CMPE 209 Network Security

Web Security

Dr. Younghee Park

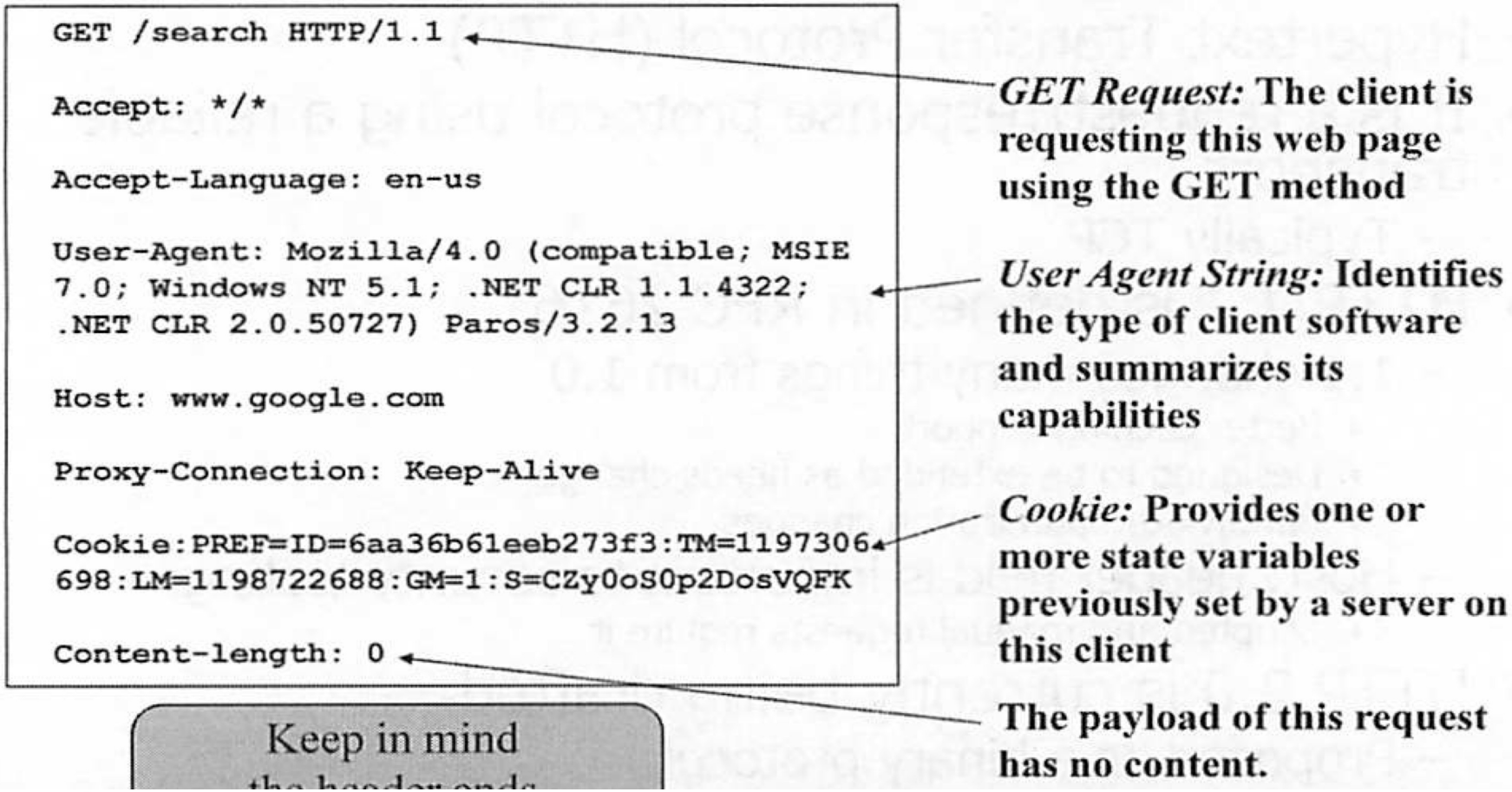
SAN JOSÉ STATE UNIVERSITY

Web Penetration Testing

- Many websites have grown significantly and are more complex
  - Critical business functionality available to employees and possibly even the public across the Internet via portals
  - Fully functional office suites are being offered via your web browser
  - Web-based administration of critical business applications and even security infrastructures
- Now, much large attack surfaces for Web
  - Open Source Vulnerability Database (https://blog.osvdb.org) -> https://vuldb.com/
  - XSS information and archive (http://xssed.com)

HTTP Protocol

- Request/response protocol
- Example HTTP Request:



Web Client (User-Agent)

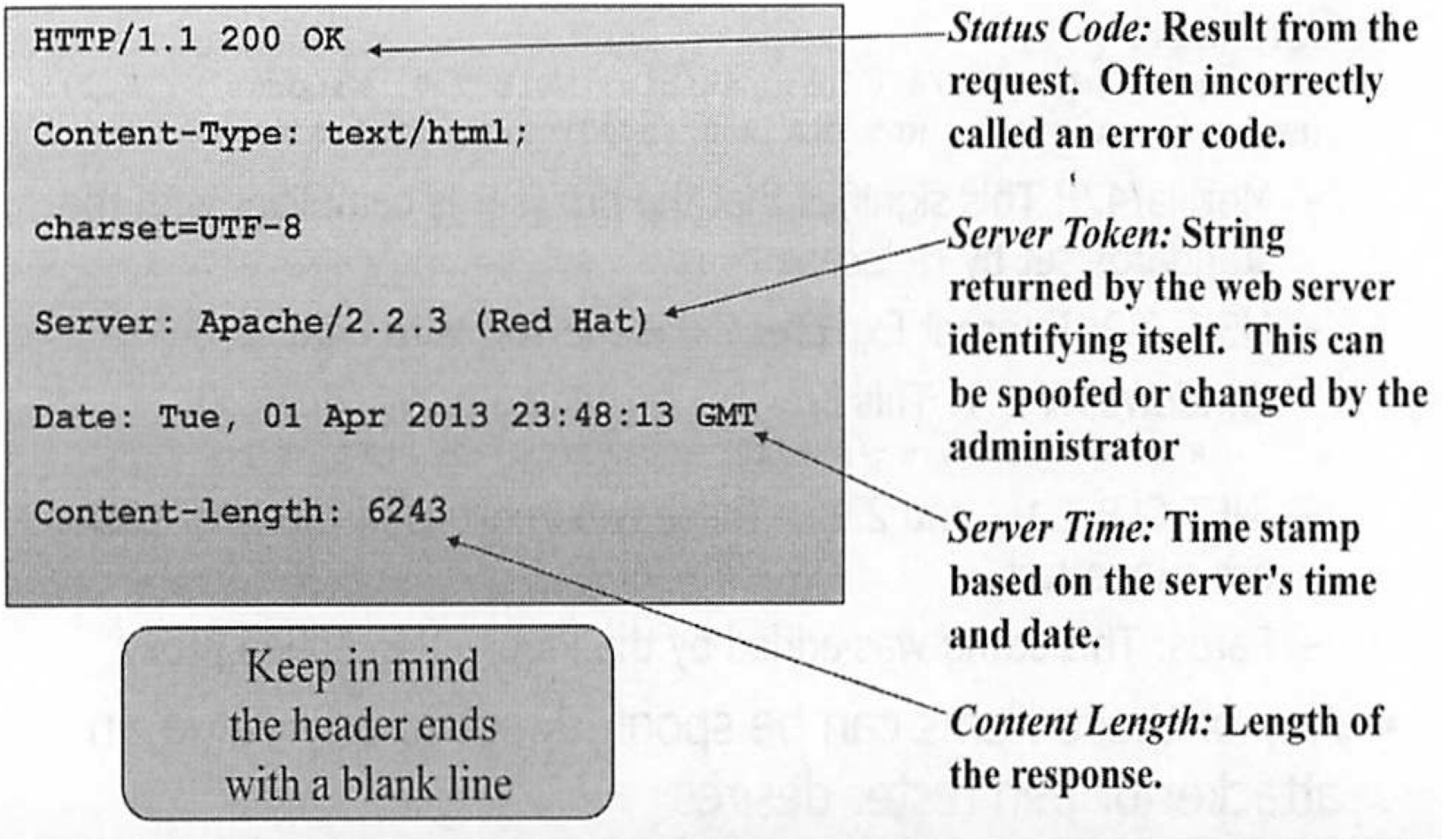
- Typically identified in an HTTP header field

Example) header

```
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1;
.NET CLR 1.1.4322; .NET CLR 2.0.50727) Paros/3.2.13
```

  - Any of these items can be spoofed, set to any value an attacker or pen tester desires
- Web clients can use various request types when accessing a web server
  - GET/POST/HEAD/TRACE/OPTIONS/CONNECT/PUT/DELETE

Example of HTTP Response



Uniform Resources Identifier (URIs)

- A URI is the address of a resource including how to retrieve it
- The term Uniform Resource Locator (URL) is now used interchangeably by most people
- URI contains
  - Protocol://[user:password@]host.domainname[:port]/resource?param=value
  - Consider: http://www.amazon.com/index.php?id=42
    - http is the protocol
    - www.amazon is the host domain name
    - Index.php is the resource
    - Id is a parameter/42 is the value of id

Query String Formats

- Query strings are used to pass data via a URL request
- Popular target for attackers
  - Trivial to manipulate because they are in the browser's location line
- The format of query parameter is determined by the web application developer or the production environment running the application
  - Typical format:
    - index.php?id=42&name=Kevin
  - Other formats:
    - index.php/id/42/name/Kevin
      - Uses Apache's mod_rewrite module
    - index.php/id=42$name=Kevin
      - Data parsed by server side code
    - index.php?param=id:42&param=name:Kevin
      - Data used by an application executed using a system call in the web application

HTTP Status Codes

- Numeric Code to identify the response types
- Five Classes
  - 1XX Informational (100 continue/101 switching protocols)
  - 2XX Success (200 OK)
  - 3XX Redirection (302 Redirect/304 Not Modified)
  - 4XX Client Error (401 Unauthorized / 404 File not found)
  - 5XX Server Error(500 Server Error/502 Bad Gateway)

Stateless HTTP

- HTTP is Stateless
  - The app. must implement a method for grouping a series of requests together in a session.
  - The application implements a state tracking mechanism
- Server-side code has to identify that each request is part of the same session
  - Typically, a session token is passed to and from the client
  - Cookies (It allows the server to identify clients)
    - HTTP header over HTTPS connections
    - **Ex) Set-Cookie: USERID=kevin; expires=Fri, 27-Feb-2016; path =/; domain=.amozon.com**
  - **Reference: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies**

Stateless HTTP (cont'd)

- Server-side code has to identify that each request is part of the same session
  - URI Parameters (Param =value format)
    - Ex) http://www.amazon.com/index.php?sessionid=1234
    - The data in an HTTP GET request
    - Separate parameters with &
  - Hidden Form Fields
    - Forms are used for user input
    - Marked with "hidden" : No display for the user, but its values are passed back to the server as part of the next request.
    - Ex) <input type ="hidden" name="username" value="c_smith">
- As testers, we need to evaluate these tokens.

Types of Flaws

- Information leakage flaws
- Configuration flaws
  - Miscommunication between the app needs and the server is configured
- Bypass flaws
  - Authentication bypass/authorization bypass/file control bypass/front-end bypass
- Injection flaws
  - Ex.)SQL injection

Information Leakage Flaws

- Information leakage flaws allow an attacker to discover information regarding the application
  - Infrastructure information
    - Web server type, back-end database type, operating system type, version numbers of each of these, etc.
  - Path
    - Where are the application components installed on the target machines file system?
  - Codebase : Can we download the application code?
  - Data store: Where the backend data store and what is it?
  - Usernames and/or passwords
    - Obvious targets

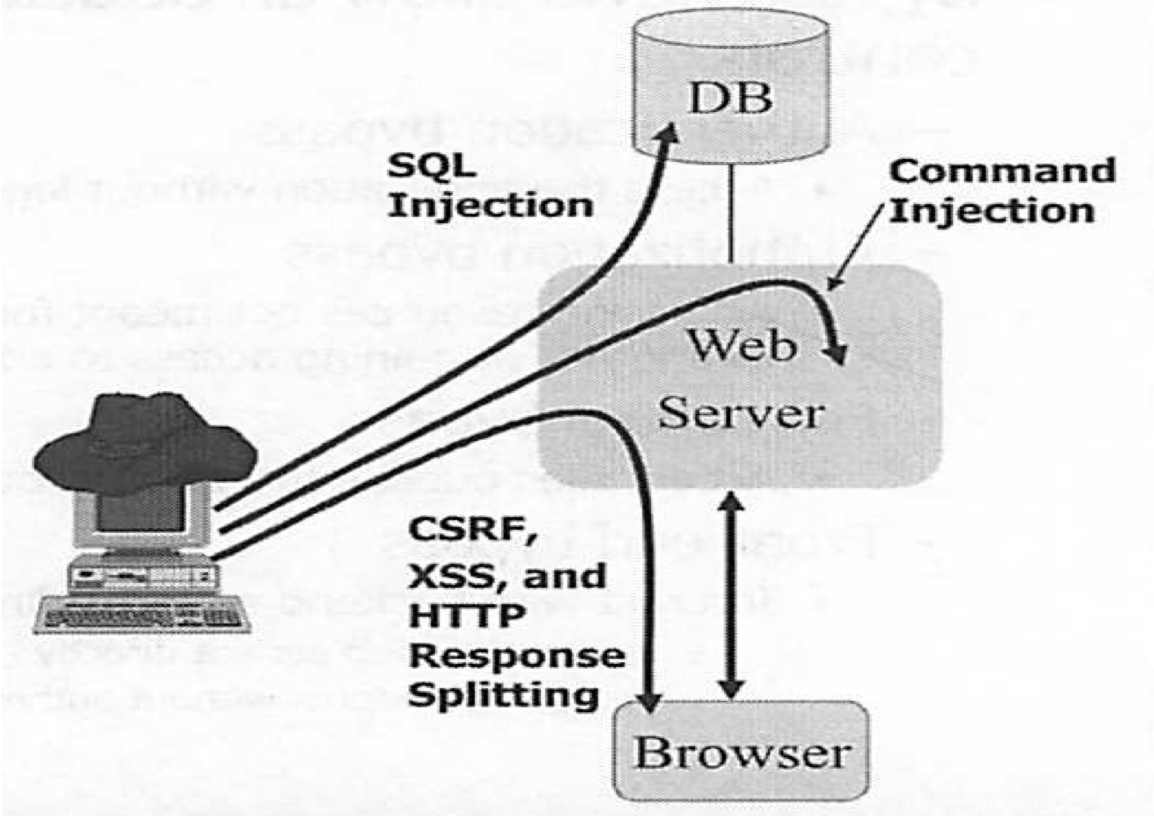
Injection Flaws

- Injection code into some form of user input, with the goal of an interpreter somewhere processing it
- Some examples
  - **SQL injection:**
    - **Target**

    the backend

    data store
  - **XSS :**
    - **Target**

    the clients of an app



JavaScript for Web App Pen Testers

- JavaScript is the most common client-side scripting language today
- Dynamic language for client-side scripting
  - Mainly used in websites, but is also used by other apps like Adobe Reader
- Web app pen testers must therefore understand JavaScript
  - To read and understand scripts from target systems
    - Discover issues with the app and how the app runs on the client
  - To write our own attacks against target systems
    - Change the contents of a page to redirect a form submission
    - Steal cookie to hijack sessions

JavaScript Use in Web Pages

- JavaScript can be inline with HTML
  - As a script tag
    - <script>alert("EE/CMPE 209 Rocks")</script>
  - As part of an HTML item
    - <img src="images/logo.gif" onload="javascript:alert('EE/CMPE 209');">
  - Can also be loaded from another document
    - <script src=http://professional.com/malicious.js>
  - It is common to see any one or more of these in the same HTML pages
- Each statement is a command to the browser

```
if (location.port != 443) {
   alert("No SSL support!")
   }
```

This snippet determines if the web page was accessed on port 443 and pops up an alert dialog box on the browser if it wasn't.

Gaining Access

- Using Applications
- Using Operating System Attacks
- Using Network Attacks

Gaining Access Using Apps and OSs

- Goal: Administrative-access to other systems
- Possible skills
  - Script kiddie
    - Step 1)Surfing for vulnerability exploits from well-known DB
      - Exploit Database: http://exploit-db.com
      - Packet Storm Exploits: http://packetstormsecurity.com/files/tags/exploit/
    - Step 2) Find vulnerability about targets
  - Password attacks
  - Web application attacks
  - Buffer overflow attacks

Web Application Attacks

- Targets: Misconfigured and vulnerable Web Server
- Techniques
  - Account Harvesting
  - Undermining session-tracking mechanisms
  - SQL piggybacking
  - Cross Site Scripting (XSS)

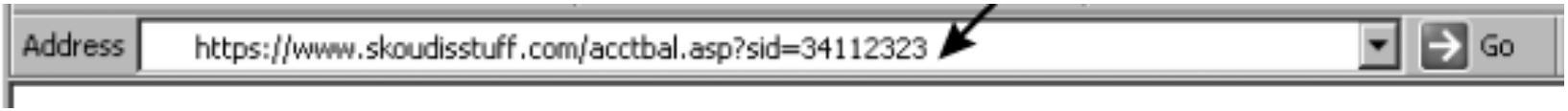
Account Harvesting

- Scenario
  - both usernames and passwords unknown
  - different behavior for wrong usernames or passwords
    - return code : error message

    - different URL displayed in the browser
    - different HTML source file

Session Tracking

- Most web applications create a session to track a user interaction through a series of interactions
  - Session ID is used to maintain state of HTTP connections
  - Session ID implementations
    - Call it as s session token, session credential
    - URL session tracking



    - Hidden form element
      - <INPUT TYPE="HIDDEN" NAME="Session" VALUE="34112323">
    - Cookies
      - Per-session cookie (lives in memory)
      - Permanent cookie (cookies.txt)

Session Token Variables

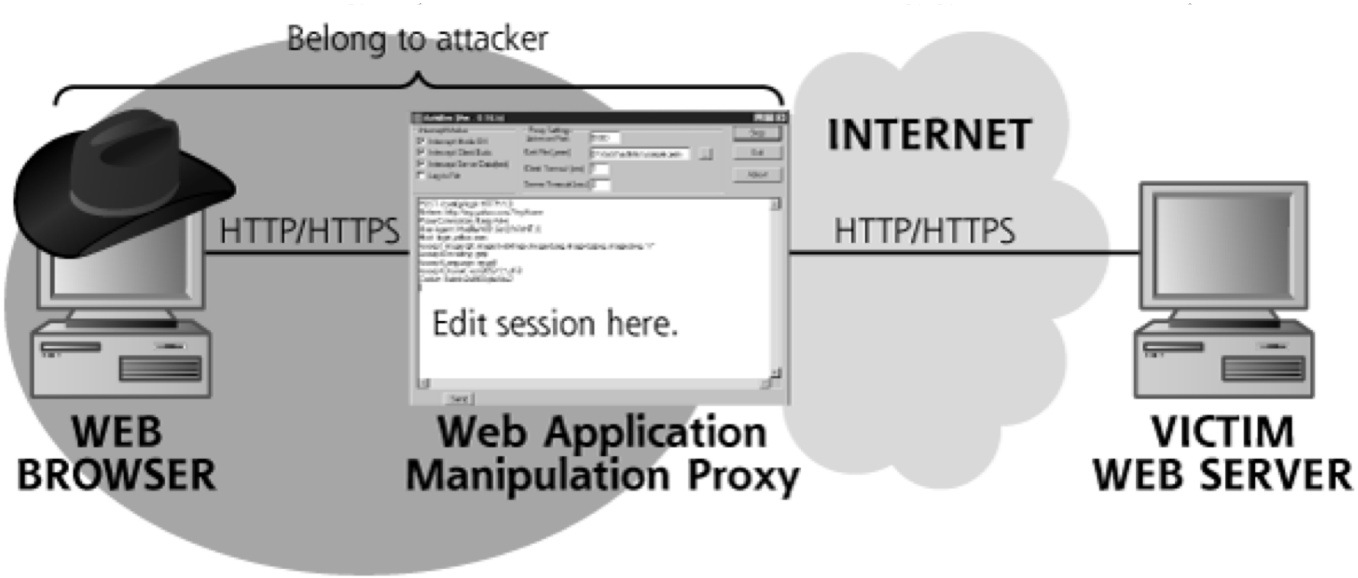
- Applications pass session tokens back to browsers using different mechanisms
  - URL parameters passed via HTTP GET
  - Hidden form elements passed vis HTTP POST
  - Cookies
  - Some application use multiple means…
    - Either on separate pages in the app
    - Or, even on the same page

Cookies

- A cookie is a small piece of information that HTTP server sends to the browser connecting the first time
  - Browser returns a copy of the cookie each time it reconnects
  - HTTP server uses the cookie to create the impression the the "session" spans multiple pages
  - Can store access control and/or authentication information, thus is susceptible to snooping and sniffing
- Defense
  - To implement expiration time and date
  - To include the browser IP address
  - To hash the cookie using MAC (message authentication code)

Undermining Session Tracking

- Session ID vulnerability
  - Attackers may alter the session ID
    - URL session tracking – retype URL (session cloning)
    - Hidden form – change in a local copy of HTML page
    - Permanent session ID – change cookie file
    - Per-session ID - using Achilles proxy



Session Token Predictability

- Session tokens may be predictable
- Consider incremental tokens
  - First login: `74eb2cd93f2a95ba`
  - Next login: `74eb2cd93f2a95bb`
  - Next login: `74eb2cd93f2a95bc`
  - Next login: `74eb2cd93f2a95bf`

  Why the gap? See next bullet!
- Realize that other users may access a production site while you are sampling and you may not get the entire series - significant gaps may appear
- Other predictable assignments:
  - Change by fixed constant (42, 84, 126, 168, etc.)
  - Or, consider:
  - `c4ca4238a0b923820dcc509a6f75849b`,
    `c81e728d9d4c2f636f067f89cc14862c`,
    `eccbc87e4b5ce2fe28308fd9f2a7baf3`,
    `a87ff679a2f3e71d9181a67b7542122c`

    Can you name the algorithm? If not, see notes.
  - Other possible patterns include:
  - IDs based on client IP addresses or data from the session encoded

Defense against Session Tracking Attacks

- If you are developing web applications:
  - Ensure the integrity by hashing the variables (see below)
  - Encrypting information in URL, hidden form element, or cookies, in addition to SSL
  - Generating long enough session IDs to prevent collision
  - Dynamic session IDs, changing from page to page
  - Applying timestamp to session IDs

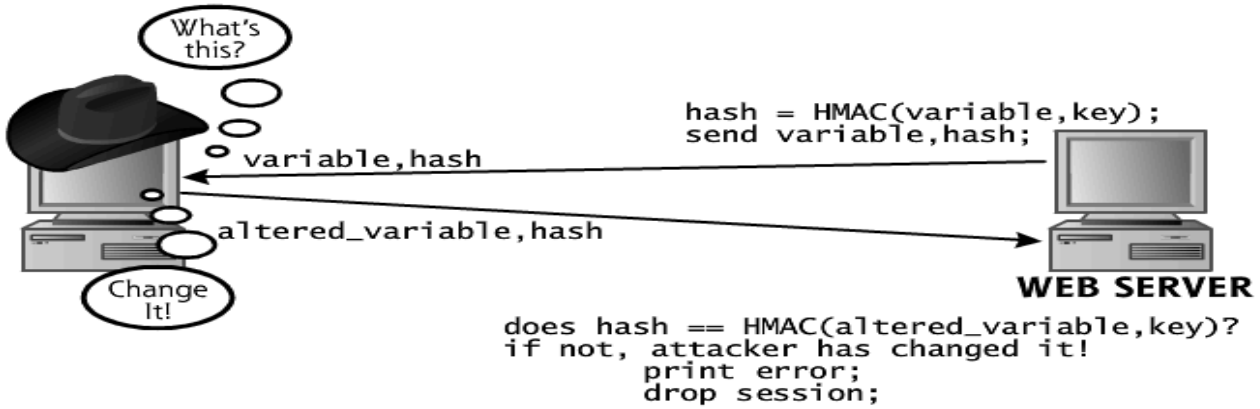
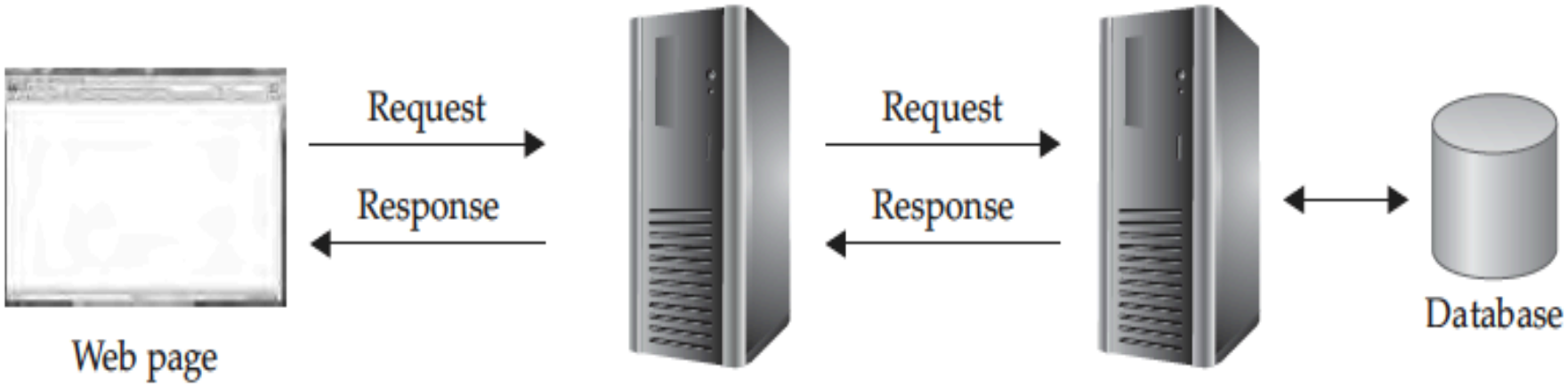


Figure 7.35 Applying an integrity check to a variable passed to a browser using the HMAC algorithm.

Web Application Setup



Request received: Authenticate, authorize, build query, and send to database server

Response: Build web page based off query response; return to browser

Request received: Verify authentication, verify authorization, process query

Response: Return error on results

SQL Injection

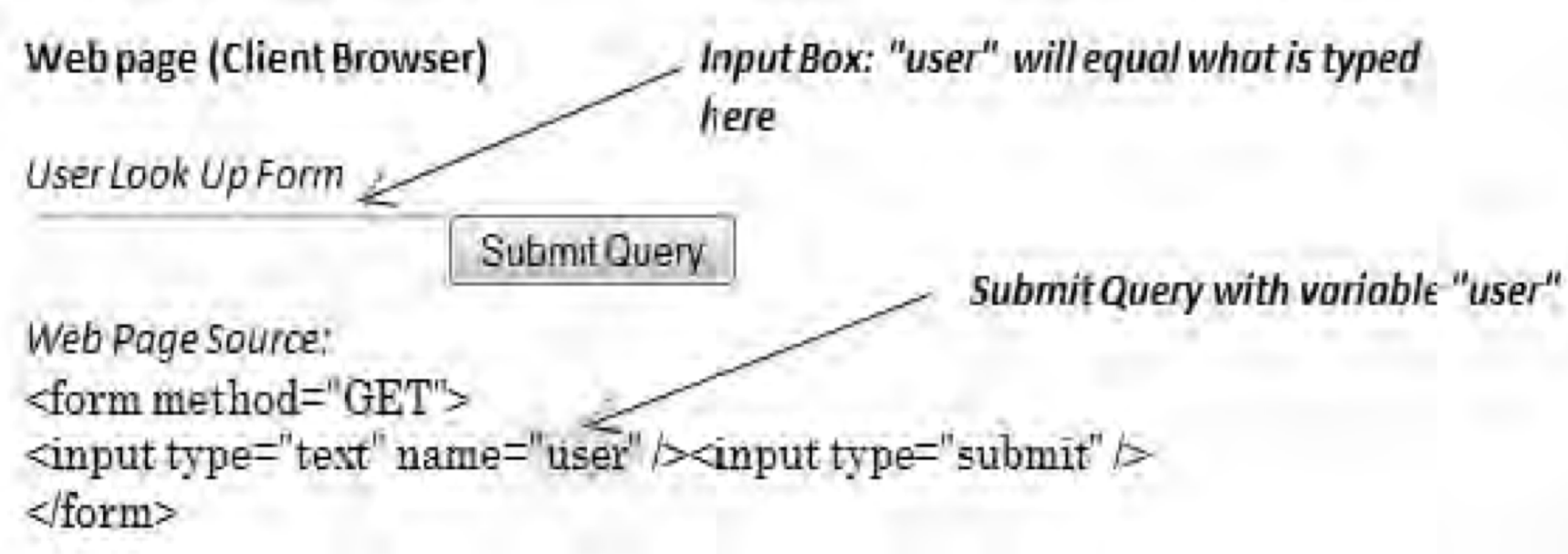
- SQL is structured query language, used to interact with the back-end database
  - SQL injection is a flaw that allows us to control what query is run by the application
  - Requires an understanding of SQL and database structures
- General idea
  - Guessing how input is formulated into SQL query by web applications
  - Append extra information to a SQL query to gain unauthorized access

SQL Injection

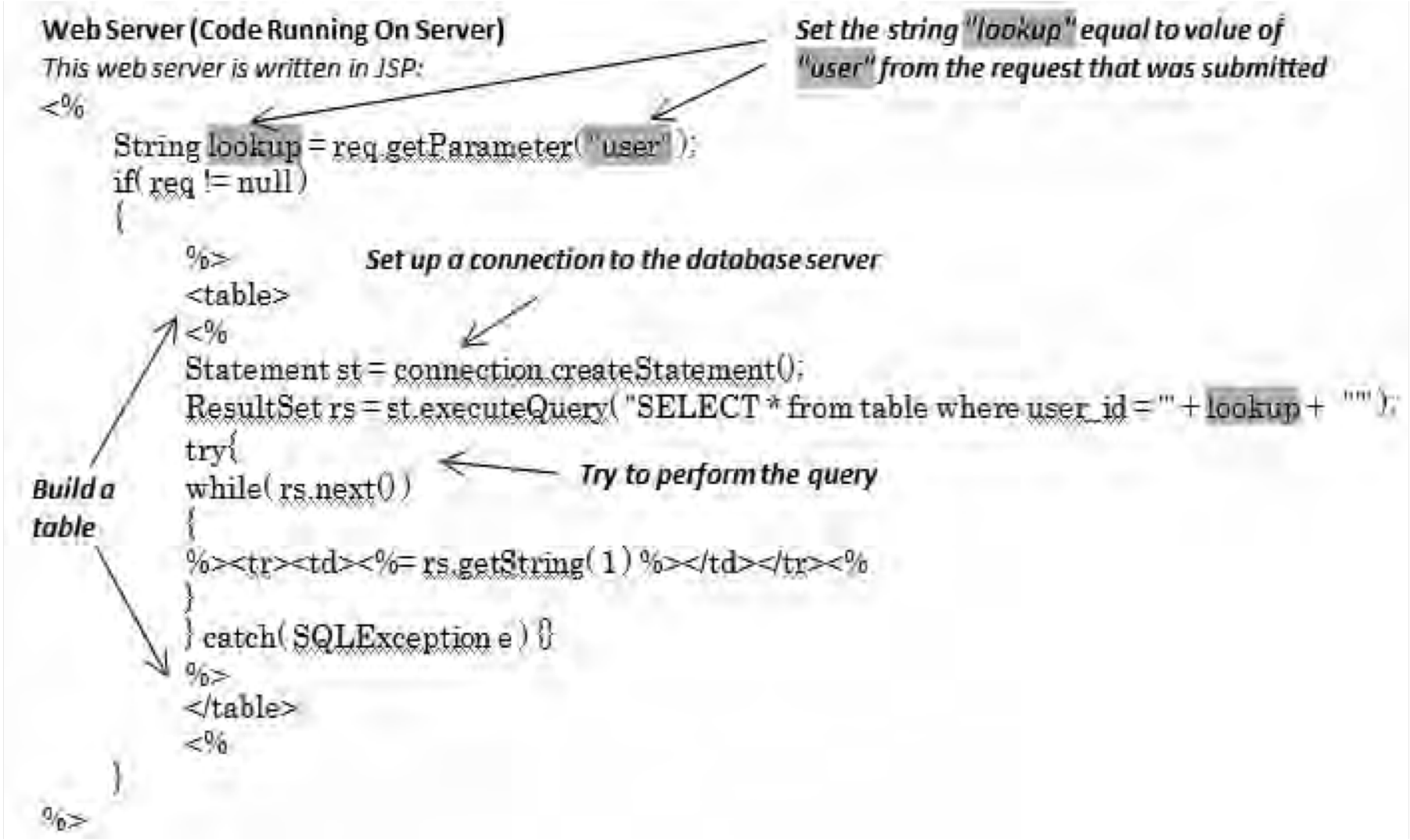
- These types of flaws allows for data disclosure and other attacks against the application
  - We could create user
  - Modify transactions
  - Change records
  - Port scan the internal network
- Causes of SQL Injection
  - No validation when they accepted input
  - User input is then made part of a SQL query
  - Specially formatted input will cause the SQL DB to run attacker's choice of queries

SQL Injection – Simple Web Page

- Tool: Sqlmap
  - "sqlmap" is a python application that perform SQL injection discovery and exploitation.
  - E.g.) ./sqlmap.py –u full-URL - -readfile=/etc/passwd
- Accepting User Input

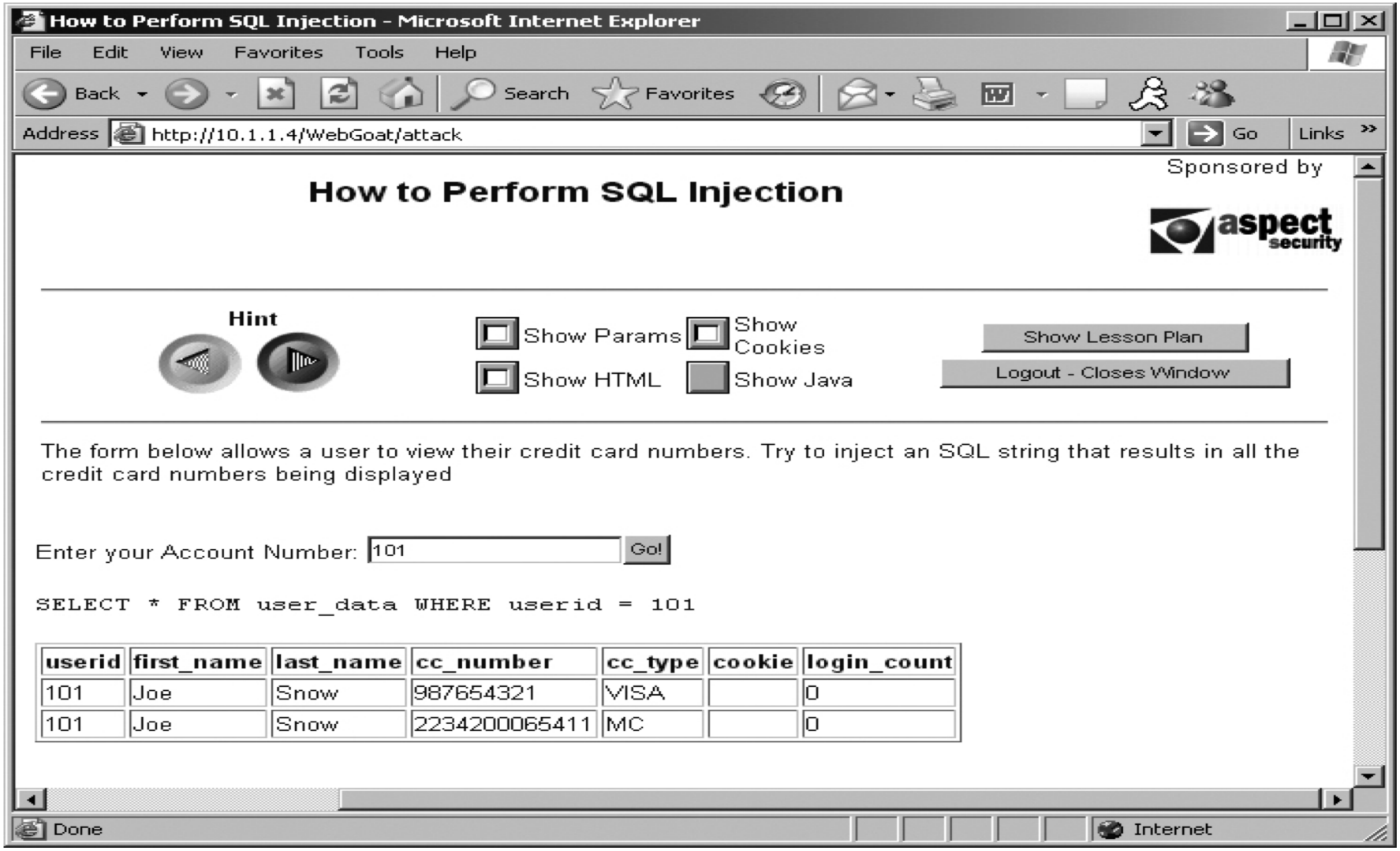


SQL Injection – Server Code

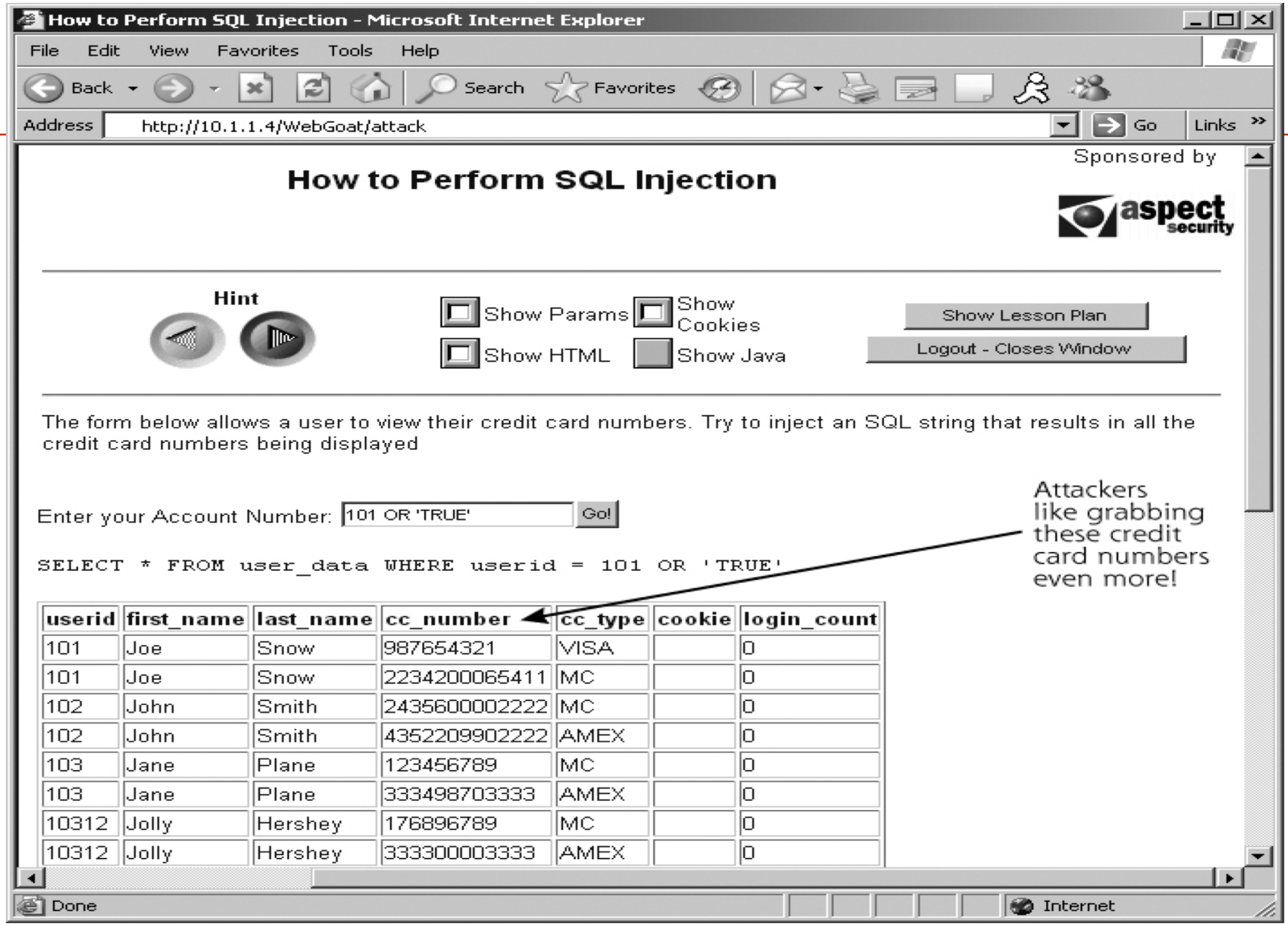



```
<%
    String lookup = req.getParameter( "user" );
    if( req != null )
    {
        %>
        <table>
        <%
        Statement st = connection.createStatement();
        ResultSet rs = st.executeQuery( "SELECT * from table where user_id = '" + lookup + "'" );
        try{
        while( rs.next() )
        {
        %><tr><td><%= rs.getString( 1 ) %></td></tr><%
        }
        } catch( SQLException e ) {}
        %>
        </table>
        <%
    }
%>
```

Entering your account number



Example: A valid query input



Example: A "sweet" query!

Defending against SQL Injection

- Rationale: never trust user input
- Carefully validate and filter user input
  - Quotes (‘, “, `)
  - Semicolons (;) – query terminators
  - Asterisk (*) – wildcard selector
  - Shell metacharacters (& \|*?....
- Not safe to filter input on the browser side

Cross Site Scripting (XSS)

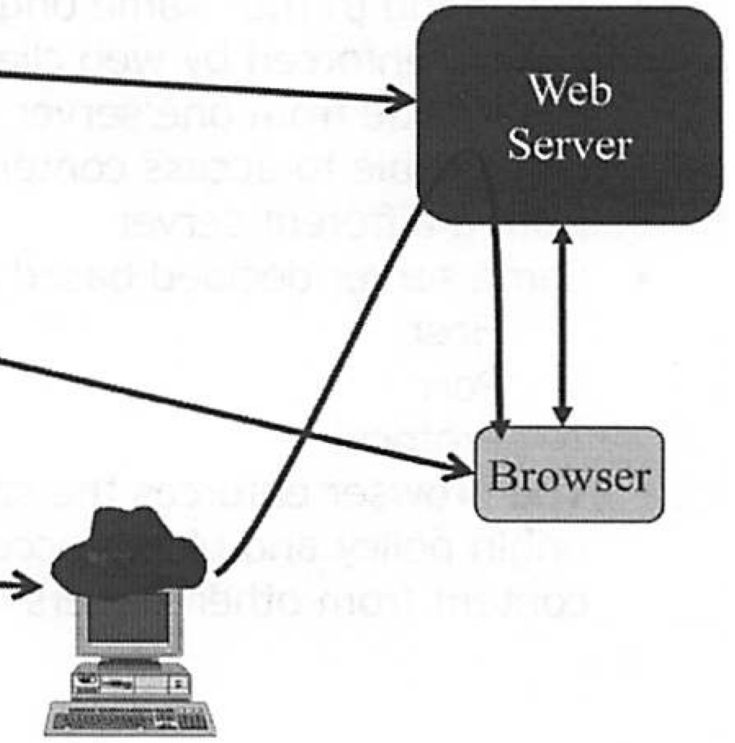
- Most major websites use script to perform calculations, page formatting and other client-side actions.
- The script typically runs on the client side, not the server side. Here is a sample.

```
<html>
<head> </head>
<body>
    <script type="text/javascript">
    document.write("A script was used to display this text");
    </script>
</body>
</html>
```

- https://en.wikipedia.org/wiki/Cross-site_scripting#cite_note-17

Parts of a XSS Attack

- Application
  - Running a vulnerable application
- Browser
  - Tricked into Running the code
- Attacker
  - Evil Person
- Code
  - Usually JavaScript



XSS Attack

- XSS injects scripts into a web application
  - XSS involves tricking the browser into execution code
  - The browser believes that the code is part of the site and runs it in that context
- **XSS attacks the browser(client), not the server**
  - The goal is to get the client (browser) to execute the script
- XSS can be used to
  - Hijack sessions
  - Gain access to restricted content stored by a website
  - Execute commands on the target
  - Record keystrokes

XSS Attack (2)

- XSS attacks web applications that do not perform input sanitization
- Example Java Script:

  <script> alert("XSS Test") </script>
- Enter the attack script on the Username input box and hit Submit

FIGURE 6.10
Example of input boxe

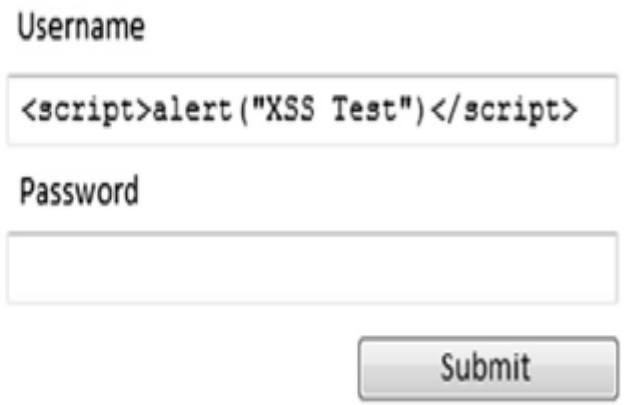


FIGURE 6.11
XSS test code.

Summary: Types of XSS

- Three Types of XSS flaws
  - Reflected (Non-persistent)
    - Easiest to test
    - Place script in URL
    - E.g. Reflected XSS payloads in the request -> send the link to the victim via email or other delivery method.
  - Persistent
    - Requires attacker to input script
    - Then view resulting pages
      - E.g. message boards or account settings
  - DOM-based
    - Arbitrary parameters used by client-side code

XSS Videos

Two examples of XSS

- Reflected XSS
- Store XSS (Persistent XSS)

Video Demo: Cross Site Scripting (Reflected XSS) Demo - YouTube

https://www.youtube.com/watch?v=V79Dp7i4LRM

XSS explained

reference: https://www.ibm.com/developerworks/library/wa-secxss/

Defending Against XSS

- For users:
  - disabling scripting when it is not required
  - Don't trust links sent by emails or in message boards.
    - They may contain scripts.
  - accessing any sensitive sites (banking, e-commerce) carefully while checking Logo
    - It is not good to type its address directly
    - Do not following through any 3rd-party links
- For programmers
  - validating inputs before sending back to the client!
- Security on the web is based on a variety of mechanisms, including an underlying concept of trust known as the same-origin policy.

Other Information

- Security level should be “low” during exercise
- XSS
  - https://www.youtube.com/watch?v=bC10UV8cWUw
- Blind SQL

  https://www.youtube.com/watch?v=gwTthztskAg

Cross-Site Request Forgery

- Cross-Site Request Forgery (**CSRF**) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated
- **CSRF** attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request.

Cross-Site Request Forgery