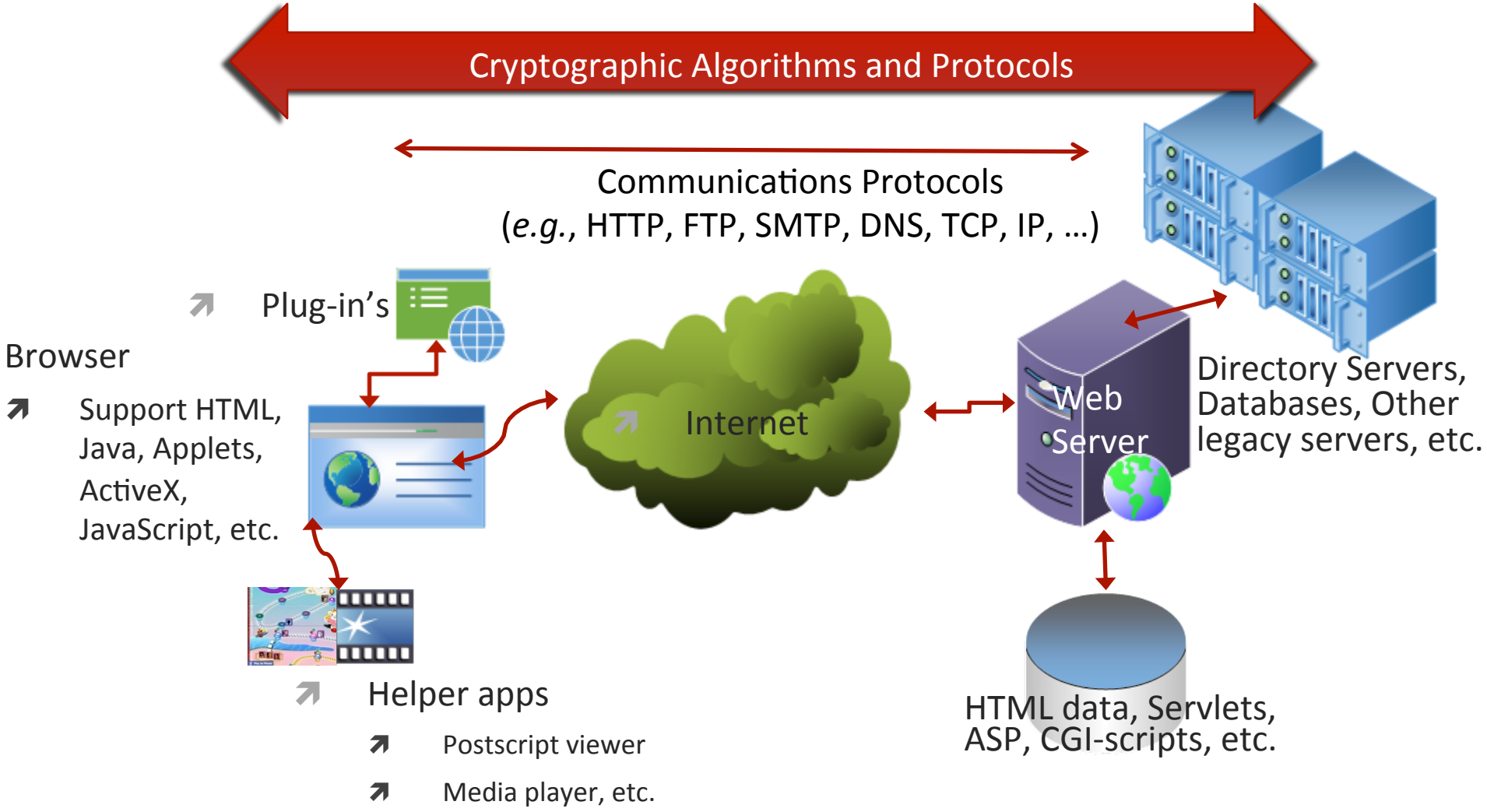
Introduction to Cyber Security

Fall 2017 | Sherman Chow | CUHK IERG 4130

Chapter 4
Web Security

Many Facets of Cyber Security



Open Web Application Security Project '10

- **A1 Injection**
- **A2 Cross-Site Scripting (XSS)**
- **A3 Broken Authentication and Session Management**
- A4 Insecure Direct Object References
- **A5 Cross-Site Request Forgery (CSRF)**
- A6 Security Misconfiguration
- A7 Insecure Cryptographic Storage
- A8 Failure to Restrict URL Access
- A9 Insufficient Transport Layer Protection
- A10 Unvalidated Redirects and Forwards

https://www.owasp.org/index.php/Top_10_2010-Main

Open Web Application Security Project '13

- A1 Injection
- A2 Broken Authentication and Session Management
- A3 Cross-Site Scripting (XSS)
- A4 Insecure Direct Object References
- A5 Security Misconfiguration
- A6 Sensitive Data Exposure
- A7 Missing Function Level Access Control
- A8 Cross-Site Request Forgery (CSRF)
- A9 Using Components with Known Vulnerabilities
- A10 Unvalidated Redirects and Forwards

Source: https://www.owasp.org/index.php/Top10#OWASP_Top_10_for_2013

Changes from ’10 to ’13

- Sensitive Data Exposure = Insecure Cryptographic Storage + Insufficient Transport Layer Protection
- “Failure to Restrict URL Access” in 2010 is broadened into “Missing Function Level Access Control”
- “Using Components with Known Vulnerabilities” was buried in “Security Misconfiguration”

Explanations

- Insecure Direct Object Reference: A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

- Security Misconfiguration: Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. Secure settings should be defined, implemented, and maintained, as defaults are often insecure. Additionally, software should be kept up to date.

- Sensitive Data Exposure: Many web applications do not properly protect sensitive data, such as credit cards, tax IDs, and authentication credentials. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data deserves extra protection such as encryption at rest or in transit, as well as special precautions when exchanged with the browser.

- Missing Function Level Access Control: Most web applications verify function level access rights before making that functionality visible in the UI. However, applications need to perform the same access control checks on the server when each function is accessed. If requests are not verified, attackers will be able to forge requests in order to access functionality without proper authorization.

- Unvalidated Redirects and Forwards: Web applications frequently redirect and forward users to other pages and websites, and use untrusted data to determine the destination pages. Without proper validation, attackers can redirect victims to phishing or malware sites, or use forwards to access unauthorized pages.

Web Security (Server-Side)

- How to supply data to a web-server?
  - URL parameter
  - HTTP packet
- Browser can constrain what is in an URL and prepare a well-formed packet.
  - But an attacker can prepare a tailor-made HTTP packet anyway
- The web-server should always be available
  - “Traditional” f/w’s are not effective in defending such attacks (except web application proxies and web application firewall)

GET and POST Methods

- Get: via the URL, name-value pairs in the query string
  - *e.g.*, http://www.lmgtfy.com/?q=lmgtfy
- Post: both the query string and the HTTP packet message body
  - In HTML (hypertext markup language) <form method="post" …>
  - In the browser, the form elements (text field, radio buttons)

Sequences of Operations

- User types in the login name and password
- The browser composes an HTTP packet containing the login name and password via the POST method
  - Why not use GET method?
- An application (*e.g.*, ASP page) processes (login, password) pair
- That application may need to talk to another server (*e.g.*, DB server) for checking if the (login, password) pair is valid

Input Validation

- Let's start with URL as an input
- A web page is stored in a hard-disk of the web server anyways
  - *e.g.*, C:\InetPub\wwwroot for Microsoft's IIS server
  - *i.e.*, when you access www.server.com/index.html, the IIS server actually returns C:\inetpub\wwwroot\index.html to you
- Will http://www.sillywebserver.com/../../ bring you to C:\ ?
  - ".." is the "command" for going up one level of directory
  - "." denotes the current directory
- Expect the unexpected
- Always do input validation and do not allow attackers to insert commands

Common Attacks

- 1. Exploit encoding weakness
  - Having an URL containing cmd.exe or \ looks suspicious?
    - OK, the server may just remove every "\" from the URL
  - Attacker can *circumvent* by hex. or unicode conventions, *e.g.*, %c1%9c
    - Normal usage: Consider Chinese domain name or accent like "é" in café
  - Remember etc/passwd?
  - Download ../../../etc/passwd file by encoding ".." and "//" using hex. or unicode conventions
- 2. Buffer Overflows in Web Technologies, MS IIS, Apache Server
  - Using URL parameter-passing capability to pass the exploit code, a.k.a. "egg" to the server through the URL
  - We also have "omelet" in buffer overflow (details omitted).

IIS Extended Unicode Directory Traversal

#	URL
0	/msadc/..%255c../..%255c../..%255c../winnt/system32/cmd.exe?/c+dir+c:\
1	/msadc/..%25%35%63../..%25%35%63../..%25%35%63../winnt/system32/cmd.exe?/c+dir+c:\
2	/msadc/..%255c..%255c..%255c..%255cwinnt/system32/cmd.exe?/c+dir+c:\
3	/msadc/..%25%35%63..%25%35%63..%25%35%63..%25%35%63winnt/system32/cmd.exe?/c+dir+c:\
4	/scripts/..%255c..%255cwinnt/system32/cmd.exe?/c+dir+c:\
5	/scripts/..%252f..%252f..%252f..%252fwinnt/system32/cmd.exe?/c+dir+c:\
6	/scripts/..%255c..%255cwinnt/system32/cmd.exe?/c+dir+c:\
7	/msadc/..%255c../..%255c../..%255c../winnt/system32/cmd.exe?/c+dir+c:\
8	/msadc/..%%35c../..%%35c../..%%35c../winnt/system32/cmd.exe?/c+dir+c:\
9	/msadc/..%%35%63../..%%35%63../..%%35%63../winnt/system32/cmd.exe?/c+dir+c:\
…	(70 of them listed at http://www.sans.org/security-resources/idfaq/iis_unicode.php)

(Web application running) SQL Statements

- SQL, or sometimes Structured Query Language, is a language for relational database management system (DBMS)
- A web application (*e.g.*, ASP) can work with a DBMS via SQL
- http://m.buynow.com/scripts/order.asp?id=2
  - SQL below appears somewhere in the ASP code (web app. hosted at IIS):
  - SELECT * from ORDER where ID = $id;
  - DB server checks against each record (ID = 2, ID = 10, etc...)
  - if ID = 2, *i.e.*, what's after "where" evaluated to "true", return that record
  - if ID is not 2, *i.e.*, what's after "where" evaluated to "false", that record is not returned

SQL Injection / SQL Code Poisoning

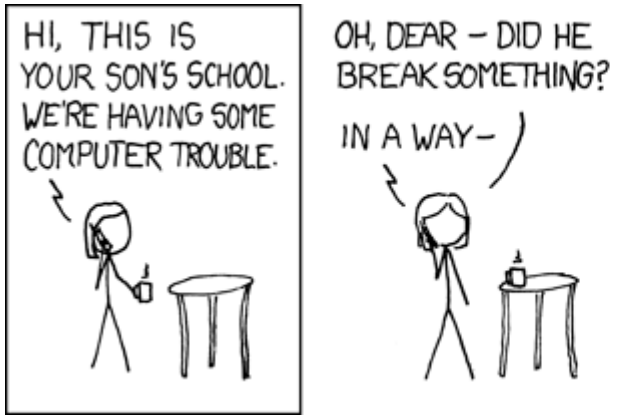
- http://m.buynow.com/scripts/order.asp?id=4%20OR%201=1
  - SELECT * from ORDER where ID = $id OR 1=1;
  - if ID = 4 or 1 = 1, return that record
  - if ID is not 4 or 1 =1, return that record
  - Return all records!
- Note that this example assumes ID is a number.
- For general text, SQL statements quote it with single quotes ['].

Blind SQL Injection

- Perhaps the web program just returns a single record
  - instead of looping all records in the result set
  - *e.g.*, the "first" (according to some order) record is returned
  - In this case, the attacker is somewhat "blind"
- But, different response from the "oracle" can return "1-bit" of information
  - SELECT * FROM db WHERE ID = '5' AND '1'='1'; (1 record)
  - SELECT * FROM db WHERE ID = '5' AND '1'='2'; (error)
  - Attacker can distinguish between these 2 cases
- Embed "ID=5 AND substring(@@version,1,1)=4" to check if this MySQL server is of version 4
- or, Second Order SQL Injection

What else the attacker can do?

- SELECT * FROM USERDB where ACCT = $name and PWD = $pwd
- DROP TABLE USERDB
- EXEC … cmd.exe
- EXEC … tftp.exe
- EXEC … nc.exe



https://xkcd.com/327 (Exploit of a Mon)

tftp.exe, a handy hacking tool

- http://www.file.net/process/tftp.exe.html
- Trivial File Transfer Protocol App
- Located in the folder C:\Windows\System32
- Known file sizes on Windows 7/XP are 17408/16896 bytes resp.
- The program has no visible window.
- Now you've uploaded a file to the victim server, so what?

What else you can do?

- Netcat (nc.exe) is a simple networking utility which reads and writes data across network connections.
- You can then use that machine with SQL server running for…
- Creating (and connecting) to a (reverse) shell (*e.g.*, bash)
- Port scanning (so your own machine is not a scanner)
  - Scanning for what kind of services / vulnerabilities are available

A Recent Incident

- On Jun 27, 2013, hacker group "RedHack" breached Istanbul Administration Site and claimed that they can erase people's debts to utilities fee
  - They did not use “DROP Table”
- “RedHack has published a username and a password to allow others to access the Istanbul Special Provincial Administration’s systems.”
  - http://news.softpedia.com/news/RedHack-Breaches-Istanbul-Administration-Site-Hackers-Claim-to-Have-Erased-Debts-364000.shtml
- User: ' or ''=' Pass: ' or ''='
  - (those are two single quotes, not a double quote)
  - https://twitter.com/RedHack_EN/status/350461821456613376

Escaping Characters

- Single quote character ['] has special meaning in SQL
- Escape character: type it twice
  - ['] ['] to denote a single quote
- Do you know all possibilities of escape characters?
- How about cases like O'Neill as a last name?
  - [O] ['] ['] [N] [e] [i] [l] [l]
- Or, use pattern checking

More Mitigations

- Prepared/Parameterized statements
  - compiled
  - parameter needs not to be escaped since they are transferred using a different protocol
- Enforcement at the coding level
- Database permissions
  - *e.g.* Why a web application has the privilege to remove a whole table from the DB?

Session Management

- Last time we talked about password and access privilege
- Do you want to type in the password at every single page?
  - Session managed by the server-side
  - Cookie stored at the *client-side*
- Common: Logged in but forgot to logout
  - "*The main thing is not to ask the user to do too many things, preferably none.*"
- Exploit Session Management Weakness
  - The insecure use of cookies, session identifier, etc.

Scenario (from OWASP Report)

- Application's timeouts aren't set properly.
- User uses a public computer to access a site.
- Instead of selecting "logout" the user simply closes the browser (tab) and walks away.
- Attacker uses the same browser an hour later, and that browser is still authenticated.

Cookies

- enables the server to send items of data to the client
- which the client stores and resubmits back to the server
- sent in the HTTP packet with the line *Cookie:…*
- Cookies continue to be resubmitted in each subsequent request without any particular action required by the application / user
  - unlike the other types of request parameters
    (those within the URL query string or the HTTP message body)

Attributes of a Cookie

- expires: a date until which the cookie is valid
  - This will cause the browser to save the cookie to persistent storage, and
  - it will be reused in subsequent browser sessions until the expiration date is reached.
- domain: the domain for which the cookie is valid
  - This must be the same or a parent of the domain from which the cookie is received.
- path: the URL path for which the cookie is valid.
- secure: If set, the cookie will only ever be submitted in HTTPS Requests.
- HttpOnly: If set, the cookie can't be directly accessed via client-side JavaScript (can only be accessed by the server which received HTTP requests)
  - Requires the browser to support it

Careful Session Management

- 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
- Session Fixation: the server knows whom to serve, and has already prepared
- 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

Another Scenario

- Airline reservations application supports URL rewriting, putting session IDs in the URL:
  - http://example.com/sale/saleitems;jsessionid=2P0OC2JSNDLPSKHCJUN2JV?dest=Hawaii
- An authenticated user of the site wants to let his friends know about the sale.
- He e-mails the above link without knowing he is also giving away his session ID.
- When his friends use the link they will use his session and credit card.

General Countermeasures

- Perform security-oriented code-review for server's codes, scripts, servlets
  - Independent review, penetration tests
- Pro-actively scan for known vulnerabilities (*e.g.*, Nessus, Nitko, Whisker, etc.)
- Keep up with Vendor Patch, Patch and Patch …
- Beware of latest vulnerabilities (BugTraq)
- Backup your system
- Have an incident handling and disaster recovery procedure
- Load-balancer, server-redundancy: esp. against DDoS attacks

Some Specific Countermeasures

- Install all Web content on separate volume, not system disk
- Set Access control lists (ACLs) on the file system
  - *e.g.* cmd.exe to SYSTEM and Admins only
- Do not use Plaintext-based protocols to manage your server
  - *e.g.*, telnet, rlogin, ftp, …;
  - use the secure version instead: ssh (terminal access and ftp)
- The principle behind can be applied elsewhere

JavaScript

- Scripting language embedded in HTML Webpages
  - usually surrounded by <SCRIPT> tags
  - to be downloaded to the client browsers
- JavaScript code is (traditionally) interpreted directly by the 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, dynamic visual elements, *e.g.*, moving banners, status lines
- It is difficult to filter JavaScript out of webpages
  - it is possible to send HTML to web browser that appears to be free of JavaScript but that actually contains JavaScript programs
  - functionality vs. security (more functions, more possible vulnerabilities)

JavaScript Security

- In this course, we restrict ourselves to client-side JavaScript
  - *e.g.*, not Node.js
- There are no JavaScript methods that can directly access the files 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”
- This is “by design”, but what is the reality?

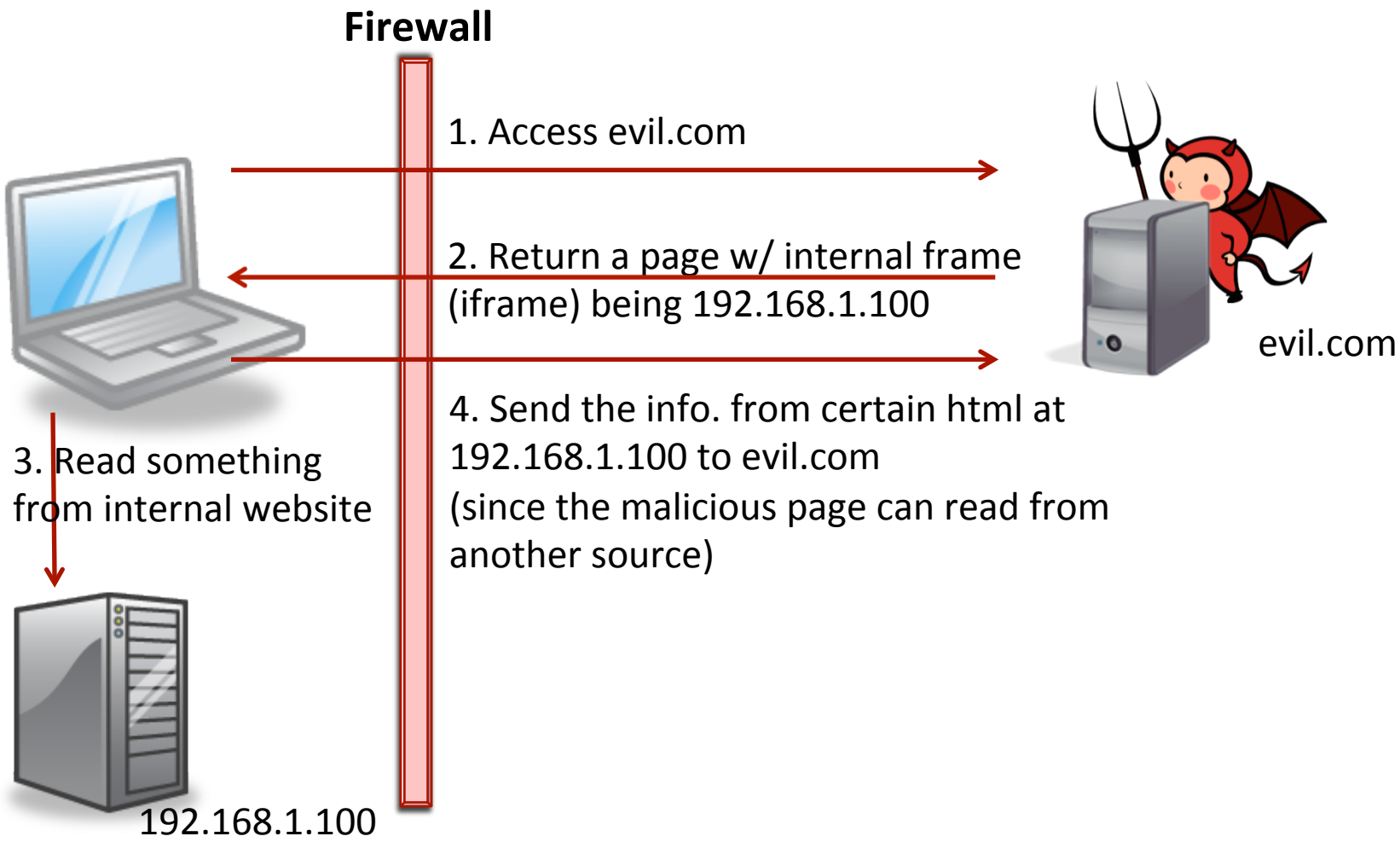
JavaScript (Implementation) Insecurity

- Potentially has access to any information that the browser has, *e.g.*, history list
- Could be used to create forms that automatically submitted themselves by email
  - forge email in the name of the user, harvest email address for spammers, etc.
- "Popup" windows / dialog-boxes of arbitrary text without your permission
  - can be exploited to trick browser user to enter important info, password…
- Lock up your browser so that it is unusable (DoS?)
- Can register a function that can be called when the current one is unloaded
  - *e.g.*, if the Back button is hit or if the window is closed, pop-up 2 new windows!
- The "Same-Origin Policy" can be circumvented

Same Origin Policy (SOP)

- a security concept for browser-side programming languages
- the policy permits scripts running on pages originating from the *same origin* to access each other's methods and properties with no specific restrictions
  - full network access, read/write DOM object, storage, etc.
  - Document Object Model (DOM) represents objects in HTML, XML, etc.
  - Write: send content, *e.g.*, navigate to a URL, manipulate a frame's property
- but prevents access to most methods and properties across pages on different sites
  - limited interaction, *e.g.*, Import of library resources, forms, hyperlinks
  - *e.g.*, consider <iframe src="http://www.iamanevilsite.com">

What if we don't have SOP protection?



Threats SOP intended to deal with

- Prevent an attacking script inadvertently downloaded (from a hostile website) by the victim, *e.g.*:

- initiate HTTP requests on behalf of the victim
- impersonate the victim entirely for subsequent transactions on victim's other web-based accounts/services.
- redirecting the victim to a seemingly legitimate Phishing site

What is “Origin”?

- Origin = domain name + protocol + port
  - all three must be equal for original to be the same
  - all of these are vital, as changing one may lead to accessing something outside your own control
  - however, some access allowed for pages from same domain, but not same host (see later)

URL (http://store.company.com)	Result	Reason
http://store.company.com/dir/inner/another.html	✓	
https://store.company.com/secure.html	✗	protocol
http://store.company.com:81/dir/etc.html	✗	port
http://news.company.com/dir/other.html	✗	host

SOP Exceptions, Issues, and Workaround

- There is NO SINGLE well-defined SOP standard
  - The actual details of SOP is highly implementation-specific

    e.g., differ for different browsers
  - Many vulnerabilities due to different exceptions allowed by different systems
- New ways to bypass SOP keep showing up, *e.g.*,
  - Flash browser plugins allow cross-domain requests if allowed by a rule in crossdomain.xml
  - https://nealpoole.com/blog/2011/10/java-applet-same-origin-policy-bypass-via-http-redirect

Parent Domain Traversal

- For example, store.y.com and login.y.com can be considered as the same “origin”?
- Why make this exception? Convenience
- A possible implementation is to just “blindly” check the second-level domain
- Problematic with international domains (*e.g.*, co.uk)
- Now, goodltd.co.uk could be considered as the same origin as badltd.co.uk

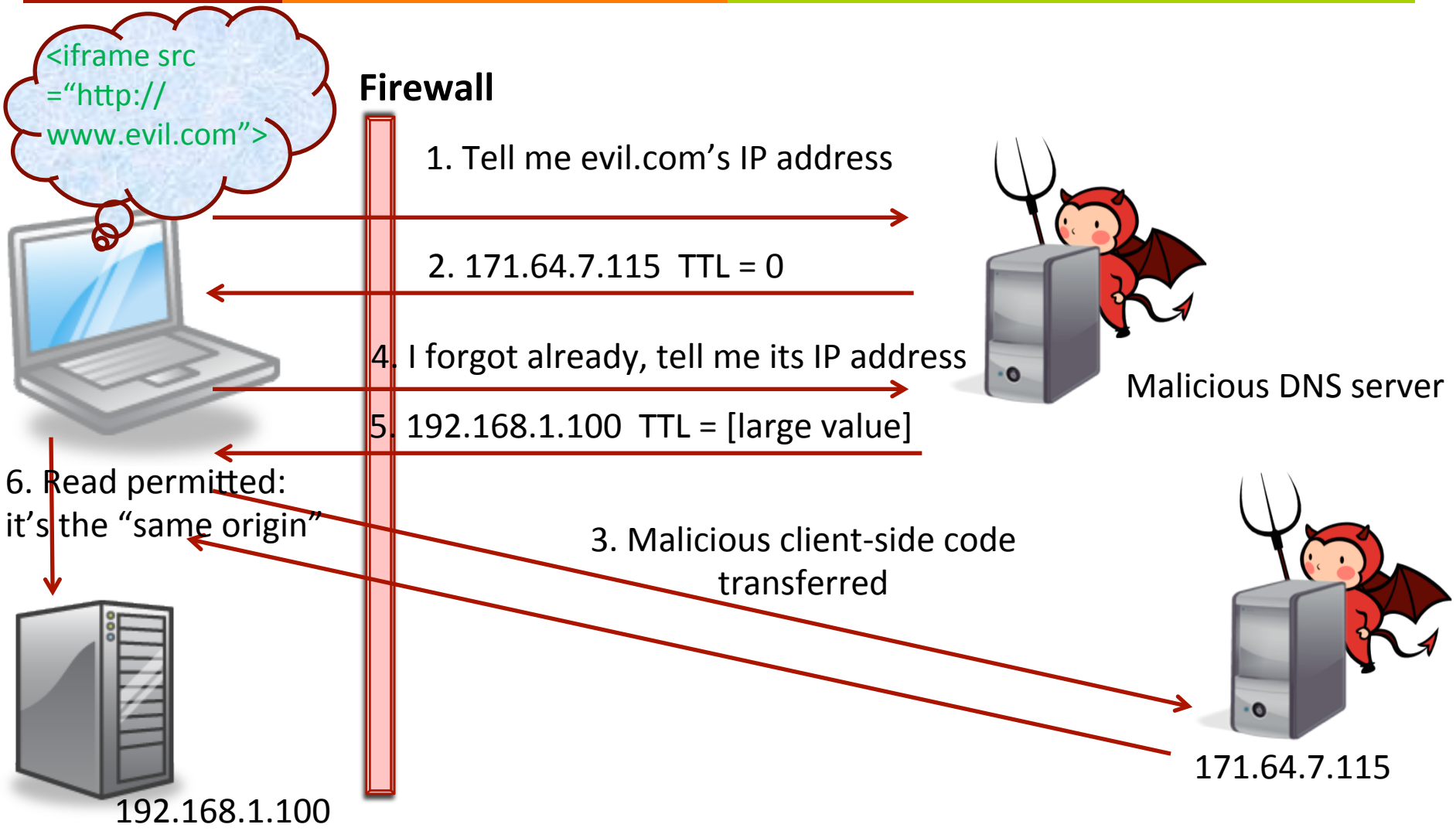
Even more SOP exceptions [**]

- Collaborative Cross-origin Access / Mashup Security
  - (if used correctly)
- 1. Domain Relaxation: Use of document.domain
- 2. Programmatic Form Submission
- 3. Script Inclusion and JSONP
- 4. Use of Fragment Id (#)
- 5. Use of window.postMessage()
- 6. Cross-Origin Resource Sharing (CORS) - XMLHttpRequest Level 2
  - Clickjacking: UI redressing with opacity=0

Attacking “SOP-protected” browser

- Decision is based on the “origin”. Manipulate the origin then!
- At first, a DNS says that www.evil.com points to a “wrong” IP
- For the next request, this DNS says it points to 192.168.0.100!
- How? This malicious DNS just replies with TTL = 0
- “I thought it’s of the malicious DNS server’s best interest to make its invalid record’s *living time* as long as possible…” =O

DNS Rebinding Attack



Firewall

1. Tell me evil.com's IP address

2. 171.64.7.115 TTL = 0

4. I forgot already, tell me its IP address

5. 192.168.1.100 TTL = [large value]

Malicious DNS server

6. Read permitted: it's the "same origin"

3. Malicious client-side code transferred

192.168.1.100

171.64.7.115

Alternative Attack Strategy

- The above attacks visit a malicious (code hosted by a malicious) website, with some references to the target (good) server.
- How about putting malicious code “at the legitimate server”?
- SQL injections tricks a legitimate (but flawed) target webserver by supplying “wrong” input parameters/commands
  - Get unauthorized data, bypass login, issue command on server…
- How about injecting malicious code to taint the webpage?

From injecting code for you, to injecting for others

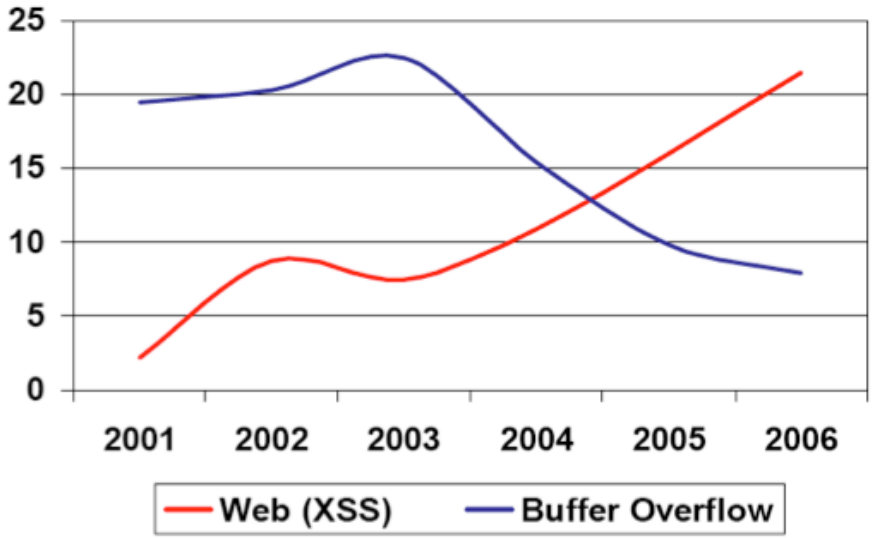
- When a victim visits the tainted webpage (but hosted by a legitimate server), the malicious code is loaded into and run by the victim user's browser
- Get sensitive data from the victim user's machine, while using *the legitimate* but flawed website (login, password, cookie)

Cross-Site Scripting (XSS) Attack

- XSS attacks inject malicious code to this website, to be downloaded and executed by the client.
- The victim “trusts” a website, and grants “access privilege”.

XSS vs. BO

- Web App based Vulnerabilities surpassed Buffer Overflow ones
- Source: MITRE CVE trends



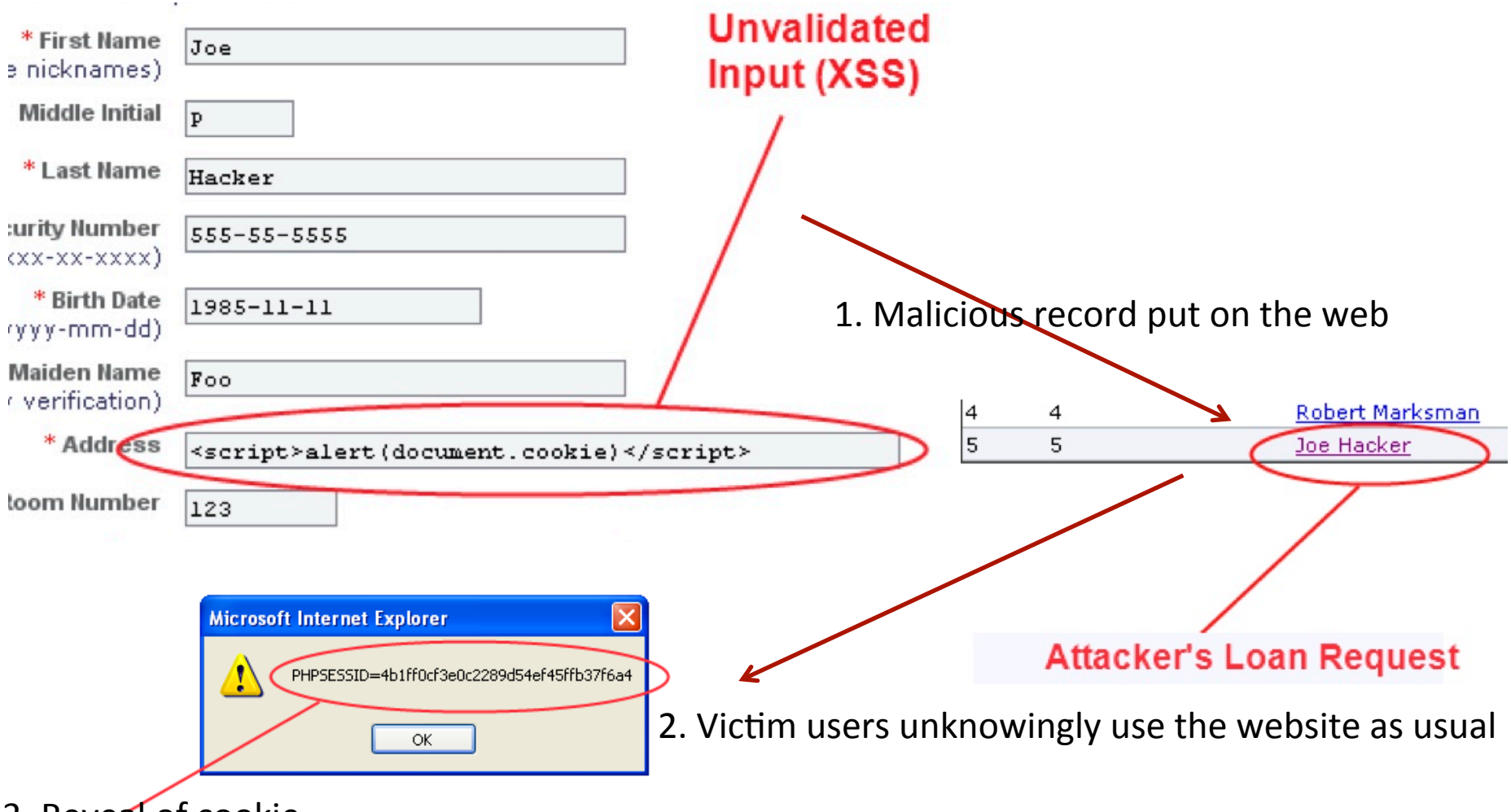
(Artificial) Persistent Cross-User Attack

- Caffè Sospeso
- Suspended / Pending Coffee
  - Paying the price of two coffees but receiving and consuming only one, the next doesn't need to pay
- What if the policy is "Paying the price of, and hence receiving two coffees; but putting back one for the next" instead?
- A bad guy can inject things to the coffee for the next victim!
  - Both guys are "users of" / drinking coffee

Characteristics of the Attack

- The attack is “persistent”. It leaves trail.
- The attack is against the next user (or multiple upcoming users, if a pot of coffee is left).
- The attack is possible since the barista did not check the input.
- The next customers will never check the coffee because they trust the café.
- The café is potentially liable since they store and offer the problematic coffee.

(Persistent) XSS Example



Non-Persistent XSS

- Nothing is stored in the vulnerable website
- Vulnerable web employs user input in HTML pages
- returned to the user's browser
- **without** validating the input first
- *e.g.*, the link corresponds to the query result from Google is in the form of google.com/url?...url=http://thewebyouwant.com
- Reflection attack: User's "input" is "reflected" to the user
- Also known as "Reflected XSS"

Idea of XSS (illustrations)

- How to make CUHK's homepage *says* "Sherman Chow is the best"?
- Just search! http://cloud.itsc.cuhk.edu.hk/cuhksearch/search.aspx?domain=www.ie.cuhk.edu.hk&query=sherman+chow+is+the+best
- At least, the query result will contain the html saying "No results found for [query]".
  - Not really… the above link is Google Custom Search, but yes for Google
- How to make facebook.com notify your friend the following message: "No one ever posted on your time"?
- Rename your name to "No one ever" and post on his/her timeline

Steps in Non-Persistent XSS

- Inject code
- Check if the code is returned in the original form
- And more importantly, the code is executed
- *e.g.*, http://cloud.itsc.cuhk.edu.hk/cuhksearch/search.aspx?domain=www.ie.cuhk.edu.hk&query=<script>Malicious__Fx(...)</script>
- Just a single request / response cycle

Non-Persistent XSS Example

- The attacker first identifies a vulnerable website and crafts the attack
- "Attack vector": the "means" that makes the attack work
  - Finding injection points, *e.g.*, URL, http header
  - Crafting the malicious URL
- The attacker can send an email with an innocent-looking link
  - pointing to a trusted yet vulnerable web, but with the "attack vector"
- The attacker can also put the link on a gray-ware website
- The victim clicks the link which causes the victim's browser to execute the injected script, if the web-server is vulnerable to XSS

Consequence

- The payload (code) will get executed in the victim's context.
- Will call home to the attacker to communicate the result
- Authentication cookies theft; Data theft; Malware installation
- Ping your network to find servers, access web-based configuration
  - Like http://192.168.1.1 of your router's web-based admin page
- Depend on your logged account (admin?), the browser, the OS, and of course the attack. In short, your mileage may vary
- http://www.blackhat.com/presentations/bh-jp-06/BH-JP-06-Grossman.pdf

Universal Cross-Site Scripting (UXSS)

- Common XSS targets website / web applications.
  - If you didn't go to that web, you are safe
  - Usually, the consequence is within the context of that web.
    - Thanks to the security functionality of the browser.
- Universal: exploits *client-side* vulnerability
  - in browser / browser extension
  - may bypass the security feature of a browser
- A recent example: Chrome for Android

UXSS in PDF

- Run malicious scripts on a victims computer when (nearly any) browser executed such a link and uses Acrobat in embedded mode:
- http://host/file.pdf#anyname=javascript:your_code_here
  - (Nearly) every computer has Acrobat Reader installed.
  - (Nearly) every web site has PDF files
  - Now a web site is vulnerable to every imaginable XSS scenarios without having a local XSS vulnerability
- https://www.owasp.org/images/4/4b/OWASP_IL_The_Universal_XSS_PDF_Vulnerability.pdf

Consequence and Fix

- Consider the linked is pointing to a PDF in your local machine!
- Client-side solution: just (ask Adobe to) fix the bug
- Can the server detect that?
  - # in the URL is for redirecting to a specific segment
  - Server cannot see it
- Well, the server can always pass it as octet-stream
  - a.k.a. ask you to download, but can't see the PDF in browser
- There are also other server-side mechanisms (details omitted).

XSS Defense

- CHECK data types and lengths
- DISALLOW unwanted data (*e.g.*, HTML tags, JavaScript)
- ESCAPE questionable characters (ticks, --, ;, [], etc.)

- Sounds familiar? Yes, ~ defending against SQL injection
- Caution! Scripts not only within <script>
  - Inline JavaScript, *e.g.*, <img src="javascript.alert(…)">
    - Browser should have checked if that is really an image in this case
  - JavaScript events, *e.g.*, OnSubmit

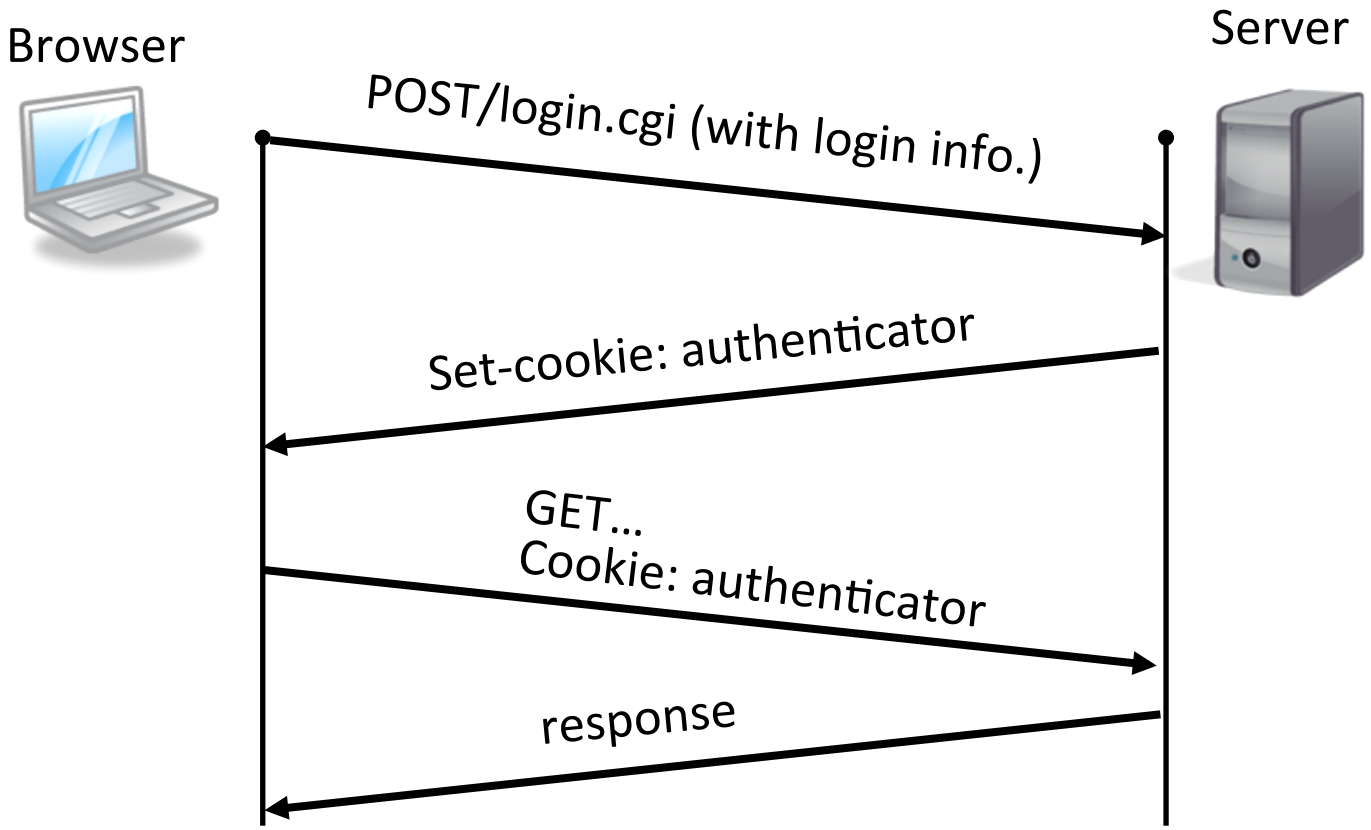
Summary of XSS

- Web Application is used to store, transport, and deliver malicious active content to an unsuspecting user.
- **Cause:** Failure to proactively reject or scrub malicious characters from input vectors.
- **Impact:**
  - Persistent XSS is stored and executed at a later time, by a user.
  - Allows cookie theft, credential theft, risks of confidentiality, integrity, & availability
  - Browser hijacking &unauthorized access to web application is possible by existing exploits.
- **Solution:**
  - A global, as well as Form- and Field- specific policy, for handling untrusted content
  - Use white lists and “regular expressions” to ensure input data conforms to the required character set, size, and syntax. (Tricky to know where to check and what to allow.)

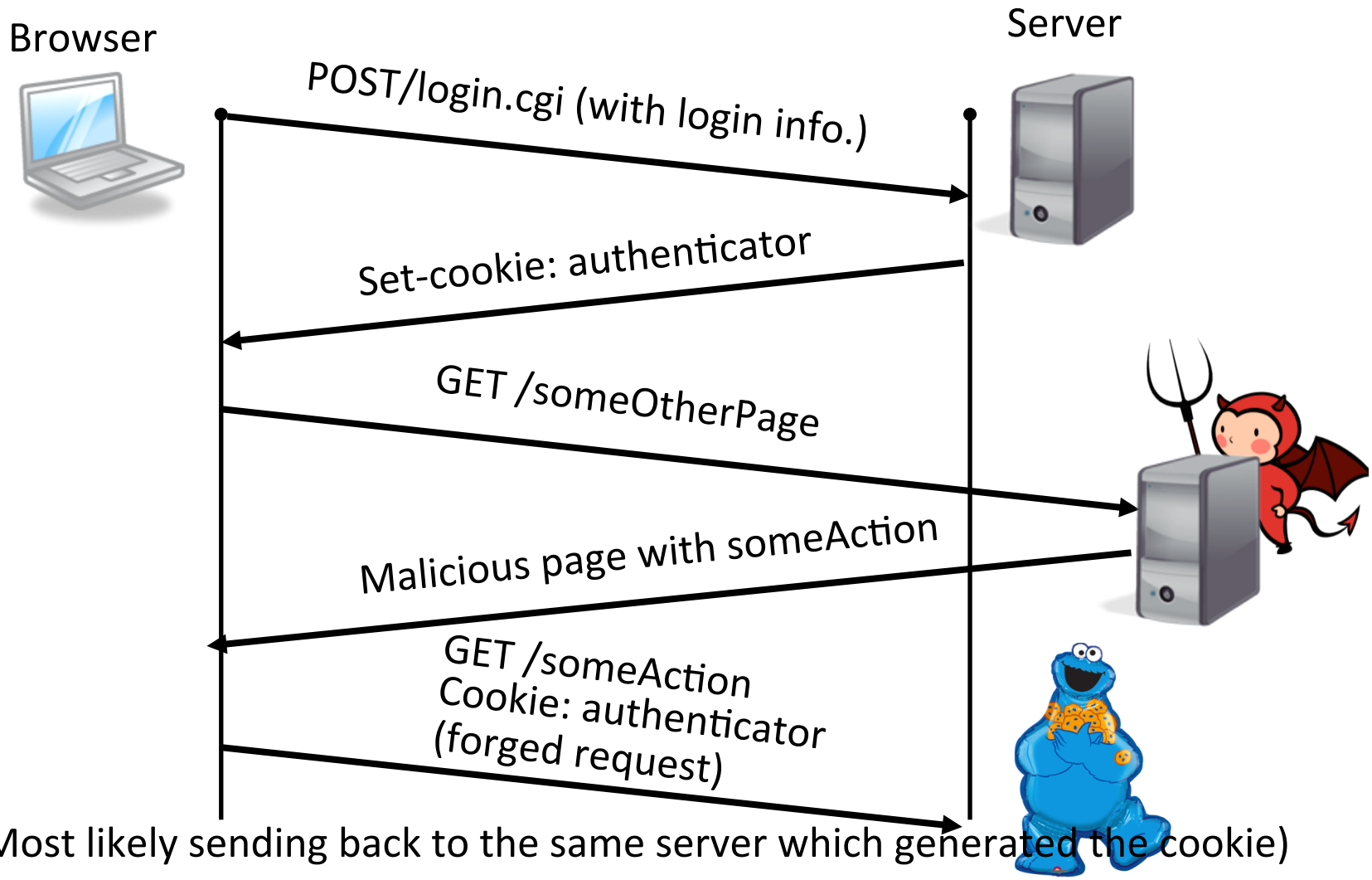
From the user trusting a site, to a site trusting the user

- Cross-site request forgery (XSRF, or CSRF) exploits the trust that a site has in a user's browser
- The site trusts the user's browser "*unwanted*" actions won't happen
  - The word "unwanted" implies that an HTTP request has "*side effect*"
- Attacker *"forges" a request* to do those actions "for" the *victim* user
  - The word "Forgery" implies that the server has authenticated the user
- **Root cause**: Existing browsers do not check whether a client actually *initiates* an HTTP request.

Typical Web Traffic using Cookie



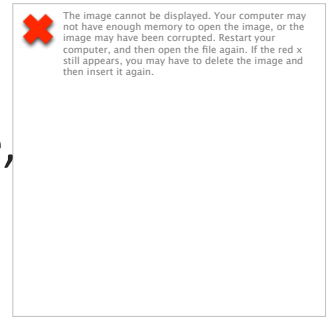
Basic Flow of XSRF Attack



(Most likely sending back to the same server which generated the cookie)

XSRF Example

- Eve posts on a Internet forum Alice may visit with the following:
  - Hello Alice! Look here:
  - <img src="http://m.naivebank.com/withdraw?account=Alice&amount=1000000&for=Eve">
  - (The “referer” field in the HTTP packet’s header is actually not naivebank.com but that forum)
  - (BTW, misspelling above)



- If Alice's bank keeps her authentication information in a cookie,
- *and* if the cookie hasn't expired,
- then the attempt by Alice's browser to load the image will submit the withdrawal form with her cookie
- thus authorizing a transaction without Alice's (explicit) approval.

Limitations of XSRF

- The attacker must
  - either target a site that doesn't check the referrer header (which is common)
  - or a victim with a browser or plugin that allows referrer spoofing (which is rare).
- The attacker must find a form submission at the target site, or a URL that has side effects, that does something
  - (*e.g.*, transfers money, or changes the victim's e-mail address or password).
- The attacker must determine the right values for all the forms or URL inputs;
  - if any of them are required to be secret authentication values or IDs that the attacker can't guess, the attack will fail.
- The attacker must lure the victim to a Web page with malicious code while the victim is logged into the target site.

More Discussions

- The attack is blind, *i.e.*, the attacker can't see what the target website sends back to the victim in response to the forged requests
  - XSS may help the attacker to see that!
- Why a website may check the referrer header?
  - *e.g.*, an image-hosting web may prevent "hot-linking" of those hosted image from other web like Internet forums
- Why a browser may (have plugins which) support referrer spoofing?
  - Browser users may think it is a convenient feature.