Lecture Introduction into Cyber Security

Web Security

Asya Mitseva, M.Sc.
Prof. Dr.-Ing. Andriy Panchenko

Chair of IT Security
Brandenburg University of Technology Cottbus-Senftenberg

17 January 2019

Brandenburgische Technische Universität Cottbus - Senftenberg

UNIVERSITÉ DU LUXEMBOURG

Erasmus+

Introduction

- **Significant part of our daily life is in the Web**
  - eLearning, eGovernment, eHealth, social media, etc.
- **Websites contain different technologies**
  - Static content is represented by *Hypertext Markup Language (HTML)*
  - *Cascading Style Sheets (CSS)* specifies how display webpage
  - *JavaScript* code requiring client-side computation and interaction
- **Web browser**
  - Use of *Domain Object Model (DOM)* to represent web pages
  - Use of *Same-origin policy (SOP)* to perform access control when scripts within webpage are executed

Same-Origin Policy (SOP)

- Aim to prevent unattended interaction between JavaScript scripts
- **Rule:** JavaScript script in webpage *A* is allowed access data from another webpage *B* if and only if *protocols*, *host names*, and *ports* associated with the documents in question match exactly

Originating doc	Accessed doc	SOP
http://abc.com/a/	http://abc.com/b/	Access OK
http://ab.com/	http://www.abc.com	Host mismatch
http://www1.abc.com/	http://www2.abc.com	Host mismatch
http://abc.com/	https://abc.com/	Protocol mismatch
http://abc.com:81/	http://abc.com/	Port mismatch

HyperText Transfer Protocol (HTTP) (1/8)

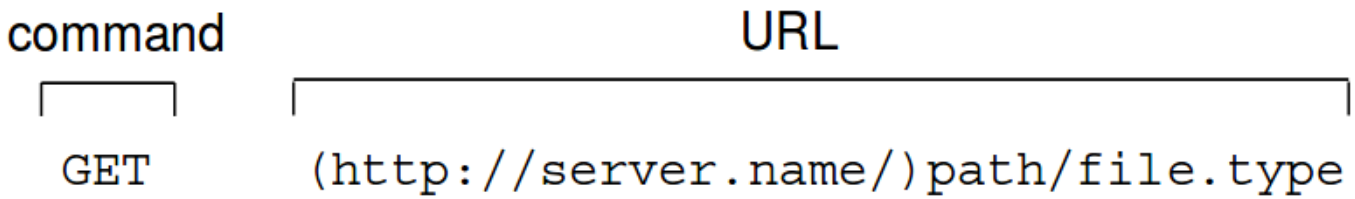
- **Purpose of HTTP**
  - Access to linked documents which are distributed over several computers on the Internet
  - Developed for *exchanging text documents or files* between a large number of geographically distributed humans
  - Standard language for web documents is Hypertext Markup Language (HTML)
  - Simple ASCII-based protocol

HyperText Transfer Protocol (HTTP) (2/8)

- **HTTP Communication Model – simple client/server model**
  - Client (Browser)
    - Starts request for content download
    - Uses *Uniform Resource Locator (URL)* for addressing content, e.g., `http://www.b-tu.de/`
  - Server
    - Locally manages contents (i.e., web pages)
    - Searches contents identified by requested URL
    - Transmits addressed file to client
  - HTTP
    - Syntax for request/response messages
    - Semantics of request/response messages
    - Rules on how to react on a certain message

HyperText Transfer Protocol (HTTP) (3/8)

- **Uses TCP, port 80**
- **Defines allowed request / response messages**

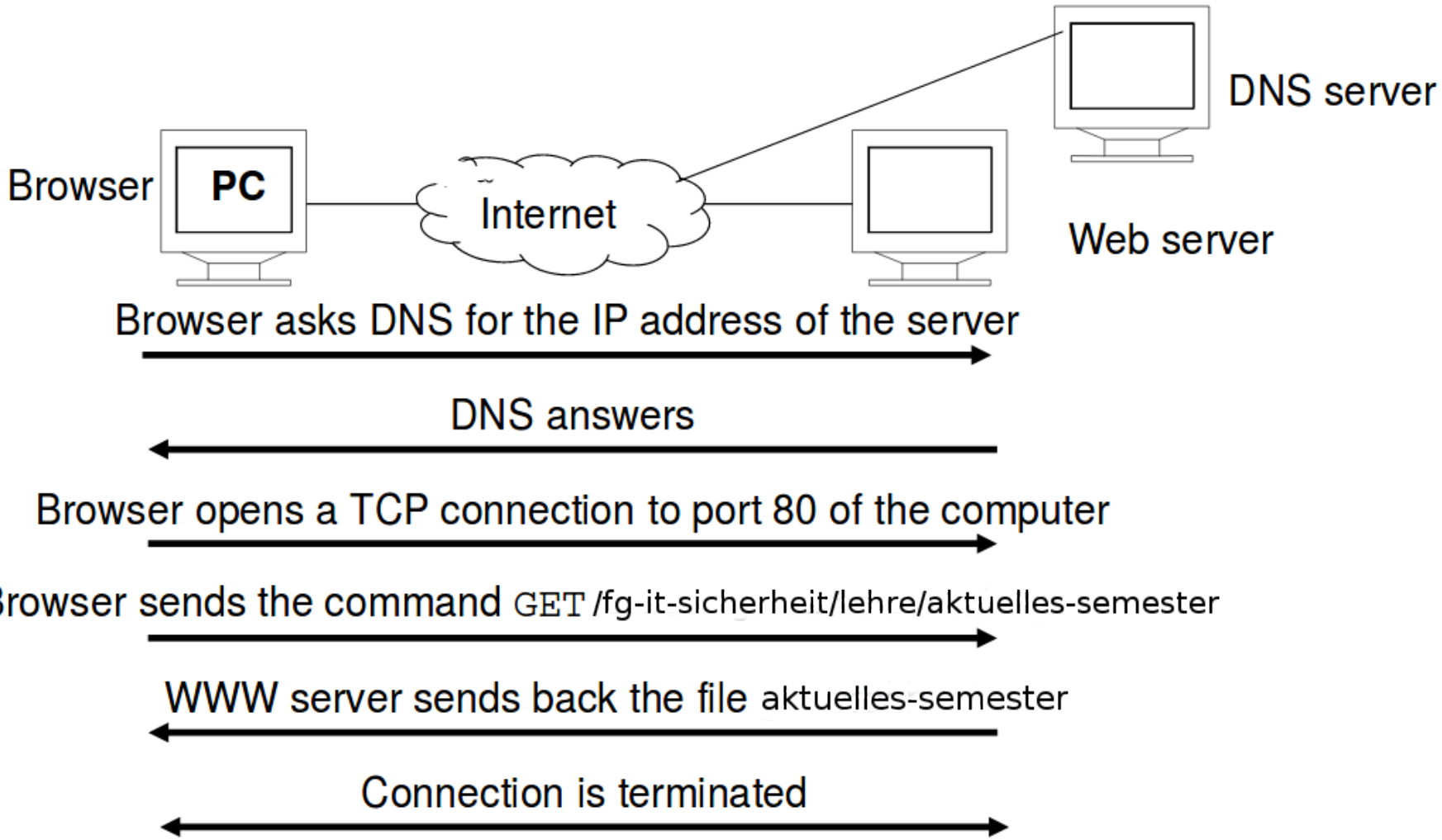


- **Important commands**
  - **GET**: request for contents (file)
  - HEAD: only request header information of a file
  - PUT: store contents on a server
  - **POST**: append content to a web page
  - DELETE: remove contents

HyperText Transfer Protocol (HTTP) (4/8)

- **Request of web pages**
  - Call of

`http://www.b-tu.de/fg-it-sicherheit/lehre/aktuelles-semester`



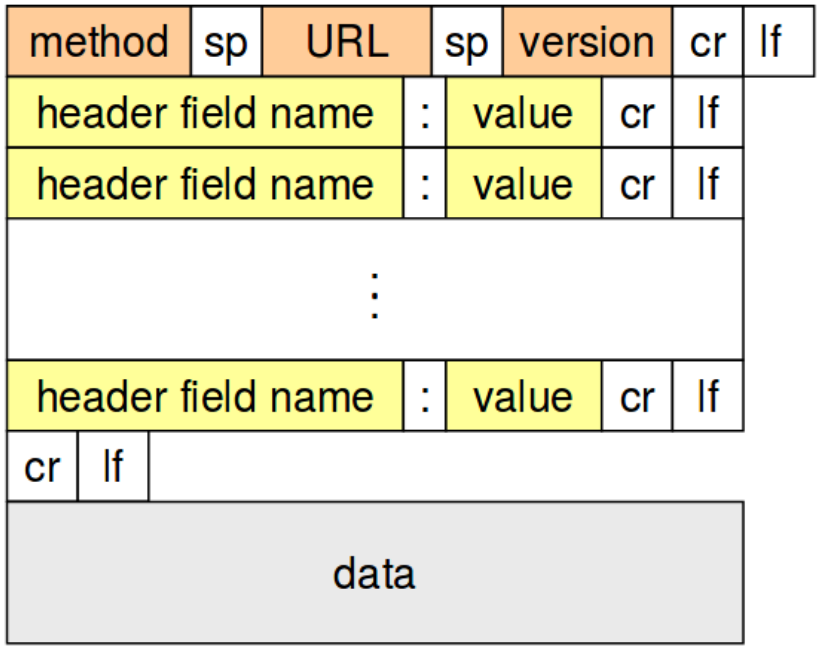
HyperText Transfer Protocol (HTTP) (5/8)

- **Request of web pages**
  - Call of

`http://www.b-tu.de/fg-it-sicherheit/lehre/aktuelles-semester`

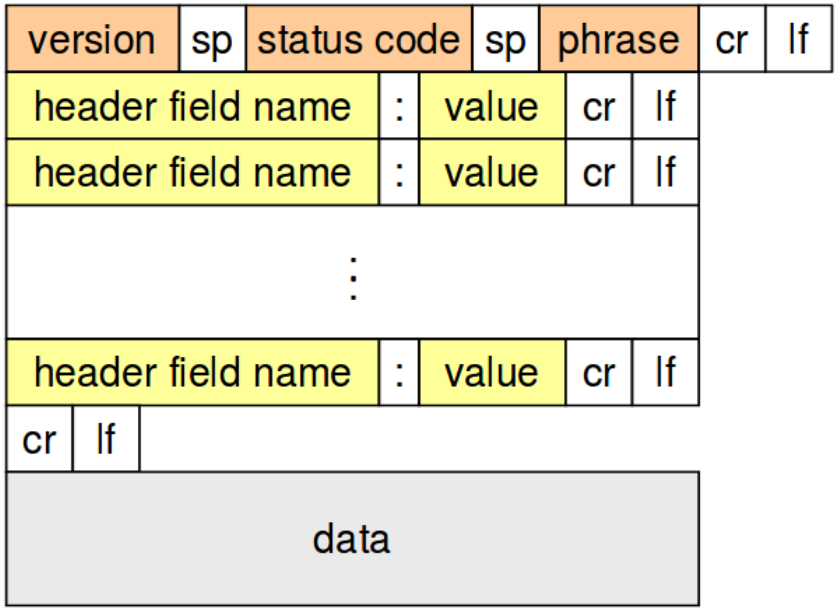
1. Browser determines URL (which was clicked or typed)
2. Browser asks the DNS for the IP address of the server `b-tu.de`
3. DNS answers with 141.43.208.20
4. Browser opens TCP socket to port 80 of computer 141.43.208.20
5. Browser sends command GET `/fg-it-sicherheit/lehre/aktuelles-semester`
6. Server sends back file `aktuelles-semester`
7. After receiving the file, the browser analyses `course-overview` and presents it to the user
8. If the page contains images or other objects, these are included via an URL. They are requested following the same requst/response structure
9. TCP connection is terminated

HyperText Transfer Protocol (HTTP) (6/8)



- **HTTP request message**
  - *Request line:* mandatory part, e.g., GET path/file.type
  - *Header lines:* optional, additional information to the request
  - *cr/lf:* announce end of header
  - *Entity body:* optional; additional information if client submits data to server (e.g., POST command)

HyperText Transfer Protocol (HTTP) (7/8)



- **HTTP response message**
  - *Entity body:* requested data; *Status line:* status code and phrase
  - Groups of status messages
    - 1xx: Only for information
    - 2xx: Successful inquiry, e.g., **200 OK**
    - 3xx: Further activities are necessary
    - 4xx: Client error (syntax), e.g., **400 Bad Request**
    - 5xx: Server error

HyperText Transfer Protocol (HTTP) (8/8)

- **Problem with HTTP: stateless protocol**
  - A HTTP session corresponds to a TCP connection
    - After connection termination the server *forgets* about the request
  - Simple principle, enough for simple browsing
  - *What about online shops? Storing information about related sessions?*

- **Use of cookies:** *new header line* `Set-cookie:..`
  - Instructs client to store the received *cookie* as identifier and fill it in its own header in later requests to that server
  - Thus the server is able to identify related requests
  - Cookie content
    - Name-value pair for identification, defined by the server
    - Optional name-value pairs for, e.g., comments, date, TTL

HTTP Cookies: Security and Privacy Issues

- **Purpose of HTTP cookies**
  - Keep state between different pages
  - Used for access control, once server authenticated the user
- **Problem:** If attacker *knows* your *cookie*, attacker *owns* your *session*
  - *Cookie poisoning*
    - Malicious client tries to elevate its permissions to website
    - Outside attacker tries to impersonate other users
    - Cookies are either modified or stolen from victim clients
  - *Privacy issues*
    - Cookies are defined to be domain-specific
    - Violation of user privacy still possible, e.g., user profiling within domain

Cross-site Scripting (XSS) (1/2)

- *Elevation of privilege* attack that exploits client's trust in server
- Use case scenario
  - Assume: trustworthy website X is given
  - Attacker inserts piece of JavaScript code into a webpage of X
    - E.g., POST message in forum, comment section, etc.
    - Web server does not sanitize users' input
  - Victim user accesses website X and is lured to visit the same webpage
    - Server sends attackers script
    - Not sanitized as printable text but as script
  - Attackers script is run by browser in user's context
  - Attacker can steal cookie/session ID by sending them to its secret server

Cross-site Scripting (XSS) (2/2)

- **Defense against XSS**
  - Change modus operandi: Disable execution of scripts
  - Input validation: Filter client inputs
  - Compare request and response to check if too many requests are mirrored in the response

Cross-Site Request Forgery (XSRF) (1/2)

- Attack that exploits the trust that website has in user's browser
- Use case scenario
  - Assume: trustworthy website X is given
  - User logs into website X (i.e., opens session)
  - Attacker tricks user to browse to his own site
    - Phishing, XSS, social engineering, etc.
  - On the attacker's website, user receives malicious link to be executed in the authenticated context of X
- *Vulnerable websites*
  - Perform action based on input from *trusted* user
  - Do not require any additional authorization for the specific action

Cross-Site Request Forgery (XSRF) (2/2)

- **Defense against XSRF**
  - Temporary secret is shared between the user and the website
  - Website requires proof of knowledge of the secret value before acting on URL
  - Client stores secret in location not accessible to scripts executing in the browser
  - Reliable if secret values cannot be guessed
  - Client sends the secret
    - Inserted as token in the Uniform Resource Identifier (URI) of GET request
    - Inserted in hidden form field of POST request
  - Authentication at layer of web application, i.e., a layer above the browser

SQL Injection

- *Structured Query Language (SQL):* Standard database query language
- SQL commands are placed between single quotes
- If SQL queries consist of query fragments and user input, attacker can change the logic of the query

statement = "SELECT * FROM client WHERE name = '" + userName + "';"

- What, if the attacker enters
  - 'Bob' OR 1 = 1 - -
  - Bob'; drop table client - -
- *Countermeasures:*
  - Input validation: Make sure that no unsafe input is used in the command
  - Change modus operandi: Modify the way commands are constructed and executed

Conclusion

- **Web security:** very broad topic
- **Browser same-origin policy does not provide sufficient protections**
- **Three basic threats considered**
  - Cross-site Scripting
  - Cross-Site Request Forgery
  - SQL Injection