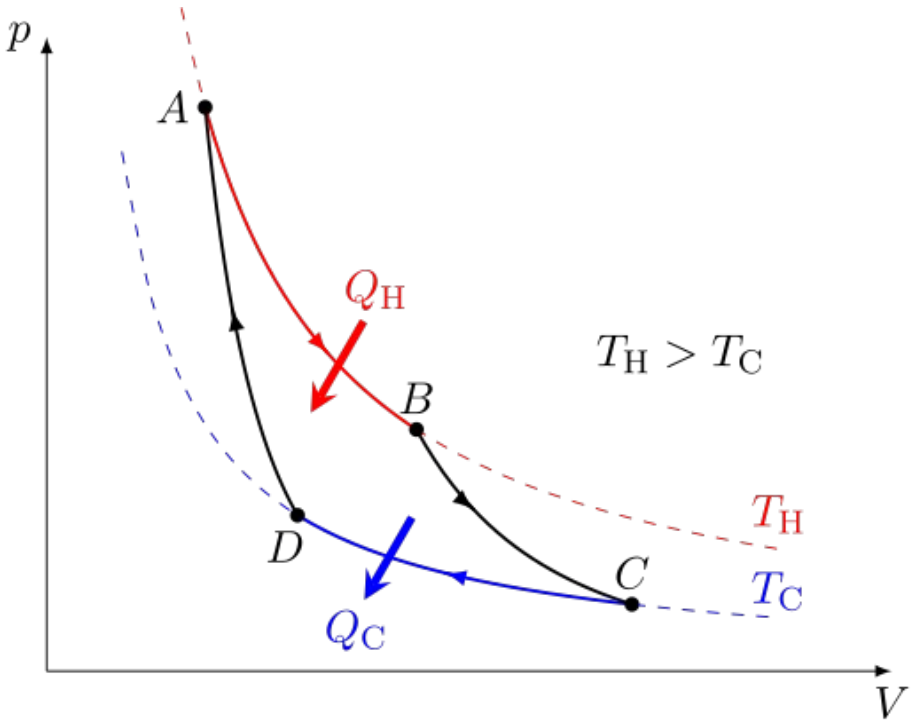
Port scanning and network side channels

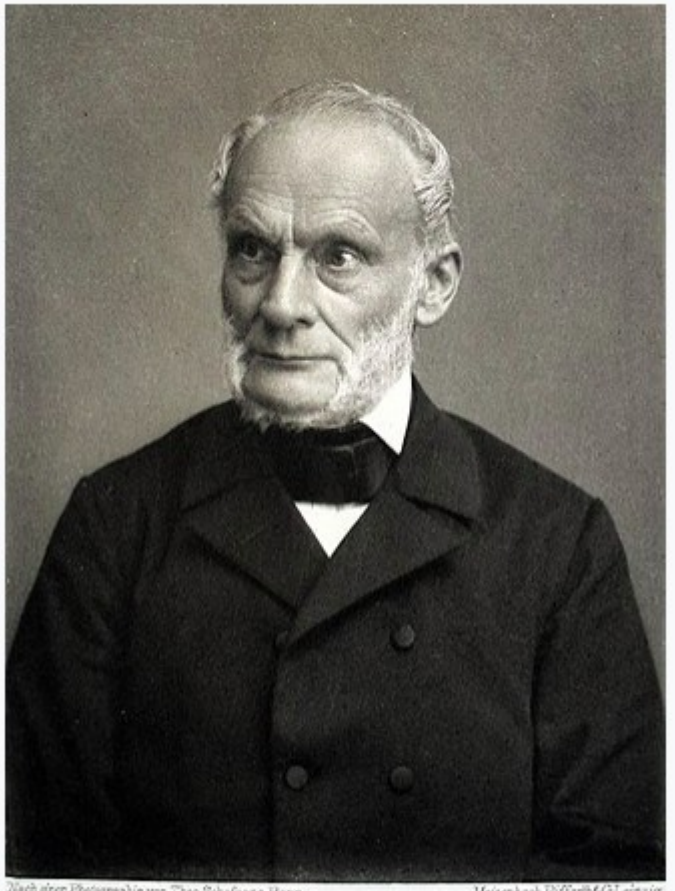
CSE 468 Fall 2026
jedimaestro@asu.edu





https://en.wikipedia.org/wiki/Nicolas_L%C3%A9onard_Sadi_Carnot
https://en.wikipedia.org/wiki/Carnot_heat_engine

Rudolf Clausius



“entropy”

(from Greek ἐν en "in" and τροπή tropē "transformation")

Like energy, but you can’t use it.

https://en.wikipedia.org/wiki/Rudolf_Clausius

Review… Entropy

- Statistical foundation by Gibbs, Boltzmann, Maxwell, Planck, *etc.*
- Directly inspired the name of entropy in Shannon’s information theory

$$H = -\sum_i p_i \log_2 (p_i)$$

Requirements (Shannon, 1948)

1) $I(p) \geq 0$ (information is non-negative, $p \geq 1$)
2) $I(1) = 0$ (events that always occur carry no information)
3) $I(p_1 p_2) = I(p_1) + I(p_2)$ (information due to independent events is additive)

Also, continuity, symmetry, and maximum when all possible events are equiprobable.

$$I(p) = \log(1/p)$$

1) $I(1/2) = 0.30102999566...$
2) $I(1) = 0$
3) $I(1/2) + I(1/3) = \log(2) + \log(3) = 0.77815125038...$

Joint probability: $I(1/6) = 0.77815125038...$

Continuity: $I(1/2.01) = 0.30319605742...$

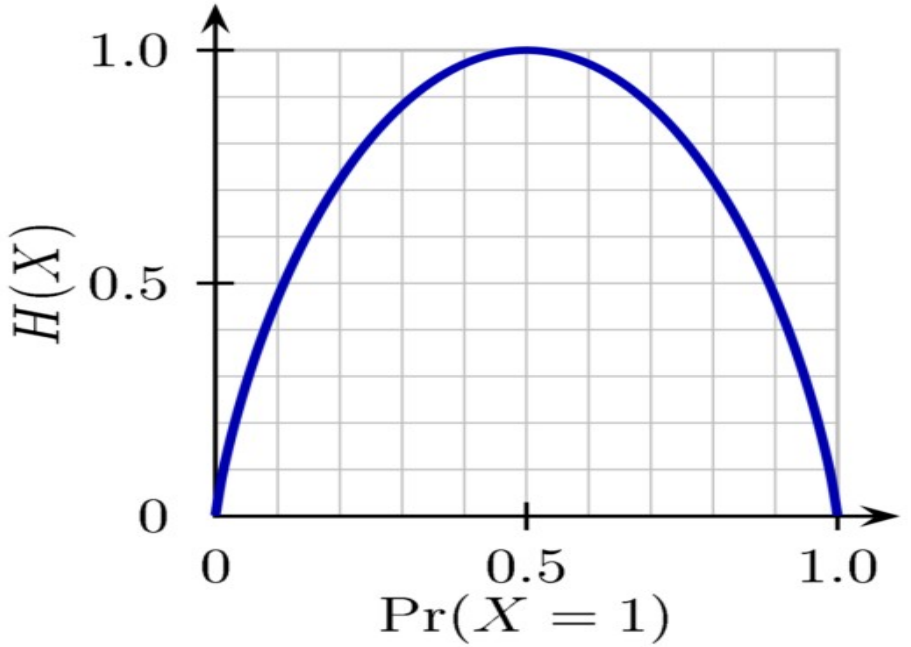
Symmetry: $\log(3) + \log(2) = 0.77815125038...$

Maximum: $\log 3 + \log 3 + \log 3 = 1.43136376416...$

$\log 2 + \log 4 + \log 4 = 1.20411998266...$

Information = Entropy = Surprise

$$H[p] = -\sum_{i=1}^{k} p_i \log p_i$$



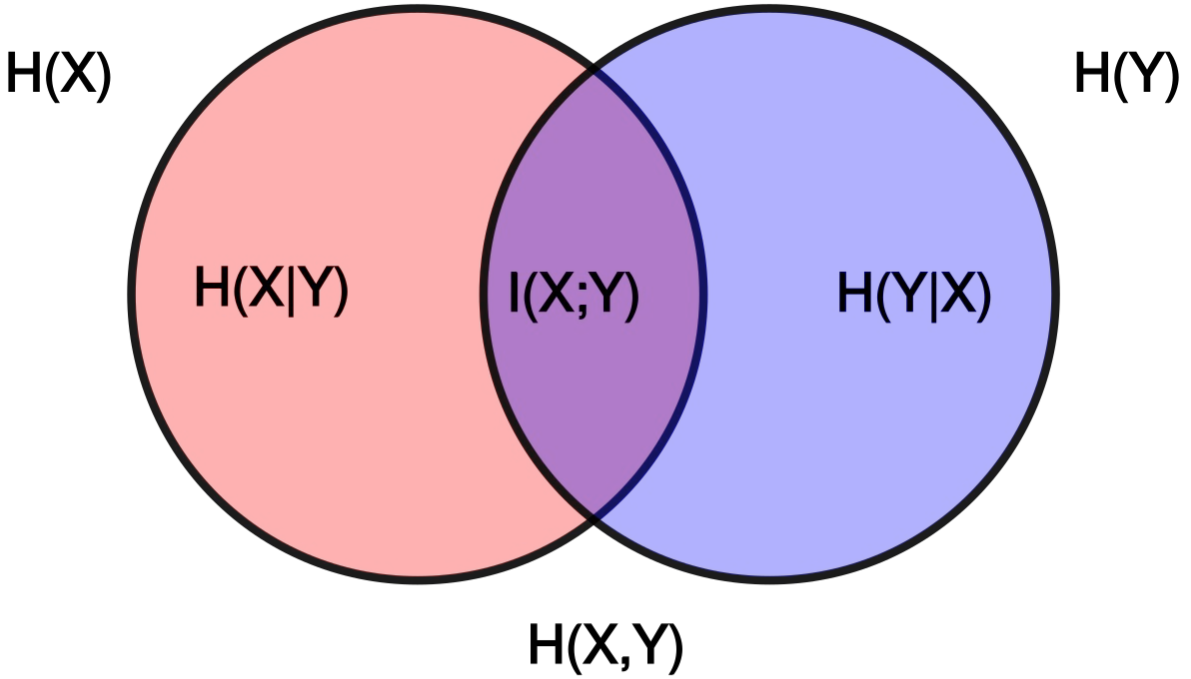
Not a pop quiz #3

You have 12 coins, one is counterfeit. The counterfeit is either slightly heavier or slightly lighter, otherwise it’s impossible to tell. You have a balance. Using the balance the fewest number of times, find the counterfeit coin.



https://en.wikipedia.org/wiki/Mutual_information

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$



WarGames (1983)



Something you know → Some new information

Discovery method	What gets tried	What the response reveals
War dialing	Telephone numbers	A modem or another answering service
X.25 NUA scanning	Packet-network addresses	Reachable systems, PADs, or modem outdials
TCP port scanning	IP addresses and ports	Reachability and listening services

Stuff nmap can do

Task	Flags	Example
Find live hosts Without scanning ports	`-sn`	`nmap -sn 192.168.1.0/24` ("horizontal" scan)
Find open TCP ports SYN scan or full connection	`-sS` (SYN scan) `-sT` (TCP connect)	`sudo nmap -sS HOST` ("vertical" scan, 1,000 most common TCP ports)
Find UDP services For example, DNS and NTP	`-sU`	`sudo nmap -sU -p 53,123,161 HOST` ("vertical" scan)
Identify service / version Send application probes	`-sV`	`nmap -sV HOST` e.g. `SSH-2.0-OpenSSH_9.6p1\r\n`
Guess the operating system Fingerprint network responses	`-O`	`sudo nmap -O HOST` e.g. `SYN/ACK: TTL=64, window=64240`

HOST = IP address or hostname

nmap.org/book/man-briefoptions.html

Compare to prehistoric examples...

- Paul Stira (Scorpion from MoD) got “SW” while wardialing higher numbers in a block, blew into the phone, was a DMS-100
  - Gave his friend 3-way calling
- Parmaster with an Apple IIe and a 1200 baud modem… DEFCON NUA scanner… Lower numbers in 223 and 224 blocks (as per Force)...
  - CitiSaudi debit card numbers
- Morris worm tried SMTP/sendmail (TCP 25), Finger/fingerd (TCP 79), rexec/rexecd (TCP 512, password-authenticated remote command execution), and rsh/rshd (TCP 514, remote shell execution using host/user trust)
  - Crashed the Internet

TCP 3-way handshake (review)

- SYN: I'd like to open a connection with you, here's my initial sequence number (ISN)
- SYN/ACK: Okay, I acknowledge your ISN and here's mine
- I ACK your ISN

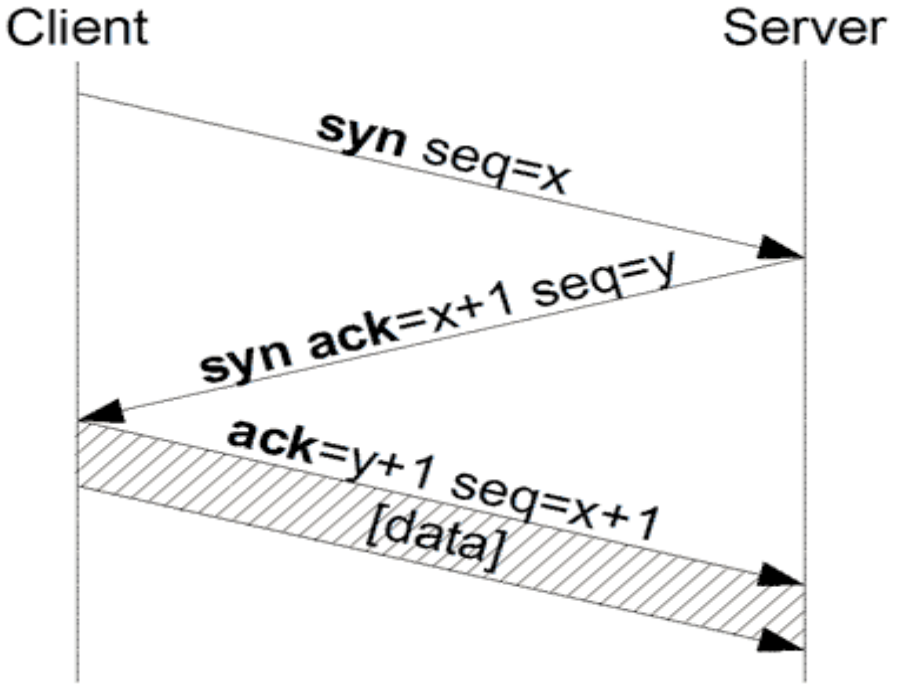


Image from Wikipedia

Open port == listening

- If you send a SYN packet to port 80 (the HTTP port) on a remote host and that host replies with a SYN/ACK, then we say that port 80 on that machine is “open”
  - In this example, that probably means it's a web server
- If it responds with a RST, we say it's “closed”
- If there is evidence of filtering (no response or ICMP==Internet Control Message Protocol error), we say it's “filtered”
  - UDP is more complicated: open|filtered *vs.* closed

Port scans follow a pattern, can be detected.

Network Intrusion Detection System (NIDS) Rules for Detecting Port Scans

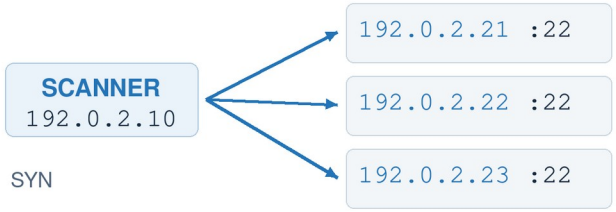
Suricata

ET Open SSH scan / SID 2001219 rev. 20

```
alert tcp $EXTERNAL_NET any ->
  $HOME_NET 22 (
  flow:to_server; flags:S,12;
  threshold:type both,
    track by_src, count 5,
    seconds 120;
  sid:2001219; rev:20;
  ... )
```

Detection excerpt; message and metadata omitted

FIXED: source IP + destination port
VARY: destination IP



5 SYN matches within 120 seconds

Different destination IPs are not required.

Sources: OISF users archive, 2016-Dec/014521

Zeek

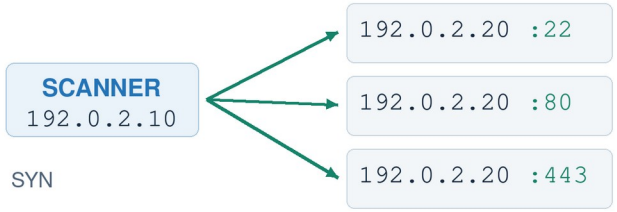
Legacy scan.zeek / Zeek 6.0

```
const port_scan_interval = 5min &redef;
const port_scan_threshold = 15.0 &redef;

SumStats::observe("scan.port.fail",
  [$host=scanner,
   $str=cat(victim)],
  [$str=cat(scanned_port)]);
```

Policy excerpts; retired from Zeek core in 6.1

FIXED: source IP + destination IP
VARY: destination port



15 distinct failed ports within 5 minutes

One source–destination pair per counter.

github.com/zeek/zeek / release/6.0
scripts/policy/misc/scan.zeek

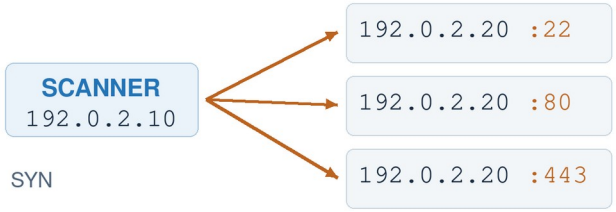
Snort

Published SYN-scan rule / Snort 2.9

```
alert tcp any any -> $HOME_NET any (
  msg:"TCP SYN";
  flow:stateless; flags:S;
  detection_filter:track by_dst,
    count 100, seconds 5;
  sid:1000001; rev:1;
)
```

Actual rule used in the SecControl evaluation

FIXED: destination IP
VARY: destination port



>100 SYN matches within 5 seconds

Counts matches, not ports; pools all sources.

Wang & Wu, SecControl, §6.1 (2018)
doi:10.4108/eai.10-1-2019.156242

Faster alternatives to nmap...

https://web.archive.org/web/20150607104352/http://dankaminsky.com/2002/11/18/77/

Paketto Simplified (1.0)

November 18, 2002 Dan Kaminsky Leave a comment Go to comments

SCANRAND

========

Really, really fast port scanner, that can also trace network paths. Port scanning is simply the act of asking a machine if you can start up a conversation with a certain port of its, and marking down “yes” or “no” depending on the response. Normally, there’s lots of overhead as you keep track of who you sent requests to and thus who you’re expected responses from. Overhead, or “state”, makes things slow. So scanrand is stateless — right when you start up, it splits in two. One half asks everyone, “Heh! What are you hosting!” The other half picks up responses, “Hmmm, some guy just said he has a web server.”

Now, there’s a problem: If someone knows I’m not keeping track of who I’m scanning, they can just throw fake responses back at me. But TCP lets me embed a little signature with every connection request — the “Sequence Number”. This number will be returned to me when I get a valid response from a host that I scanned. So I take the IP and the port of the machine I scan, encrypt it into the sequence, and send off the request. When I get the response back, I look at the ACKnowledgement, compare it to the IP and port of the machine that’s talking to me, and immediately know whether I ever scanned this guy in the first place.

Unicornscan (2005)

https://www.kali.org/tools/unicornscan/

```
root@kali:~# unicornscan -mTsf -Iv -r 1000 192.168.0.102:a
adding 192.168.0.102/32 mode `TCPscan' ports `a' pps 1000
using interface(s) eth0
scaning 1.00e+00 total hosts with 6.55e+04 total packets, should take a little lo
connected 192.168.103.227:23221 -> 192.168.0.102:445
TCP open 192.168.0.102:445  ttl 128
connected 192.168.103.227:50006 -> 192.168.0.102:443
TCP open 192.168.0.102:443  ttl 128
connected 192.168.103.227:54487 -> 192.168.0.102:161
TCP open 192.168.0.102:161  ttl 128
connected 192.168.103.227:47765 -> 192.168.0.102:80
TCP open 192.168.0.102:80  ttl 128
connected 192.168.103.227:4267 -> 192.168.0.102:1884
TCP open 192.168.0.102:139  ttl 128
sender statistics 963.9 pps with 65536 packets sent total
listener statistics 131180 packets recieved 0 packets droped and 0 interface drop
TCP open                 http[   80]     from 192.168.0.102  ttl 128
TCP open          netbios-ssn[  139]     from 192.168.0.102  ttl 128
TCP open                 snmp[  161]     from 192.168.0.102  ttl 128
TCP open                https[  443]     from 192.168.0.102  ttl 128
TCP open        microsoft-ds[  445]     from 192.168.0.102  ttl 128
root@kali:~#
```

Zmap (2013)

- Fast scanner designed for horizontally scanning the entire Internet
- $2^{32} + 15 = 4294967311$ (prime number)
- Stateless scanner like the above examples, but open source, DMA to network card, ethical framework for measurements, *etc.*
  - Main contribution was a way to statelessly scan every IP (+ports in later versions) in a block

Three ways to enumerate scan targets

Every selected target once per pass

Original ZMap [1]

IPv4 addresses

$\mathrm{GF}(p)$

$p = 2^{32} + 15$

Period : $p - 1$

Skip excess values

Multiport ZMap [2]

IPv4 × port: 32 + 16 bits

$\mathrm{GF}(p)$

$p \approx 2^{48}$ (max)

Period : $p - 1$

Generator g; each prime q dividing p – 1:

$g^{(p-1)/q} \bmod p \neq 1$

$g < 2^{16} \rightarrow$ 64-bit products

Skip excess values

Binary extension field

n bits encode IP + port

$\mathrm{GF}(2^n)$

No excess values

Period : $2^n - 1$

Insert zero once for all 2^n targets

LSFR could have linear patterns

All three cover the full period before repeating.

ZMap: zmap.io

[1] Durumeric et al. “ZMap: Fast Internet-wide Scanning and Its Security Applications.” USENIX Security, 2013.

[2] Durumeric et al. “Ten Years of ZMap.” ACM Internet Measurement Conference (IMC), 2024.

Side channels...

Idle scan

- Every IP packet sent has an IP identifier
  - In case it gets fragmented along the way
- Old machines (or just that are configured that way) use a globally incrementing IPID that is shared state for all destinations

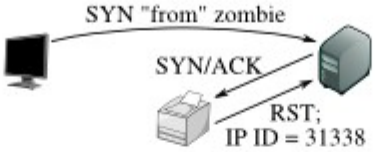
Figure 5.6. Idle scan of an open port

Step 1: Probe the zombie's IP ID.



The attacker sends a SYN/ACK to the zombie. The zombie, not expecting the SYN/ACK, sends back a RST, disclosing its IP ID.

Step 2: Forge a SYN packet from the zombie.



The target sends a SYN/ACK in response to the SYN that appears to come from the zombie. The zombie, not expecting it, sends back a RST, incrementing its IP ID in the process.

Step 3: Probe the zombie's IP ID again.



The zombie's IP ID has increased by 2 since step 1, so the port is open!

https://nmap.org/book/idlescan.html

Figure 5.7. Idle scan of a closed port

Step 1: Probe the zombie's IP ID.



The attacker sends a SYN/ACK to the zombie. The zombie, not expecting the SYN/ACK, sends back a RST, disclosing its IP ID. This step is always the same.

Step 2: Forge a SYN packet from the zombie.



The target sends a RST (the port is closed) in response to the SYN that appears to come from the zombie. The zombie ignores the unsolicited RST, leaving its IP ID unchanged.

Step 3: Probe the zombie's IP ID again.



The zombie's IP ID has increased by only 1 since step 1, so the port is not open.

https://jedcrandall.github.io/courses/cse468fall2026/idle-port-scan.svg

Assuming a 50%/50% chance of the target's port being open and no noise (*i.e.*, the zombie is idle), what's the mutual information between the port status and the IPID the attacker sees in the last step?

H(X) is the entropy of the port status
H(Y) is the entropy of the IPID

Mutual information
$I(X;Y) = H(X) – H(X|Y)$

$1 = 1 - 0$

Fragment injection bypasses DNS challenge fields

The real first fragment supplies the random port and TXID; one predicted IPID completes the forged response.

Legitimate first fragment from name server S

IPv4	total length	IPID	flags	frag offset	protocol	header checksum
v4/IHL5	1500	0x91B4	MF=1	0	17 UDP	0xC4F1

source IP	destination IP	UDP src port	UDP dst port	UDP length	UDP checksum
203.0.113.53	192.0.2.53	53	53012	2388	valid

DNS header	question	payload
TXID 0x7A3C	www.foo.com A	answers + DNSSEC records ...

keep fragment 1

Spoofed second fragment planted by attacker A

IPv4	total length	IPID	flags	frag offset	protocol	header checksum
v4/IHL5	928	0x91B4	MF=0	185	17 UDP	0xE674

spoofed source IP	destination IP	DNS answer tail	address
203.0.113.53	192.0.2.53	www.foo.com A	6.6.6.6

No UDP or DNS header appears here. A 16-bit field such as an RR TTL compensates so fragment 1's UDP checksum still verifies.

replace authentic fragment 2

Reassembled UDP DNS response accepted by the resolver

authentic IP/UDP/DNS headers + attacker-controlled DNS tail -> poisoned cache records

What the resolver uses

IPv4 reassembly key

```
src 203.0.113.53
dst 192.0.2.53
protocol UDP
IPID 0x91B4
```

From authentic fragment

UDP port 53012

DNS TXID 0x7A3C

Both arrived in the authentic first fragment.

Learned before query

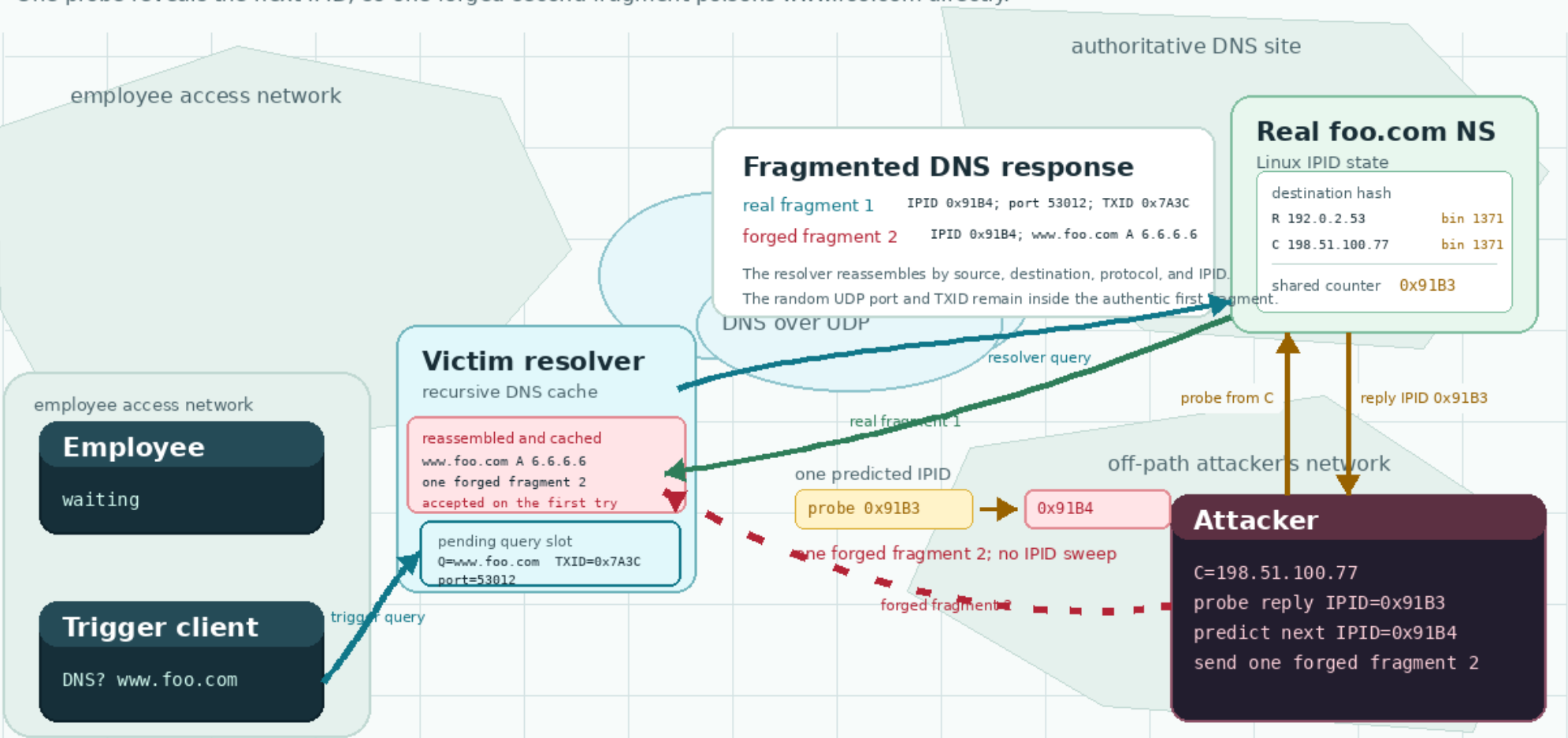
IPID 0x91B4

IPv4 example: Palmer and Somers, DEF CON 27 (2019); Herzberg and Shulman, 2013

offset 185 = 1480 bytes / 8

First-try DNS cache poisoning through IPID inference

One probe reveals the next IPID, so one forged second fragment poisons www.foo.com directly.



Sources Travis Palmer and Brian Somers, "First-Try DNS Cache Poisoning with IPv4 and IPv6 Fragmentation," DEF CON 27 slides/talk, 2019. doi.org/10.5446/48744

Amir Herzberg and Haya Shulman, "Fragmentation Considered Poisonous, or: One-Domain-to-Rule-Them-All.ORG," IEEE CNS 2013, pp. 224-232. doi.org/10.1109/CNS.2013.6682711

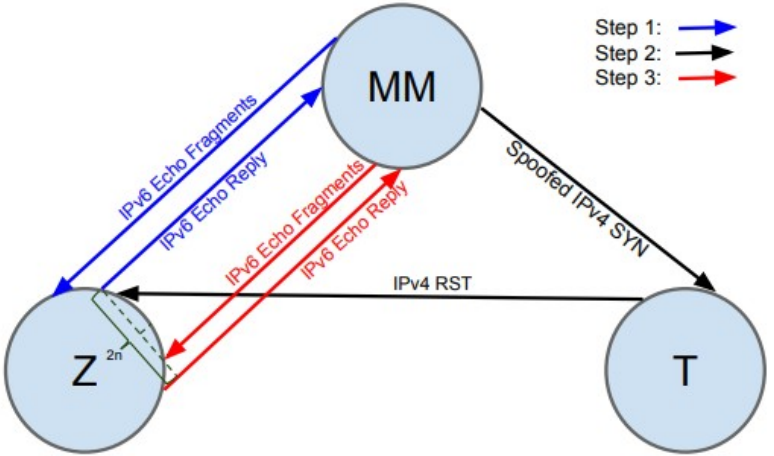


Fig. 4. Scan of a closed port with a dual stack zombie using ONIS.

https://jedcrandall.github.io/INFOCOM2018.pdf

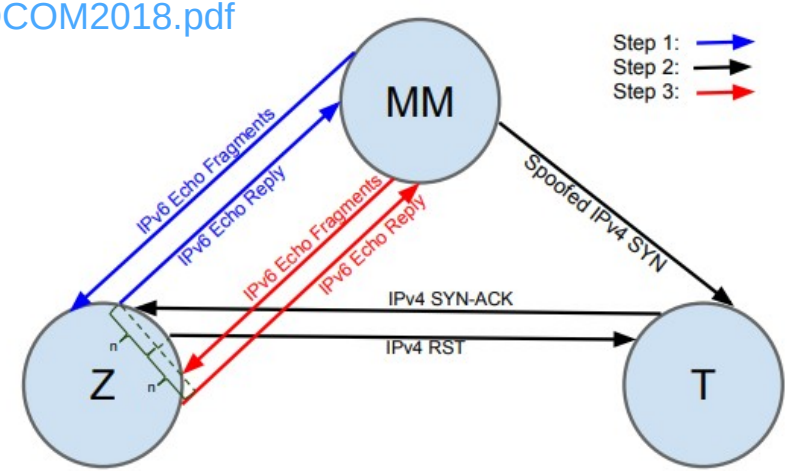
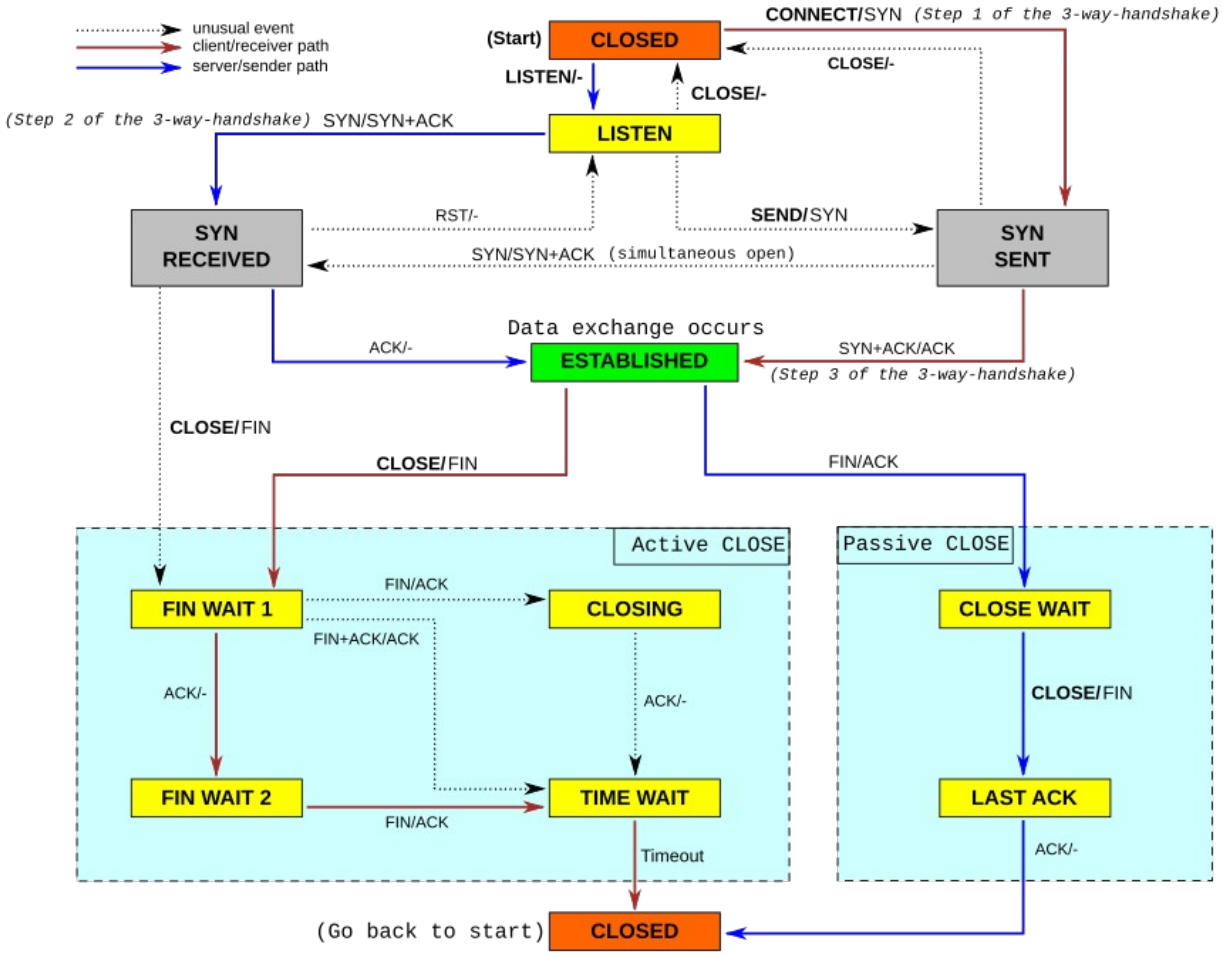


Fig. 5. Scan of an open port with a dual stack zombie using ONIS.

DoS in general

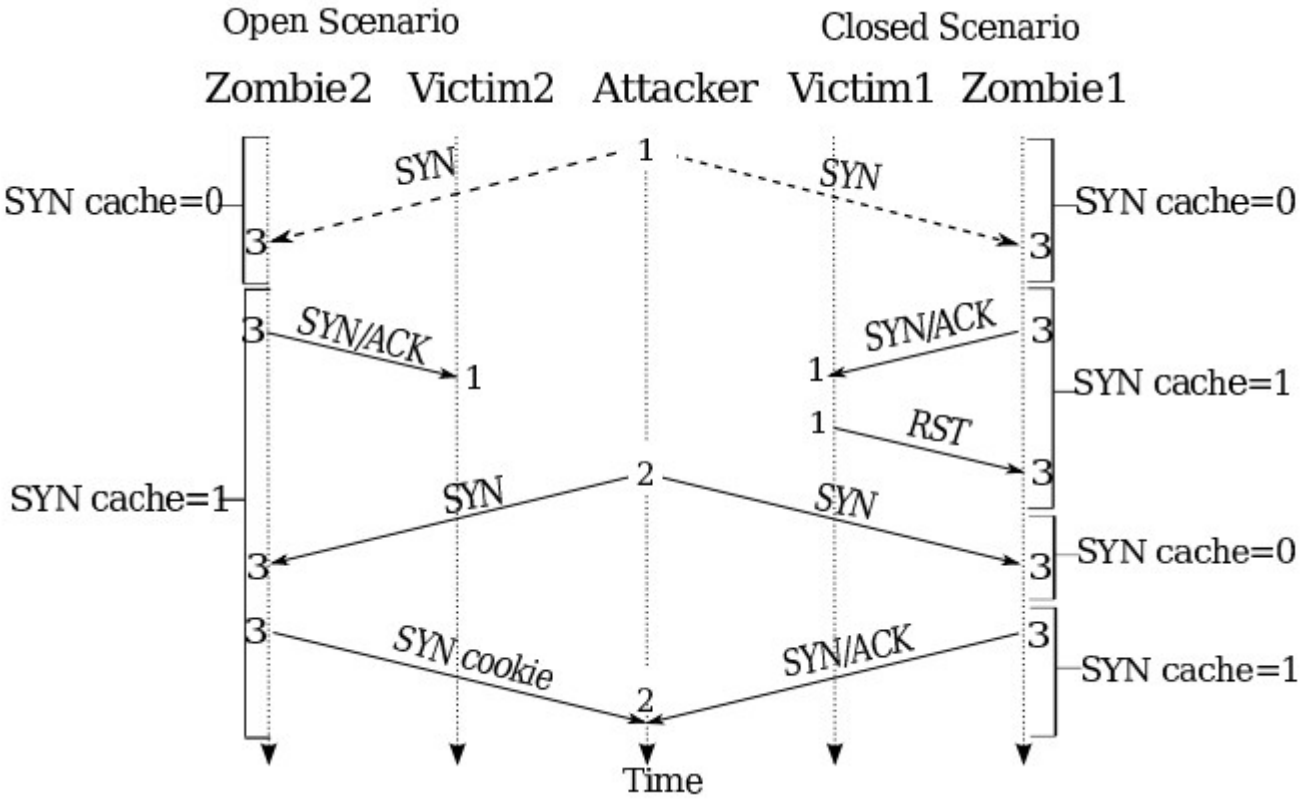
- Exhaust some kind of resource, *e.g.*:
  - Optimistic ACK to exhaust bandwidth
    - See https://homes.cs.washington.edu/~tom/pubs/CCR99.pdf
  - PING of death (*e.g.*, large PING) causes crash
  - Exhaust CPU in layer 7
  - More examples: http://www.isi.edu/~mirkovic/bench/attacks.html
  - SYN flood: Older hosts had either a fixed amount of half-open connections they could keep track of or no limitations at all; attack is to send lots of SYNs and never ACK or RST
    - Defenses: SYN backlog policies and SYN cookies

https://commons.wikimedia.org/wiki/File:Tcp_state_diagram_fixed_new.svg

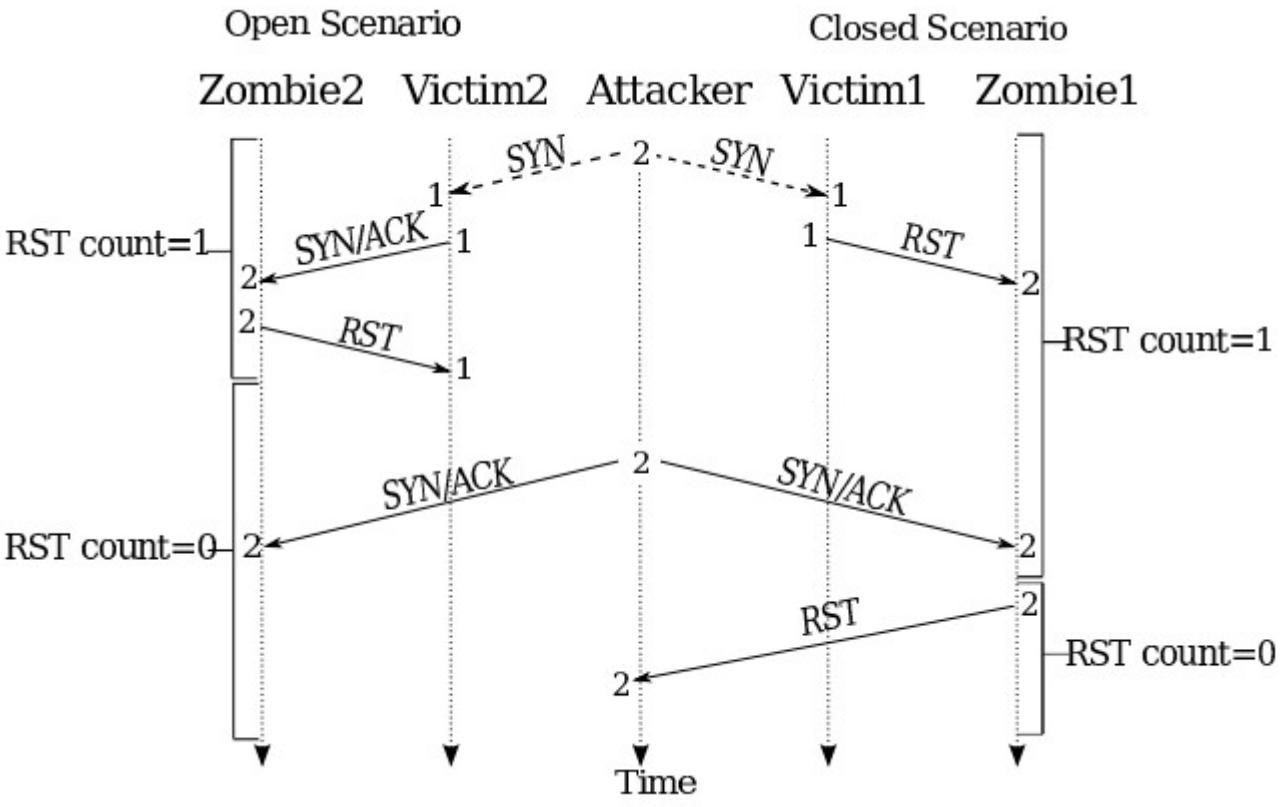


SYN cookies and SYN backlogs

- SYN cookies
  - Special kind of SYN/ACK
  - See https://cr.yp.to/syncookies.html
  - Can confirm ACK number and reconstruct the necessary state for a connection without having kept any state after sending the SYN cookie
    - Tuple info (source and destination IP addresses and ports) are hashed
- SYN backlog examples
  - Linux reserves ½, ¼, 1/8th, and so on for successively older SYNs, prunes 5 times a second
  - FreeBSD has 512 buckets of 30, you can't predict what bucket you fall into (in theory)



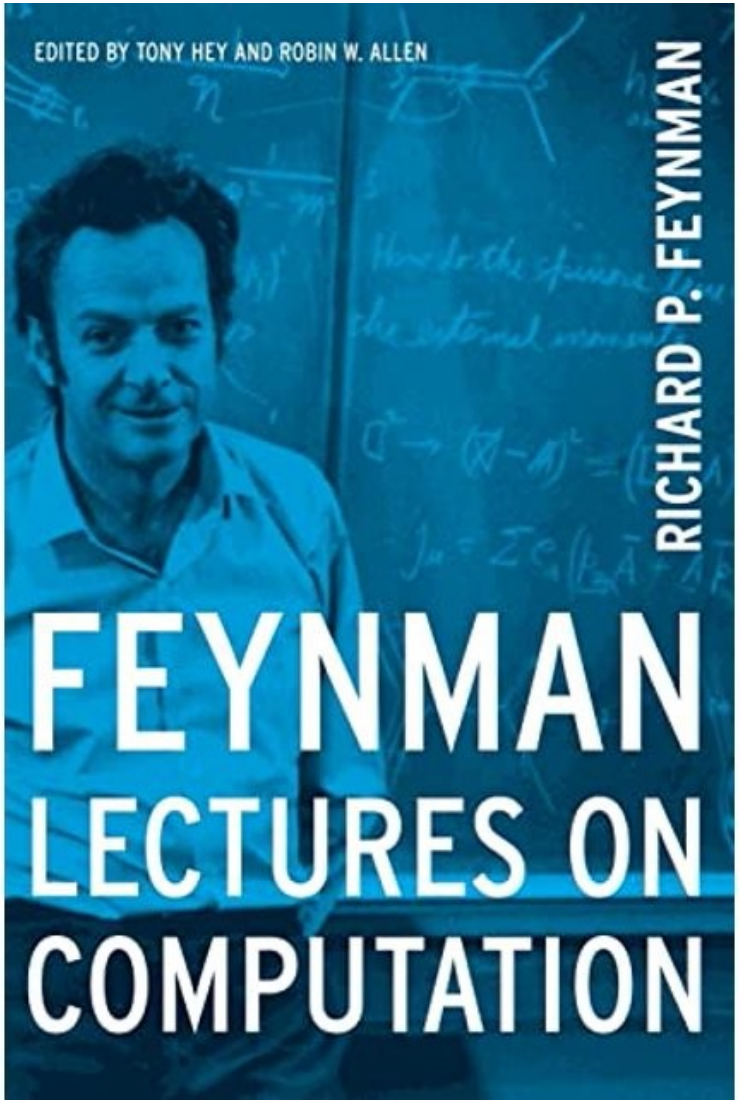
From… https://jedcrandall.github.io/usenix10.pdf



From… https://jedcrandall.github.io/usenix10.pdf

References

- *NMAP NETWORK SCANNING*, by Gordon “Fyodor” Lyon
- Google “nmap”, “idle scan”, etc.
- Other references were linked to inline


EDITED BY TONY HEY AND ROBIN W. ALLEN
RICHARD P. FEYNMAN
FEYNMAN
LECTURES ON
COMPUTATION