Capture the flag exercise

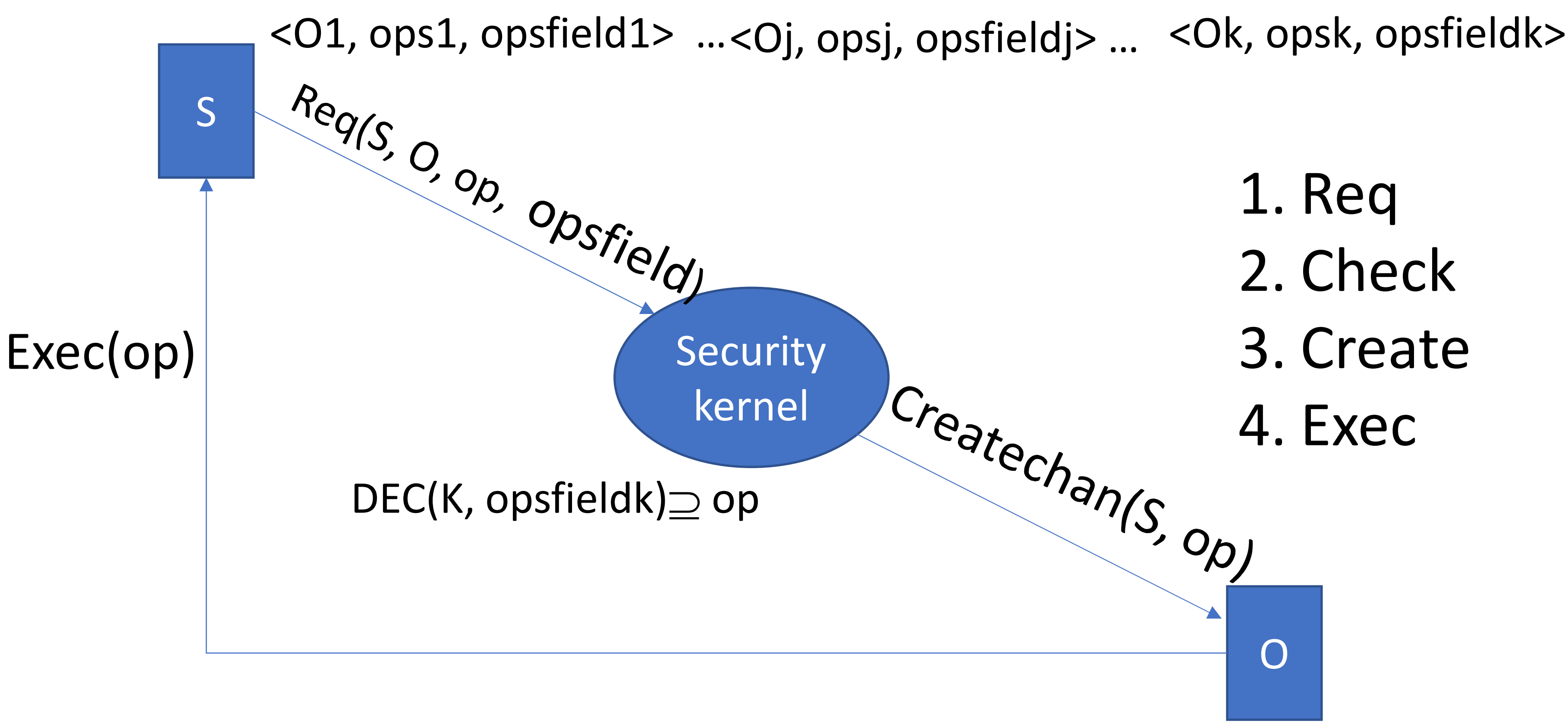
My challenge to you to discover vulnerabilities in a protocol using capability

Outline

1) Each subject manages a list with its own capabilities
2) The operation field of a capability is encrypted with a key private to the security kernel SK
3) To request operation Op on object O, a subject S sends to SK a message with S, O, Op and the encrypted capability
4) SK decrypts the capability and, if it enables Op on O, it asks O to create a channel with S to execute OP
5) O destroys the channel when Op ends

Capture the flag exercise

opsfieldm = ENC(K, ops in ACM[S,Om])



Capture the flag exercise

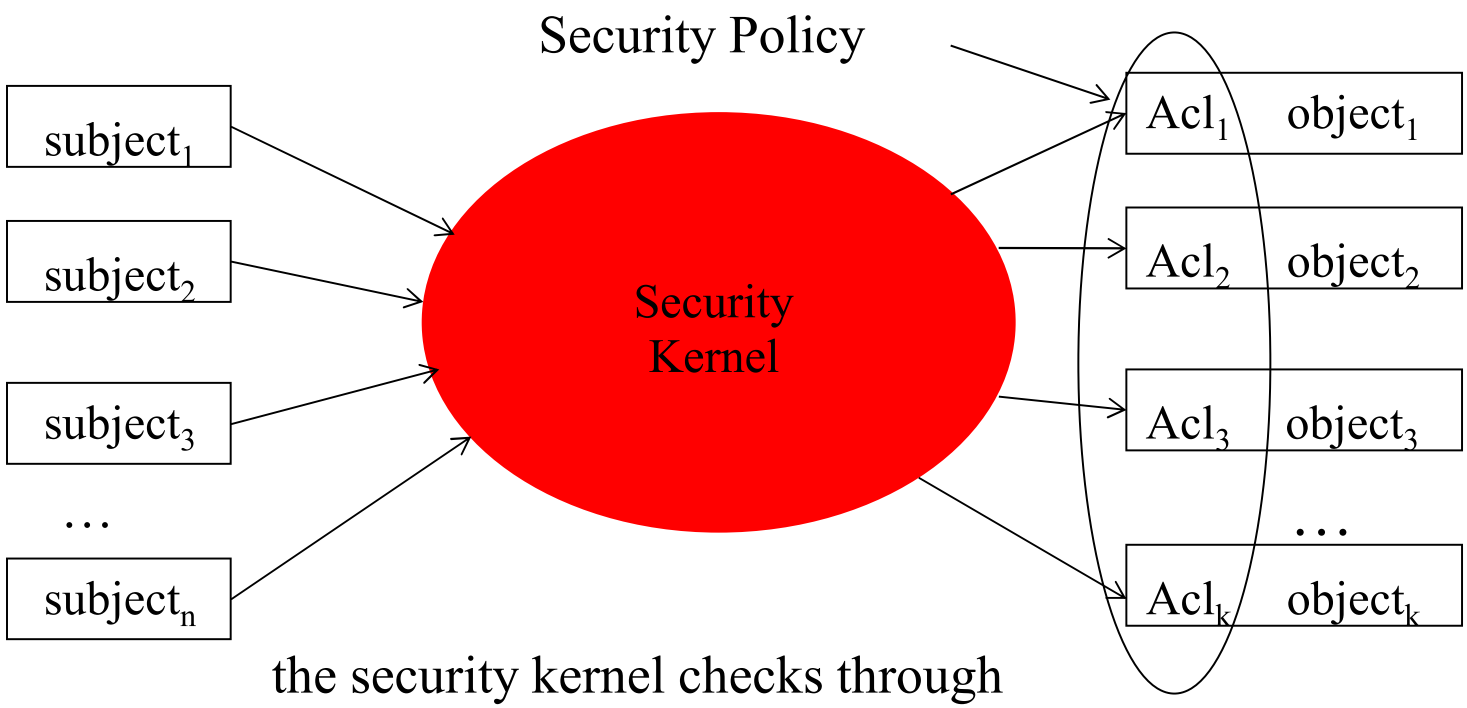
Challenge to you

Discover vulnerabilities in the proposed protocol or in the overall system under the assumption that there are no vulnerabilities in the encryption algorithm ie K cannnot be discovered because of mathematical vulnerabilities

Complete Mediation - 2

- Access control list = a column based organization of the acm
- One list for each object
- Each list element stores the rights of all the subjects on a distinct object
- Now the control can be implemented by the Security Kernel or be delegated to the object
- A centralized structure for each object

ACM: ACL



the security kernel checks through the object ACL that the security policy is satisfied
The checks may also be implemented by the object

Access control list

- A more flexible solution may be achieved through
  - Partition of the subjects
  - The sequential scanning of the list (no direct access is possible because the subject does not know its position)

If subject $\in$ Set1 *then* {op1, op2}
  else If subject $\in$ Set2 *then* {op3, op4}
    else {op5}

this is an ACL!

- the subjects are partitioned into three sets
- this can grant rights even to subjects not known in advance. This is not possible for capabilities and it may be adopted to define acls for web services

HW/FW support for ACL

- Associative memory where the key may be
  - Subject $\rightarrow$ set of rights
  - Subject, operation $\rightarrow$ boolean
- FPGA that implements a function that is a chain of *if statements* about
  - Sets of users
  - Priority among sets

ACL vs Unix files

- Each file is paired with a bit array that defines
  - Owner rights
  - Group owner rights
  - Other users rights
- this is an implementation of the file ACL
  - It adopts classes of users due to missing information on all the system users

ACL and file descriptor

```
struct stat {
mode_t st_mode; // File type & mode                access control list + set uid bit
ino_t st_ino; // i-node number
dev_t st_dev; // device number (file system)
dev_t st_rdev; // device n. for special files
nlink_t st_nlink; // number of links
uit_t st_uid; // user ID of owner
gid_t st_gid; // group ID of owner
off_t st_size; // size in bytes, for reg. files
time_t st_atime; // time of last access
time_t st_mtime; // time of last modif.
time_t st_ctime; // time of last status change
long st_blksize; // best I/O block size
long st_blocks; // number of 512-byte blocks
}
```

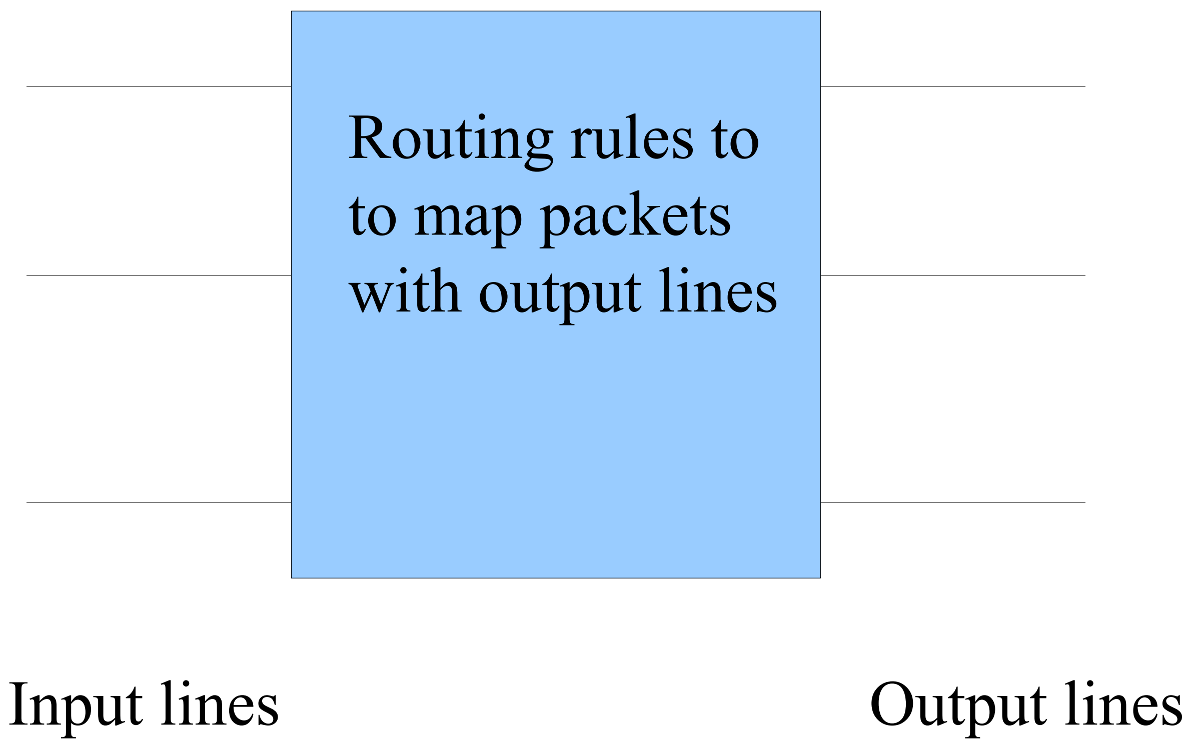
Unix/Linux -I

- ACL are defined in terms of process identifier
  - Real user ID   owner
  - Effective user ID
  - Saved user ID

in Linux we also have

  - File system ID

ACL for message routing in routers



ACL for both input and ouput lines

ACL for message routing

- Router ACLs are built by composing two cases

  IP $\text{Range}_1 \rightarrow$ route

  packets from these nodes are routed

  IP $\text{Range}_2 \rightarrow$ drop

  packets from these nodes are dropped

- A list is built for each input/output connection to specifies the IP addressed in the packets that can cross the connection
- List = order is important
- Ranges because some addresses may be unknown
- This protects the network where messages are routed

ACL & Router

- ACL of input 1
  - 131.114.*.* $\rightarrow$ route
  - 131.4.5.6 $\rightarrow$ route
  - 131.4.*.* $\rightarrow$ drop

  Traffic from 131.114.*.* is routed and all the traffic from 131.4.*.* is dropped but that from 131.4.5.6
- ACL of output 1
  - 131.114.*.* $\rightarrow$ drop
  - 131.4.*.* $\rightarrow$ drop

  No address in 131.4.*.* and in 131.4.*.* can send traffic to the network connected to output 1

swapping two rules changes everything

Routing in Linux: iptables

- Input chain: rules for the packets addressed to the node
- Output chain: rules for the packets produced by the node
- Forward chain: rules for the packets that cross the node
- Default allow $\rightarrow$ transform into a default deny by creating the list of packets to be routed and add “drop all” at the end

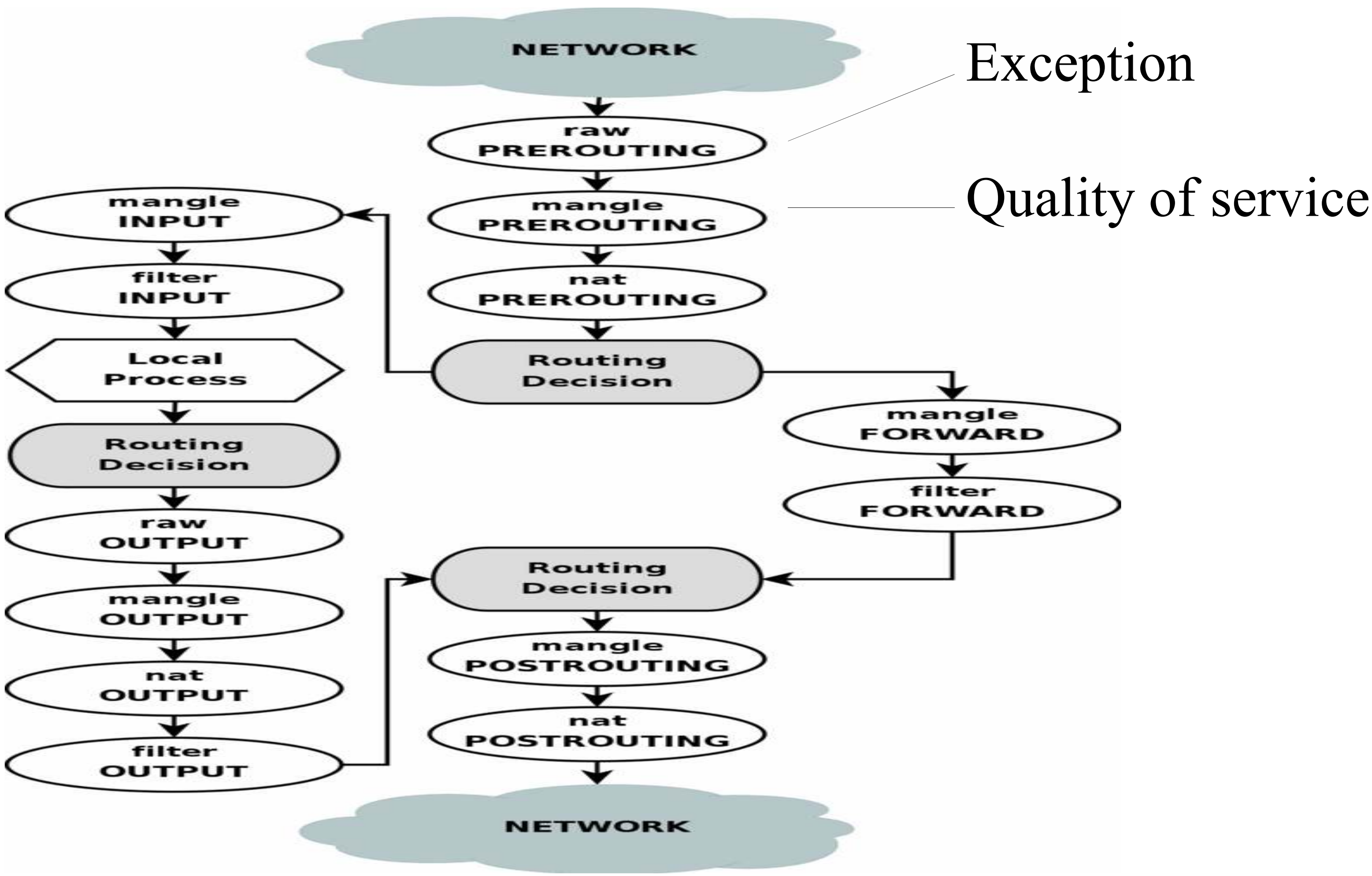
Routing in Linux

- Drop
- Route
- Return – return to the invoking chain
- Queue – transmit to user space
- Log
- Reject
- Dnat/Snat/Masquerade

Nat table

- Prerouting chain= any input packet
- Postrouting chain = any output packet
- NAT may change the addresses in a packet
- Applied before INPUT and after OUTPUT/FORWARD

The overall architecture



Examples

- iptables –A INPUT –p UDP drop

  A new rule is inserted in the input chain to drop any UDP packet
- iptables –A INPUT –p TCP –dport 156 drop

  Drop any TCP packet addressed to port 156
- iptables – N newcontrol

  Create a new chain where new controls can be later inserted

An important point

- Anyone is aware and agrees on the importance of controlling the network traffic that enters a network
- These controls are critical and they are mostly implemented in the border router that connects a network to a pubblic one
- Are there any reasons to check the traffic leaving a network?

Controlling the output traffic

- The control of output traffic is an important mechanism to discover successful attacks against the network (egress filtering)
- If someone is controlling a node X and stealing information in X we can discover the illegal connections of X to some outside network
- These controls can discover Zombies to implement a DDOS

Egress filtering

- It controls the traffic that is attempting to leave the network.
- Before an outbound connection is allowed, it has to pass the filter's rules
- Advantages
  - Discover malware
  - Stop contributing to attacks
  - Block unwanted services

ACMatrix, subjects and objects

- As the number of subjects and objects increases, the complexity of
  - defining the ac matrix
  - checking its correctness
  - achieving full mediation

strongly increases

- Some solutions have been proposed to simplify the definition of this matrix

Role vs subject

- The notion of role is useful when (subject = a final user)
- Role =
  - A professional profile and the corresponding rights
  - Strongly depends upon the applicative environment
- Any role is paired with
  - A set of users that can be assigned that role
  - A set of rights
- Role Based Access Control
- Rights are not assigned to users but to roles
- A user U acquires the rights when a role is assigned to U
- When U leaves the role, it loses the role rights