexpected activities
- Examine database for strange entries
- Check for odd user accounts, groups
- Check for incorrect user rights, group memberships
- Use correct config files (apache, php, mysql) settings

Don't trust input data

- Form input, POST/GET
- Command line arguments
- Configuration files
- Environment variables
- Cookies
- Input files

Validate all input data

- Server and client
  - Before saving input
  - Before using input
- Escape printed/executed user input data
  - Validate input printed in error messages
- Deny by default if unsure
- Use regexes to validate
- Careful with user-provided file names

Error messages

- Catch exceptions
- Check result codes
- Don't display "too" helpful errors:
  - Variables in scope
  - Failing SQL query
  - Stack trace
- Print error details to log instead of in app (filter passwords, sensitive data)

Sensitive data

- Use encrypted external files to store passwords to DB connections, other passwords (not hardcoded)
- Check credentials upon each load of restricted page
- Store config files outside of web-accessible directory (.htaccess "deny from all")
- Not stored in cookies, sessions
- Not logged in log files

Sessions

- Stealing a session id: using web app as someone else
- Store sensitive session information in database keyed by session ID instead of in session variable
- Make log-out button prominent
- Expire sessions unused past ~20 min
- Expire sessions on server and client

Files

- Use absolute paths
- Set file permissions, directory permissions
  - For already-existing files
  - For files created by application
- Throw errors when overwriting already existing files
- Check file is not a symbolic link before opening
- Unique/difficult to guess file names for temporary files (symbolic link attack)
- Open files with lowest level of permission needed

Ruby on Rails

- Use `escapeHTML()` / `h()` to escape input in HTML
- Use `escape_javascript()` for input within JS functions
- Use `sanitize_sql()` for `connection,execute()`, `Model.find_by_sql()`
- Pass array or hash in conditions fragments (`:conditions => ["login = ? AND password = ?", name, pass]`)
- Use built-in active record validations
- Use private and protected in controllers for methods that should not be actions
- Mass assignment: use `attr_accessible` to specify attributes accessible for mass-assignment
- Use `filter_parameter_logging` on sensitive attributes so Rails logs do not store them
- Use `before_filter :only =>` [...] instead of `:except =>` [..]

Java/JSP

- Use `PreparedStatements` to update databases
- Don't try to do HTML-encoding yourself; use library:
  - `lang` package in Apache Commons Project (http://commons.apache.org/lang/)
  - `StringEscapeUtils`: `escapeXML`, `escapeHTML`
- Perform logging from a .jsp page using the global `log()` function
- Use a `SecurityManager` when running untrusted code
- Limit publicly accessible static/global shared data
- Use encryption algorithms found in `javax.crypto.*` instead of writing own/using others'

PHP

- Use `htmlspecialchars()` to escape input in HTML
- Use `mysql_real_escape_string` / `pg_escape_string` for SQL statements
- Use `is_numeric()`, `ctype_digit()`, regexes, variable handling functions for validation
- Deploy with `register_globals`, `display_errors` off; `log_errors` on
- Commonly disabled functions: `ini_set()`, `exec()`, `fopen()`, `popen()`, `passthru()`, `readfile()`, `file()`, `shell_exec()` and `system()`
- Tools: Spike PHP Security Audit Tool, PHP Security Scanner PhpSecInfo

Security Audit Checklist: Attacker Perspective

General

- View source
- Trigger error messages
  - May contain useful information, filenames, etc

URL discovery

- Directory traversal
- Increment/decrement numeric ids
- Guess filenames
- Try connecting to different ports (SSH, FTP, mail, etc)
- Modify query parameters
- Google hacking

Bypass client-side validation

- Disable/modify validating javascript
- Modify pre-set form values
  - Hidden
  - Radio
  - Select
- Modify cookies

Injection

- HTML
  - User-provided data is output unescaped
  - Could be used for XSS
- SQL in username/password fields
  - `; DROP TABLE foo --`
  - `' OR 1=1 --`
- SQL In URLs
  - http://abc.com/index.php?id=10 AND id=11
- JavaScript/Ajax requests
- Anything that should be escaped but isn't

Login

- Repeatedly submit login form; is there a lock-out?
- Try various user names for "wrong password" feedback (gives details into login/password scheme)
- See if log in locks out after N failed attempts; if there is a delay, captcha
- Weak "forgot password" setup?
- Check cookies when logged in; see if storing vital information
- Login done over a secure channel? (man-in-the-middle)

Other

- DoS: look for slow/computationally intensive things to request multiple times in succession
- Check for weak or breakable forms of encryption
- Check for unsigned security certificates

Useful Tools

- Firebug
- Life HTTP Headers Firefox extension
  - Useful for capturing, modifying, and re-playing AJAX requests