

ACLs & AAA

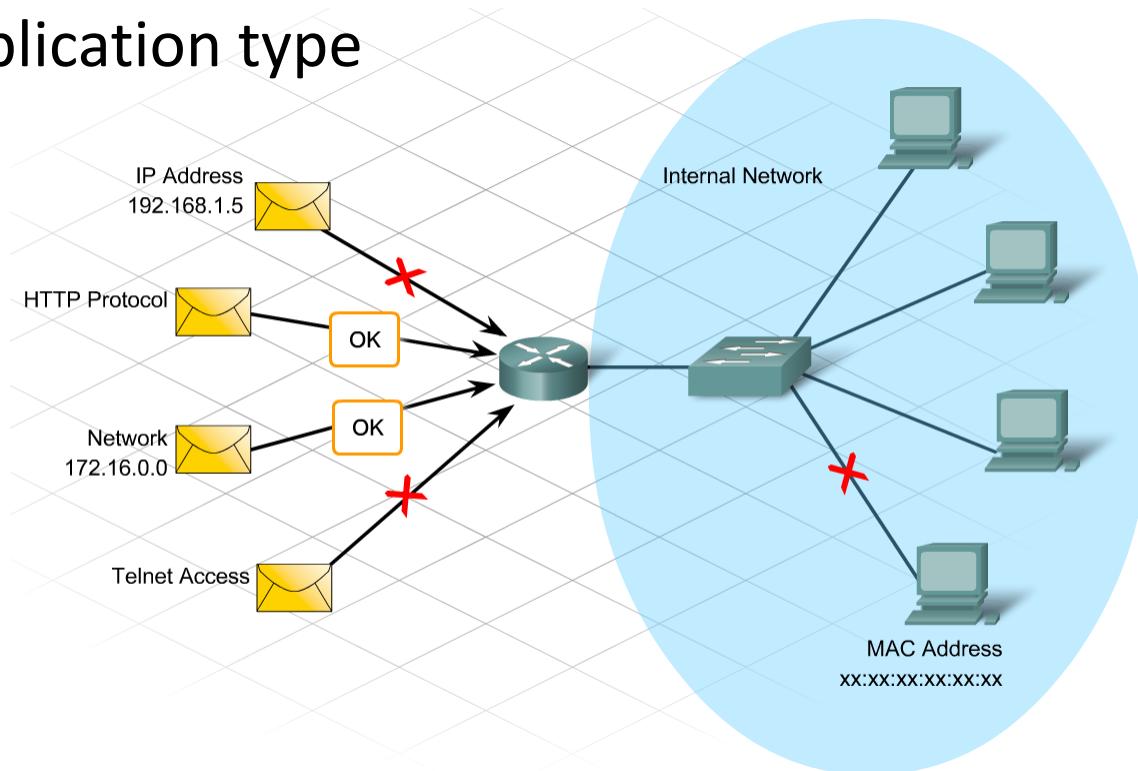
October 28, 2014

What this lecture is about:

- ▶ Traffic filtering with access lists
 - ▶ Understanding access lists
 - ▶ Configuring access lists
- ▶ AAA
 - ▶ A different approach to security
 - ▶ Explaining those three A's

Traffic filtering

- ▶ Analyze the contents of a packet
- ▶ Allow or block the packet
- ▶ Make a decision based on MAC or IP addresses, protocol or application type



Devices providing traffic filtering

- ▶ Firewalls built into dedicated routers
- ▶ Dedicated security appliances
- ▶ Servers



Cisco Security Appliances



Server-Based Firewall



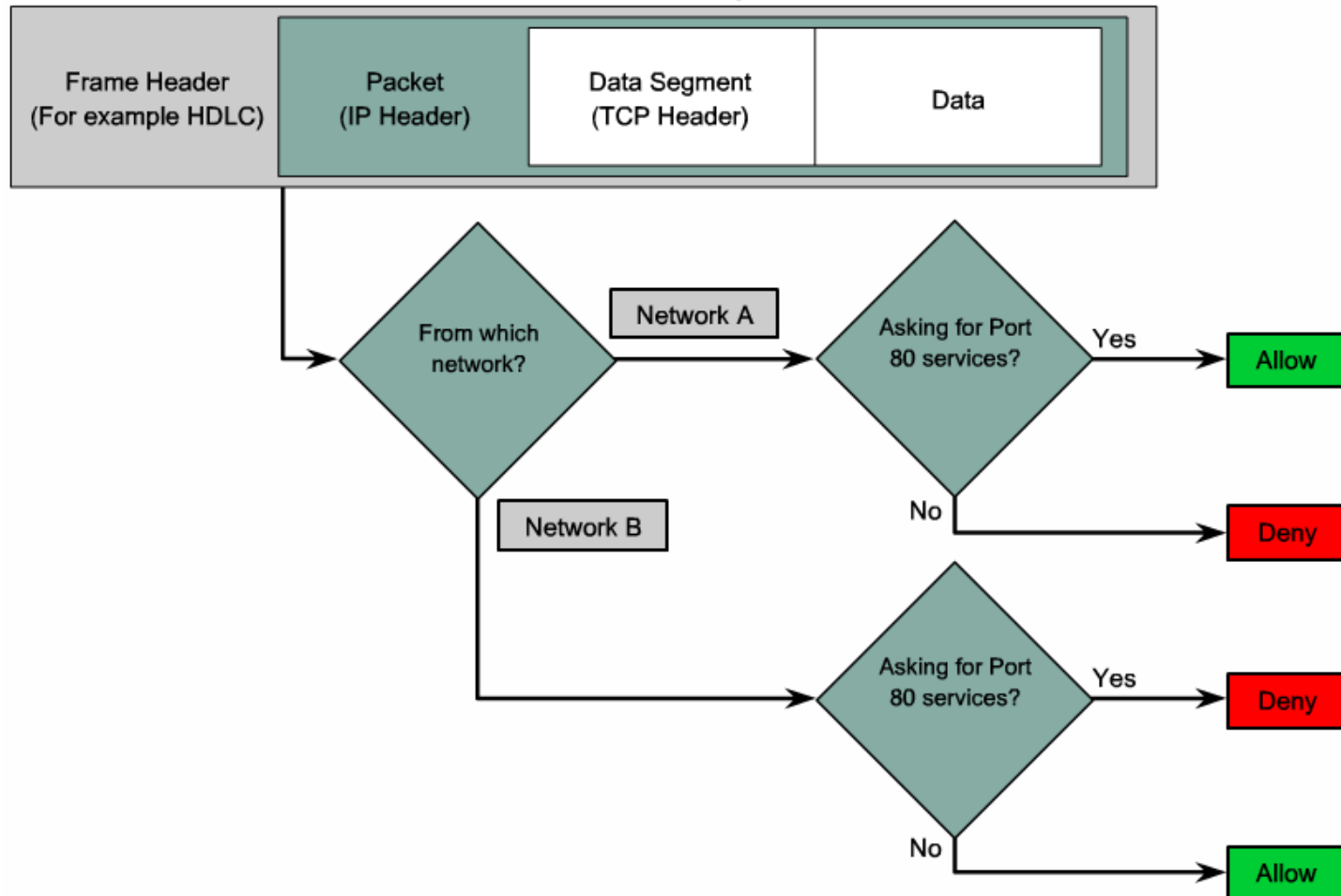
Linksys Wireless Router with Integrated Firewall



Cisco Router with IOS Firewall

Traffic filtering example

- Packets coming from network A or network B



What is an access list?

- ▶ A sequential list of instructions
- ▶ These instructions tell the router which packets to SELECT
- ▶ If you know a little bit about access lists, you might say:
 - ▶ “Why select? Access lists permit or deny traffic!”

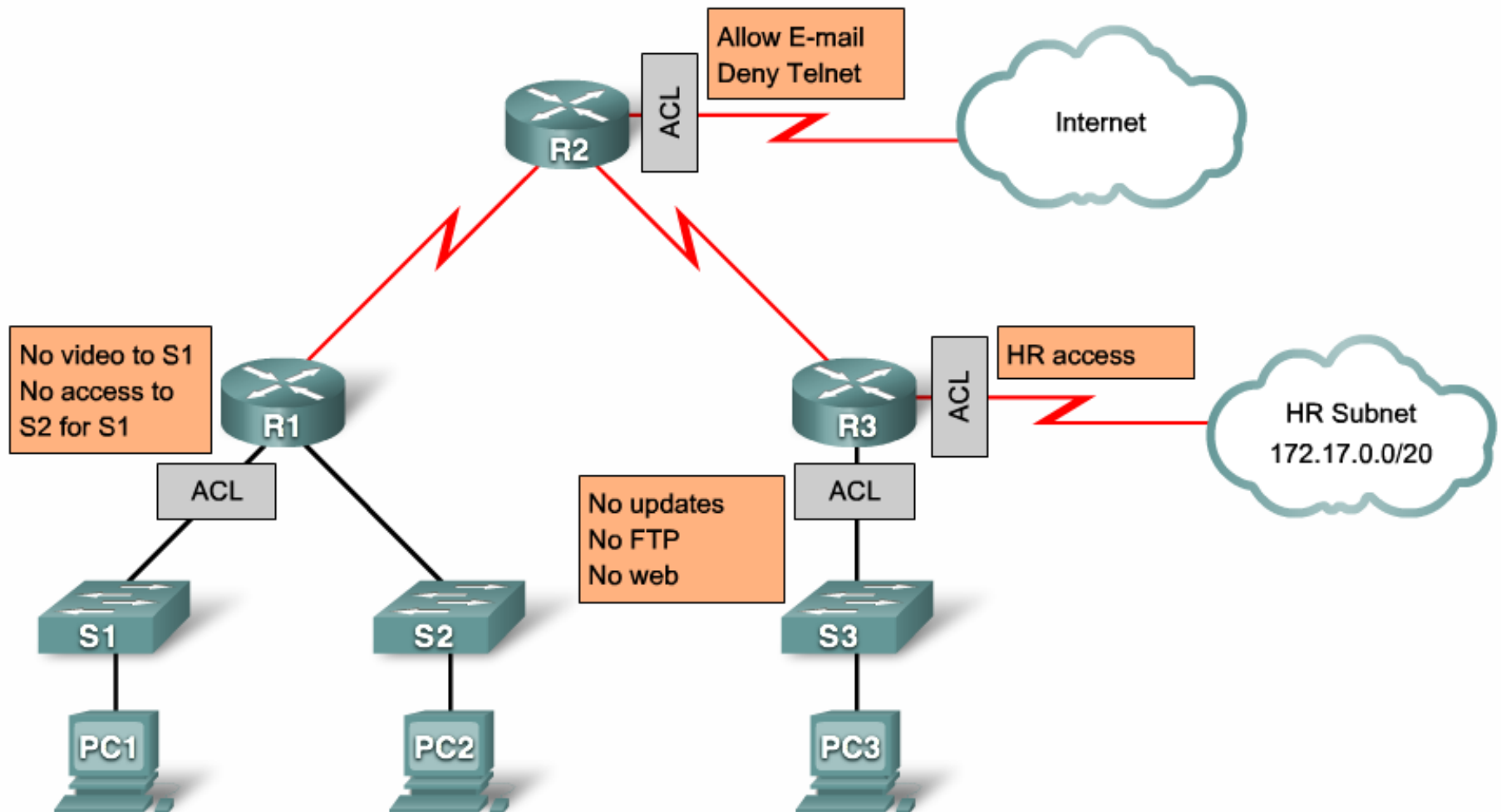
So what does an access list do?

- ▶ An access control list (ACL) selects traffic based on specific parameters
- ▶ The effect of the ACL is dictated by the way it is used
- ▶ For example:
 - ▶ If applied to an interface, an ACL specifies which traffic is permitted or denied
 - ▶ If applied to a vty, an ACL specifies who can Telnet to a router
 - ▶ If used with a firewall, it specifies which traffic will be inspected
 - ▶ If used with a VPN tunnel, it specifies which is the “interesting traffic” that will get encrypted in the tunnel
 - ▶ In QoS implementations, ACLs are used to select priority traffic

The downside?

- ▶ More processing means increased latency over the network
 - ▶ The router has to check every packet against the corresponding ACL on the interface
- ▶ Higher load on the router's processor
 - ▶ Less processing time available for other services and for routing itself

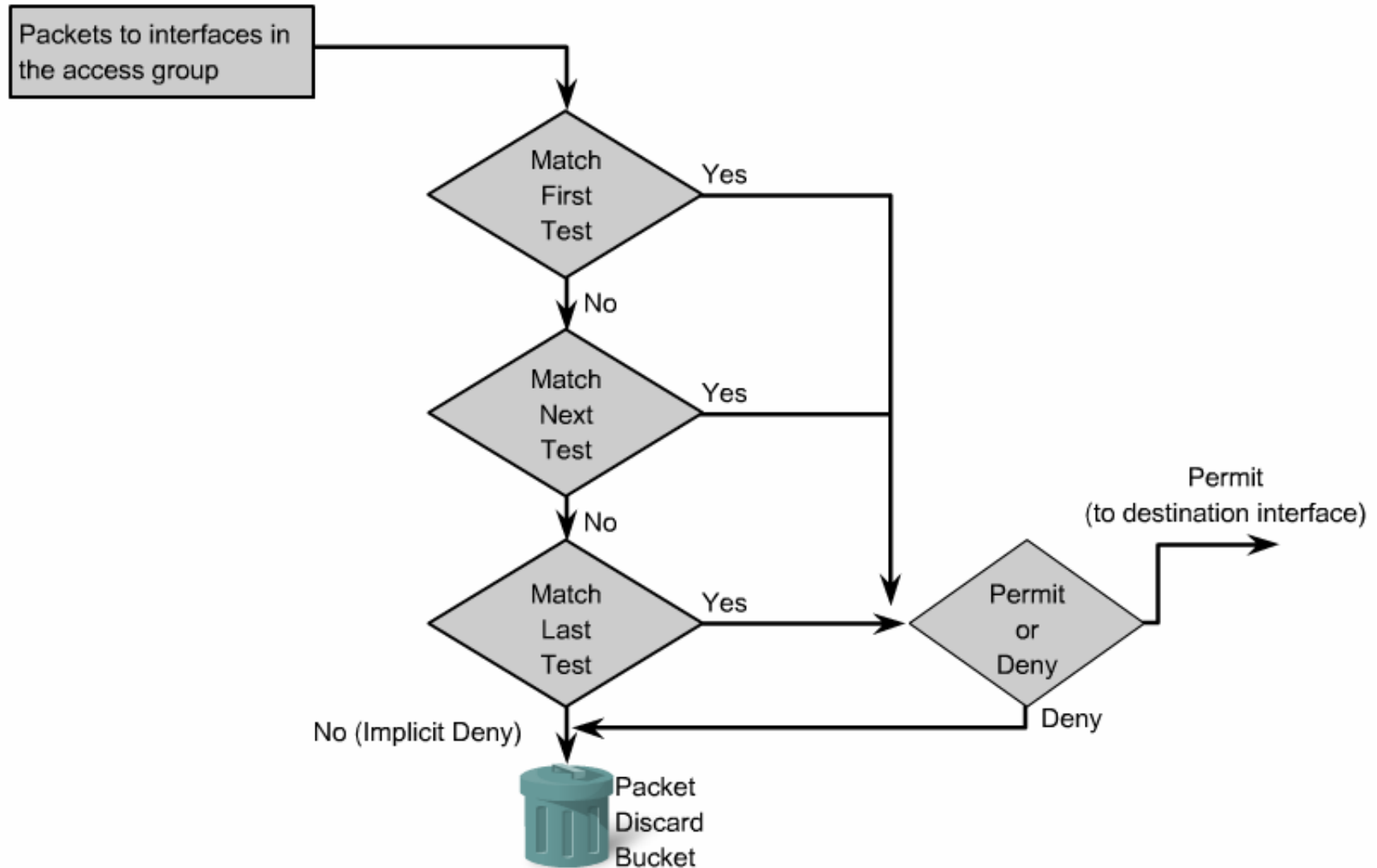
ACL scenario example



General stuff about ACLs

- ▶ All the entries in an ACL share the same syntax
- ▶ The ACL is always read from top to bottom
 - ▶ So the order of its statements is important
- ▶ Packets are checked against all statements until a match is found
- ▶ As soon as a packet matches a statement, all subsequent statements will not be checked anymore
 - ▶ The matched statement says “permit” or “deny”
- ▶ There is an implicit “deny any” statement at the end of each ACL
 - ▶ If a packet does not match any ACL entries, it is automatically dropped

ACL decision process (inbound interface)



Router behaviour with ACLs

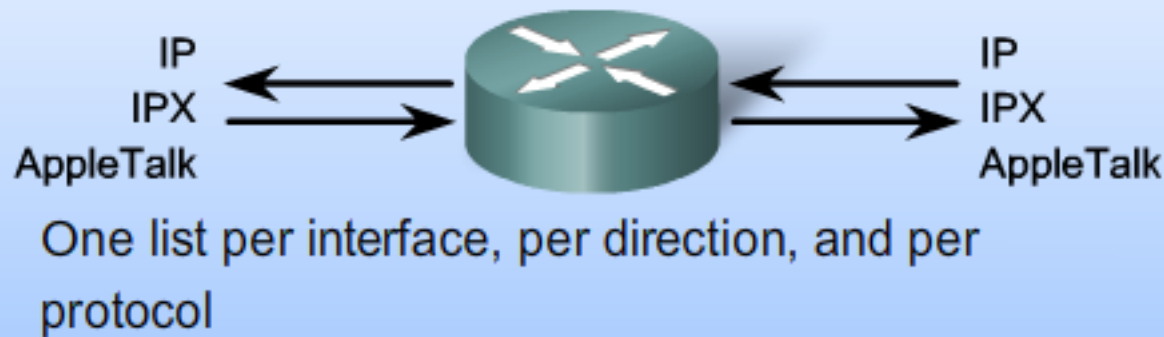
1. Packet arrives on router's interface
2. Check if the layer 2 address matches [otherwise drop]
3. Is there an inbound ACL on the interface? [if not, skip this step]
 1. If yes, is there a match on the ACL?
 2. If permitted, is the packet destined to the router itself?
 3. If not, should the packet be forwarded to another interface?
4. If yes, is there an outbound ACL on the interface? [if not, simply forward]
 1. If yes, is there a match on the ACL?
 2. If it is permitted, send it to the next device

Filtering traffic with ACLs

- ▶ First of all, create the ACL
 - ▶ ACLs are stored in the router's configuration file
- ▶ Apply the ACL to a router interface in order to make it filter packets
 - ▶ Packets that match a “permit” statement will be processed by the router
 - ▶ Packets that match a “deny” statement will be discarded
- ▶ ACLs DO NOT filter traffic originated within the router itself (pings, traceroutes, telnets...)

How many ACLs?

- ▶ You can have ONE ACL for EVERY:
 - ▶ Interface (Serial, FastEthernet etc)
 - ▶ Direction (in/out)
 - ▶ Routed protocol to be filtered (IP, AppleTalk, IPX)



With two interfaces and three protocols running, this router could have a total of 12 separate ACLs applied.

Basic ACL types

- ▶ Numbered ACLs are identified by a number
 - ▶ Standard
 - ▶ Extended

- ▶ Named ACLs (recommended) are identified by a string
 - ▶ Standard
 - ▶ Extended

Standard ACLs

- ▶ Standard ACLs allow you to permit or deny traffic from certain source IP addresses
- ▶ The destination or the port numbers are not involved

- ▶ Example of a standard ACL:

```
R1(config)#access-list 1 permit 192.168.2.0 0.0.0.255
```

```
R1(config)#access-list 1 permit host 192.168.100.1
```

- ▶ This access list will allow traffic for the entire 192.168.2.0/24 network AND from the 192.168.100.1 host
- ▶ All other traffic will be denied. Why?

Analyzing the example

- ▶ Example of a standard ACL:

```
R1(config)#access-list 1 permit 192.168.2.0 0.0.0.255
```

```
R1(config)#access-list 1 permit host 192.168.100.1
```

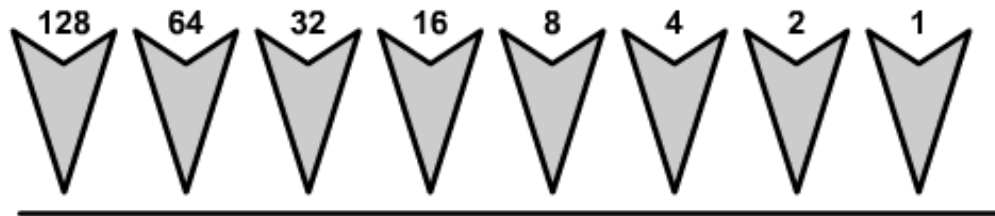
- ▶ Each “access-list” command adds another entry
- ▶ The general syntax for defining a standard ACL is:
 - ▶ *access-list number permit/deny source_address wildcard_mask*
- ▶ Entries with the same number belong to the same ACL
- ▶ Allowed numbers for standard ACLs:
 - ▶ 1-99
 - ▶ 1300-1999
- ▶ Statements are stored in the ACL in the order they are entered

Wildcard masks

- ▶ In ACLs, source addresses are defined using a subnet address and a wildcard mask
- ▶ The wildcard mask tells the router how much of the packet's source IP addresses needs to match
- ▶ Wildcard mask bits mean:
 - ▶ 1: don't care
 - ▶ 0: match
- ▶ For example, to match all packets coming from the network 172.16.12.0/24, you can use the following statement:

```
R1(config)#access-list 5 permit 172.16.12.0 0.0.0.255
```
- ▶ Unlike network masks, wildcard bits can be discontinuous
 - ▶ For example, a wildcard mask of 0.7.128.0 is allowed

Wildcard mask example



Octet Bit Position and Address Value for Bit

Examples

0 0 0 0 0 0 0 0 =

Match All Address Bits (Match All)

0 0 1 1 1 1 1 1 =

Ignore Last 6 Address Bits

0 0 0 0 1 1 1 1 =

Ignore Last 4 Address Bits

1 1 1 1 1 1 0 0 =

Ignore First 6 Address Bits

1 1 1 1 1 1 1 1 =

Do Not Check Address (Ignore Bits in Octet)

Wildcard mask keywords

- ▶ The “host” and “any” keywords facilitate entering the wildcard masks for hosts and networks
- ▶ The “host” option substitutes for the “0.0.0.0” mask. This mask states that all IP address bits must match, so only one host can match
- ▶ The “any” option substitutes for the IP address and the 255.255.255.255 mask. The mask says to ignore the entire IP address, so any source address will match

Wildcard mask keywords examples

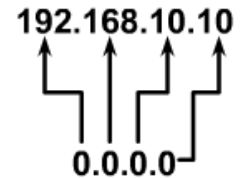
▶ Example 1:

```
access-list 10 permit 192.168.10.10 0.0.0.0
```

▶ Is equivalent to

```
access-list 10 permit host 192.168.10.10
```

Wildcard Mask:



(Match All Bits)

▶ Example 2:

```
access-list 20 permit 0.0.0.0 255.255.255.255
```

▶ Is equivalent to

```
access-list 20 permit any
```

Wildcard Mask:



(Ignores All Bits)

Ranges with wildcard masks

Match subnets 172.30.16.0 to 172.30.31.255

```
access-list 20 permit 172.30.16.0 0.0.15.255
```

Must match

172.30.16.0	10101100	.	00011110	.	00010000	.	00000000
0.0.15.255	00000000	.	00000000	.	00001111	.	11111111

172.30.16.0	10101100	.	00011110	.	00010000	.	00000000
172.30.16.1	10101100	.	00011110	.	00010000	.	00000001

through . . .

172.30.31.254	10101100	.	00011110	.	00011111	.	11111110
172.30.31.255	10101100	.	00011110	.	00011111	.	11111111

Ranges with wildcard masks

Match subnets 172.30.16.0 to 172.30.31.255

```
access-list 20 permit 172.30.16.0 0.0.15.255
```

Any Value

172.30.16.0	10101100	.	00011110	.	00010000	.	00000000
0.0.15.255	00000000	.	00000000	.	00001111	.	11111111

172.30.16.0	10101100	.	00011110	.	00010000	.	00000000
172.30.16.1	10101100	.	00011110	.	00010000	.	00000001

through . . .

172.30.31.254	10101100	.	00011110	.	00011111	.	11111110
172.30.31.255	10101100	.	00011110	.	00011111	.	11111111

Ranges with wildcard masks

Match subnets 172.30.16.0 to 172.30.31.255

```
access-list 20 permit 172.30.16.0 0.0.15.255
```

Must match

Any Value

172.30.16.0	10101100	.	00011110	.	00010000	.	00000000
0.0.15.255	00000000	.	00000000	.	00001111	.	11111111

172.30.16.0	10101100	.	00011110	.	00010000	.	00000000
172.30.16.1	10101100	.	00011110	.	00010000	.	00000001

through . . .

172.30.31.254	10101100	.	00011110	.	00011111	.	11111110
172.30.31.255	10101100	.	00011110	.	00011111	.	11111111

Extended ACLs

- ▶ Can filter packets based on source IP address, but also on:
 - ▶ Destination IP address
 - ▶ Source and destination port numbers
 - ▶ Protocol type
 - ▶ ...and other “special” features, like established TCP connections, time of day, etc.
- ▶ Allowed numbers:
 - ▶ 100-199
 - ▶ 2000-2699
- ▶ They work just like standard ACLs (decision process, sequence, deny any at the end)

Extended ACL syntax

```
access-list access-list-number {deny | permit | remark} protocol source [source-wildcard]  
[operator operand] [port port-number or name] destination [destination-wildcard] [operator  
operand] [port port-number or name] [established]
```

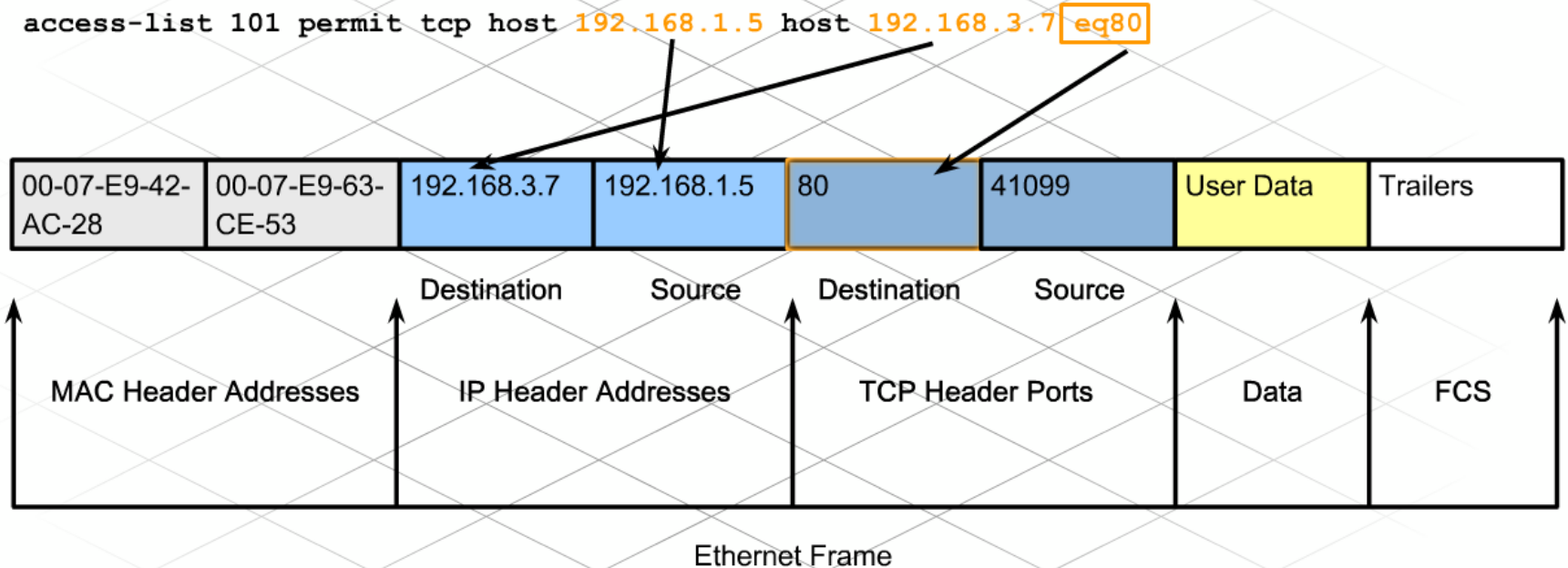
- ▶ **Access-list-number:** Identifies the ACL
- ▶ **Protocol:** Name or number of an Internet protocol. Keywords include **icmp**, **ip**, **tcp**, **udp**. To match any Internet protocol (including icmp, tcp and udp) use ip
- ▶ **Source:** Number of the network or host from which the packet is sent
- ▶ **Source-wildcard:** Wildcard bits applied to the source

Extended ACL syntax

```
access-list access-list-number {deny | permit | remark} protocol source [source-wildcard]
[operator operand] [port port-number or name] destination [destination-wildcard] [operator
operand] [port port-number or name] [established]
```

- ▶ **Destination:** Number of the network or host to which the packet is sent
- ▶ **Destination-wildcard:** Wildcard bits applied to the destination
- ▶ **Operator** (optional): Compares source or destination ports. Keywords include **gt** (greater than), **lt** (less than), **eq** (equal to), **neq** (not equal to) and **range**
- ▶ **Port** (optional): The decimal name or number of a TCP or UDP port
- ▶ **Established** (optional): For TCP only, indicates an already established TCP connection

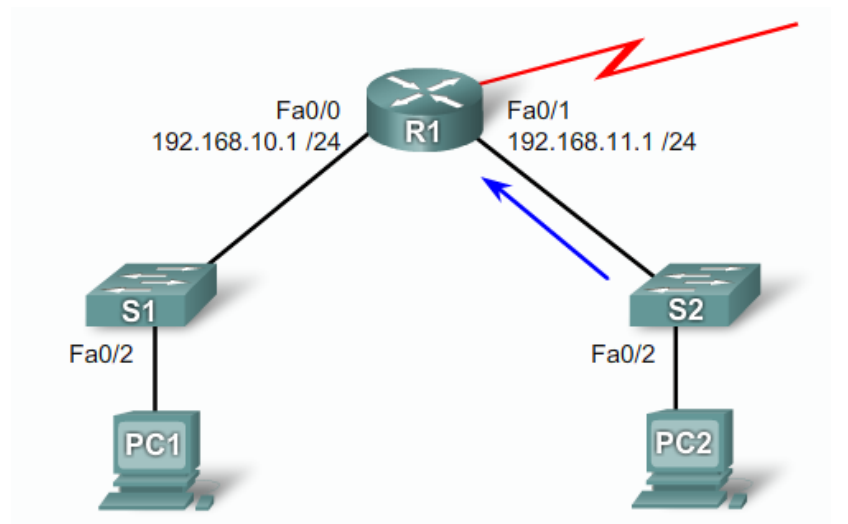
Extended ACLs entries



- ▶ Layer 3 and layer 4 data from each packet is used in order to match an extended access list entry
- ▶ All fields must match for the packet to match an entry

Extended ACLs example

- Purpose: to deny all FTP traffic from PC2's network to PC1's network



```
R1(config)#access-list 101 deny tcp 192.168.11.0 0.0.0.255  
192.168.10.0 0.0.0.255 eq 21
```

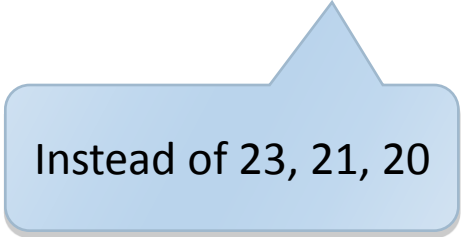
```
R1(config)#access-list 101 deny tcp 192.168.11.0 0.0.0.255  
192.168.10.0 0.0.0.255 eq 20
```

```
R1(config)#access-list 101 permit ip any any
```

Using keywords

- ▶ Extended ACLs support the same keywords as the standard ones
 - ▶ You can use “host” and “any” when appropriate.
- ▶ Extended ACLs can also be created using keywords instead of port numbers
- ▶ Examples:

```
R1(config)#access-list 111 permit tcp 192.168.1.0 0.0.0.255 any eq telnet
R1(config)#access-list 111 permit tcp 192.168.1.0 0.0.0.255 any eq ftp
R1(config)#access-list 111 permit tcp 192.168.1.0 0.0.0.255 any eq ftp-data
```



Instead of 23, 21, 20

Applying ACLs to interfaces

- ▶ An ACL can be applied to an interface on the inbound or the outbound direction
- ▶ ACLs are applied at interface configuration level
- ▶ One command for both standard and extended ACLs:

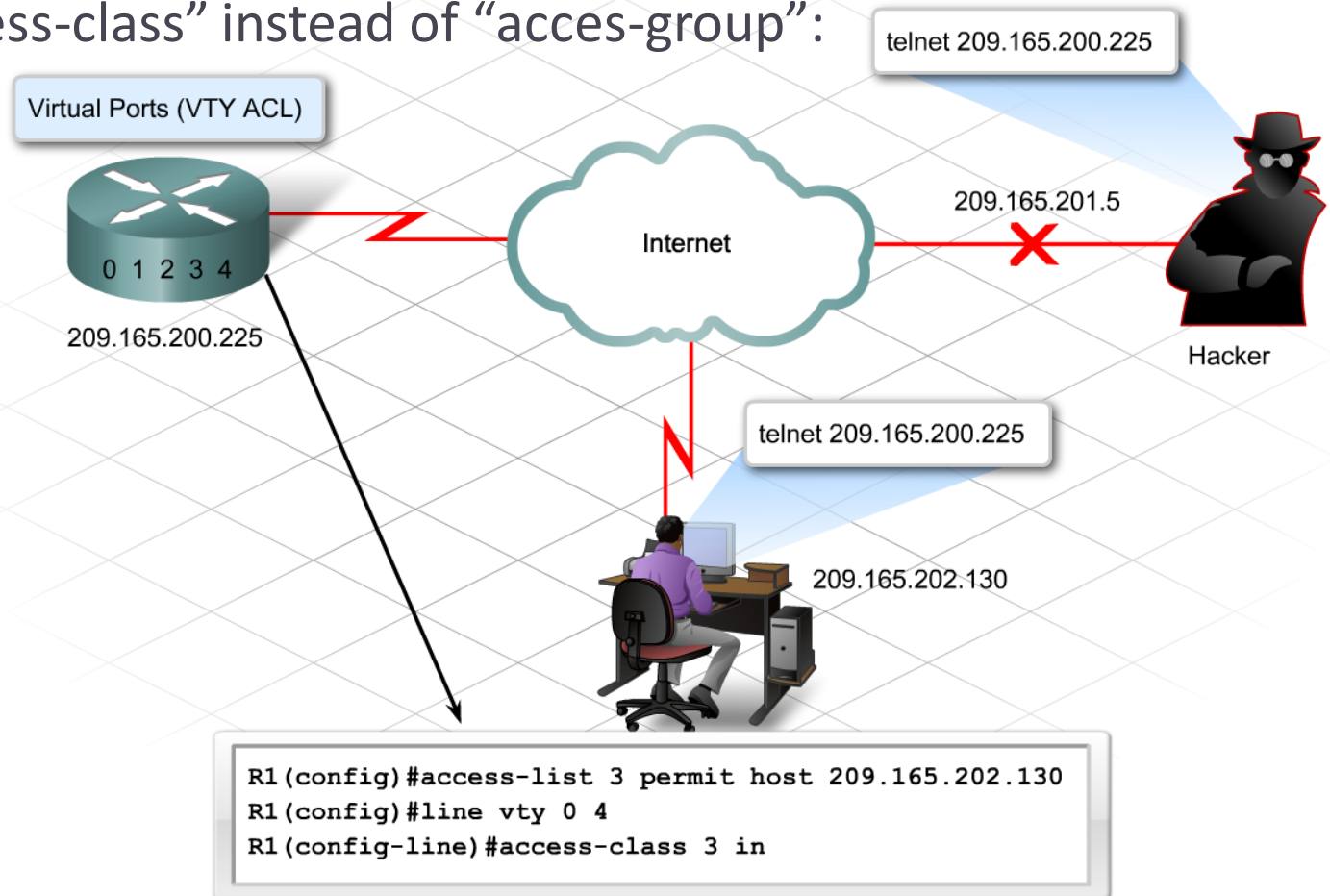
```
R1(config-if)#ip access-group 50 in
```

```
R1(config-if)#ip access-group 101 out
```

- ▶ Standard ACLs should be placed as close as possible to:
 - ▶ Answer: the destination
- ▶ Extended ACLs should be placed as close as possible to:
 - ▶ Answer: the source

Protecting VTY access

- ▶ Access lists can be placed on VTY lines as well
 - ▶ Use “access-class” instead of “access-group”:



Named ACLs

- ▶ A descriptive name replaces the number range used to identify the access list
- ▶ Use the “ip access-list” initial command to enter NACL configuration mode
- ▶ Start succeeding statements with permit or deny, followed by the required conditions
- ▶ Apply in the same way as numbered ACLs.

Named ACL example

- ▶ Named ACLs must specify the “standard” or “extended” keywords in the initial command:

```
R1(config)#ip access-list standard STD_NACL
R1(config-std-nacl)#deny host 10.0.0.1
R1(config-std-nacl)#deny 192.168.2.0 0.0.0.63
R1(config-std-nacl)#permit any
R1(config-std-nacl)#exit
R1(config)#
```

- ▶ Extended NACL:

```
R1(config)#ip access-list extended EXT_NACL
R1(config-ext-nacl)#deny tcp host 172.33.15.1 host
200.17.2.99 eq www
R1(config-ext-nacl)#deny icmp 192.168.12.0 0.0.0.255 any
R1(config-ext-nacl)#permit ip any any log
R1(config-ext-nacl)#exit
R1(config)#
```

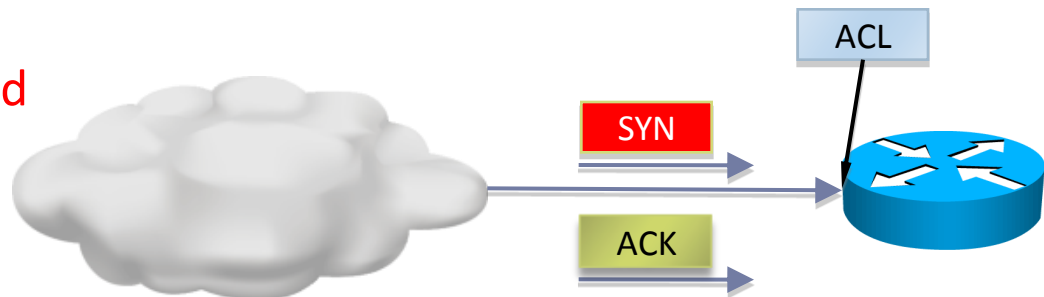
Allowing established connections

- ▶ Extended ACLs allow the use of “established” keyword
- ▶ These entries match packets with the ACK bit set in a TCP stream
- ▶ It doesn't really check if the connection is established or not
 - ▶ An attacker can easily spoof the ACK bit in packets to penetrate this access list

```
R1(config)# access-list 110 permit tcp any 192.168.0.0 0.0.0.255  
established
```

- ▶ Name an attack that can be prevented with such an access list

▶ **Answer: TCP SYN flood**



More IP and TCP options

- ▶ Several other options for filtering TCP packets are available with extended access lists:
 - ▶ ACK bit (similar to “established”)
 - ▶ SYN, FIN bits
 - ▶ RST (reset)

```
R1(config)# access-list 110 permit tcp any 192.168.0.0  
0.0.0.255 ack
```

```
R1(config)#access-list 110 permit tcp any any fin
```

- ▶ “fragments” match on non-initial fragments of same packet:

```
R1(config)#access-list 110 permit ip any any fragments
```

QoS filtering

- ▶ QoS = Quality of Service
- ▶ Extended ACLs can select traffic based on DiffServ values
 - ▶ DiffServ is a newer model for classifying IP packets into different priority levels, using the value in the ToS IP header field.
 - ▶ “newer” than the original RFC

QoS filtering

▶ Original ToS bit fields:

P2	P1	P0	T2 (D)	T1 (T)	T0 (R)	CU1	CU0
----	----	----	--------	--------	--------	-----	-----

- ▶ P2,P1,P0: IP Precedence
- ▶ T2,T1,T0: Delay(min), Throughput(max), Reliability(max)
- ▶ CU: “Currently Unused”

▶ New DiffServ bit fields:

DS5	DS4	DS3	DS2	DS1	DS0	ECN	ECN
-----	-----	-----	-----	-----	-----	-----	-----

- ▶ [DS5, DS4, DS3]: Assured Forwarding (AF) class: higher is better
- ▶ [DS2, DS1, DS0]: Drop Probability: lower is better (DS0 = 0)
- ▶ Notation example: AF31 means Assured Forwarding class 3, with Drop Probability 1.

QoS filtering

- ▶ Using DiffServ values in extended access lists:

```
R1(config)#access-list 110 permit ip any any dscp af31
```

is equivalent to:

```
R1(config)#access-list 110 permit ip any any dscp 26
```

- ▶ Because AF31 is [011010] = 26
- ▶ Using IP Precedence values:

```
R1(config)#access-list 110 permit ip any any precedence ?
```

<0-7>	Precedence value
critical	Match packets with critical precedence (5)
flash	Match packets with flash precedence (3)
flash-override	Match packets with flash override precedence (4)
immediate	Match packets with immediate precedence (2)
internet	Match packets with internetwork control precedence (6)
network	Match packets with network control precedence (7)
priority	Match packets with priority precedence (1)
routine	Match packets with routine precedence (0)

Verifying ACLs

Entries have index numbers.

The “show access-lists” command:

```
R1#show access-lists
```

```
Standard IP access list 50
```

```
10 deny 192.168.13.10
```

```
20 permit 192.168.13.20 (85 matches)
```

```
Standard IP access list STD_NACL
```

```
10 deny 10.0.0.1
```

```
20 deny 192.168.2.0, wildcard bits 0.0.0.63
```

```
30 permit any
```

```
Extended IP access list 101
```

```
10 permit ip any any
```

```
Extended IP access list EXT_NACL
```

```
10 deny tcp host 172.33.15.1 host 200.17.2.99 eq www
```

```
20 deny icmp 192.168.12.0 0.0.0.255 any
```

```
30 permit ip any any log
```

The number of matched packets.
The ACL must be applied on an interface.

Editing ACLs

- ▶ Numbered ACLs (both standard and extended) cannot be edited
 - ▶ All subsequent statements are added to the bottom of the ACL
 - ▶ To modify an ACL, you need to paste it in a text file, delete it from the router, modify it in the text editor and paste it back in the configuration
- ▶ Named ACLs CAN be edited.
 - ▶ ACL entries are numbered with an increment of 10
 - ▶ Precede your statement with the desired index number:

```
R1(config)#ip access-list standard STD_NACL  
R1(config-std-nacl)#15 deny 17.0.0.1
```



Sequence number

Resequencing ACLs

- ▶ The previous example looks like this:

```
R1#show access-lists STD_NACL
Standard IP access list STD_NACL
    10 deny    10.0.0.1
    15 deny    17.0.0.1
    20 deny    192.168.2.0, wildcard bits 0.0.0.63
    30 permit any
```

- ▶ Statements are still consecutive, don't worry 😊
- ▶ You can resequence this access list:

```
R1(config)#ip access-list resequence STD_NACL 20 15
R1#show access-lists STD_NACL
Standard IP access list STD_NACL
    20 deny    10.0.0.1
    35 deny    17.0.0.1
    50 deny    192.168.2.0, wildcard bits 0.0.0.63
    65 permit any
```



Increment



Start

Commenting ACLs

► Commenting numbered ACLs:

```
R1(config)#access-list 50 remark Permit only Gigi's computer through.  
R1(config)#access-list 50 permit 192.168.13.20  
R1(config)#access-list 50 remark Do not allow the secretary through.  
R1(config)#access-list 50 deny 192.168.13.10
```

► Commenting named ACLs:

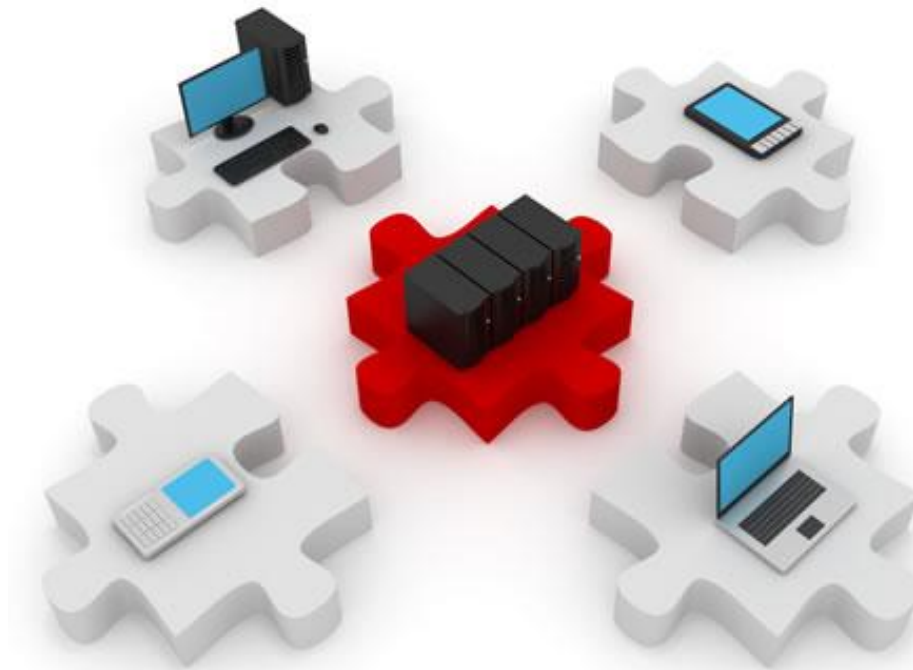
```
R1(config)#ip access-list standard STD_NACL  
R1(config-std-nacl)remark This is a standard access list.  
R1#show run  
ip access-list standard STD_NACL  
deny 10.0.0.1  
deny 17.0.0.1  
deny 192.168.2.0 0.0.0.63  
permit any  
remark This is a standard access list.
```

ACLs best practices

- ▶ Start by creating an ACL in a text editor, not directly on the router
 - ▶ And not while it is in use!
- ▶ Verify your ACL first in a test environment
- ▶ If you're having second thoughts, use the “reload in” command before applying ACLs, in case you lock yourself out of the router:

```
R1#reload in ?
```

```
Delay before reload (mmm or hhh:mm)
```



AAA

Authentication, Authorization, Accounting

Major concepts

- ▶ The purpose of AAA and various implementation techniques
- ▶ Implement AAA using a local database
- ▶ Implement AAA using RADIUS and TACACS+ protocols
- ▶ Authorization
- ▶ Accounting

AAA – credit card similarity

Authentication

Who are you?



Authorization

which resources the user is allowed to access and which operations the user is allowed to perform?

Account number: 1234-567-890 Statement Closing Date: 01-31-01 Current Amount Due: \$278.50

JOE EMPLOYEE
456 SKYVIEW DRIVE
HOMETOWN, USA 99900-1234

EA BANK
132 VINE STREET
ANYTOWN, USA 67500-0010

872919345 00178255000000003

Statement of Personal Credit Card Account

Cardmember Name: JOE EMPLOYEE Account Number: 1234-456-890 Statement Closing Date: 01-31-01

Statement Date: 02-01-01 Payment Due Date: 03-01-01

Closing Date: 01-31-01

Credit Limit: \$1,500.00 Credit Available: \$1221.50

New Balance: \$278.50 Minimum Payment Due: \$20.00

Account Summary

Previous Balance:	+74.24	Transaction Fees:	+3.00
Purchases:	+250.50	Annual Fees:	+25.00
Cash Advances:	+0	Current Amount Due:	+250.50
Payments:	-74.25	Amount Past Due:	+0
Finance Charge:	+0	Amount Over Credit Line:	+0
Late Charge:	+0	NEW BALANCE:	\$278.50

Reference Number	Sold	Posted	Activity Since Last Statement	Amount
43210987	01-03	01-13	Payment, Thank You	-\$74.25
01234567	01-12	01-13	Wings 'N' Things Anytown, USA	\$25.25
78901234	01-14	01-17	Record Release Anytown, USA	\$40.00
45678901	01-14	01-17	Sports Stadium Anytown, USA	\$75.25
3210987	01-22	01-23	Tie Tack Anytown, USA	\$20.75
76543210	01-29	01-30	Electronic World Anytown, USA	\$89.25
2345678		01-30	Transaction Fees	\$3.00
34567890		01-01	Annual Fee	\$25.00

PAGE 1 OF 1

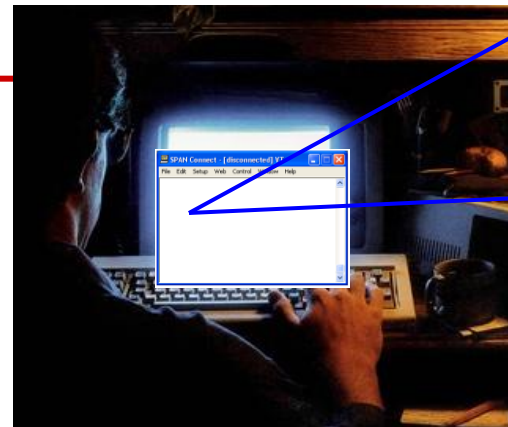
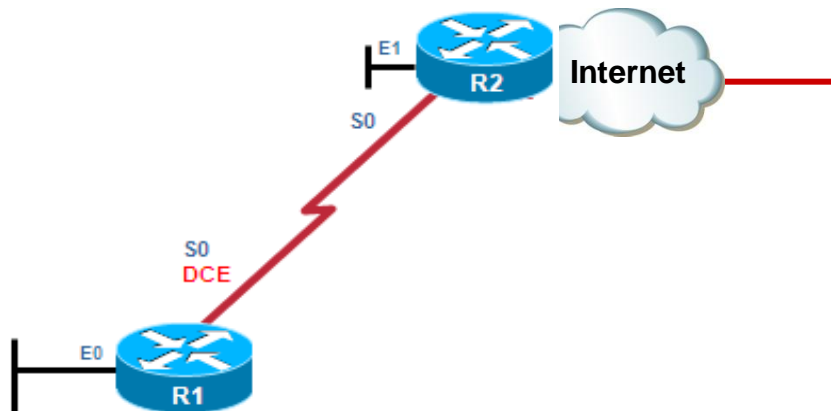
Accounting

What did you spend it on?

Authentication – password only

- ▶ Uses only a password for line access
- ▶ Easiest to implement, but provides little security
- ▶ Vulnerable to brute-force attacks
- ▶ Provides no accountability (who's doing what?...)

```
R1(config)#line vty 0 4  
R1(config-line)#password cisco  
R1(config-line)#login
```



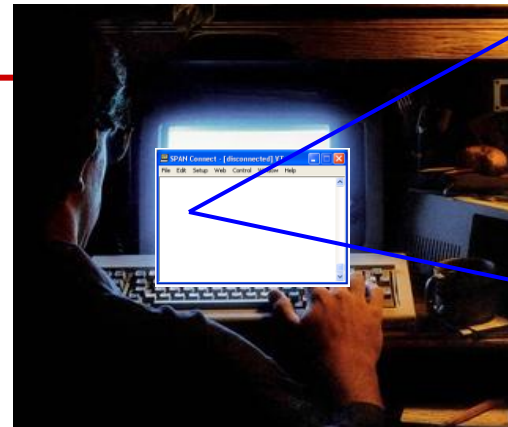
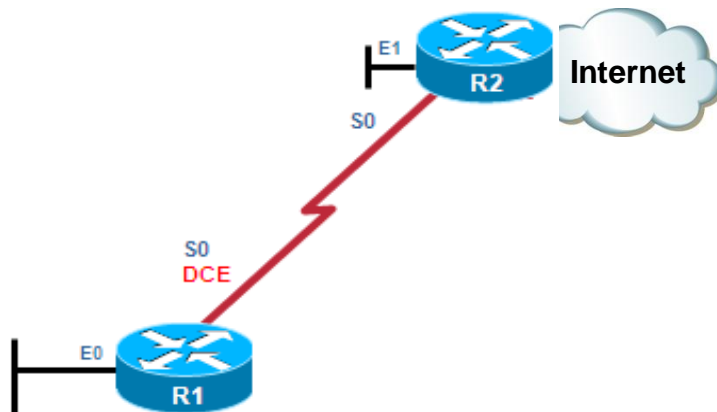
User Access Verification

Password: cisco
Password: cisco1
Password: cisco12
% Bad passwords

Authentication – local database

- ▶ The database is made up of user accounts
- ▶ Each user account has a username and a password
- ▶ Accounts are configured and stored locally
- ▶ Provides accountability
- ▶ But does not provide any fallback authentication method

```
R1(config)#username student secret restanta
```



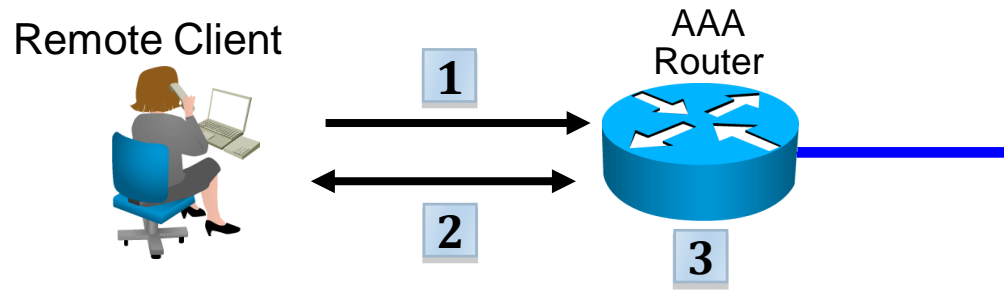
User Access Verification

Username: Admin
Password: cisco1
% Login invalid

Username: Admin
Password: cisco12
% Login invalid

Self-contained AAA authentication

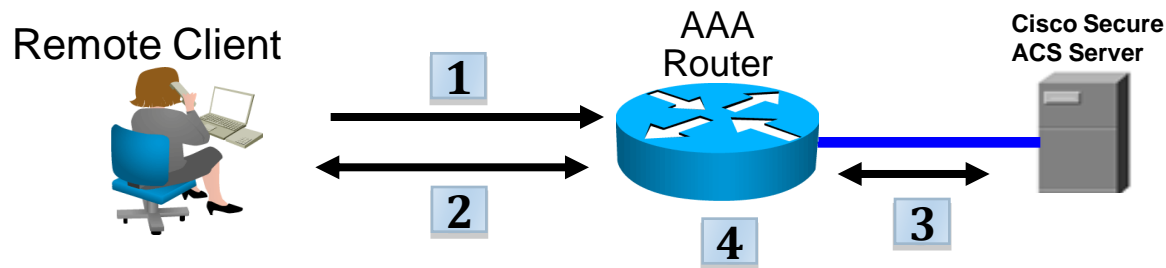
- ▶ Self contained = everything is stored locally, in the router
- ▶ Works well for small networks



- ▶ The user establishes a connection with the router
- ▶ The AAA router prompts the client with a username and a password
- ▶ The router authenticates the credentials using the local database
- ▶ The user receives the corresponding authorization level

Server-based AAA authentication

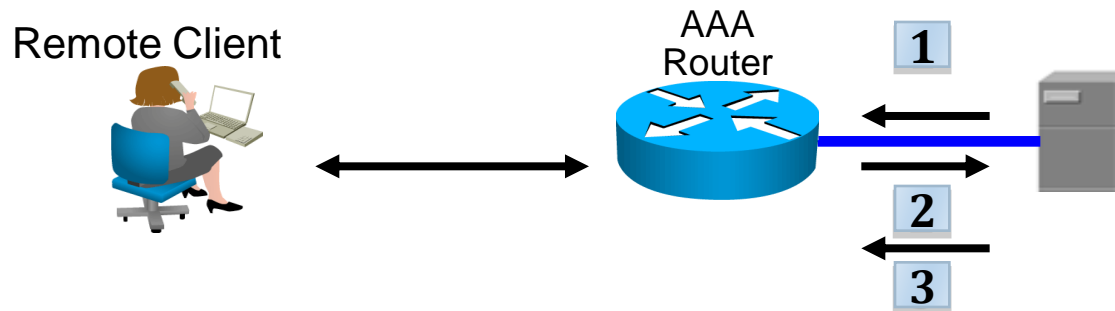
- ▶ Uses an external database, stored on a server
- ▶ Appropriate if there are multiple devices in the network that require user authentication



- ▶ The user establishes a connection to the router
- ▶ The AAA router prompts the client for a username and a password
- ▶ The router authenticates the credentials by contacting a remote AAA server
- ▶ The user receives authorization to use the network based on the information stored on the remote AAA server

AAA authorization

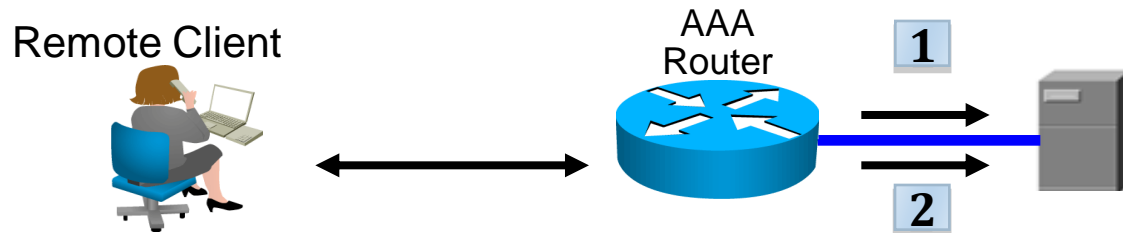
- ▶ Typically implemented using an AAA server-based solution
- ▶ The user receives a set of attributes that describe its level of access on the network



- ▶ The user establishes a session with the AAA server
- ▶ The router requests authorization for the requested user service from the AAA server
- ▶ The AAA server returns a PASS/FAIL response

AAA accounting

- ▶ Usually implemented using a server-based AAA solution
- ▶ Keeps a detailed log of what an authenticated user is doing while connected to a device



- ▶ After authentication, the AAA accounting process generates a start message to begin accounting
- ▶ When the user finishes, a stop message is sent in order to stop accounting

Local AAA authentication commands

- ▶ Creating the user database:

```
R1(config)#username student secret R3st4nt4
```

```
R1(config)#username profesor secret d1n0saur
```

- ▶ Enabling AAA on the router:

```
R1(config)#aaa new-model
```

- ▶ Configure the authentication method list:

```
R1(config)#aaa authentication login default local-case
```

The default
authentication
list

Use local
database (case
sensitive)

AAA authentication command elements

```
R1(config)#aaa authentication login {default|list-name} method1  
[method2] ...
```

- ▶ **default:** define the authentication methods for the default list
- ▶ **list-name:** specify a different authentication method list

method#: Identifies a list of authentication methods that the algorithm will try, in sequence

- ▶ At least one method is required
- ▶ Four methods can be defined, as a maximum
- ▶ Subsequent methods are ONLY used if the first ones fail due to errors or timeouts
 - ▶ If a FAIL answer is received from one method, the others are not checked any more

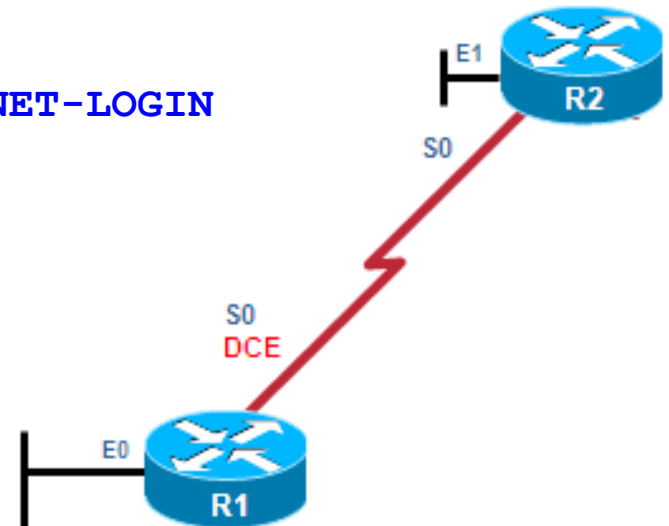
Available method types

Keywords	Description
enable	Uses the enable password for authentication. This keyword cannot be used.
krb5	Uses Kerberos 5 for authentication.
krb5-telnet	Uses Kerberos 5 telnet authentication protocol when using Telnet to connect to the router.
line	Uses the line password for authentication.
local	Uses the local username database for authentication.
local-case	Uses case-sensitive local username authentication.
none	Uses no authentication.
cache <i>group-name</i>	Uses a cache server group for authentication.
group radius	Uses the list of all RADIUS servers for authentication.
group tacacs+	Uses the list of all TACACS+ servers for authentication.
group <i>group-name</i>	Uses a subset of RADIUS or TACACS+ servers for authentication as defined by the aaa group server radius or aaa group server tacacs+ command .

No authentication means authentication always succeeds

Sample authentication configuration

```
R1(config)#username student secret R3st4nt4
R1(config)#username profesor secret dln0saur
R1(config)#aaa new-model
R1(config)#aaa authentication login default local-case enable
R1(config)#aaa authentication login TELNET-LOGIN local-case
R1(config)# line vty 0 4
R1(config-line)#login authentication TELNET-LOGIN
```



Debugging AAA

R1# debug aaa ?

accounting	Accounting
administrative	Administrative
api	AAA api events
attr	AAA Attr Manager
authentication	Authentication
authorization	Authorization
cache	Cache activities
coa	AAA CoA processing
db	AAA DB Manager
dead-criteria	AAA Dead-Criteria Info
id	AAA Unique Id
ipc	AAA IPC
mlist-ref-count	Method list reference counts
mlist-state	Information about AAA method list state change and notification
...	

Successful login in debug mode

```
R1# debug aaa authentication
```

```
113123: Feb 4 10:11:19.305 CST: AAA/MEMORY: create_user (0x619C4940) user=''
```

```
ruser='' port='tty1' rem_addr='async/81560' authen_type=ASCII service=LOGIN priv=1
```

```
113124: Feb 4 10:11:19.305 CST: AAA/AUTHEN/START (2784097690): port='tty1' list=''
```

```
action=LOGIN service=LOGIN
```

```
113125: Feb 4 10:11:19.305 CST: AAA/AUTHEN/START (2784097690): using "default" list
```

```
113126: Feb 4 10:11:19.305 CST: AAA/AUTHEN/START (2784097690): Method=LOCAL
```

```
113127: Feb 4 10:11:19.305 CST: AAA/AUTHEN (2784097690): status = GETUSER
```

```
113128: Feb 4 10:11:26.305 CST: AAA/AUTHEN/CONT (2784097690): continue_login
```

```
(user='(undef)')
```

```
113129: Feb 4 10:11:26.305 CST: AAA/AUTHEN (2784097690): status = GETUSER
```

```
113130: Feb 4 10:11:26.305 CST: AAA/AUTHEN/CONT (2784097690): Method=LOCAL
```

```
113131: Feb 4 10:11:26.305 CST: AAA/AUTHEN (2784097690): status = GETPASS
```

```
113132: Feb 4 10:11:28.145 CST: AAA/AUTHEN/CONT (2784097690): continue_login
```

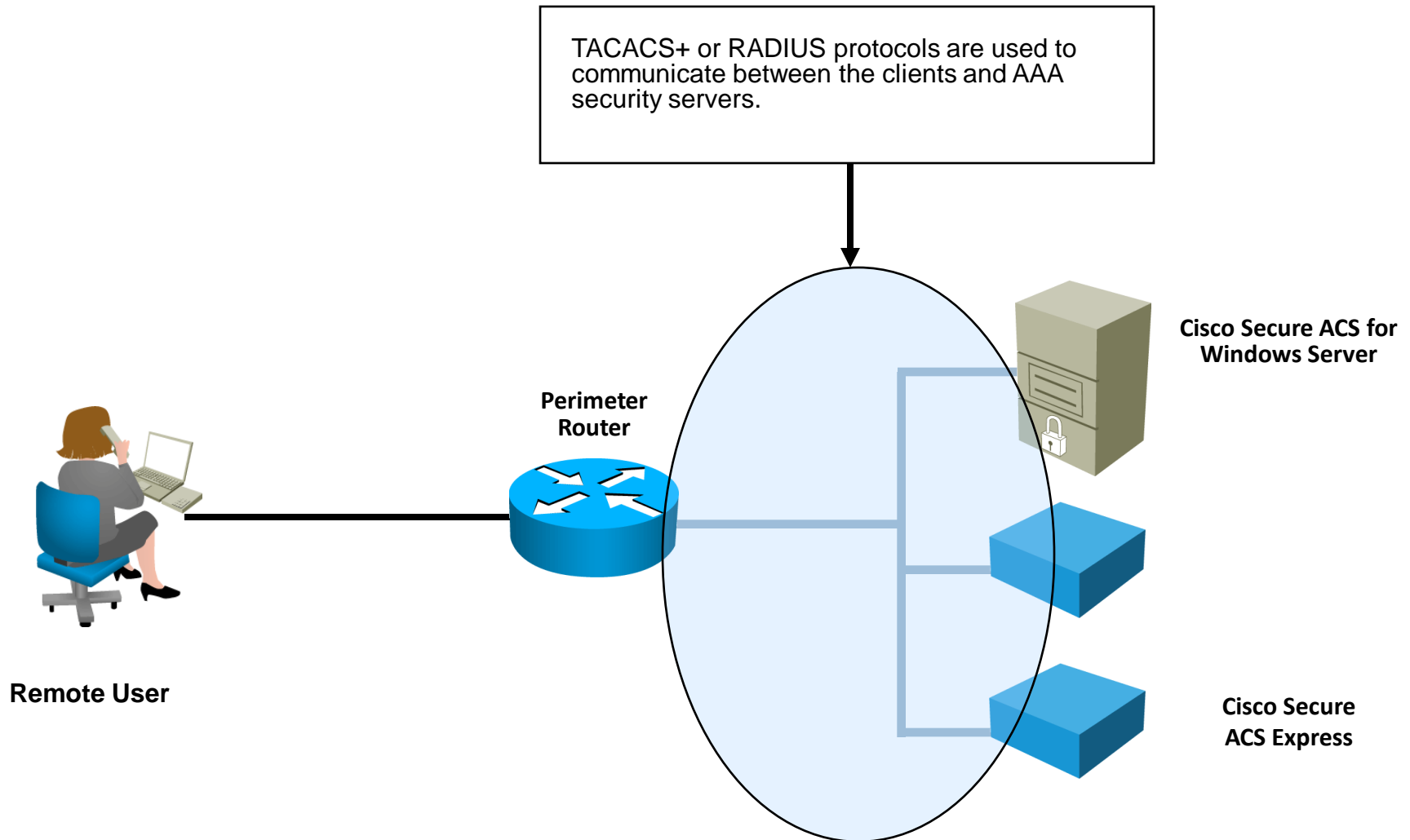
```
(user='diallocal')
```

```
113133: Feb 4 10:11:28.145 CST: AAA/AUTHEN (2784097690): status = GETPASS
```

```
113134: Feb 4 10:11:28.145 CST: AAA/AUTHEN/CONT (2784097690): Method=LOCAL
```

```
113135: Feb 4 10:11:28.145 CST: AAA/AUTHEN (2784097690): status = PASS
```

Overview of TACACS+ and RADIUS



RADIUS

- ▶ Remote Authentication Dial-In User Service
- ▶ IETF standard
- ▶ Uses UDP ports 1812, 1813
 - ▶ Cisco Secure ACS is more special: uses UDP ports 1645, 1646
- ▶ Authenticating users:

System database

- For UNIX servers, a file like `/etc/passwd`

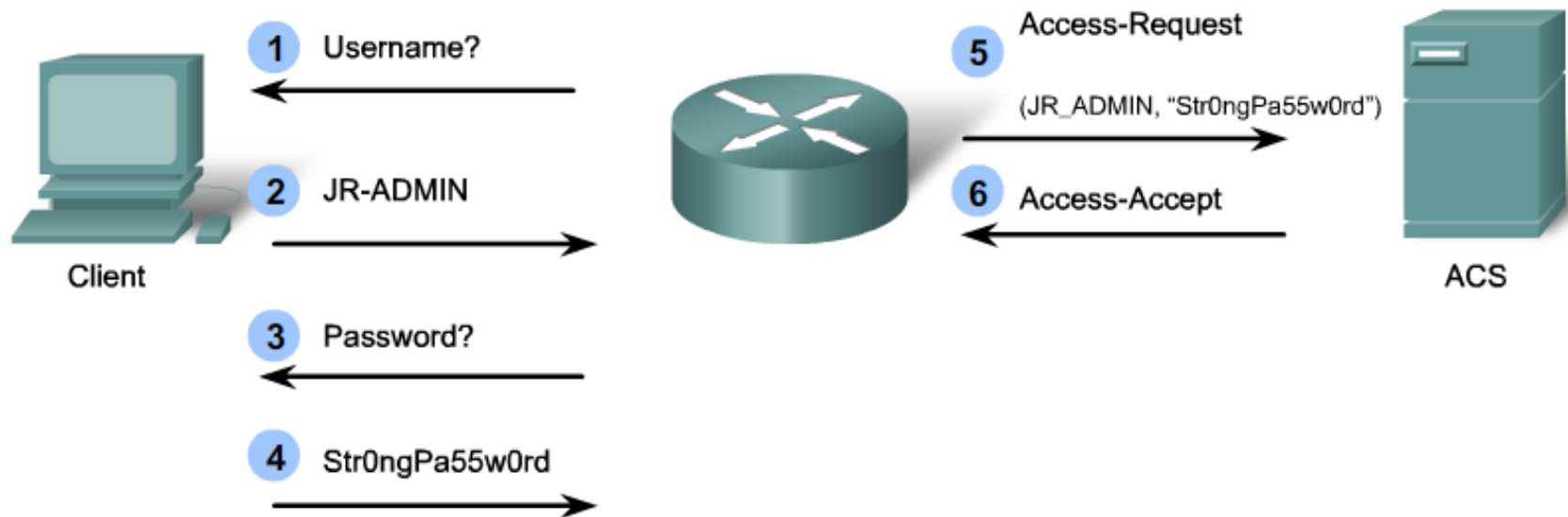
Internal database

- Use RADIUS internal database

SQL authentication

- External SQL database. No format restrictions.

RADIUS authentication process



RADIUS security

- ▶ **Security:**

- ▶ Packet contents are not encrypted so data can be sniffed
- ▶ Only passwords are encrypted (RADIUS uses PAP)
- ▶ RADIUS cannot separately control user authorization
 - ▶ It is by default part of the authentication process

- ▶ **Message authentication:**

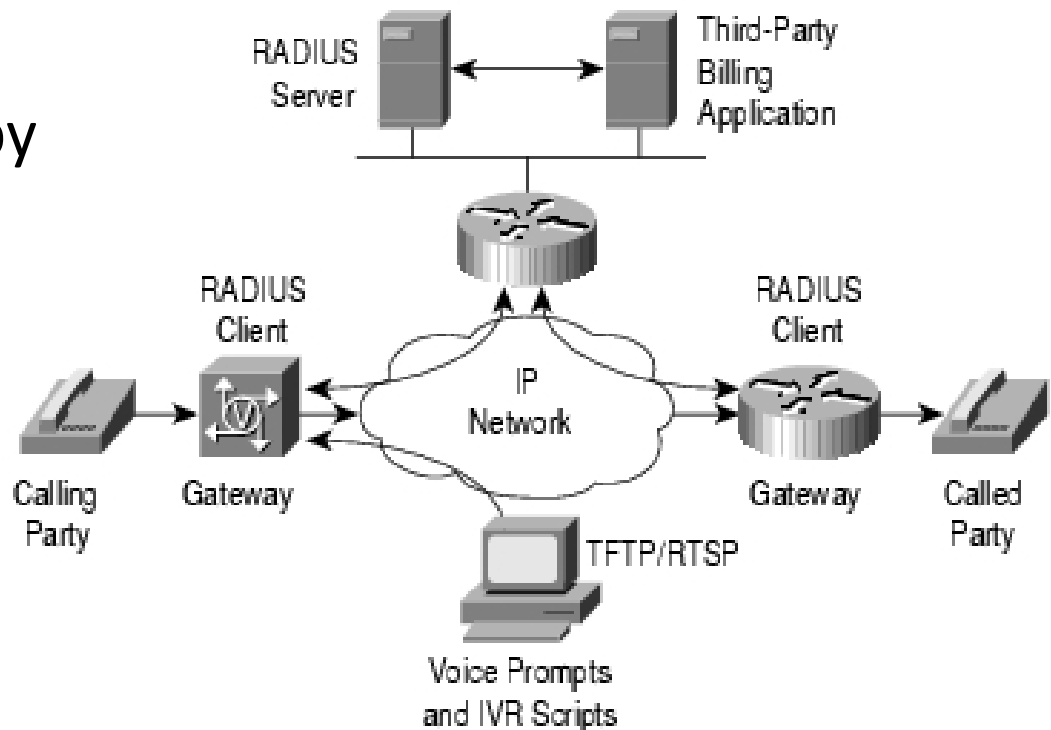
- ▶ Messages are authenticated using an MD5 fingerprint

- ▶ **Message structure:**

- ▶ The router gathers all authentication details before forwarding them
- ▶ Uses Attribute-Value messages between the router and the server

RADIUS usage

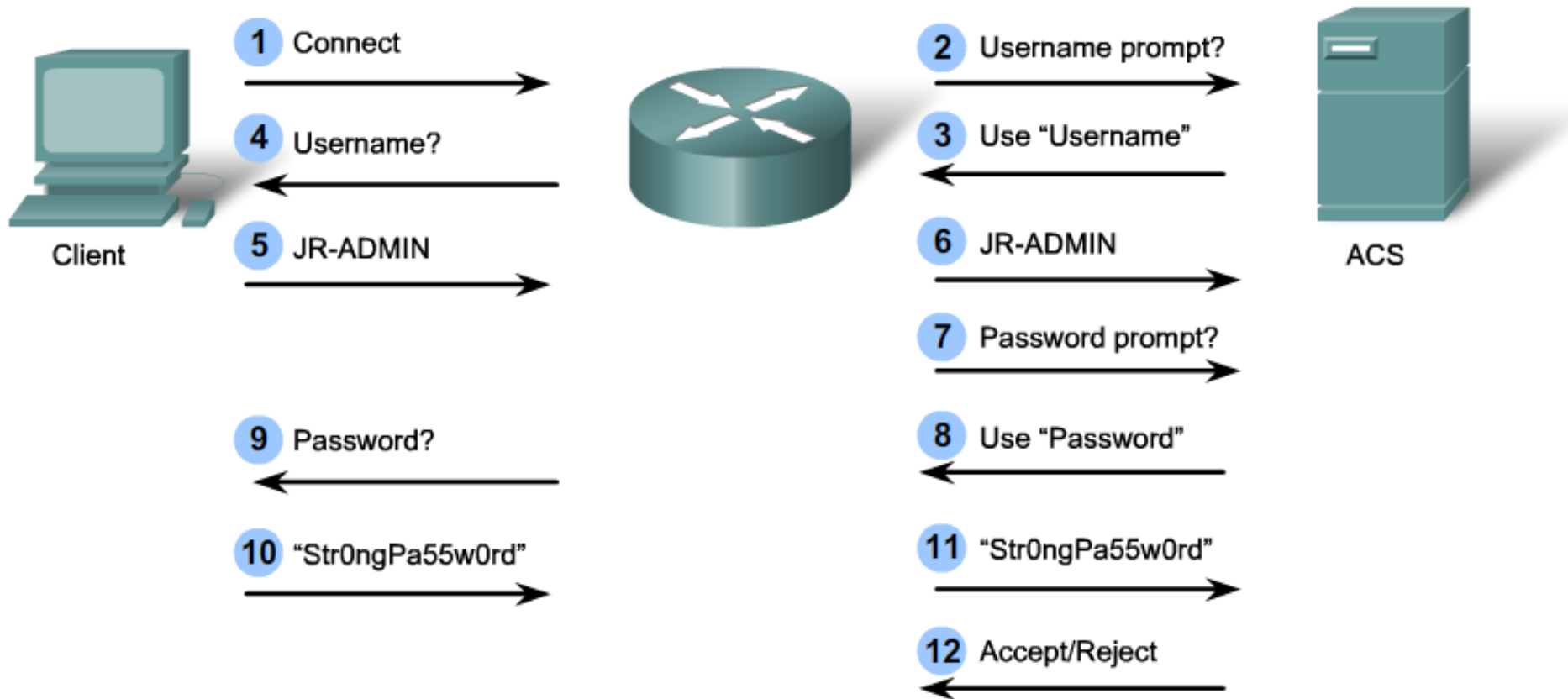
- ▶ Widely used by VoIP service providers
- ▶ It passes login credentials of a SIP endpoint (like an IP phone) to a SIP registrar and then to a RADIUS server
- ▶ Also, commonly used by the 802.1x security standard
 - ▶ switchport security
 - ▶ WLAN security



TACACS+

- ▶ Supported by Cisco ASA, PIX firewalls, Cisco routers
- ▶ Uses TCP port 49
- ▶ Extensive authorization features:
 - ▶ Can upload per-user ACLs and static routes to the router upon user authorization
 - ▶ The router queries the TACACS+ server on behalf of the user in order to check whether it should execute a command
 - ▶ Sets a TCP session for each authorization request
 - ▶ Induces delays -> Cisco Secure ACS supports a single connection

TACACS+ authentication process

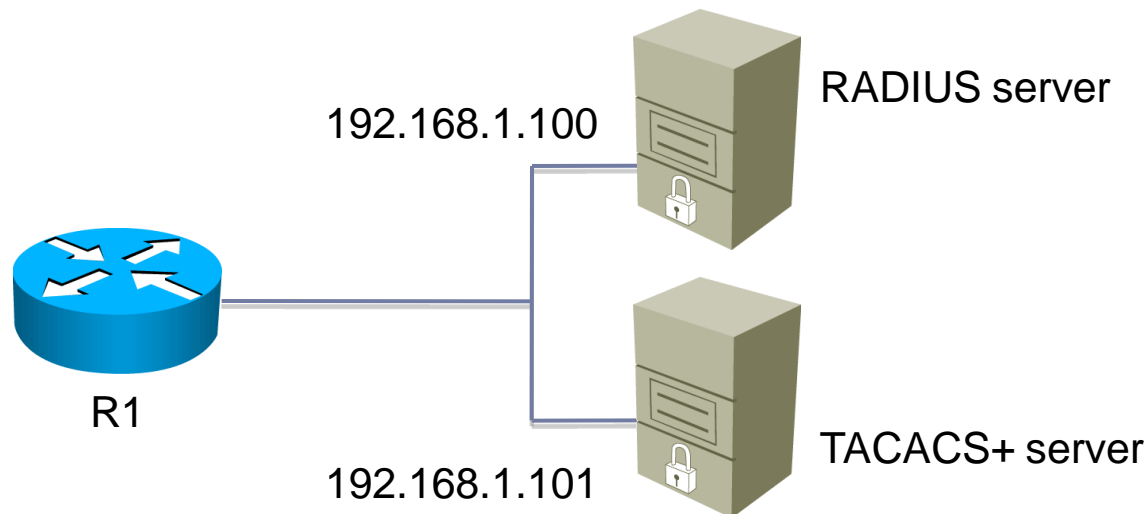


TACACS+ vs RADIUS comparison breakdown

	TACACS+	RADIUS
Functionality	Separates AAA according to the AAA architecture, allowing modularity of the security server implementation	Combines authentication and authorization but separates accounting, allowing less flexibility in implementation than TACACS+.
Standard	Mostly Cisco supported	Open/RFC standard
Transport Protocol	TCP	UDP
CHAP	Bidirectional challenge and response as used in Challenge Handshake Authentication Protocol (CHAP)	Unidirectional challenge and response from the RADIUS security server to the RADIUS client.
Protocol Support	Multiprotocol support	No ARA, no NetBEUI
Confidentiality	Entire packet encrypted	Password encrypted
Customization	Provides authorization of router commands on a per-user or per-group basis.	Has no option to authorize router commands on a per-user or per-group basis
Accounting	Limited	Extensive

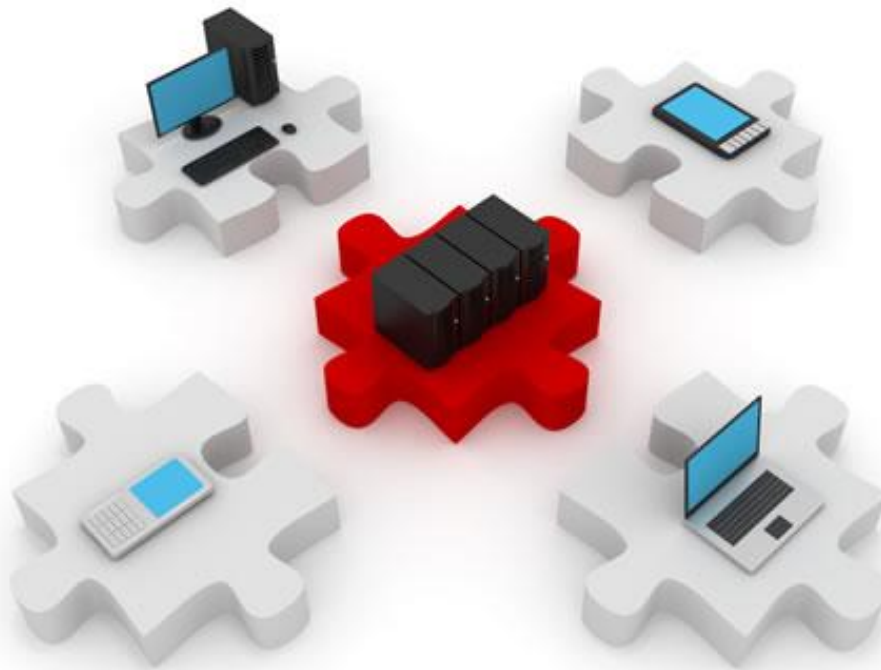
RADIUS and TACACS+ authentication commands

```
R1 (config)#aaa new-model
R1 (config)#tacacs-server host 192.168.1.101 single-connection
R1 (config)#tacacs-server key TACACS-p@$word
R1 (config)#radius-server host 192.168.1.100
R1 (config)#radius-server key RADIUS-p@$word
R1 (config)#aaa authentication login default group tacacs+ group
radius local-case
```



AAA authentication modes

- ▶ AAA can be used to authenticate users for:
 - ▶ Remote network access
 - ▶ Administrative access
- ▶ There are two modes to request AAA services:
 - ▶ Character mode: the user sends a request to access an EXEC process, for administrative purposes
 - ▶ login, exec, enable commands
 - ▶ Packet mode: the user sends a request to establish a connection with a device on the network
 - ▶ ppp and network commands



AAA: Authorization

Server-based AAA authorization

- ▶ RADIUS does not separate authentication from authorization
- ▶ TACACS+ considers them as separate processes
- ▶ Cisco IOS allows authorization to control user access to certain commands
- ▶ Authorization process:
 - ▶ The user send a command to the router
 - ▶ The router asks the AAA server if the user is authorized to run the command
 - ▶ The AAA server gives a PASS/FAIL response and the router acts accordingly

Configuring server-based authorization

► Authorization command syntax:

```
aaa authorization {network | exec | commands level} {default | list-name}  
method1...[method4]
```

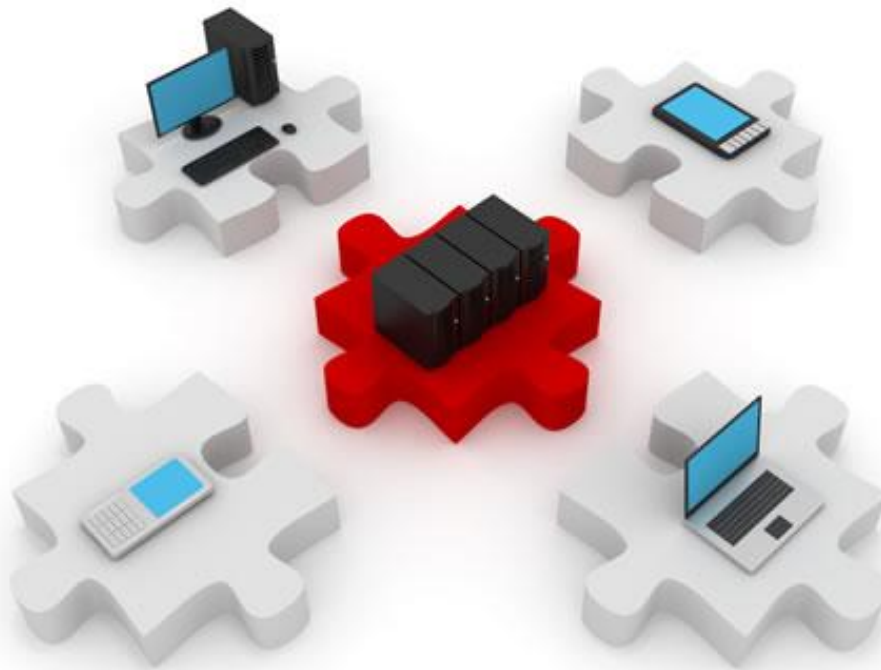
Network: for network services (SLIP, PPP)

Exec: for starting an exec (shell) session

Commands *level*: for exec commands

Configuring authorization example

```
R1(config)#username student secret R3st4nt4
R1(config)#username profesor secret dln0saur
R1(config)#aaa new-model
R1(config)#aaa authentication login default group radius local-
case
R1(config)#aaa authentication login TELNET-LOGIN group tacacs+
local-case
R1(config)#aaa authorization exec default group tacacs+
R1(config)#aaa authorization network default group tacacs+
R1(config)#aaa authorization commands 15 default group tacacs+
R1(config)# line vty 0 4
R1(config-line)#login authentication TELNET-LOGIN
R1(config-line)#authorization commands 15 default
R1(config-line)#authorization exec default
```



AAA: Accounting

Server-based AAA accounting

- ▶ Keep track of individual activity
- ▶ Important from a financial, management but also from a security point of view
 - ▶ Why did that user access the server at 4am?
- ▶ Useful for creating a list of changes occurring on the network
 - ▶ Who made the changes?
 - ▶ When did they occur?
 - ▶ What was changed?
- ▶ The default accounting list automatically applies to all interfaces

Server-based accounting configuration

- ▶ To configure AAA accounting, use the global configuration mode command:

```
aaa accounting { network | exec | commands level } {default | list-name}  
{start-stop | stop-only | none} [broadcast] method1...[method4]
```

- ▶ The default or the <list-name> list can be defined.
- ▶ The trigger (start-stop, stop-only, none) specify which events cause the accounting records to be updated.

Server-based accounting example

- For example, to log the use of exec sessions and network connections:

```
R1(config)# aaa accounting exec default start-stop group tacacs+
R1(config)# aaa accounting network default start-stop group
tacacs+
```

The final slide 😊

*Amateurs hack systems,
professionals hack people.*

Bruce Schneier