CS 161 Midterm 2 Review

2019-04-04

CS 161 Course Staff

Topics

- Network Security
  - Network Protocols
  - DNS
  - DNSSEC
  - DoS + Firewalls
  - TLS
  - WPA2
- Web Security
  - Cookies policy
  - Same-origin policy
  - SQL Injection
  - XSS
  - CSRF
- *A little bit of crypto:*
  - Key Management (as part of TLS)

Disclaimer: There could be stuff on the exam that is not covered in the review, and stuff in the review is not necessarily in the exam.

Networking Basics

Attacker Definitions
(for the purposes of this class)

- Man-in-the-middle (aka in-path): Attacker can see and modify traffic.
- On-path: Attacker can see but not modify traffic.
- Off-path: Attacker can’t see anything. (Still knows public information)

Remember all attackers can **send** whatever packets they want!

Think about what fields an attacker would need to guess correctly!

Protocols

A protocol is an **agreement on how to communicate**

- **Syntax**: How a communication is structured.
  - Format and order of how messages are sent and received.
- **Semantics**: What a communication means
  - Actions taken when transmitting, receiving, or timer expires

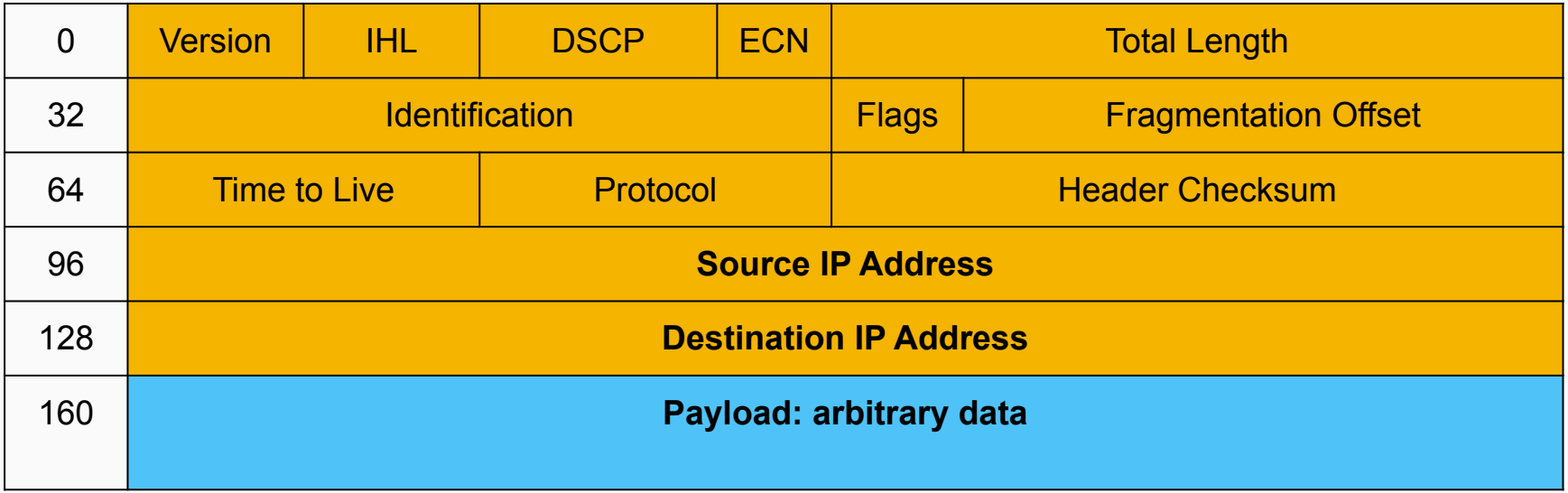
Examples: Wi-Fi, ARP, DHCP, TCP, UDP, IP, TLS, HTTP

Protocols: Example

Making a comment in lecture:

1. Raise your hand.
2. Wait to be called on.
3. Or: Wait for the speaker to **pause** and vocalize
4. If unrecognized (after **timeout**): Say “Excuse me”

Headers: IP


0
Version
IHL
DSCP
ECN
Total Length
32
Identification
Flags
Fragmentation Offset
64
Time to Live
Protocol
Header Checksum
96
Source IP Address
128
Destination IP Address
160
Payload: arbitrary data

Headers: UDP

0	**16-bit source port**	**16-bit destination port**
32	16-bit length field	16-bit checksum
64	**Payload: arbitrary data**	

Headers: TCP

0	**16-bit source port**			**16-bit destination port**
32	**Sequence number**			
64	**Acknowledgment number (if ACK set)**			
96	Data Offset	Reserved	Flags	Window size
128	Checksum			Urgent pointer
160	Payload: arbitrary data			

Headers: DNS

0	**Identification**							
16	QR	Opcode	AA	TC	RD	RA	(Just zeroes)	Response Code
32	QDCOUNT: Number of resource records in question section							
48	ANCOUNT: Number of resource records in answer section							
64	NSCOUNT: Number of resource records in authority section							
80	ARCOUNT: Number of resource records in authority section							
96	Arbitrary Data							

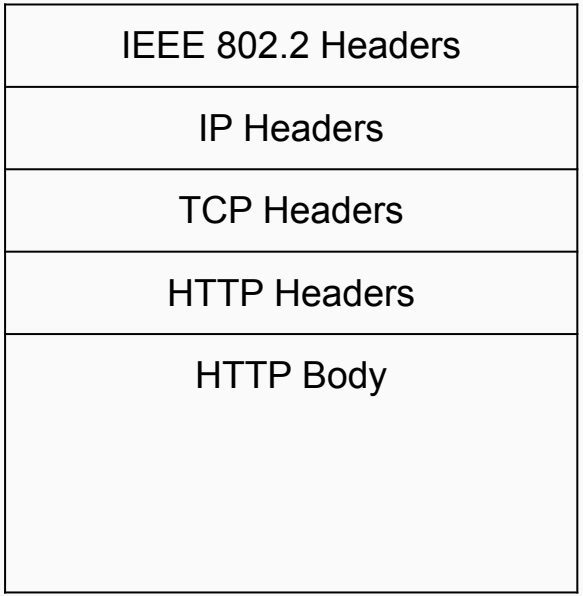
Layering

- The internet is like an onion: layers!
- Each layer relies on services provided by next layer below… and provides services to layer above it
- Similar to levels of abstraction when you program

Layering

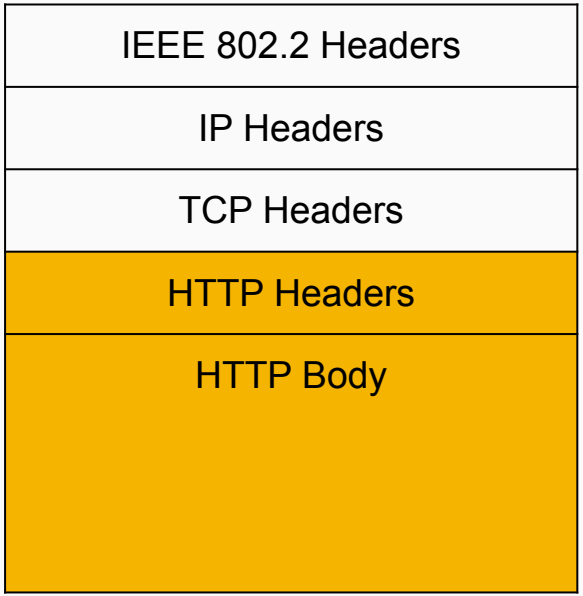
- **Layer 1**: Physical (bits on a wire, radio waves for WiFi, etc.)
- **Layer 2**: Link -- communication between "local" hosts
- **Layer 3**: InterLink -- communication between different "remote" hosts (IP)
- **Layer 4**: Transport -- TCP / UDP (more details soon)
- **Layers 5/6**: nobody cares.
- **Layer 7**: Application: things like HTTP (websites), SSH, etc.

If you receive an HTTP Packet it might look like this



What is the TCP payload?
What is the IP payload?

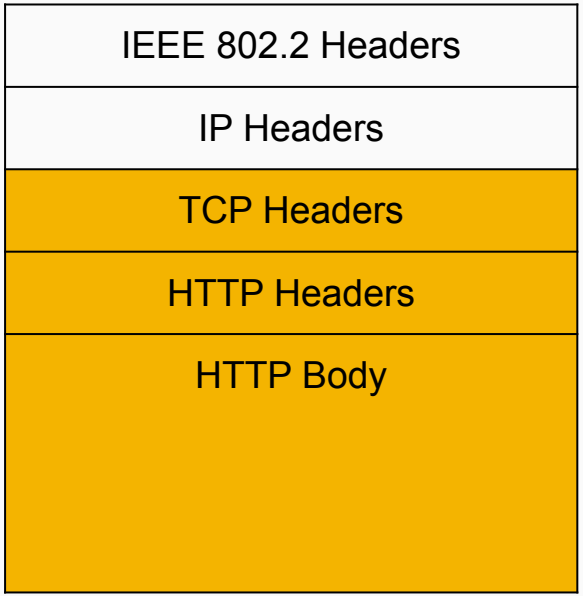
If you receive an HTTP Packet it might look like this



What is the TCP payload?

What is the IP payload?

If you receive an HTTP Packet it might look like this



What is the TCP payload?

What is the IP payload?

OSI Layer 1: Physical layer

- Bits on the wire(less)
- Copper cables, lasers, radio waves (standards: 802.11(a,b,g,n,ac), 802.3ae)
- line codes (e.g. 8b/10b, 64b/66b) for clock recovery
- You don’t need to worry about details in this layer.

OSI Layer 2: Link layer

- Defines communication between local hosts (think Wi-Fi network)
- Support for broadcast protocols
- Main protocol we care about here: ARP

ARP: Address Resolution Protocol

Converts IP addresses to MAC addresses.

- Host A wants to send a message to Host B
- Host A gets Host B's IP address (Call it X) from DNS
- Host A **broadcasts** "What is the MAC address of X?"
- Host B sends a message **only to A** "My IP is X and my MAC address is _".
- The IP, MAC pair gets cached

(If the X is outside the local network, then the gateway responds with their MAC address)

ARP: Spoofing

The ARP request is **public**. What happens if

- Alice sends out an ARP request for Bob’s MAC address
- Mallory responds with hers instead and beats Bob’s response.

ARP: Spoofing

Alice now believes Mallory’s MAC address is Bob’s and will send her packets intended for Bob.

OSI Layer 3: (Inter)network Layer

- Global routing
- No guarantee of reliability
- Packet delivery between different local networks, “best effort”

OSI Layer 4: Transport Layer

- Connection: one service (on a host) to another
- TCP: **Reliable**, in-order, connection-oriented transport
- UDP: **Unreliable**, datagram-based transport
  - A datagram is a single packet message

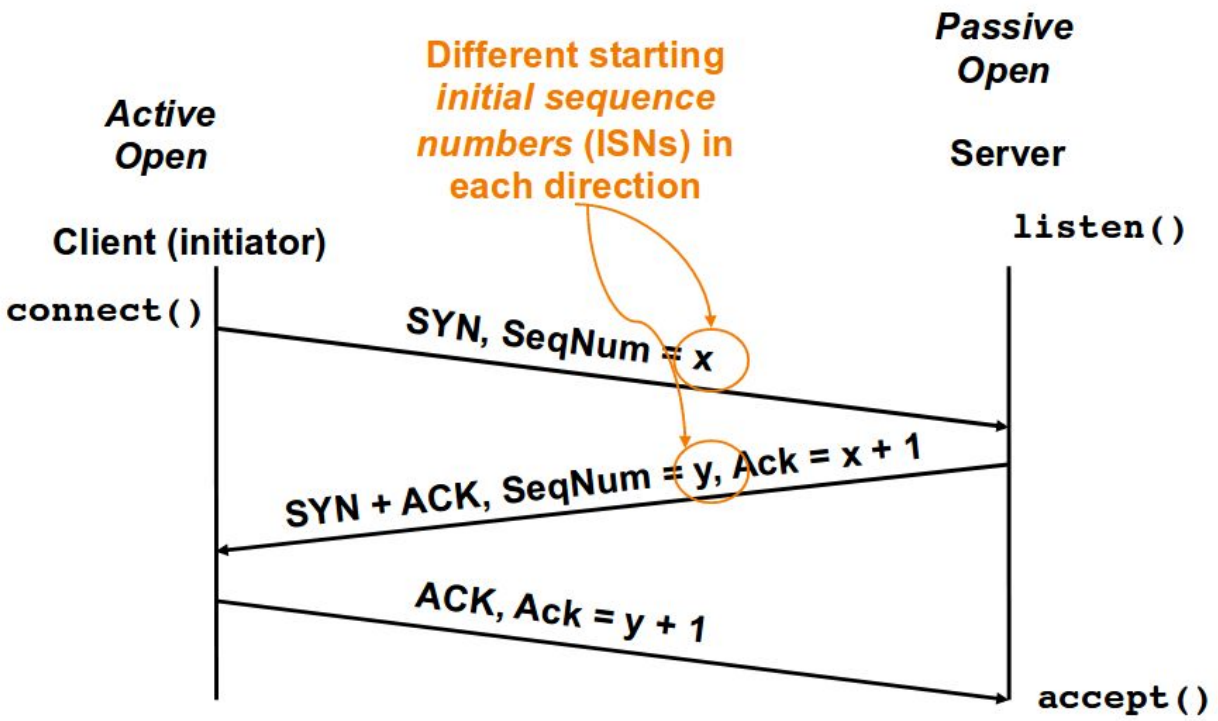
TCP: Transmission Control Protocol

Three-way handshake:

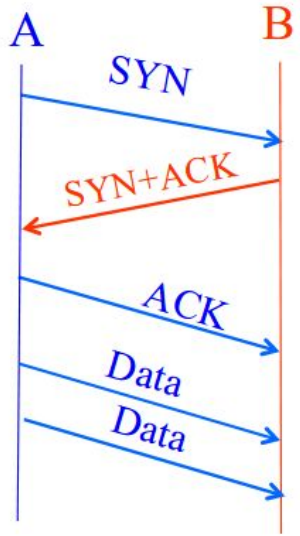
- Client sends: SYN, SEQ= x (arbitrary by the client)
- Server sends: SYN-ACK, SEQ= y (arbitrary by the server), ACK = x + 1
- Client sends: ACK, ACK = y + 1

The SEQ and ACK values swap depending on who’s sending the message. (You ACK the SEQ you receive.)

Timing Diagram: 3-Way Handshaking



Establishing a TCP Connection



Each host tells its *Initial Sequence Number* (ISN) to the other host.

(Spec says to pick based on local clock)

- Three-way handshake to establish connection
  - Host A sends a **SYN** (open; "synchronize sequence numbers") to host B
  - Host B returns a SYN acknowledgment (**SYN+ACK**)
  - Host A sends an **ACK** to acknowledge the SYN+ACK

TCP: Vulnerabilities

- TCP is not inherently secure
  - No confidentiality or integrity of data by default
- A malicious entity who knows the sequence numbers, port numbers, and IP addresses can spoof a connection
  - This means on-path attackers can easily inject data.
  - If an off-path attacker can guess this information, they can also inject data
- It also does not provide *availability*
  - TCP RST injection lets an attacker terminate connections

OSI Layer 7: Application Layer (aka L5)

- Application-to-application connection
- DHCP, HTTP, DNS, SSH, TLS, etc.

DHCP: Dynamic Host Configuration Protocol

When you first connect to a network, you need a few things:

- An IP address
- The DNS server’s IP address
- The gateway’s (router's) IP address
- [Not really important for 161: Netmask]

DHCP: Handshake

Client Discover: “I need configuration information!”

Server Offer: “Here’s some possible configurations: A, B, C”

Client Request: “I want to use configuration C”

Server Acknowledge: “Okay, I have given you config C”

This is all broadcast across the local network

DHCP: Spoofing

What happens when Alice accepts Mallory's configuration?

Mallory now controls Alice's:

- Gateway
- DNS (we talk about this later)

What can she do with this?

True/False Answers

If a laptop joining a WIFI network uses both DHCP and DNS, it will first use DHCP before using DNS.

When establishing a TCP connection, the client and the server engage in a three way handshake to determine the shared ISN they will both use for that connection.

.

Hosts that use DHCP on a wired networking technology such as Ethernet are protected against possible DHCP spoofing attacks.

ISPs are obligated to verify the IP source address on any traffic entering their network.

It’s difficult for an off-path attacker sending IP packets with a spoofed source to view the responses to those packets.

True/False Answers

If a laptop joining a WIFI network uses both DHCP and DNS, it will first use DHCP before using DNS. **True**

When establishing a TCP connection, the client and the server engage in a three way handshake to determine the shared ISN they will both use for that connection. **False.**

Hosts that use DHCP on a wired networking technology such as Ethernet are protected against possible DHCP spoofing attacks. **False.**

ISPs are obligated to verify the IP source address on any traffic entering their network. **False.**

It’s difficult for an off-path attacker sending IP packets with a spoofed source to view the responses to those packets. **True.**

(c) (24 points) Now that he has tested his WiFi access, Bob then tells Alice: "I want to buy that last muffin at the counter. Let me check if I have enough money in my bank account." Eve hears this and panics! She wants the last muffin too but is waiting for her friend Mallory to bring enough cash to buy it. She is now determined to somehow stop Bob from buying that last muffin by preventing him from checking his bank account. Through the corner of her eye, Eve sees Bob start to type `https://bank.com` in his browser URL bar ...

Describe two network attacks Eve can do to prevent Bob from checking his bank account. For each attack, describe clearly in one or two sentences how Eve performs the attack.

Attack #1:

Attack #2:

Solution:

Note that Eve cannot do an ARP or DHCP spoofing attack as Bob has already connected to the WiFi network, so already knows the IP and hardware addresses of the local network's gateway and DNS resolver. (This assumes that extraneous ARPs are not accepted by Bob's system. ARP spoofing is a viable answer for this problem if accompanied by specific mention of this consideration.)

1. TCP RST injection attack — Eve can sniff Bob's transmitted (and received) packets, so she can observe the sequence numbers of TCP packets. Thus, Eve can send a valid TCP RST packet to Bob's browser (or to the bank website), resetting the TCP connection.

2. DNS response spoofing — When Bob tries to load the bank website, his browser will generate a DNS request for the bank's domain. Eve can spoof a response with an incorrect answer, preventing Bob from loading the bank website properly.

3. DoS attack on either Bob's system or the coffee shop network. This can be done through various means, such as DNS amplification attacks directed at Bob.

Questions on Networking Basics?

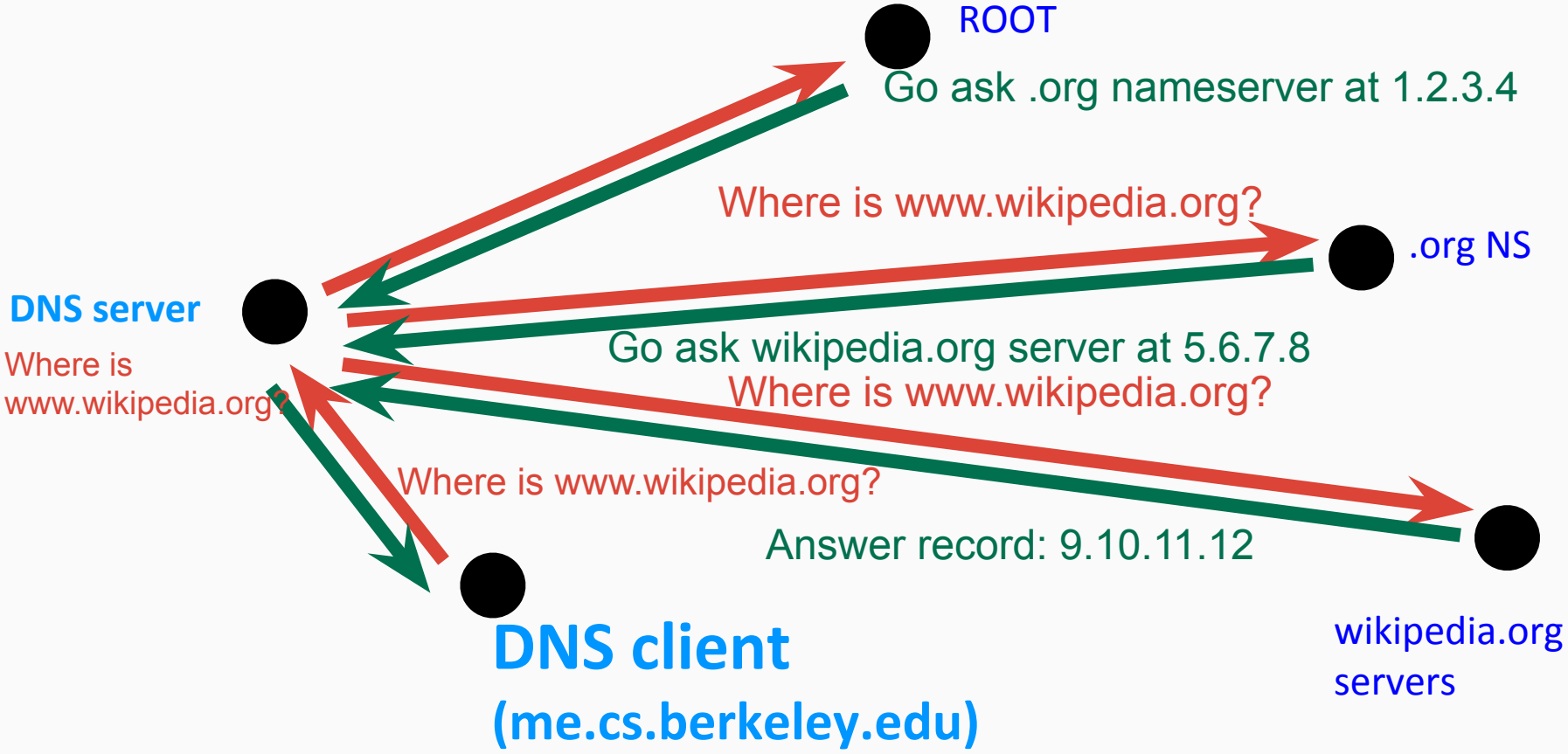
DNS

DNS: Domain Name System

Protocol for translating human-friendly domain names to a numerical address: www.berkeley.edu -> 52.10.99.247

Nameservers at least responsible for pointing to you in the right direction: .edu's name server might not know how to get to www.berkeley.edu, but it knows that berkeley.edu's name server knows and that name server's IP address.

DNS: Example (borrowed from cs168)



DNS: Resource Record (RR) Types

Each RR has a type and TTL (How long the record should be cached)

- A or AAAA: Matches a name w/ an IPv4 or IPv6 address

```
berkeley.edu      600     A       35.163.72.93
```

- NS: Gives you a name server responsible for a domain

```
edu.              172800    NS    a.edu-servers.net
```

A typical DNS response: “Hey I found what you’re asking for”

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 39459
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 2, ADDITIONAL: 3

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:      (What domain was queried for)
;berkeley.edu.                  IN      A

;; ANSWER SECTION:        (What the address of the queried domain is)
berkeley.edu.           16      IN      A       35.163.72.93

;; AUTHORITY SECTION:     (What the name servers responsible for berkeley.edu are)
berkeley.edu.           43860   IN      NS      adns2.berkeley.edu.
berkeley.edu.           43860   IN      NS      adns1.berkeley.edu.

;; ADDITIONAL SECTION:    (Possibly helpful additional information: Where the nameservers are)
adns1.berkeley.edu.     90569   IN      AAAA    2607:f140:ffff:fffe::3
adns2.berkeley.edu.     693     IN      A       128.32.136.14
adns2.berkeley.edu.     1598    IN      AAAA    2607:f140:ffff:fffe::e
```

Another typical DNS response: “I don’t know, but I can tell you who does and where they are”

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53577
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 2, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;berkeley.edu.                  IN      A

;; ANSWER SECTION:

;; AUTHORITY SECTION:
edu.                    172800  IN      NS      a.edu-servers.net.
edu.                    172800  IN      NS      b.edu-servers.net.

;; ADDITIONAL SECTION:
a.edu-servers.net.      172800  IN      A       192.5.6.30
b.edu-servers.net.      172800  IN      A       192.33.14.30
```

(What domain was queried for)

(What the address of the queried domain is. Note the lack of value)

(What the name servers responsible for .edu are)

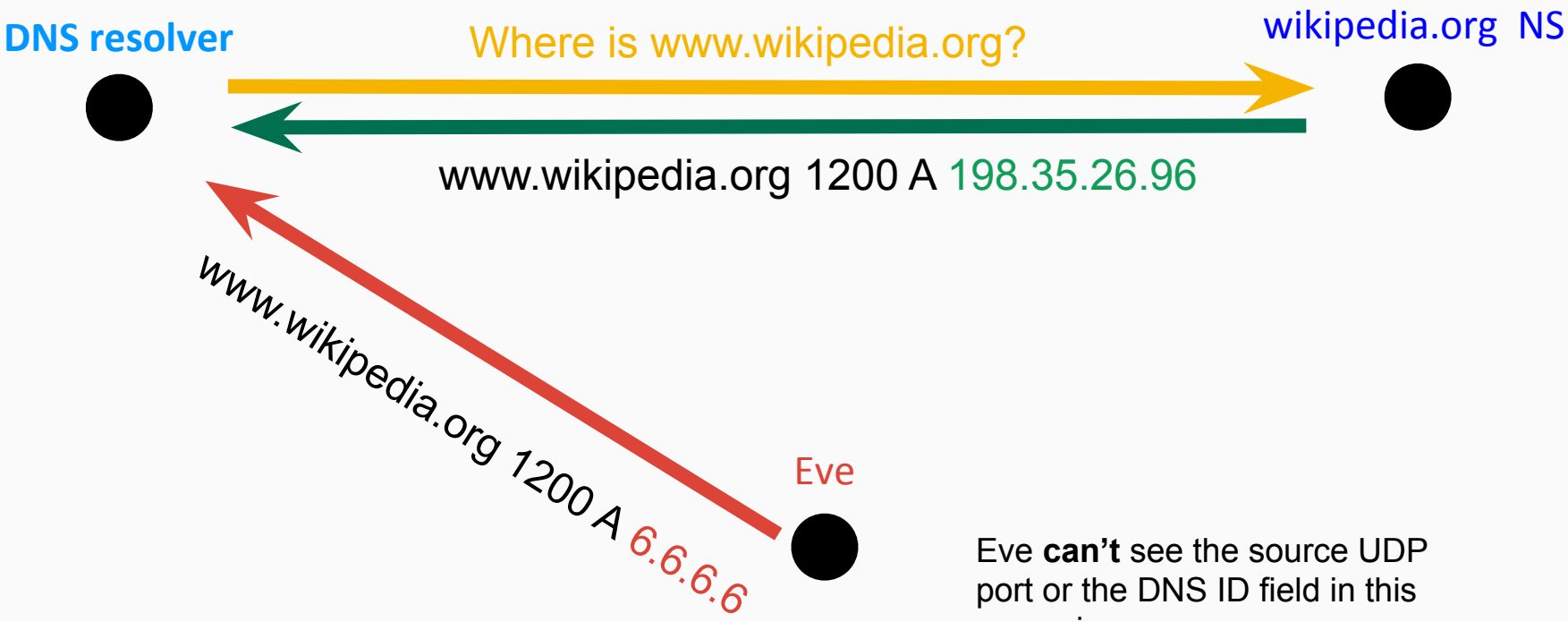
(Possibly helpful additional information: Where the nameservers are)

DNS: Vulnerabilities

- Malicious DNS server
  - .edu name server telling us berkeley.edu's name server is ns1.stanford.edu's IP address
    - Why doesn't bailiwick protect against this?
- On-path eavesdropper
  - Spoofing
- Off-path attacker
  - Blind-spoofing

What do spoofers need to guess correctly to be accepted by the victim?

DNS: Off-path spoofing



Eve **can't** see the source UDP port or the DNS ID field in this scenario.

DNS: Kaminsky Attack

- Once a resolver receives a response, it will **cache** the results. So not only does the spoofer need to beat the real result, once it loses, it can't try again for a bit.
- Kaminsky's attack subverts this by trying to poison the name server's IP address in the Additional section.

DNS: Kaminsky attack

If this attack succeeds, then any future query for any domain in wikipedia.org will be sent to Eve.

If it doesn't, it doesn't matter. The cache has not been altered. Just ask on another random subdomain.

True/False Questions

Source port randomization helps defend against an off-path attacker performing the Kaminsky DNS cache poisoning attack.

“Bailiwick” checks in modern DNS resolvers will prevent a malicious name server responsible for foo.com from using the Additional fields in its DNS responses to poison cache entries for bar.com.

In the event where the domain name to IP address binding changes, the DNS server responsible for the given domain name sends invalidation messages to clients in order to flush their mappings.

Randomizing the DNS query identifier prevents an on-path attacker from spoofing DNS responses.

True/False Answers

Source port randomization helps defend against an off-path attacker performing the Kaminsky DNS cache poisoning attack. **True.**

“Bailiwick” checks in modern DNS resolvers will prevent a malicious name server responsible for foo.com from using the Additional fields in its DNS responses to poison cache entries for bar.com. **True.**

In the event where the domain name to IP address binding changes, the DNS server responsible for the given domain name sends invalidation messages to clients in order to flush their mappings. **False.**

Randomizing the DNS query identifier prevents an on-path attacker from spoofing DNS responses. **False.**

Questions on DNS?

DNSSEC

Why DNSSEC?

DNS provides poor security, so DNSSEC was introduced as an alternative, guaranteeing these properties:

- **Integrity** (not confidentiality) via signatures
- Cache-ability of the signatures
- Extends and remains backwards compatible with DNS

Trust Delegation without DNSSEC

- Everyone has root's [.] nameserver IP hardcoded
- Remember requesting `www.berkeley.edu`:
  - Ask root
  - Root gives an answer:
    - `edu. NS a.edu-servers.net.`
      `a.edu-servers.net. A 192.5.6.30`
  - "Delegated" trust to .edu's nameservers
  - Repeat recursively
- **Problem**: no integrity, so response may have been modified

Key Idea: PKI

- DNSSEC provides a public-key infrastructure (PKI)
- Everyone has root's [.] nameserver IP and Key Signing Key (KSK)
- Remember requesting `www.berkeley.edu`:
  - `edu. NS a.edu-servers.net.`
  - `a.edu-servers.net. A 192.5.6.30`
  - `. DNSKEY <my Zone Signing Key (ZSK)>`
  - `. DNSKEY <my Key Signing Key (KSK)>`
  - `. RRSIG DNSKEY <SIGN(all DNSKEY records, my KSK)>`
  - `edu. DS <HASH(.edu's KSK)>`
  - `edu. RRSIG DS <SIGN(all DS records, my ZSK)>`
  - `. RRSIG NS <SIGN(all NS records, my ZSK)> [technically unnecessary]`
  - `. RRSIG A <SIGN(all A records, my ZSK)> [technically unnecessary]`

So in (Long) Summary

- If you trust my Key Signing Key (KSK)...
- I can give you my Zone Signing Key (ZSK) and KSK in DNSKEY
  - RRSIG DNSKEY contains signature on DNSKEY records with my KSK
  - So you now trust my ZSK
- I can give you the answer to your query (typically A and NS records)
  - RRSIG A and RRSIG NS are signatures on these records with my ZSK
- Finally, I give you a DS record (delegated signer)
  - DS contains a hash of my child zone's KSK
  - RRSIG DS contains signature on DS with my ZSK
  - So you now know what my child's KSK is
- If you trust me, I can **answer** your query and **delegate trust**

Practice Question

- Say you make a request for www.berkeley.edu's A record (IPv4 address), which is answered by .berkeley.edu's authoritative nameserver
- [Assume all caches empty.]
- While doing so, what nameservers do you contact?
- Which of the following record types do you need from each nameserver?
  - DNSKEY from:
  - DS records from:
  - RRSIG records (specify what records the RRSIGs sign):

Practice Answer

- Say you make a request for www.berkeley.edu's A record (IPv4 address), which is answered by .berkeley.edu's authoritative nameserver
- [Assume all caches empty.]
- While doing so, what nameservers do you contact?
  - Authoritative nameservers for . (root), .edu, .berkeley.edu
- Which of the following record types do you get from each nameserver?
  - DNSKEY from: . (root), .edu, .berkeley.edu
  - DS records from: . (root), .edu
  - RRSIG records (specify what records each RRSIG signs): RRSIG DNSKEY from root, .edu, .berkeley.edu. RRSIG DS from root, .edu. RRSIG A from root, .edu, .berkeley.edu. RRSIG NS from root, .edu. (A/NS RRSIGs for intermediates are technically unnecessary.)

Practice Question

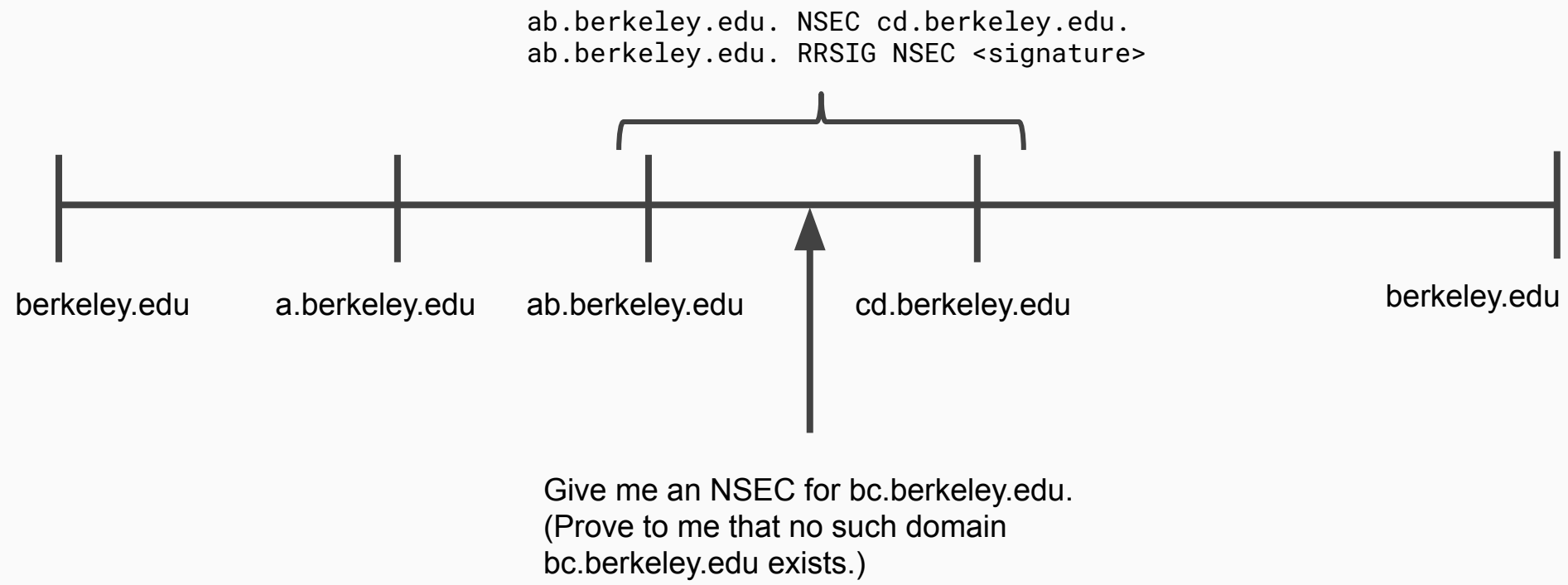
- Does DNSSEC allow for…
  - changing one's ZSK without contacting the parent nameserver?
  - changing one's KSK without contacting the parent nameserver?
  - changing one's A records without contacting the parent nameserver?
- Explain what (cryptographic) operations you need to do in order to update each of these.

Practice Answer

- Does DNSSEC allow for…
  - changing one's ZSK without contacting the parent nameserver? Yes.
    - You will need to update all RRSIGs on all records, and use your KSK private-key to update the RRSIG on DNSKEY records.
  - changing one's KSK without contacting the parent nameserver? No.
    - Parent's DS record will be wrong (hash of your KSK public-key).
    - Ask parent to update their DS record to the new hash. Then use your new KSK private-key to sign a new RRSIG for DNSKEY records.
  - changing one's A records without contacting the parent nameserver? Yes.
    - You will need to update RRSIG on A records by signing with your ZSK private-key.

NSEC

- Problem: how to assert a domain DOES NOT exist?
  - We could sign: "Domain X does not exist" using our ZSK
  - But that would require online cryptography ⇒ added latency
- NSEC provides *offline* authenticated denial to solve this
  - Can do all the crypto "in advance"


ab.berkeley.edu. NSEC cd.berkeley.edu.
ab.berkeley.edu. RRSIG NSEC <signature>
berkeley.edu
a.berkeley.edu
ab.berkeley.edu
cd.berkeley.edu
berkeley.edu
Give me an NSEC for bc.berkeley.edu.
(Prove to me that no such domain
bc.berkeley.edu exists.)

Enumerating NSEC

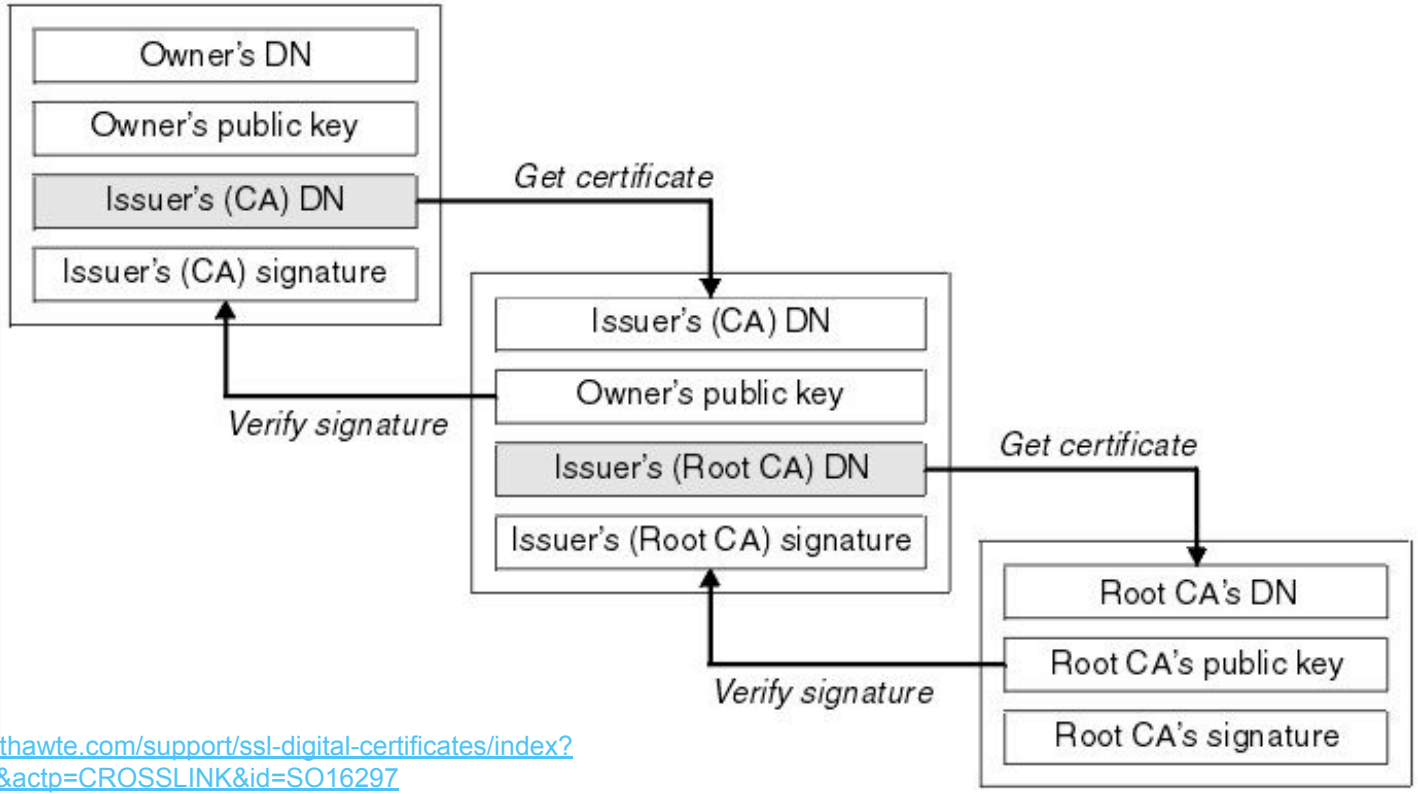
- [Perceived] Problem with NSEC: it allows an attacker to discover all subdomains of a site
  - Keep making requests for invalid domain names, and collect all NSEC records
  - Faster than trying to guess valid subdomains and making DNS requests for them
- NSEC**3** tries to solve this by using hashes
  - Attest that there are no domains whose HASH *H* falls between H(N) and H(M)
  - Not very effective against modern GPUs which can make many hash tries per second

Questions on DNSSEC?

TLS

SSL/TLS

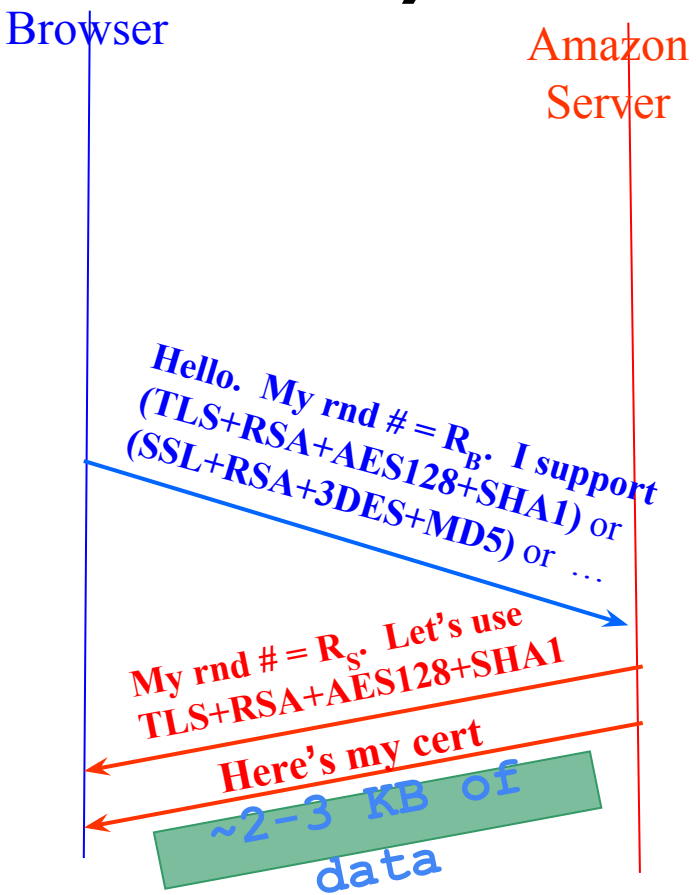
- Goal is to create a **secure channel**
  - Confidentiality: nobody can see the data communicated
  - Integrity: you know the data has not been changed in-transit
  - Timeliness: you are guaranteed that old data is not being *replayed* by a MITM
- DOES NOT PROVIDE:
  - ~~Anonymity~~: easy to figure out **who** is talking with whom
  - ~~Availability~~: TLS does not prevent DoS or messages being adversarially dropped
  - ~~Padding~~*: one can tell **how much** data is being communicated
  - ~~Non-repudiability~~: not possible to **prove** that this was the message you received



Source:
https://search.thawte.com/support/ssl-digital-certificates/index?page=content&actp=CROSSLINK&id=SO16297

HTTPS Connection (SSL / TLS)

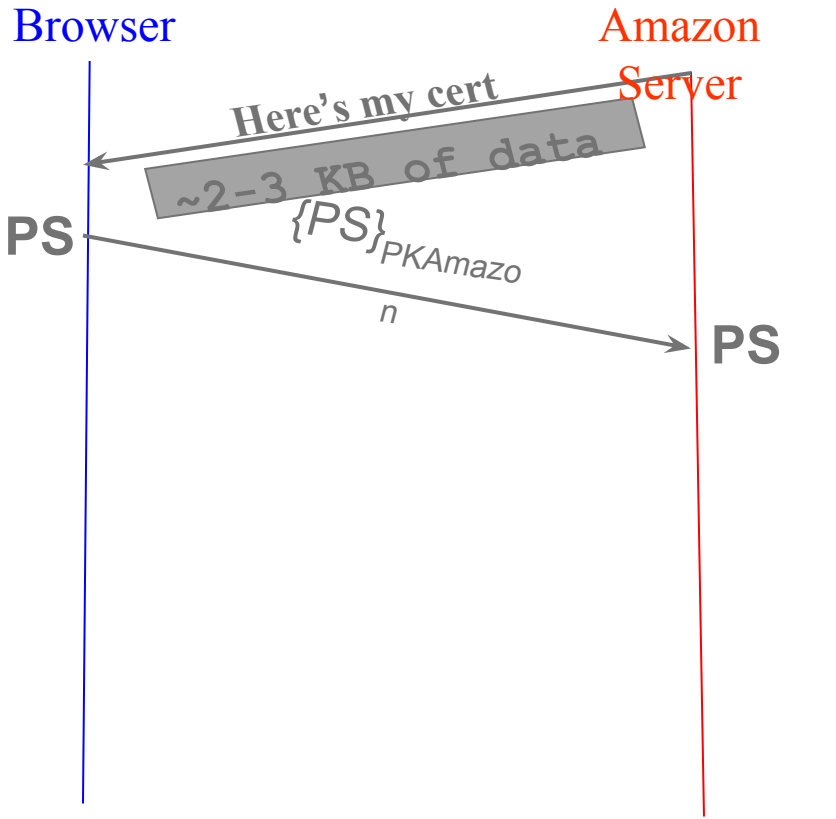
- Browser (client) connects via TCP to Amazon's **HTTPS** server
- Client picks 256-bit random number R_B, sends over list of crypto protocols it supports
- Server picks 256-bit random number R_S, selects protocols to use for this session
- Server sends over its certificate
- ***Client now validates cert***



PS via RSA

Q: Forward secrecy?
A: No forward secrecy because attacker can decrypt PS and knows R_B, and R_S and computes secrets

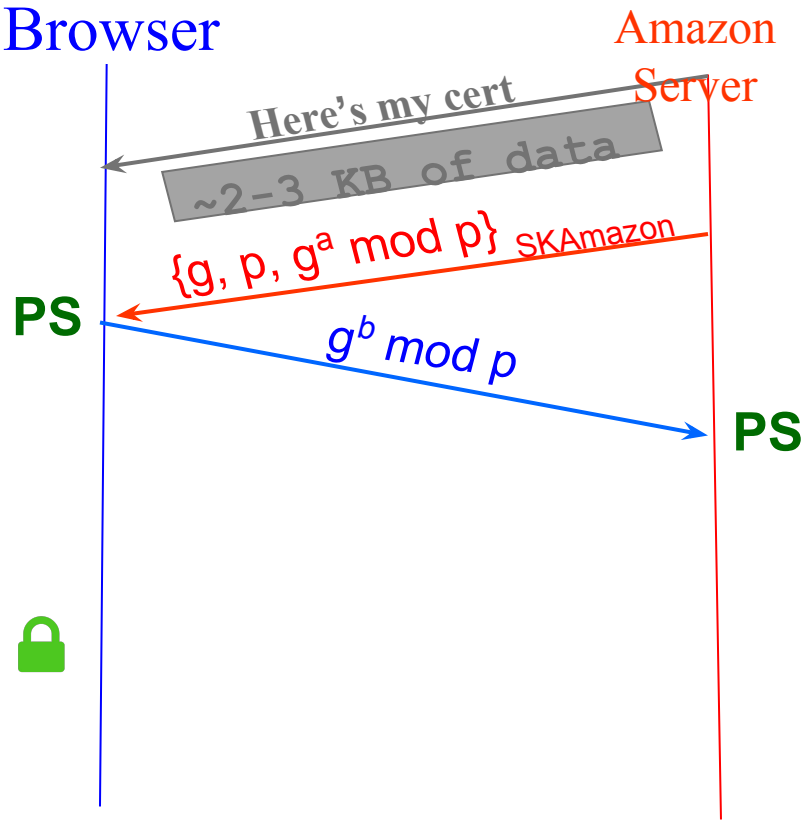
- Client generates premaster secret (PS), encrypts it with server's public key, and sends it to server
  - "If you really have the RSA private key, prove it to me by decrypting this PS."
- Both sides use PS (along with R_b, R_s) to derive **symmetric** keys.



PS via Diffie-Hellman

Q: Forward secrecy?
A: Has forward secrecy because shared secret never sent over the network! If attacker as SK_{Amazon}, cannot find a.

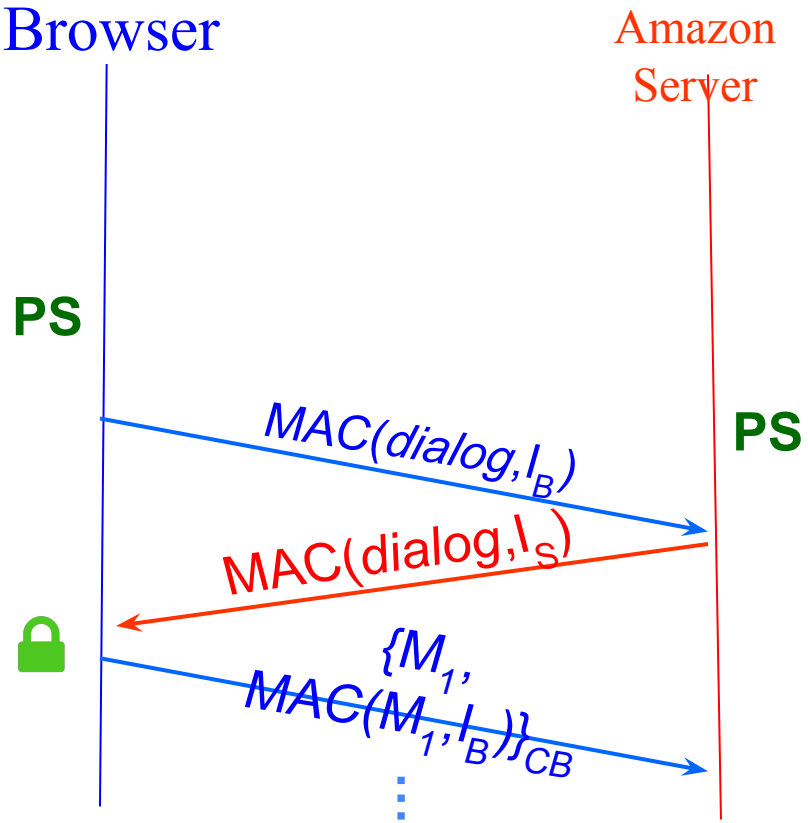
- For Diffie-Hellman, server generates random a, sends public params and g^a mod p
  - **Signed** with server's private key
- Browser verifies signature using PK from certificate
  - (provides security against classic Diffie-Hellman MITM)
- Browser generates random b, computes **PS** = g^{ab} mod p, sends g^b mod p to server
- Server also computes **PS** = g^{ab} mod p
- Both sides use PS to derive symmetric keys.



Symmetric Key Generation

- Both sides now have PS.
- Using PS, both sides derive four keys:
  - C_b - encryption key for messages from **b**rowser to the server
  - C_s - encryption key for messages from **s**erver to the browser
  - I_b, I_s - integrity keys (directional as above)

- Both sides now compute a MAC on the *dialog* (all messages from both sides).
  - If a MITM tampered with the earlier steps (say, R_b, R_s, cipher suite agreement, &c), then they will be detected now.



Spring 2017 - Final

(b) Thanks to strong cryptography, a TLS connection to your bank is secure even if your home router's TCP/IP implementation has a buffer overflow vulnerability.

○ **True** ○ **False**

Spring 2017 - Final

(b) Thanks to strong cryptography, a TLS connection to your bank is secure even if your home router's TCP/IP implementation has a buffer overflow vulnerability.

● **True** ○ **False**

- A key property of TLS is how it provides end-to-end security: two systems can communicate using TLS without having to trust any of the intermediaries that forward their traffic. Thus, even if an attacker completely pwns your home router, the worst they can do to you is deny you service to your bank.

Spring 2017 - Final

(a) (6 points) Suppose an attacker steals the private key of a website that uses TLS, and remains undetected. What can the attacker do using the private key? **Mark ALL that apply.**

- ○ Decrypt recorded past TLS sessions that used RSA key exchange.
- ○ Successfully perform a MITM attack on future TLS sessions.
- ○ Decrypt recorded past TLS sessions that used Diffie–Hellman key exchange.
- ○ None of these.

Spring 2017 - Final

(a) (6 points) Suppose an attacker steals the private key of a website that uses TLS, and remains undetected. What can the attacker do using the private key? **Mark ALL that apply.**

- ● Decrypt recorded past TLS sessions that used RSA key exchange.
- ● Successfully perform a MITM attack on future TLS sessions.
- ○ Decrypt recorded past TLS sessions that used Diffie–Hellman key exchange.
- ○ None of these.

- RSA key exchange offers no forward secrecy, so all past sessions can be decrypted
- With the private key, a MITM can forge the server's signature. The MITM can negotiate a separate TLS connection to client and server, masquerading as the server to the client and vice versa

(b) (24 points) Turns out that Brewed Awakening's network has no encryption. Alice warns Bob that its not safe to use this connection, but Bob disagrees. Bob connects to the WiFi, and tests that he has Internet connectivity by going to `https://kewlsocialnet.com`. It loads without issues. Bob says the Alice: "See, no problem! That access was totally safe!"

If Bob is correct and the access to `kewlsocialnet.com` was safe, explain why he is correct. If he is not correct, provide a network attack against Bob.

(b) (24 points) Turns out that Brewed Awakening's network has no encryption. Alice warns Bob that its not safe to use this connection, but Bob disagrees. Bob connects to the WiFi, and tests that he has Internet connectivity by going to `https://kewlsocialnet.com`. It loads without issues. Bob says the Alice: "See, no problem! That access was totally safe!"

If Bob is correct and the access to `kewlsocialnet.com` was safe, explain why he is correct. If he is not correct, provide a network attack against Bob.

Solution: Bob is correct.

Bob is visiting an HTTPS website, which uses TLS to provide an end-to-end secure channel. As Bob's browser did not encounter any certificate warnings, then unless there's been a CA breach or some other CA issue, the network connection has confidentiality, authentication, and integrity.

We allowed full credit for solutions that specified that Bob was incorrect and provided a valid approach for undermining his HTTPS connection to the site, including the threat of obtaining fraudulent certs from misbehaving CAs.

We allowed only partial credit for solutions that framed attacks that would work in the situation if TLS did not provide all of the strong security properties that it does. These solutions received more credit if they clearly stated that the attack is relevant for Bob's *subsequent* connections, rather than his test connection. These solutions received less credit if they were simply stating that because the WiFi network is unencrypted, an attacker could read Bob's private information, since use of HTTPS prevents that.

TLS Limitations/Issues

- The system requires us to trust ALL Certificate Authorities
- Certificate management is complicated
- TLS can’t protect against logical errors on hosts.

Questions on TLS?

WPA2

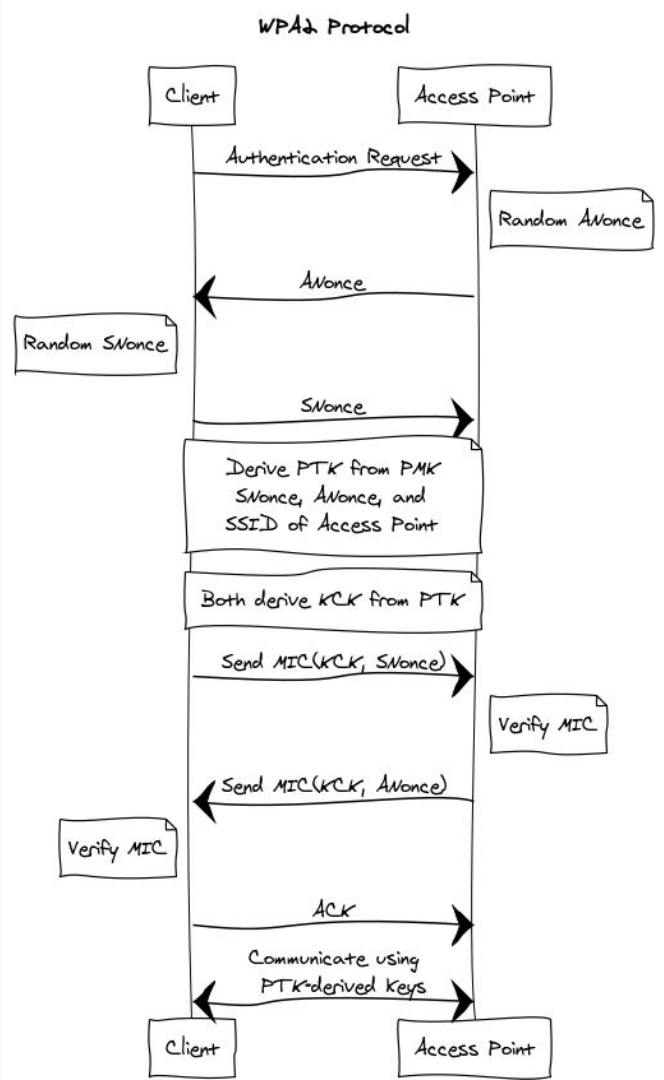
Local security

- Goal is to secure Layer 2 (between local hosts) communication
- Two major types:
  - WPA2-PSK: everyone has a shared "WiFi password", used to derive keys to communicate
    - Anyone who knows the WiFi password can see your traffic.
  - WPA2-Enterprise: everyone has a different username and WiFi password
    - Like AirBears2
    - **Main advantage:** other people cannot derive your *keys*, because they do not have your password!

WPA2-PSK is like TLS

They're basically the same:

- Use of nonces to prevent replay attacks
- No need for certificates or to agree on a PS here -- the WiFi password / PMK is sufficient for shared knowledge
- MIC(KCK, nonces) is like the MAC on the dialogues for TLS



Problem 6 ***Coffee Shop Worries*** **(54 points)**

Alice and Bob just arrived at Brewed Awakening, the local coffee shop. Eve is already there, enjoying a cup of tea.

(a) (6 points) Alice wants to connect to Brewed Awakening's WiFi network. Under which protocols would her connections be safe from sniffing attacks by other coffee shop visitors, such as Eve? **Mark all that apply.**

not in scope, but if you care: WEP is like WPA but it sucks

- O WEP
- O WPA2 - Enterprise mode
- O WPA2 - Personal mode
- O None of these

Problem 6 ***Coffee Shop Worries*** **(54 points)**

Alice and Bob just arrived at Brewed Awakening, the local coffee shop. Eve is already there, enjoying a cup of tea.

(a) (6 points) Alice wants to connect to Brewed Awakening's WiFi network. Under which protocols would her connections be safe from sniffing attacks by other coffee shop visitors, such as Eve? **Mark all that apply.**

- ○ WEP
- ● WPA2 - Enterprise mode
- ○ WPA2 - Personal mode
- ○ None of these

Solution: Only "WPA2 - Enterprise mode" provides per-connection secret keys with the WiFi access point to secure each connection separately.

Questions on WPA2?

Web Security

Web Security

"I asked my grad students for a joke about web security, and their response was: 'Isn’t web security already a joke?' " - Raluca (Spring 2018)

HTML

A language to create structured documents
One can embed images, objects, or create interactive forms

index.html

```
<html>
    <body>
        <div>
            foo
            <a href="http://google.com">Go to Google!</a>
        </div>
        <form>
            <input type="text" />
            <input type="radio" />
            <input type="checkbox" />
        </form>
    </body>
</html>
```

CSS (Cascading Style Sheets)

Style sheet language used for describing the presentation of a document

index.css

```
p.serif {
font-family: "Times New Roman", Times, serif;
}
p.sansserif {
font-family: Arial, Helvetica, sans-serif;
}
```

Javascript

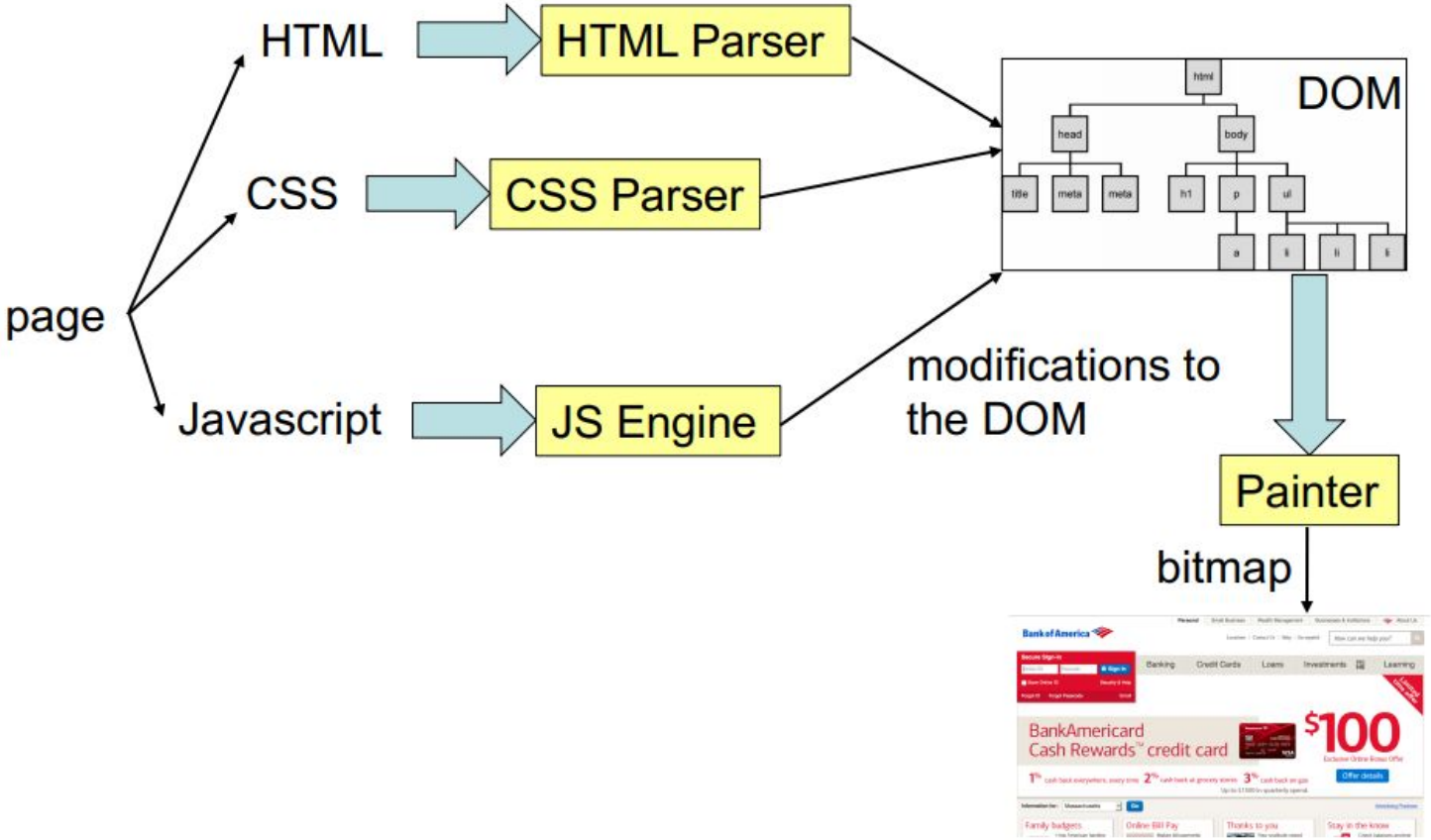
Programming language used to manipulate web pages. It is a high-level, untyped and interpreted language with support for objects.

Supported by all web browsers

```
<script>
function myFunction() {
document.getElementById("demo").innerHTML = ”Text changed.";
}
</script>
```

Very powerful!

Page rendering



What can you do with Javascript?

Change HTML content, images, style of elements, hide elements, unhide elements, change cursor, read and change cookies...

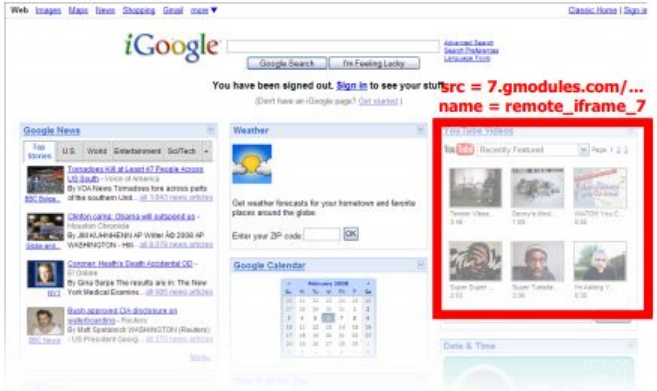
Read cookie with JS:

```
var x = document.cookie;
```

Change cookie with JS:

```
document.cookie = "username=John Smith; expires=Thu, 18
Dec 2013 12:00:00 UTC; path=/";
```

Frames



- Modularity
  - Brings together content from multiple sources
  - Client-side aggregation
- Delegation
  - Frame can draw only on its own rectangle

Outer page can specify only sizing and placement of the frame in the outer page.

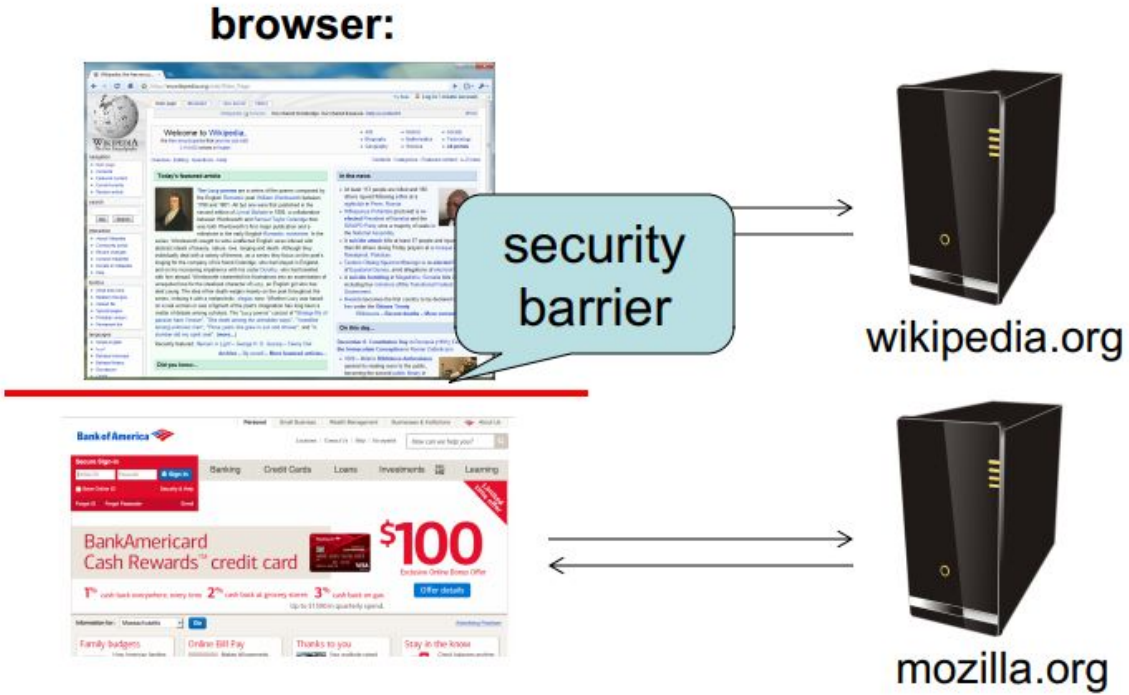
We **can** load frames from different websites!

Frame isolation: Outer page cannot change contents of inner page. Inner page cannot change contents of outer page.

Slide from Dan Boneh

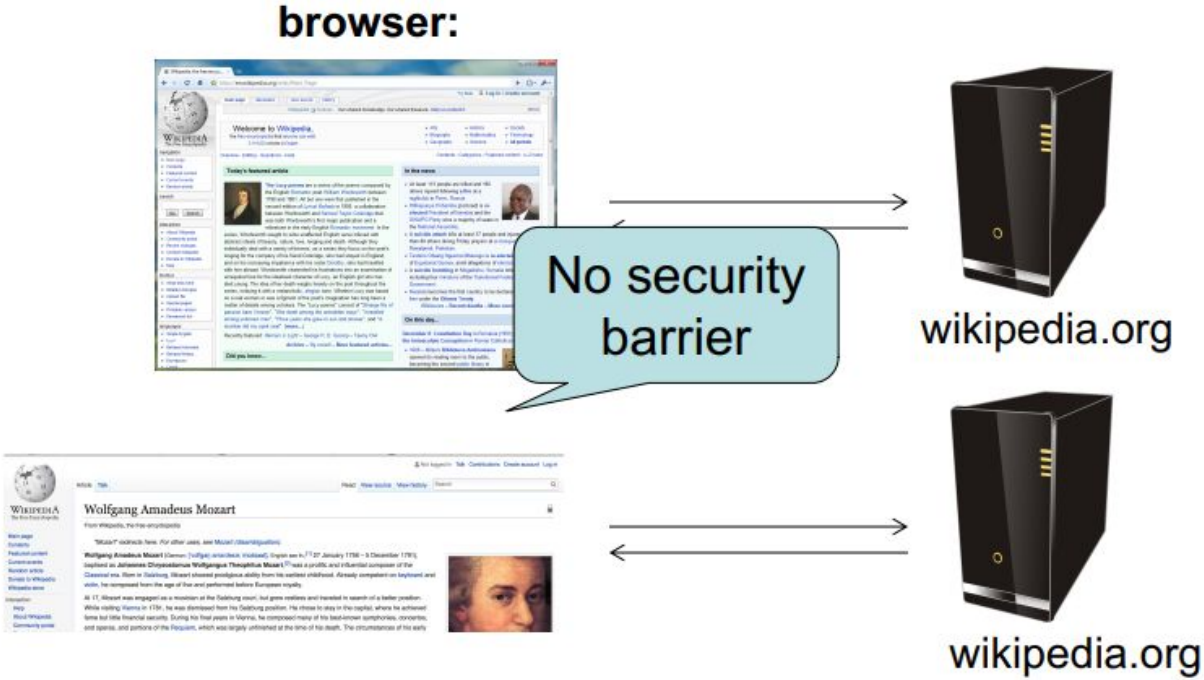
Same-origin policy

- Each site in the browser is isolated from all others



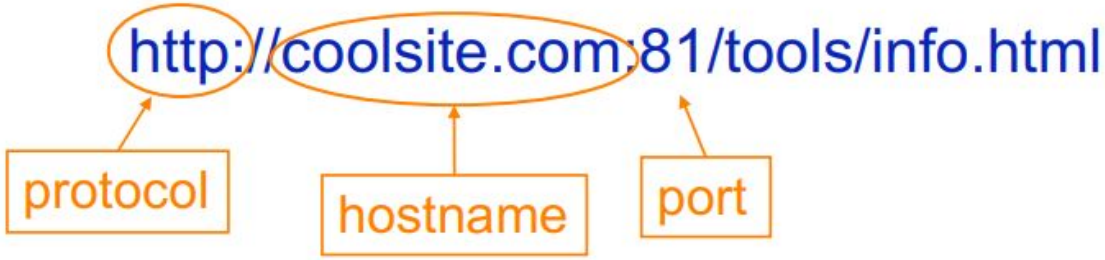
Same-origin policy

- Multiple pages from the same site are not isolated



Origin

- Granularity of protection for same origin policy
- Origin = protocol + hostname + port

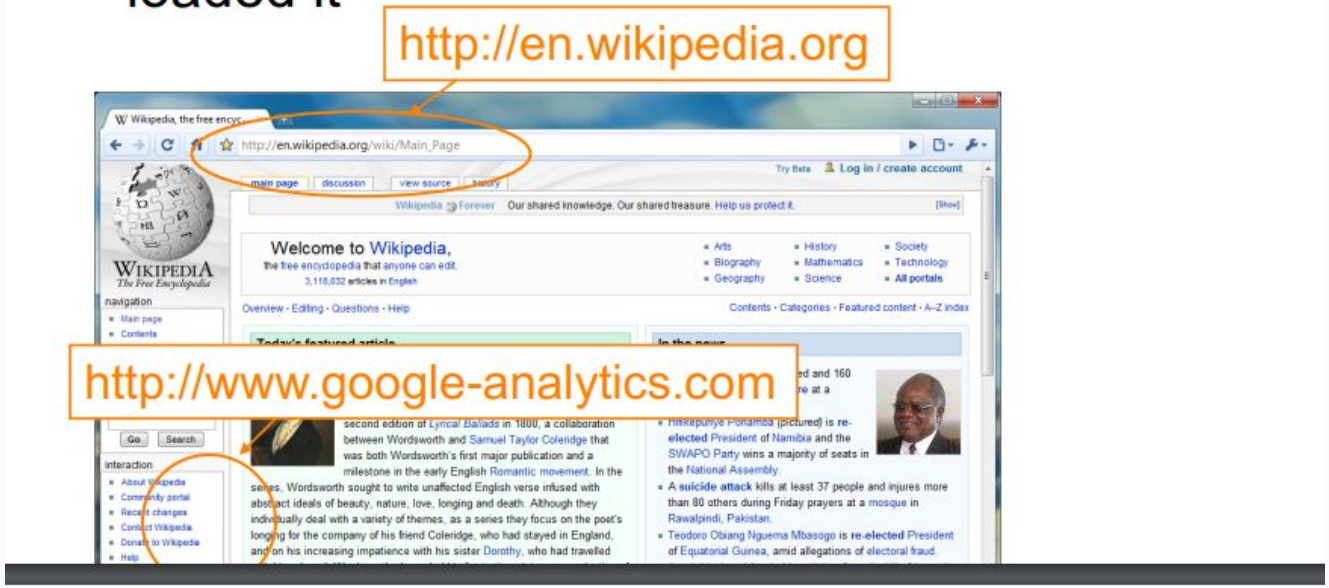


- It is **string matching**! If these match, it is same origin, else it is not. Even though in some cases, it is logically the same origin, if there is no match, it is not

Same-origin policy

This is an important exception to the same-origin policy!

- The origin of a page is derived from the URL it was loaded from
- Javascript runs with the origin of the page that loaded it



Origins of other components

- **<img src=“”>** the image is “copied” from the remote server into the new page so it has the origin of the embedding page (like JS) and not of the remote origin
- iframe: origin of the URL from which the iframe is served, and not the loading website.

Can Access?

Originating	Trying to Access
http://wikipedia.org/a/	http://wikipedia.org/b/
http://wikipedia.org/	http://www.wikipedia.org/
http://wikipedia.org/	https://wikipedia.org/
http://wikipedia.org:81/	http://wikipedia.org:82/
http://wikipedia.org:81/	http://wikipedia.org/

Can Access?

Originating		Trying to Access
http://wikipedia.org/a/	=	http://wikipedia.org/b/
http://wikipedia.org/	≠	http://www.wikipedia.org/
http://wikipedia.org/	≠	https://wikipedia.org/
http://wikipedia.org:81/	≠	http://wikipedia.org:82/
http://wikipedia.org:81/	≠	http://wikipedia.org/

Cookies!

- Cookies are **name-value** pairs.
- Server can ask browser to store a cookie for the current site
- Whenever the browser visits the site, it sends the corresponding cookies
  - We typically use this to maintain login state (through session cookies)
- There are cookie *flags*:
  - Domain: which domain this cookie is for (viz., **www.google.com**/foo/bar)
  - Path: which path this cookie is for (viz., www.google.com/**foo/bar**)
  - secure: indicates the browser should only send this cookie via HTTPS
  - HttpOnly: indicates the browser should stop accesses to this cookie through Javascript
- Cookie policy (not same-origin policy) dictates cookie security

When browser sets a cookie

Browser allows setting a cookie only if:

- cookie domain is a domain-suffix of the URL domain
  - i.e., you can set cookies for **less specific** domains
  - also, do not allow TLDs like **.com** or **.org** for security & privacy reasons
- cookie path can be anything
- [protocol = HTTPS if cookie has "secure" flag set]

When browser sends a cookie

Browser sends all cookies in URL scope:

- cookie domain is a domain-suffix of the URL domain
- cookie path is a prefix of the URL path
- [protocol = HTTPS if cookie has "secure" flag set]

A web server on foo.example.com tries to set a cookie with the following domains. Will the cookie be set? If so, which domains will receive the cookie? (Some rows are filled in as examples.)

Domain	**Where Cookie Would Be SENT (if set)**
(value omitted)	foo.example.com (exact)
bar.foo.example.com	?
foo.example.com	foo.example.com and *.foo.example.com
baz.example.com	?
example.com	?
ample.com	Cookie not set! Different domain.
.com	?

A web server on foo.example.com tries to set a cookie with the following domains. Will the cookie be set? If so, which domains will receive the cookie? (Some rows are filled in as examples.)

Domain	**Where Cookie Would Be SENT (if set)**
(value omitted)	foo.example.com (exact)
bar.foo.example.com	Cookie not set! Domain too specific
foo.example.com	foo.example.com and *.foo.example.com
baz.example.com	Cookie not set! Domain mismatch
example.com	example.com and *.example.com
ample.com	Cookie not set! Different domain.
.com	Cookie not set! Too broad.

Cross-Site Scripting (XSS)

- Attacker inserts malicious script into another site's webpage that will be viewed by a user.
  - Script runs in user's browser **and bypasses same-origin policy!**
- For example, if there was an XSS in Piazza, an attacker could make a user run Javascript *as if* Piazza had sent it
  - Since Javascript is very powerful, this can perform actions as the victim user

Two types of XSS

Stored XSS:

- Attacker leaves Javascript lying around on a benign web service for victim to load
- *Think*: Facebook Wall Post.
  - Attacker makes a post like `<script>alert(1)</script>`
  - **IF** the site does not escape HTML before showing it back to the user, then this could be a stored XSS.

Reflected XSS:

- Attacker gets user to load a specially-crafted URL with Javascript encoded in the URL
- *Think*: Google's search results
  - To search: google.com/?query=X
  - Outputs "No results for *X*."
  - Parts of URL appear in output
  - **IF** Google did not do the proper escaping, then this could be a reflected XSS!

Cross Site Request Forgery (CSRF)

- Example:
  - User logs in to bank.com
    - Session cookie remains in browser state
  - User visits malicious site containing:

```
<form name=F action=http://bank.com/BillPay.php>
<input name=recipient value=badguy> …
<script> document.F.submit(); </script>
```

  - Browser sends user auth cookie with request
    - Transaction will be fulfilled
- Problem:
  - cookie auth is insufficient when side effects occur

CSRF Defenses

- Secret Validation Token


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

- Referer Validation


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

- Others (e.g., custom HTTP Header)


```
X-Requested-By: XMLHttpRequest
```

Conceptual questions

1) Summarize same-origin policy.

2) Does same-origin policy protect against an XSS attack? Why or why not?

3) Does setting the secure flag (https only) on a cookie protect against a CSRF attack? Why or why not?

4) Compare and contrast reflected XSS attacks and CSRF attacks.

Conceptual answers

1) A policy enforced by the browser that isolates the resources of an origin from another, where an origin is defined by protocol+host+port.

2) Same-origin policy does not protect against XSS because the attack is carried within the same origin.

3) Setting the secure flag does not protect against a CSRF attack because in this attack, the browser automatically attaches the cookie to the request (as long as the attacker used a https request).

4) Similar in the sense: both rely on making the user's browser load a URL. Different: a reflected XSS executes Javascript on the victim's computer, a CSRF attack tries to fool the server into performing an operation that they don't intend.

Good Luck!