OWASP Top Ten Security Threats

Threats

- Web defacement

  ⇒ loss of reputation (clients, shareholders)

- Information disclosure (lost data confidentiality)

  e.g. business secrets, financial information, client database, medical data, government documents

- data loss (or lost data integrity)

- unauthorized access

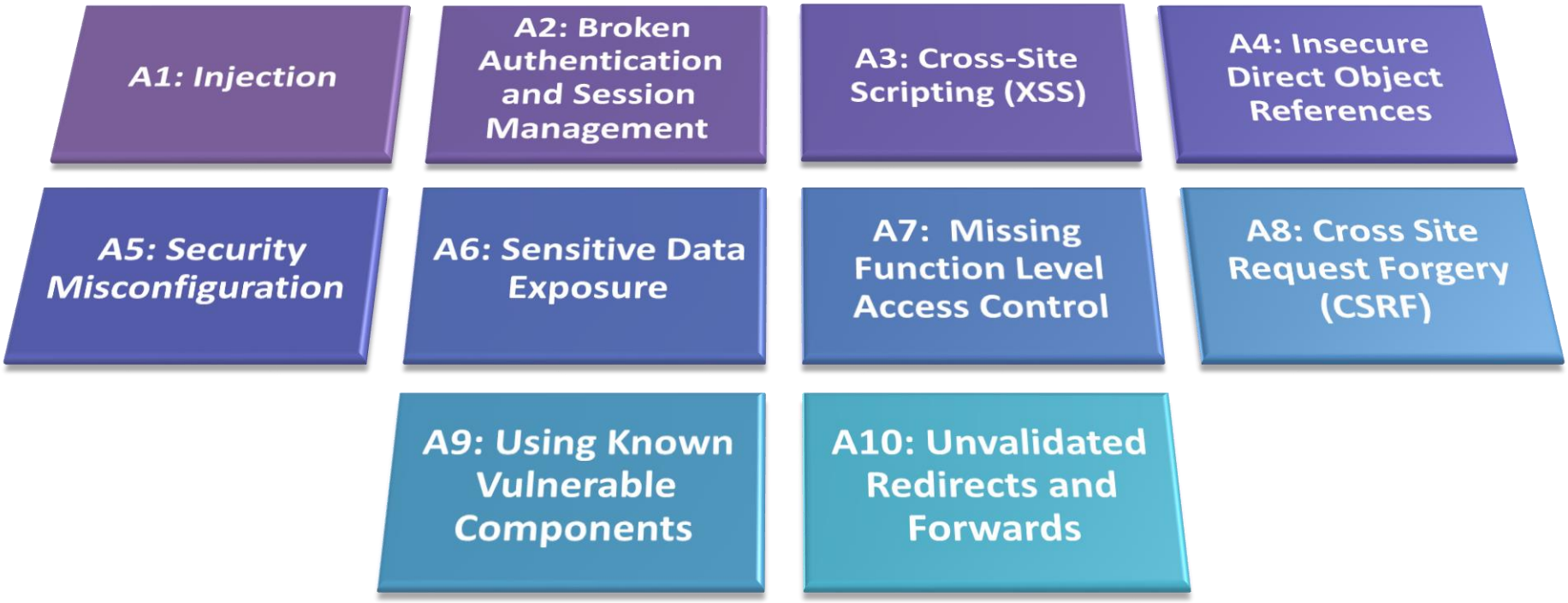
  ⇒ functionality of the application abused

- denial of service

  ⇒ loss of availability or functionality (and revenue)

OWASP Top Ten (2013 Edition)
(Open Web Application Security Project)

http://owasp.org/index.php/Category:OWASP_Top_Ten_Project



A1: Injection flaws

- Executing code provided (injected) by attacker
  - SQL injection

Server Code

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

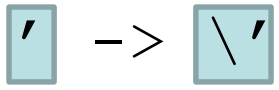
UserId:

105 or 1=1

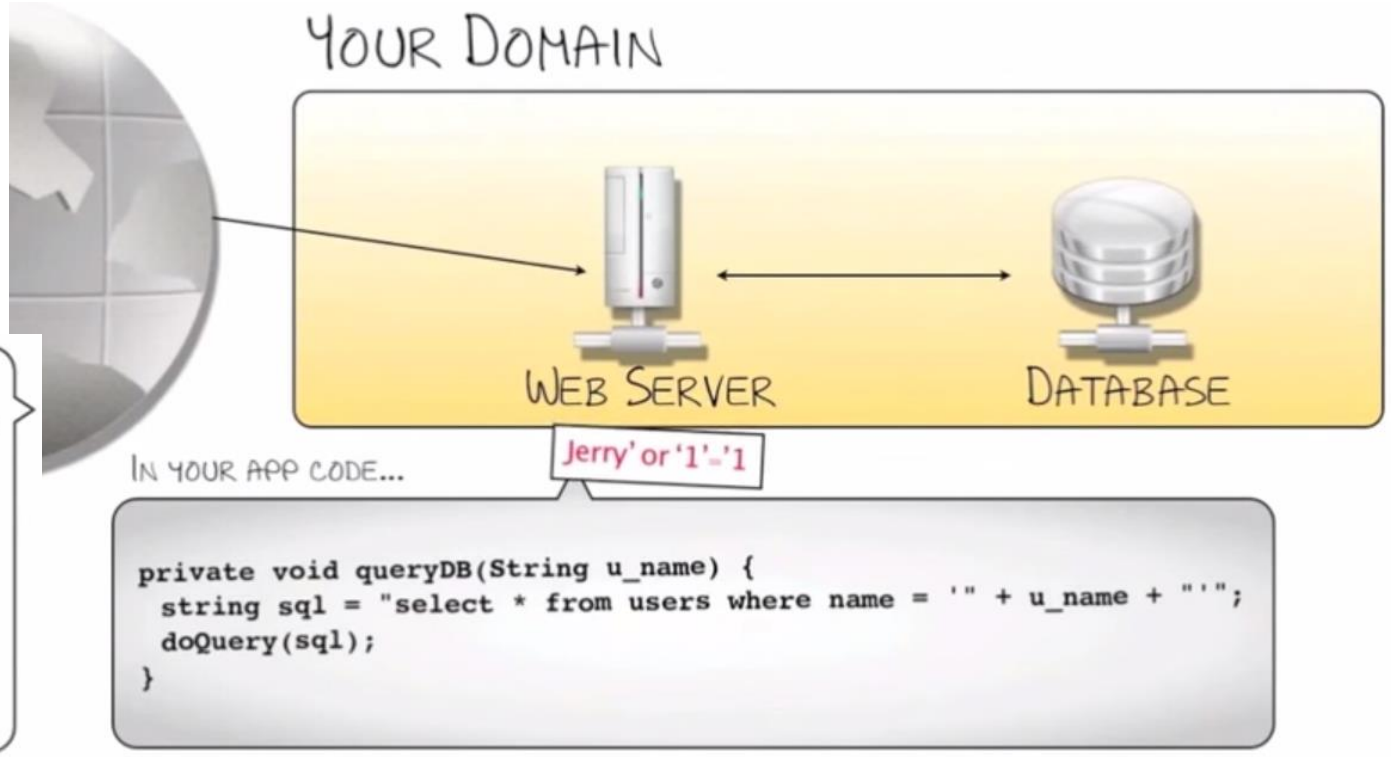
Server Result

```
SELECT * FROM Users WHERE UserId = 105 or 1=1
```

- Solutions:
  - validate user input
  - escape values (use escape functions)
  - use parameterized queries (SQL)
  - enforce least privilege when accessing a DB, OS etc.




Leave a Comment
http://www.appgoat.com
Google
Please leave a comment
Name:
Country:
Hong Kong
Comments:
OUR USERS WILL ENTER THEIR NAME HERE


YOUR DOMAIN
WEB SERVER
DATABASE
IN YOUR APP CODE...
Jerry' or '1'='1
private void queryDB(String u_name) {
string sql = "select * from users where name = '" + u_name + "'";
doQuery(sql);
}
RETURNS ROWS FOR JERRY OR WHEREVER ONE EQUALS ONE
(ALL ROWS)

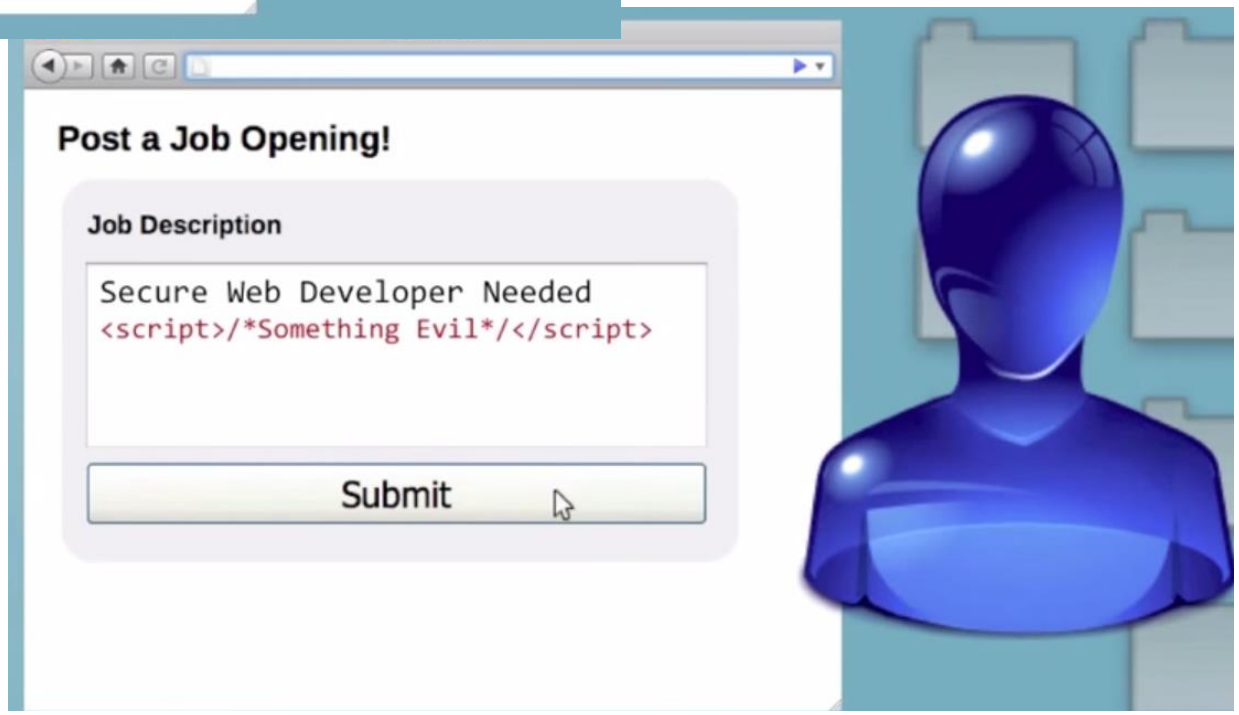
A2: Broken Authentication & session Management

- Session hijacking by stealing session id (e.g., using eavesdropping if not https)
- Solutions:
  - generate new session ID on login (do not reuse old ones)
  - use cookies for storing session id
  - set session timeout and provide logout possibility
  - require https (at least for the login / password transfer)

A3: Cross-site scripting (XSS)

- Cross-site scripting (XSS) vulnerability
  - an application takes user input and sends it to a Web browser without validation or encoding
  - attacker can execute JavaScript code in the victim's browser
  - to hijack user sessions, deface web sites etc.
- Solution: validate user input, encode HTML output


Mozilla Firefox
New Job Posting!
Static Content
Job Description
Secure Web Developers Needed!
User Supplied Content
Apply Now
Save Job


Post a Job Opening!
Job Description
Secure Web Developer Needed
<script>/*Something Evil*/</script>
Submit

Visitors will get the evil script


Mozilla Firefox
New Job Posting!
Job Description
Secure web develo
lol!
Apply Now
ATTACKERS CAN USE JAVASCRIPT TO....
STEAL YOUR SESSION ID:
document.cookie
REWRITE ANY PART OF THE PAGE


https://www.yourbanksite.com
YourBankSite
Please Log in
User Name:
Password:
LOGIN
ATTACKERS CAN USE JAVASCRIPT TO....
OVERLAY THE LOGIN SCREEN WITH THEIR OWN, ALLOWING ATTACKS TO HARVEST USERNAMES AND PASSWORDS

A4: Insecure Direct Object Reference

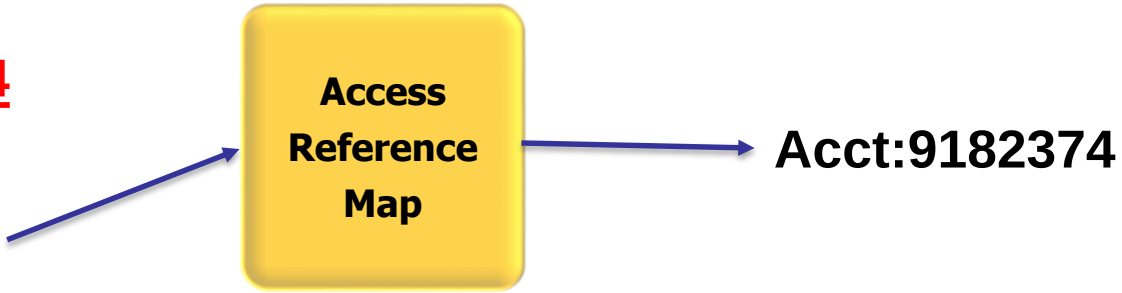
- Attacker manipulates the URL or form values to get unauthorized access
  - to objects (data in a database, objects in memory etc.):
    `http://shop.com/cart?id=`**`413246`** (your cart)
    `http://shop.com/cart?id=`**`123456`** (someone else's cart ?)
  - to files:
    `http://s.ch/?page=`**`home`** **-> home**
    `http://s.ch/?page=`**`/etc/passwd`** **-> /etc/passwd**
- Solution:
  - avoid exposing IDs, keys, filenames to users if possible
  - validate input, accept only correct values
  - verify authorization to all accessed objects (files, data etc.)

A4 – Avoiding Insecure Direct Object References

- Eliminate the direct object reference
  - Replace them with a temporary mapping value such as a random mapping

Instead of :
http://app?id=9182374

Use:
http://app?id=7d3J93



- Validate the direct object reference
  - Verify the user is allowed to access the target object
  - Verify the requested mode of access is allowed to the target object (e.g., read, write, delete)

5 – Security Misconfiguration

Web applications rely on a secure foundation

- **Everywhere from the OS up through the App Server**
- **Hackers can take advantage of poor server configuration to gain unauthorized access to application functionality or data**

Solution

- Verify your system's configuration by scanning to find misconfiguration or missing patches
- Secure configuration by "hardening" the servers:
  - Disable unnecessary packages, accounts, processes & services
  - patch OS, Web server, and Web applications
  - run Web server as a regular (non-privileged) user

6 – Sensitive Data Exposure

Storing and transmitting sensitive data insecurely

- Failure to identify all sensitive data
- Failure to identify all the places that this sensitive data gets stored
  - Databases, files, directories, log files, backups, etc.
- Failure to identify all the places that this sensitive data is sent
  - On the web, to backend databases, to business partners, internal communications
- Failure to properly protect this data in every location

Typical Impact

- Attackers access or modify confidential or private information
  - e.g, credit cards, health care records, financial data (yours or your customers)
- Attackers extract secrets to use in additional attacks
- Company embarrassment, customer dissatisfaction, and loss of trust
- Expense of cleaning up the incident, such as forensics, sending apology letters, reissuing thousands of credit cards, providing identity theft insurance
- Business gets sued and/or fined

Sensitive Data Exposure – Example

1

Victim enters credit card number in form

Log files

2

Error handler logs credit card details because merchant gateway is unavailable

3

Logs are accessible to all members of IT staff for debugging purposes

4

Malicious insider steals credit card numbers

A7: Missing Function Level Access Control

- “Hidden” URLs that don’t require further authorization
  - to actions:

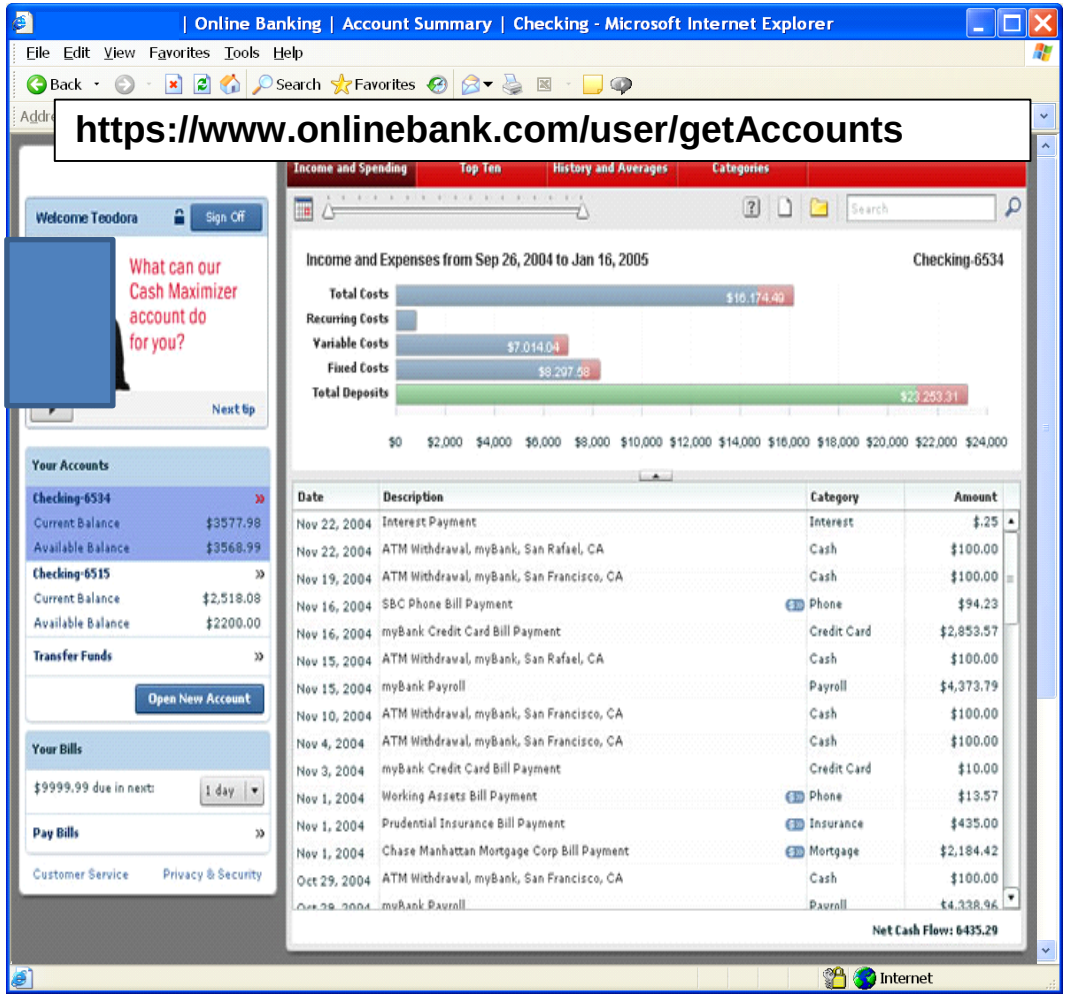
    `http://site.com/`**`admin/adduser?name=x&pwd=x`**

    (even if `http://site.com/`**`admin/`** requires authorization)
  - to files:

    `http://site.com/`**`internal/salaries.xls`**

    `http://me.com/`**`No/One/Will/Guess/82534/me.jpg`**
- Problem: missing authorization
- Solution
  - add missing authorization ☺
  - don‘t rely on security by obscurity – it will not work!

Missing Function Level Access Control Illustrated



- Attacker notices the URL indicates his role

  /user/getAccounts

- He modifies it to another directory (role)

  /admin/getAccounts, or

  /manager/getAccounts

- Attacker views more accounts than just their own

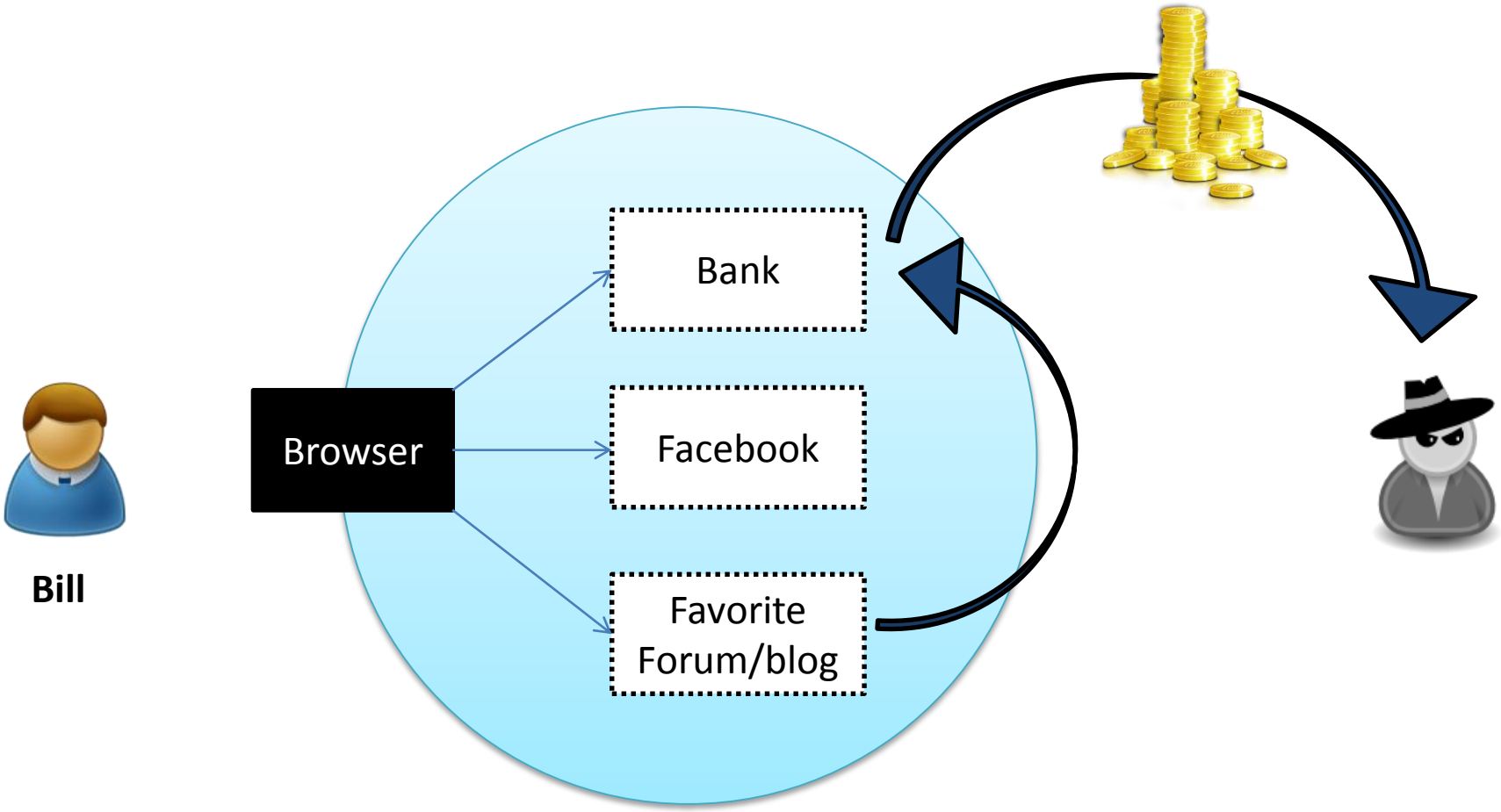
A8: Cross-site request forgery

- Cross-site request forgery (CSRF) – a scenario
  - Ali logs in at bank.com, and forgets to log out
  - Ali then visits a evil.com (or just webforums.com), with:

```
<img src="http://bank.com/
    transfer?amount=1000000&to_account=123456789">
```

  - Ali's browser wants to display the image, so sends a request to bank.com, without Ali's consent
  - if Ali is still logged in, then bank.com accepts the request and performs the action, transparently for Ali (!)
- There is no simple solution, but the following can help:
  - expire early user sessions, encourage users to log out
  - use secret hidden fields
  - use POST rather than GET, and check referrer value
  - Re-authenticate or CAPTCHA for extra-sensitive pages

CSRF Example



http://mybank.com/showaccount?id=bill

http://mybank.com/transfer?from=bill&amount=10000&for=someguy

****

9 – Using Known Vulnerable Components

Vulnerable Components Are Common

- Some vulnerable components (e.g., framework libraries) can be identified and exploited with automated tools

Widespread

- Virtually every application has these issues because most development teams don't focus on ensuring their components/libraries are up to date

Typical Impact

- Full range of weaknesses is possible, including injection, broken access control, XSS ...
- The impact could range from minimal to complete host takeover and data compromise

10 – Unvalidated Redirects and Forwards

Web application redirects are very common

- Sometimes parameters define the destination URL
- If they aren't validated, attacker can send victim to a site of their choice
- If not validated, attacker may be able to use unvalidated forward to bypass authentication or authorization checks

Typical Impact

- Redirect victim to phishing or malware site
- Attacker's request is forwarded past security checks, allowing unauthorized function or data access

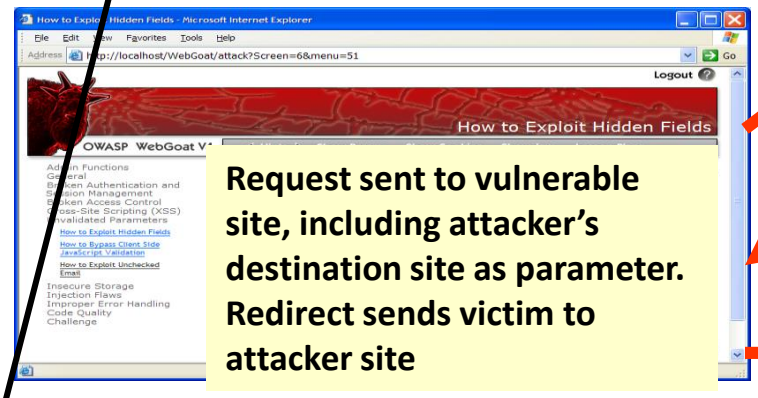
Unvalidated Redirect Illustrated

1 Attacker sends attack to victim via email or webpage

From: Internal Revenue Service
Subject: Your Unclaimed Tax Refund
Our records show you have an unclaimed federal tax refund. Please click here to initiate your claim.

3 Application redirects victim to attacker's site

2 Victim clicks link containing unvalidated parameter



Request sent to vulnerable site, including attacker's destination site as parameter. Redirect sends victim to attacker site

Evil Site

http://www.irs.gov/taxrefund/claim.jsp?year=2006 & … &dest=www.evilsite.com

4 Evil site installs malware on victim, or phish's for private information

Client-server – no trust

- Security on the client side doesn't work (and cannot)
  - don't rely on the client to perform security checks (validation etc.)
  - e.g. `<input type="text" maxlength="20">` is not enough
  - authentication should be done on the server side, not by the client
- Don't trust your client
  - HTTP response header fields like referrer, cookies etc.
  - HTTP query string values (from hidden fields or explicit links)
  - e.g. `<input type="hidden" name="price" value="299">` in an online shop can (and will!) be abused
- Do all security-related checks on the server
- Don't expect your clients to send you SQL queries, shell commands etc. to execute – it's not your code anymore
- Put limits on the number of connections, set timeouts

Summary

- understand threats and typical attacks
- validate, validate, validate (!)
- do not trust the client
- read and follow recommendations for your platform
- use web scanning tools
- harden the Web server and programming platform configuration