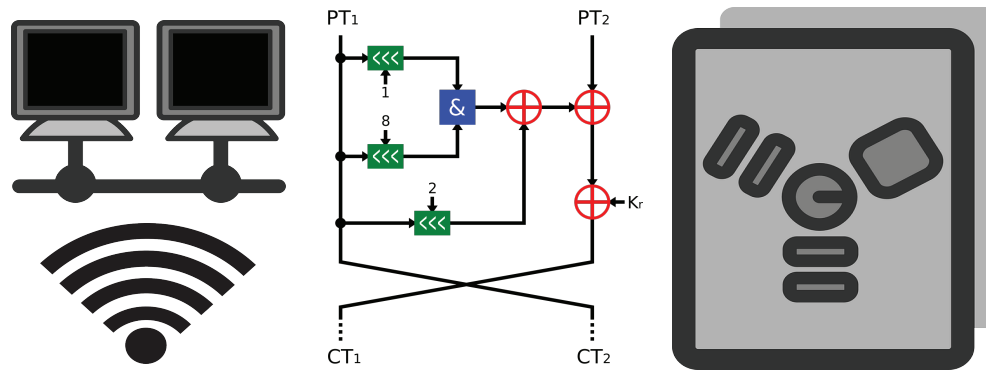


# CSE 40567 / 60567: Computer Security



Network Security 3

Homework #6 has been released. It is due on  
4/16 at 11:59PM (your timezone)

See **Assignments Page** on the course  
website for details

# Network Reconnaissance

# Network ports

- TCP and UDP identify applications using 16-bit port numbers
- Well-known service ports traditionally between 1-1023
- Ephemeral ports between 1024-5000
- Other services above 5000

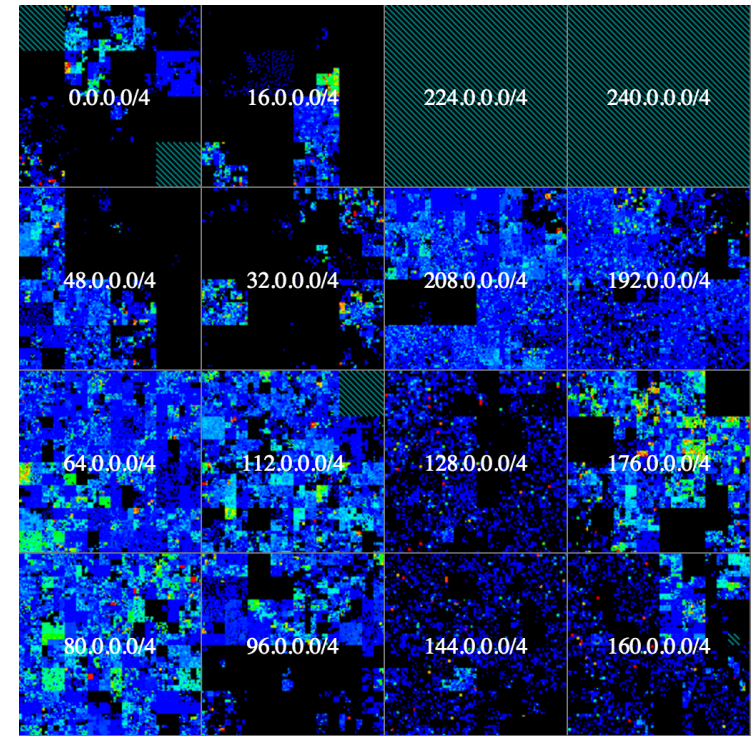
## Server



Web: TCP 80  
SMTP: TCP 25  
SSH: TCP 22

# Portscanning

- Scan for open ports on a machine or network
- Used by attackers to search for vulnerable services
- Not necessarily an attack
  - Used by administrators to secure networks and assess exposure to risk
  - Used by researchers to map the Internet



# nmap (<https://nmap.org/>)

```
$ nmap
Nmap 5.21 ( http://nmap.org )
Usage: nmap [Scan Type(s)] [Options] {target
specification}
```

## SCAN TECHNIQUES:

- sS/sT/sA/sW/sM: TCP SYN/Connect()/ACK/Window/  
Maimon scans
- sU: UDP Scan
- sN/sF/sX: TCP Null, FIN, and Xmas scans
- scanflags <flags>: Customize TCP scan flags
- sI <zombie host[:probeport]>: Idle scan
- sY/sZ: SCTP INIT/COOKIE-ECHO scans
- sO: IP protocol scan
- b <FTP relay host>: FTP bounce scan

# TCP connect () Scanning

Classic method: simply use connect() call to establish a full connection (3-way handshake) with every open port on target

```
$ nmap -sT 140.247.178.194
```

```
Starting Nmap 5.21 ( http://nmap.org ) at 2016-02-15 16:31 EST  
Nmap scan report for dhcp-140-247-178-194.fas.harvard.edu  
(140.247.178.194)
```

```
Host is up (0.00016s latency).
```

```
Not shown: 998 closed ports
```

PORT	STATE	SERVICE
22/tcp	open	ssh
5900/tcp	open	vnc

# What ends up in the logs?

```
/var/log/auth.log on 140.247.178.194
```

```
Feb 15 16:27:37 eve sshd[25848]: Did not  
receive identification string from  
140.247.178.71
```

**Red Flag**



**Host we  
scanned from**

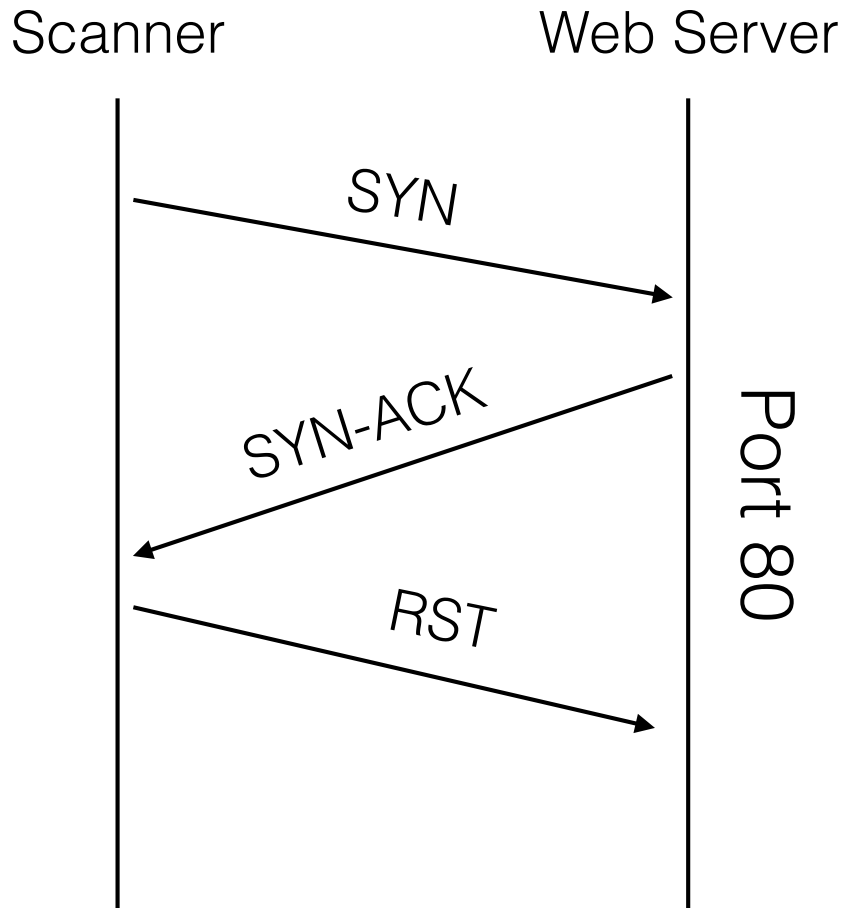


# Stealth Scanning

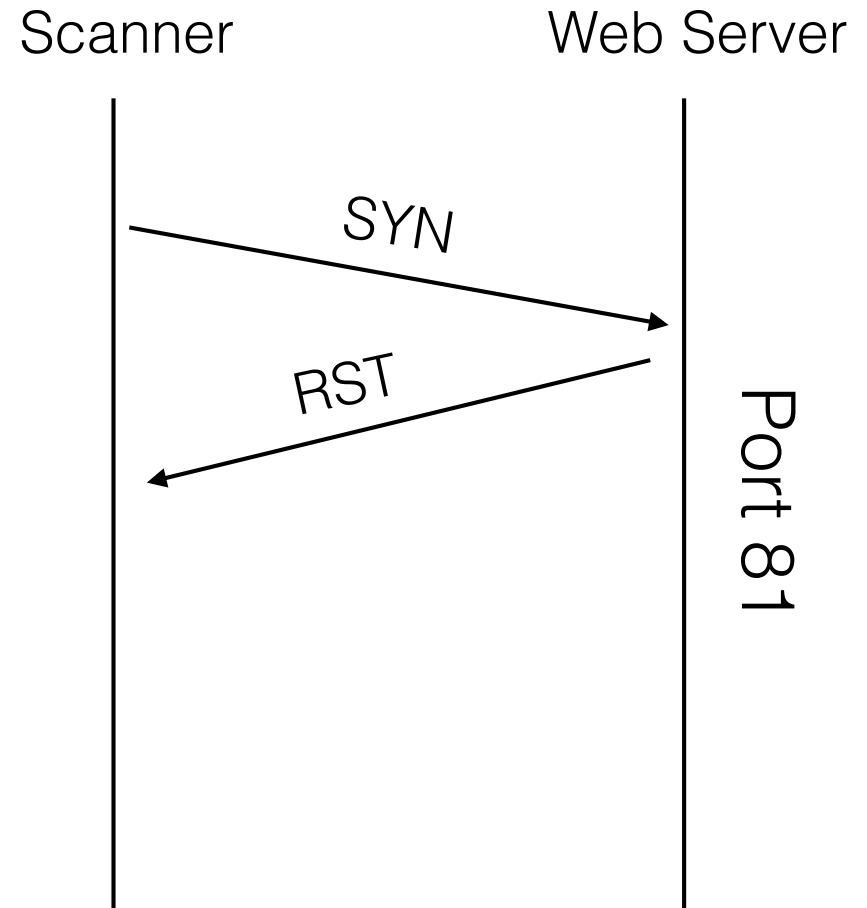
- Goal: no record in system logs
  - Can be achieved by leveraging established protocol behavior in creative ways
  - Raw sockets (need root privilege)
- NIDS can easily catch these approaches
  - *In most cases* — we'll learn about some evasion techniques shortly

# TCP SYN Scanning (-sS)

## 1. Open Port



## 2. Closed Port



## 3. Filtered Port: no response

# TCP NULL, FIN, Maimon and Xmas scans (-sN/sF/sM/sX)

Page 65 of RFC 793: “if the [destination] port state is CLOSED .... an incoming segment not containing a RST causes a RST to be sent in response.”

What to do if packets are sent to an open port without the SYN, RST, or ACK bits set: “you are unlikely to get here, but if you do, drop the segment, and return.”

Any packet not containing SYN, RST, or ACK bits will result in a returned RST if the port is closed and no response if the port is open.

# TCP NULL, FIN, Maimon and Xmas scans (-sN/sF/sM/sX)

- Null scan: Does not set any bits (TCP flag header is 0)
- FIN scan: Sets only the TCP FIN bit.
- Maimon: Sets FIN/ACK flags
- Xmas scan: Sets the FIN, PSH, and URG flags, “lighting the packet up like a Christmas tree” (nmap documentation)

These four scan types are functionally equivalent

Closed port: RST packet is received

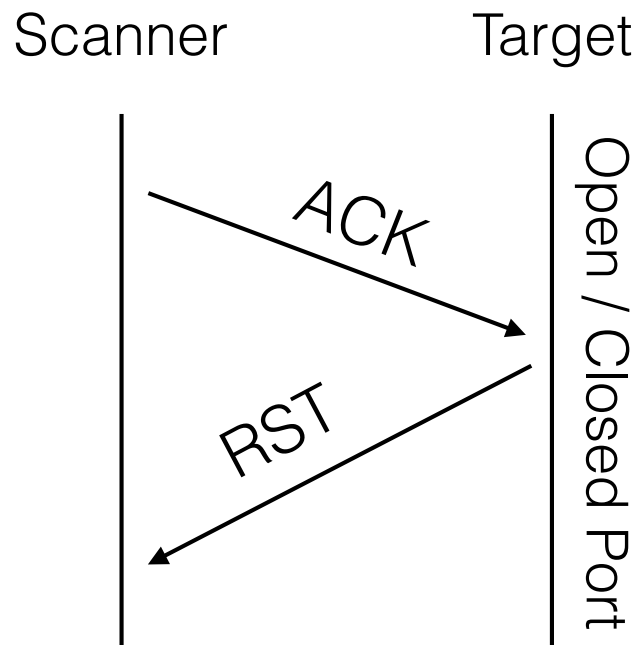
Open or filtered port: no response

# TCP ACK scan

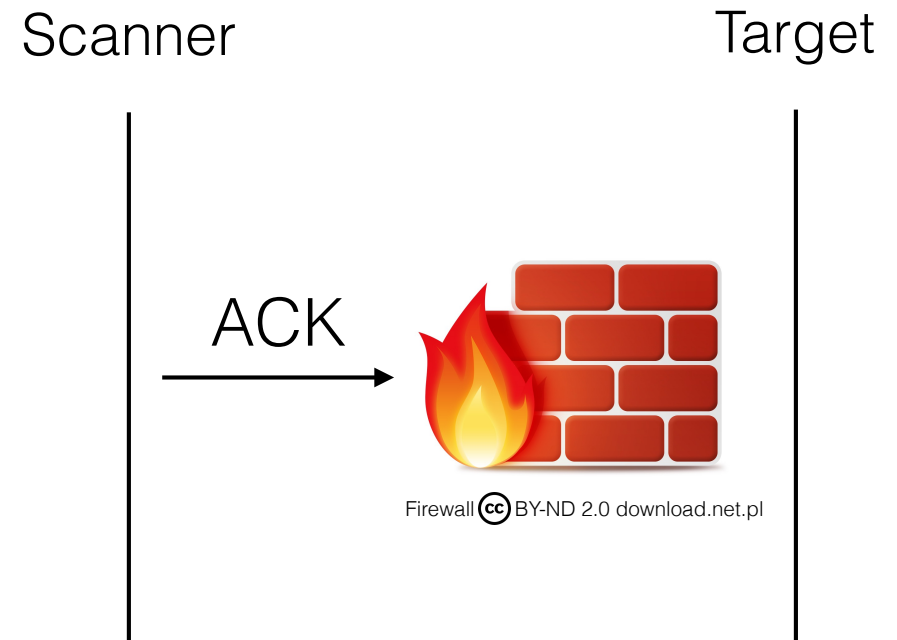
Used to map firewall rulesets

- ▶ Determine whether they are stateful
- ▶ Determine which ports are filtered

## Firewall Passes Traffic



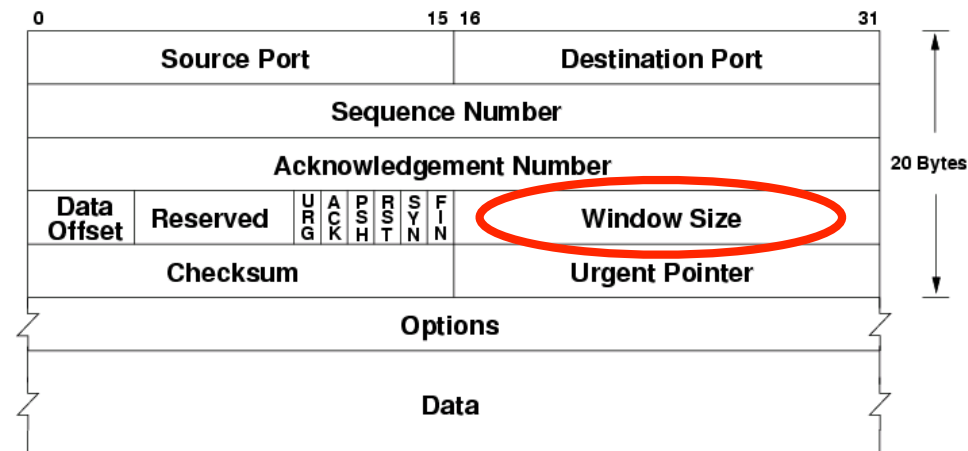
## Firewall Blocks Traffic



# TCP window scan

Exactly the same as the TCP ACK scan, but can distinguish open ports from closed ports

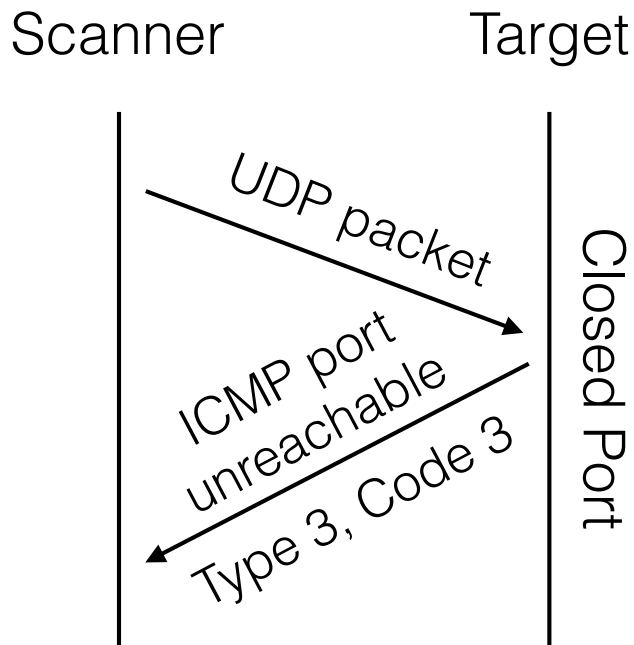
- ▶ Examine TCP Window field of RST packets
- ▶ Open ports often return a positive Window size
- ▶ Closed ports often return a Window size of 0



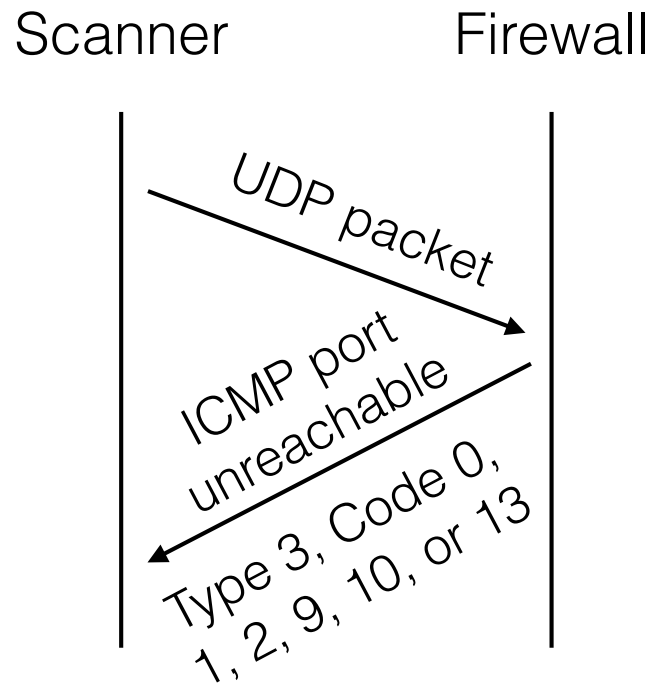
# UDP scan

UDP isn't stateful like TCP, but there are various conditions that can be checked to infer the state of a port:

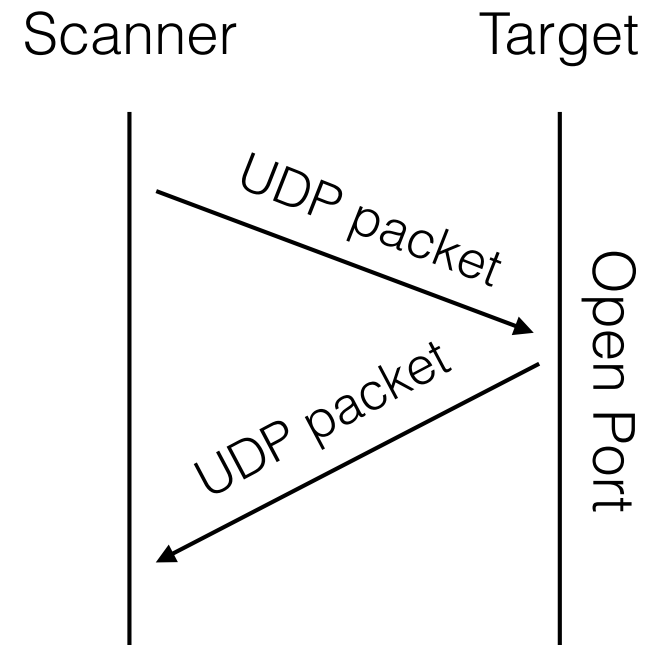
## Closed



## Filtered



## Open



# Slow is the way to go

-T **paranoid** | **sneaky** | polite | normal | aggressive | insane

- There is often an inclination to scan networks as fast as possible
  - ▶ NIDS will catch this right away
- A better strategy: scan very slowly
  - ▶ Better chance of evading NIDS
  - ▶ Paranoid mode serializes the scan so only one port is scanned at a time, and waits five minutes between sending each probe



# OS Fingerprinting (-O)

```
# nmap -O 128.198.147.13
```

```
Starting Nmap 5.21 ( http://nmap.org ) at 2016-02-15 21:03 MST  
Insufficient responses for TCP sequencing (3), OS detection may  
be less accurate
```

```
Interesting ports on printer.vast.uccs.edu (128.198.147.13):
```

```
Not shown: 1708 closed ports
```

PORT	STATE	SERVICE
------	-------	---------

21/tcp	open	ftp
--------	------	-----

23/tcp	open	telnet
--------	------	--------

80/tcp	open	http
--------	------	------

280/tcp	open	http-mgmt
---------	------	-----------

515/tcp	open	printer
---------	------	---------

631/tcp	open	ipp
---------	------	-----

9100/tcp	open	jetdirect
----------	------	-----------

```
MAC Address: 00:60:B0:71:68:0C (Hewlett-packard CO.)
```

```
Device type: print server
```

```
Running: HP embedded
```

```
OS details: HP 170X JetDirect print server (J3258B)
```

```
Network Distance: 1 hop
```

<https://nmap.org/book/osdetect.html>

# Motivation for remote OS fingerprinting

- Determining vulnerability of target hosts
- Tailoring exploits
- Network inventory and support
- Detecting unauthorized and dangerous devices
- Social engineering

# Probes Sent

## **16 probe packets**

- Sequence generation (SEQ, OPS, WIN, and T1)
- ICMP echo (IE)
- TCP explicit congestion notification (ECN)
- TCP (T2–T7)
- UDP (U1)

# Sequence generation (SEQ, OPS, WIN, and T1)

## SYN Packets

Packet #1: window scale (10), NOP, MSS (1460), timestamp (TSval: 0xFFFFFFFF; TSecr: 0), SACK permitted. The window field is 1.

Packet #2: MSS (1400), window scale (0), SACK permitted, timestamp (TSval: 0xFFFFFFFF; TSecr: 0), EOL. The window field is 63.

Packet #3: Timestamp (TSval: 0xFFFFFFFF; TSecr: 0), NOP, NOP, window scale (5), NOP, MSS (640). The window field is 4.

Packet #4: SACK permitted, Timestamp (TSval: 0xFFFFFFFF; TSecr: 0), window scale (10), EOL. The window field is 4.

Packet #5: MSS (536), SACK permitted, Timestamp (TSval: 0xFFFFFFFF; TSecr: 0), window scale (10), EOL. The window field is 16.

Packet #6: MSS (265), SACK permitted, Timestamp (TSval: 0xFFFFFFFF; TSecr: 0). The window field is 512.

# ICMP echo (IE)

Packet #1: IP DF bit set, a type-of-service (TOS) byte value of 0, a code of 9 (ordinarily 0), sequence number 295, random IP ID and ICMP request identifier, and 120 bytes of 0x00 for the data payload.

Packet #2: TOS of four (IP\_TOS\_RELIABILITY) is used, code is 0, 150 bytes of data is sent, and the ICMP request ID and sequence numbers are incremented by one from the previous query values.

# TCP explicit congestion notification (ECN)

SYN packet which also has the ECN CWR and ECE congestion control flags set.

- ▶ For an unrelated (to ECN) test, the urgent field value of 0xF7F5 is used even though the urgent flag is not set.
- ▶ The acknowledgment number is 0, sequence number is random, window size field is 3, and the reserved bit which immediately precedes the CWR bit is set.
- ▶ TCP options are WScale (10), NOP, MSS (1460), SACK permitted, NOP, NOP. The probe is sent to an open port.

# TCP (T2–T7)

T2 sends a TCP null (no flags set) packet with the IP DF bit set and a window field of 128 to an open port.

T3 sends a TCP packet with the SYN, FIN, URG, and PSH flags set and a window field of 256 to an open port. The IP DF bit is not set.

T4 sends a TCP ACK packet with IP DF and a window field of 1024 to an open port.

T5 sends a TCP SYN packet without IP DF and a window field of **31337** to a closed port.

T6 sends a TCP ACK packet with IP DF and a window field of 32768 to a closed port.

T7 sends a TCP packet with the FIN, PSH, and URG flags set and a window field of 65535 to a closed port. The IP DF bit is not set.

# UDP (U1)

- This probe is a UDP packet sent to a closed port.
  - Expect ICMP port unreachable packet if no firewall
- The character 'C' (0x43) is repeated 300 times for the data field.
- The IP ID value is set to 0x1042 for operating systems which allow us to set this.



# Response Tests

- Probe packets exploit ambiguities in the standard protocol RFCs
- Dozens of attributes in those responses are analyzed and combined to generate a **fingerprint**

Apple Mac OS X Server 10.2.8 (Jaguar) (Darwin 6.8, PowerPC)  
W1=8218, W2=8220, W3=8204, W4=80E8, W5=80F4, W6=807A



OS-specific Window values

# nmap reference fingerprint

Fingerprint Apple Mac OS X Server 10.2.8 (Jaguar) (Darwin 6.8, PowerPC)

Class Apple | Mac OS X | 10.2.X | general purpose

CPE cpe:/o:apple:mac\_os\_x:10.2.8

SEQ(SP=FB-111%GCD=1-6%ISR=104-10E%TI=I%II=I%SS=S%TS=1)

OPS(O1=M5B4NW0NNT11%O2=M5B4NW0NNT11%O3=M5B4NW0NNT11%O4=M5B4NW0NNT11%O5=M5B4NW0NNT11%O6=M5B4NNT11)

WIN(W1=8218%W2=8220%W3=8204%W4=80E8%W5=80F4%W6=807A)

ECN(R=Y%DF=Y%T=3B-45%TG=40%W=832C%O=M5B4NW0%CC=N%Q=)

T1(R=Y%DF=Y%T=3B-45%TG=40%S=O%A=S+%F=AS%RD=0%Q=)

T2(R=N)

T3(R=Y%DF=Y%T=3B-45%TG=40%W=807A%S=O%A=S+%F=AS%O=M5B4NW0NNT11%RD=0%Q=)

T4(R=Y%DF=Y%T=3B-45%TG=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)

T5(R=Y%DF=N%T=3B-45%TG=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)

T6(R=Y%DF=Y%T=3B-45%TG=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)

T7(R=Y%DF=N%T=3B-45%TG=40%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=)

U1(DF=N%T=3B-45%TG=40%IPL=38%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=0%RUD=G)

IE(DFI=S%T=3B-45%TG=40%CD=S)

# Passive Fingerprinting

Downside to active fingerprint: easily detected by NIDS

Alternative: sniff the network and analyze packets that come your way

pof v3:

```
Detected OS      = Mac OS X [generic]
HTTP client      = Chrome 43.x or newer (User-Agent string
seems legit)
Network link     = unknown
Distance        = 9
Language        = English
Uptime          = 16 days 15 hrs 15 min (modulo 55 days)
```

<http://lcamtuf.coredump.cx/p0f3/>

# Attacks Against DNS

# Domain Name System (DNS)

**Distributed** database that maps between hostnames and IP addresses

Each site maintains its own database of information and runs a server program (e.g., `bind`) that systems across the Internet can query

`gethostbyname(3)`

```
$ host nd.edu
nd.edu has address
52.6.129.12
```

`gethostbyaddr(3)`

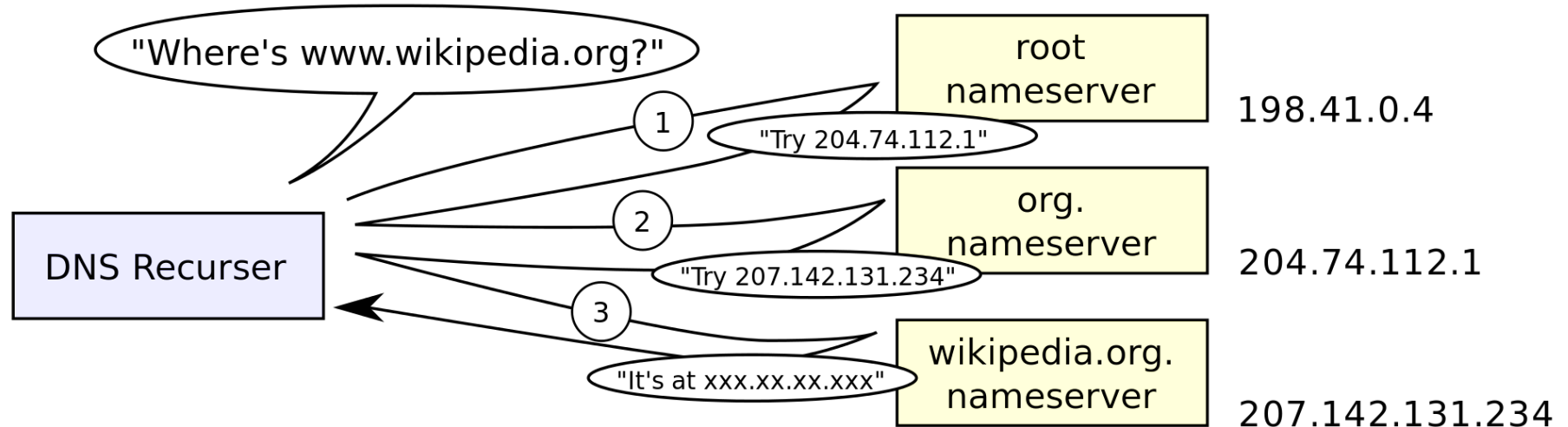
```
$ host 52.6.129.12
12.129.6.52.in-addr.arpa
domain name pointer
ec2-52-6-129-12.compute-1.
amazonaws.com. (!)
```

# Resource Record Set

- DNS stores Resource Records (RRs)
- Each DNS resolver or authoritative server stores RRs in its cache or local zone file
  - Examples:
    - An A record holds a mapping from a domain name to an IP address
    - An NS record holds a mapping from a domain name to the name of an authoritative name server for that domain
- No two RRs in the cache may have the same label, class, type, and data

# Recursive Resolution

If the matching label for a query isn't in cache, follow referrals till you find it



# Cache Poisoning

- Most prominent and dangerous attack against DNS
- Server caches invalid or malicious mappings between names and IP addresses
- Process of resolving a name depends on authoritative servers
  - ▶ Attacker can compromise an authoritative name server (hard)
  - ▶ Or forge a response to a recursive DNS query sent by a resolver to an authoritative server (easier)

```
$ host nd.edu  
nd.edu has address 52.6.129.12 ← Normal
```

```
$ host nd.edu  
nd.edu has address 128.11.75.2 ← Attack is difficult to detect
```



# Cache poisoning without response forgery

In the early days: owner of any authoritative DNS server could compromise records for **any** domain name

When responding to a query, a malicious authoritative server can send (in the additional section of its response) an arbitrary mapping from any domain name to IP.

- **Including those outside its authority**

Example:

1. Client queries malicious authoritative server for `bad.com` to resolve `www.bad.com`
2. Server includes response with additional section mapping `ns1.good.com` to a malicious IP address

# Defense: Bailiwick Rule

- Response is accepted by the DNS resolver and stored in its cache only if the RRset of each section passes the *bailiwick rule*
- Set of specific conditions
- Not part of the DNS specification and depends on the implementation of the resolver
  - ▶ BIND is not the same as MaraDNS

# BIND-specific checks (Bailiwick Rule)

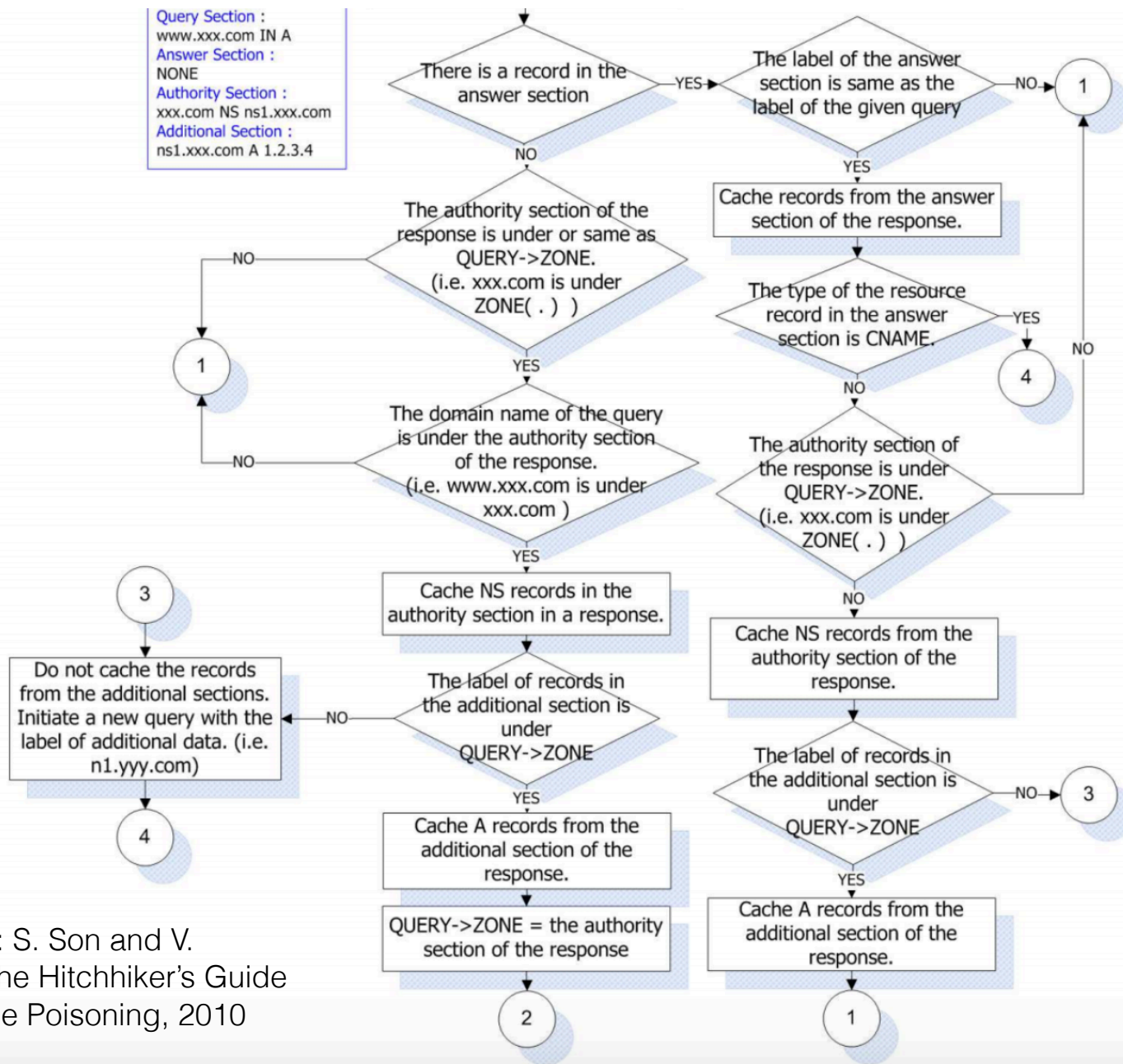


Image credit: S. Son and V. Shmatikov, The Hitchhiker's Guide to DNS Cache Poisoning, 2010

# Blind response forgery using birthday attack

The basic DNS protocol does not authenticate responses to recursive queries. Only two checks:

1. Query section and 16-bit transaction ID (TXID) of the response must match those of the query
2. Source IP address and destination port of the response must match, respectively, the destination IP address and source port of the query

## **Birthday Paradox**

If the TXID has only  $N$  bits of entropy, attacker needs only  $O(2^{N/2})$  trials on average to generate a forged response

# Blind response forgery using birthday attack

## Query Section:

www.google.com IN A

## Answer Section:

www.google.com IN A 1.2.3.4

## Authority Section:

.google.com IN NS ns2.google.com

.google.com IN NS ns1.google.com

## Additional Section:

ns1.google.com IN A 1.2.3.5

ns2.google.com IN A 1.2.4.6

Authoritative  
server



Generic server icon © BY-SA 4.0 Jorgeprof

Resolver



server, icon © BY-SA 3.0 RRZEicons

## Query Section :

www.google.com IN A

Forged TXID

Forged TXID

Forged TXID



## Query Section:

www.google.com IN A

## Answer Section:

www.google.com IN A 6.6.6.6

## Authority Section:

.google.com IN NS ns2.google.com

.google.com IN NS ns1.google.com

## Additional Section:

ns1.google.com IN A 1.2.3.5

ns2.google.com IN A 1.2.4.6

# Kaminsky's attack

Three additional facets to the basic birthday attack:

1. Attacker can force the target resolver to initiate a query to a chosen authoritative server
2. Modern attackers have enough network bandwidth to generate a large number of spoofed responses
3. The malicious “payload” of the forged response is the additional section (as opposed to the answer section in the conventional attack)


# Kaminsky's attack

1. Attacker chooses a target domain name to compromise (e.g., `www.google.com`)
2. Attacker queries target resolver with any subdomain that is not already cached on the resolver (e.g., non-existent subdomain `xyz12.google.com`)
  - ▶ This causes the target resolver to send a query to the authoritative server(s) for this domain
3. Attacker floods the resolver with a large number of forged responses, each containing a different guess of the query's transaction ID

# Kaminsky's attack

## Forged Response

```
Query Section:  
xyz12.google.com IN A  
Answer Section:  
NONE  
Authority Section:  
.google.com IN NS www.google.com  
Additional Section:  
www.google.com IN A 6.6.6.6
```



Referral, not an answer

False Information

- The attack, if successful, introduces into the target resolver's cache a false mapping for an authoritative server
- Future queries from the compromised resolver will be sent directly to an attacker-controlled IP address



# Defenses

- Source port randomization
- 0x20-bit encoding (mix the upper and lower case spelling of the domain name in the query)
- Extended Query ID (63 alpha-numeric characters in the QNAME)
- WSEC-DNS (wild-card domain names)
- DNSSEC (cryptographic extensions to DNS)
  - Origin authentication of DNS data, authenticated denial of existence, and data integrity

No confidentiality or authentication

# Response forgery using eavesdropping

DNS remains vulnerable to non-blind attacks

- compromised servers
- network eavesdroppers

