(kernel)(Hacking)

if i really know, i can hack

Kernel x Segurança

Rodrigo Rubira Branco

rodrigo@kernelhacking.com
bsdaemon@bsdaemon.org

OBJETIVOS

- Demonstrar recursos existentes no kernel do Linux
- Explicar estruturas internas do kernel do Linux
- Explicitar possíveis melhorias de seguranca para o kernel
- Exemplificar recursos de segurança e de ataques
- Exemplificar recursos de módulos / injeção de código em kernel

CIDAL

C onfidencialidade
I ntegridade
D isponibilidade
A utenticidade
L egalidade

LEMBRAR-SE SEMPRE DISTO NO DECORRER DA AULA!

Vida Real

Trabalhamos em um mundo real de sistemas mal configurados:

**** Bugs de Software***
**** Empregados Insatisfeitos***
**** Administradores de Sistemas Sobrecarregados***
**** Acomodação de necessidades empresariais***
*** Falta de educação em segurança**
**** B2B,B2C,B2E,C2C,X2X?***
**** Tempo disponível para desenvolvimento***
**** Linguagens de 3 e 4 gerações***

Camadas

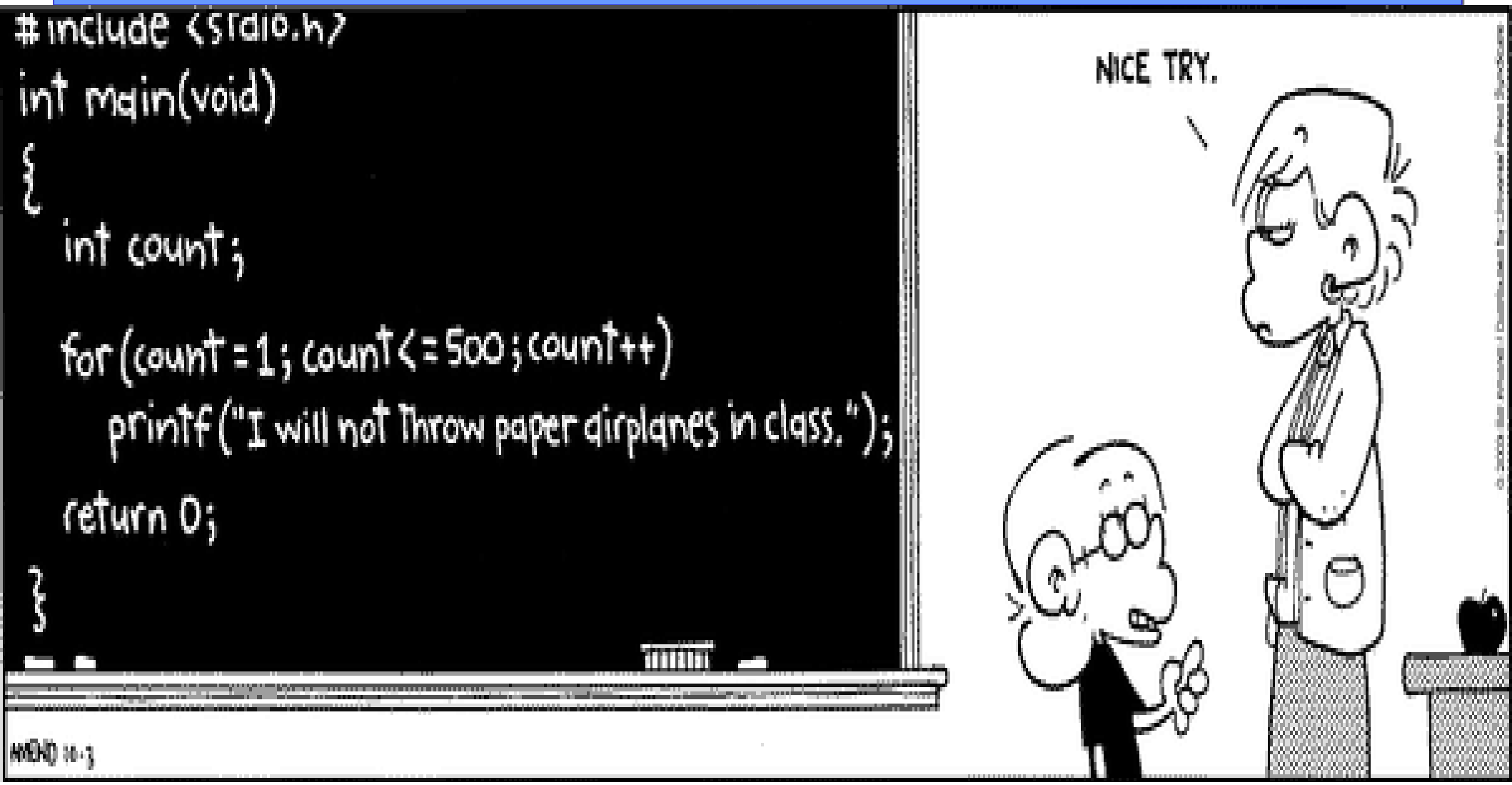
- Softwares sempre terão falhas
- Falhas inerentes ao sistema operacional, se não previstas pelo software causam problemas (vide race condition do /tmp)
- Fortificações possiveís no sistema operacional, poderiam tornar as aplicações "livre de falhas" (vide Win2003, OpenBSD e patchs para o kernel do Linux)

Quando pensar em Segurança pense em uma CEBOLA.

Problemas com a Segurança Atual

- Evitar / Identificar / Corrigir
- O estado atual da segurança é uma corrida sem fim
- Ciclo sem fim de descobrimento/correção de vulnerabilidades

"O" Problema


#include <stdio.h>
int main(void)
{
int count;
for (count = 1; count <= 500; count++)
printf ("I will not Throw paper airplanes in class.");
return 0;
}
NICE TRY.

Seguro por “default” (*)

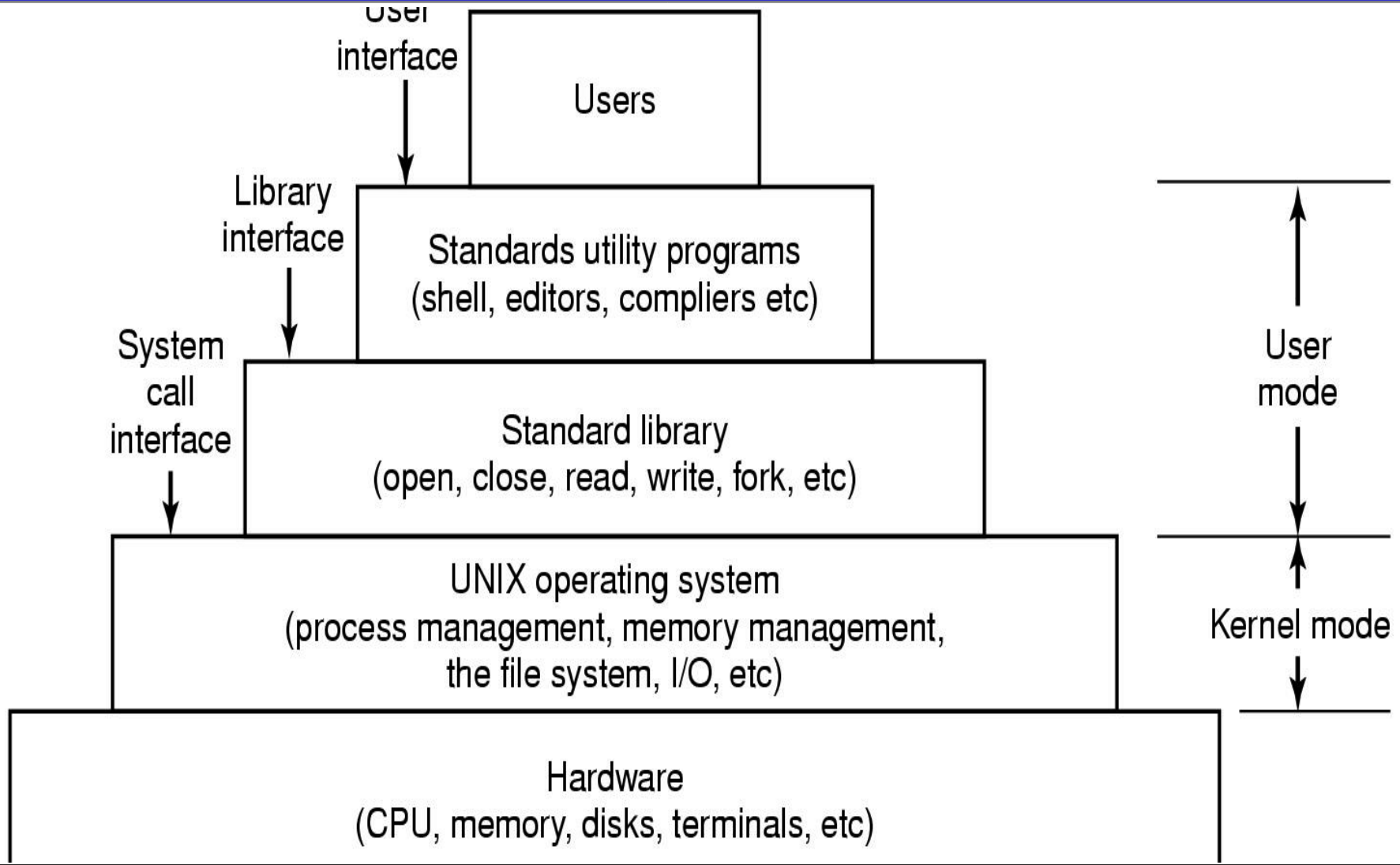
- **A função deve ser segura por si própria, e não contar com “testes” antes de sua chamada**
  - Ex: funcao insegura: strcpy()

- **char teste[10], teste2[12];**
- **if (! (strlen(teste2) > strlen(teste)))**
  - **strcpy(teste,teste2);**

Function Hook (*)

- **Para testarmos códigos prontos, podemos utilizar hook de funcoes**
- **Consiste em interceptarmos a chamada da funcao feita pelo nosso software e passarmos novos argumentos**

GNU/Linux


User interface
Users
Library interface
Standards utility programs
(shell, editors, compliers etc)
System call interface
Standard library
(open, close, read, write, fork, etc)
UNIX operating system
(process management, memory management,
the file system, I/O, etc)
Hardware
(CPU, memory, disks, terminals, etc)
User mode
Kernel mode

STDLIB (Standard Library)

- Chamadas de sistema são os métodos utilizados por programas de usuário para pedirem ações do sistema operacional
- Esta chamada é efetuada através de um trap:
  - Mudança do modo usuário para modo kernel
  - Controle dos parâmetros corretos
  - Execução efetuada pelo sistema operacional
  - Retorno ao modo usuário
- Como é impossível executar um trap em C, este é provido pela standard lib através de funções para cada system call. Estas funções são escritas em ASM e então chamadas a partir do C.
- A stdlib define um conjunto de funções através das quais as aplicações podem interagir com o kernel, e implementa muitas funcionalidades do sistema operacional que não necessitam da interação do kernel
- O padrão POSIX estabelece quais são as funções da biblioteca que o sistema necessita providenciar, seus parâmetros e tarefas.

Kernel

- Responsável por manter as importantes abstrações do sistema operacional
- O kernel provê as principais funções da “máquina abstrata” (system calls, interrupções e traps).

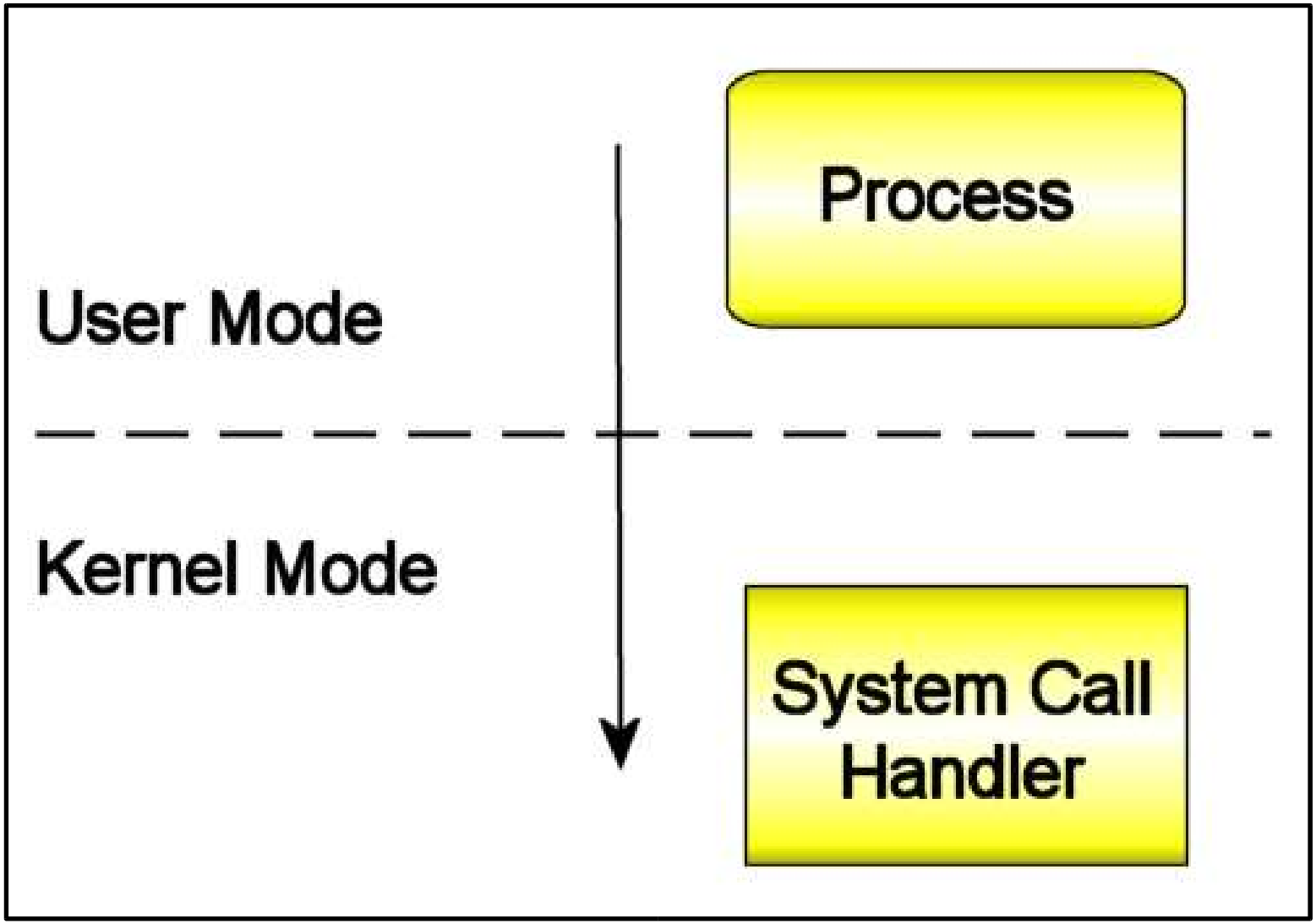
Estrutura Genérica de um Kernel Unix

<table>
<tr><td colspan="5">System calls</td><td colspan="3">Interrupts and traps</td></tr>
<tr><td colspan="2">Terminal handing</td><td>Sockets</td><td>File naming</td><td>Map-ping</td><td>Page faults</td><td rowspan="2">Signal handling</td><td rowspan="2">Process creation and termination</td></tr>
<tr><td rowspan="2">Raw tty</td><td>Cooked tty</td><td>Network protocols</td><td>File systems</td><td colspan="2">Virtual memory</td></tr>
<tr><td>Line disciplines</td><td>Routing</td><td>Buffer cache</td><td colspan="2">Page cache</td><td colspan="2">Process scheduling</td></tr>
<tr><td colspan="2">Character devices</td><td>Network device drivers</td><td colspan="2">Disk device drivers</td><td colspan="2">Process dispatching</td></tr>
<tr><td colspan="8">Hardware</td></tr>
</table>

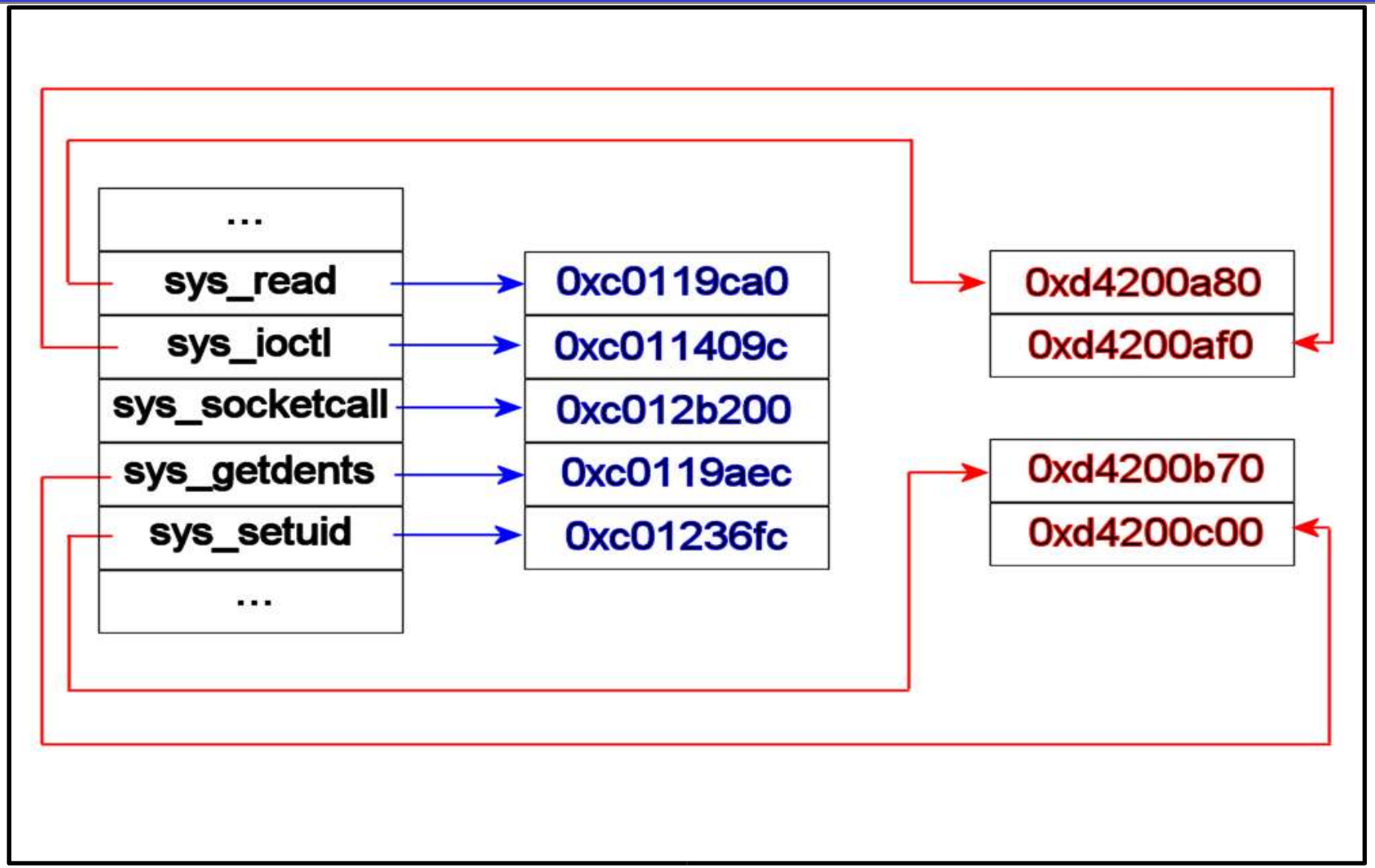
Kernel

- O Kernel do Linux possui um modelo monolítico e não o "novo" modelo cliente-servidor (ex: micro-kernels, como minix). Isto por motivos de performance.
  - O kernel é executado em modo kernel, com total controle aos recursos físicos do computador.
  - Todo o código do kernel e os dados são mantidos em um único espaço de endereços.
- Embora o kernel execute como um simples processo em um único espaço de endereçamento, este possui modularidade.

Syscalls (chamadas ao sistema)



Syscalls (chamadas ao sistema)



Syscalls (chamadas ao sistema)

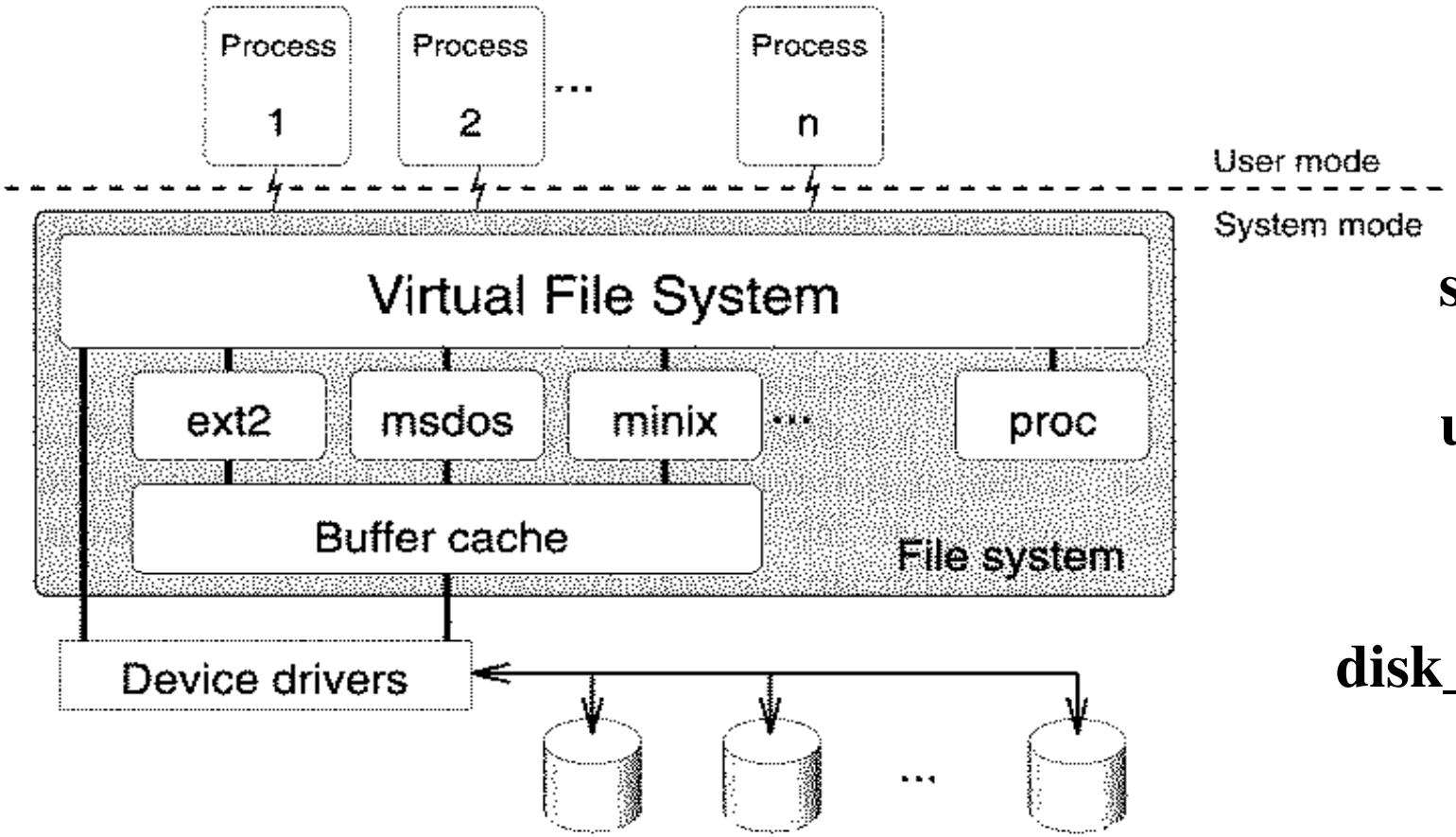
```
extern void *sys_call_table[];
int (*o_getdents) (uint, struct dirent *, uint);

o_getdents=sys_call_table[SYS_getdents];
sys_call_table[SYS_getdents]=(void*)h_getdents;
```

Syscalls (chamadas ao sistema) (*)

```
void *hacked_sys_call_table;
hacked_sys_call_table=kmalloc(256*sizeof(long int),
    GFP_KERNEL);
memcpy(hacked_sys_call_table, sys_call_table, 256*sizeof(long
    int));

(int)*((int*)ptr) =(int) hacked_sys_call_table;
```

- PTR aponta para o endereço original da tabela (chamada pela função system_call)

VFS (Virtual File System)



read()

sys_read()

ufs_read()

disk_dev_read()

VFS – Exemplo de Hooks (*)

- Interceptando sem o uso das syscalls

old_readdir_root = proc_root.FILE_OPS->readdir;

old_lookup_root = proc_root.INODE_OPS->lookup;

proc_root.FILE_OPS->readdir = &new_readdir_root;

proc_root.INODE_OPS->lookup = &new_lookup_root;

old_load_binary=elf_format->load_binary;

_elf_format->load_binary = &h_load_binary;

VFS Security

- Mecanismos de segurança impostos pelo VFS e alguns file systems:
  - Mount flags (super_block→s_flags)
  - File attributes (inode→i_mode)
  - Extended attributes (Kernel 2.6)
  - File ownership (inode→u_uid)
  - Restrições adicionais aos modelos existentes:
    - Atributos extras ext2, ext3
    - Append-Only, Immutable

VFS - Fraquezas

Um atacante que ganhe acesso root, possui completo controle sobre o sistema de arquivos

- Apagar entradas em logs
  - Ext2 append-only flag não impede o usuário root
- Insmod
  - Checagens de segurança do VFS/sistema de arquivos podem ser passadas
    - Interceptando-se chamadas de sistema ao VFS
    - Substituindo-se operações nas tabelas linkadas na criação de arquivos/inodes
- Comunicando-se diretamente com o device driver (/dev)

Stack Overflow

- Falha muito comum (provavelmente a mais comum entre todas e a mais facil de ser explorada)

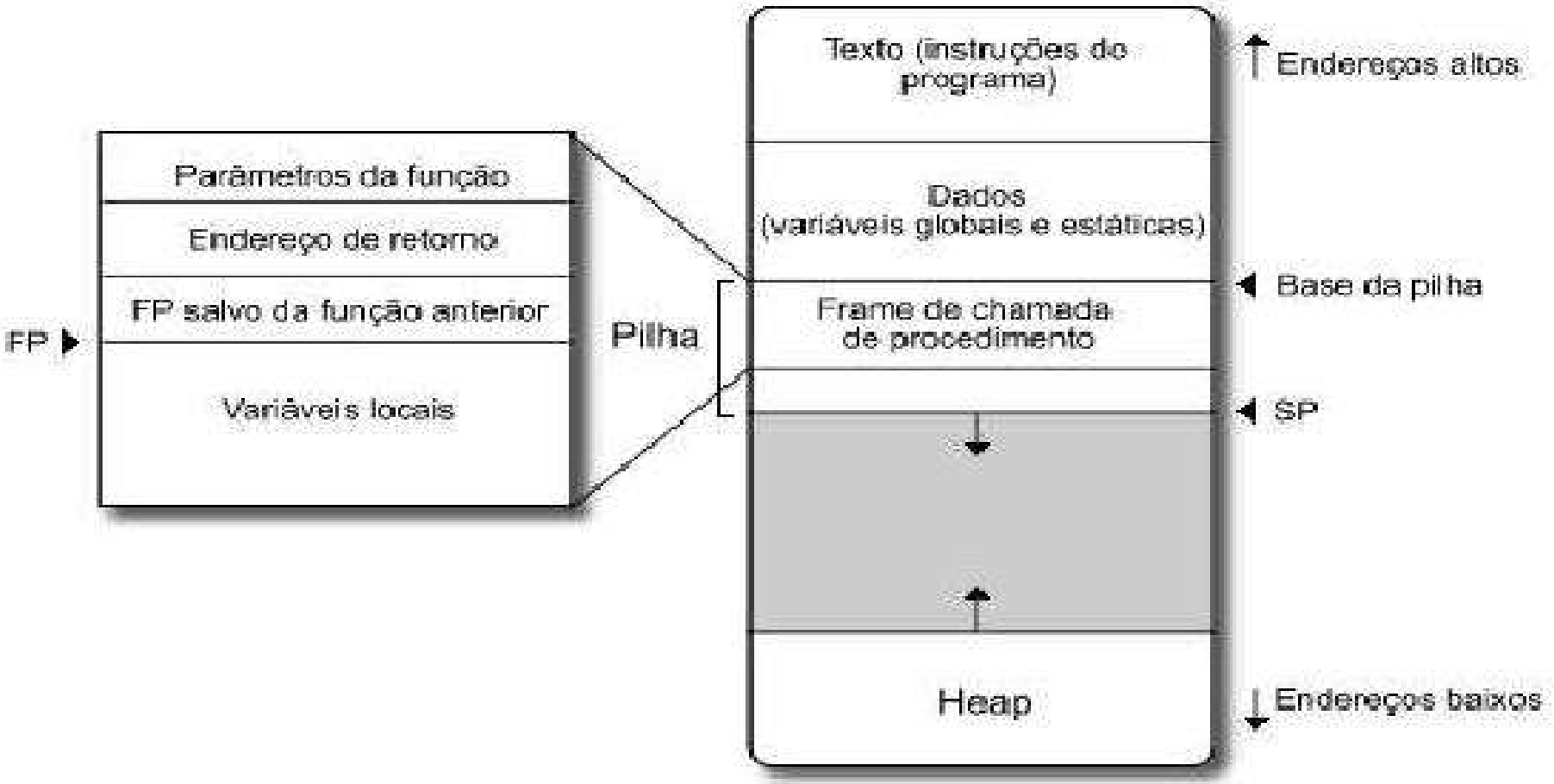
- Existe grande dificuldade de detecçao por tratar-se de interaçoes e loops, uso de funçoes do proprio usuario

- Provavelmente poderia ser detectada atraves de analise de funçoes perigosas e como estao sendo usadas, dando-se grande numero de FALSOS NEGATIVOS

- Consiste da alocaçao de uma variavel com tamanho fixo
e da tentativa de armazenar-se nela mais dados do que sua declaraçao continha (geralmente dado pelo usuario).

- Sobrescreve-se na pilha o RET, que quando a funçao retornar sobrescreve o registrador EIP e permite execuçao de codigo arbitrario.

Stack Overflow



Integer Overflow (*)

- Enquanto os programadores e auditores de codigo cada vez mais se atentam para falhas de Stack/Heap Overflow, uma nova modalidade surge

- Integer overflow nada mais e do que a tentativa de armazenamento de valores maiores do que os possiveis em variaveis inteiras (short, long, etc)

- Devido a um integer overflow, pode-se passar por testes de condiçoes e conseguir-se acarretar situaçoes de Stack/Heap overflows mesmo onde nao existiam, dai a dificuldade em detecçao automatizada deste tipo de procedimento

Race Condition (*)

- Uma das falhas mais dificeis de se detectar em auditorias manuais de codigo e em auditorias automaticas de fonte

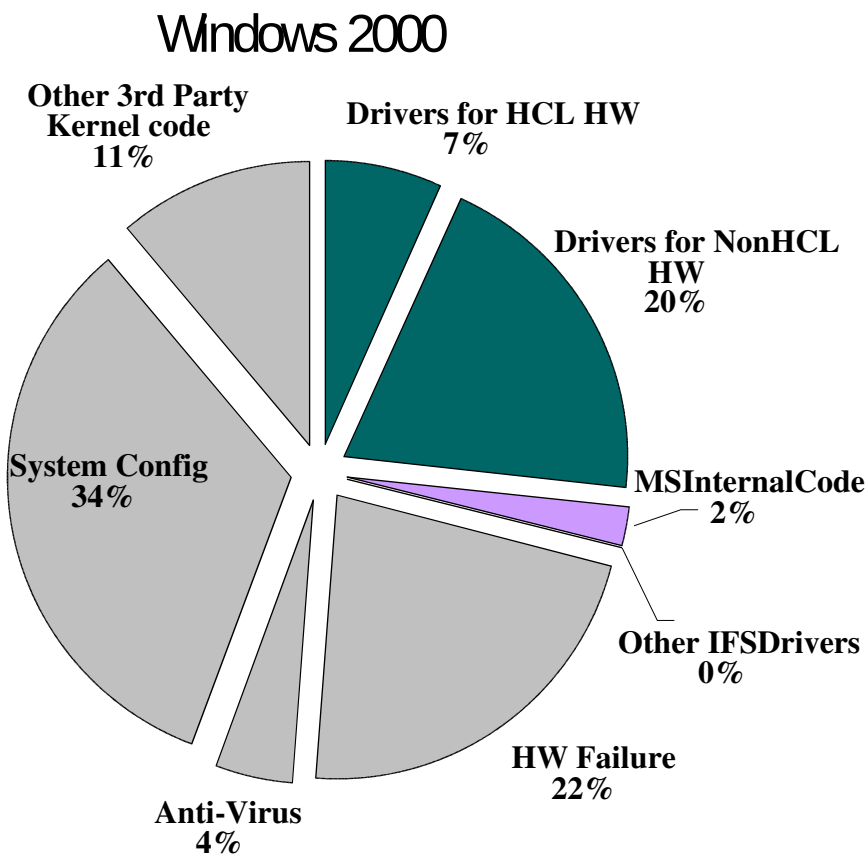
- Geralmente pode ser descoberta automaticamente atraves do uso do FUZZER, por acarretar a situaçao de condiçao de corrida com o stress do software

- Consiste em se explorar condiçoes de acessos a recursos compartilhados, onde geralmente o resultado obtido consiste em execuçao de codigos com privilegios elevados (escalaçao de privilegios)

Drivers de Dispositivos e Problemas de Segurança

- Windows 2000: #1 em bugs de kernel reportados [Murphy '00]

➢ Os drivers não são controlados pelos Sistemas Operacionais, afetando e muito a segurança dos mesmos



Source: Brendan Murphy, Sample from PSS Incidents

Drivers de Dispositivos e Problemas de Segurança

- Podemos aumentar a segurança do SO através de “tolerância” de erros de drivers de dispositivos
- Necessita manter-se compatibilidade com os drivers existentes

➢ Possível Solução: Isolar os device drivers em uma “sandbox”, mantendo as APIs existentes

Drivers de Dispositivos e Problemas de Segurança

- Performance do isolamento depende:
  - Nível do isolamento requerido
  - Custo das comunicações entre isolamentos
  - Custo das movimentações de dados entre os isolamentos
  - Custo da execução do código isolado

➢ Problema: Necessita-se conhecimento dos drivers antes de se isolar os mesmos.

Drivers de Dispositivos e Problemas de Segurança

- Diferenças ao lidarmos com drivers:
  - Já existem!
  - Lidam com quantidade absurda de dados
  - Possuem apelas recursos limitados de aplicação

Drivers de Dispositivos e Problemas de Segurança

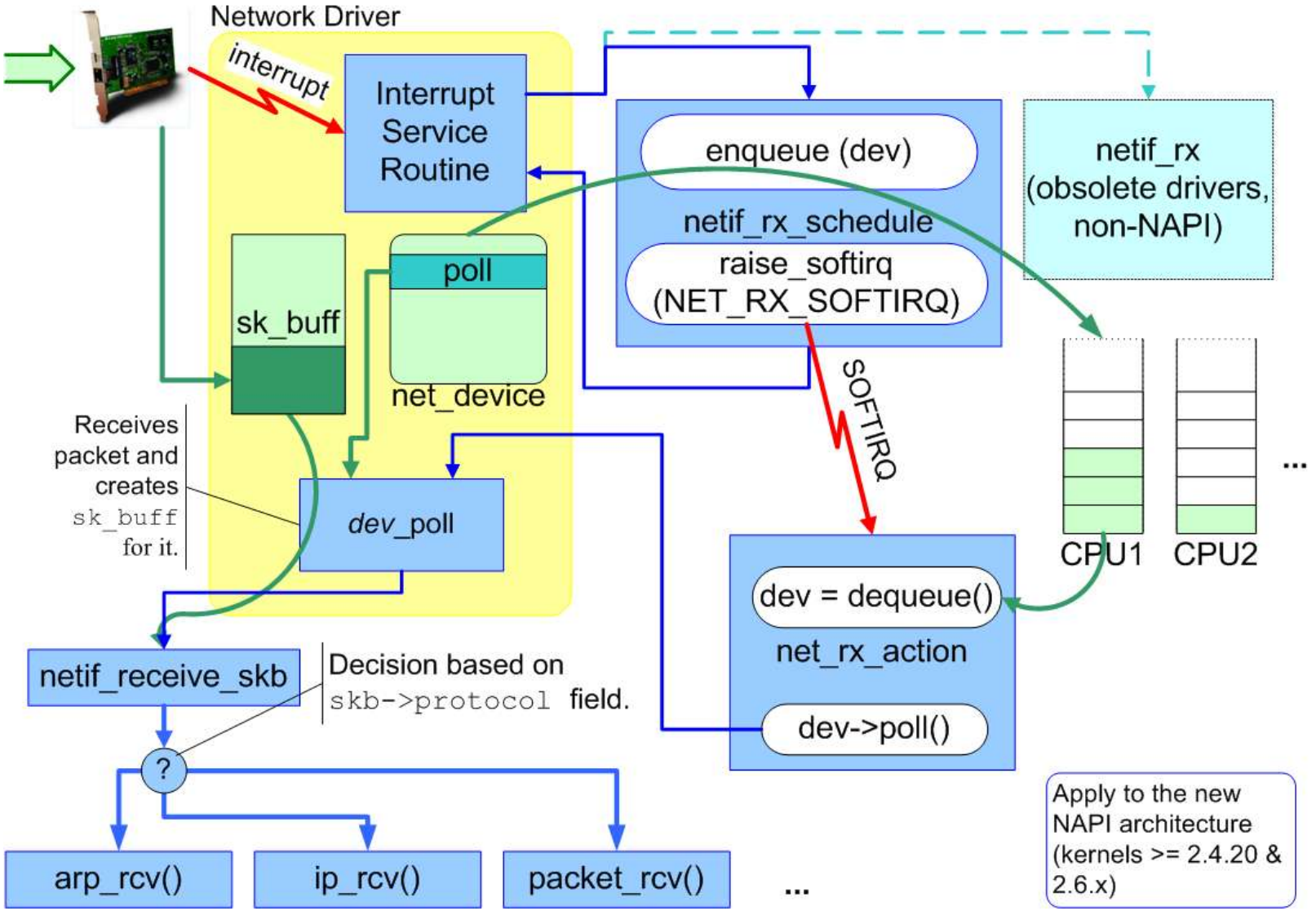
- A maioria dos problemas em drivers são derivados de: [Chou '01, Linux kernel Bugzilla]
  - Acesso ilegal a memória
  - Uso inválido de locks
  - Mantimento de interrupções desabilitadas

- Estes problemas podem ser detectados verificando-se os acessos a memória e as condições anteriores/posteriores na execução do driver.

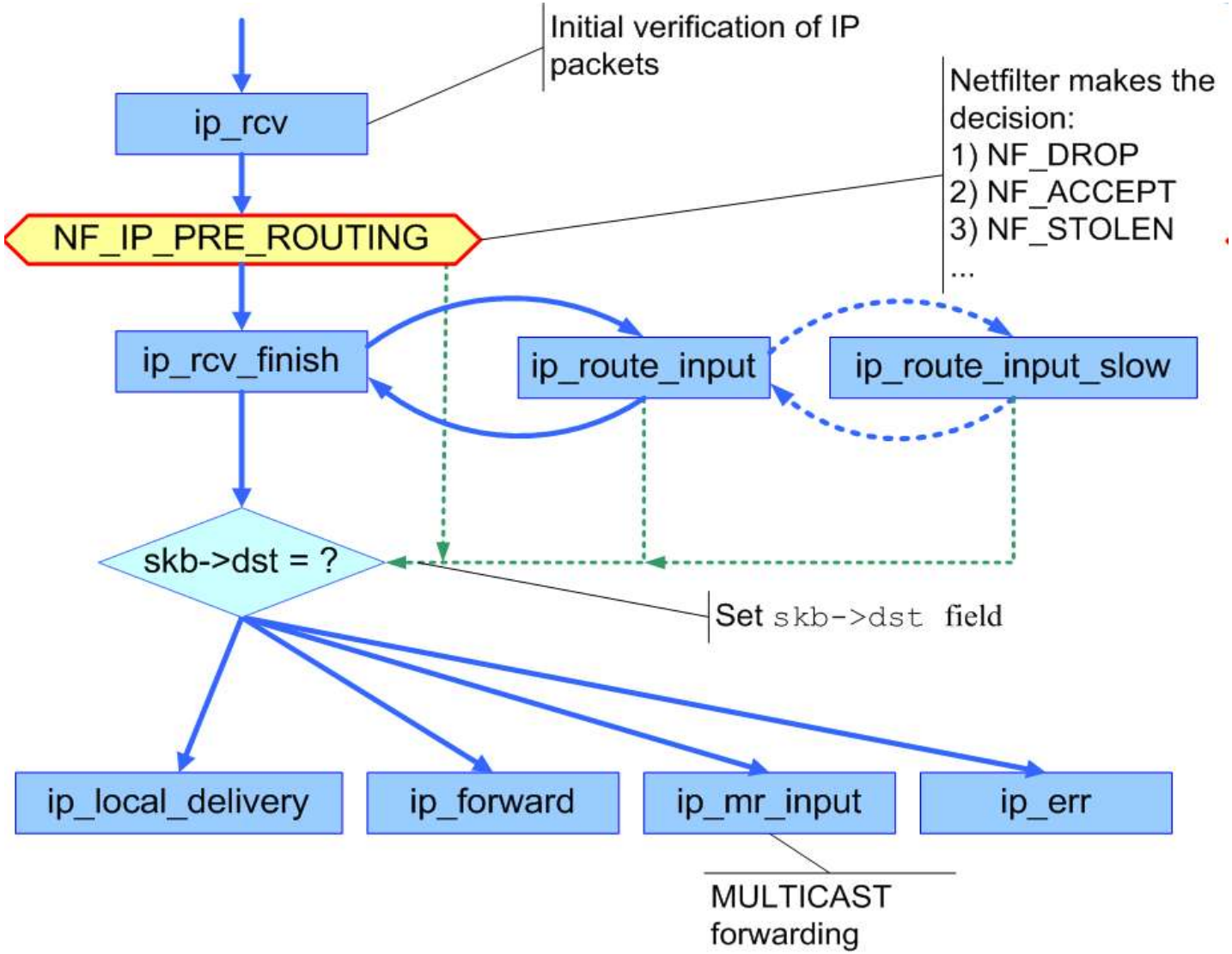
Drivers de Dispositivos e Problemas de Segurança

- Kernel já suporta inicialização/paralização de drivers dinamicamente
- Devido ao grande número de interfaces driver/kernel no Linux, temos muitas oportunidades de otimização:
  - Muitos parâmetros read-only
  - Maioria das funções são de inicialização/limpeza

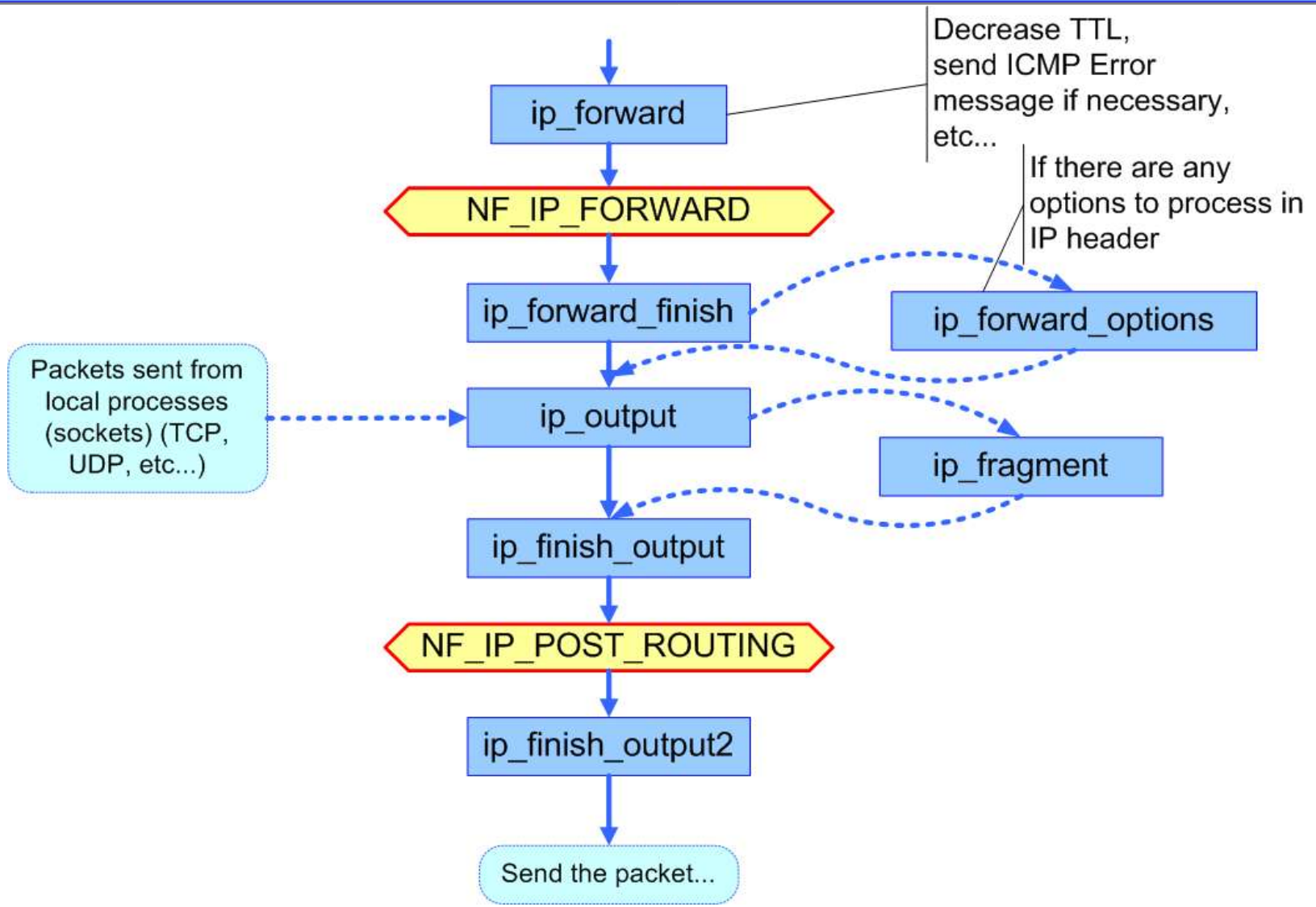
Network Hacks



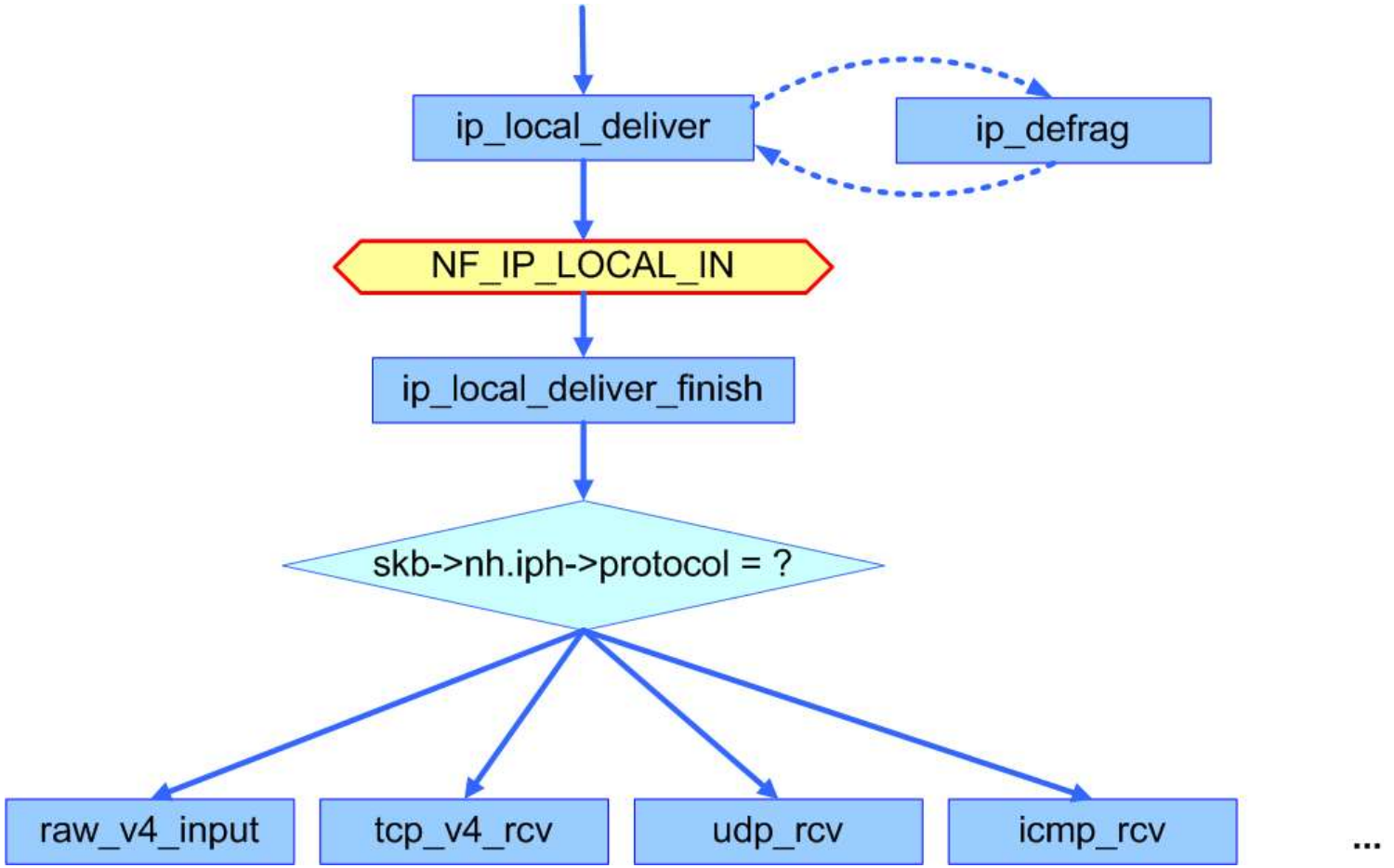
Network Hacks



Network Hacks



Network Hacks



Técnicas de Hook de Rede

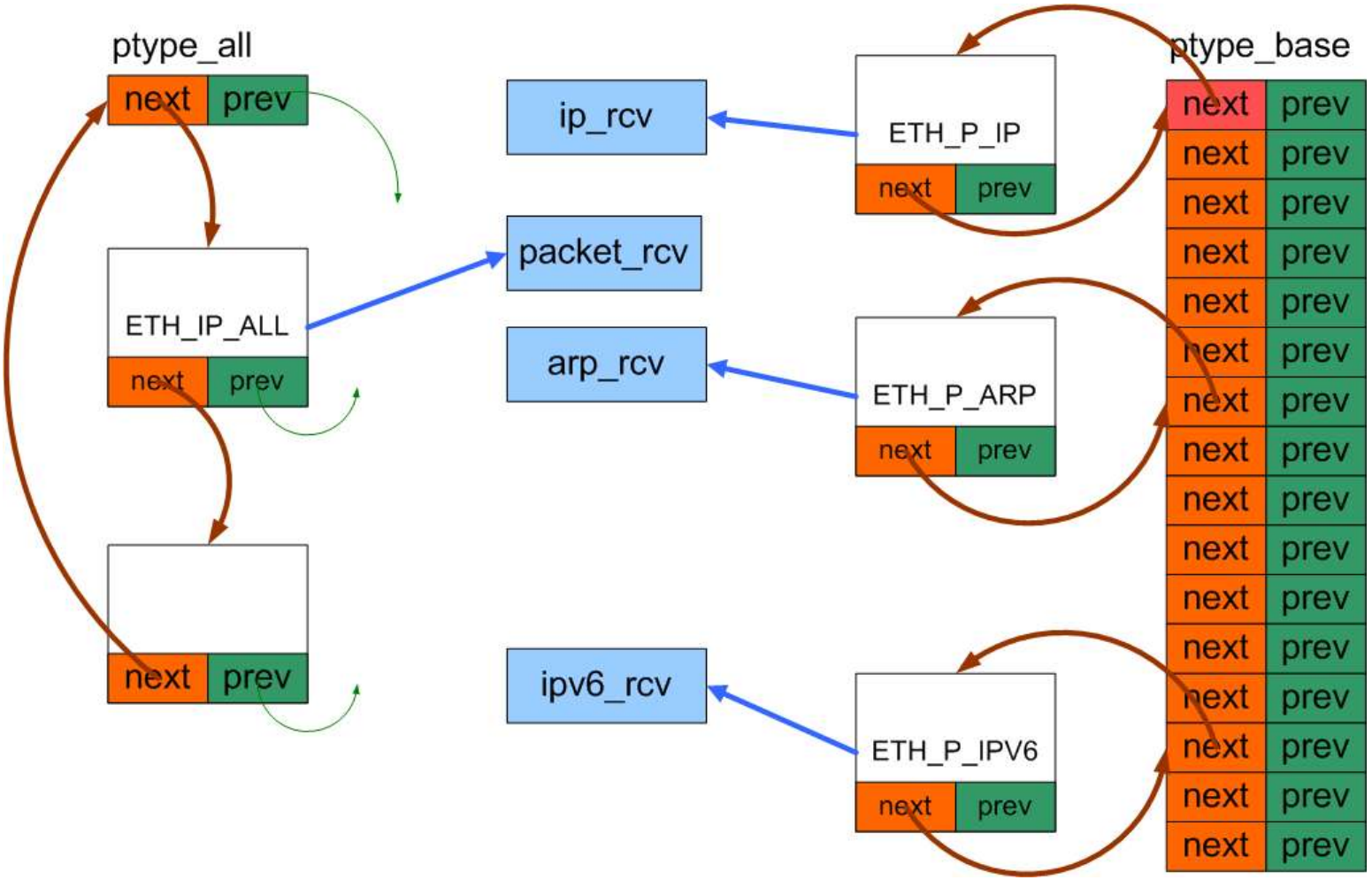
`ptype_*` handlers

Adicionar novo handler de protocolo, que pode reconhecer determinados pacotes ou modificar conteúdo de pacotes on-the-fly

Netfilter hooks

Adicionar novos hooks do NF, efetuando o mesmo que os handlers. Esta técnica também pode passar pelas regras locais de Firewall (ipfilter em Linux é implementado como um hook NF)

Protocol Handlers



Estrutura Utilizada: packet_type

```
struct packet_type
{
 unsigned short    type;
 struct net_device *dev;
 int (*func) (...);
 void *data;
 struct list_head list;
};
```

Funções exportadas pelo kernel para adicionar/remover handlers:

- `void dev_add_pack(struct packet_type *pt)`
- `void dev_remove_pack(struct packet_type *pt)`

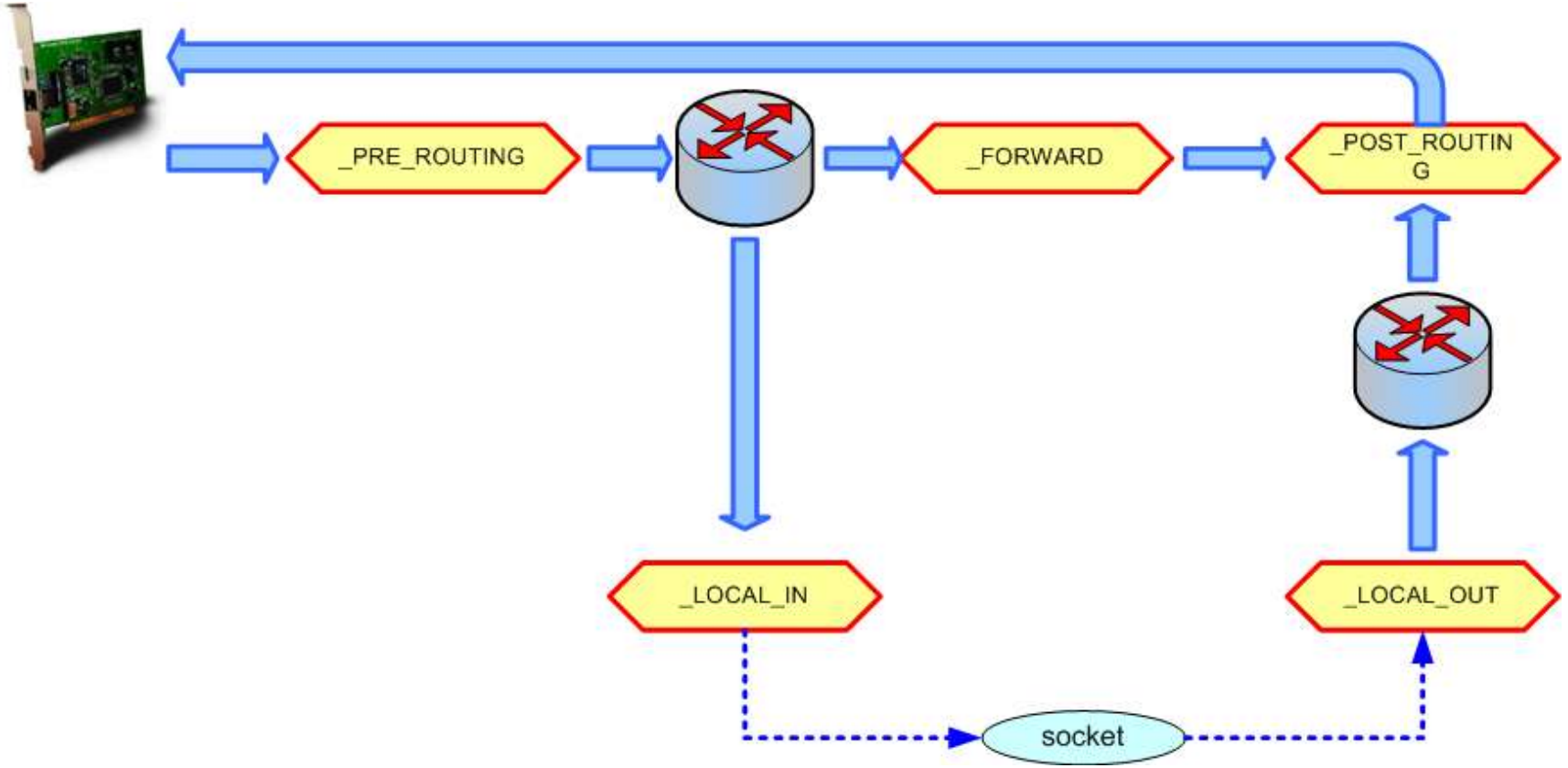
Adição do Handler (*)

```
struct packet_type myproto;

myproto.type = htons(ETH_P_ALL);
myproto.func = myfunc;
myproto.dev  = NULL;
myproto.data = NULL;

dev_add_pack (&myproto)
```

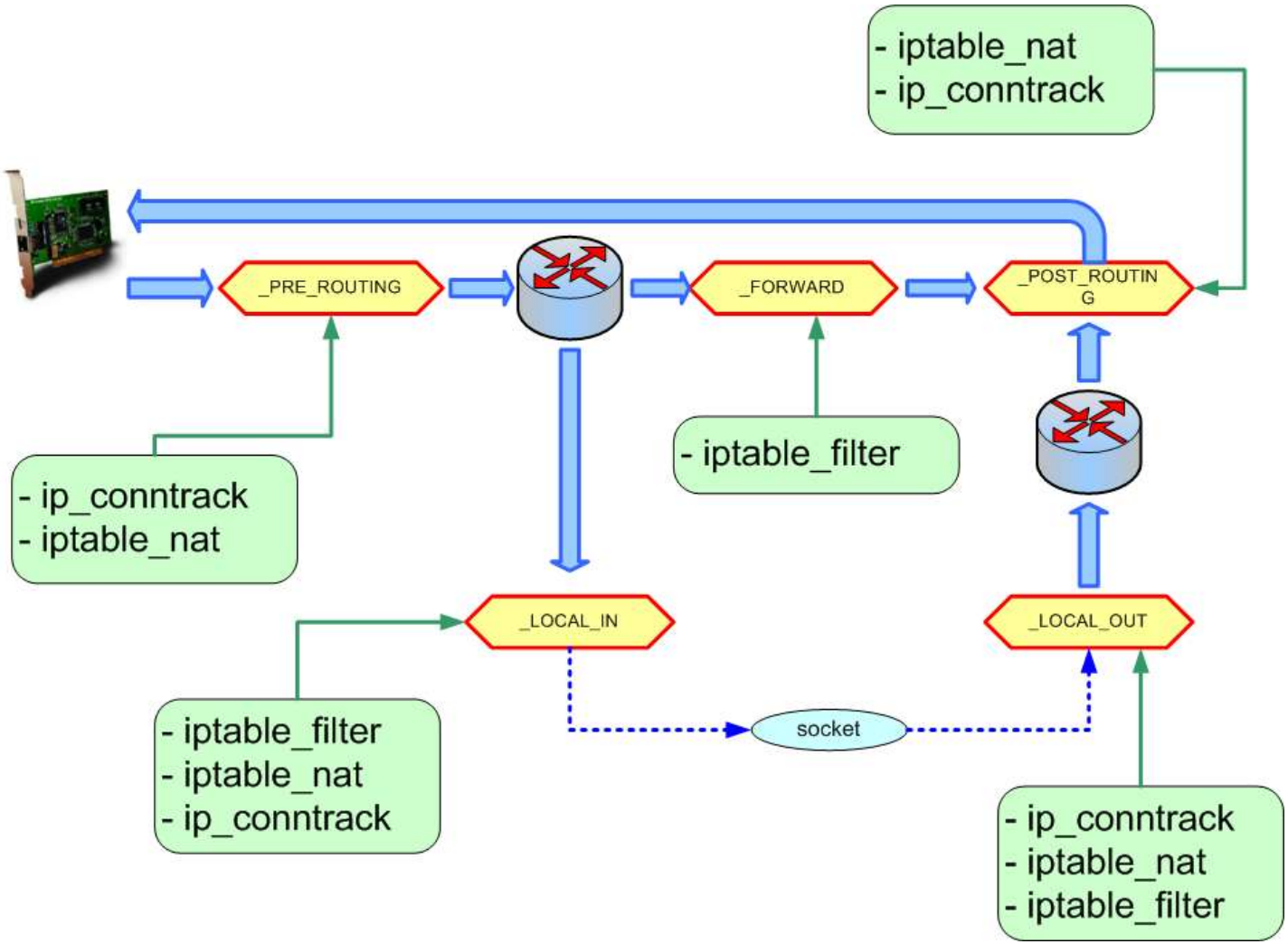
Possível detectar?

- Listando todos os handlers de protocolo suportados?
- Checagem manual das listas:
  - `ptype_all`
  - `ptype_base`
- Não são exportadas pelo kernel (mesmo método que usado para exportar sys_call_table em kernel 2.6)

Estrutura do Netfilter



Estrutura do Netfilter



Estrutura Utilizada: nf_hook_ops (*)

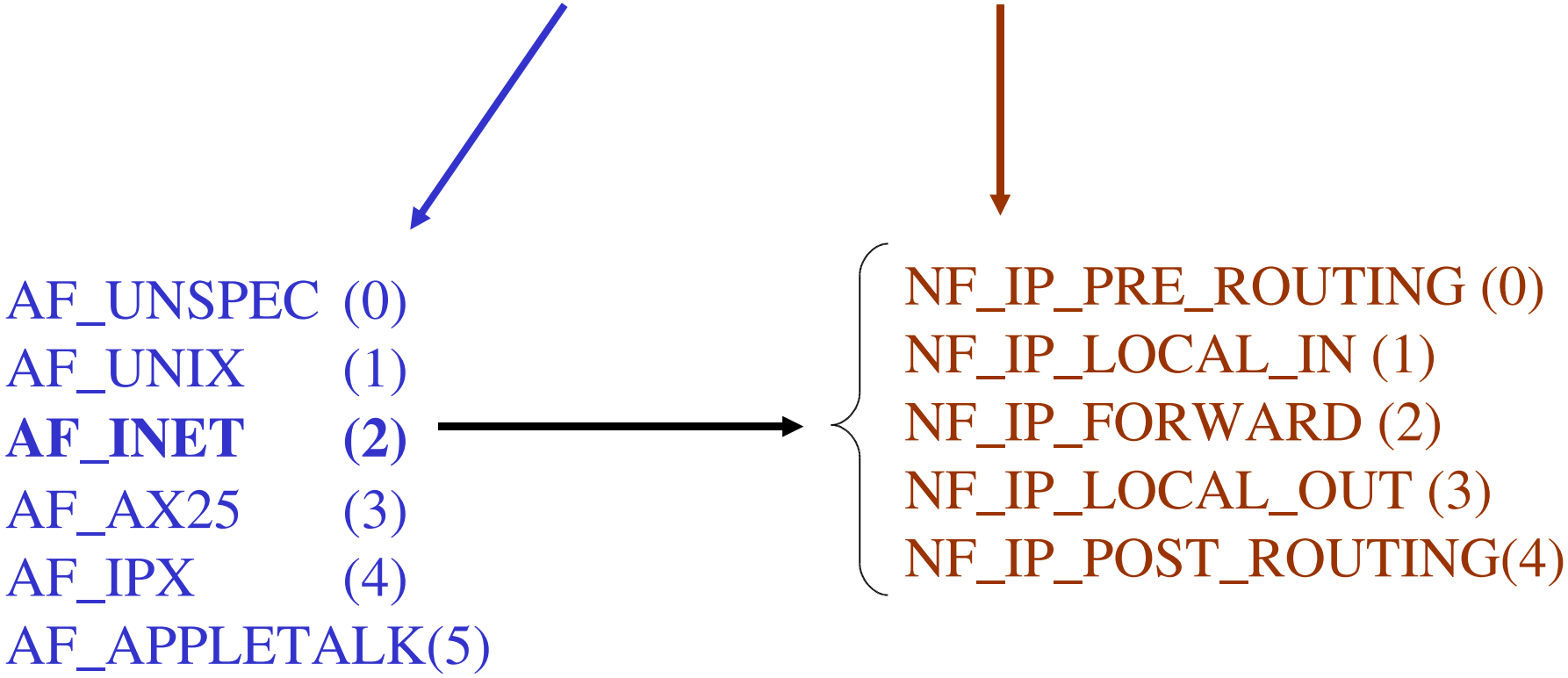
```
struct nf_hook_ops
{
  struct list_head list;
  nf_hookfn *hook; ← handler address
  int pf; ← family (AF_INET, AF_INET6, AF_IPX, ...)
  int hooknum; ← (NF_IP_*, NF_IP6_*, NF_IPX_*, ...)
  int priority;
};
```

Funções exportadas pelo kernel para adicionar hooks NF:

- `int nf_register_hook(struct nf_hook_ops *reg);`
- `void nf_unregister_hook(struct nf_hook_ops *reg);`

Listas de Hooks NF Registradas

```
struct list_head
     nf_hooks[NPROTO][NF_MAX_HOOKS];
```



Possível detectar?

- O endereço de nf_hooks é exportado
- Sabendo tal endereço, apenas precisamos verificar toda a lista e encontrar hooks suspeitos
- Lê-se suspeito como, por exemplo:
  - Hooks que não pertençam a módulos existentes
    - Módulos que foram escondidos
    - Módulos que não “limparam” seus hooks ao serem descarregados
    - Código injetado via /dev/(k)mem

Exemplos Finais (*)

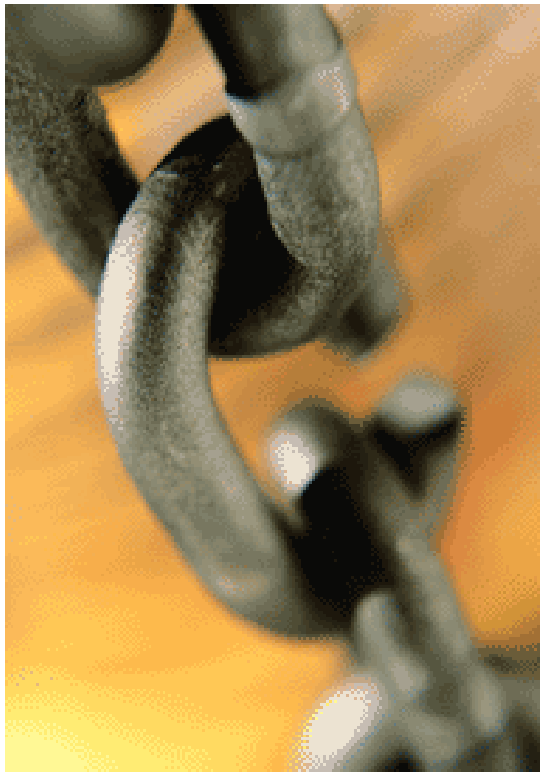
- Modificacao de modulos
- Interceptacao de memoria de processos
- Modificacao de memoria do kernel
- Integridade interna de kernel
- Acesso remoto via kernel (backdoor?)

Agradecimentos

- **Eventos, Workshops, Seminários, Palestras, RoadShows, Conversas**

- **Comunicação**

- **Elo mais fraco da segurança da informação**



(kernel)(Hacking)

if i really know, i can hack

FIM! Será mesmo?

DÚVIDAS ?

Rodrigo Rubira Branco

rodrigo@kernelhacking.com
bsdaemon@bsdaemon.org