Lec II

* protocol : an agreement on how to communicate
  - syntax : how a communication is specified & structed
  - semantics : what a communication means.

wire → local network → wide area network.
(router)

1° MAC address : LAN中识别设备

2° Hop-to-Hop vs. End-to-End Layers

3° IP Addresses : WAN中识别设备

* Layering.

7	Application
4	Transport
3	Network
2	Link
1	physical

} different for each Internet hop.

1° break communications up into packets ⇒ unreliable, self-contained.

2° receiving node put it back together ⇒ best-effort

* IP : packet perspective.

8-bit protocol : Specify how to interpret the start of the Payload. which is the header of a Transport protocol (TCP/UDP)

1° build based on layers

2°. msg sent as individual packets.

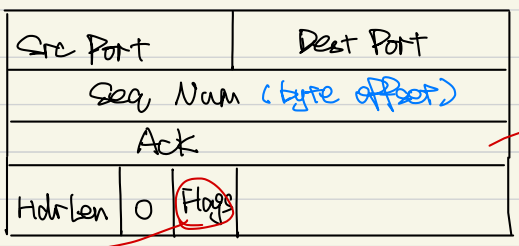
- smart builds on "end-to-end", not "hop-by-hop"

* TCP : connection perspective

— 2 byte stream, bidirection

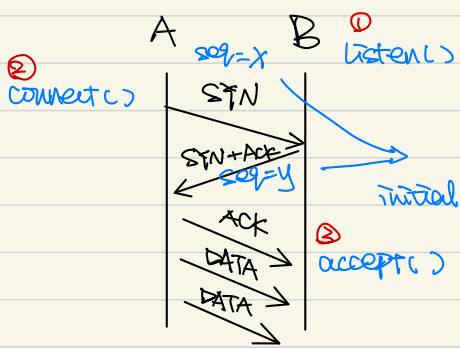
— reliable, in-order

— (IP src, Ip dest, Port src, Port dest) → uniquely identify a TCP connection. (port 80 : HTTP)



sender sends N bytes ending at seq S. then ack for it will be S+N.

SYN, ACK, RST



i) SYN

ii) SYN+Ack

iii) Ack

* UDP (User Datagram Protocol)

— faster, no handshape [skype]

Networking

Layer	Protocols
7. Application	Web Security
4.5 Secure transport	TLS
4. Transport	TCP, UDP
3. Internet	IP
2. Link	
1. Physical	

Extra Protocols.

	Protocols
connect for the first time	DHCP
Convert hostname to IP address	DNS, DNSSEC

1. join the wireless network
2. configure connection
   (IP address, IP address of the router, dns server)
   ↑ config
   access point running server
3. find the address of google.com
   ↑ UDP
   → DNS resolver
4. connect to google.com server
   → SYN
   ← SYN+ACK
   → ACK
5. establish a secure connection using TLS (https)
6. using http inside the TLS channel

Lec 12 Networking Attacks: TCP and DHCP

1° link layer threats

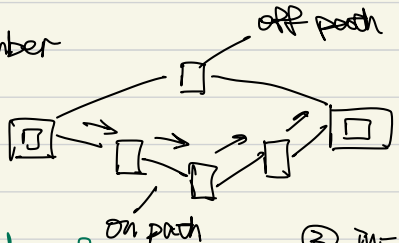
- confidentiality: eavesdropping (aka sniffing): wireshark, wifi, ethernet
- integrity: injection of spoofed packets: lie about source addr
- availability: delete legit packets (e.g. jamming)

spoofed packet on TCP connection

- Problem, sequence number

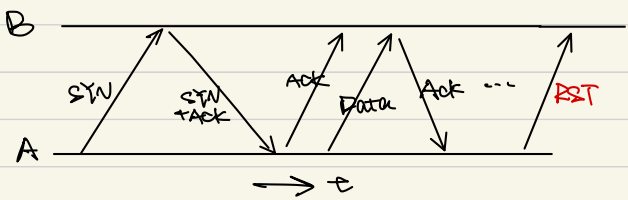
① on path attacker
- can see victim's traffic

② off path attacker
- have to resort to blind spoofing
- often must guess/infer header values
- can brute force: 16 bits $\rightarrow 2^{16} \approx 65536$



③ in-path attacker
both eavesdrop & drop the packet

2° TCP RST injection



TCP Packets with RST flag to B and sequence number etc, connection is terminated.

- man in the middle: modify flag (can observe & modify)
- on-path attacker: spoofed RST packet (can observe)
- off-path attacker: can't

ex: another host in the same ethernet ⇒ on-path attacker
⇒ × man-in-the-middle

3° TCP Data injection.

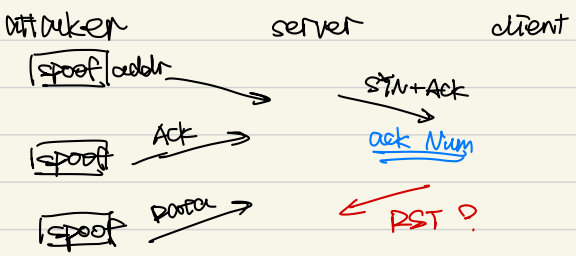
- If attacker knows ports & sequence numbers

↑ ← inject data into any TCP connect.

connection hijacking : take over an already-established connection!

4° TCP off-path attack. Blind hijacking

- IP addressed-based authentication
  - spoof src addr



Note #1 : client → RST [SYN+ACK 就被送給真正 client，其可能會RST]

Note #2 : don't know server ack seq number

Initial Sequence Number (ISN) : based on local clock.

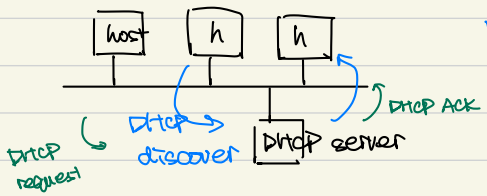
- non-spoofed connection first, and see what server used for ISN.

- make ISN random

5° host name v.s. IP address

- Host names
  - ex: www.···.com
  - little information about location
- IP addr

6° DHCP & DNS



DHCP offer: ① IP addr. ② DNS server. ③ "gateway router" and ④ how long client can have these

— threats
: malicious offer < malicious dns server / malicious gateway router ⇒ MITM

— fix
: hardcode dns server

7° take away

- Broadcast protocols at risk of local attacker spoofing [MITM]
- systems are particularly vulnerable because lack of a trusted foundation to build upon

① blind hijacking: off-path attacker 猜测端口、序列号后注入。

② blind spoofing: off-path attacker 猜测序列号、伪造新链接。

Lec 13 TLS

I° Intro — HTTP over TLS (HTTPS)

* Applying crypto technology in practice

— two abstractions

① "Sealed blob": Data that is encrypted and authenticated under a particular key. ("object security")

② Secure channel: Communication channel that can't be eavesdropped on or tampered with ("channel security")

TLS ↗

* end-to-end: even if one part of the communication is compromised, data being sent cannot be eavesdropped or modified.

— Dealing with threats:

① eavesdropping: Encryption

② Manipulation (Injection, MITM): Integrity (use of a MAC). replay protection

③ Impersonation: Signatures.

× not about availability

* use

① https

② secure mails sent between server (STARTTLS)

③ virtual private networks

* TLS = Transport Layer security

* Secure channel for applications that use TCP

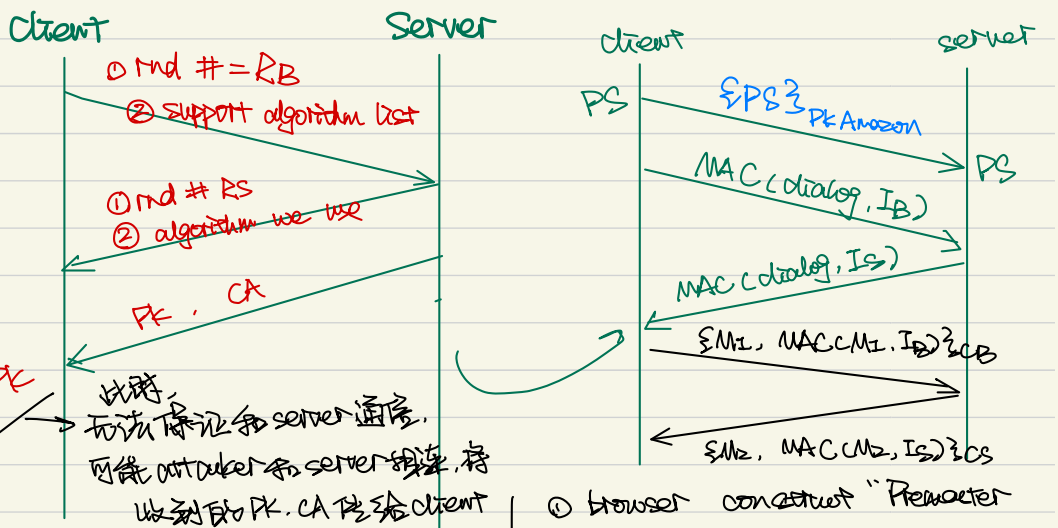
— Secure = encryption / confidentiality + integrity + authentication (of server, but not client)

2° RSA TLS

i) browser(client) connects to a server which returns CA

ii) exchange symmetric key using TLS

iii) send encrypted & authenticated traffic to each other



□ Why R_B and R_S.

— defend replay attack. R_B & R_S are random ⇒ different symm. key / integrity key

one side replay, another side different random key

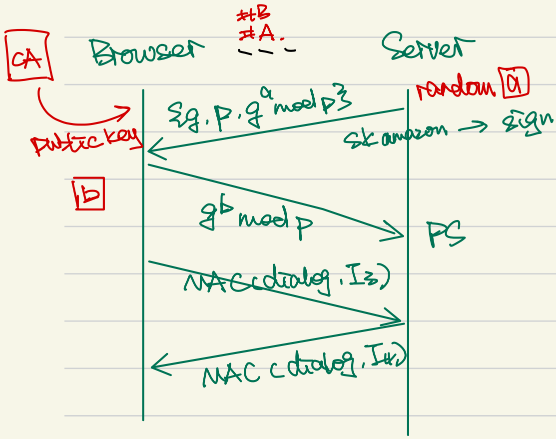
① browser constructs "Premaster Secret" PS

② using PS, R_B and R_S to derive symm. cipher keys (C_B, C_S). and MAC integrity kes (I_B, I_S).

③ Brower & server exchange MACs computed over entire dialog so far

○ 另一种生成PS的方式

3° Diffie Hellman TLS



① server generates random a, sends public params and $g^a \bmod p$

② Browser verifies signature using PK from CA.

- public key from CA to verify signature ⇒ defend MITM

③ Browser generates random b, computes $PS = g^{ab} \bmod p$, sends $g^b \bmod p$.

④ server also computes $PS = g^{ab} \bmod p$

⑤ using PS, R_B and R_S to derive symm. cipher keys (C_B, C_S). and MAC integrity kes (I_B, I_S).

○ 简化版,TLS握序列号.

4° DH v. RSA

☆ Forward secrecy:

- Forward secrecy: If attacker steals long term secret key of server, SK_{Amazon}, should not be able to read past conversations (cannot compromise past session keys (C_B, C_S) & (I_B, I_S))

— RSA: decrypt $\{PS\}_{PK_{Amazon}}$ → get PS → get all keys

— DH: X SK_{amazon} is used to sign the info. not hide data ⇒ can't know PS ⇒ cannot know all keys.

* TLS limitations:

1° TCP-level denial of service
- SYN flooding
- RST injection

2° server-side coding / logic flaws

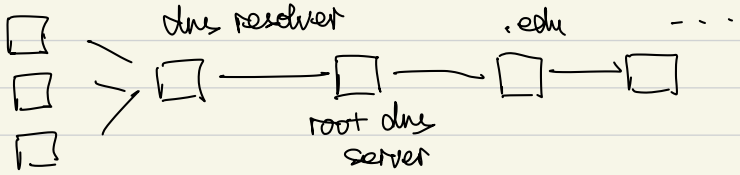
3° browser coding / logic flaws

4° user flaws
- weak password
- Phishing

Lec 14. DNS ⇒ performance critical & secure.

* look up: root (.) name server, the .org name server, the ocslab1.org name server

* domain names: human friendly names to identify services
  - www.google.com
    - .com as TLD (Top Level domain) is a subdomain of root
    - google.com is a subdomain of com
    - www.google.com is a subdomain of google.com



- use cache → performance

1° DNS Records

DNS Protocol

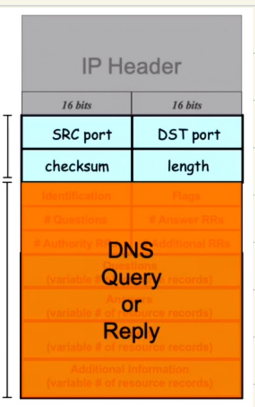
Lightweight exchange of *query* and *reply* messages, both with same message format

UDP Header

Primarily uses UDP for its transport protocol, which is what we'll assume

UDP Payload

Frequently, both clients and servers use port 53

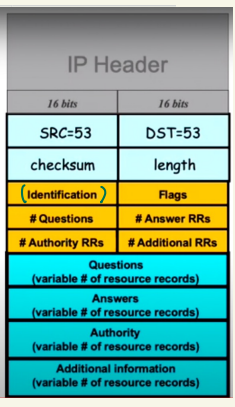


SU20 Lecture 14.5 - DNS Records

DNS Protocol, cont.

Message header:

- Identification: 16 bit # for query, reply to query uses same #
- Along with repeating the Question and providing Answer(s), replies can include **"Authority"** (name server responsible for answer) and **"Additional"** (info client is likely to look up soon anyway)
- Each *Resource Record* has a Time To Live (in seconds) for **caching** *(not shown)*



① id field: randomly selected per query and used to match requests to responses

② next 16 bits for flags: specify whether a msg is a query or a response. / whether a query was successful

③ # questions: always 1

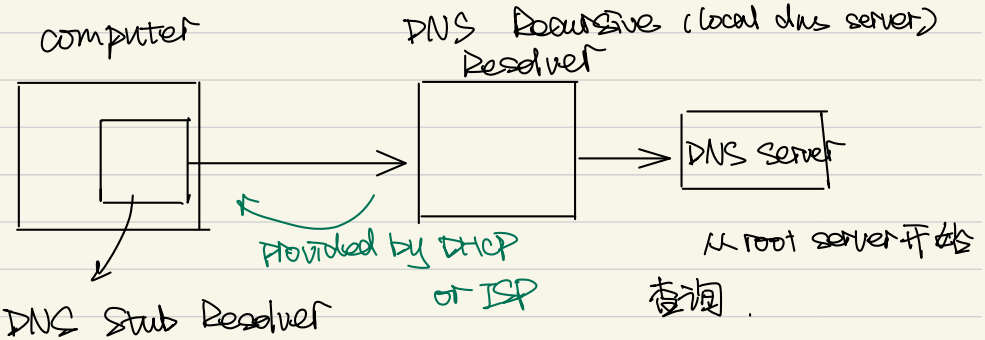
④
- # Answer RRs
- # Authority RRs
- # Additional RRs

⑤ actual content of the DNS query / response.

⇒ structured as a set of RRs. each RR is a key-value pair with an associated type.

key: <Name, Class, Type>
- Class: IN for Internet
- Type: specify the record type

record: <TTL, Value>
- TTL: Time-to-live: how long, in seconds, the record can be cached)



```
$ dig +norecurse eecs.berkeley.edu @198.41.0.4

;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26114
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 13, ADDITIONAL: 27

;; QUESTION SECTION:
;eecs.berkeley.edu.          IN   A

;; AUTHORITY SECTION:
edu.                172800   IN   NS   a.edu-servers.net.
edu.                172800   IN   NS   b.edu-servers.net.
edu.                172800   IN   NS   c.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
c.edu-servers.net.  172800   IN   A    192.26.92.30
...
```

TTL Internet Type.

* 向 root server 发送 query，得 answer

Type:

① NS, map from a DNS zone to a name server

② A : map from names to IP addresses

.com 顶级管理 .com 子域名的映射

DNS Security:

DNS ✗ confidentiality for queries.

* dns cache poisoning

Sol: — only accept additional records unless they're for the domain were looking up

e.g. looking up eecs.mit.edu ⇒ only accepts *.mit.edu

① Security risk #1: malicious DNS server

— malicious answer to DNS query

— used to used cache poisoning to fool us about the answer to other queries, now fixed.

To prevent any malicious name server from doing too much damage, resolvers use **bailiwick checking**, which only allows a name server to provide records under its domain. This means that the `berkeley.edu` name server can only provide records for `berkeley.edu` (not `stanford.edu`), the `.edu` name server can only provide records for `.edu` domains (not `google.com`), and the root name servers can provide records for anything.

② Security risk #2: on-path eavesdropper

— construct a answer using id in the query

③ Security risk #3: off-path attack

— blind spoofing, since don't know the id.

① force user visit web page under our control

⇒ know the ip address user are looking for

② guess id

— originally, id incremented in browser

— 注册域名, force user to search for it and get id'

⇒ id = id' + 1

——— randomize the identification

- attacker can send lots of replies
- if legit answer arrives, it's cached and no more opportunity to poison it.

④ Kaminsky Blind Spoofing Attack ⇒ query non-existent domains.

- Two key ideas:
  - Attacker can get around caching of legit replies by generating a series of different name lookups:

```
<img src="http://random1.google.com" …>
<img src="http://random2.google.com" …>
<img src="http://random3.google.com" …>
                    ...
<img src="http://randomN.google.com" …>
```

  - Trick victim into looking up a domain you don't care about, use **Additional** field to spoof the domain you do care about

— off-path: brute-force IDs.

— on-path: race with legit. answer

[will not be cached]

⇒ randomize the ports. → source port ✱ ⇒ dest port 通常一定.

— bypass bailiwick checking

⇒ make queries for random subdomains.

lec 14 DNSSEC

goal: ensure integrity of the DNS lookup results.

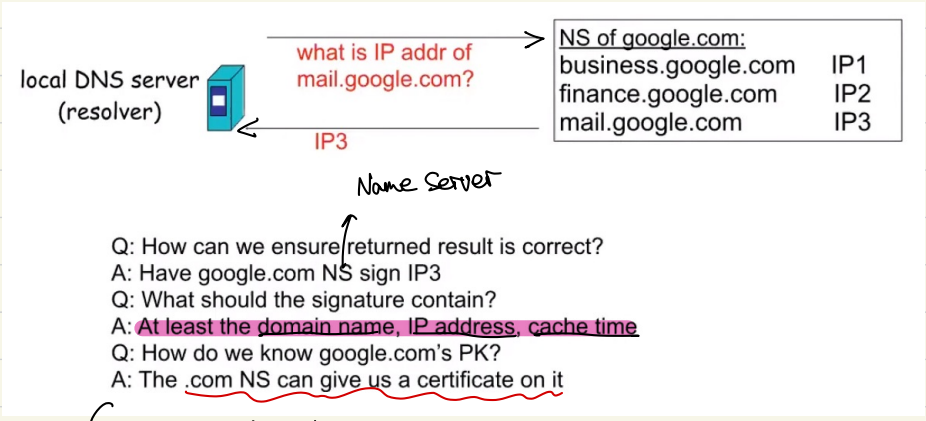
① Idea #1: do DNS lookups over TLS

- performance: TLS↓
- caching
- security: object security ☆ TLS → channel security

② Idea #2:

- signature: offline sign.

id ⇒ replay



↳ .com endorsed by root server

root server's public key hardcoded in resolver

③ non-existent answer

→ sign no answer reply

⇒ expensive, performance↓

⇒ DoS attack

existent host

pre-sign the hostname & IP address

✡ sign the no-record response effective.

— sign consecutive domains that do exist

— sign (["business.google.com", "finance.google.com"])

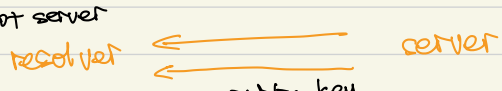
↳ indicates absence of a name in the middle and is cacheable.

o problem:

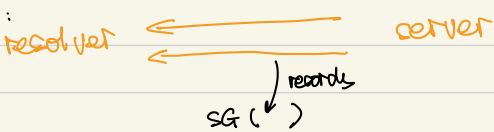
can retrieve all hosts names google has

✡ All keys as well as signed results are cacheable.

:) middle/root server



:) last server:



④ Issues of DNSSEC

#1: partial deployment.

#2: negative results ("no such name")

#3: big response → dos attack.

Summary:

- **TLS**: provides channel security (for communication over TCP)
  - Confidentiality, integrity, authentication
  - Client & server agree on crypto, session keys
  - Underlying security dependent on:
    - Trust in Certificate Authorities / decisions to sign keys
    - (as well as implementors)
- **DNSSEC**: provides object security (for DNS results)
  - Just integrity & authentication, not confidentiality
  - No client/server setup "dialog"
  - Tailored to be caching-friendly
  - Underlying security dependent on trust in Root Name Server's key, and all other signing keys

Q. A startup, `ThoughtlessSecurity`, decides to modify DNSSEC to only return a signature of the *requested domain* on a negative result. They claim that this change will drastically reduce the packet-size of a negative result.

A company implements `ThoughtlessSecurity`'s product on their nameserver. What attack is now possible? Specify exactly how an attacker could execute this attack.

A: DoS攻击，attack发送随机的不存在的域名，导致服务器始终在计算 signature.

⇒ 需要使用不同的域名，因为 negative 结果通常会被缓存。

Lec 15. Denial of Service

* attacks on availability

① application level
- program flaw
- resource exhaustion: infinite loop

◦ defense: i) isolate users ii) limit resource a user can use

② network layer
- ddos: flood with packets from many sources.
- Amplification: abuse parties who will amplify traffic for you
- resources attack needs:
  - i) sending capacity on the bottle link
  - ii) overwhelm the rate the bottle router can process

◦ defense:

i) network filter to discard any packets that arrive with attackers IP (Attacker's IP = means of identifying misbehaving user)
- ① spoof src address
- ② use many hosts to send traffic ⇒ ddos

⇒ website sign up for Cloudflare's DoS protection service, then it can acquire a huge amount of bandwidth to withstand DDoS attack. When a website suffering DDos, Cloudflare can use its bandwidth to stop traffic and only forwards cleansed-up traffic.

③ Amplification.

ex: DNS amplification attack.

- attacker spoofs DNS request to a poorly DNS server, seemingly from the target.

⇒ short request ↔ long response

⇒ doesn't increase # of packets, but total volume

- Note #1: blind spoofing ⇒ only work for UDP
- Note #2: victim doesn't see spoofed source addresses

④ Transport-Layer Attack

⇒ TCP uses memory to serve for connection

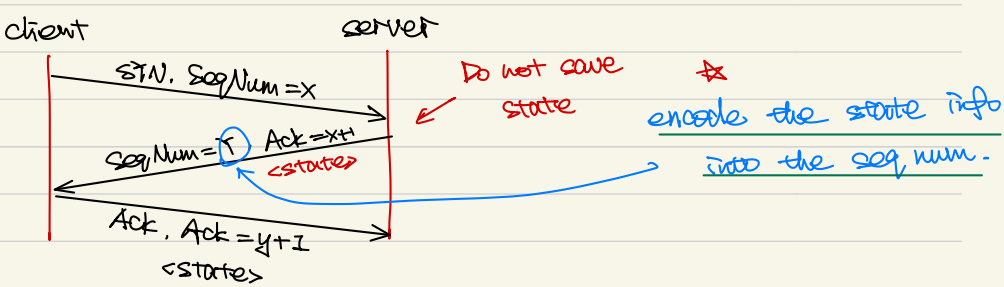
— ATTACKer targets memory
— every unique SYN burdens the target

- defense,

i) memory ↑

ii) identify bad actors

iii) × keep state ("SYN cookies")



— encode state in seq number

→ attacker must complete 3-way handsha
  — can't use spoofed source addresses
  — client's seq number and server's seq number are included in the ACK packet
  — Any extra state is encoded in server seq number
    y (ex: y = state + HMAC(key, state), key is only known to the server)

⑤ Application DoS. — Algorithm complexity attacks
- O(1) —attack→ O(N)

⇒ ex: cause hash collisions

⇒ use algorithms with good worst-case run time.

- force victim to consume resources
- overwhelm a service's processing capacity
- Approach #1: Only let legit users issue expensive requests
  — identify / authenticate
- Approach #2: force legit users to use resource

⑥ Summary

use firewall to filter ⇒ can overwhelm firewall

i) defense against program flaws

ii) defense resources from exhaustation
  — Isolation & scheduling mechanism
  — Reliable identification

lec 16 Firewalls

— Harden a set of systems against external network.

① on each system. turn off unnecessary network services
- scaling problem

② firewall: single point control.

☑ security policy: who is allowed to talk to whom, accessing what services?

— inbound: external → internal

— outbound: int → ext.

1. default – deny: start off permitting just a few known, well-secured services
2. default – allow: start off permitting external access services.

— flexibility v.s. conservative design.

☑ stateless packet filter

— inspect each packet for certain rules to determine whether to block.

① deny all inbound connections
- How to distinguish inbound packet associated with outbound connection?

⇒ allow all outbound TCP packets.

⇒ allow all inbound TCP packets, except those with SYN but without ACK flags set.

⇒ block inbound connection.

- X UDP

☑ **Stateful** packet filter

— router that checks each packet against rules and decide whether to forward or drop it.

⇒ keep track of all connections.

allow tcp connection *:*/int → 3.1.1.2:80/ext.

ex.: allow inbound connection to server, but block any attempts to login as "root".

— to log in, FTP client sends "USER Alice" then "PASS ...." over a TCP connection to the server

▫ state need to record:

i) ftp connection?   iii) ip addr, port

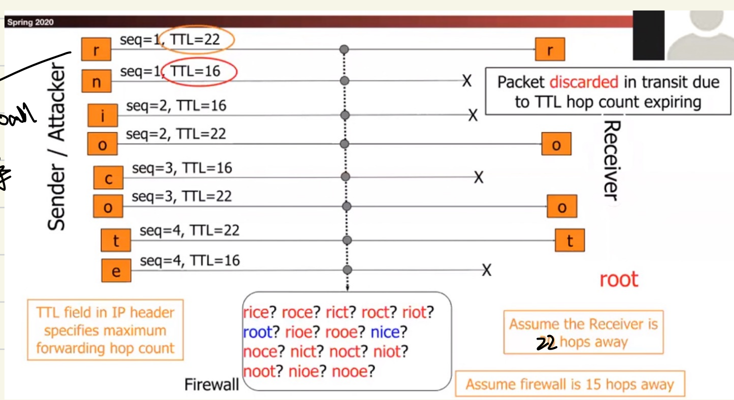
ii) username   iv) inbound / outbound connection

▫ challenge:

— "root" may span multiple packets

— packets may be re-ordered.

相同序列号, firewall 需要决定保留的对象

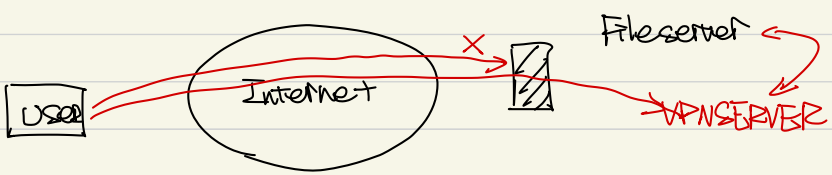


Application-level Firewalls

- Firewall act as a proxy. TCP connection from client to firewall, which then makes a second TCP connection from firewall to server.
- Eliminates risks of stateful packets filter interpreting packets different from host.

VPN

- goal: provide secure remote access to a network protected by a firewall
- create secure channel (Virtual Private Network, or VPN) to tunnel from outside host/network to inside network.



firewall = weak authentication (ex: limit ip addr)
VPN = strong authentication

- Pros:
  - i) central control
  - ii) easy to deploy
  - iii) address an important problem: vulnerabilities in network services.

☑ attack firewall

- compromise a internal machine
- the malicious insider problem

☑ take away

- Reference monitors and access control at the network level
- Attack surface reduction
- Centralized control

lec 17 Intrusion Detection.

Network Intrusion Detection (NIDS)

- #1: look at the network traffic.
  - scan http requests
  - look for "/etc/passwd", "../.." in requests.
  - has a table of all active connections, and maintains state for each
    - ⇒ stateful packet filter will block connection
    - ⇒ NIDS just detect.
  - pros:
    - i) cheap
    - ii) convenient: install only on server side

NIDS Evasion Attack

- when detect a RST packet..
  - ⇒ assume RST won't be received.
    - TTL was set before packet reaches the dest.
      - ⇒ if delete connection, loss states in the table.
    - too many states.
- nids and server both parse the packet
  - inconsistency: interpreted differently between the monitor and the end system
  - ambiguity: information needed to interpret correctly is missing.
- issue: only inspects network packets, not the actual msg

☑ Host-Based Intrusion Detection (HIDS)

- #2: instrument the web server
- pros: — works for encrypted HTTPS!
  — no problems with HTTP complexities

☑ Log Analysis

- #3: scripts run to analyze the log files in the back end.
- detection delayed.

☑ System call Monitoring (HIDS)

- #4: monitor system call activity of backend processes
- may fake positive

☑ Detection accuracy

— Two types of detector errors

- false positive (FP): alerting about a problem when no problem
- false negative (FN): failing to alert about a problem

I: the event that an instance of intrusive behavior occurring

A: the event that the detector generating alarm

⇒ False positive: $P(A|\neg I)$

⇒ False negative: $P(\neg A|I)$.

Base Rate Fallacy ⇒ 基本比率谬误：忽略 attacks 在总体中的比例对分析准确性的重要。

- Suppose our detector has a FP rate of 0.1% (!) and a FN rate of 2% (not bad!)
- Scenario #1: our server receives 1,000 URLs/day, and 5 of them are attacks
  - Expected # FPs each day = 0.1% * 995 ≈ 1
  - Expected # FNs each day = 2% * 5 = 0.1 (< 1/week)
  - Pretty good!
- Scenario #2: our server receives 10,000,000 URLs/day, and 5 of them are attacks
  - Expected # FPs each day ≈ 10,000 :-(
- Nothing changed about the detector; only our environment changed
  - Accurate detection very challenging when base rate of activity we want to detect is quite low

styles of detection.

1° Signature-Based Detection

look for activity that match for known attacks.

e.g.: search for "../../" or "/etc/passwd"

— blacklist [default allow]

variant: vulnerability signatures

— match for known problems, not known attacks

2° Anomaly-Based Detection 基于异常的检测

attacks look peculiar

— develop a model normal behavior (e.g. based on analyzing historical data). flag activity that deviates from it.

- problems
  - i) can fail to detect known attacks
  - ii) API changes. FP↑
  - iii) high FP ☆ ← academic but not generally used.

3° specification-based detection

specify what's allowed.

- pros: low FP
- cons: labor to derive / update specification

4° behavioral detection

x look for attacks. look for evidence of compromise

eg: inspect output web traffic payload for lines that matches passwd file.

eg: trace sys calls. flag those that shouldn't have shown up.

- pros: cheap. flexible
- cons: post facto detection, detect after having a problem

summary

- Evasions arise from uncertainty (or incompleteness) because detector must infer behavior/processing it can't directly observe
  - A general problem any time detection separate from potential target
- One general strategy: impose canonical form ("normalize")
  - E.g., rewrite URLs to expand/remove hex escapes
  - E.g., enforce blog comments to only have certain HTML tags
- Another strategy: analyze all possible interpretations rather than assuming one
  - E.g., analyze raw URL, hex-escaped URL, doubly-escaped URL …
- Another strategy: Flag potential evasions
  - So the presence of an ambiguity is at least noted
- Another strategy: fix the basic observation problem
  - E.g., monitor directly at end systems

Anti-virus

- URL/Web access blocking
  - Prevent users from going to known bad locations
- Protocol scanning of network traffic (esp. HTTP)
  - Detect & block known attacks
  - Detect & block known malware communication
- Payload scanning
  - Detect & block known malware
  - (Auto-update of signatures for these)
- Cloud queries regarding reputation
  - Who else has run this executable and with what results?
  - What's known about the remote host / domain / URL?
- Sandbox execution
  - Run selected executables in constrained/monitored environment
  - Analyze:
    - System calls
    - Changes to files / registry
    - Self-modifying code (polymorphism/metamorphism)
- File scanning
  - Look for malware that installs itself on disk
- Memory scanning
  - Look for malware that never appears on disk
- Runtime analysis
  - Apply heuristics/signatures to execution behavior

take away

① signature-based v.s. anomaly detection
(black-listing) (white-listing)

② Evasion attacks

③ Evaluation metrics: FP, FN rate

④ Base rate problems.