



Web Application Vulnerabilities



Approaching Layer 7



“The art of war teaches us to rely not on the likelihood of the enemy's not coming, but on our own readiness to receive him; not on the chance of his not attacking, but rather on the fact that we have made our position unassailable.”

- Sun Tzu, The Art of War

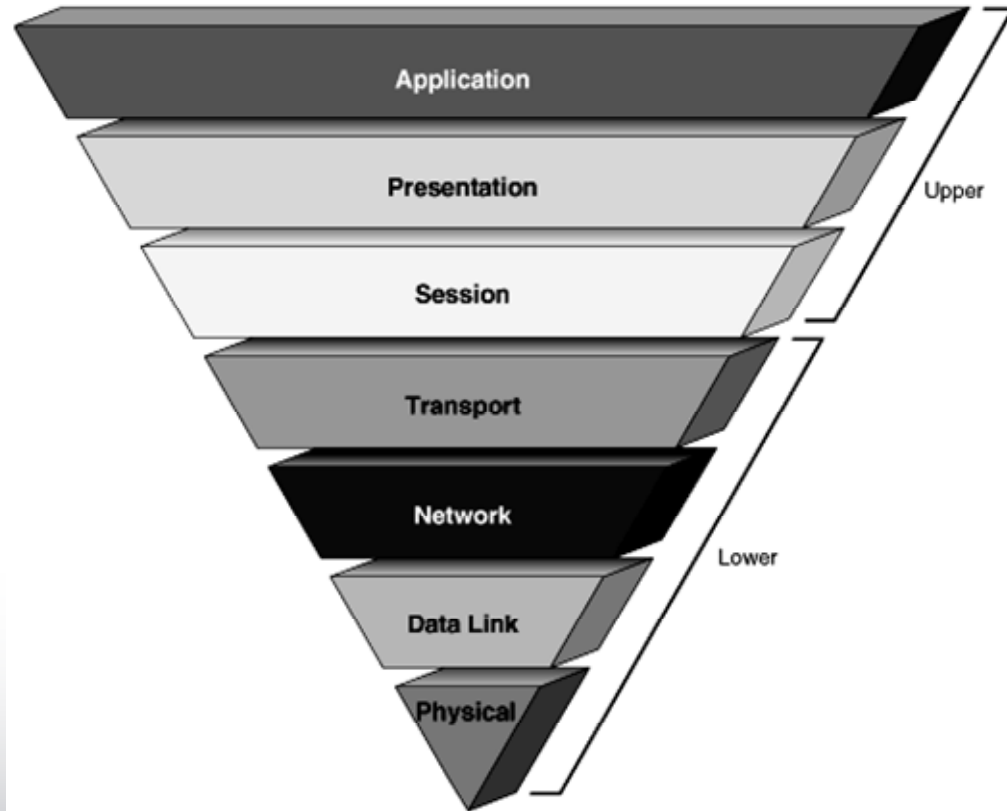


./outline

- **Common Web Application Architecture and Concepts**
- **Common Web Application Vulnerabilities**
- **Demo**



Layer 7



So what would you consider a Web Application?

Web application — sometimes called a web app or webapp and much less frequently a weblication — is an application that's accessed with a web browser over a network such as the Internet or an Intranet.



Source: Wikipedia - http://en.wikipedia.org/wiki/Web_application

Scary Statistics

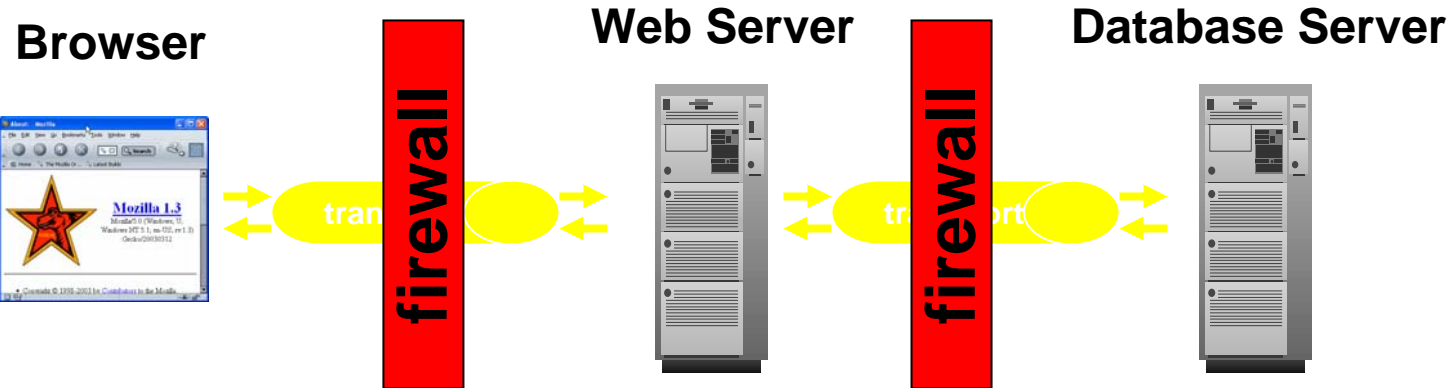


- **95% of all Web Applications have Vulnerabilities**
 - Cross-site scripting (80 per cent)
 - SQL injection (62 per cent)
 - Parameter tampering (60 per cent)
 - Cookie poisoning (37 per cent)
 - Database server (33 per cent)
 - Web server (23 per cent)
 - Buffer overflow (19 per cent)

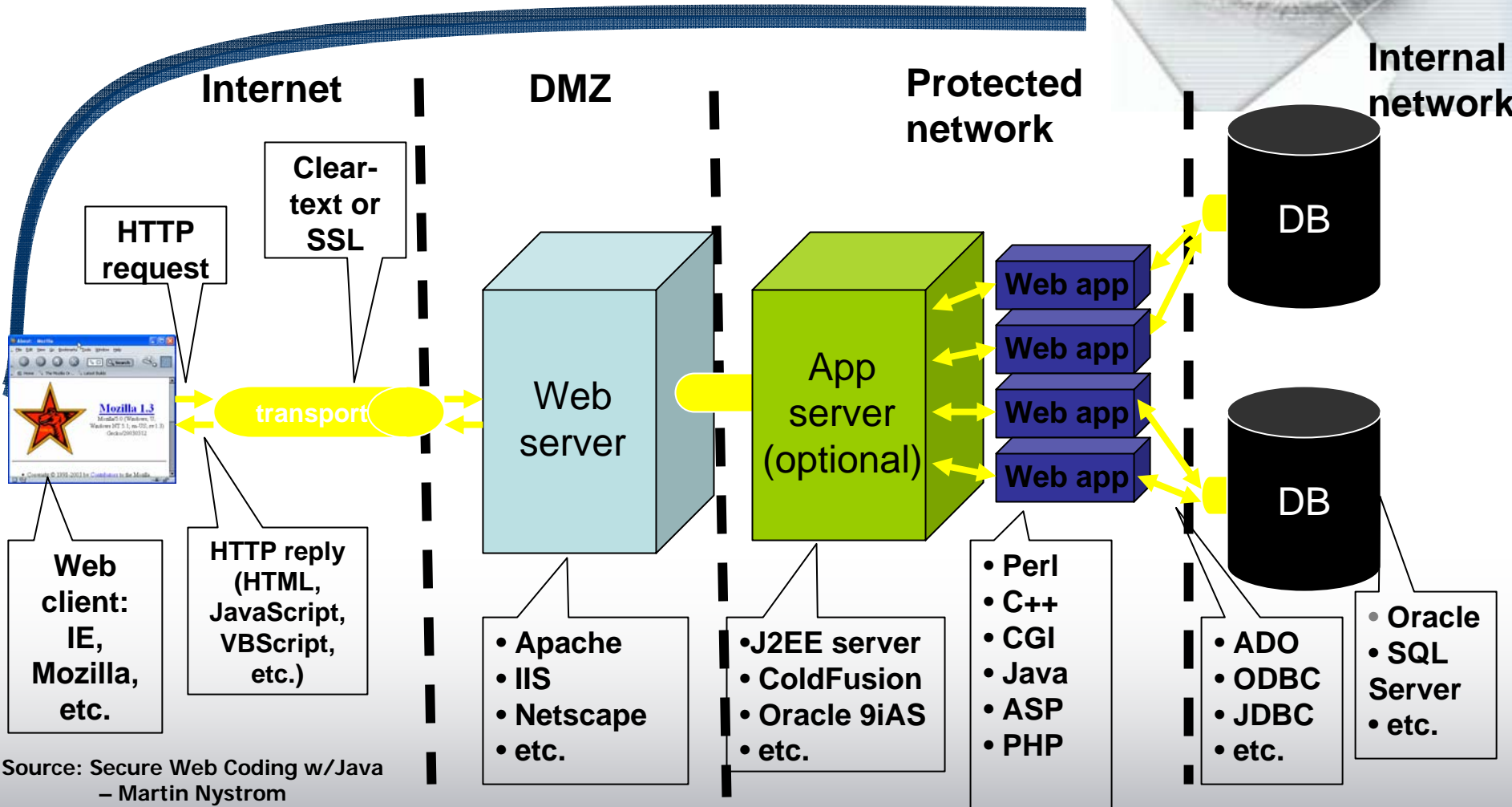
Source: <http://www.vnunet.com/news/1152521>



Simple Web Application Architecture



Complex Commercial Web Application Architecture



Source: Secure Web Coding w/Java – Martin Nyström

Nessus ≠ Secure Web Apps



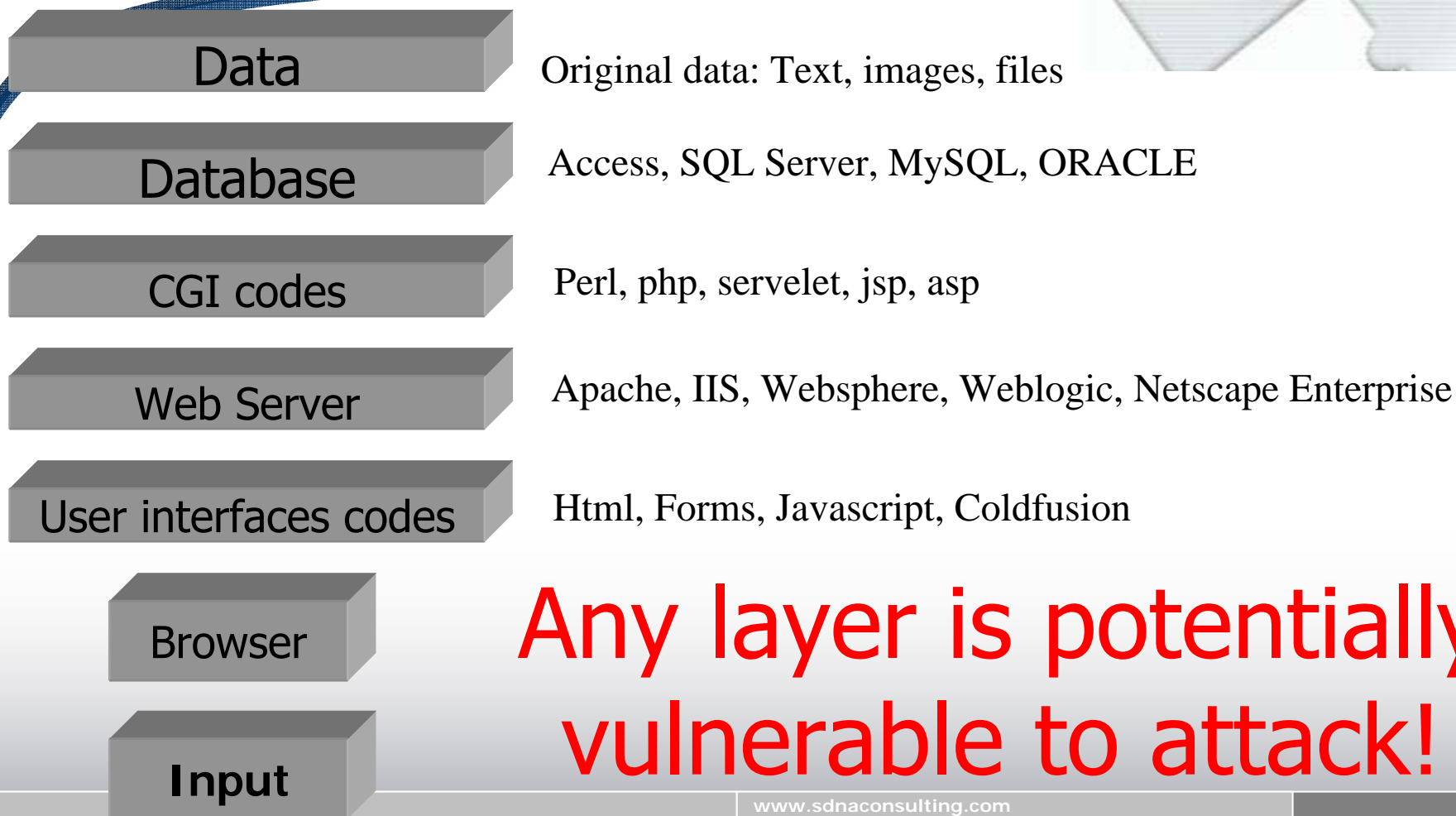
- So once we've run Nessus against this infrastructure and patched all the vulnerabilities that were detected, we're ok right?

Wrong

- Detection of Web Application vulnerabilities requires:
 - Detailed manual testing and code review
 - Use of a commercial Web Application Vulnerability Scanners (WebInspect, Appscan)



Where do we begin?



Any layer is potentially vulnerable to attack!

OWASP TOP 10



http://www.owasp.org/index.php/OWASP_Top_Ten_Project

- **Unvalidated input**
- **Broken access control**
- **Broken account/session management**
- **Cross-site scripting (XSS) flaws**
- **Buffer overflows**
- **Injection flaws**
- **Improper error handling**
- **Insecure storage**
- **Denial-of-service**
- **Insecure configuration management**



Overview of OWASP 10

- 1. Unvalidated input**
Information from web requests is not validated before being used by a web application. Attackers can use these flaws to attack backend components through a web application.
- 2. Broken access control**
Restrictions on what authenticated users are allowed to do are not properly enforced. Attackers can exploit these flaws to access other users' accounts, view sensitive files, or use unauthorized functions.
- 3. Broken account/session management**
Account credentials and session tokens are not properly protected. Attackers that can compromise passwords, keys, session cookies, or other tokens can defeat authentication restrictions and assume other users' identities.
- 4. Cross-site scripting (XSS) flaws**
The web application can be used as a mechanism to transport an attack to an end user's browser. A successful attack can disclose the end user's session token, attack the local machine, or spoof content to fool the user.
- 5. Buffer overflows**
Web application components in some languages that do not properly validate input can be crashed and, in some cases, used to take control of a process. These components can include CGI, libraries, drivers, and web application server components.



Overview of OWASP 10

6. Injection flaws

Web applications pass parameters when they access external systems or the local operating system. If an attacker can embed malicious commands in these parameters, the external system may execute those commands on behalf of the web application.

7. Improper error handling

Error conditions that occur during normal operation are not handled properly. If an attacker can cause errors to occur that the web application does not handle, they can gain detailed system information, deny service, cause security mechanisms to fail, or crash the server.

8. Insecure storage

Web applications frequently use cryptographic functions to protect information and credentials. These functions and the code to integrate them have proven difficult to code properly, frequently resulting in weak protection.

9. Denial-of-service

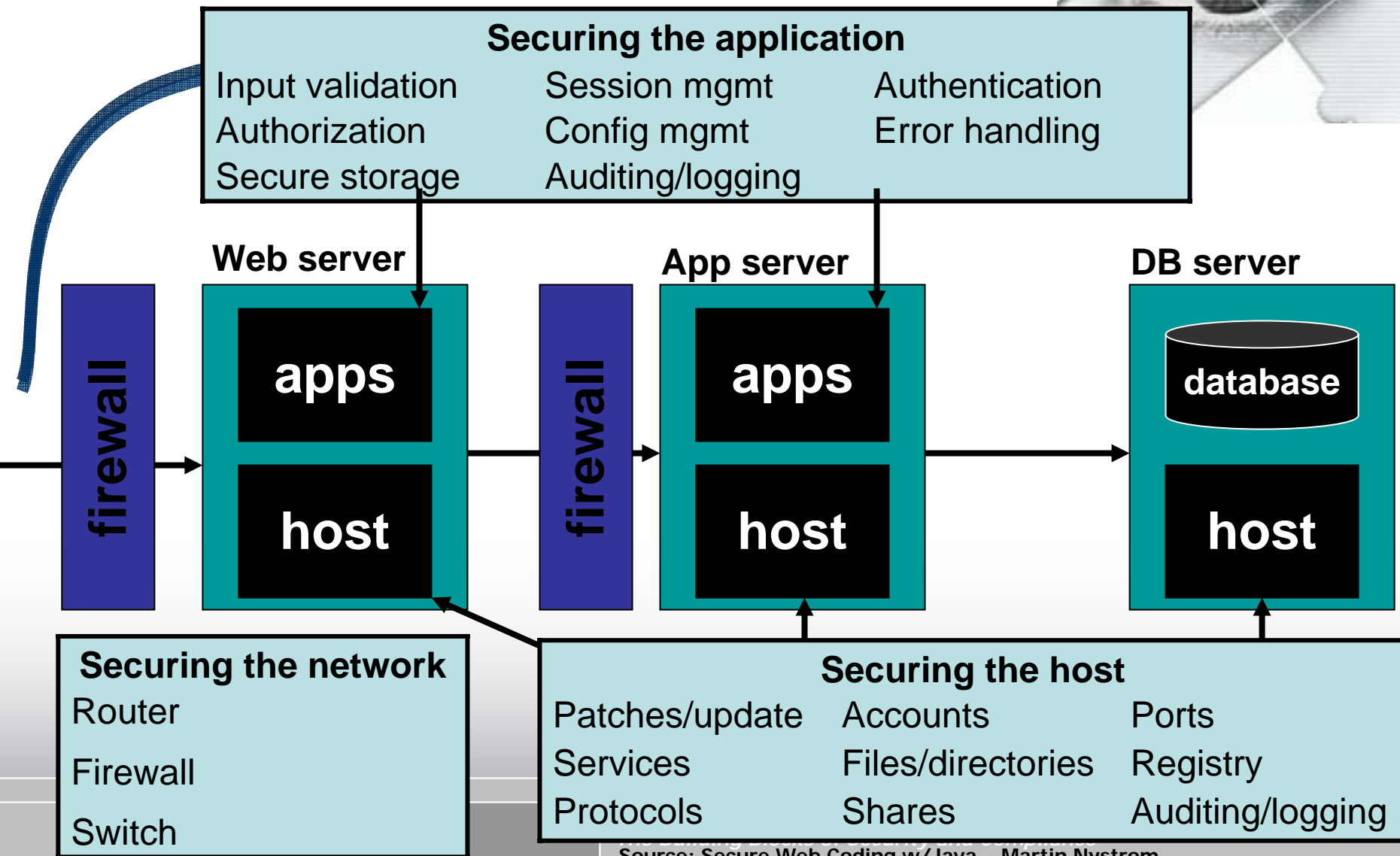
Attackers can consume web application resources to a point where other legitimate users can no longer access or use the application. Attackers can also lock users out of their accounts or even cause the entire application to fail.

10. Insecure configuration management

Having a strong server configuration standard is critical to a secure web application. These servers have many configuration options that affect security and are not secure out of the box.



How do you secure this architecture?



What can you do?



Secure Coding:

- **Don't trust input from user**
- **Watch for logic holes**
- **Leverage common, vetted resources**
- **Only give information needed**
- **Leverage vetted infrastructure & components**
- **Build/test to withstand load**

Approaching Layer 7



“If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle.”

- Sun Tzu, The Art of War





DEMOS

