Cybersecurity – EC521
Web Security

Manuel Egele
PHO 337
megele@bu.edu
Boston University

Some slides based on material from Engin Kirda & Wil Robertson @ NEU

Web Application Security

- When an organization puts up a web application, they invite everyone to send them HTTP requests.
- Attacks buried in these requests sail past firewalls without notice because they are inside legal HTTP requests.
- Even “secure” websites that use SSL just accept the requests that arrive through the encrypted tunnel without scrutiny.
- This means that your web application code is part of your security perimeter!

Web Application Security

- The security issues related to the Web are not new. In fact, some have been well understood for decades
  - For a variety of reasons, major software development projects are still making these mistakes and jeopardizing not only their customers' security, but also the security of the entire Internet.
  - There is no "silver bullet" to cure these problems. Today's assessment and protection technology is improving, but can currently only deal with a limited subset of the issues at best.
  - To address the security issues, organizations will need to change their development culture, train developers, update their software development processes, and use technology where appropriate.

Why is it Important?

- Easiest way to compromise hosts, networks and users (Remember Equifax?)
- Widely deployed
- No Logs! (POST Request payload)
- Difficult to defend against or to detect
- Firewall? What firewall? I don't see any firewall…
- Encrypted transport layer does not help much

News (well 2017) From The Field

Equifax got hacked

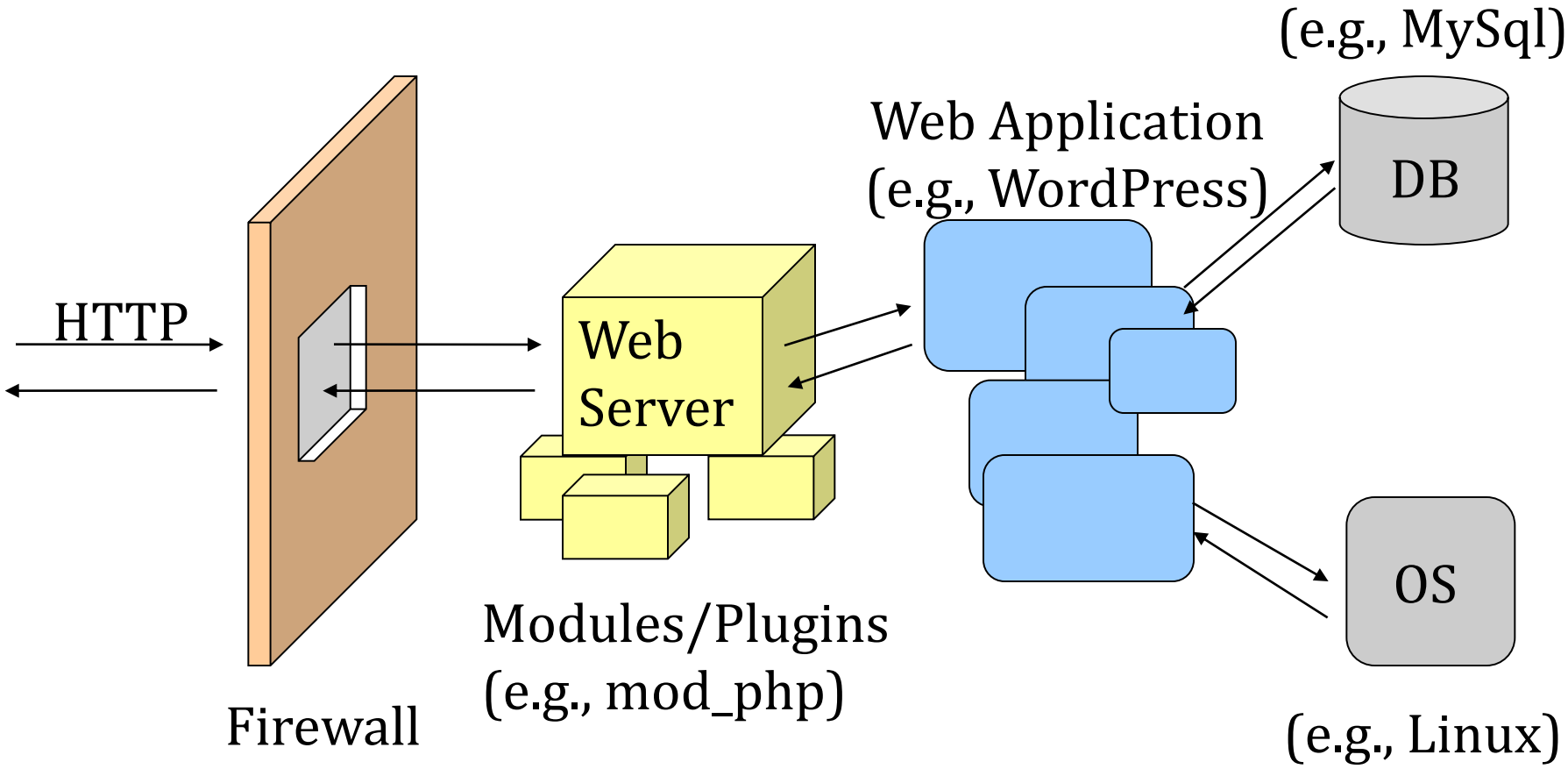
- One of the big three credit bureaus
- Has credit history and PII on almost all Americans (lost data of 143M) and many foreigners (e.g., millions of Brits)
- Vulnerability in Apache Struts framework (CVE-2017-9805) ***remote code execution***
- Security vulnerabilities happen and hopefully get fixed but …

Big Problem: Vulnerability was known ***and*** patches released two months before Equifax got hit! i.e., They didn't update (negligent!)

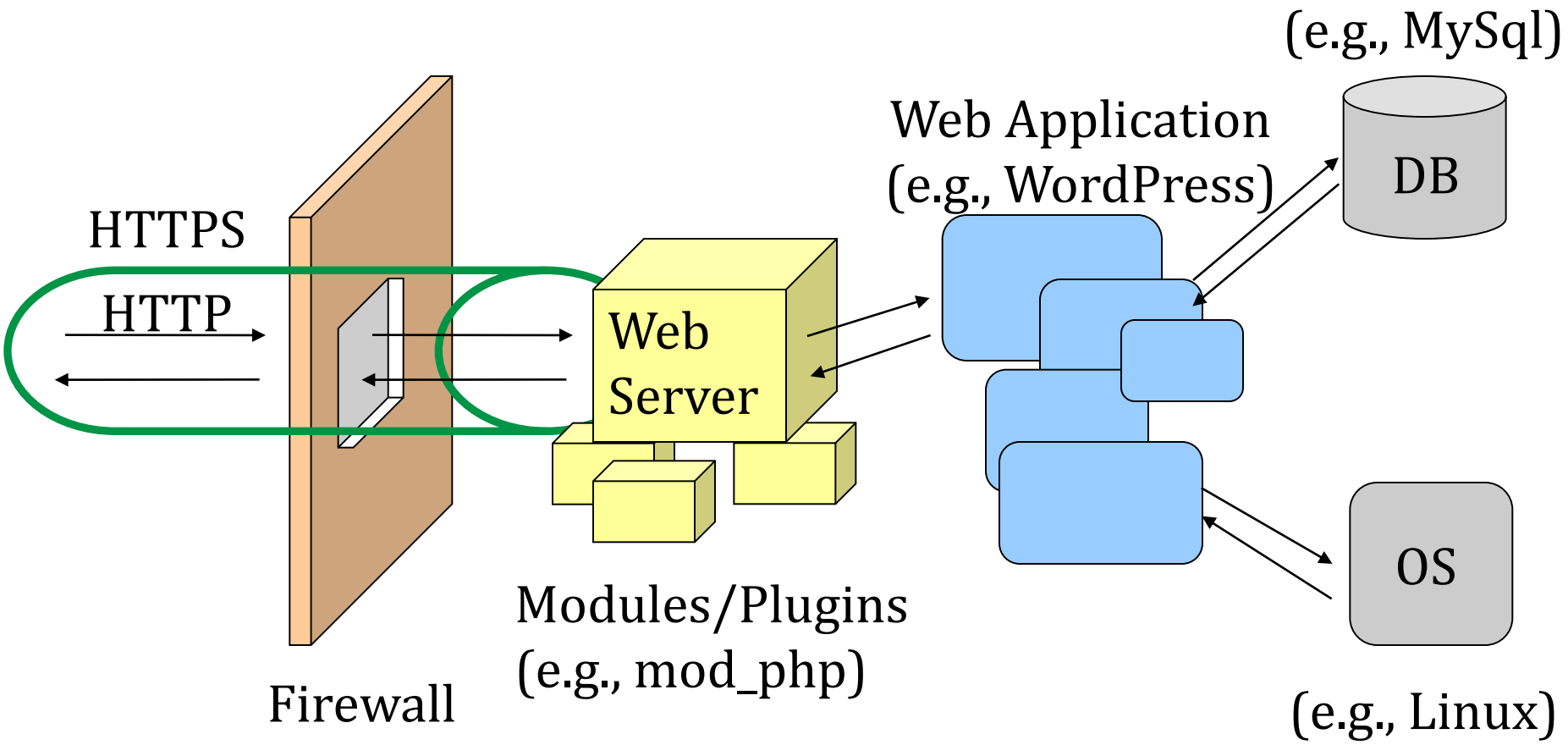
Why is it Important?

- Easiest way to compromise hosts, networks and users (Remember Equifax?)
- Widely deployed
- No Logs! (POST Request payload)
- Difficult to defend against or to detect
- Firewall? What firewall? I don't see any firewall…
- Encrypted transport layer does not help much

On a Typical Web Server



On a Typical Web Server (w/ https)



On a Typical Web Server…

Your host has open 80/8080 port (firewall)

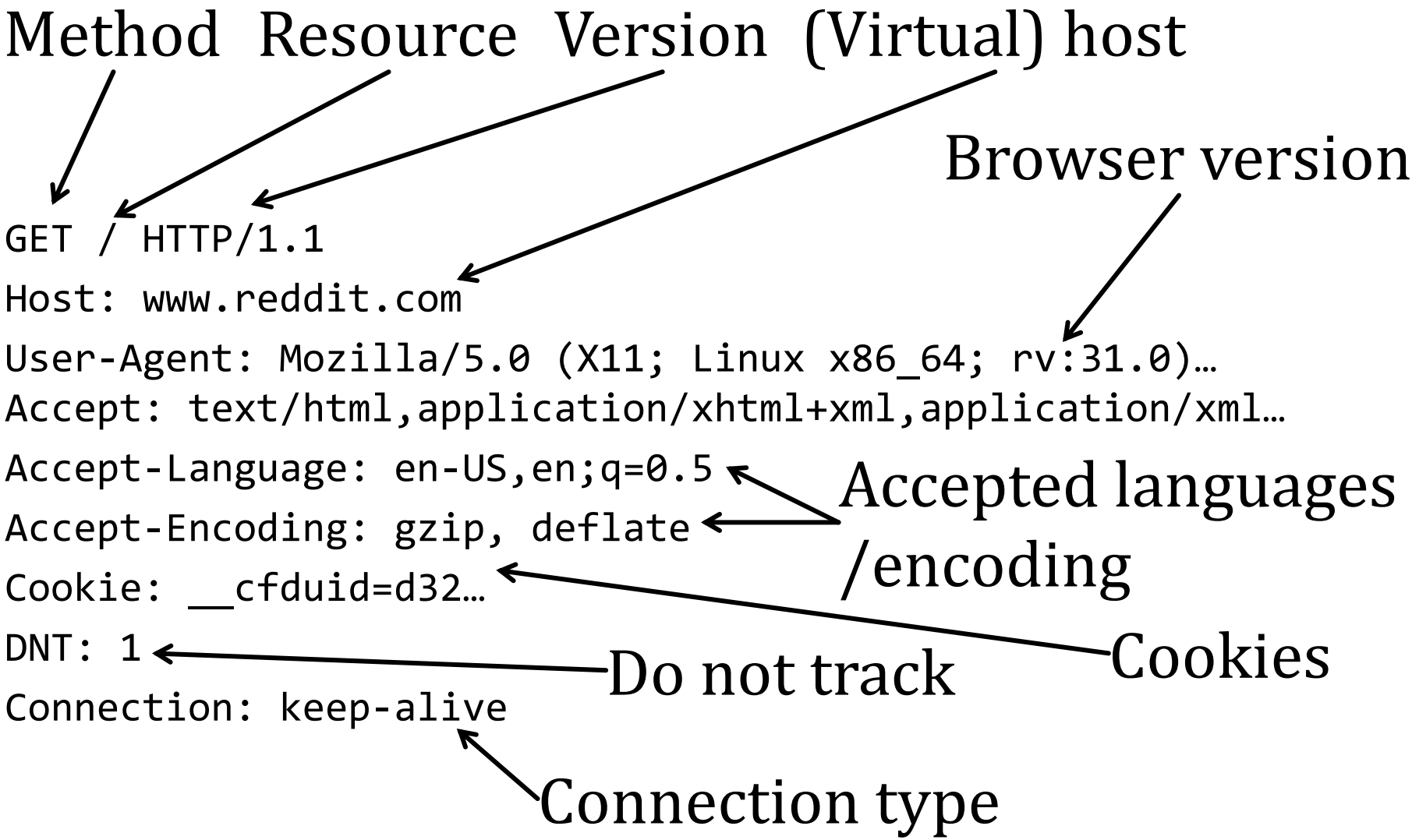
Following components are running

- OS
- Web Server
  - Main application (e.g., Apache)
  - Plugins (e.g., mod_php, mod_perl)
  - Servlets
  - Scripts (CGI, Perl, php, ...)

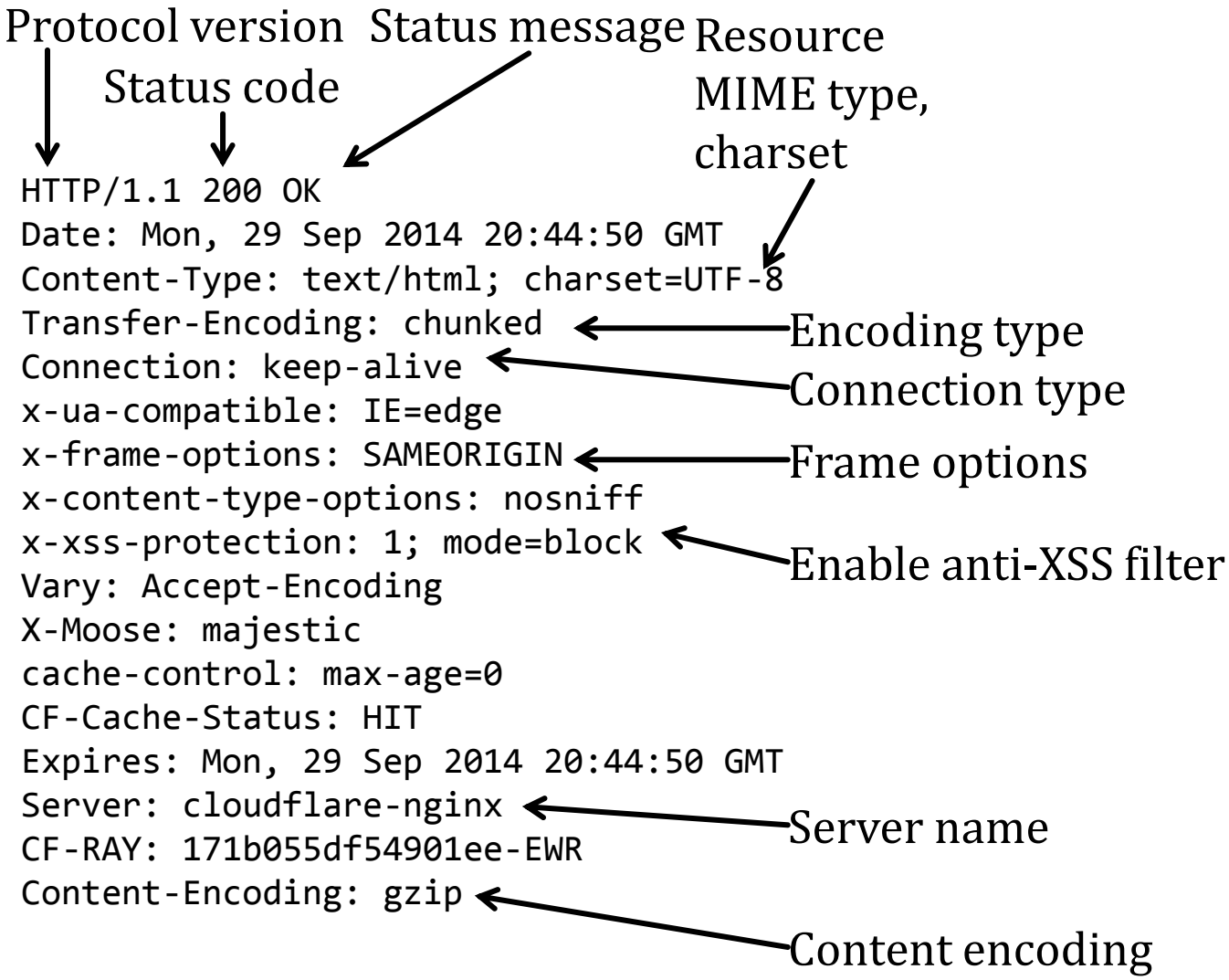
HTTP

- The web is built on the Hypertext Transfer Protocol (HTTP)
  - (Originally) text-based and stateless
- Messages have a header & an optional body
  - Header: request method, response status, resource paths, versions, key-value pairs
  - Body can contain request parameters and resource contents

HTTP Request Example



HTTP Reply Example

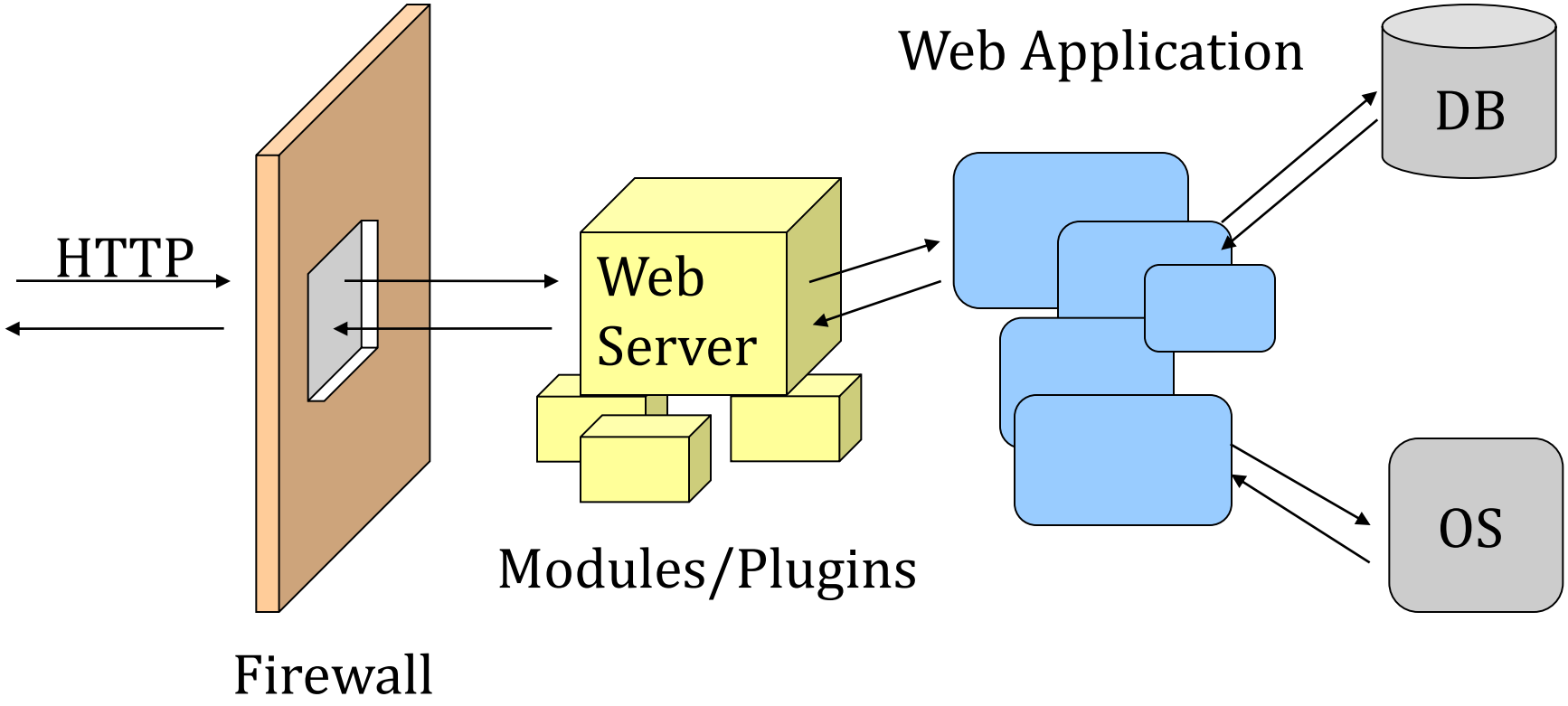


HTTP Methods

GET	Retrieve resource at given path
HEAD	Identical to GET, but response omits body
POST	Submit data to a given path, might create resources at new paths
PUT	Submit data to a given path, creating or modify resource *at that path*
DELETE	Deletes resource at a given path
TRACE	Echoes request
OPTIONS	Returns supported HTTP methods given a path
CONNECT	Creates a tunnel to a given network location

Demo
http://www.bu.edu

On a Typical Web Server



Web Server Scripting

- Allows easy implementation of functionality (also for non-programmers – Think: Is this good?)
- Example scripting languages are Perl, Python, ASP, JSP, PHP
- Scripts are installed on the Web server and return HTML as output that is then sent to the client
- Template engines are often used to power web sites
  - E.g., Cold Fusion, Cocoon, Zope
  - These engines often support/use scripting languages

Web Application Example

Objective: Write an application that accepts username and password and displays them

- First, we write HTML code and use forms

```
<html><body>
<form action="/scripts/login.pl" method="post">
Username: <input type="text" name="username"> <br>
Password: <input type="password" name="password"> <br>
<input type="submit" value="Login" name="login">
</form>
</body></html>
```

Web Application Example

Second, here is the corresponding Perl script that prints the username and password passed to it:

```
#!/usr/local/bin/perl
uses CGI;
$query = new CGI;
$username = $query->param("username");
$password = $query->param("password");
…
print "<html><body> Username: $username <br>
Password: $password <br>
</body></html>";
```

OWASP

The Open Web Application Security Project (www.owasp.org)

- OWASP is dedicated to helping organizations understand and improve the security of their web applications and web services.
- The Top Ten vulnerability list was created to point corporations and government agencies to the most serious of these vulnerabilities.
- Web application security has become a hot topic as companies race to make content and services accessible though the web. At the same time, attackers are turning their attention to the common weaknesses created by application developers.

OWASP Top 10 (2007)

1 Injection Flaws **(1) (1 in 2013, and still in 2017)**
2 Broken Authentication and Session Management **(2)**
3 Sensitive Data Exposure **(-)**
4 XML External Entities **(-)**
5 Broken Access Control **(-)**
6 Security Misconfiguration **(-)**
7 Cross Site Scripting (XSS) **(3)**
8 Insecure Deserialization **(-)**
9 Using Components with Known Vulnerabilities **(-)**
10 Insufficient Logging & Monitoring **(-)**

Later!

- Insecure Direct Object Reference **(4)**
- Cross Site Request Forgery **(8)**
- Information Leakage and Improper Error Handling

Common Root for Many Problems

- Web applications use input from HTTP requests (and occasionally files) to determine how to respond
  - Attackers can tamper with any part of an HTTP request, including the URL, query string, headers, cookies, form fields, and hidden fields, to try to bypass the site's security mechanisms
  - Common input tampering attempts include XSS, SQL Injection, hidden field manipulation, buffer overflows, cookie poisoning, remote file inclusion...
- Some sites attempt to protect themselves by filtering known malicious input
  - Problem: There are many different ways of encoding information

Unvalidated Input

- A surprising number of web applications use only client-side mechanisms to validate input
  - Client side validation mechanisms are easily bypassed, leaving the web application without any protection against malicious parameters
- How to determine if you are vulnerable?
  - Any part of an HTTP request that is used by a web application without being carefully validated is known as a “tainted” parameter
  - The simplest way: to have a detailed code review, searching for all the calls where information is extracted from an HTTP request

Unvalidated Input

- How to protect yourself?
  - Ensure all parameters are validated before they are used
  - A centralized component or library is likely to be the most effective … remember complete mediation?
- Parameters should be validated against a "positive" specification that defines:
  - Data type (string, integer, real, etc…); Allowed character set; Minimum and maximum length; Whether null is allowed; Whether the parameter is required or not; Whether duplicates are allowed; Numeric range; Specific legal values (enumeration); Specific patterns (regular expressions)
  - Trying to specify negative specification (i.e., signature matching) is bound to fail / be incomplete

Injection Flaws

"***Injection flaws*** occur when an application sends *untrusted** data to an *interpreter*"

--- OWASP

* usually means attacker-controlled

Injection Attacks Overview

- Many webapps invoke interpreters
  - SQL
  - Shell command
  - Sendmail
  - LDAP
  - ...
- Interpreters execute the commands specified by the parameters or input data
  - If the parameters are under control of the user and are not properly sanitized, the user can inject its own commands in the interpreter