

TRUE Blind ip spoofed portscanning

Thomas Olofsson

C.T.O

Defcom

Agenda

- Background on TCP ip handshakes
- Background on traditional scanning
- ID numbers and their prediction
- Spoofed scanning in theory and practice.
- The code and examples
- Demo
- Q/A

Background on TCP handshakes

- Definitions
- Tcp header
- Traditional 3 way handshake

Definitions

- An open connection between two computers communicating by TCP/IP is called a socket and is defined by:
 - Source IP number
 - Source Port number
 - Destination IP number
 - Destination Port number
 - Initial source SEQ number
 - Initial destination SEQ number
 - AN ID # that is increased for each packet

TCP packet header

16-bit source port number		16-bit destination port number	
32-bit sequence number			
32-bit acknowledgement number			
length	unused	flags	16-bit window size
16-bit TCP checksum		16-bit urgent offset	
Options (if any)			
Data (if any)			

Traditional TCP/IP handshake

Src ip, Dst ip

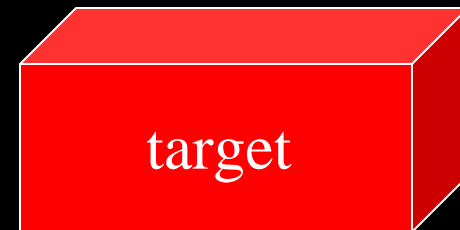
Src prt, Dst Prt

Syn = in seq#

Ack = NULL

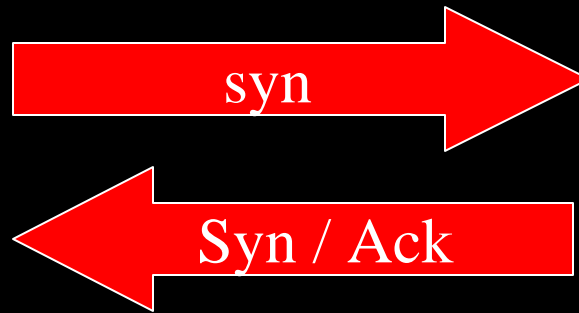
Flags = S

Src ID = src ID + 1



Traditional TCP/IP handshake

Src ip, Dst ip
Src prt, Dst Prt
Syn = src seq#
Ack = NULL
Flags = S



Src ip, Dst ip
Src prt, Dst Prt
Syn = Dst seq#
Ack = src seq# + 1
Flags =
Dst ID = Dst ID + 1



Traditional TCP/IP handshake

Src ip, Dst ip

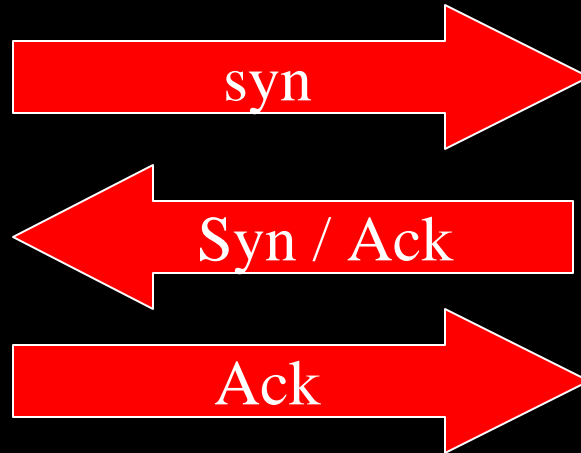
Src prt, Dst Prt

Syn = src seq#

Ack = dst seq# + 1

Flags = A

Src ID = src ID + 1



Establishing a socket



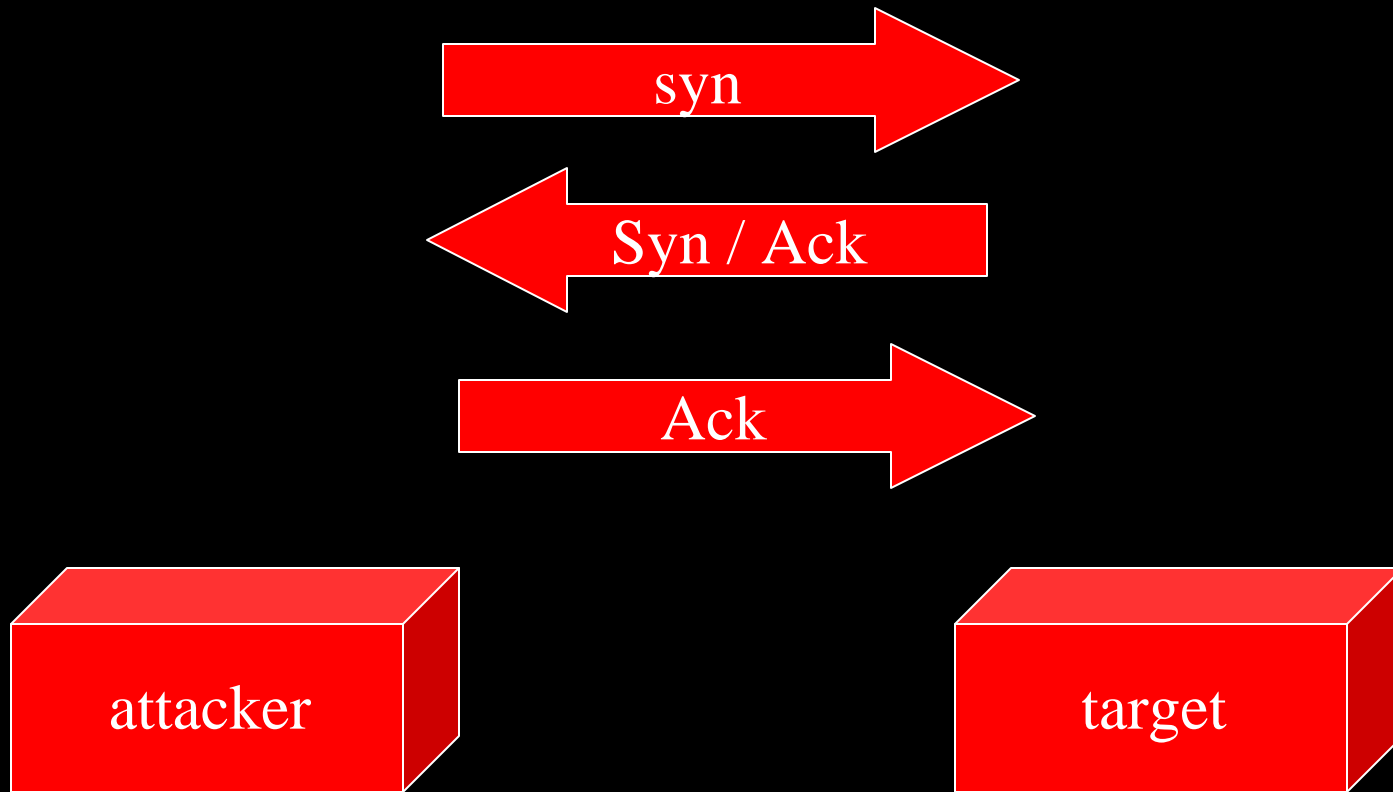
SYN (seq_a)

SYN/ACK ($seq_b/ack = seq_a + 1$)

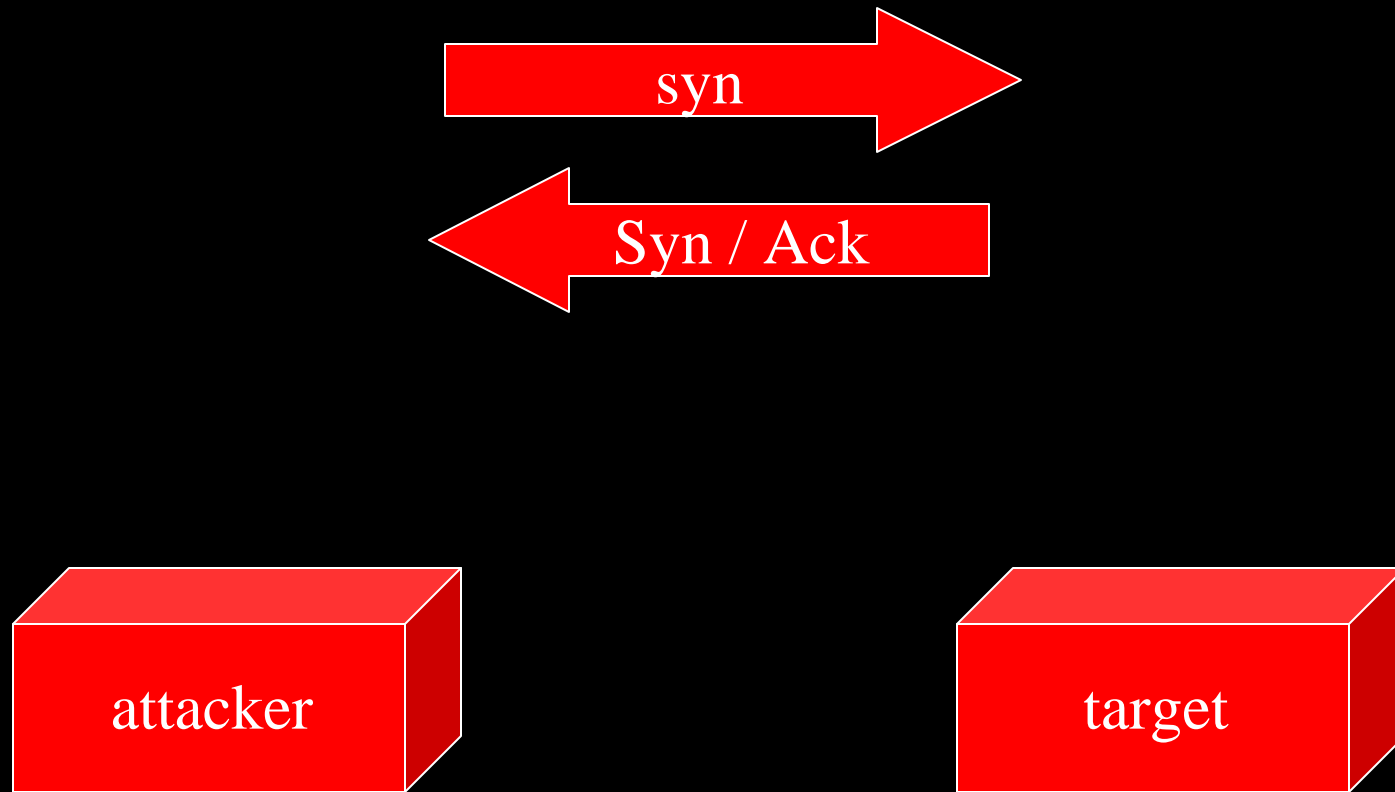
ACK ($ack = seq_b + 1$)

Traditional port scanning

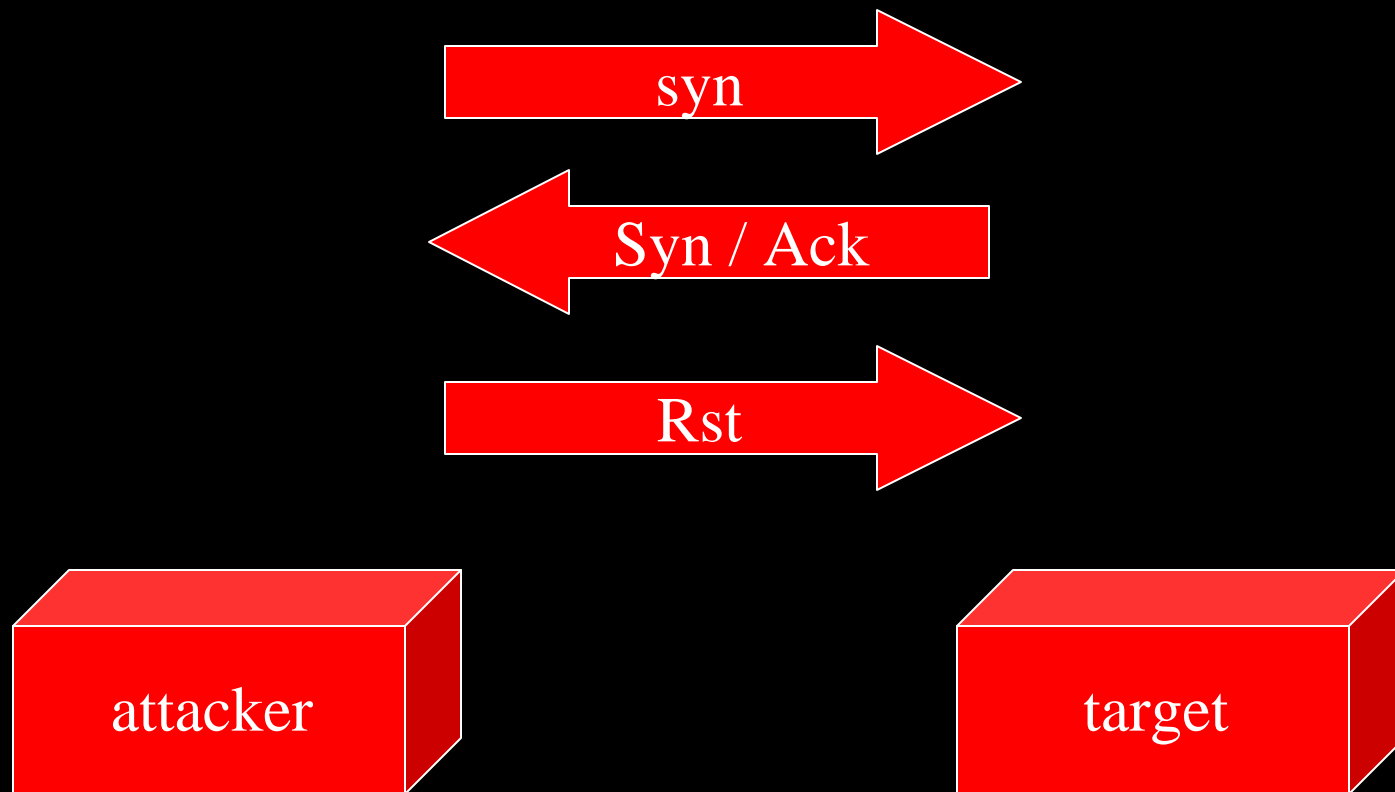
Traditional port scanning



Traditional stealth scanning 1



Traditional stealth scanning 2



Sequence numbers

Are in place to provide easy packet reassembly.

Increments each time a packet is sent.

Various incrementation schemes exist

ID flag

- Are in place to identify each tcp session
- Is also in some cases used for packet reassembly
- The id counter is increased every time a packet is sent
- This is valid for all packets including reset packets

ID flag prediction

- Most unix boxes increments the ID flag by a random or pseudo random number.
- Up till today id numbers has not been known to be security critical.
- Windows 95 boxes tend to increment id# by 1
- Windows 2000 boxes seems to increment id# by 254
- This due to reversed byte ordering of the id# in these operating systems.

Spoofer scanning in theory

- By constantly polling a decoy host for id number increments we can see If the scanned target host has sent it syn/ack or reset packets.
- By analyzing this we will know whether a port on the scanned host is open or not
- This is done totally blind from the scanned host.

Spoofer scanning in theory

- Since we know a win box will increase the id# by sending a packet we can by constantly probing the host see how many packets it has sent between our polls
- This is done by monitoring the ID# increment

Spoofed scanning in theory

- If a port is open on a scanned host the server will respond with a syn/ack
- If a port is closed on the scanned host it will respond with a rst

Spoofer scanning in theory

If a host receives a syn ack from a unknown source it will send a rst packet back

If a host receives a rst packet from a unknown source it will NOT send a packet back

Spooned scanning in practice

Or how it all fits together

Introducing our players



172.0.0.1



10.0.0.1



192.0.0.1

Why do we need three of them



www.anycompany.com:80

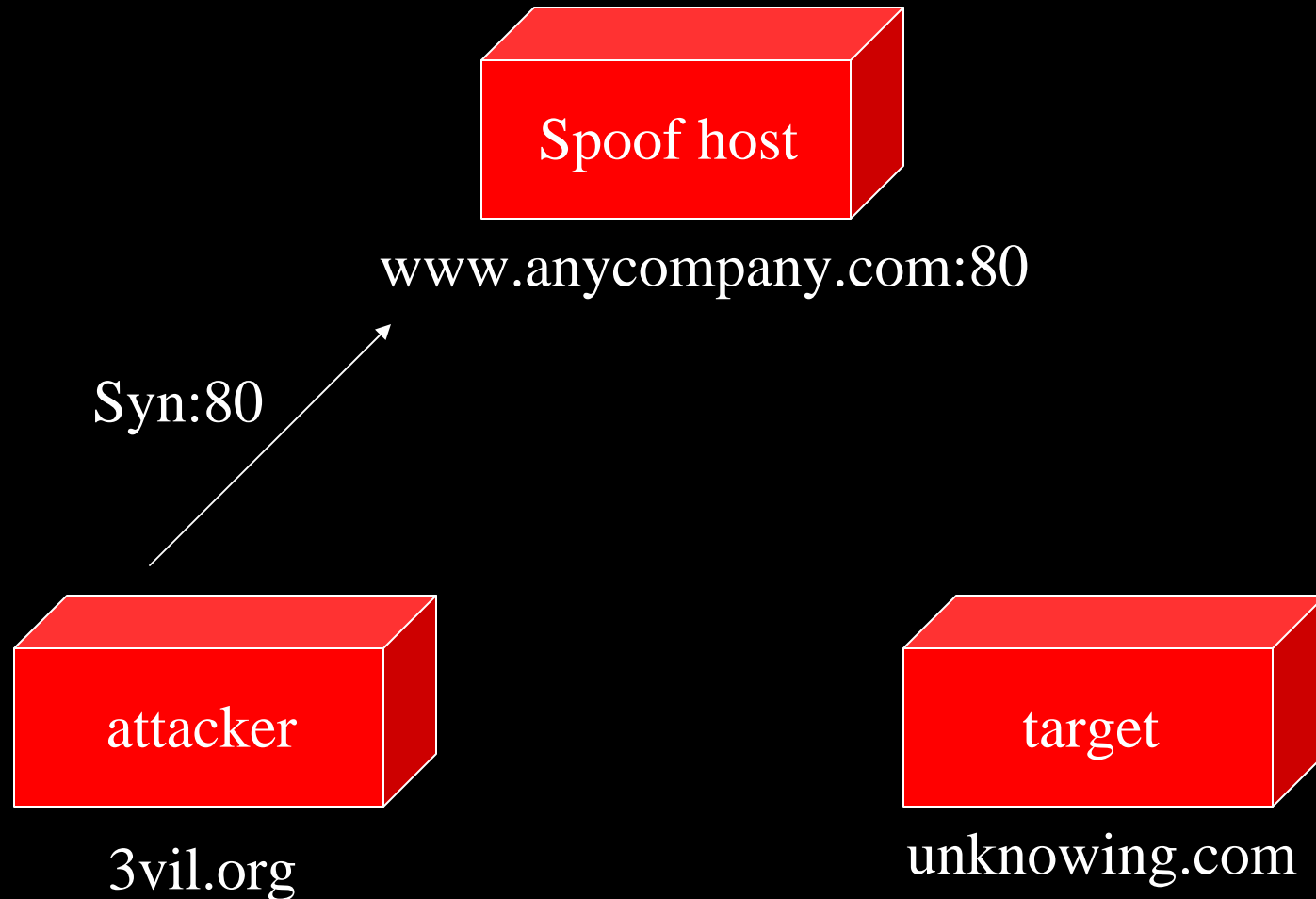


3vil.org

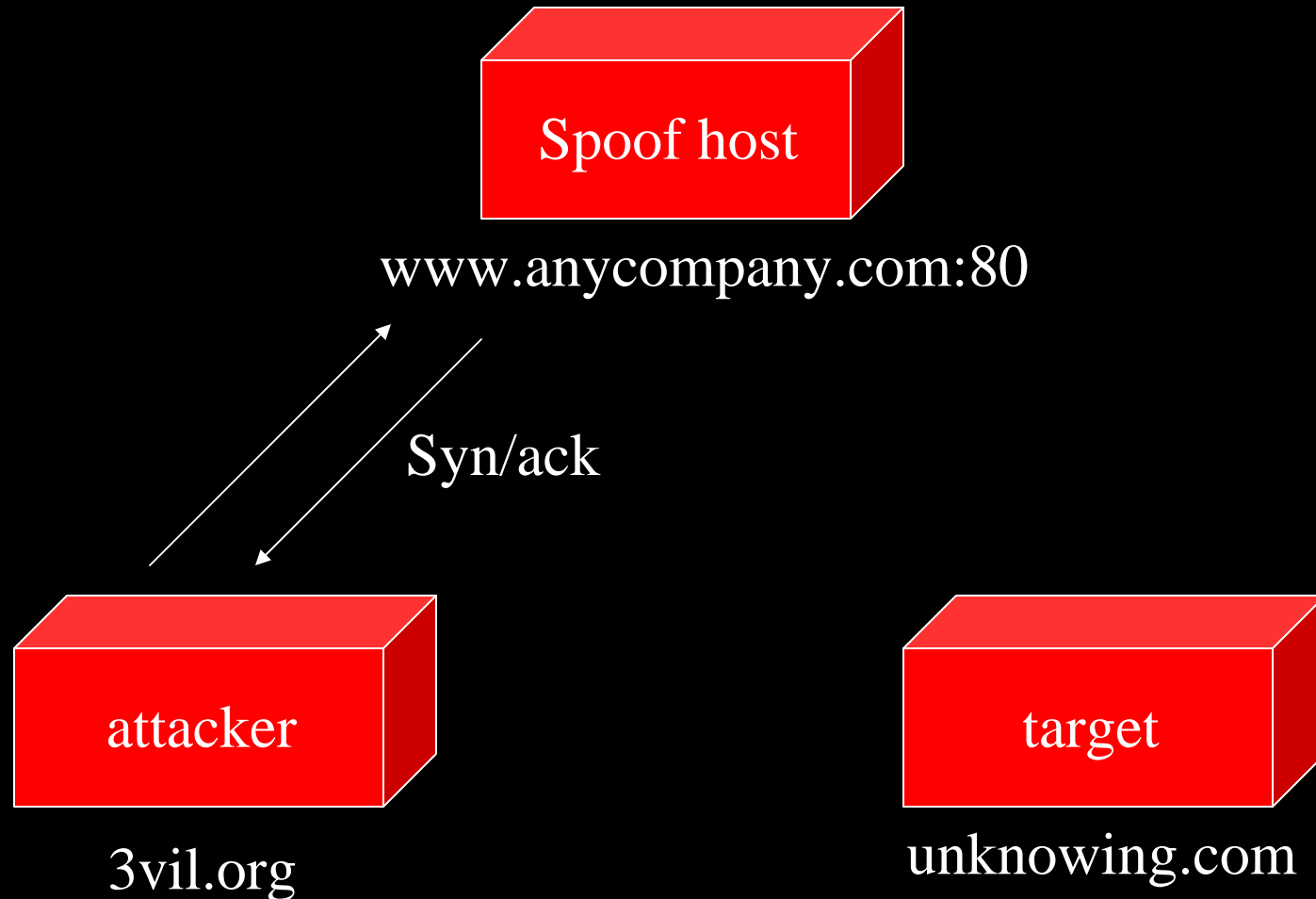


unknowing.com

Phase one (sync the id# of spoof)



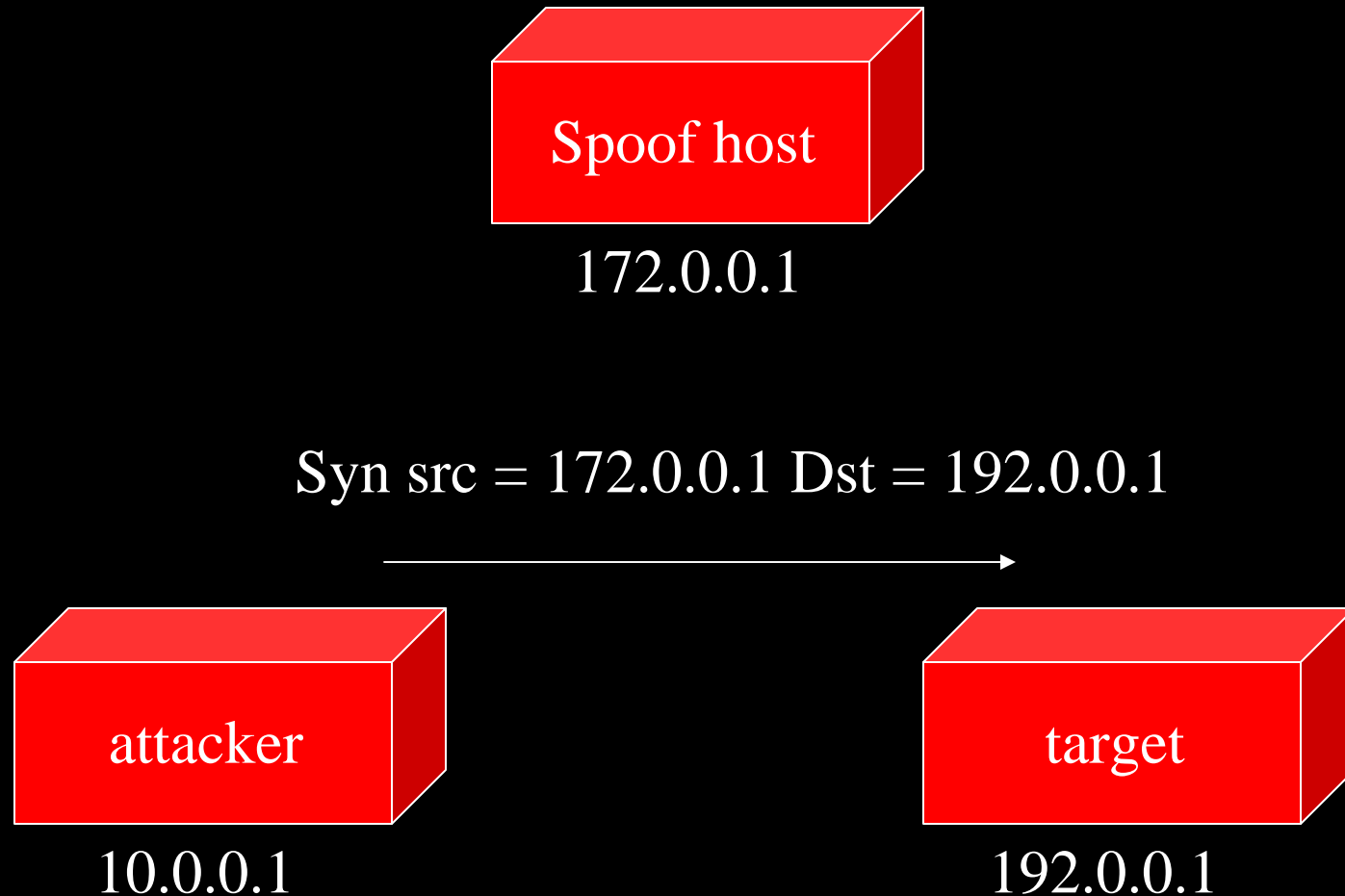
Phase one (sync the id# of spoof)



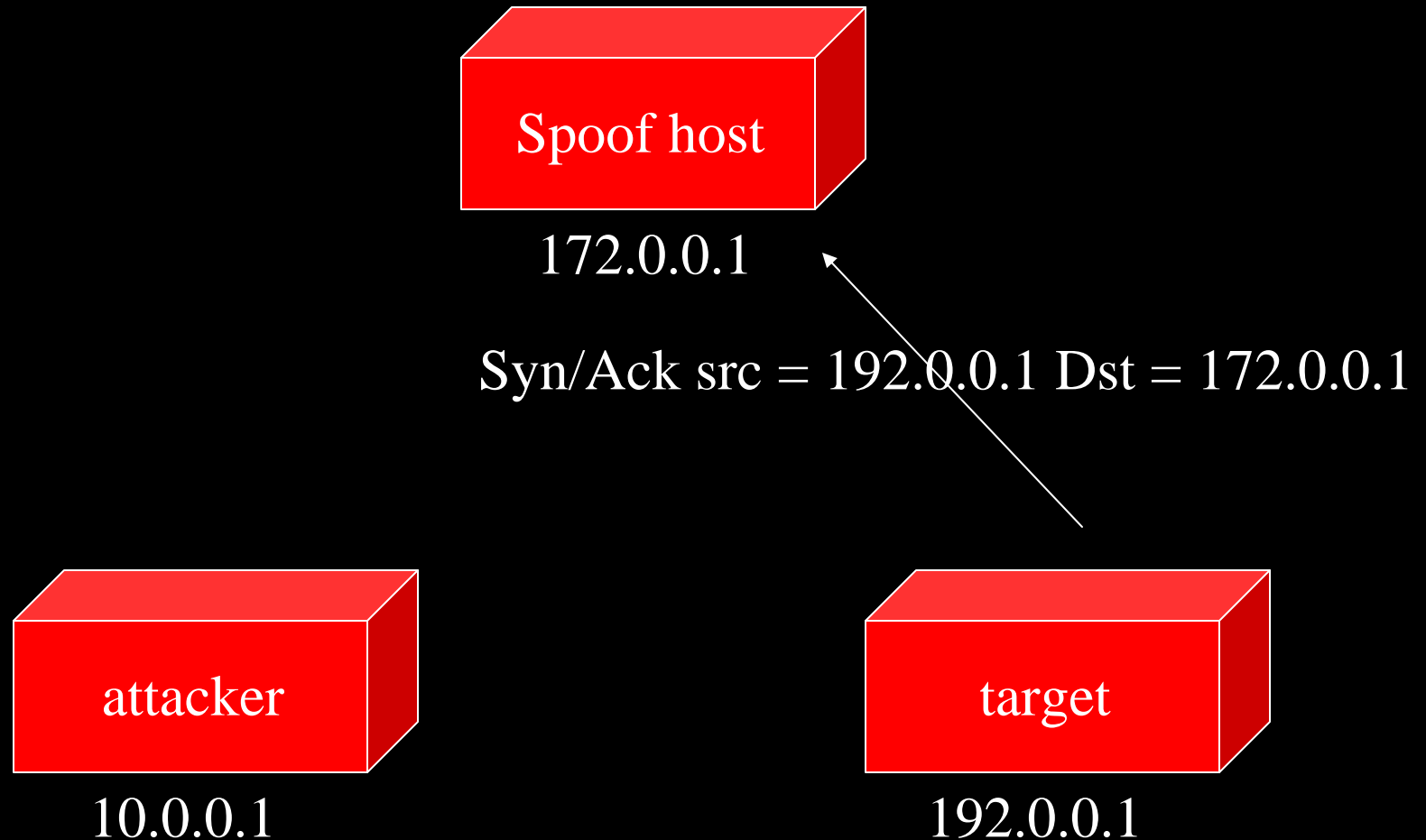
Why did we do that

- Attacker now knows the spoofs initial ID#

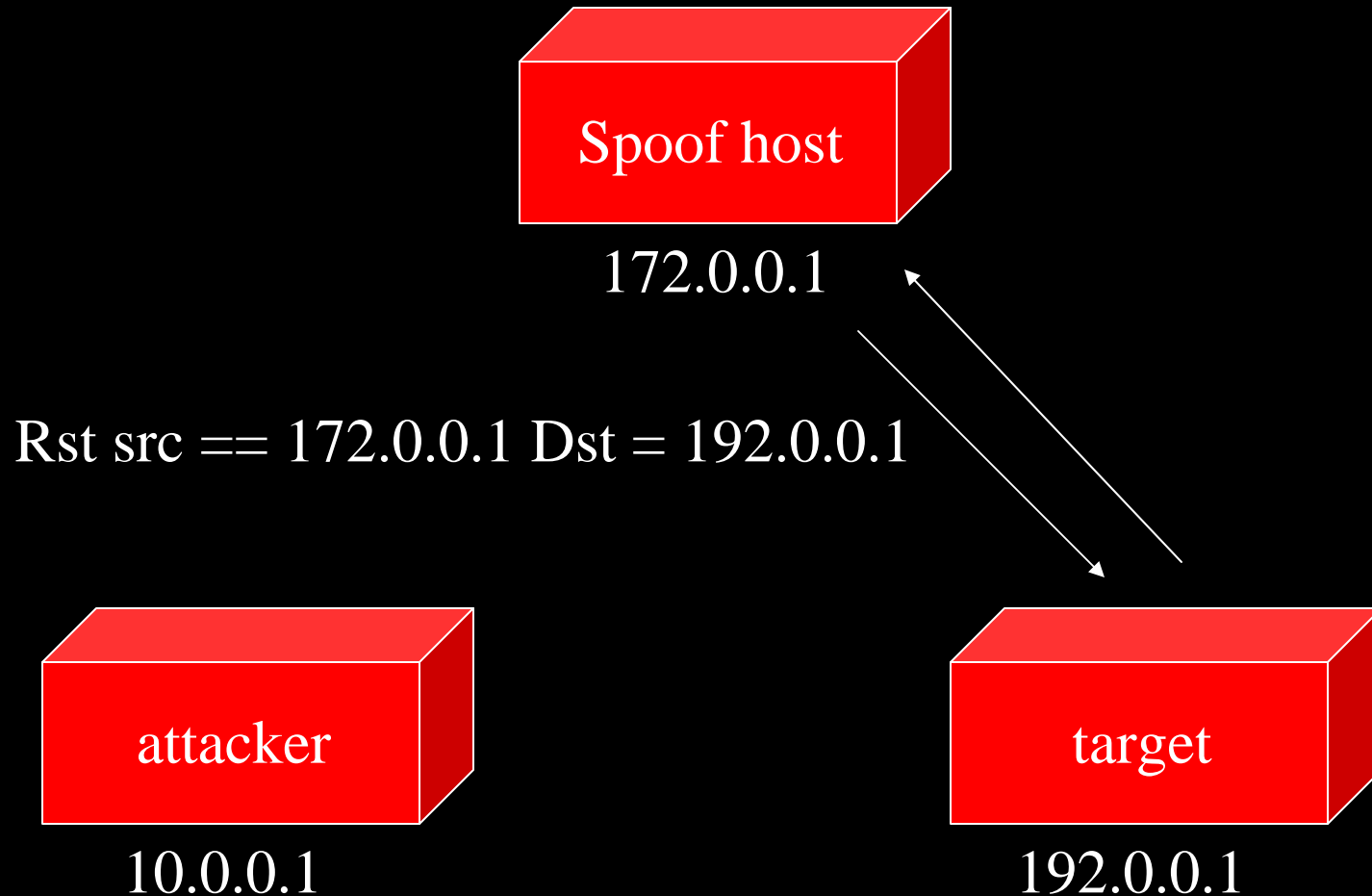
Phase2 (spoofing the source)



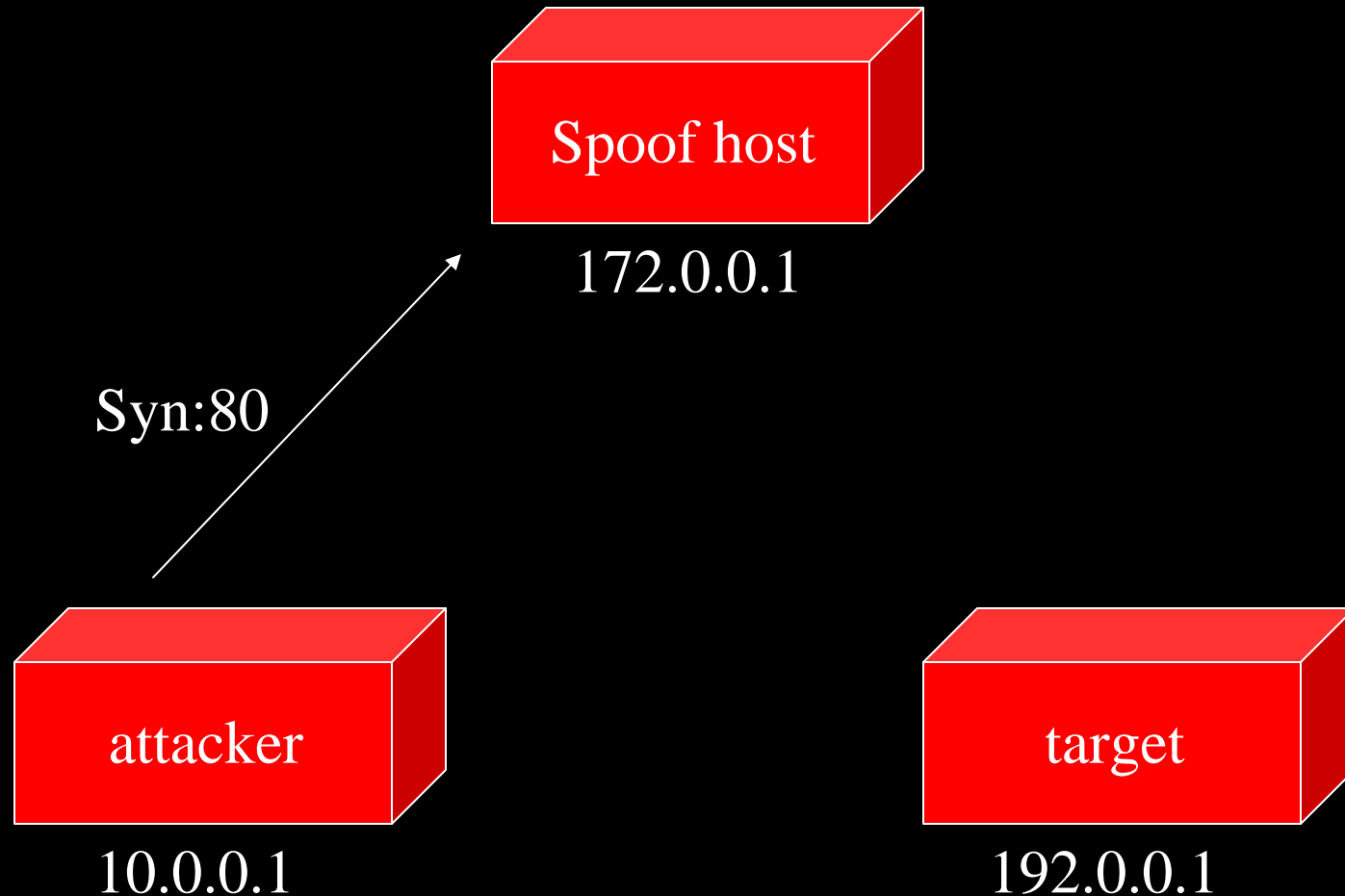
Phase 3 (fooling the respons)



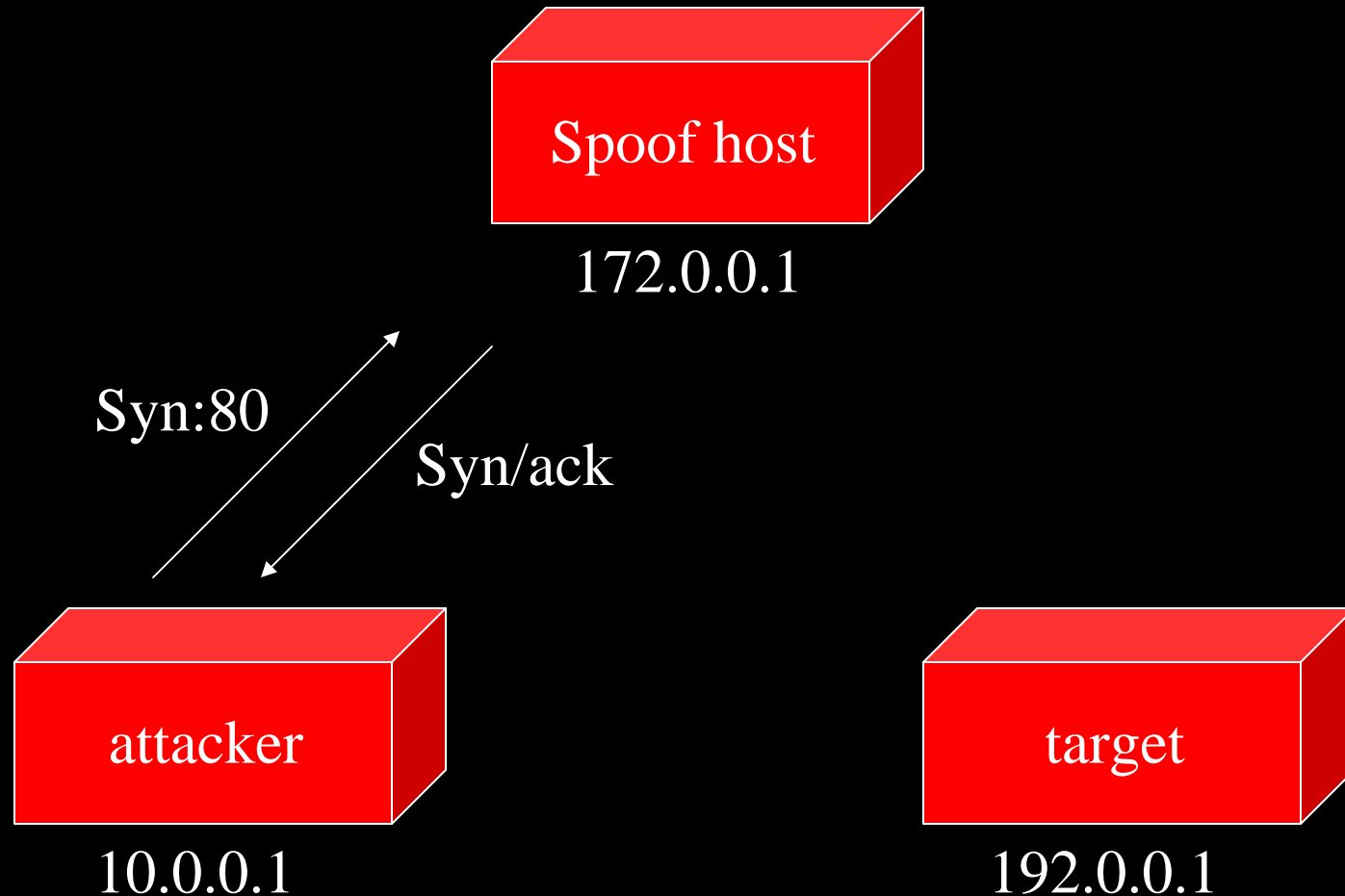
Phase 3 (fooling the respons)



Phase 4 (probing the spoof host)



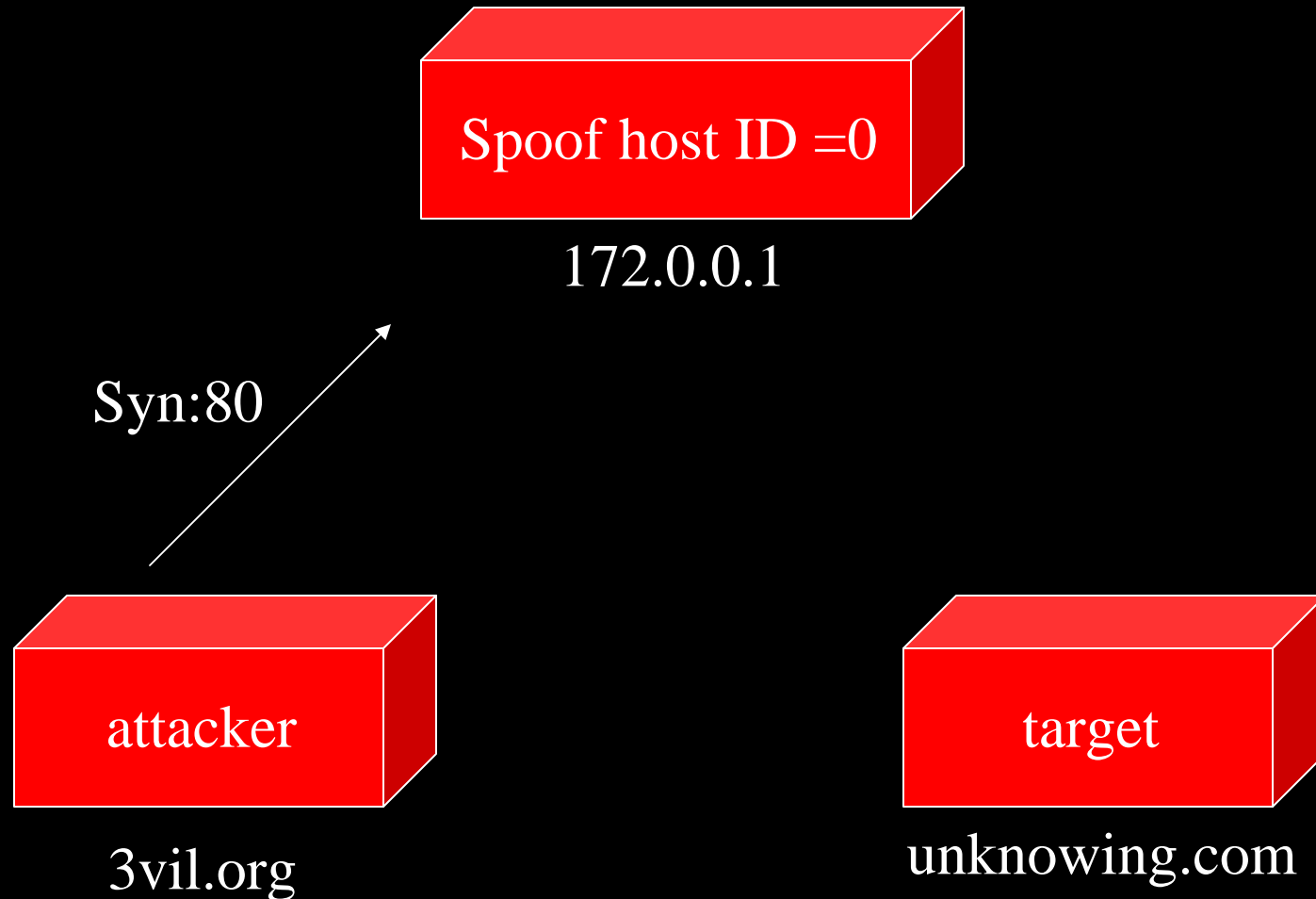
Phase 4 (probing the spoof host)



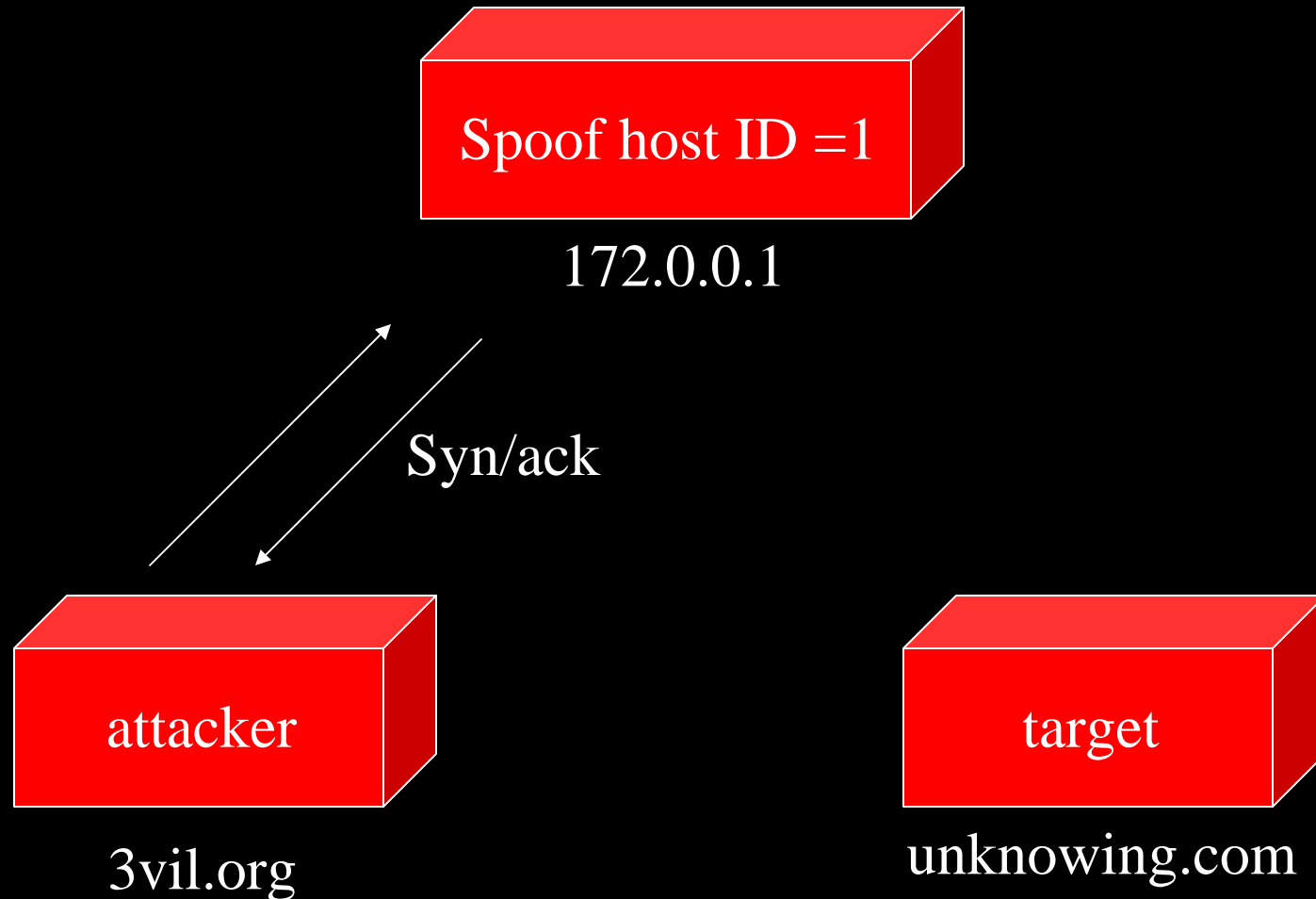
Case port open

Adding the ID counters

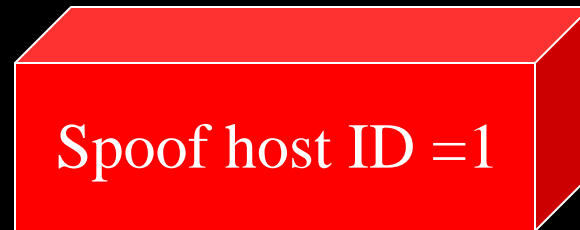
Phase one (sync the id# of spoof)



Phase one (sync the id# of spoof)



Phase2 (spoofing the source)



172.0.0.1

Syn src = 172.0.0.1 Dst = 192.0.0.1

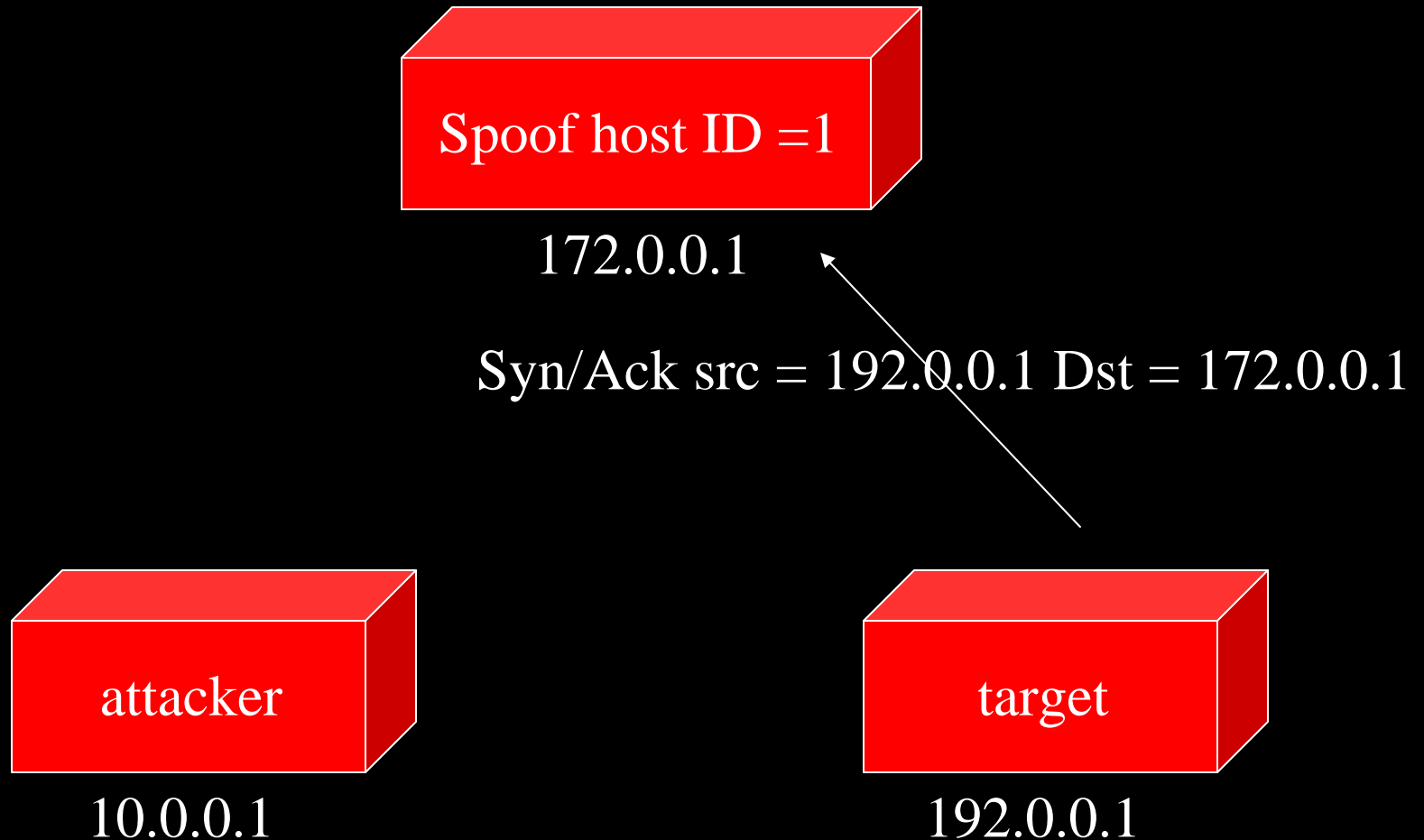


10.0.0.1

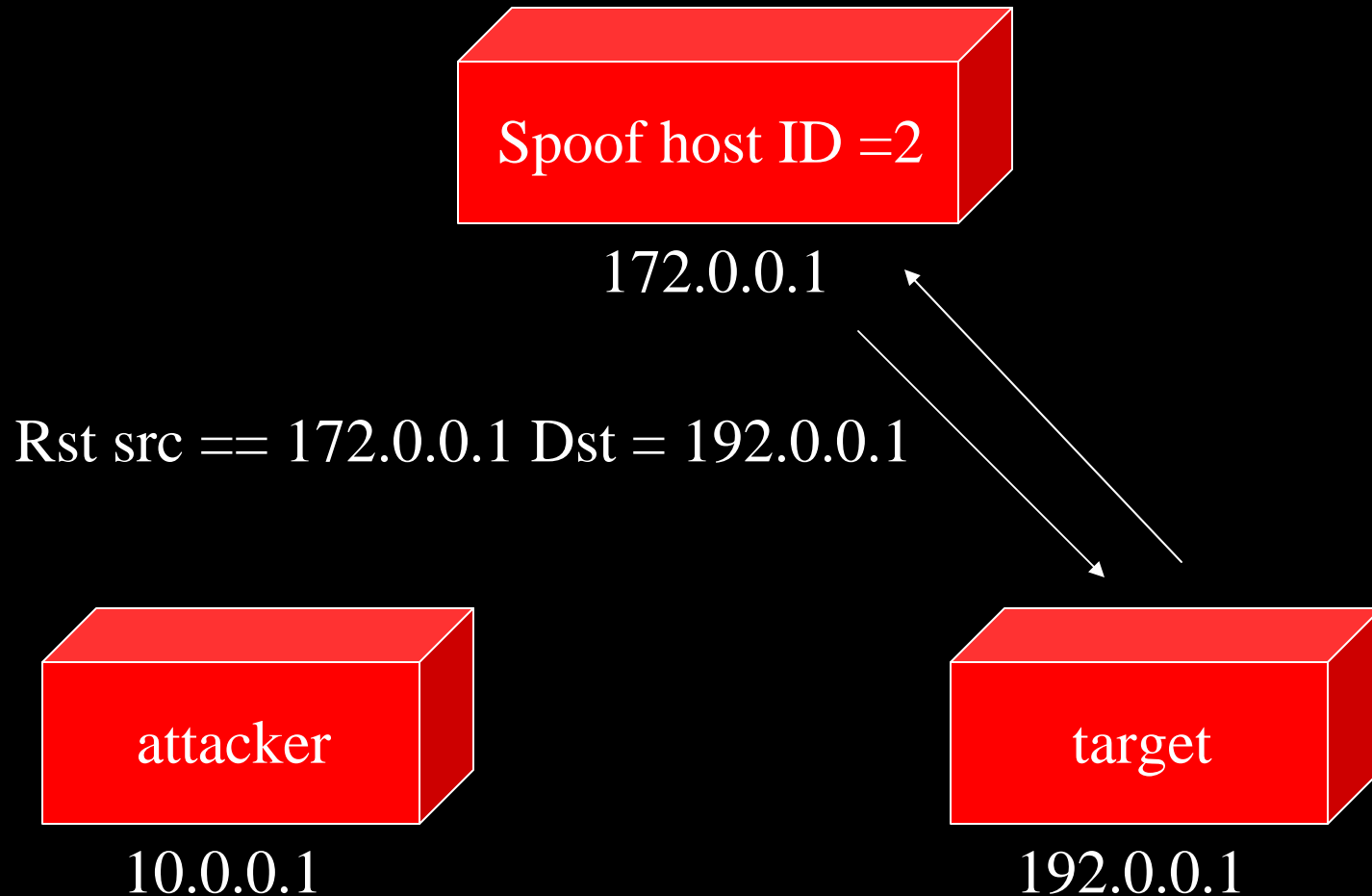


192.0.0.1

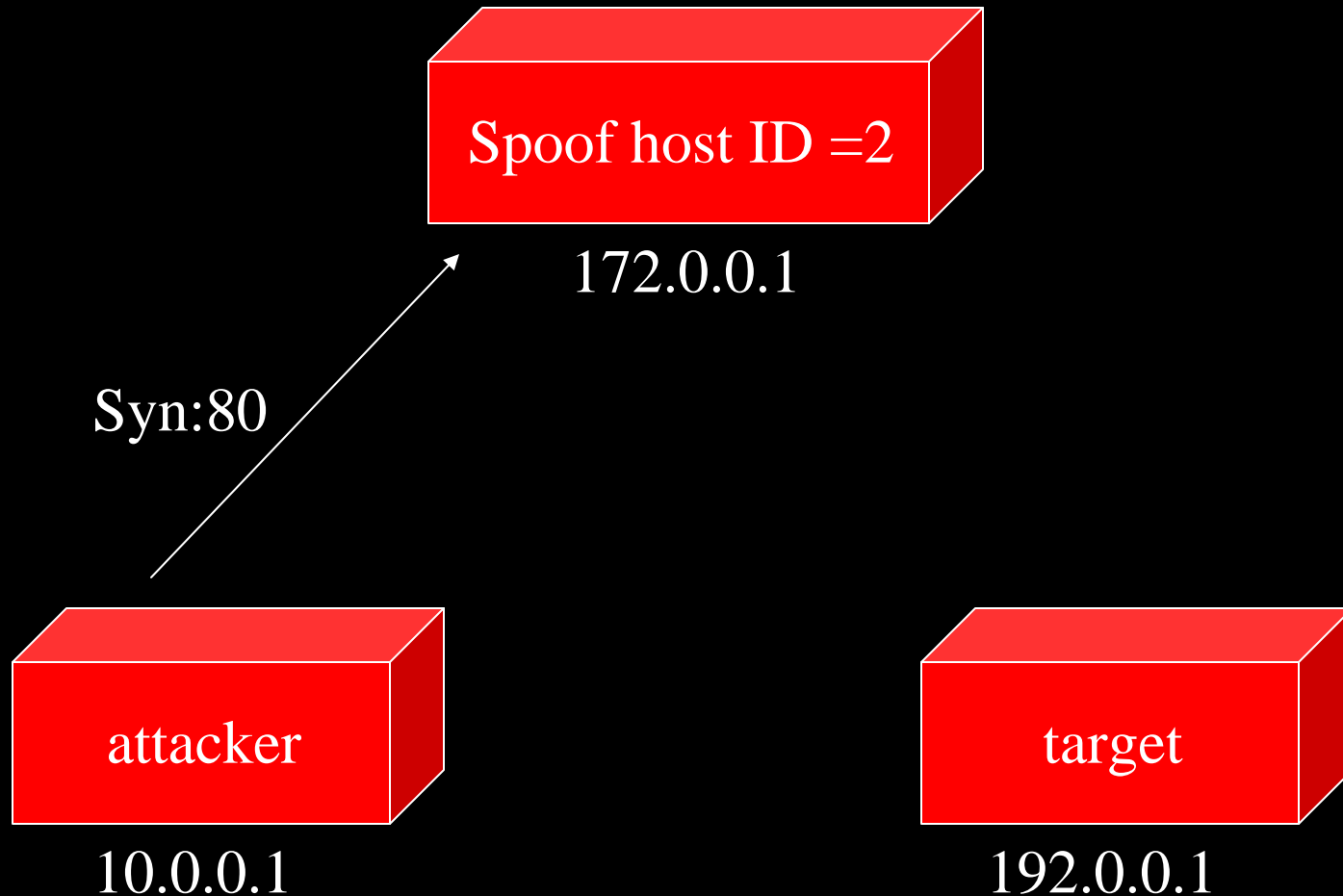
Phase 3 (fooling the respons)



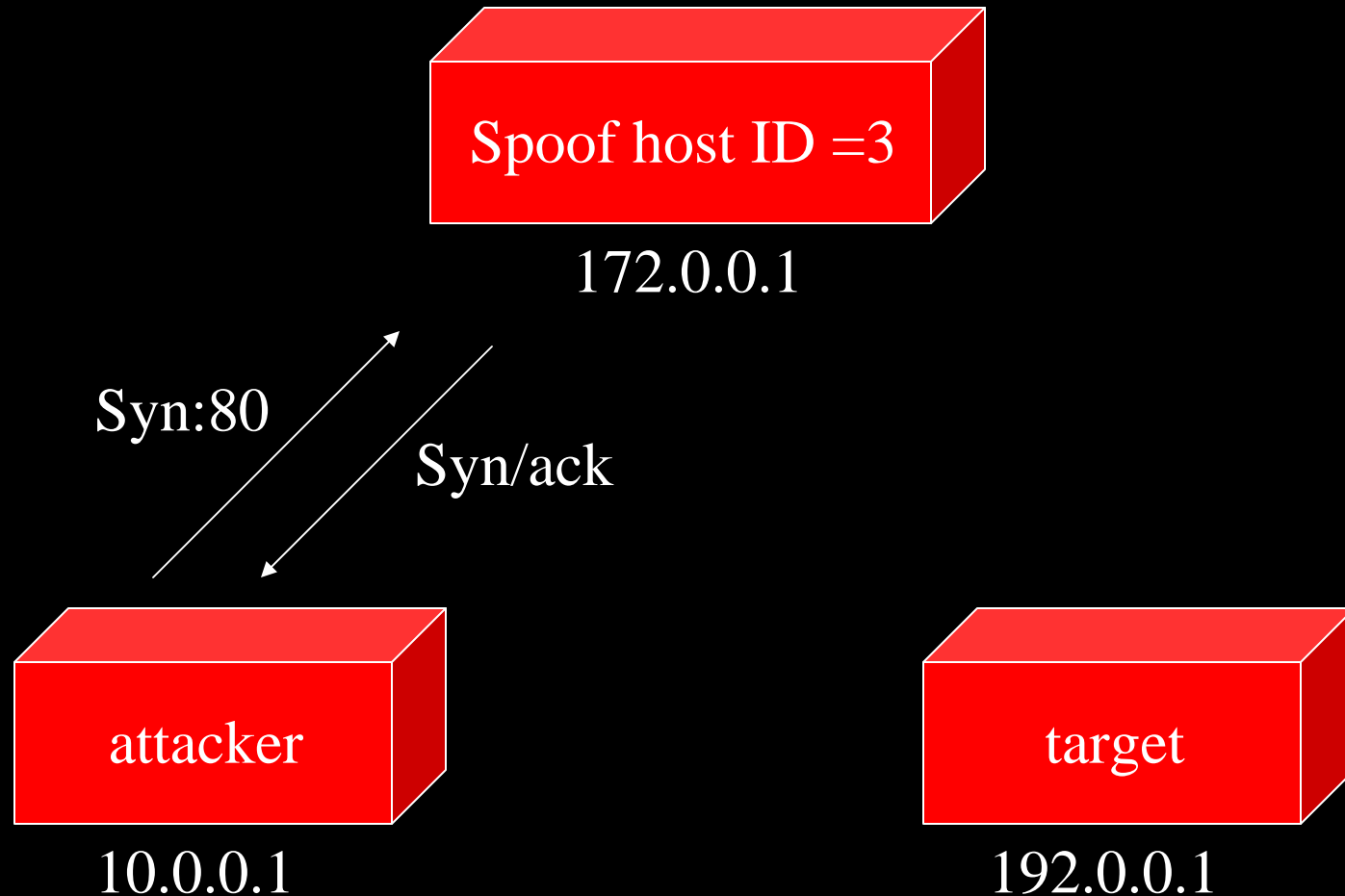
Phase 3 (fooling the respons)



Phase 4 (probing the spoof host)



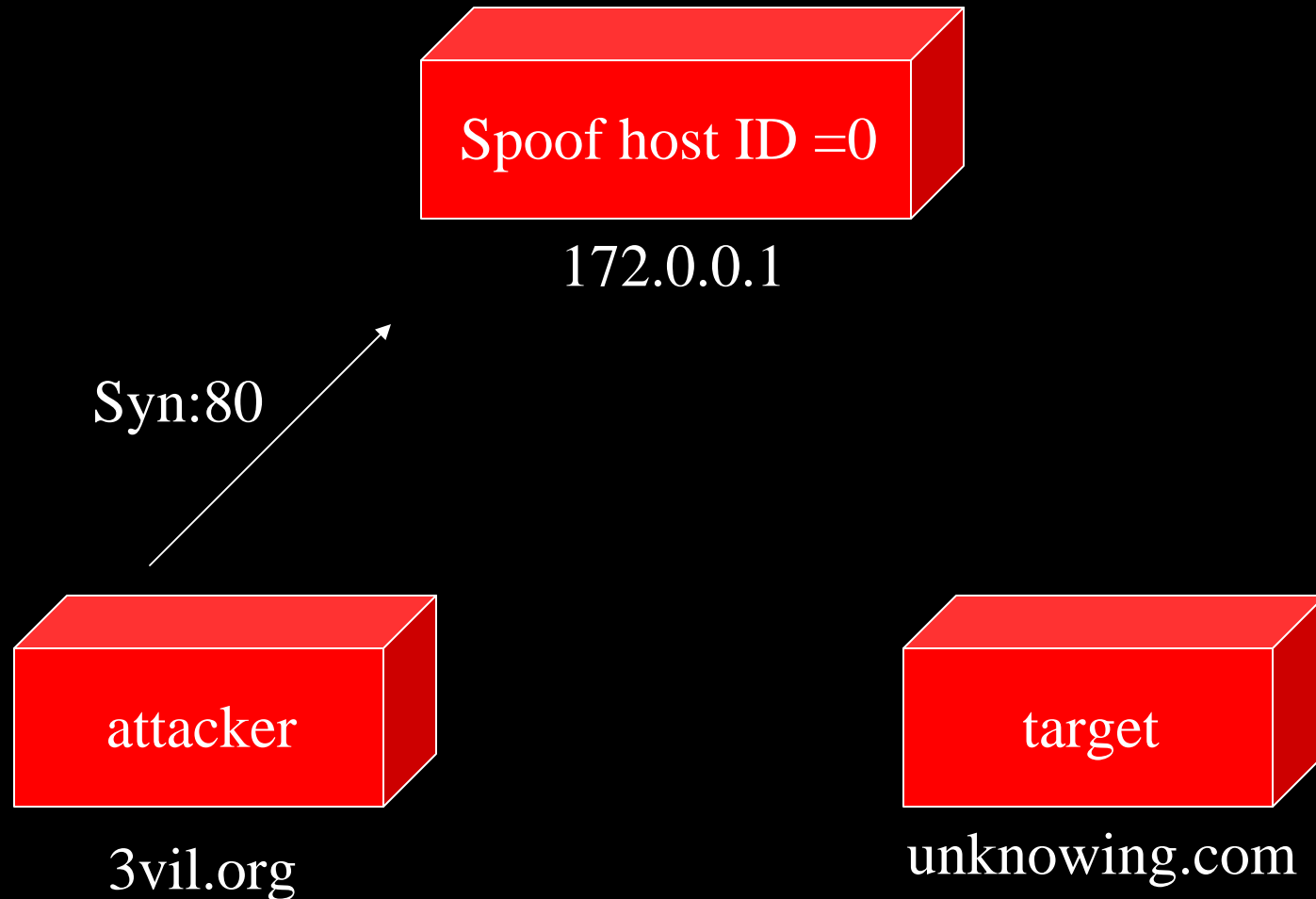
Phase 4 (probing the spoof host)



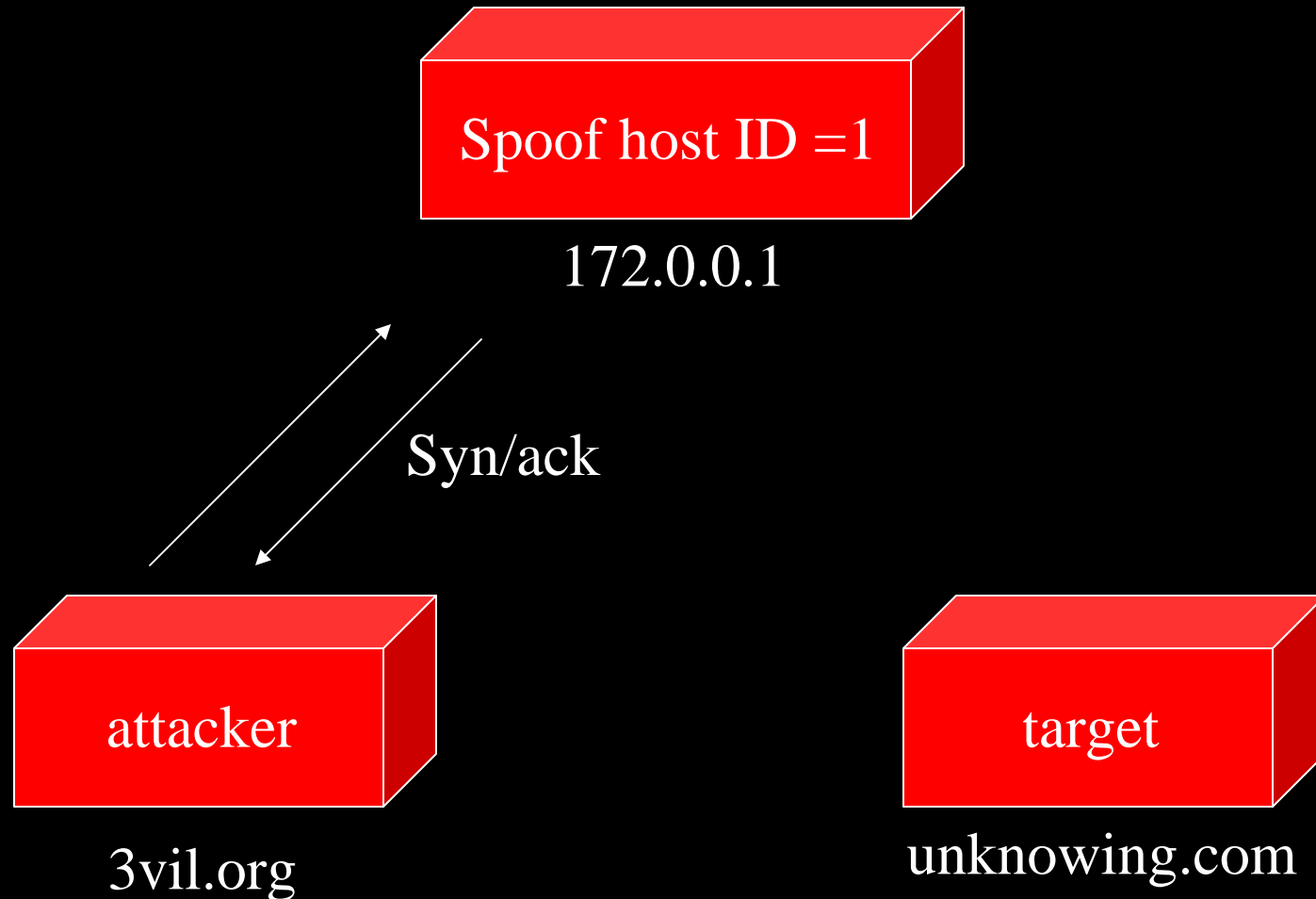
Case port closed

Adding the ID counters

Phase one (sync the id# of spoof)



Phase one (sync the id# of spoof)



Phase2 (spoofing the source)



172.0.0.1

Syn src = 172.0.0.1 Dst = 192.0.0.1

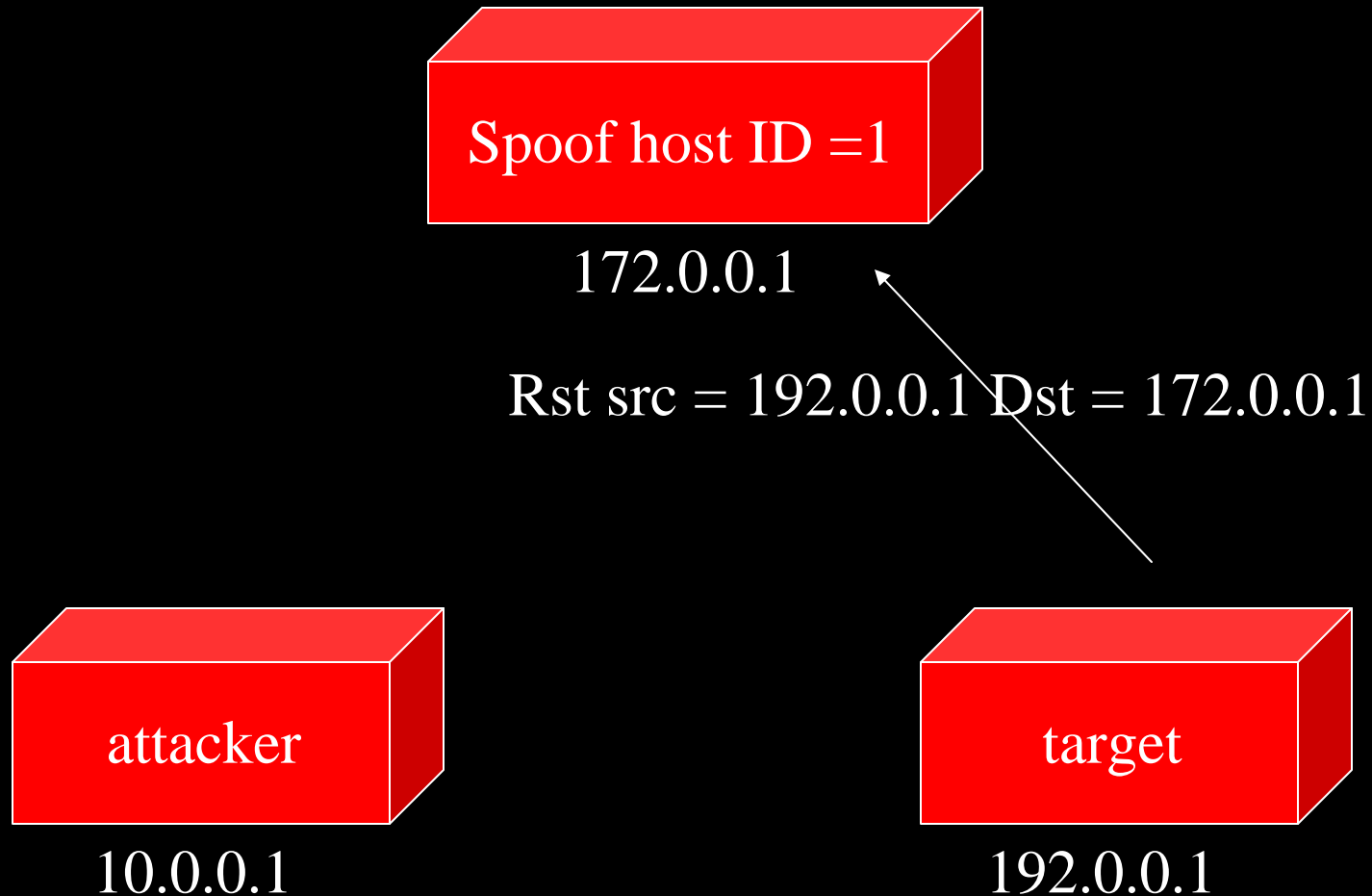


10.0.0.1

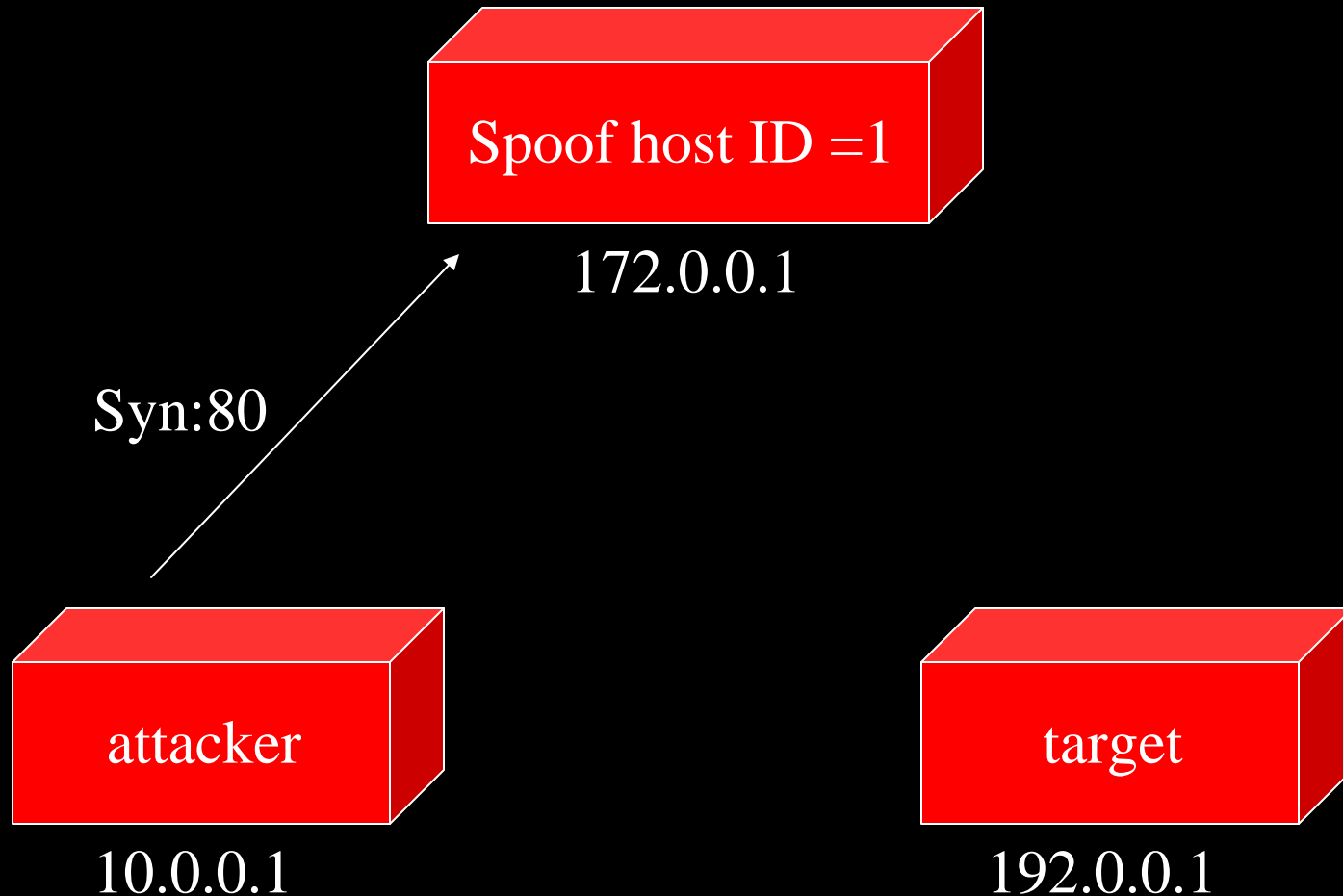


192.0.0.1

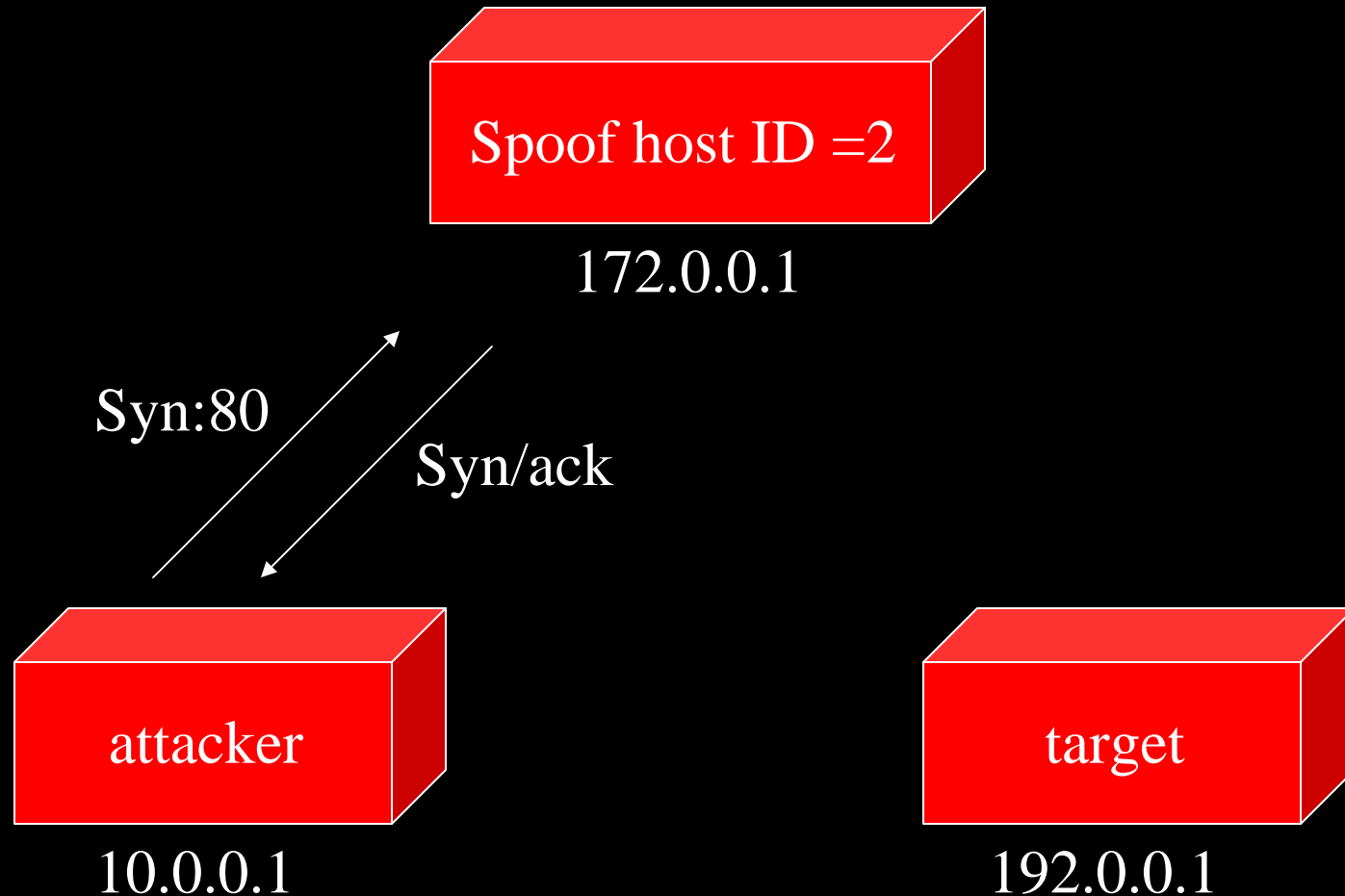
Phase 3 (fooling the respons)



Phase 4 (probing the spoof host)



Phase 4 (probing the spoof host)



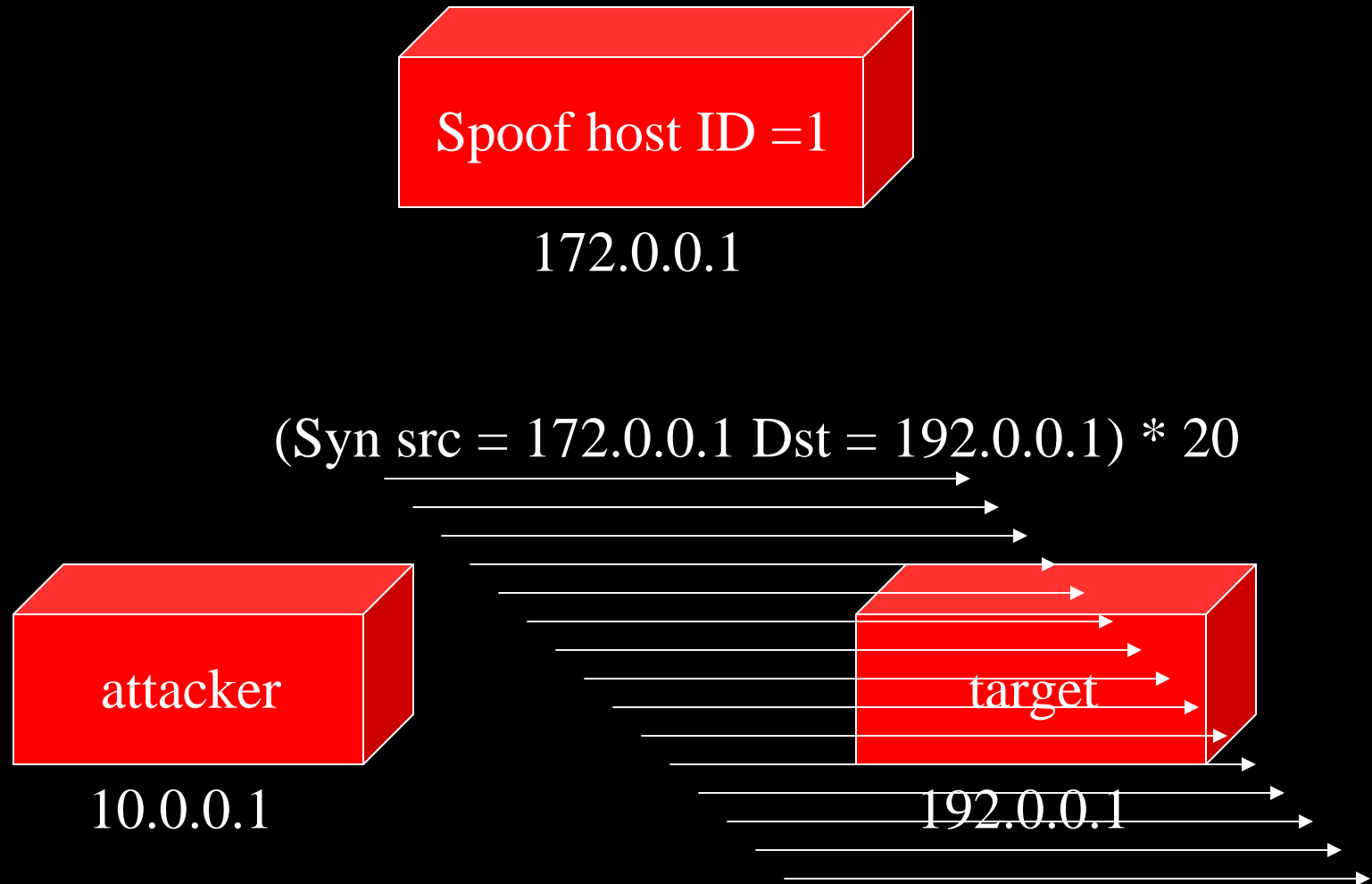
Summary

- By constantly polling a decoy host for id number increments we can see If the scanned target host has sent it syn/ack or reset packets.
- By analysing this we will know whether a port on the scanned host is open or not
- This is done totally blind from the scanned host.

The basic technique and its flaws

- If the poll host is active it will increase the id# for each connection.
- This will result in false positives.
- These false positives can be minimized by sending multiple packets for each port.
- Then calculating the increase
- The port will only show up true if the increase is $> (\#packets_sent * 255) / 2$

Phase2 (spoofing the source)



Phase 3 (fooling the respons)

