Network Defenses

INGI2347: COMPUTER SYSTEM SECURITY (Spring 2016)

Marco Canini

Université catholique de Louvain

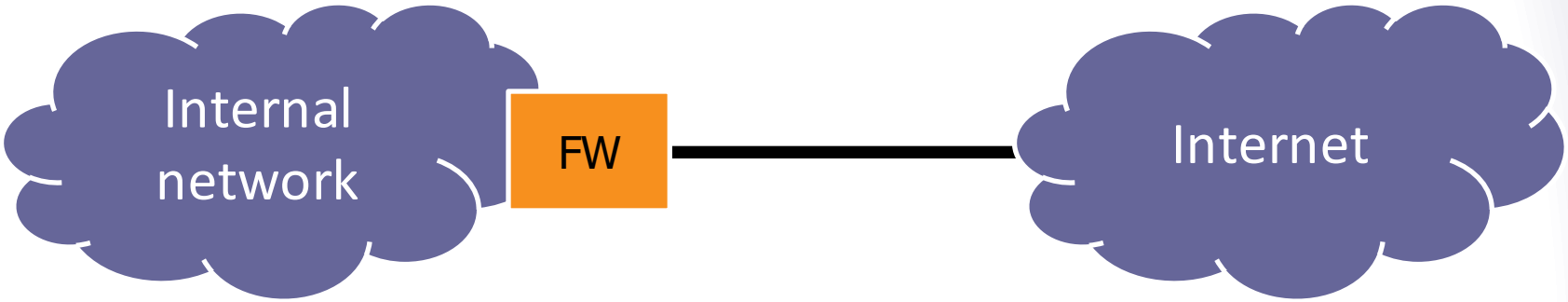
Lecture slides adapted from Upenn CSE331 by Steve Zdancewic and UC Berkeley CS161 by Vern Paxson
Reproduced with permission

Controlling Networks … On The Cheap

- How do you harden a set of systems against external attack?
  - *Key Observation:*
    - ***The more network services your machines run, the greater the risk***
  - Due to larger attack surface
- One way: on each system, turn off unnecessary network services
  - But you have to know ***all*** the services that are running
  - And sometimes some trusted remote users still require access
- Plus key question of scaling
  - What happens when you have to secure 100s/1000s of systems?
  - Which may have different OSs, hardware & users …
  - Which may in fact not all even be identified …

Taming Management Complexity

- Possibly more scalable defense: Reduce risk by blocking *in the network* outsiders from having unwanted access your network services
  - The traffic to/from the outside must traverse a **firewall**
  - Chokepoint can cover 1000s of hosts
    - Where in everyday experience do we see such chokepoints?



Selecting a Security Policy

- Effectiveness of firewall relies on deciding what policy it should implement:
  - *Who is allowed to talk to whom, accessing what service?*
- Distinguish between inbound & outbound connections
  - Inbound: attempts by external users to connect to internal services
  - Outbound: internal users to external services
  - Why? Because fits with a common ***threat model***
- Conceptually simple *access control policy*:
  - Permit inside users to connect to any service
  - External users restricted:
    - Permit connections to services meant to be externally visible
    - Deny connections to services not meant for external access

How To Treat Traffic Not Mentioned in Policy?

- **Default Allow**: start off permitting external access to services
  - Shut them off as problems recognized

How To Treat Traffic Not Mentioned in Policy?

- **Default Allow**: start off permitting external access to services
  - Shut them off as problems recognized
- **Default Deny**: start off permitting just a few known, well-secured services
  - Add more when users complain (and mgmt. approves)

How To Treat Traffic Not Mentioned in Policy?

- **Default Allow**: start off permitting external access to services
  - Shut them off as problems recognized

✓ **Default Deny**: start off permitting just a few known, well-secured services
  - Add more when users complain (and mgt. approves)

- Pros & Cons?

In general, use Default Deny

  - Flexibility vs. conservative design
  - Flaws in Default Deny get noticed more quickly / less painfully

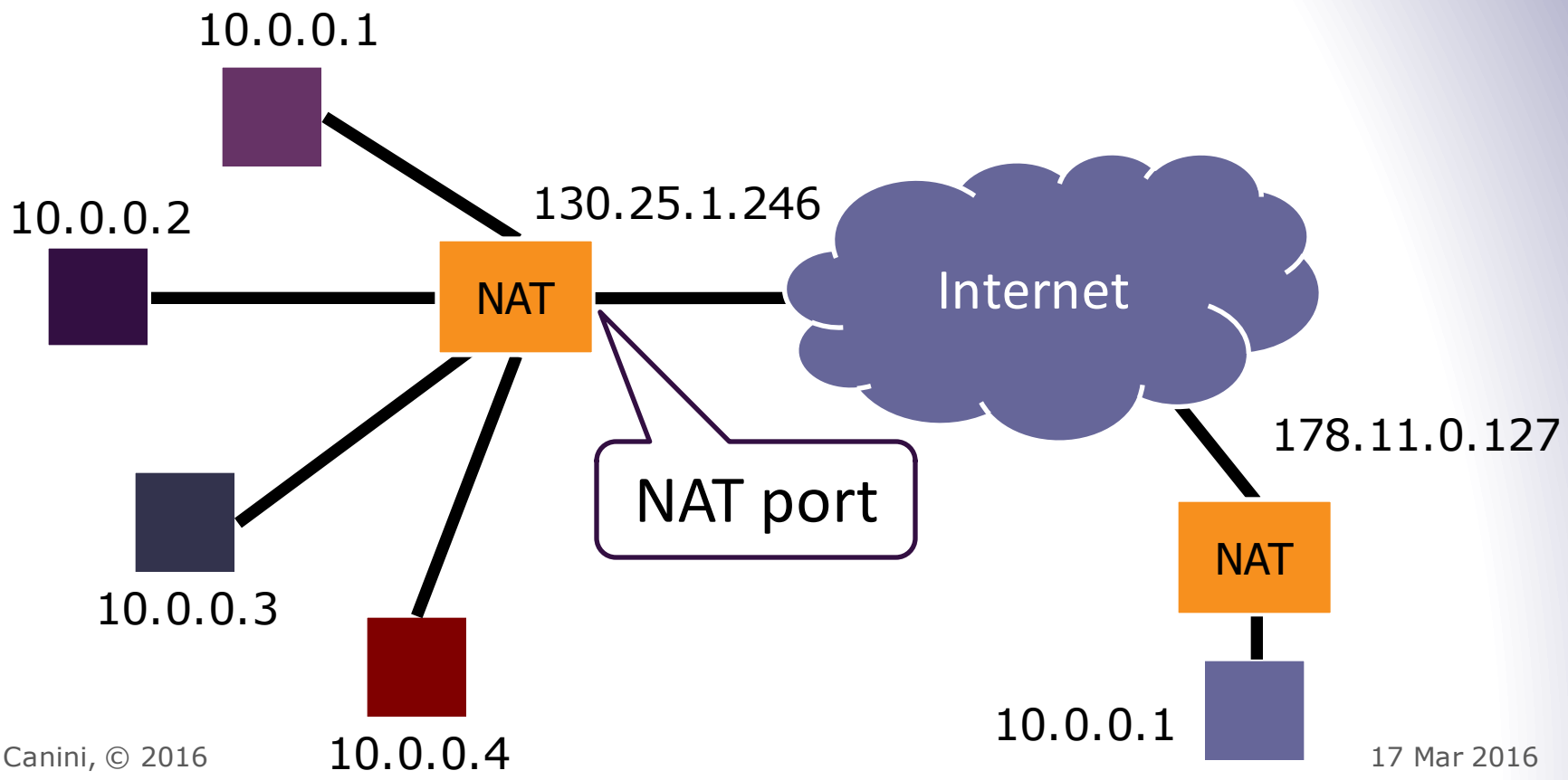
+

Network Address Translation

NAT

Network Address Translation (NAT)

- Idea: Break the invariant that IP addresses are globally unique



Dynamic NAT

- Basic principle: Maintain a table of the form:

<client IP> <client port> ⇄ <NAT ID>

- Outgoing packets (on non-NAT port):
  - Lookup client (source) IP address, client port in the mapping table
  - If not found, allocate a new unique NAT ID and replace client port with chosen NAT ID (same size as port = 2^{16})
  - If found, replace client port with previously allocated NAT ID
  - Replace client address with NAT address

Dynamic NAT

- Basic principle: Maintain a table of the form:

<client IP> <client port> ⇄ <NAT ID>

- Incoming packets (on NAT port)
  - Look up destination port number as NAT ID in port mapping table
  - If found, replace destination address and port with client entries from the mapping table
  - If not found, the packet is not for us and should be rejected

Dynamic NAT

- Unused table entries expire periodically
  - Example: after 2-3 minutes
- Dynamic NAT doesn't allow establishing incoming connections
  - Good protection by default

Mapping Table Example

Protocol	Local IP	Local port	NAT IP	NAT ID	Peer IP	Peer port
TCP	192.168.0.1	1912	81.242.186.64	1912	192.178.100.4	80
TCP	192.168.0.1	1913	81.242.186.64	23745	192.178.100.4	80
TCP	192.168.0.2	1912	81.242.186.64	55468	212.27.63.3	80
UDP	192.168.0.3	18551	81.242.186.64	1912	83.170.84.81	26000

Mapping Table Example

- NAT ID must be unique

Protocol	Local IP	Local port	NAT IP	NAT ID	Peer IP	Peer port
TCP	192.168.0.1	1912	81.242.186.64	1912	192.178.100.4	80
TCP	192.168.0.1	1913	81.242.186.64	23745	192.178.100.4	80
TCP	192.168.0.2	1912	81.242.186.64	55468	212.27.63.3	80
UDP	192.168.0.3	18551	81.242.186.64	1912	83.170.84.81	26000

Mapping Table Example

- Mapping can hide auto-increasing port numbers

Protocol	Local IP	Local port	NAT IP	NAT ID	Peer IP	Peer port
TCP	192.168.0.1	1912	81.242.186.64	1912	192.178.100.4	80
TCP	192.168.0.1	1913	81.242.186.64	23745	192.178.100.4	80
TCP	192.168.0.2	1912	81.242.186.64	55468	212.27.63.3	80
UDP	192.168.0.3	18551	81.242.186.64	1912	83.170.84.81	26000

Mapping Table Example

- Protocol info. further demultiplexes mapping entries

Protocol	Local IP	Local port	NAT IP	NAT ID	Peer IP	Peer port
TCP	192.168.0.1	1912	81.242.186.64	1912	192.178.100.4	80
TCP	192.168.0.1	1913	81.242.186.64	23745	192.178.100.4	80
TCP	192.168.0.2	1912	81.242.186.64	55468	212.27.63.3	80
UDP	192.168.0.3	18551	81.242.186.64	1912	83.170.84.81	26000

Static NAT

- To allow incoming connections, we have to define certain static entries in the mapping table
- Typically we create one entry per protocol
  - Example: SSH (22), HTTP (80), SMTP (25), …
- Example:

Protocol	Local IP	Local port	NAT port	Peer IP & port
TCP	192.168.0.1	22	22	*
TCP	192.168.0.1	8080	80	*
UDP	192.168.0.3	26000	26000	*

Benefits of NAT

- Dynamic NAT only allows outbound connections established from internal network
  - External hosts can only contact internal hosts that appear in the mapping table, which are only added once they establish a connection
- Hides the internal network structure
- Can simplify network administration
  - Divide network into small chunks
- Reuse IP address space
  - Original motivation behind NAT

IETF-allocated private addresses:

- 10.0.0.0 - 10.255.255.255
- 172.16.0.0 - 172.31.255.255
- 192.168.0.0 - 192.168.255.255

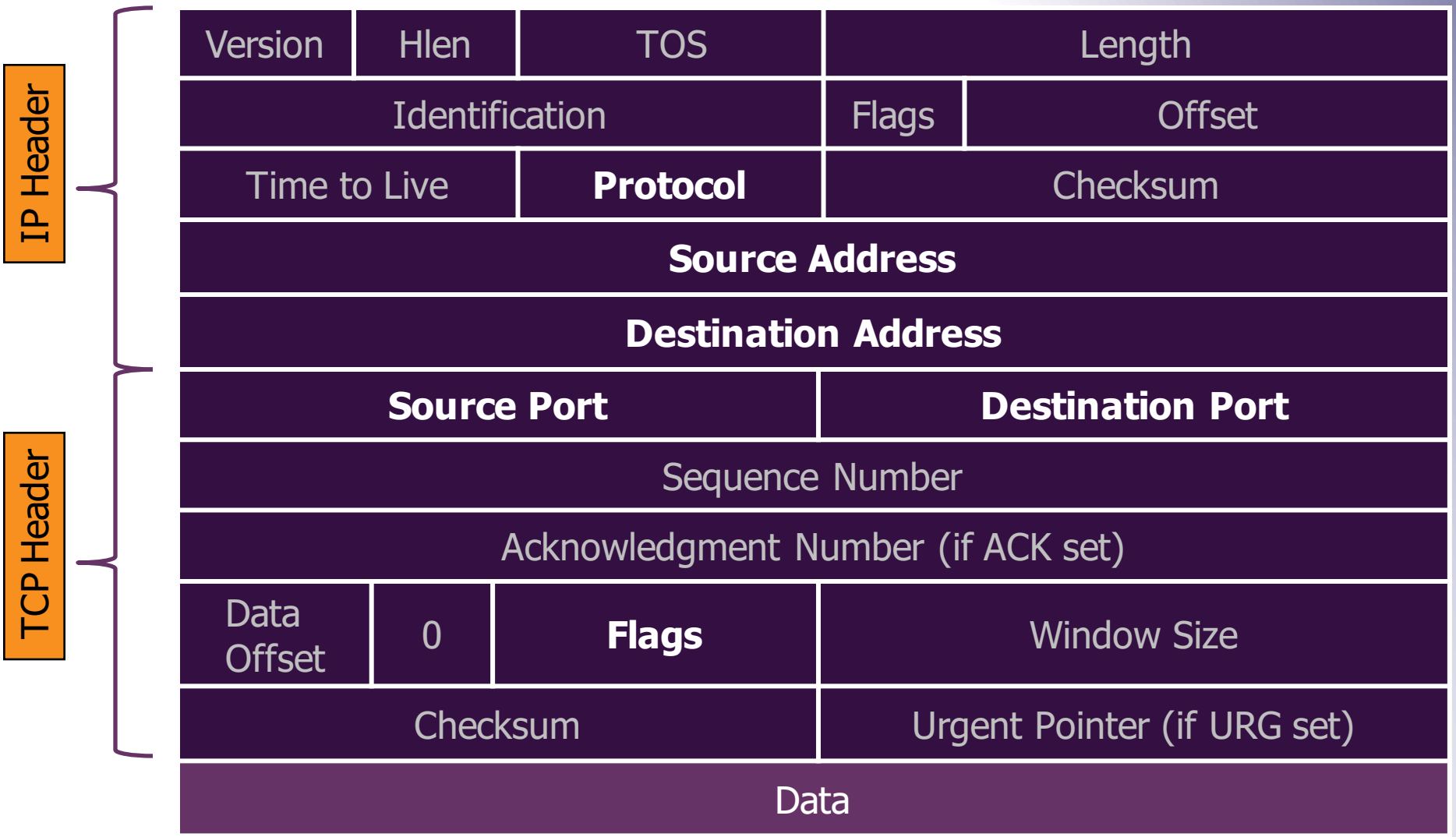
Drawbacks of NAT

- Rewriting IP addresses (and ports) isn't so easy:
  - Must validate/recalculate checksums
  - Certain protocols such as IPSec do not support packet modifications
  - Has to be aware of protocols that exchange IP addresses (e.g., FTP)
    - Must also look for IP addresses beyond packet headers and rewrite them
- Hinders throughput
- Breaks end-to-end principle
  - Prevents host-to-host connection establishment for hosts behind NAT
- Slows the adoption of IPv6?
- Limited filtering of packets

Firewall: packet filtering

Packet Filters

- Most basic kind of firewall is a *packet filter*
  - Router with list of *access control rules*
  - Router checks each received packet against security rules to decide to forward or drop it
  - Each rule specifies which packets it applies to based on a packet's header fields (stateless)
    - Specify source and destination IP addresses, port numbers, and protocol names, or wild cards


IP Header
TCP Header
Version
Hlen
TOS
Length
Identification
Flags
Offset
Time to Live
Protocol
Checksum
Source Address
Destination Address
Source Port
Destination Port
Sequence Number
Acknowledgment Number (if ACK set)
Data Offset
0
Flags
Window Size
Checksum
Urgent Pointer (if URG set)
Data

Packet Filters

- Most basic kind of firewall is a *packet filter*
  - Router with list of *access control rules*
  - Router checks each received packet against security rules to decide to forward or drop it
  - Each rule specifies which packets it applies to based on a packet's header fields (stateless)
    - Specify source and destination IP addresses, port numbers, and protocol names, or wild cards
    - Each rule specifies the *action* for matching packets: **ALLOW** or **DROP** (aka DENY)

    <ACTION> <PROTO> <SRC:PORT> -> <DST:PORT>
  - First listed rule has *precedence*

Examples of Packet Filter Rules

```
allow tcp 4.5.5.4:1025 -> 3.1.1.2:80
```

- States that the firewall should **permit** any TCP packet that is:
  - from Internet address `4.5.5.4` **and**
  - using a source port of `1025` **and**
  - destined to port `80` of Internet address `3.1.1.2`

```
deny tcp 4.5.5.4:* -> 3.1.1.2:80
```

- States that the firewall should **drop** any TCP packet like the above, regardless of source port

Examples of Packet Filter Rules

```
deny tcp 4.5.5.4:* -> 3.1.1.2:80
allow tcp 4.5.5.4:1025 -> 3.1.1.2:80
```

- *In this order*, the rules won't allow *any* TCP packets from `4.5.5.4` to port `80` of `3.1.1.2`

```
allow tcp 4.5.5.4:1025 -> 3.1.1.2:80
deny tcp 4.5.5.4:* -> 3.1.1.2:80
```

- *In this order*, the rules allow TCP packets from `4.5.5.4` to port `80` of `3.1.1.2` ***only*** if they come from source port `1025`

Expressing Policy with Rulesets

- Goal: prevent *external access* to Windows SMB (TCP port 445)
  - Except for one special external host, `8.4.4.1`
- Ruleset:

```
allow tcp 8.4.4.1:* -> *:445
drop  tcp *:* -> *:445
allow  *  *:* -> *:*
```

- Problems?
  - No notion of inbound vs outbound connections
    - Drops outbound SMB connections from inside users
  - (This is a *default-allow* policy!)

Expressing Policy with Rulesets

- Want to allow:
  - Inbound mail connections to our mail server (`1.2.3.4:25`)
  - All outbound connections from our network, `1.2.3.0/24`
  - Nothing else
- Consider this ruleset:

```
allow tcp *:* -> 1.2.3.4:25
allow tcp 1.2.3.0/24:* -> *:*
drop   *  *:* -> *:*
```

- This policy doesn't work …
  - TCP connections are *bidirectional*
  - 3-way handshake: client sends `SYN`, receives `SYN+ACK`, sends `ACK`
    - Followed by either/both sides sending DATA (w/ `ACK` bit set)

Problem: Outbound Connections Fail

```
1. allow tcp *:* -> 1.2.3.4:25
2. allow tcp 1.2.3.0/24:* -> *:*
3. drop   *  *:* -> *:*
```

- Inside host opens TCP connection to port 80 on external machine:
  - Initial SYN packet passed through by rule 2
  - SYN+ACK packet coming back is dropped
    - *Fails* rule 1 (not destined for port 25)
    - *Fails* rule 2 (source not inside host)
    - Matches rule 3 ⇒ **DROP**

Problem: Outbound Connections Fail

1. `allow tcp *:* -> 1.2.3.4:25`
2. `allow tcp 1.2.3.0/24:* -> *:*`
3. `drop  *  *:* -> *:*`

- Fix?
  - In general, we need to distinguish between 2 kinds of inbound packets
    - Allow inbound packets *associated with* an outbound connection
    - Restrict inbound packets *associated with* an inbound connection
  - How do we tell them apart?
    - Approach #1: remember previous outbound connections
      - Requires **state**: stateful firewall
    - Approach #2: leverage details of how TCP works …

Inbound vs. Outbound Connections

- TCP feature: ACK bit set on all packets except first
  - **Plus**: TCP receiver disregards packets with ACK set if they don't belong to an existing connection
- Solution ruleset:
  1. `allow tcp *:* -> 1.2.3.4:25`
  2. `allow tcp 1.2.3.0/24:* -> *:*`
  3. `allow tcp *:* -> 1.2.3.0/24:* only if ACK bit set`
  4. `drop  *  *:* -> *:*`
  - Rules 1 and 2 allow traffic in either direction for inbound connections to port 25 on machine **`1.2.3.4`**
  - Rules 2 and 3 allow outbound connections to any port

How This Ruleset Protects

```
1. allow tcp *:* -> 1.2.3.4:25
2. allow tcp 1.2.3.0/24:* -> *:*
3. allow tcp *:* -> 1.2.3.0/24:* only if ACK bit set
4. drop   *  *:* -> *:*
```

- Suppose external attacker tries to exploit vulnerability in SMB (TCP port 445)
  - Attempts to open an inbound TCP connection to internal SMB server
- Attempt #1: Sends SYN packet to server
  - Packet lacks ACK bit ⇒ no match to Rules 1-3, dropped by Rule 4
- Attempt #2: Sends SYN+ACK packet to server
  - Firewall permits the packet due to Rule 3
  - But then **dropped** by server's TCP stack (since ACK bit set, but isn't part of existing connection)

Stateful Firewall

```
1. allow tcp *:* -> 1.2.3.4:25
2. allow tcp 1.2.3.0/24:* -> *:*
3. drop   *  *:* -> *:*
```

- Stateful FW knows about established connections and can automatically authorize returning traffic
- Safer:
  - No unsolicited packets
  - Simpler to configure, hence less errors

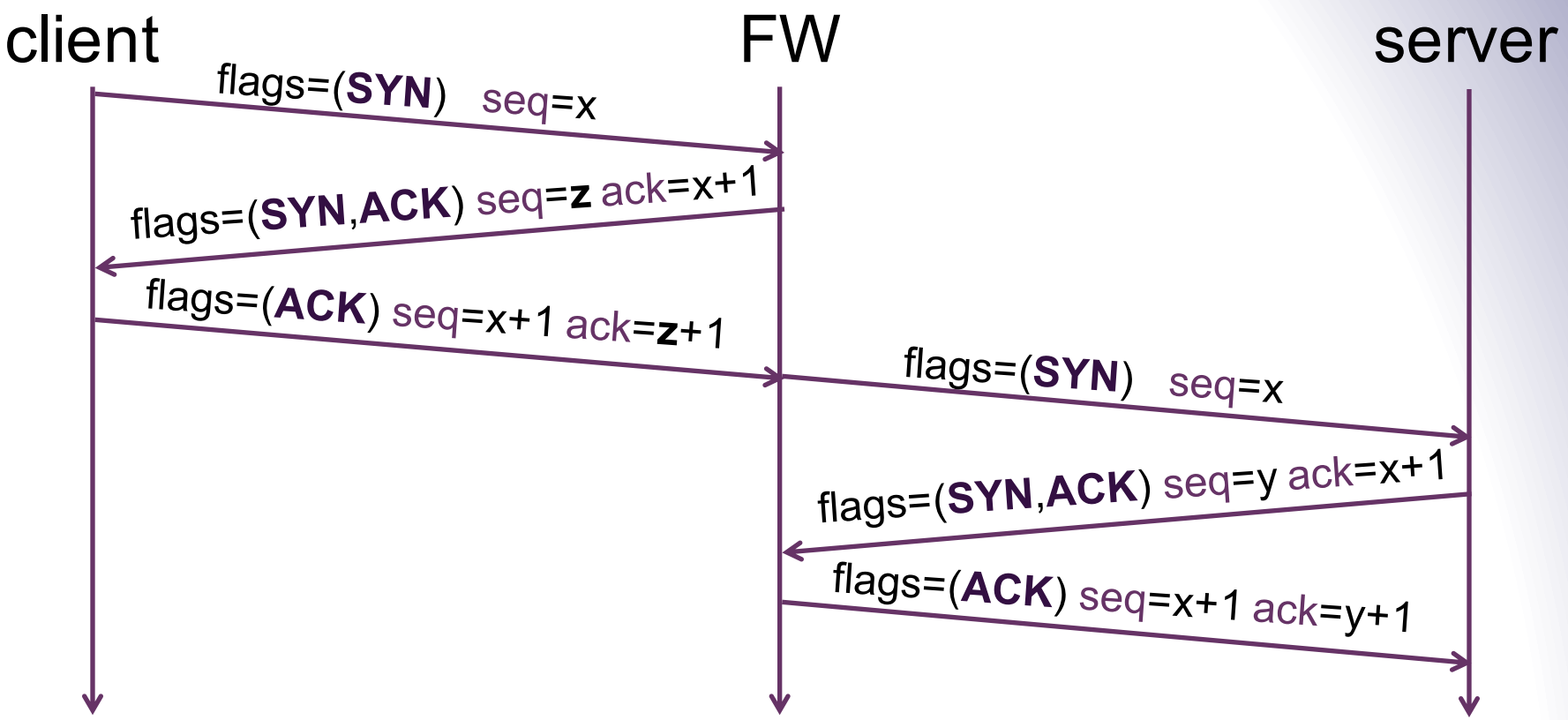
Stateful Firewall: Benefits

- For each connection it knows what the next packet should look like
  - TCP flags, sequence numbers
- It can eliminate packets that do not fit in
- It can replace sequence numbers
  - Example: to randomize initial sequence numbers
- It can prevent SYN flooding

Protection Against SYN Flooding

- Simple:
  - FW keeps track of all attempts to open a connection
  - If it judges that a connection stays half-open for too long, it sends a RST to the server
- Advanced:
  - FW delays SYN packets and generates a SYN + ACK in place of the server
  - Only when it receives an ACK does it send the original SYN to the server

Protection Against SYN Flooding



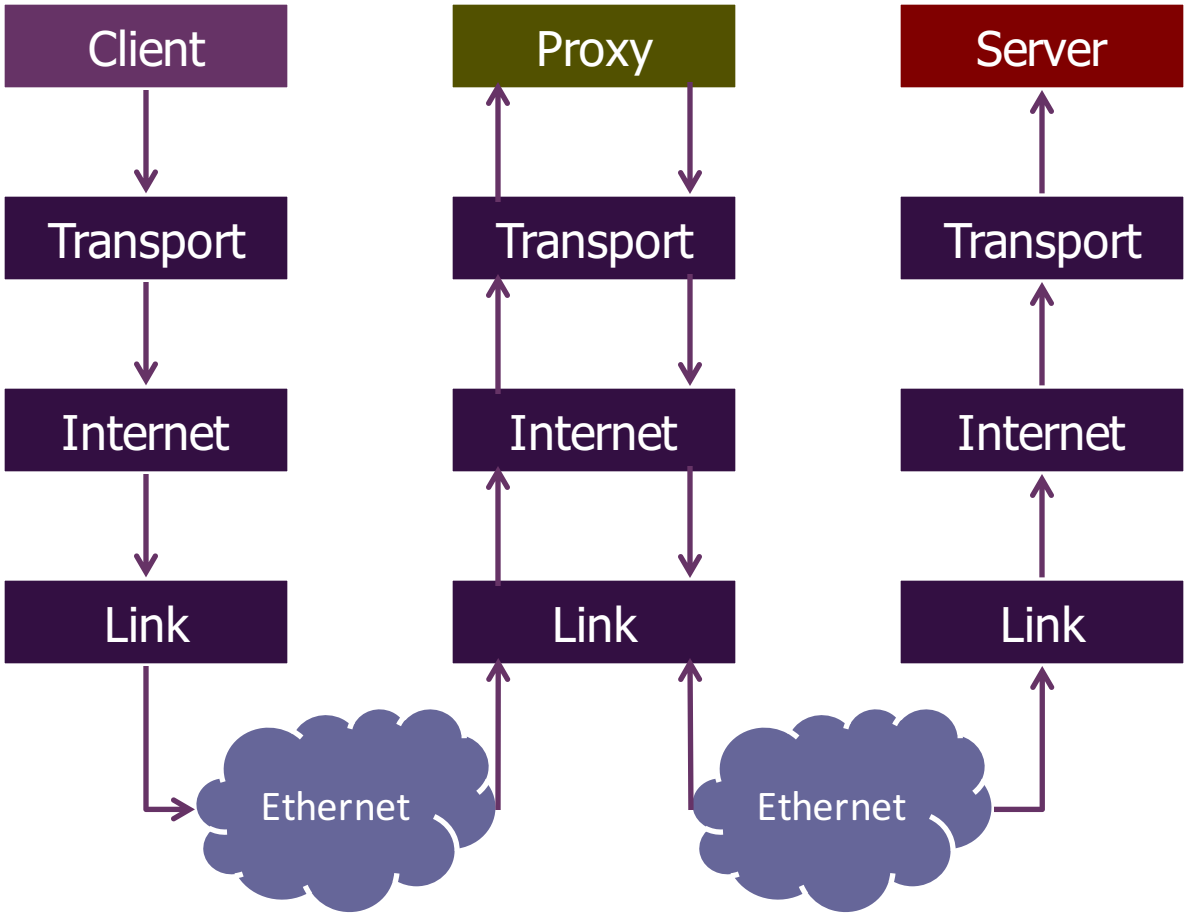
- FW must adapt all sequence numbers

Packet Analysis (Deep Packet Inspection)

- Analyze an application protocol
  - Example: Block Bittorrent
- Analyze packets to verify their format and content
- Eliminate unwanted packets
  - DoS, exploits, viruses
- Eliminate packets that do not correspond to the current protocol's state

Proxies

Proxies are application relays



Proxy

- Proxy acts like both a client and a server
- Can provide other services too
  - Examples: caching, load balancing, mobile page transformation, content transcoding/compression/translation
- A typical example of the defense in depth and choke point principles

Proxy Benefits

- Prevent direct connections from the internal network towards the Internet
  - Choke point
  - Possibly authentication
- Able to filter application-level info
  - URL or DNS blacklists, URL filtering
  - Content type (MIME) filtering, keyword filtering
  - Virus, exploit, …

Cache Feature

- The proxy can keep a copy of all the contents it has served in a cache
- When another client asks for the same content, it can provide the cached copy
  - Ensure content is up-to-date (Example: in HTTP, use header info)
- The transfer is much faster (increase in QoE)
- We can save on bandwidth (limit cost)

HTTP Without Proxy

```
$ telnet www.example.com 80
GET /index.html HTTP/1.0

HTTP/1.1 200 OK
Date: Mon, 10 Feb 2014 15:21:38 GMT
Server: Apache/2.2.3 (CentOS)
Expires: Mon, 10 Feb 2014 15:21:38 GMT
Cache-Control: no-cache
Pragma: no-cache
Connection: close
Content-Type: text/html;charset=UTF-8

<html>
<head><title>Example</title></head>
<body>…
```

HTTP With Proxy

```
$ telnet www.example.com 80
GET http://www.example.com/index.html HTTP/1.0

HTTP/1.1 200 OK
Date: Mon, 10 Feb 2014 15:21:38 GMT
Server: Apache/2.2.3 (CentOS)
Expires: Mon, 10 Feb 2014 15:21:38 GMT
Cache-Control: no-cache
Pragma: no-cache
Connection: close
Content-Type: text/html;charset=UTF-8

<html>
<head><title>Example</title></head>
<body>…
```

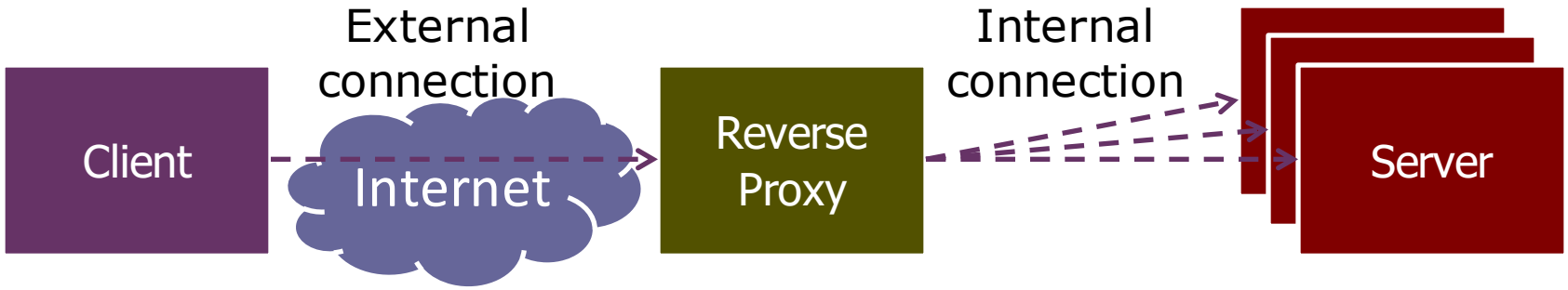
- Requires browser configuration!

Transparent Proxy (intercepting proxy)

- Traffic targeted at a certain port is automatically redirected towards the proxy by the network
- Proxy does not modify the request or response beyond what is required for proxy authentication and identification
- Pros
  - Avoid having to configure browsers
  - Enforce usage of the proxy
  - Enable load balancing
- Cons
  - Doesn't work for servers that are not on the configured port

Reverse Proxy (or Load Balancer)

- Appears to clients to be an ordinary server
- Requests are forwarded to one or more origin servers which handle the requests
- Client has no knowledge of the origin servers



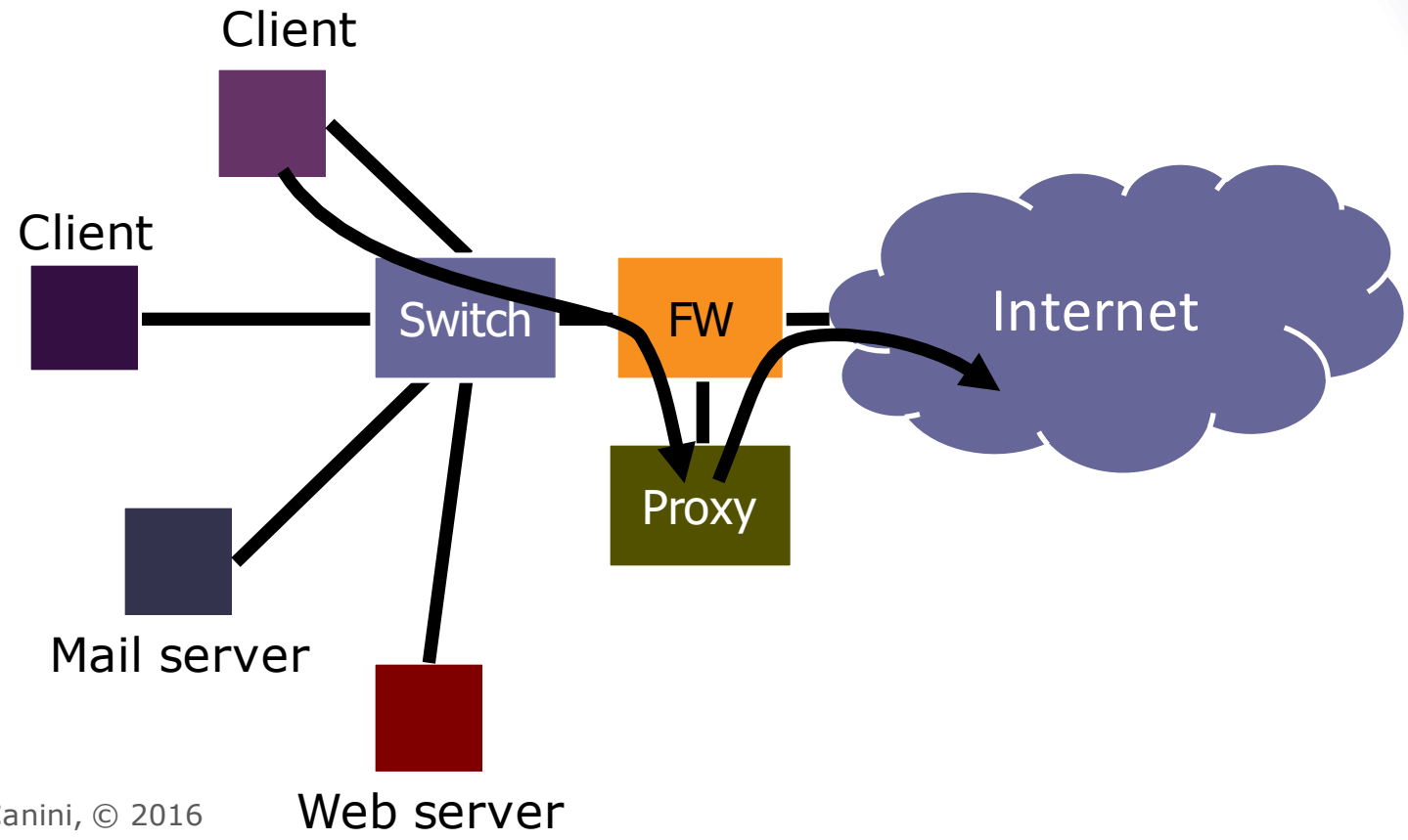
Reverse HTTP Proxy

- Filter requests (blocking exploits)
- Authenticate clients even before they communicate with the server
  - Cannot attack the server unless authenticated
- Accelerate servers
  - Encryption acceleration
  - Caching static content
  - Load balancing

- Accelerate clients via content compression

Firewall DMZ deployment

Demilitarized Zone (DMZ) simple case



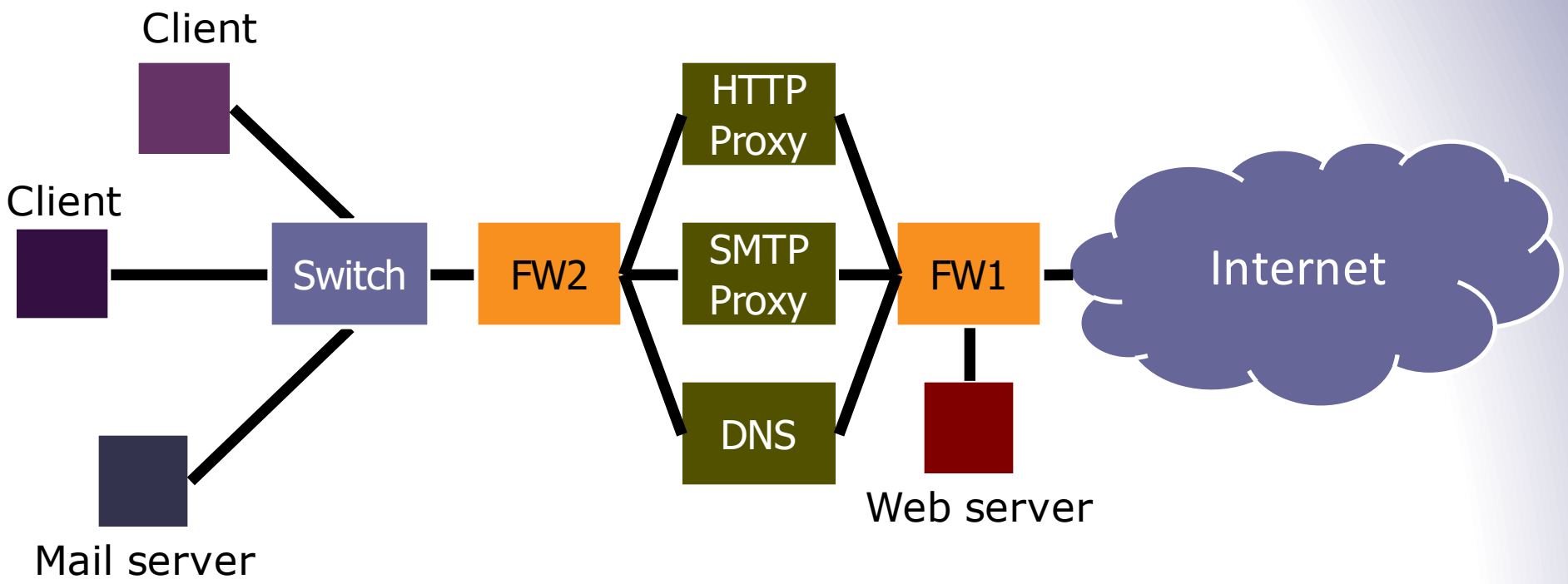
Demilitarized Zone (DMZ) simple case

- The DMZ is connected neither to the Internet, nor to the internal network
- Configuration
  - Internal machines can only connect to the proxy
  - Only the proxy can connect to the Internet
  - Outbound dynamic NAT
  - Inbound static NAT toward the proxy
  - Outbound and Inbound filtering

Demilitarized Zone (DMZ) simple case

- Limitations (**of the example**, not DMZ)
  - The firewall is a critical point
  - All services pass through the same proxy, a vulnerability on a single service can give access to all traffic

Sandwiched DMZ



Sandwiched DMZ

- Configuration
  - Internal machines can only connect to the proxies
    - (one proxy per protocol)
  - Only proxies can connect to the Internet
  - No routing in proxies
  - Outbound dynamic NAT, inbound static NAT
  - Outbound and Inbound filtering

Why Have Firewalls Been Successful?

- *Central control – easy administration and update*
  - Single point of control: update one config to change security policies
  - Potentially allows rapid response
- *Easy to deploy – transparent to end users*
  - Easy incremental/total deployment to protect 1,000's
- *Addresses an important problem*
  - Security vulnerabilities in network services are rampant
  - Easier to use firewall than to directly secure code …

Firewall Disadvantages?

- *Functionality loss – less connectivity, less risk*
  - May reduce network's usefulness
  - Some applications don't work with firewalls
    - Two peer-to-peer users behind different firewalls
- *The malicious insider problem*
  - Assume insiders are trusted
    - Malicious insider (or anyone gaining control of internal machine) can wreak havoc
- Firewalls establish a *security perimeter*
  - Threat from travelers with laptops, cell phones, …

Basic Principles

Principles: The Seven Principles

- Least privileges
- Defense in depth
- Choke point
- Weakest link
- Deny by default
- User participation
- Simplicity

Principle: Least Privilege

- Every part of the system must only have the minimal rights necessary to carry out its job
- Examples:
  - Regular users must not be administrators
  - Administrators must also use regular user accounts
  - A Web server runs under a non-privileged account
    - Unix: nobody
    - Windows: IUSR_machine_name
- Military's slogan: "Need to know"

Principle: Defense in Depth

- Layers of security are harder to break than a single defense
- Examples:
  - Anti-viruses on mail servers and on desktops
  - Patch machines even if they are protected by a firewall
  - Even if FTP connections are blocked by the firewall, workstations should not run FTP servers

Principle: Choke Point

- It is easier to control security if all data has to go through one given point
- Users should not be allowed to bypass the network policy
  - Example: not allow using alternate Internet connection
- Interconnections with other companies must go through the firewall

Principle: Weakest Link

- Attackers go after the part of the system that is the easiest to attack
  - So improving that part will improve security the most
- Example:
  - Useless to install expensive anti-virus software for HTTP traffic if you do not also install one for SMTP traffic
- How do you identify it?
- Weakest link may not be a software problem
  - Social engineering
  - Physical security

Principle: Deny by Default

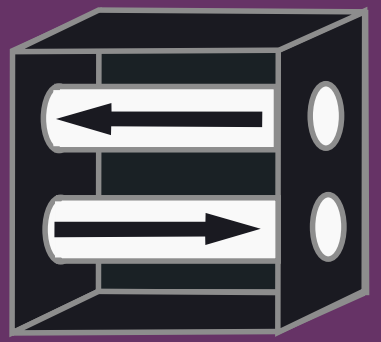
- It is better to prohibit all that is not explicitly authorized than to authorize all that is not explicitly prohibited
- We can never know in advance all the threats to which we will be exposed
- If we make an error, it is better to prohibit something useful than to allow an attack!

Principle: User Participation

- A protection system is efficient only if **all users support it**
- The goal of a firewall is to authorize all that is useful and at the same time avoid dangers
- A system that is too restrictive pushes users to be creative
  - Example: saving confidential email on personal's Gmail to read remotely
- We must understand the user's needs and make sure that reasons for restrictions are well understood by them

Principle: Simplicity

- Most security problems originate from human error
- Complexity leads to bugs and bugs lead to vulnerabilities
- Failsafe defaults
  - The default configuration should be secure (https://bettercrypto.org/)
- In a simple system:
  - The risk of error is smaller
  - It is easier to verify its correct functioning
  - Especially in evolving networks and with several administrators



VPN

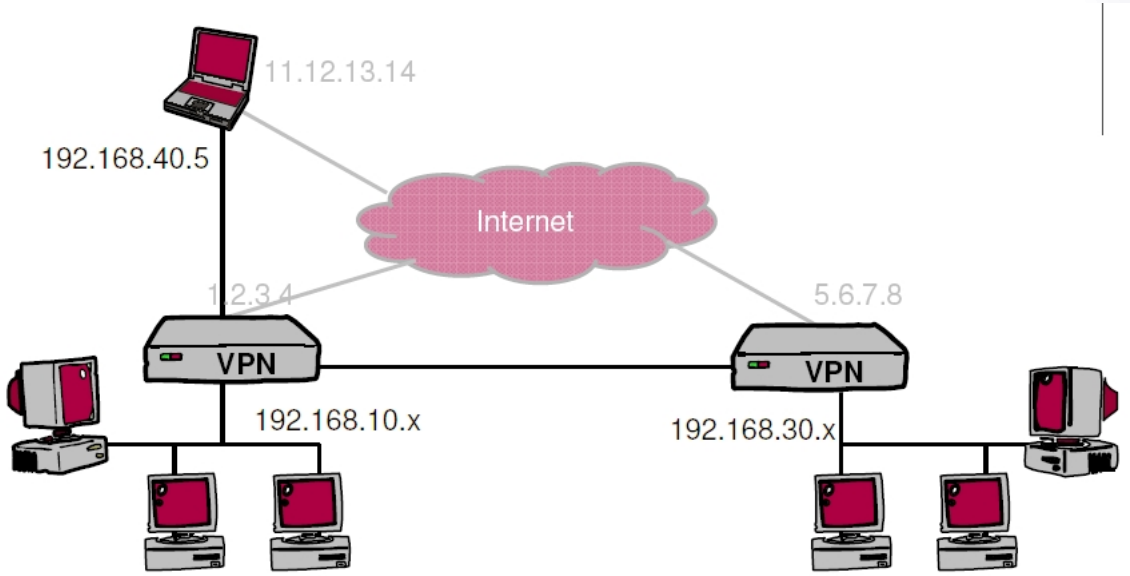
Virtual Private Network

Virtual Private Network (VPN)

Goal: extend a private network across a public network

- Scenarios:
  - Interconnection of remote sites through the Internet
  - Access to a company's network from a laptop connected to Internet

VPN Illustration



VPN basics

- VPN software on routers or PCs (e.g. laptop)
- Packet encapsulation across the Internet
- Encryption of data to guarantee confidentiality

Existing VPN Protocols

- Point to Point Tunneling Protocol (PPTP)
  - Microsoft
- Layer 2 Tunneling Protocol (L2TP)
  - IETF
  - Result of merging Cisco's Layer 2 Forwarding (L2F) protocol and Microsoft's PPTP protocol
- IP Security (IPsec)
  - IETF

IPsec

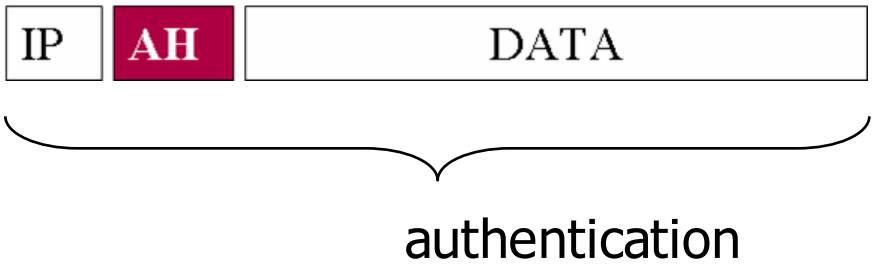
IP Security

IPsec Overview

- Open standard developed by the IETF
  - Public algorithms for confidentiality, authentication, integrity
- Authentication Headers (AH)
  - Provide connectionless **integrity** and origin **authentication** for IP packets
- Encapsulating Security Payloads (ESP)
  - Provide **confidentiality**, data-origin **authentication**, connectionless **integrity**
- Security Associations (SA)
  - Provide algorithms and parameters necessary to AH and/or ESP operations
- Internet Key Exchange (IKE)
  - Key exchange protocol
- Two operation modes: **tunnel**, **transport**

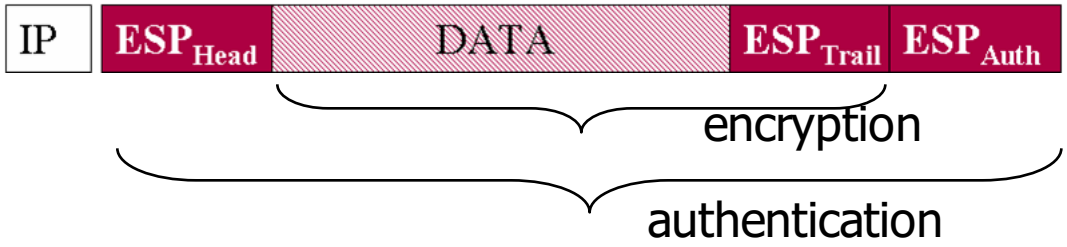
Authentication Header (AH)

- The addition of an authentication header allows verifying the packet's **authenticity** and **integrity**



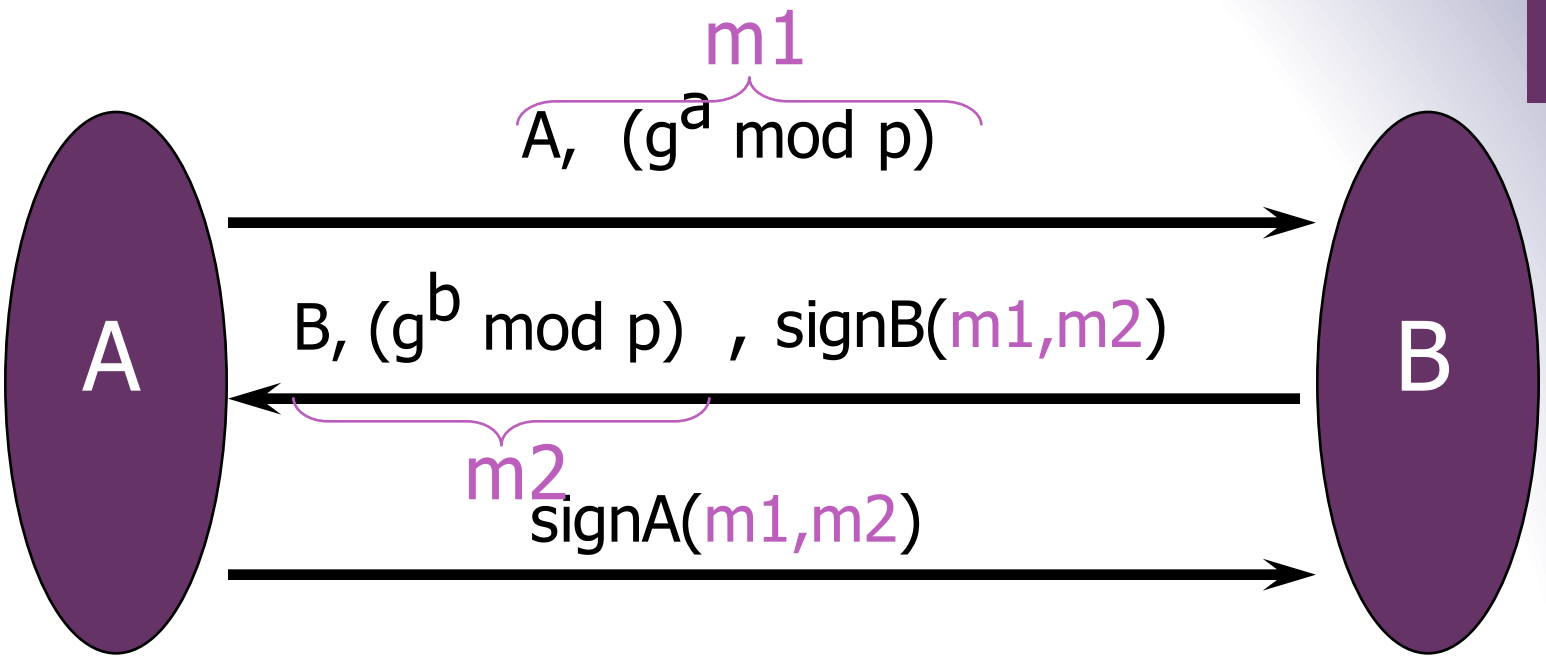
Encapsulated Security Payload (ESP)

- The ESP header allows packet encryption and authentication



- Encryption is done only on the encapsulated data and the trailer
- Encryption is done neither on the header's fields, nor on the authentication data
- Optional authentication is done on the ESP header and all that follow, but not on the IP header

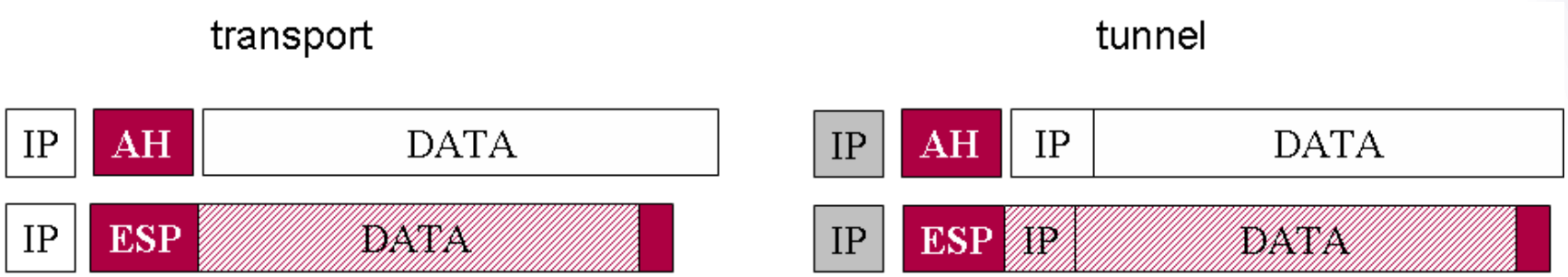
IKE subprotocol from IPsec



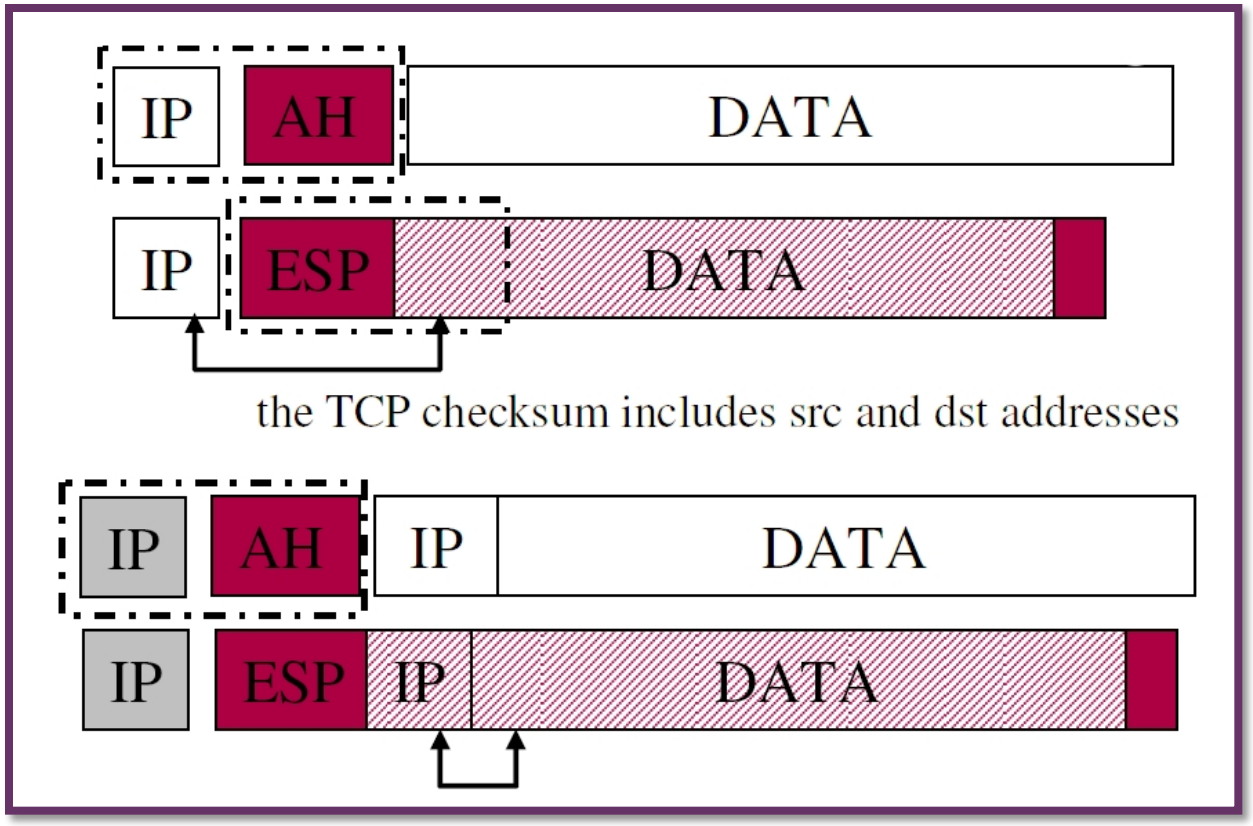
Result: A and B share secret g^{ab} mod p

Transport & Tunnel Modes

- Transport mode:
  - Only IP packet **payload** is encrypted and/or authenticated
- Tunnel mode:
  - The entire packet is encapsulated in a new packet

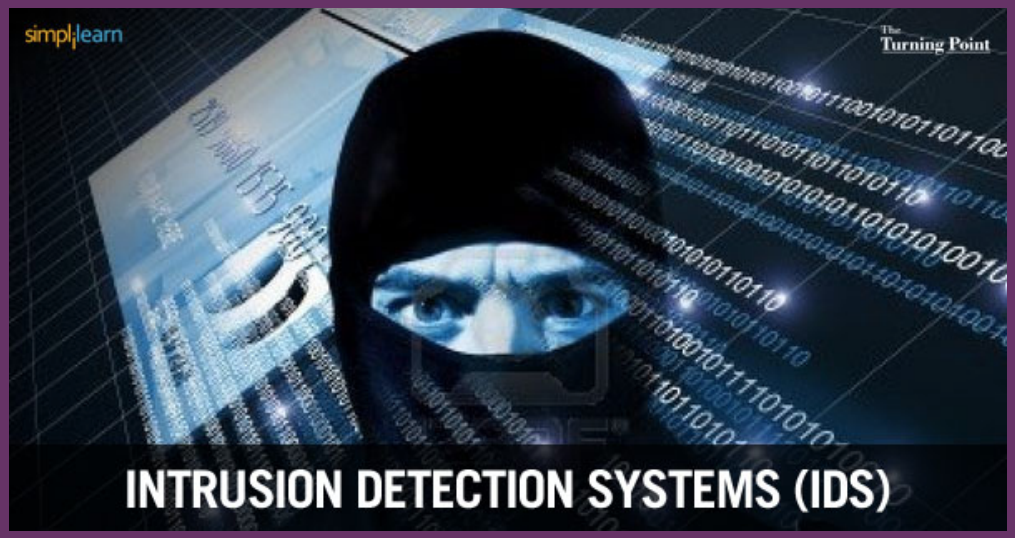


IPsec and NAT



IPsec and NAT

- The TCP and UDP checksum calculation includes a pseudo header made of src and dst IP addresses and ports
- When doing NAT, the checksum has to be readjusted every time the source IP address (and port) changes
- This does not work if the payload is encrypted or authenticated
- **NAT-T** mechanism: encapsulate IPsec in UDP to traverse NAT



Intrusion Detection System

Intrusion Detection System (IDS)

Idea: don't wait for the symptoms of an attack before reacting

- An Intrusion Detection System (IDS) monitors
  - Network traffic (Network IDS, NIDS), typically in front of the firewall
  - Events on servers (Host IDS, HIDS)
- When malicious activities are detected, launch an alarm (SMS, email, etc.)
- Can attempt prevent attacks from succeeding
  - Example: reconfigure firewalls or servers
- Analysis can be done in real-time or by analyzing logs

IDS: Approaches

Network IDS (NIDS)

Off-line Analysis	Analysis of logs and configuration of firewall, routers	Network sniffer
	Examination of system logs	Log/Registry /Sys-call watcher

Real-time Analysis

Host IDS (HIDS)

IDS with Traffic Characterization

- IDS performs statistical analysis on traffic
  - If a value goes beyond its usual limits then assume there is an attack
- Can recognize new attacks
- May also not recognize them… (false negatives)
- Or see attacks where there aren't (false positives)
- High false positives makes this IDS type unpopular
  - Example: Port Scanning (slow mode to avoid detection)

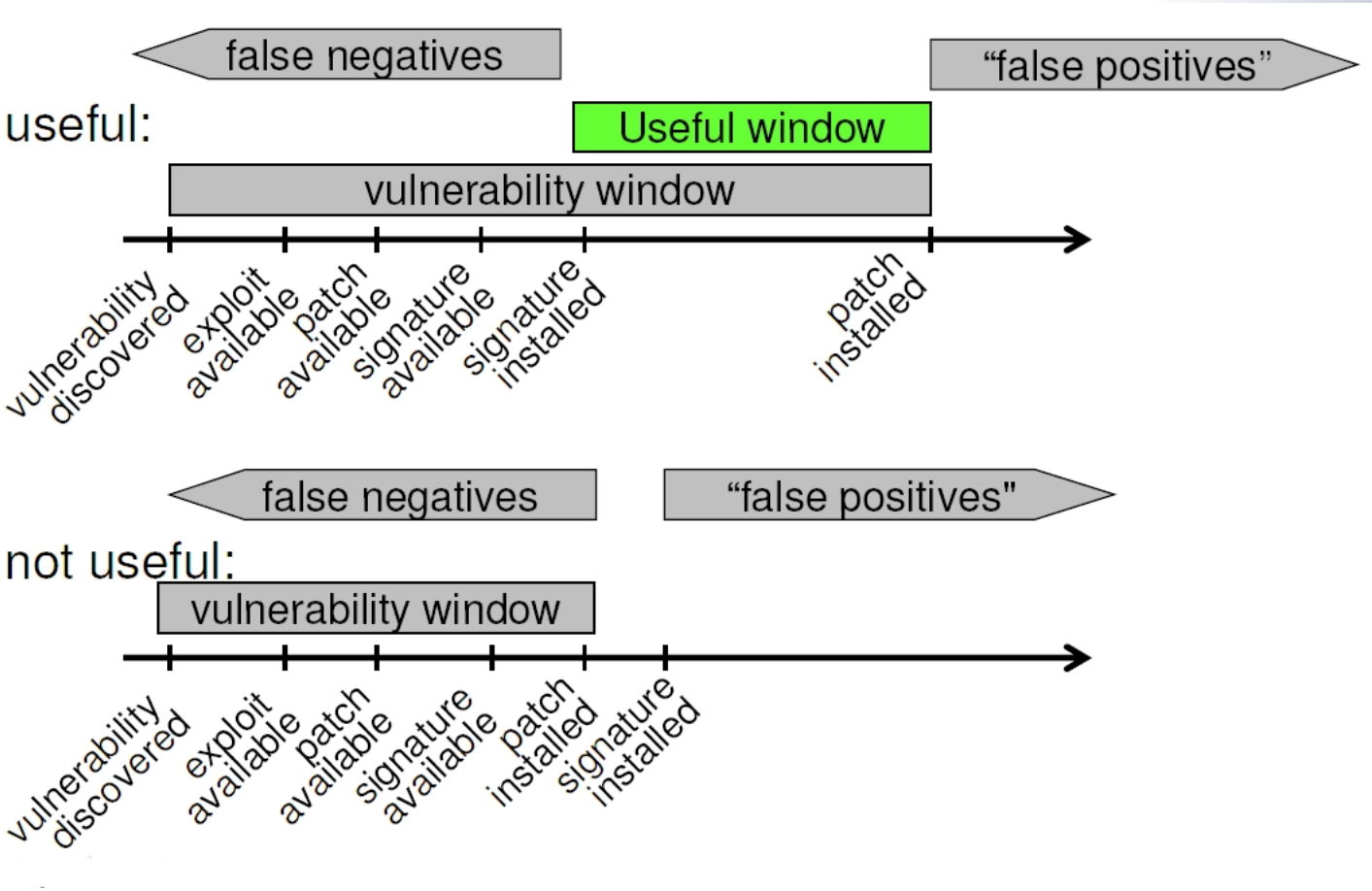
Signature-based IDS

- Use a database of known attacks
  - Example: Web request with URL of 2000 characters == buffer overflow
- Doesn't recognize new attacks
  - Must be constantly updated
  - Honeypots: traps to detect attack
- False negatives
  - Manual attacks can have variations that are not detected
  - Signatures are not always precise
- False positives
  - Doesn't know if an attempted attack was successful
  - Doesn't know if the target is vulnerable (e.g. Linux attack on Windows server)

Snort: Signature-based

- Lightweight IDS for Linux and Windows
- "Signature, protocol and anomaly based inspection methods"
- Analyze traffic, for example in front of the firewall, to detect possible attacks
- Send emails and/or update filtering rules
- Huge signature database updated by users

IDS: Efficiency



Intrusion Prevention Systems: IPS

- An IDS that reacts to an attack
  - IP level: Filters the source IP address in the firewall (for a while)
  - TCP level: Sends a spoofed TCP reset packet to the destination to kill the connection
  - Application level: "Corrects" a Web request by removing suspicious contents
- Beware of denial of service through false positives!

IDS: Discussion

- Traffic-characterization IDSes are not yet very efficient
- IDS with signatures work well but:
  - Majority of the attacks for which we have the signature can be blocked by a FW or proxies
  - We should first prevent before trying to detect
- Not sufficient to install an IDS: must also know how to react to attacks and deal with many false positives
- Automatic reactions are usually not advisable (DoS)

Any questions?

Stay tuned

+

Next time you will learn about

Cybercrime