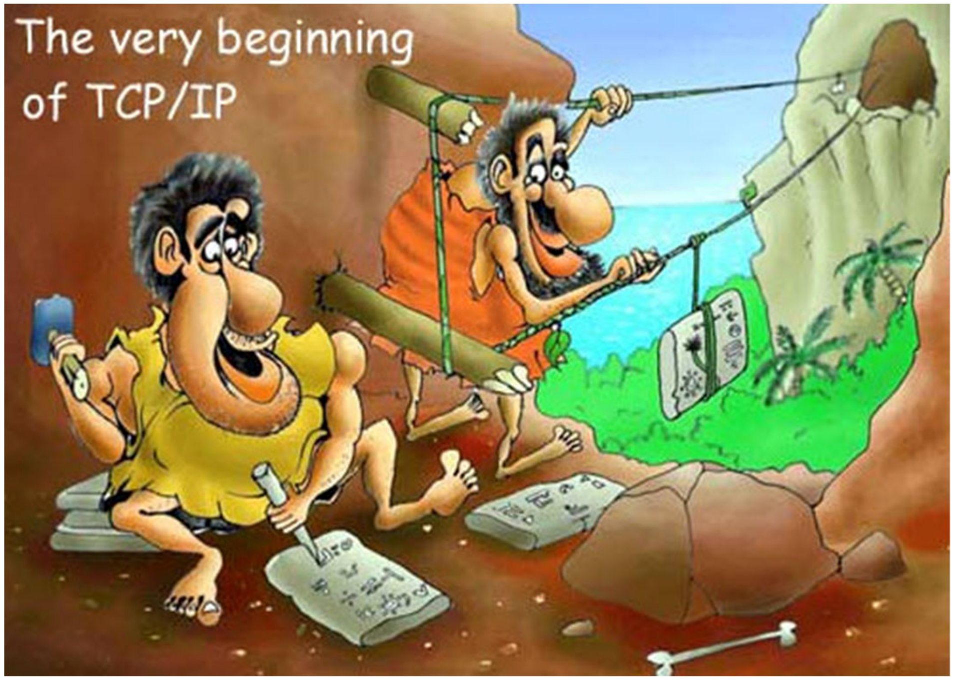
CS5321 Network Security
Week5: TCP/IP Security

Daisuke MASHIMA

2022/23 Sem 2

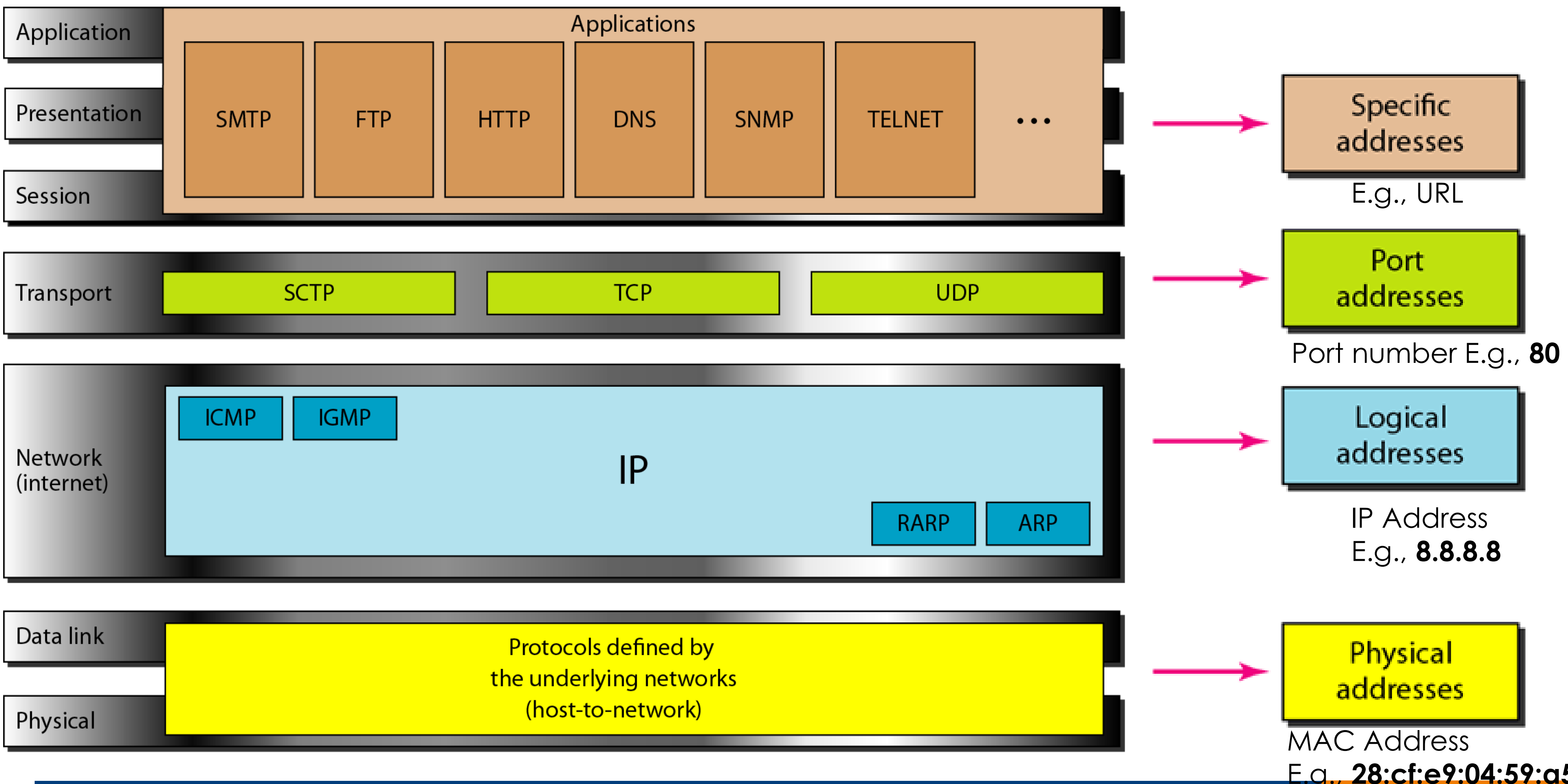
Overview

- In this lecture
  - IP vulnerabilities
  - New IP architecture
  - TCP vulnerabilities
  - TCP Hijacking



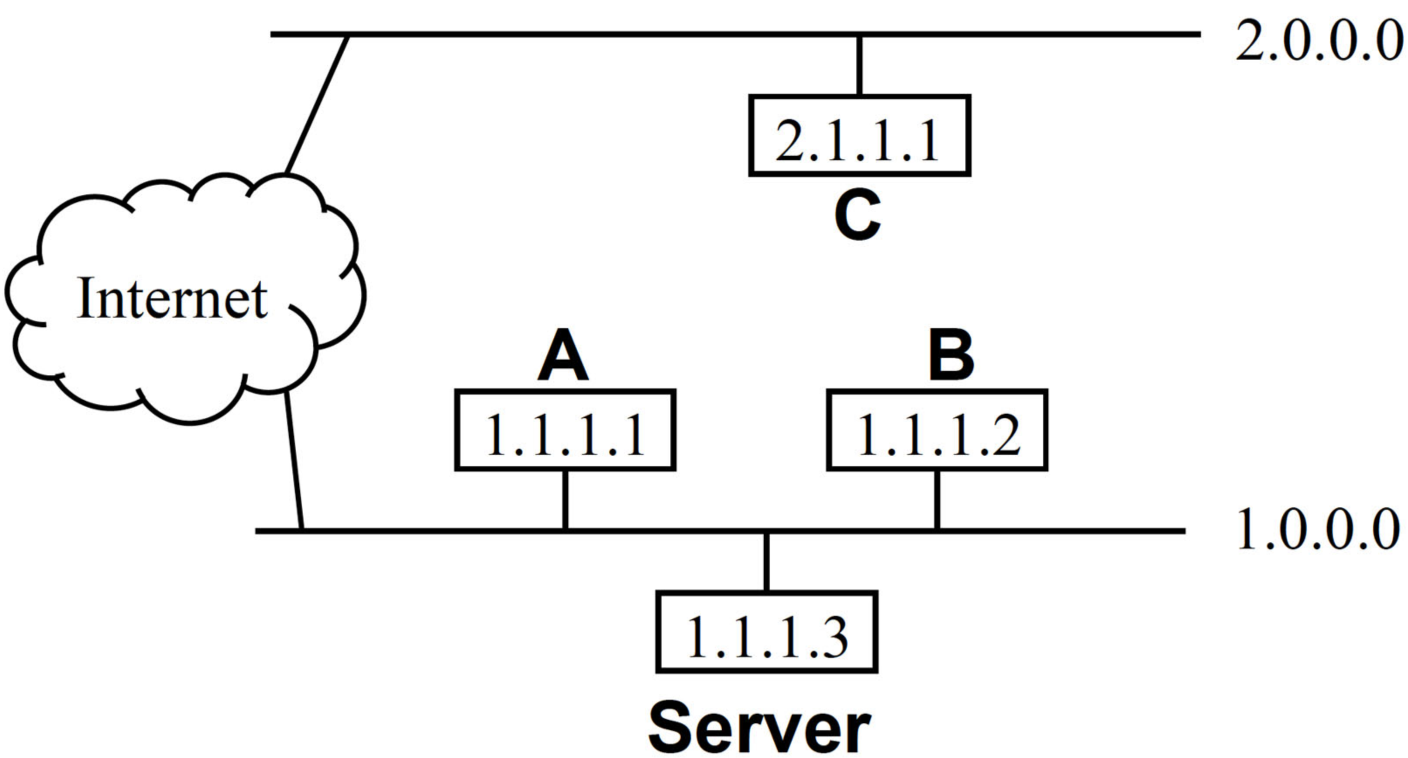
TCP/IP Stack

- Foundation for various Internet-based network services



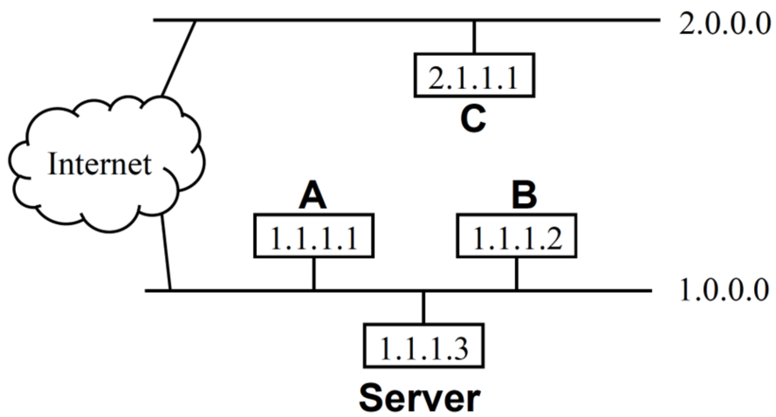
IP VULNERABILITIES

Security Issues in IP Networks



- Security issues for communication between A, B, C, and Server?

Basic Security Issues



A send S(server) a packet (P)

- A → S: P (using the IP protocol)
- How can S know that the packet originated from A?
- Can B overhear P?
- Can B impersonate A to S?
- Can C impersonate A to S?

IP Packet

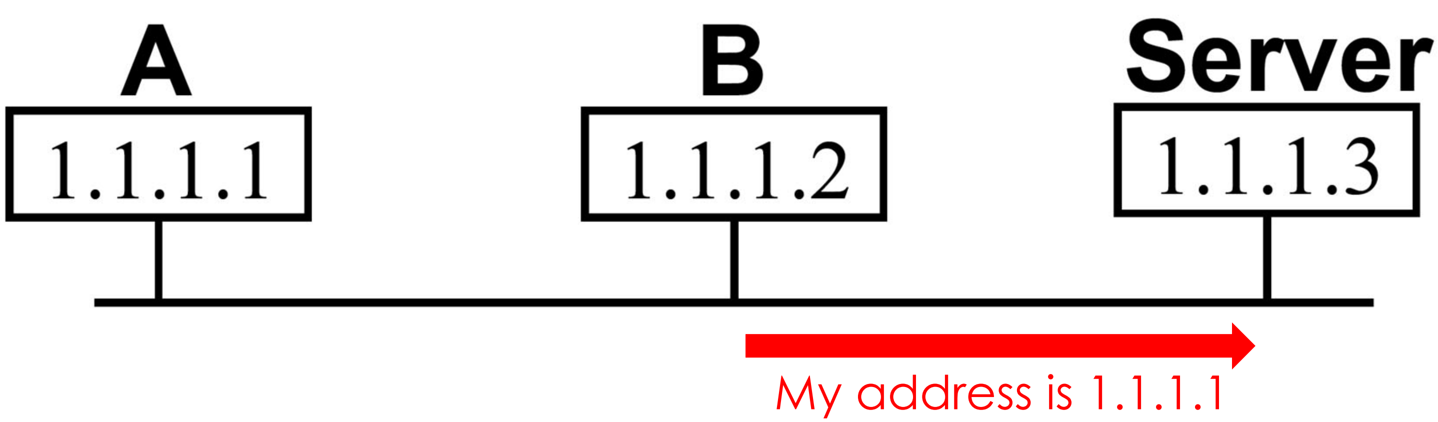
```
Internet Protocol Version 4, Src: 172.16.8.12, Dst: 172.16.1.41
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  > Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 113
    Identification: 0x07d8 (2008)
  > Flags: 0x40, Don't fragment
    Fragment Offset: 0
    Time to Live: 64
    Protocol: TCP (6)
    Header Checksum: 0xd159 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 172.16.8.12
    Destination Address: 172.16.1.41
```

IP Packet captured by Wireshark

- IP packet **"claims"** source and destination IP addresses
- Receiver **"assumes"** that the sender address is the one specified as source address.
- No authentication!

Flaw: Use IP Address for Authentication

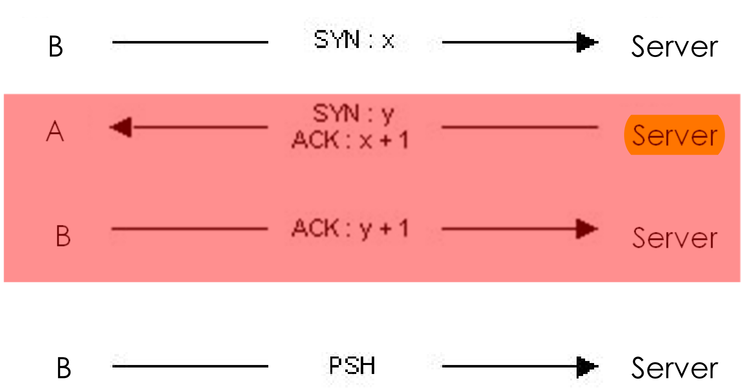
- IP source address can be easily spoofed!
- Easy to mount attack for another machine on the same network
- Example: r-utilities (rlogin, rsh, rcp)
  - Consider Server trusts admin's machine A
  - If B spoofs A's address, user on B can log in to Server
- Is it enough for meaningful attack?



Flaw: Use IP Address for Authentication

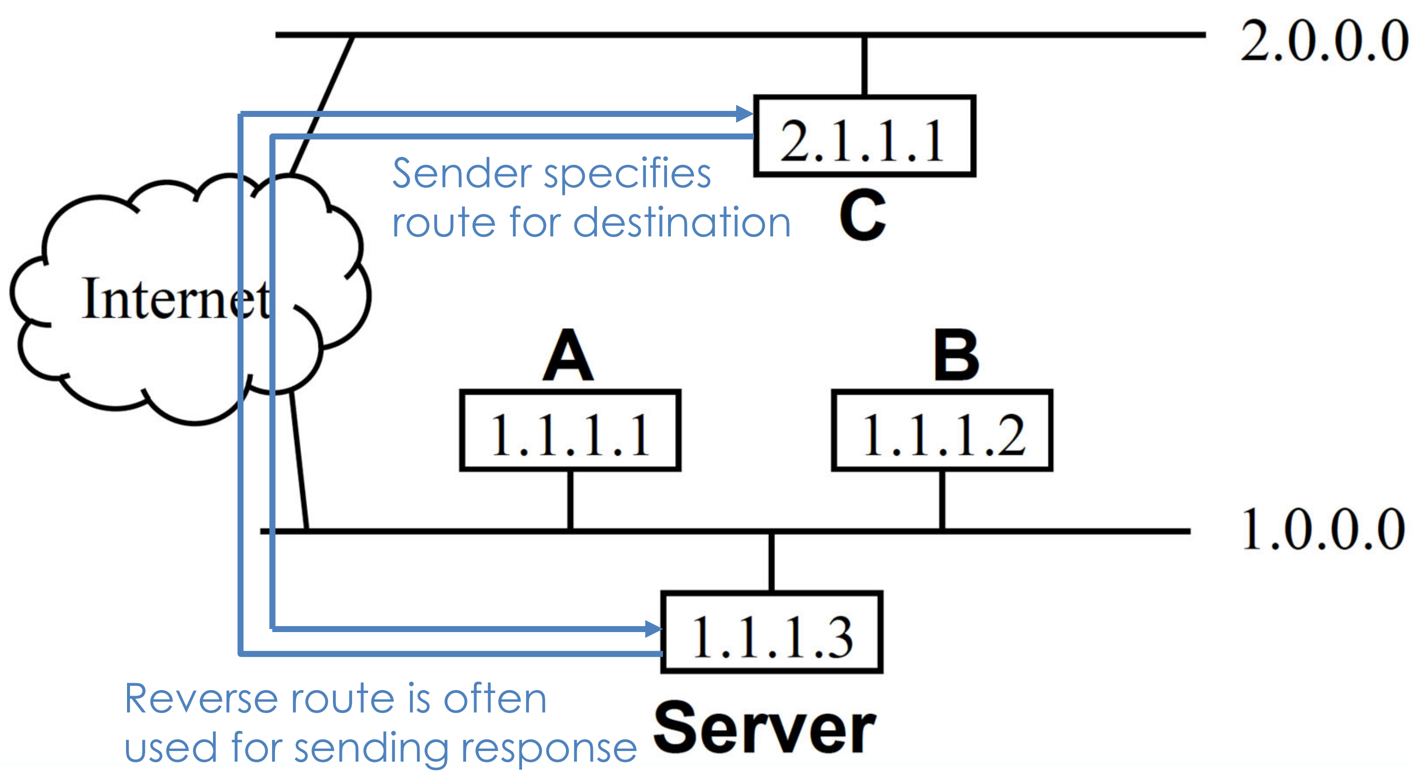
- Problem for attacker:
  - A receives S's responses to B's spoofed packets, as the destination address is A!
    - A will respond with a TCP Reset (RST) packet which closes the connection

- Solution for attacker:
  - By overflowing A's queues with connection requests, it is likely that A drops S's replies
    - Note: **DoS attack** is used to enable another attack
  - In the same network (more specifically collision domain), the attacker can still "see" the S's response to A by using **"promiscuous mode"**.



Flaw: Use IP Address for Authentication

- How can C impersonate A to S here?



Possible with **source routing**!

Big problem of current IP: IP Spoofing

- **Ingress filtering**
  - Let the upstream network block spoofed IPs
  - Lack of incentive
- **iTrace** (https://www.cs.columbia.edu/~smb/talks/ietf47/itrace.pdf)
  - 1 in 20,000 packets triggers a router to send an ICMP packet to a destination with route information for traceback
  - Needs authentication
- **Packet marking**
  - Routers mark 16-bit IP ID field with information that enables traceback
  - Needs changes to routers
- ***Open problem of the Internet!***

New IP Architecture: Accountable Internet Protocol

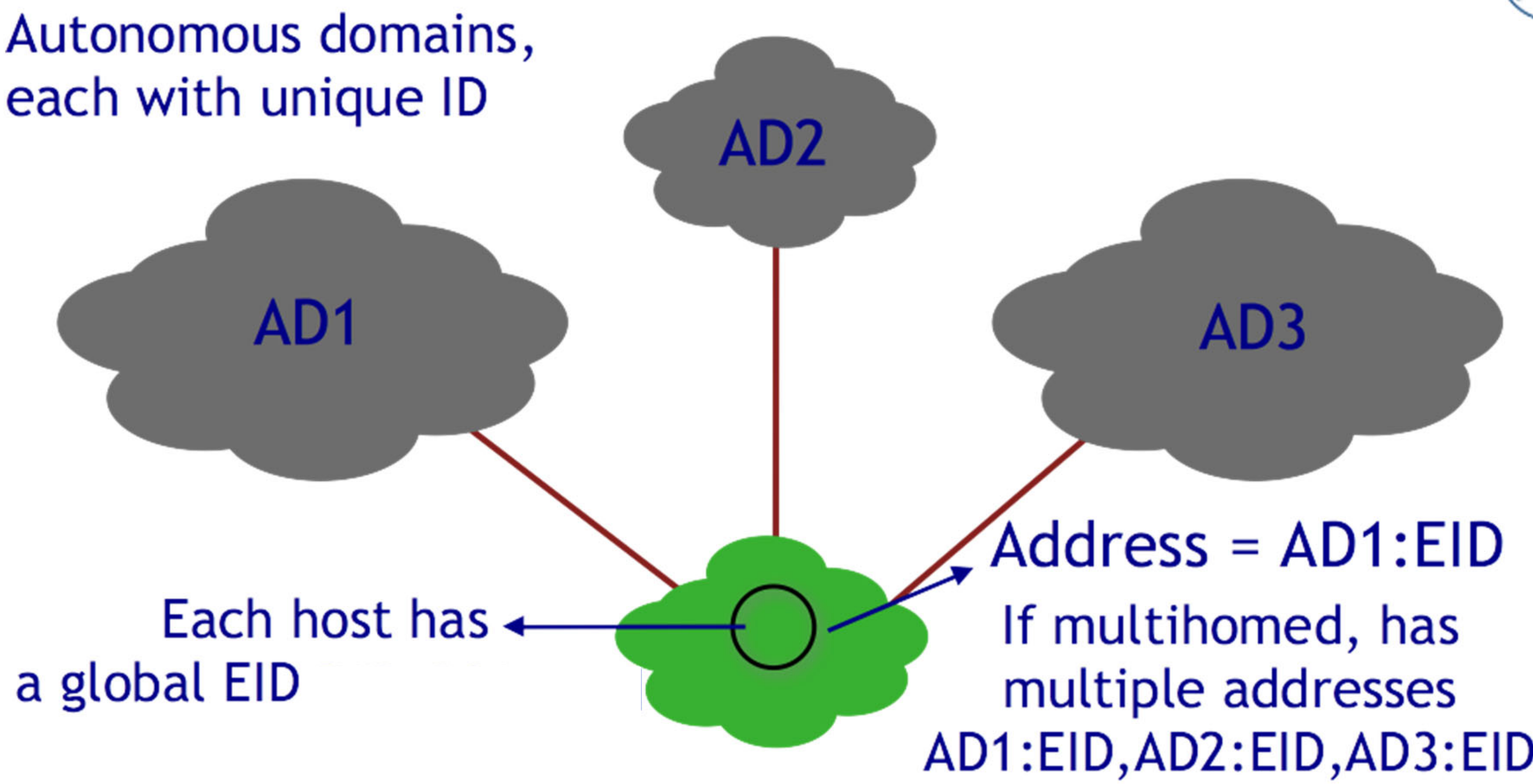
- Paper: **Accountable Internet Protocol** by David Andersen et al.
- Internet Protocol is old. No security in it.
- Lots of security patches but they are not satisfactory:
  - Complicated Mechanisms
    - Many details to circumvent IP weaknesses
  - External Sources of Trust
    - Trusted certificate authorities (e.g., S-BGP)
  - Operator Vigilance
    - Semi-manual configuration (e.g., filters, registries)

IP Layer Names Don't Have Secure Bindings

- Three kinds of IP layer names:
  - IP address, IP prefix, AS (autonomous system) number
- No secure binding of host to its IP addresses
- No secure binding of AS number to its IP prefixes

- Many problems become easier to solve with **network-layer accountability**: *Ability to associate a principal with a message*

AIP Addressing

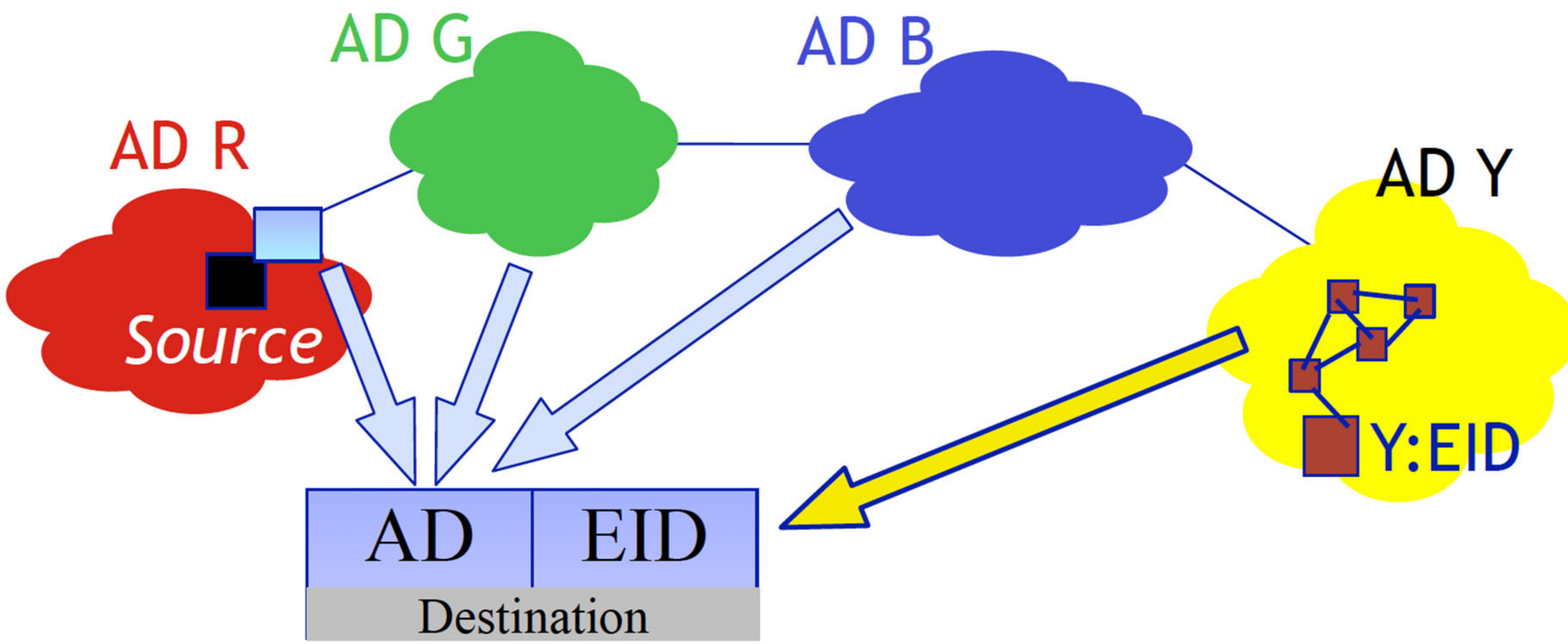


Key Idea:

AD and EID are *self-certifying flat names*

- AD = hash(public_key_of_AD)
- Self-certification binds name to named entity

AIP Forwarding and Routing



- Inter-AD routing & forwarding: AD #s only.
  - When packet is crossing AD boundary, checks are performed
- Intra-AD routing disseminates EIDs.
- Many routing protocols possible - derive security from AIP self-certification

Detecting & Preventing ***Spoofing***

- Self-certified entity can prove it sent message:

Let:

$$
\begin{aligned}
rs &= \text{Per-router secret, rotated once per minute} \\
\text{HMAC}_{\text{key}}\langle \text{M} \rangle &= \text{Message authentication code of M} \\
H\langle P \rangle &= \text{Hash of } P \\
\text{iface} &= \text{Interface on which packet arrived}
\end{aligned}
$$

Source $S_{AD} : S_{EID} \rightarrow$ **Dest** $D_{AD} : D_{EID}$

Packet P.

Router R1 $\rightarrow$ **Source**:

Verification packet V =

$$\text{HMAC}_{rs}\langle S_{AD} : S_{EID} \rightarrow D_{AD} : D_{EID}, H\langle P \rangle, \text{iface} \rangle$$

Source $\rightarrow$ **R1**:

$$\{\text{accept}, K_{S_{EID}}, V\}_{K^{-1}_{S_{EID}}}$$

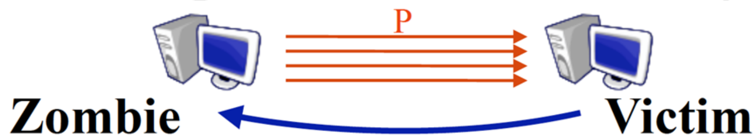
  - Routers or hosts seeing packet can check the AD or EID using a challenge-response protocol
  - Verification is done at the nearest router as well as intermediate ADs

Address Minting?

- Self-certifying address of AIP enables spoofing detection
- However, it does not prevent malicious hosts from creating large number of EIDs (or minting many identities!)
  - New EIDs can be used for DoS or filter circumvention
- Unfortunately, no clear technical solution
  - Simple solution would be to use public key infrastructure but it violates the core philosophy of AIP
- Engineering solutions (or operational countermeasures):
  - Each first-hop router limits number of new EIDs/second

AIP Enables Secure *Shut-Off*

- Problem: Compromised zombie sending stream of unwanted DoS traffic to victim
  - Zombie is “well-intentioned”, (i.e., owner benign) but compromised owing to its vulnerability.



Zombie → Victim: Packet P.

Victim → Zombie:

$$\{\text{key} = K_{\text{victim EID}}, \text{TTL}, \text{hash} = H\langle P \rangle\}_{K^{-1}_{\text{victim EID}}}$$

- Shut-off scheme implemented in **“smart-NIC”** (NIC firmware update requires physical access)
- If smart-NIC saw P that matches H(P), it installs a filter to block it
- Hardware requirements are practical
  - Bloom filter* for replay prevention (8MB SRAM)

* Space-efficient data structure “that is used to test if a certain element is a member of a set” (wikipedia)

Takeaways from AIP paper

- A whole new IP is needed!
  - AIP is one nice proposal: focusing on **accountability**
- Good properties:
  - Spoofing detection
  - Shut-off protocol
  - Secure routing
- Yet, some concerns:
  - Scalability
  - Inability to do CIDR-like aggregation of addresses (e.g., 198.51.100.xxx/24)
  - Perhaps more importantly, lack of backward compatibility

TCP VULNERABILITIES

TCP Primer

- Transmission Control Protocol
  - Works at **Transport Layer**, on top of IP
- TCP provides reliable data transfer using the best effort IP service
- Typical TCP packet exchange
  - A → B: SYN(ISN_A)
  - B → A: SYN(ISN_B), ACK($ISN_A + 1$)
  - A → B: ACK($ISN_B + 1$)
  - A → B: data …

TCP 3-way handshake

ISN: Initial Sequence Number

TCP network trace

No.	Time	Source	Destination	Protocol	Length	Info
67	3.307390	172.16.2.100	172.16.2.41	TCP	66	1133 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
69	3.310325	172.16.2.41	172.16.2.100	TCP	66	80 → 1133 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PERM=1 WS=32
71	3.311980	172.16.2.100	172.16.2.41	TCP	54	1133 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
74	3.385125	172.16.2.100	172.16.2.41	HTTP	466	GET / HTTP/1.1
75	3.387219	172.16.2.41	172.16.2.100	TCP	60	80 → 1133 [ACK] Seq=1 Ack=413 Win=15680 Len=0
76	3.389015	172.16.2.41	172.16.2.100	HTTP	189	HTTP/1.1 307 Temporary Redirect

```
> Frame 69: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface \Device\NPF_{B871E50A-4813-41B4-BA95-95654B00ECB6}, id 0
> Ethernet II, Src: WAGOKont_40:d0:8d (00:30:de:40:d0:8d), Dst: GoodWayI_17:fb:5d (00:50:b6:17:fb:5d)
> Internet Protocol Version 4, Src: 172.16.2.41, Dst: 172.16.2.100
v Transmission Control Protocol, Src Port: 80, Dst Port: 1133, Seq: 0, Ack: 1, Len: 0
    Source Port: 80
    Destination Port: 1133
    [Stream index: 1]
    [TCP Segment Len: 0]
    Sequence Number: 0    (relative sequence number)
    Sequence Number (raw): 3189983741
    [Next Sequence Number: 1    (relative sequence number)]
    Acknowledgment Number: 1    (relative ack number)
    Acknowledgment number (raw): 287198956
    1000 .... = Header Length: 32 bytes (8)
  > Flags: 0x012 (SYN, ACK)
    Window: 14600
    [Calculated window size: 14600]
    Checksum: 0x6c65 [unverified]
    [Checksum Status: Unverified]
    Urgent Pointer: 0
  > Options: (12 bytes), Maximum segment size, No-Operation (NOP), No-Operation (NOP), SACK permitted, No-Operation (NOP), Window scale
  > [SEQ/ACK analysis]
  > [Timestamps]
```

TCP packets captured with Wireshark

TCP ISN Prediction Attack

- Typical TCP packet exchange
  - A → B: SYN(ISN_A)
  - B → A: SYN(ISN_B), ACK(ISN_A + 1)
  - A → B: ACK(ISN_B + 1)
  - A → B: data …
- Attack:
  - M(A) → B: SYN(ISN_A)
  - B → A: SYN(ISN_B), ACK(ISN_A + 1)
  - M(A) → B: ACK(ISN_B + 1)
  - M(A) → B: data …

M(A): Malicious party impersonating A

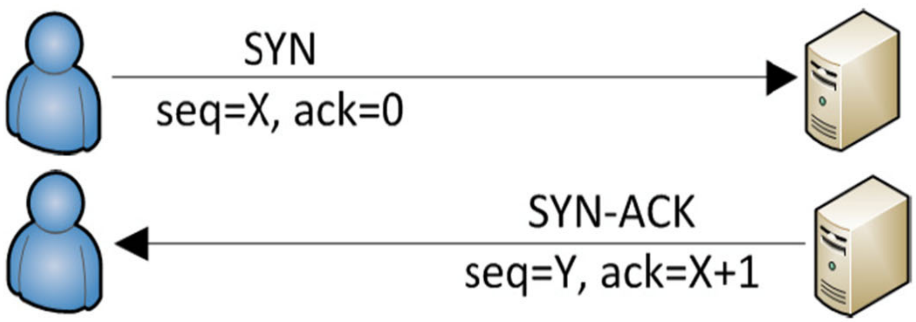
TCP ISN Prediction

- Are these good choices for next TCP ISN?
  - Always start at same ISN
  - After each connection, ISN++
- **No, attacker can predict next ISN!**
- Better choices for ISN?
  - ISN = rand() function of C library?
  - current ISN = H(prev ISN, k)?
  - ISN = DES_K(counter++)?

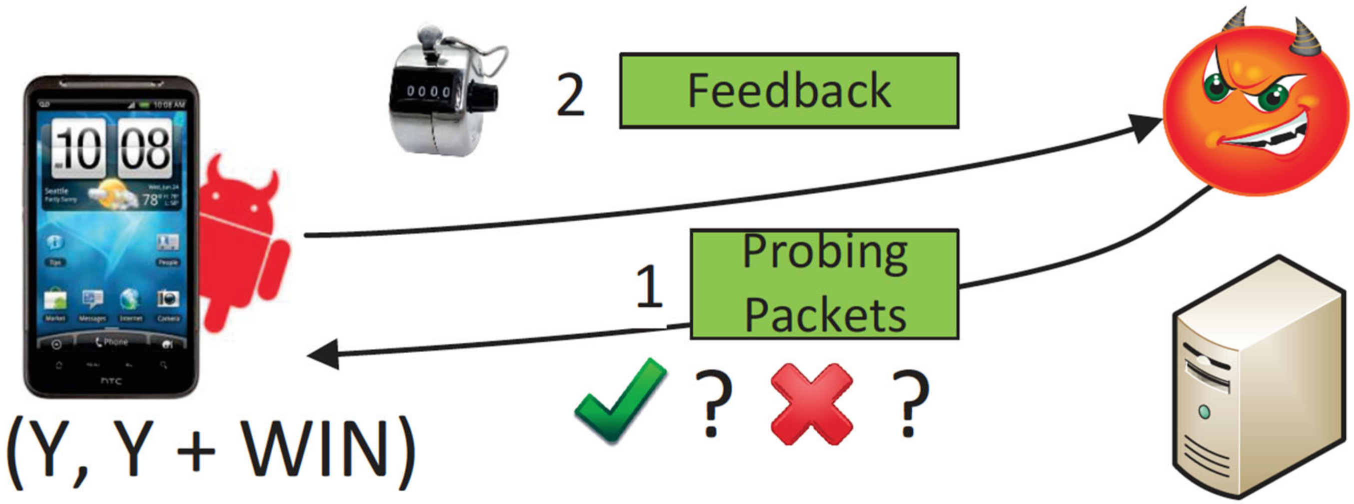
TCP Sequence Inference Attack

- Paper: "**Collaborative TCP Sequence Number Inference Attack — How to Crack Sequence Number Under A Second**" by Zhiyun Qian et al.
- Off-path attacks
  - Can write to existing TCP connection by guessing sequence numbers
  - Works even though Initial sequence number nowadays are randomized (2^{32})

X = ? Y = ?



TCP sequence number inference attack

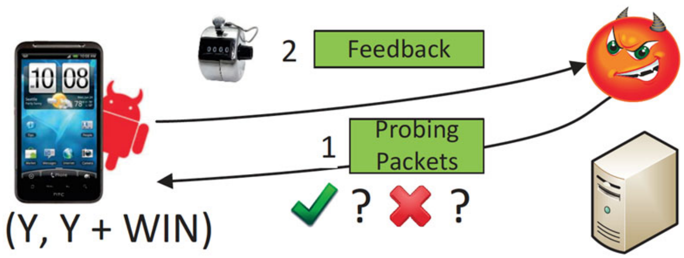


- Required information
  - Target four tuples (source/dest IP, source/dest port)
  - ***Feedback*** on whether guessed sequence numbers are correct
- *"Can an **unprivileged** malware accurately learn if the probing packet is in or out of receive window?*
- *"Or, can learn even more useful information?"*

Preliminary step: obtaining target four tuples

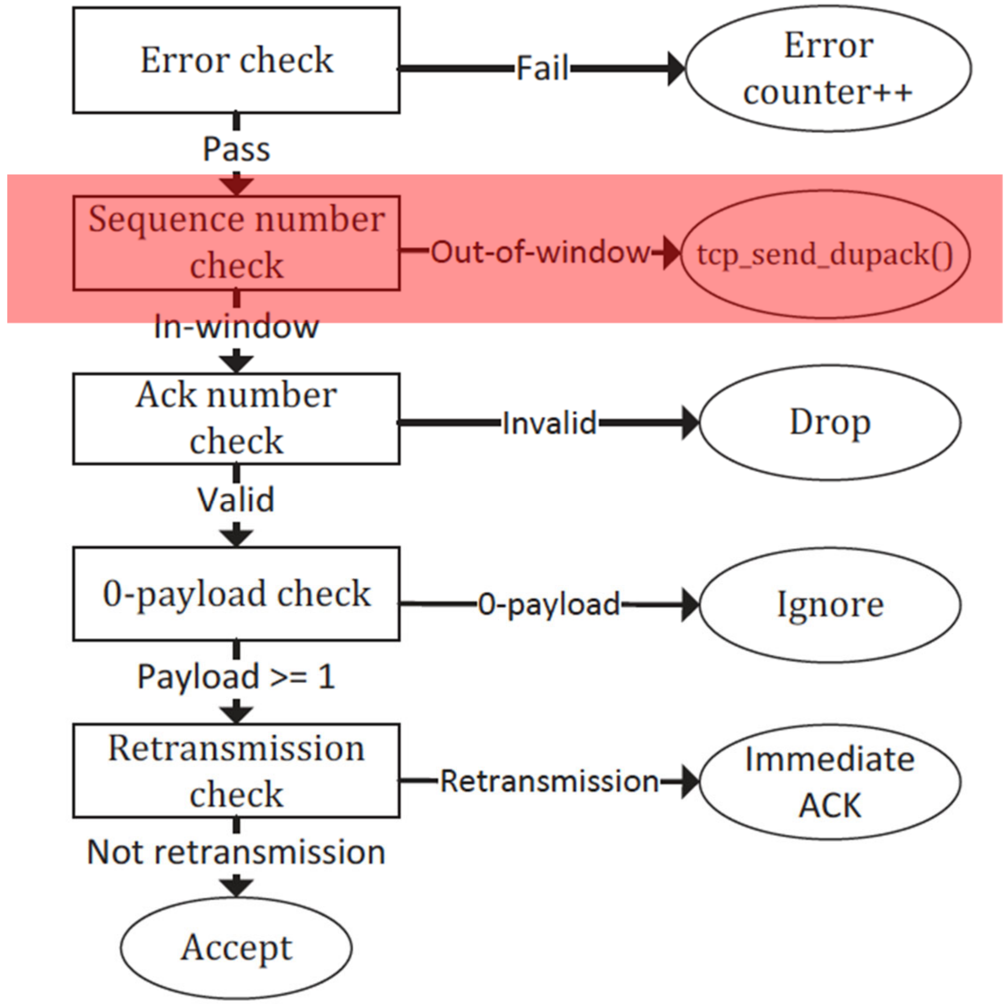
- On-site **unprivileged** malware
  - **netstat** (no root required) provides four tuples:
    - (srcIP, dstIP, srcPort, dstPort)

netstat -nn
Active Internet connections
Proto Recv-Q Send-Q Local Address Foreign Address
probing
Initiate fake connections
(state)
tcp4 37 0 192.168.1.102.50469 199.47.219.159.443
CLOSE_WAIT
tcp4 37 0 192.168.1.102.50468 174.129.195.86.443
CLOSE_WAIT
tcp4 37 0 192.168.1.102.50467 199.47.219.159.443
CLOSE_WAIT
tcp4 0 0 192.168.1.102.50460 199.47.219.159.443
LAST_ACK
tcp4 0 0 192.168.1.102.50457 199.47.219.159.443
LAST_ACK
tcp4 0 0 192.168.1.102.50445 199.47.219.159.443



Linux TCP incoming packet validation logic

- 5 steps to filter out invalid TCP packets (Linux)
- **tcp_send_dupack()**
  - update the global states when out-of-window packet received
  - if received seq# < Y
    - DelayedACKLost+=1
  - otherwise
    - DelayedACKLost+=0
  - DelayedACKLost can be read by any malware!

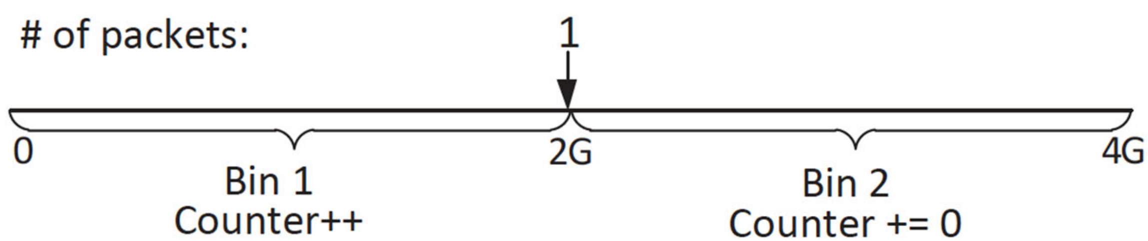


TCP incoming packet validation logic in Linux 3.2.6

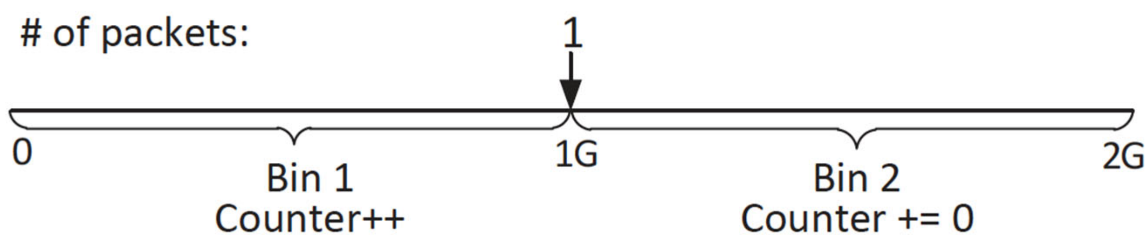
perfect side-channel for seq# inference!

How to find Y efficiently?

- Binary search!
  – total search space: 2^{32} (~ 4G)
  – each iteration, we can eliminate half of the space!



(a). First iteration



(b). Second iteration

  – at most 32 iteration (i.e., 32 probing packets) needed
  – see the paper for more optimizations

Client-side TCP injection attack

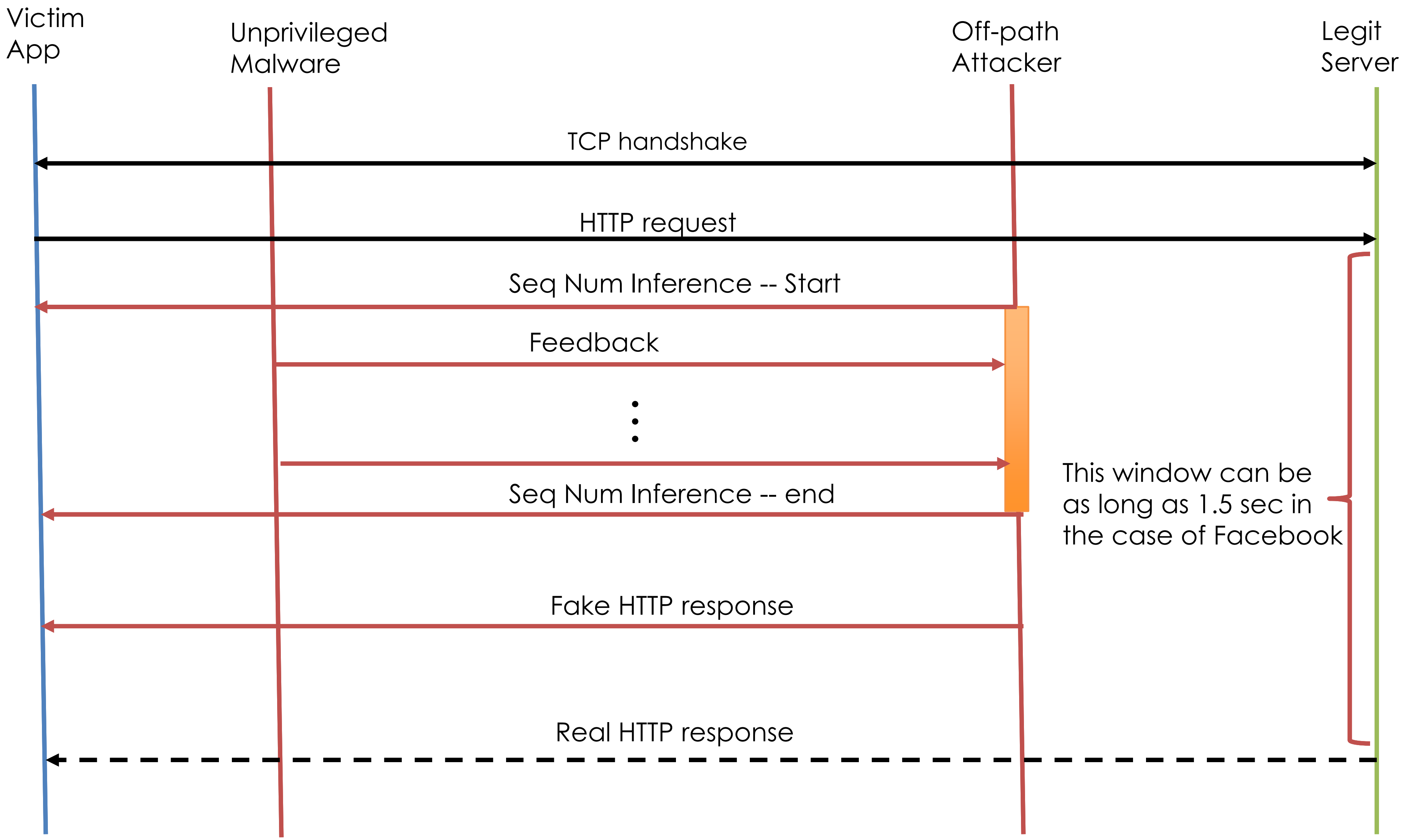
- A mobile phone app initiates a TCP connection to a server
- After TCP handshake, launch TCP seq# inference attack
  - Attacker now knows Y!
- Problem: need to compete with legitimate server
  - valid HTTP response may come first

No.	Time	Source	Destination	Protocol	Length	Info
67	3.307390	172.16.2.100	172.16.2.41	TCP	66	1133 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
69	3.310325	172.16.2.41	172.16.2.100	TCP	66	80 → 1133 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PERM=1 WS=32
71	3.311980	172.16.2.100	172.16.2.41	TCP	54	1133 → 80 [ACK] Seq=1 Ack=1 Win=65536 Len=0
74	3.385125	172.16.2.100	172.16.2.41	HTTP	466	GET / HTTP/1.1
75	3.387219	172.16.2.41	172.16.2.100	TCP	60	80 → 1133 [ACK] Seq=1 Ack=413 Win=15680 Len=0
76	3.389015	172.16.2.41	172.16.2.100	HTTP	189	HTTP/1.1 307 Temporary Redirect

Window for ISN inference

- Demonstration:
  - Facebook takes more than 1 sec to send the first HTTP response back to mobiles
  - With optimizations, attacker can be faster than legit Facebook's response most of times
    - Can inject malicious Facebook Javascripts

Client-side TCP Injection Attack



Passive/Active TCP hijacking

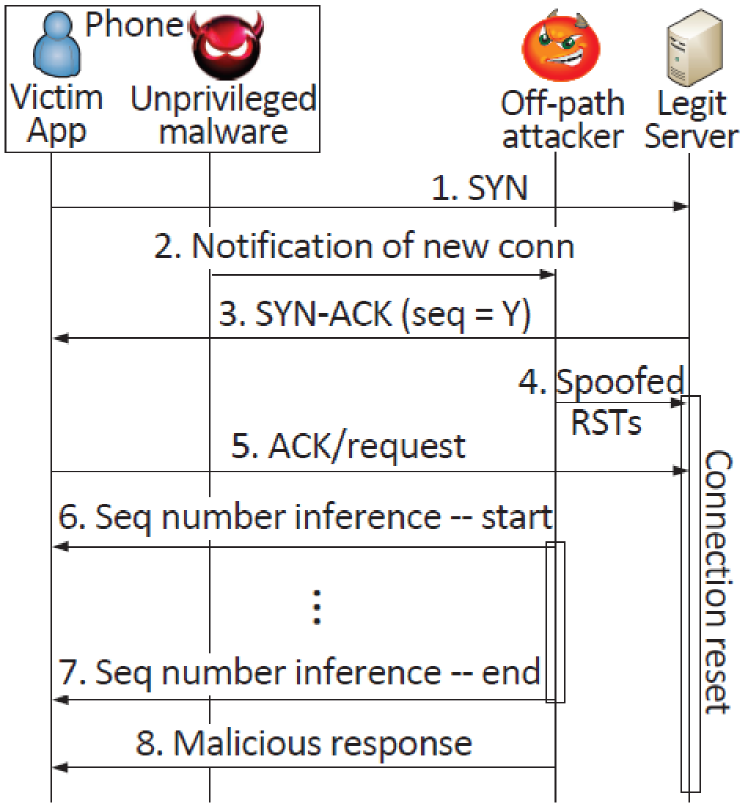


Figure 9: Passive TCP hijacking sequence

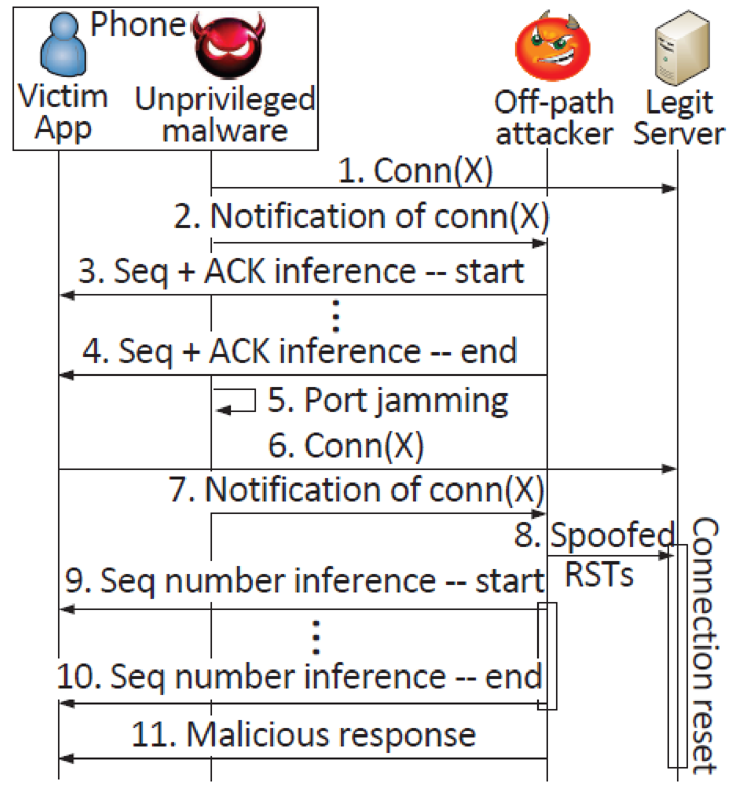


Figure 10: Active TCP hijacking sequence

(C1). Client-side ISN has only the lower 24-bit randomized. This requirement is necessary so that the malware can roughly predict the range of the ISN of a newly created TCP connection. *(holds for Linux 3.0.2 or earlier)*
(S1). The legitimate server has a host-based stateful TCP firewall. Such a firewall is capable of dropping out-of-state TCP packets.

(C2). Client-side ISN monotonically incrementing for the same four tuples. This client-side requirement is in fact explicitly defined in RFC 793 to prevent packets of old connections, with in-range sequence numbers, from being accepted by the current connection mistakenly.

Takeaways from TCP hijacking paper

- TCP hijacking is still possible!
- Why?
  - Our systems today (Linux, Android, Windows, Mac, etc.) have too much shared state
  - OS aggregated statistics (seemingly harmless) can leak critical internal network states
- Defenses?
  - Use SSL/TLS always
  - Removing unnecessary states
  - Isolating states
- There're more recent attacks (e.g., exploiting Wi-Fi protocol vulnerability to infer TCP sequence numbers)

Questions?