Information Hiding

Foundations of Cybersecurity

Luca Caviglione

Institute for Applied Mathematics and Information Technologies

National Research Council of Italy

luca.caviglione@cnr.it

University of Pavia – Department of Electrical, Computer and Biomedical Engineering

Outline

- Why information hiding in this course?
- Information Hiding
- Offensive Use of Information Hiding
- Covert Channels: Models, Metrics, Attack Model and Malware Examples
- Detection and Mitigation
- Local Covert Channels and the “Colluding Applications” Scheme
- Detection and Mitigation
- Stegomalware: Attack Model and Malware Examples
- Detection, Mitigation and Sanitization
- Side-Channel-Attacks
- Defensive Use of Information Hiding
- Watermarking: Properties and Examples
- Attacks and Challenges
- Wrap Up

Why Information Hiding in this course?

- Despite the efforts of security experts and researchers:
  - **countermeasures** against malware are progressively showing **limitations**
  - only a **fraction** of threats is **detected**
  - attackers increasingly operate **undisturbed** for **longer timeframes**.
- A basic knowledge of **Information Hiding** can be useful to**:**
  - understand how malware developers can avoid detection for long periods
  - comprehend advanced **offensive** schemes
  - knowing a **double-edged** mechanism also valuable for **defense**.
- Concepts related to cloaking mechanisms can:
  - underline how to exploit **ambiguities** and **imperfect isolation**
  - showcase how “nice thinkings” can impact software/network security
  - make you aware of some **research** results.

Information Hiding

- **Information Hiding** is an umbrella for a wide spectrum of methods that are used to make data difficult to notice.
- **Steganography** (στεγᾰνός + γραφή):
  - one of the most well-known subfields of information hiding
  - combination of *steganos* (covered, concealed) and *graphe* (writing)
  - first recorded use of the term in 1499 by Johannes Trithemius
  - **goal**: cloak **secret** data into a suitable **carrier**.
- Steganography vs cryptography:
  - steganography: information is difficult to **notice**
  - cryptography: information is difficult to **comprehend**
  - can be used **jointly**.

Information Hiding

- **Information Hiding** techniques can be utilized for:
  - intelligence or military goals
  - industry espionage purposes
  - covert communications with informants (e.g., in investigative journalism)
  - bypass censorship attempts in oppressive regimes
  - criminal activities, especially terrorism
  - exfiltration of sensitive data
  - track contents and claim ownership
  - manage integrity of digital objects
  - store (legitimate) secrets
  - …
- Information hiding is a double-edged sword.

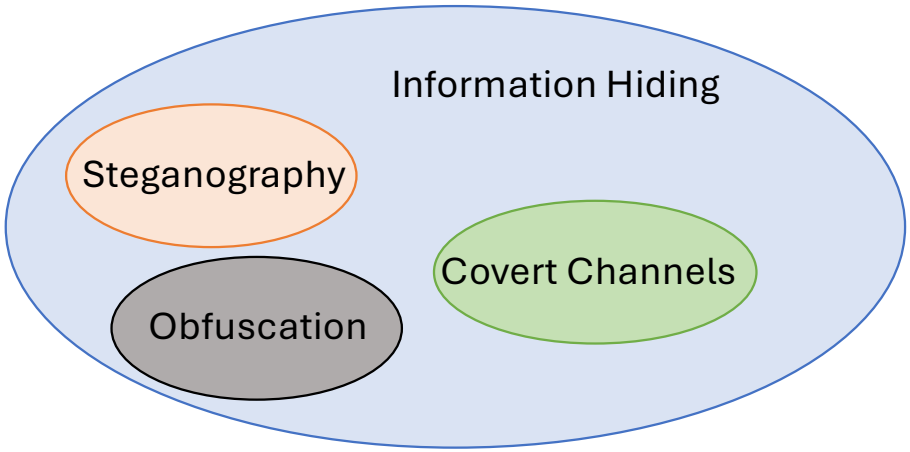
Offensive Use of Information Hiding

- **Malware developers** demonstrate a growing **ability** in:
  - avoiding detection for long periods
  - bypassing security mechanisms like firewalls and signature-based detection schemes.
- Some possible **enablers** are:
  - modular design for customization, e.g., Regin, Flamer, and Weevil
  - multistage loading, e.g., Regin, Stuxnet, and Duqu
  - Cybercrime-as-a-Service models, e.g., Tox
  - Information Hiding techniques, e.g., Platinum APT.

Malware	Discovered	Present since…
Stuxnet	2010	2007
Duqu	2011	2008
Flame	2012	2007
The Mask	2013	2007
Regin	2014	2003

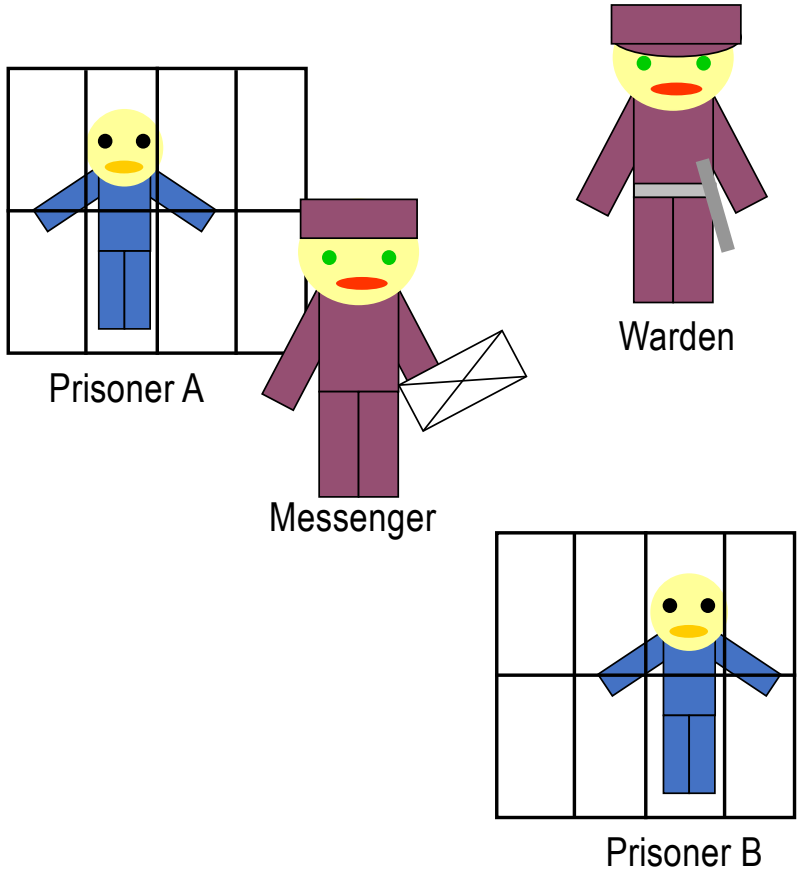
Offensive Use of Information Hiding

- Main mechanisms for cloaking data observed in real-world attacks are:
  - obfuscation
  - **steganography**
  - **covert channels**.



Covert Channels: Models

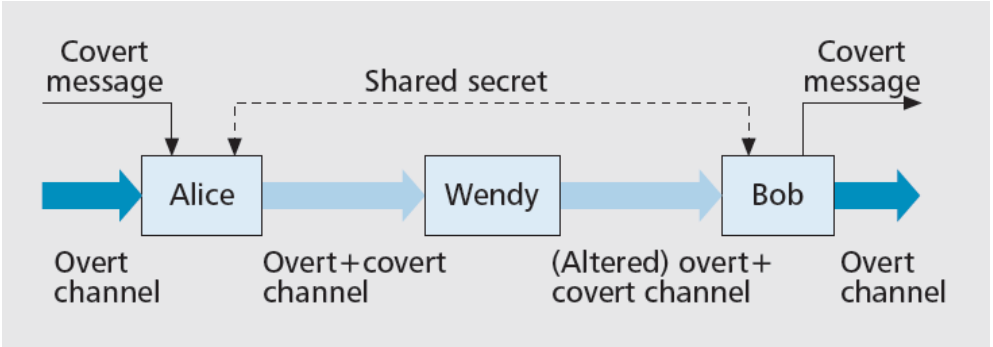
- Modeled by **Simmons** (1984) as the "*Prisoners' Problem and the Subliminal Channel*".



G. J. Simmons, "The Prisoners' Problem and the Subliminal Channel", Advances in Cryptology: Proceedings of Crypto 83, pp. 51-67, 1984

Covert Channels: Models

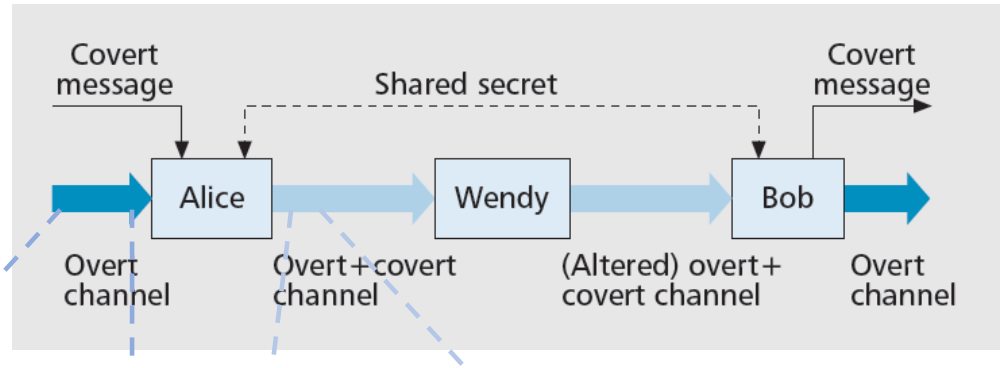
- Modeled by **Simmons** (1984) as the "*Prisoners' Problem and the Subliminal Channel*".
- According to **Lampson** (1973):
  - *"[channels] not intended for information transfer at all, such as the service program's effect on the system load*".



B. W. Lampson, "A Note on the Confinement Problem", Communications of the ACM, Vol. 16, No. 10, pp. 613-615, Oct. 1973

Covert Channels: Models

- Modeled by **Simmons** (1984) as the "*Prisoners' Problem and the Subliminal Channel*".
- According to **Lampson** (1973):
  - *"[channels] not intended for information transfer at all, such as the service program's effect on the system load*".



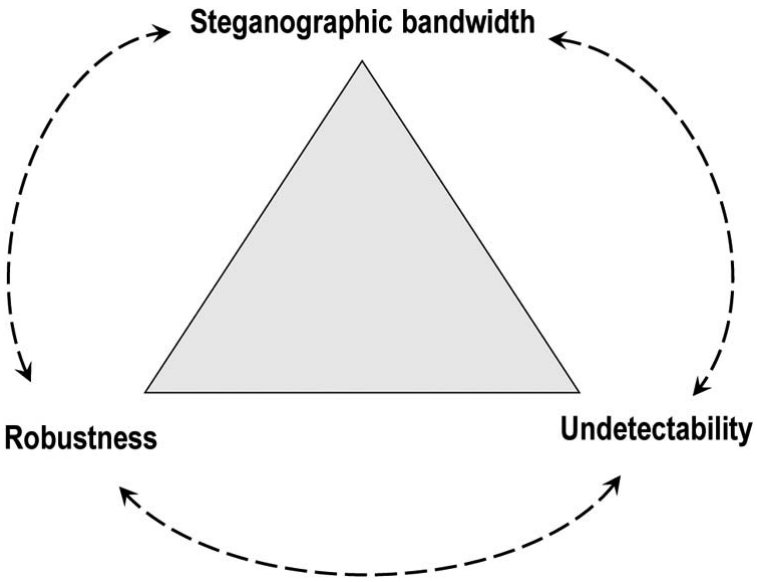
B. W. Lampson, "A Note on the Confinement Problem", Communications of the ACM, Vol. 16, No. 10, pp. 613-615, Oct. 1973

Covert Channels: Metrics

- Covert communications are usually characterized via three metrics:
  - **bandwidth**: the amount of secret data that can be sent per time unit when using a particular method
  - **undetectability**: the inability to detect secret data within a certain carrier
  - **robustness:** the amount of alteration a carrier with covert data can withstand without the secret message being destroyed.

Covert Channels: Metrics

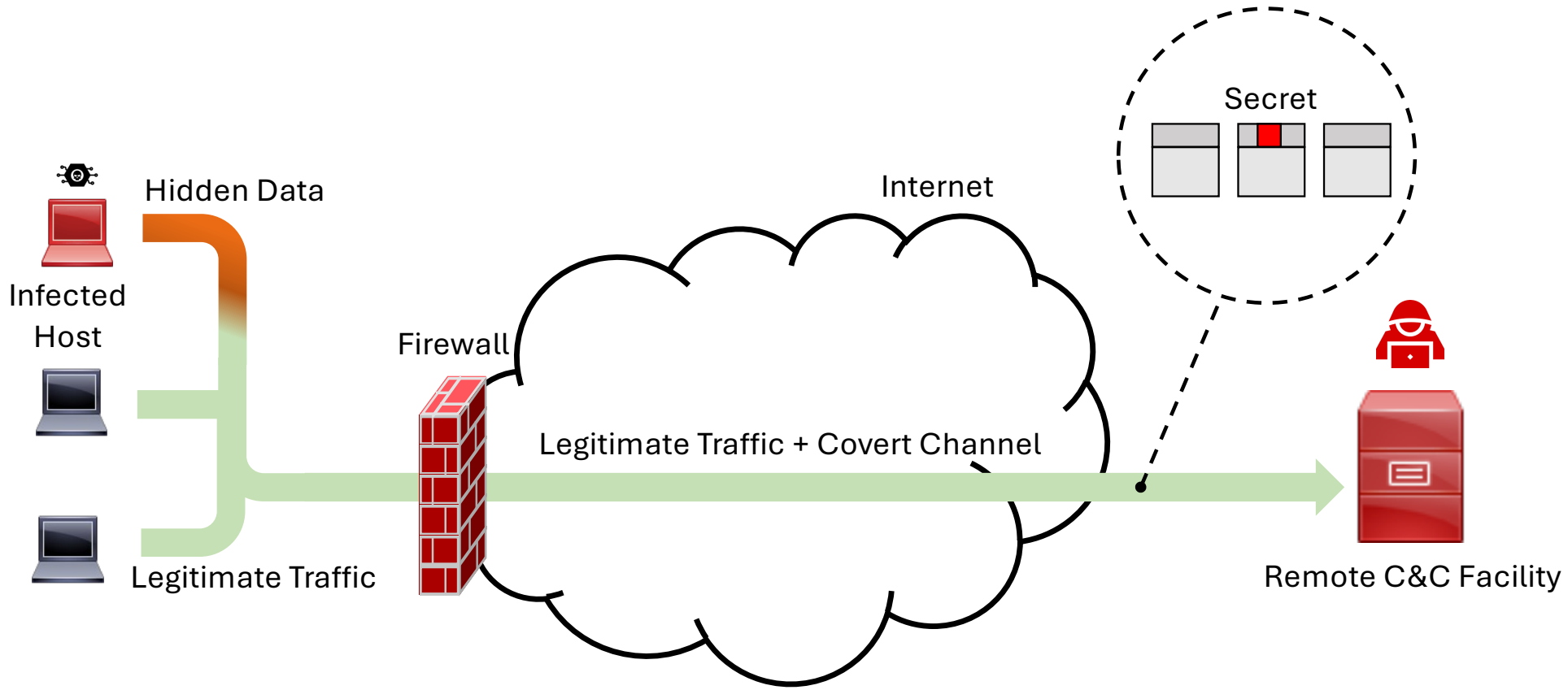
- Covert communications are usually characterized via three metrics:
  - **bandwidth**: the amount of secret data that can be sent per time unit when using a particular method
  - **undetectability**: the inability to detect secret data within a certain carrier
  - **robustness:** the amount of alteration a carrier with covert data can withstand without the secret message being destroyed.
- The metrics are not independent!



Magic Triangle

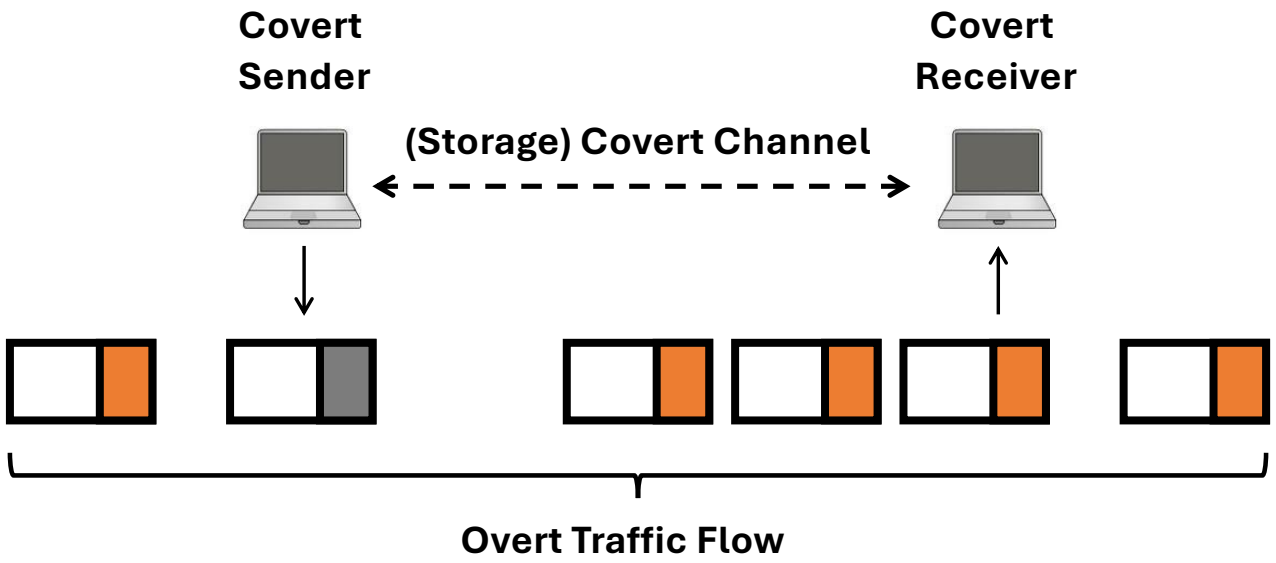
The **magic triangle** rules almost any mechanism devoted to cloak information!

Covert Channels in Network Traffic

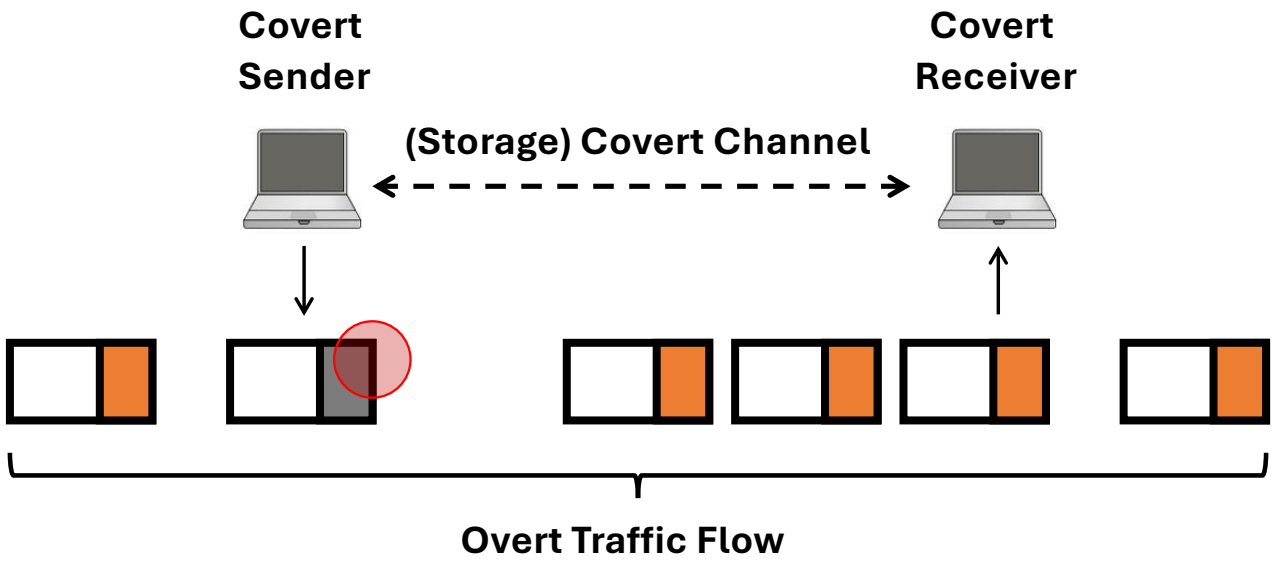


Attack Model

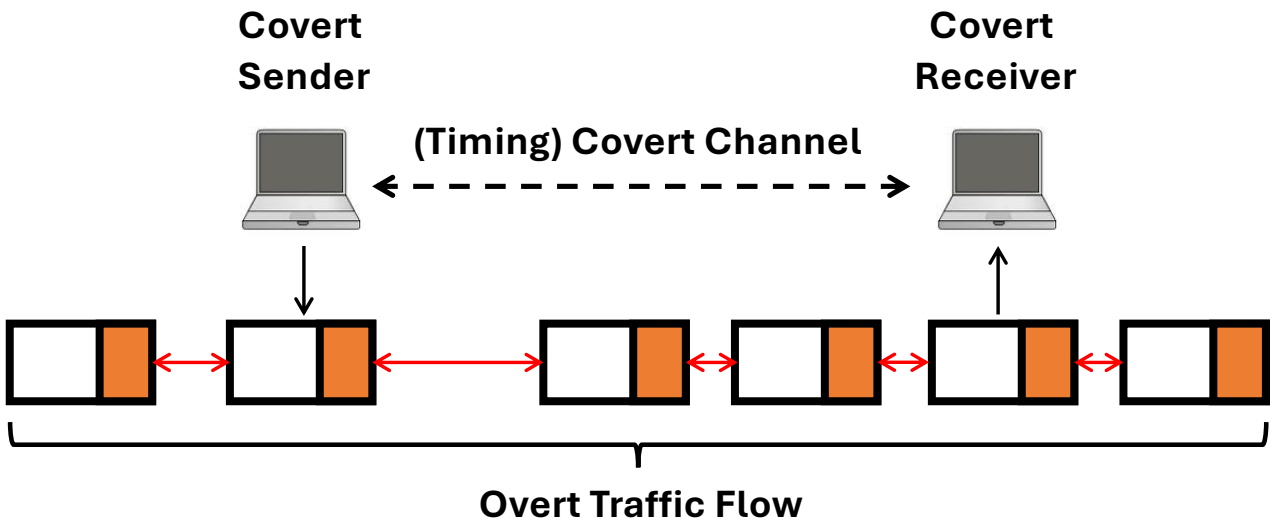
Covert Channels in Network Traffic



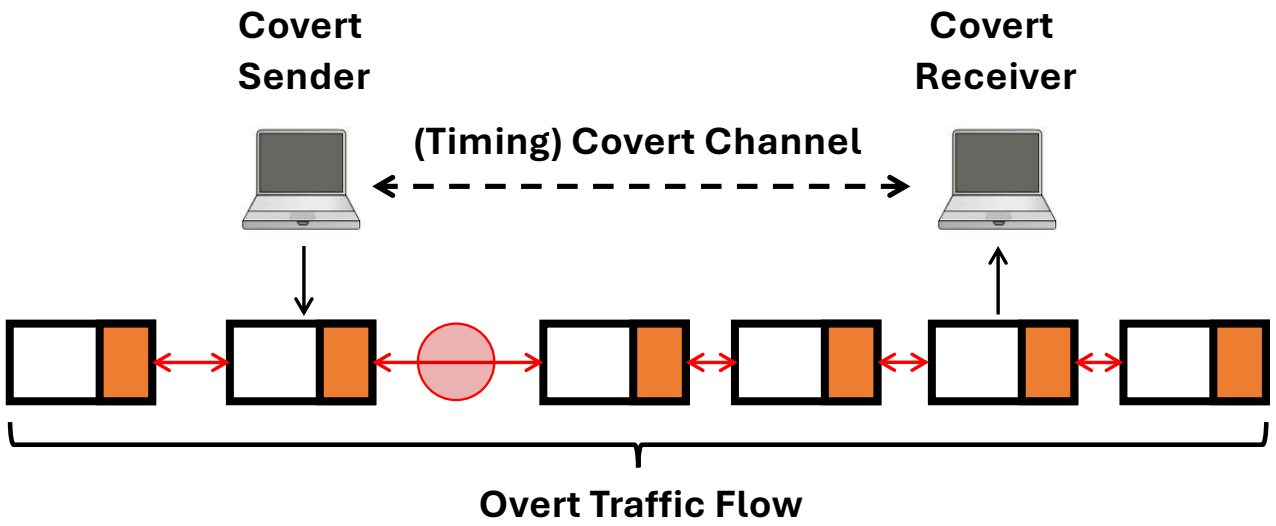
Covert Channels in Network Traffic



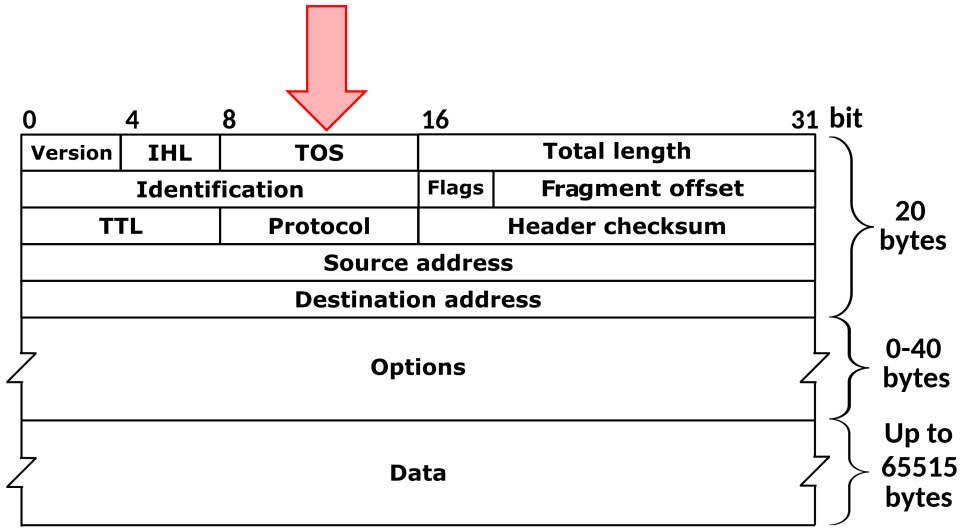
Covert Channels in Network Traffic



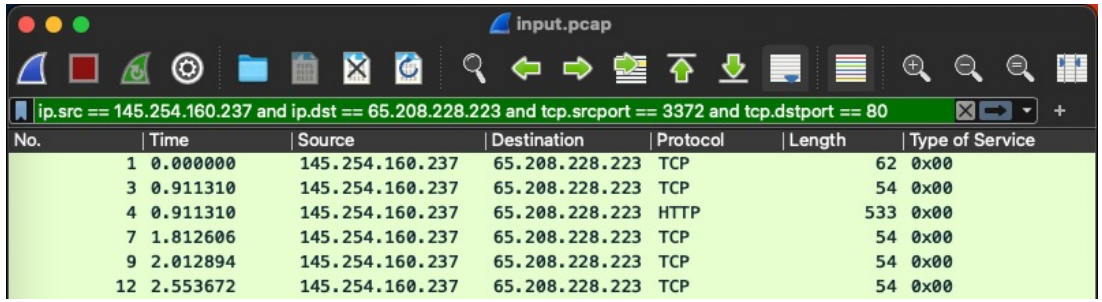
Covert Channels in Network Traffic



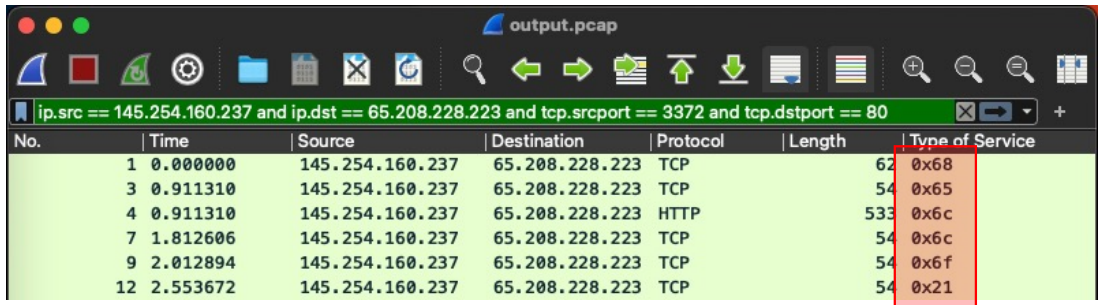
Example: Storage Channel in Type of Service



Overt



Overt + Covert

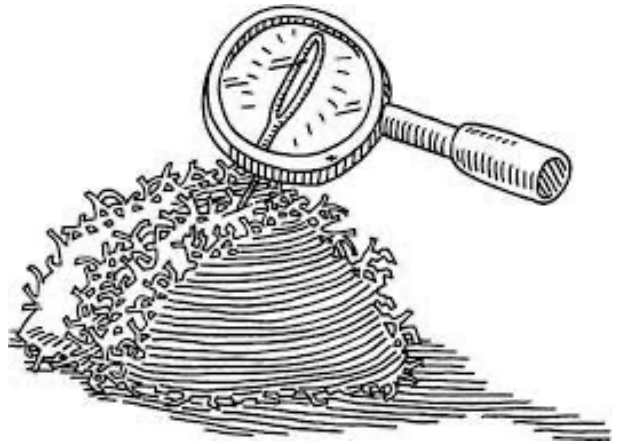


ASCII for “hello!”

https://github.com/Ocram95/pcap_injector

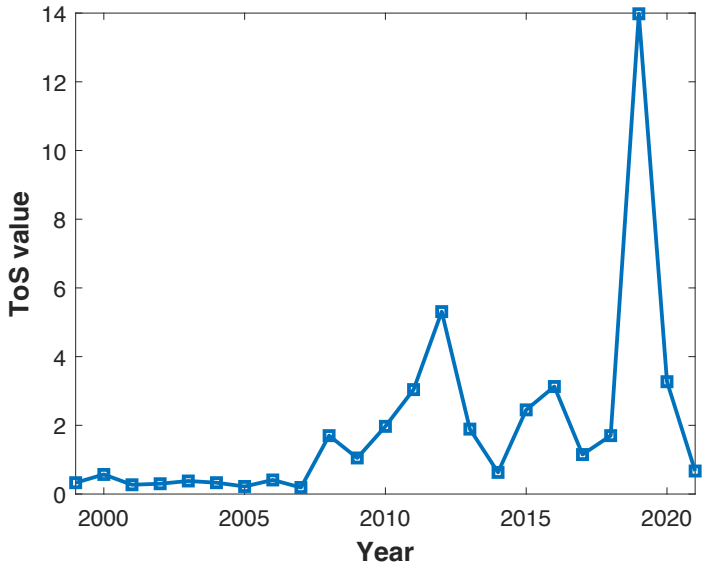
Example: Storage Channel in Type of Service

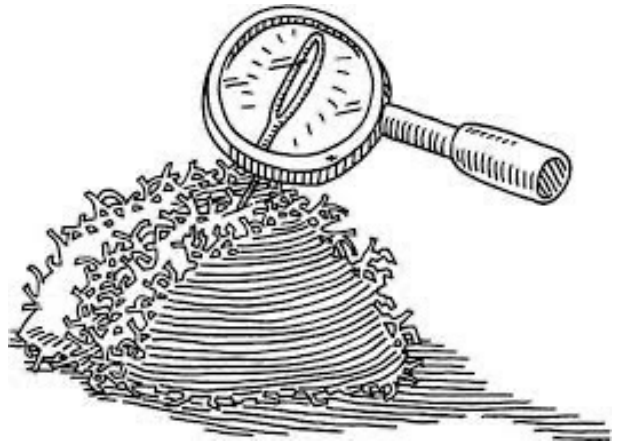
- To be effective, a (network) covert channel should:
  - use a **popular** carrier
  - not reveal the presence of the channel via **anomalous** behaviors
  - **not disrupt** the functionality of a protocol or a service.



Example: Storage Channel in Type of Service

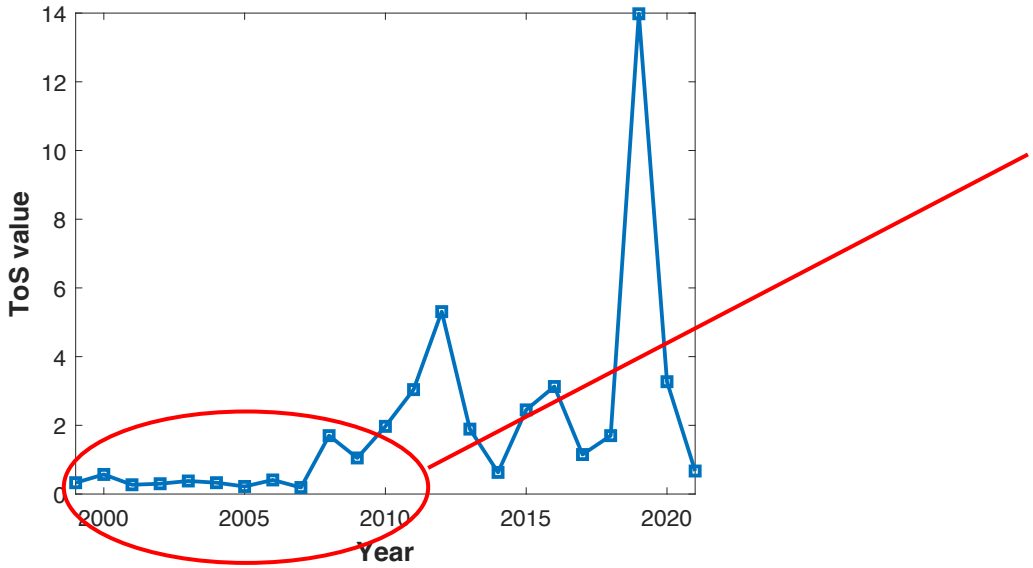
- To be effective, a (network) covert channel should:
  - use a **popular** carrier
  - not reveal the presence of the channel via **anomalous** behaviors
  - **<u>not disrupt</u>** the functionality of a protocol or a service.



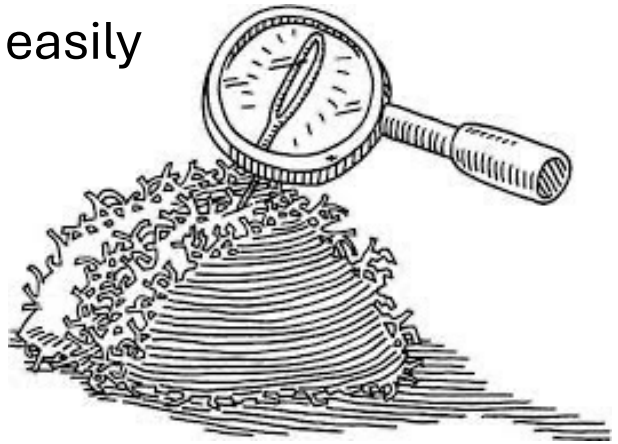


Example: Storage Channel in Type of Service

- To be effective, a (network) covert channel should:
  - use a **popular** carrier
  - not reveal the presence of the channel via **anomalous** behaviors
  - **<u>not disrupt</u>** the functionality of a protocol or a service.

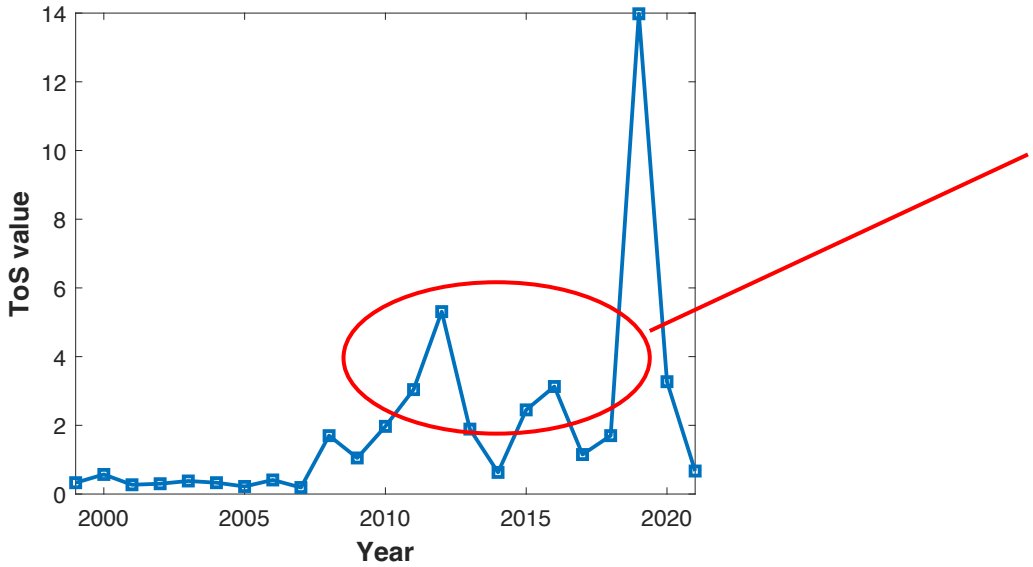


Only few values, thus making the presence of arbitrary data easily detectable.

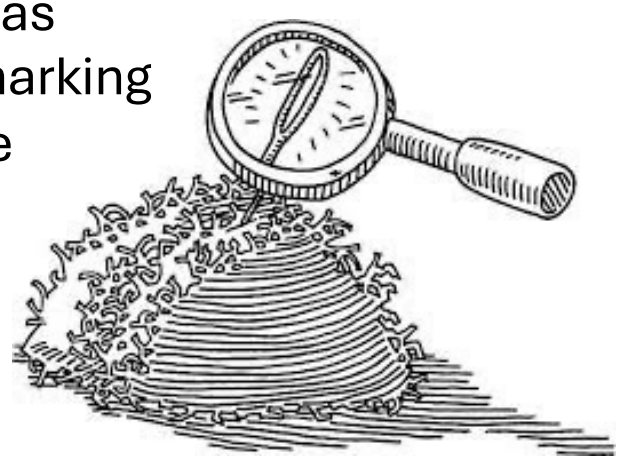


Example: Storage Channel in Type of Service

- To be effective, a (network) covert channel should:
  - use a **popular** carrier
  - not reveal the presence of the channel via **anomalous** behaviors
  - **<u>not disrupt</u>** the functionality of a protocol or a service.



Pathologies in the modification of DSCP by routers, such as bleaching or remarking could disrupt the channel even if undetected.



A. Custura, R. Secchi, G. Fairhurst. “Exploring DSCP Modification Pathologies in the Internet”, Computer Communications, Vol. 127, pp. 86 - 94, 2018

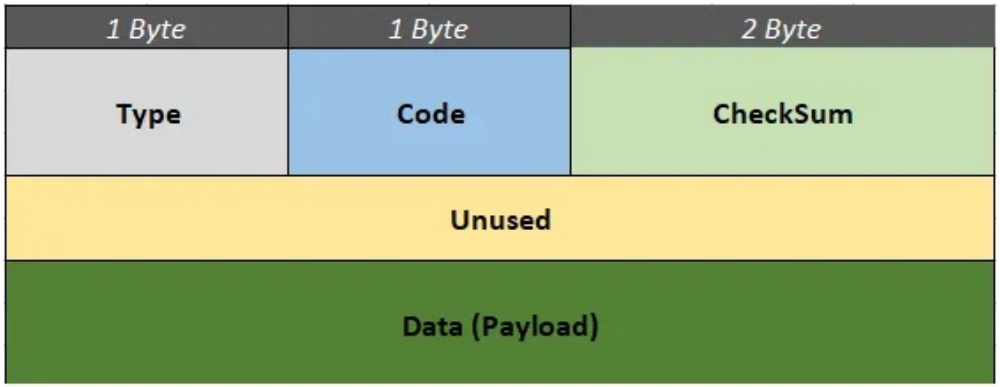
Malware Example: PlugX 2.0

- Discovered in 2014 (update of the PlugX RAT).
- It uses a custom algorithm for encryption and hides C&C communications in ICMP, DNS, and HTTP protocols.
- Attack Phases:
  - ICMP is used to join the C&C server

Malware Example: PlugX 2.0

- Discovered in 2014 (update of the PlugX RAT).
- It uses a custom algorithm for encryption and hides C&C communications in ICMP, DNS, and HTTP protocols.
- Attack Phases:
  - ICMP is used to join the C&C server
  - data is transmitted as a payload of `Echo Reply` packets (ICMP Type 0)

Malicious Data 



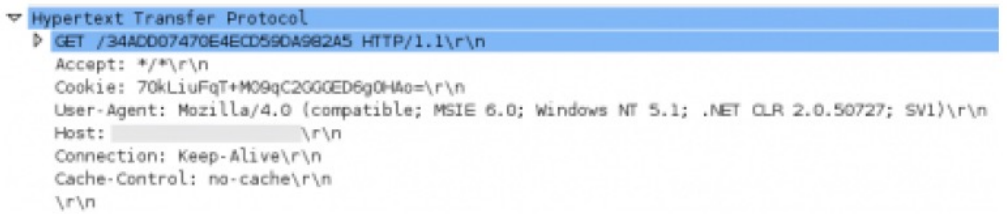
Malware Example: PlugX 2.0

- Discovered in 2014 (update of the PlugX RAT).
- It uses a custom algorithm for encryption and hides C&C communications in ICMP, DNS, and HTTP protocols.
- Attack Phases:
  - ICMP is used to join the C&C server
  - data is transmitted as a payload of `Echo Reply` packets (ICMP Type 0)
  - HTTP to transport signaling

POST/%p%p%p

Method 1

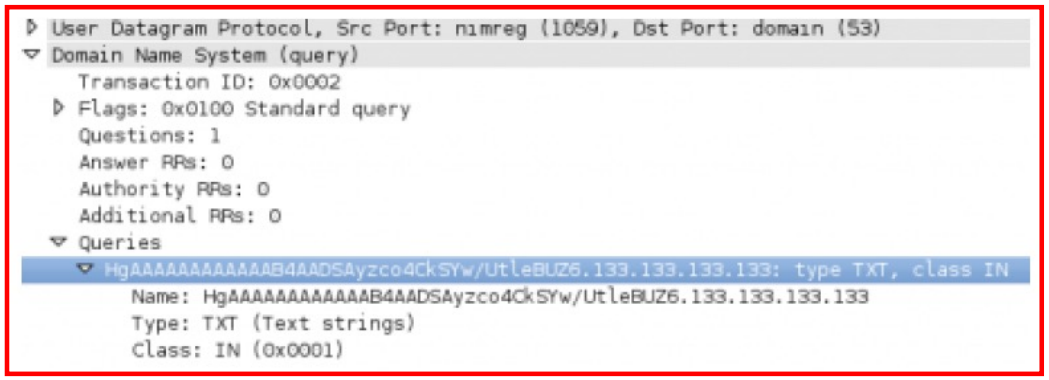
GET (data base64-encoded in Cookie header)



Method 2

Malware Example: PlugX 2.0

- Discovered in 2014 (update of the PlugX RAT).
- It uses a custom algorithm for encryption and hides C&C communications in ICMP, DNS, and HTTP protocols.
- Attack Phases:
  - ICMP is used to join the C&C server
  - data is transmitted as a payload of `Echo Reply` packets (ICMP Type 0)
  - HTTP to transport signaling
  - data is sent in DNS traffic.

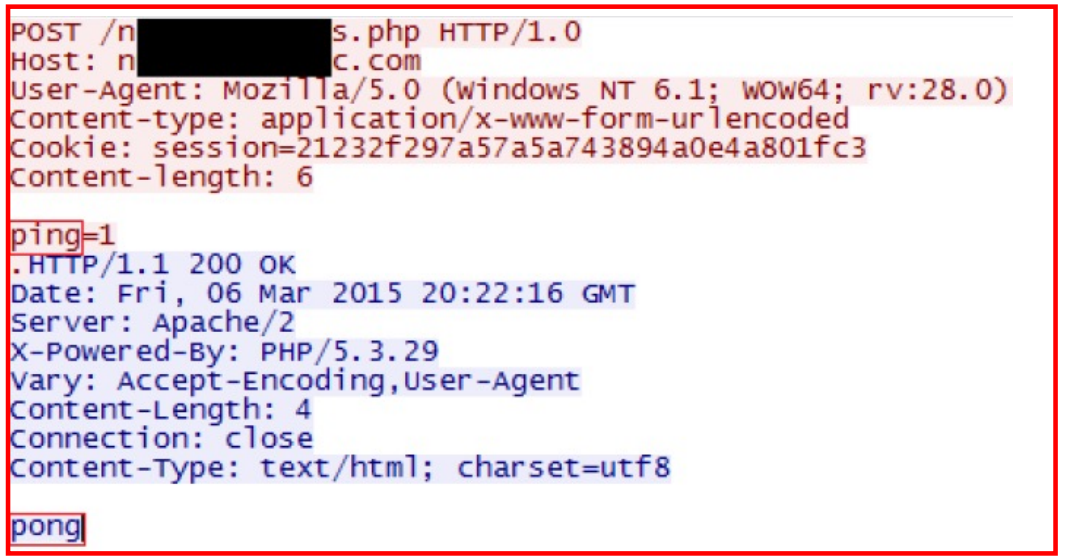


Malware Example: W32/Foreign.LXES!tr

- Discovered in April 2015.
- It hides malicious traffic within HTTP communications.
- Attack Phases:
  - the C&C server status is checked with ping-pongs `HTTP POST/200 OK` messages

Malware Example: W32/Foreign.LXES!tr

- Discovered in April 2015.
- It hides malicious traffic within HTTP communications.
- Attack Phases:
  - the C&C server status is checked with ping-pongs HTTP POST/200 OK messages



Malware Example: W32/Foreign.LXES!tr

- Discovered in April 2015.
- It hides malicious traffic within HTTP communications.
- Attack Phases:
  - the C&C server status is checked with ping-pongs HTTP POST/200 OK messages
  - data is hidden in the source code of 404 messages.

Malware Example: Sunburst

- Sunburst is a trojanized version of the Orion plugin (SolarWinds).
- It targets HTTP traffic.
- Attack Phases:
  - various checks to understand if an analysis tool is running

Malware Example: Sunburst

- Sunburst is a trojanized version of the Orion plugin (SolarWinds).
- It targets HTTP traffic.
- Attack Phases:
  - various checks to understand if an analysis tool is running
  - … (including, opening a backdoor)
  - creates a hidden C&C channel in HTTP.

Sunburst uses HTTP GET or POST requests. The server hides data within HTTP response bodies mimicking benign XML/.NET files. Hidden data is spread across many IDs and strings and extracted via the *\{[0-9a-f-]{36}\}"|"[0-9a-f]{32}"|"[0-9a-f]{16}* regexep.

An Update on Malware Using Covert Channels

- Malware exploiting covert communications:
  - **increased** in the last five years
  - target a wide-array of **protocols** and **hiding techniques**.
- Most popular approaches exploit **DNS** and **HTTP:**
  - ~79% of cases observed in the literature/reports up to June 2024
  - in some threats, **both** protocols are used at the same time.
- **DNS**:
  - TXT queries, NULL AAAA piggyback and malicious subdomains
  - malware: ChaChi, AnchorDNS, IDShell, Invisimole, and Snugy.
- **HTTP**:
  - XOR encoded JSON in the payload, hard-coded names from a list, and cookies
  - malware: ChunkyTuna, LCPDot, ScrabbleCross, and Karkoff.

F. Strachanski, D. Petrov, T. Schmidbauer, S. Wendzel, “A Comprehensive Pattern-based Overview of Stegomalware”, 19th Int. Conf. on Availability, Reliability and Security, Vienna, Austria, July 30 - August 02, 2024, pp. 1-10

Detection and Mitigation

- Detection and mitigation techniques against network covert channels can be broadly classified in three groups.
- **Normalization** aims at:
  - removing ambiguities in traffic that can be used as a carrier
  - *con*: penalizes everything and everyone.
- **Statistical methods** aim at:
  - recognizing traits that can be used to hide data
  - *con*: require modeling traffic or the “normal” behavior of the workload.
- **AI** aims at:
  - training classifiers to distinguish traffic with covert data from normal flows
  - *cons*: preparing datasets is not easy, AI should be explainable, and inspection cannot be arbitrary (e.g., GDPR).

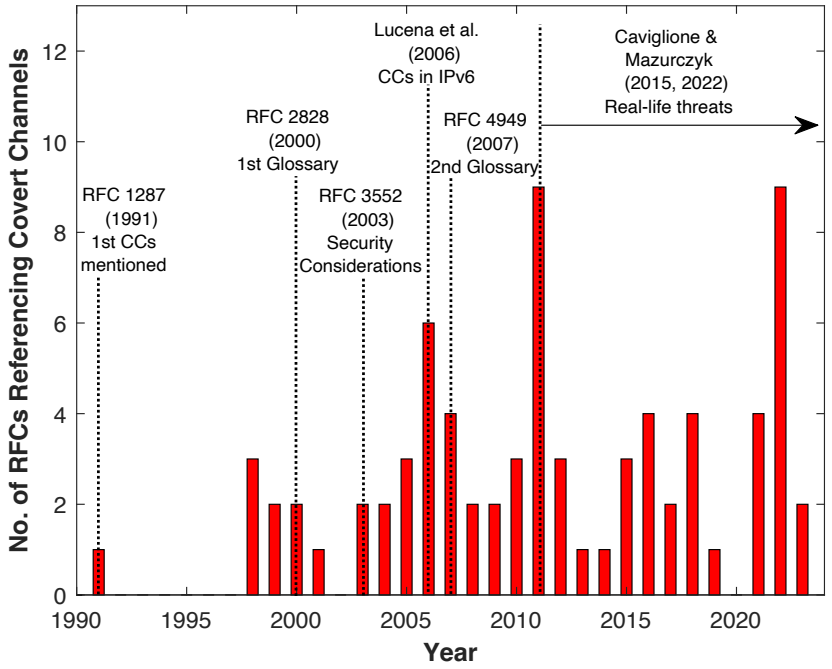
Detection and Mitigation: Challenges

- There is **not** a **unique** hiding strategy and there is **not** a **general** defensive methodology.
- **Anticipating** ambiguities rather than fixing them later:
  - this requires formalization or resistant-by-design approaches
  - bring back hiding mechanisms to recurrent **patterns**
  - complex interplays are difficult to capture “by hand”.
- Make existent approaches scalable:
  - especially for network traffic and Internet-scale scenarios
  - pursue generalization also for the **data gathering** phase.

S. Wendzel, S. Zander, B. Fechner, C. Herdin, “Pattern-Based Survey and Categorization of Network Covert Channel Techniques”, ACM Computing Surveys, Vol. 47, No. 3, pp. 1-26, 2015

Detection and Mitigation: Standardization

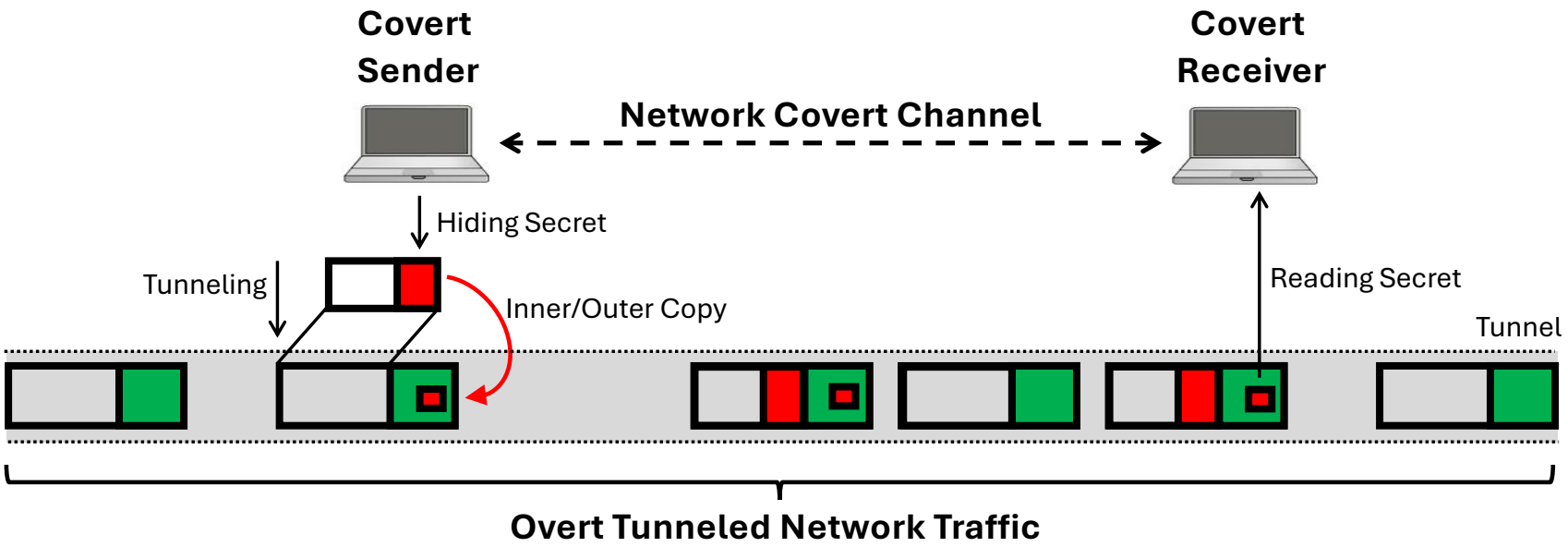
- Typically, **standardization** is the best place for:
  - anticipating ambiguities
  - issuing early warnings
  - promoting updates and fixes.
- The **Internet Engineering Task Force** (IETF) considers covert channels, but:
  - only 76 RFCs explicitly address the problem
  - only 45 RFCs propose some mitigation techniques
  - countermeasures are often **naive**.



L. Caviglione, W. Mazurczyk, “You Can’t Do That on Protocols Anymore: Analysis of Covert Channels in IETF Standards”, IEEE Network, Vol. 38, No. 5, pp. 255-263, Sept. 2024.

Detection and Mitigation: Standardization

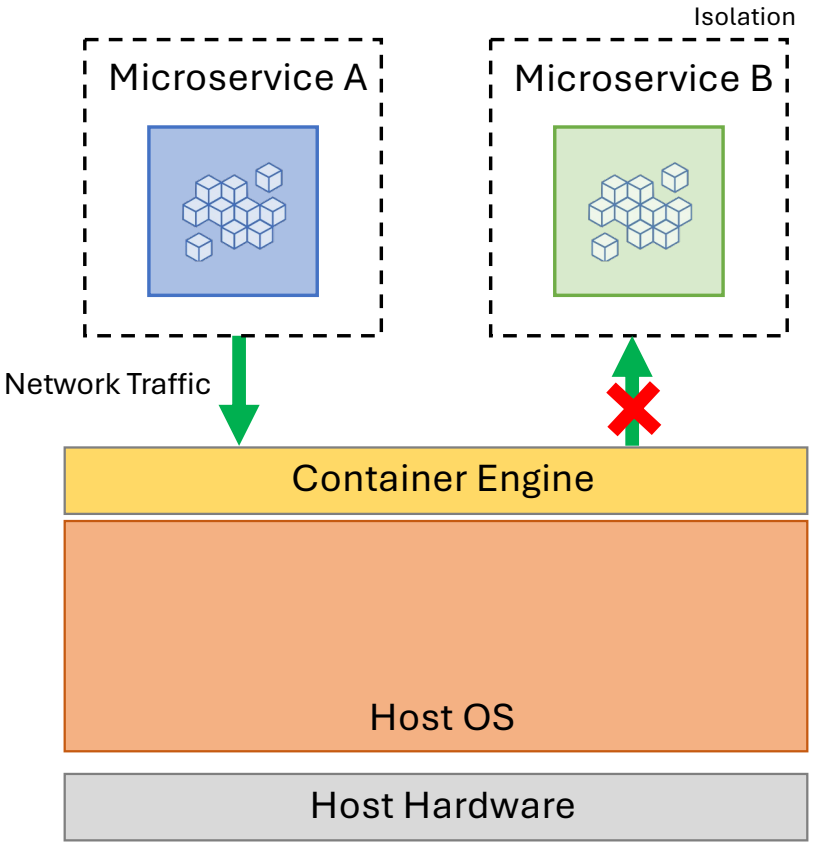
- The IETF considers the following attack models:
  - (basic) timing channels
  - storage channels, mainly in the header
  - **inner/outer copy hiding strategies**



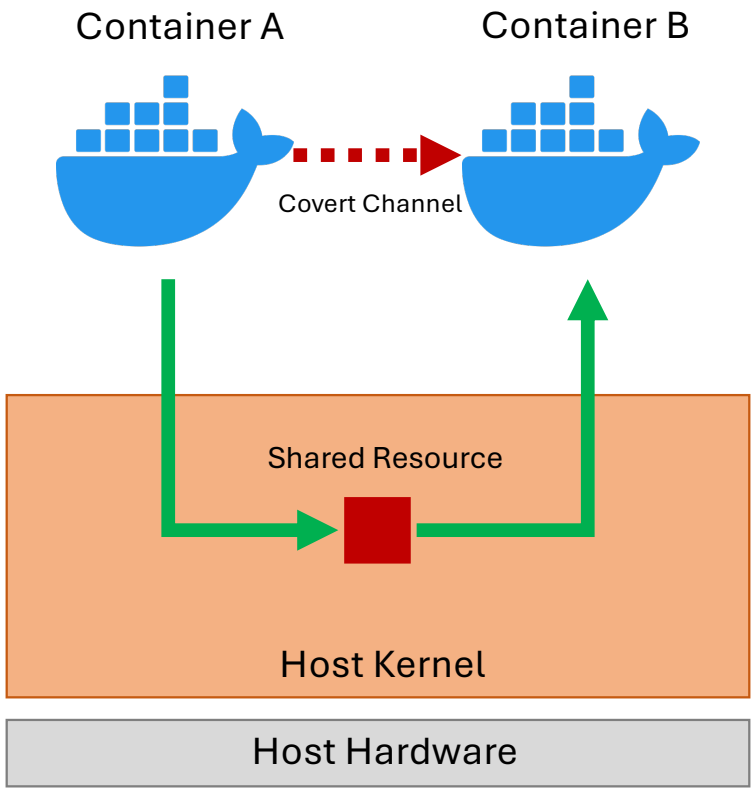
Detection and Mitigation: Standardization

- The IETF considers the following attack models:
  - (basic) timing channels
  - storage channels, mainly in the header
  - inner/outer copy hiding strategies
  - misuse of **watermarking** mechanisms (e.g., the AltMark option).
- It is desirable to improve standards by promoting:
  - zeroing when possible
  - disable unneeded functionalities
  - runtime sanity checks (e.g., reject non-alphabetical characters when not needed)
  - the use of encryption
  - align padding to reduce timing-controllable behaviors
  - …

Covert Channels and Microservices

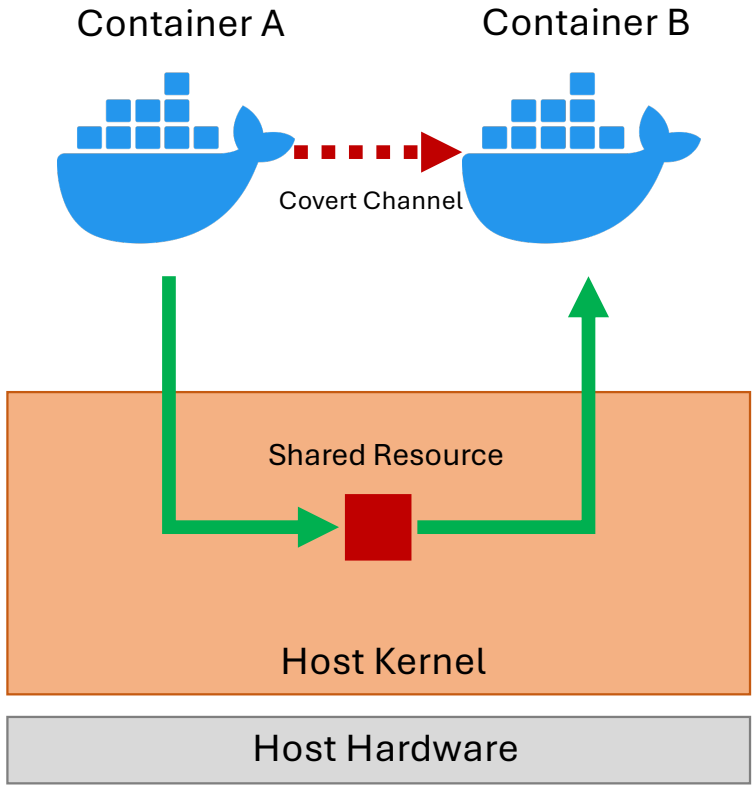


Covert Channels in Docker Containers



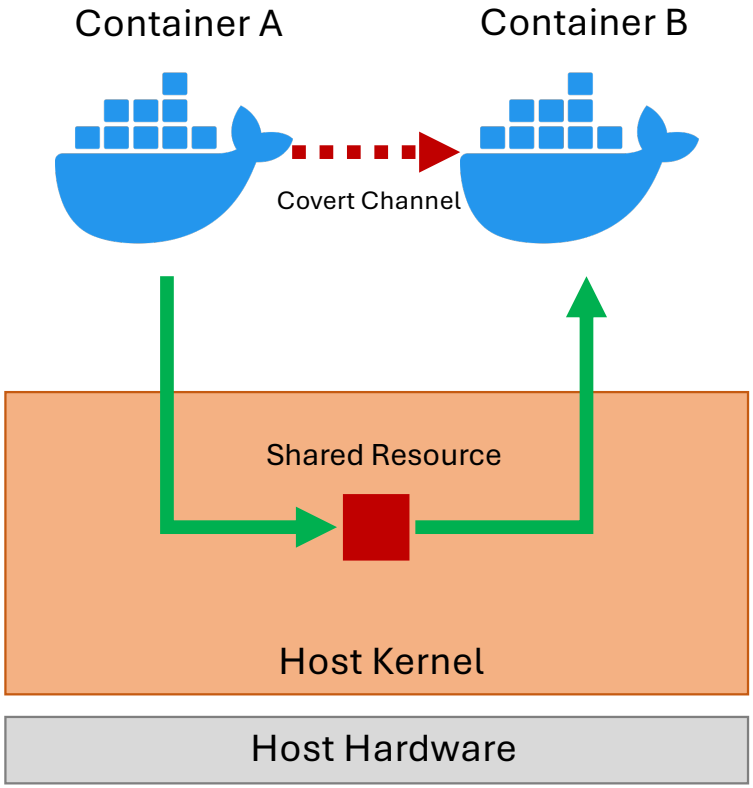
Covert channels also exist among “normal” processes or threads. Containers have been considered here just to introduce an example of a **local covert channel**.

Covert Channels in Docker Containers



- Containers can be targeted with a covert channel to:
  - leak information
  - orchestrate attacks
  - evaluate co-residence (reconnaissance).
- The carrier could be a loosely-isolated shared resource.
- Effective approaches rely on the manipulations of **software** and **hardware statistics** accessible through the `/proc` filesystem:
  - CPU load
  - threads enumeration
  - memory patterns
  - …

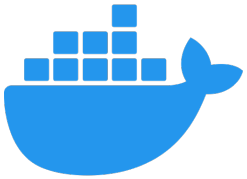
Covert Channels in Docker Containers

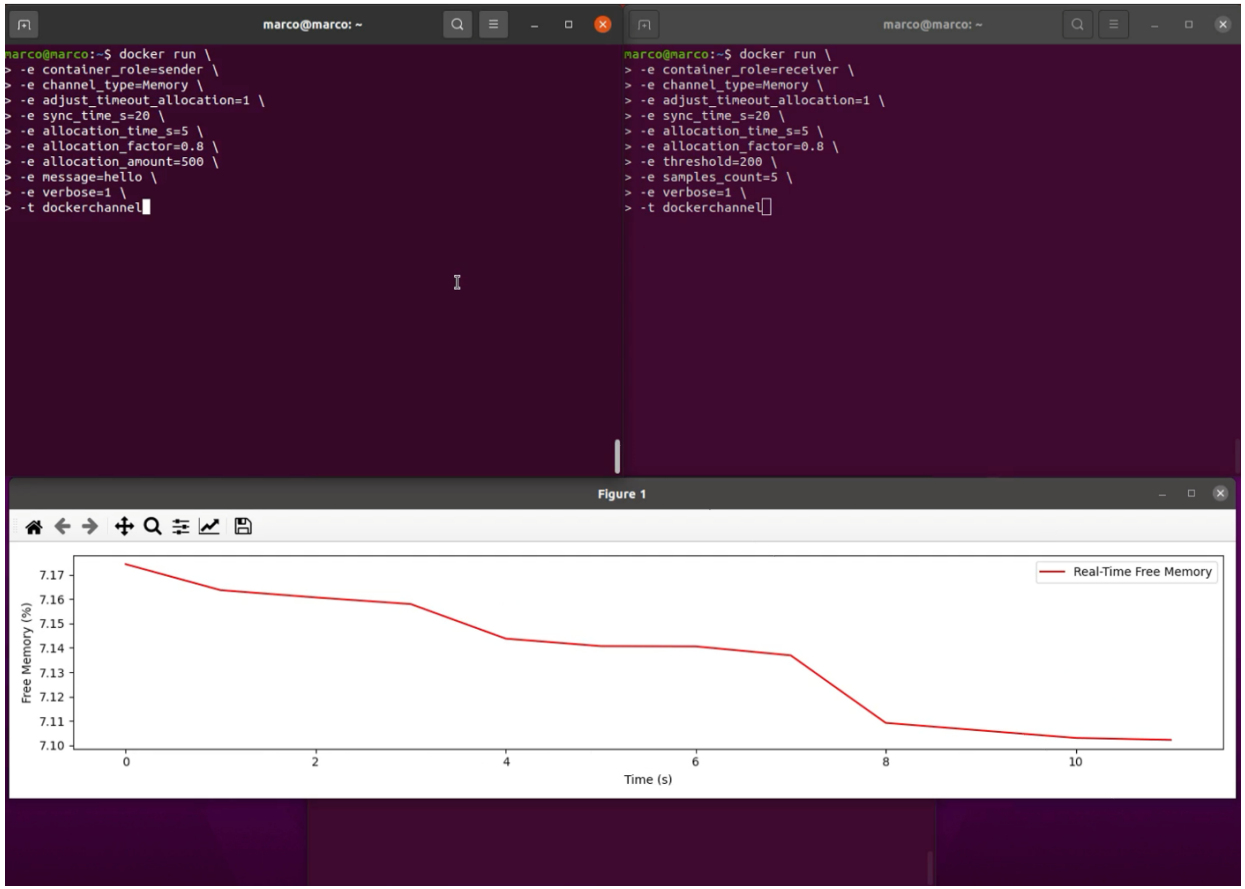


- Example with containers exploiting the free memory available of the host and visible via `/proc/meminfo`.
- **Container A** encodes a binary message by altering the amount of used memory:
  - it allocates 500 MB of memory to encode the bit 1 and sleeps for 5 seconds
  - it sleeps for 5 seconds to encode the bit 0.
- **Container B** periodically monitors the free memory:
  - the bit 1 is decoded if the difference between the average current memory usage and the previous one exceeds by a threshold of 200 MB
  - otherwise, the bit 0 is decoded.

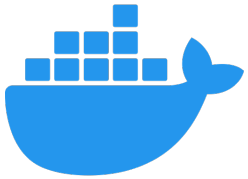
Covert Channels in Docker Containers

Container A





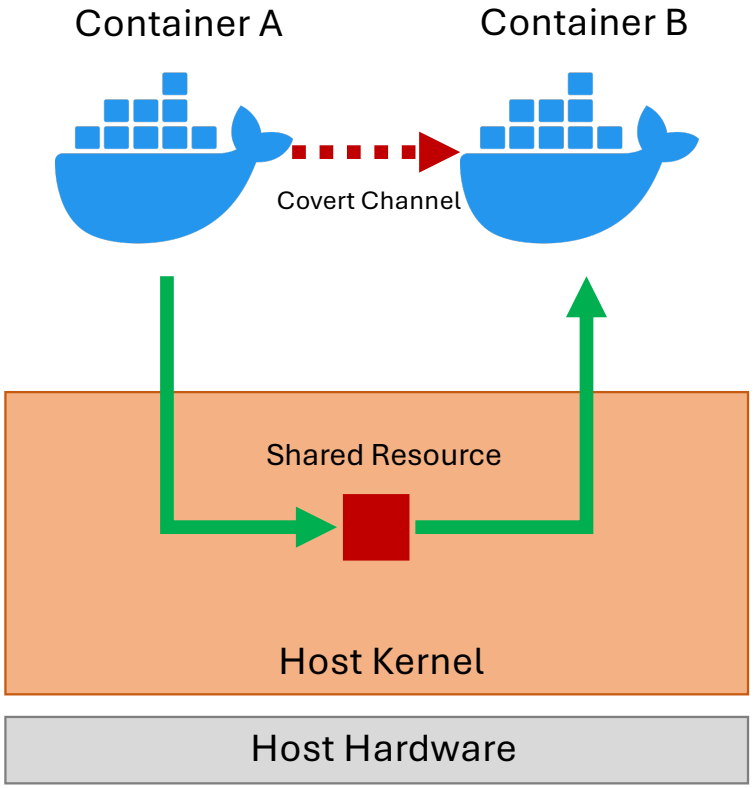
Container B



Shared Resource

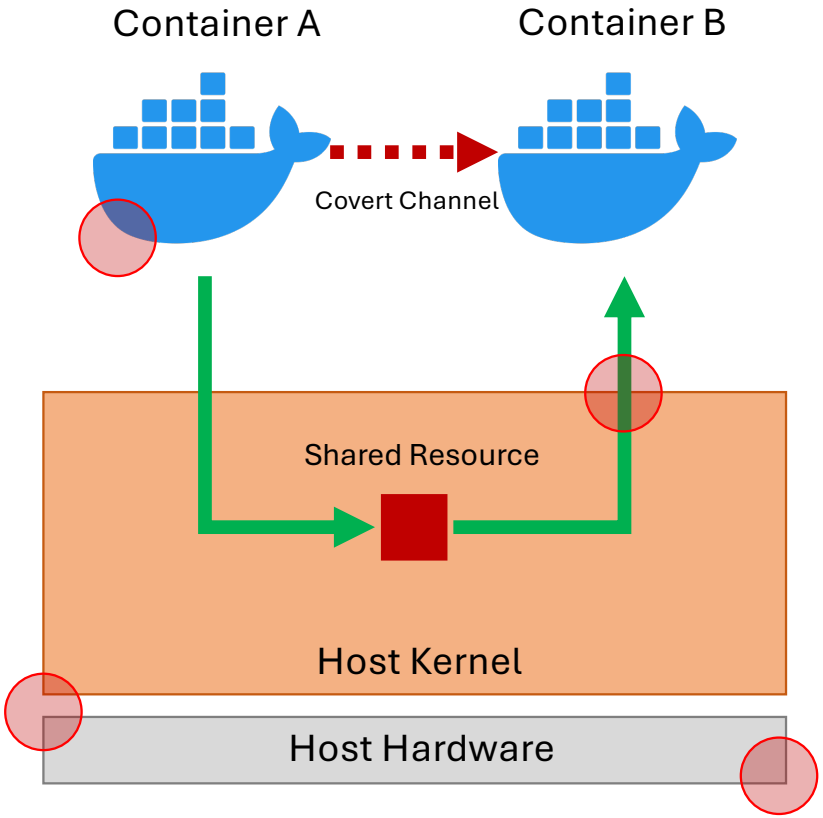
https://github.com/cybersecurity-cnr/DockerChannel

Covert Channels in Docker Containers



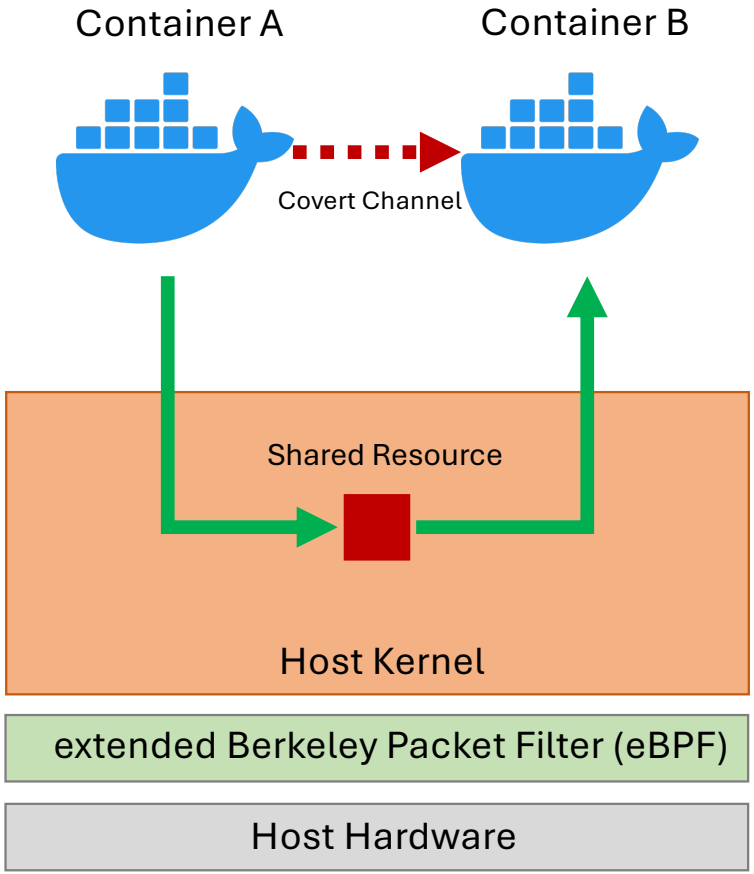
- A possible detection approach is to identify suitable patterns or specific behaviors characterizing the leakage.

Covert Channels in Docker Containers



- A possible detection approach is to identify suitable patterns or specific behaviors characterizing the leakage.
- **Problem 1**: where to perform inspection.

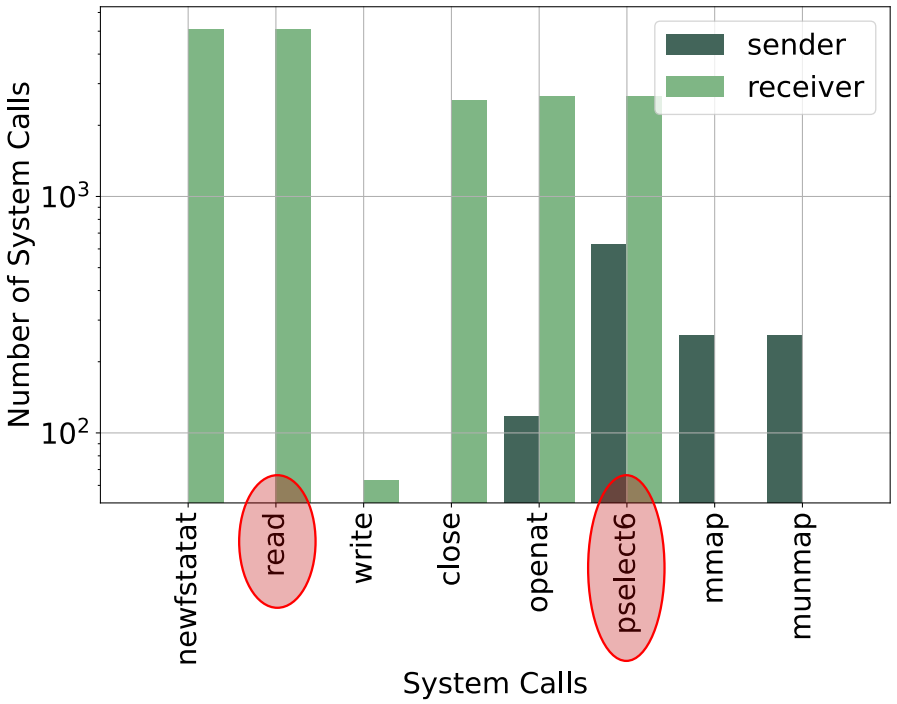
Covert Channels in Docker Containers



- A possible detection approach is to identify suitable patterns or specific behaviors characterizing the leakage.
- Problem 1: where to perform inspection.
- **Problem 2:** how to perform inspection.

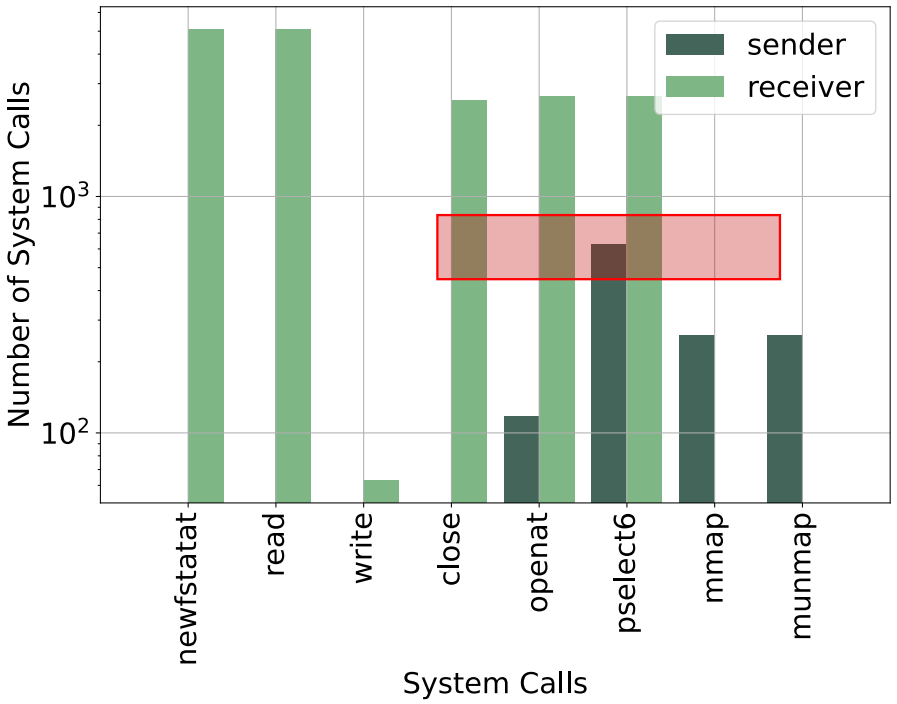
https://ebpf.io

Covert Channels in Docker Containers



- A possible detection approach is to identify suitable patterns or specific behaviors characterizing the leakage.
- Problem 1: where to perform inspection.
- Problem 2: how to perform inspection.
- **Problem 3**: what to inspect.

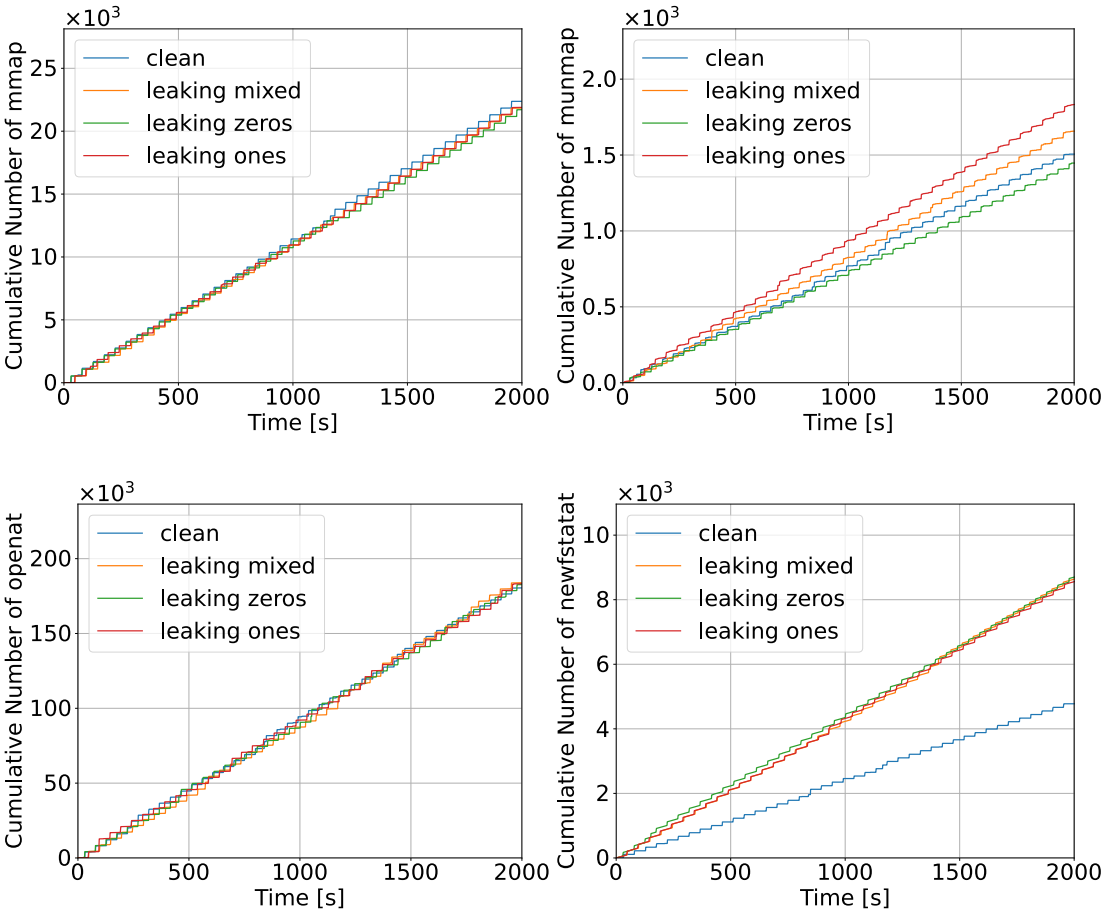
Covert Channels in Docker Containers



- A possible detection approach is to identify suitable patterns or specific behaviors characterizing the leakage.
- Problem 1: where to perform inspection.
- Problem 2: how to perform inspection.
- Problem 3: what to inspect.
- **Problem 4**: how to decide.

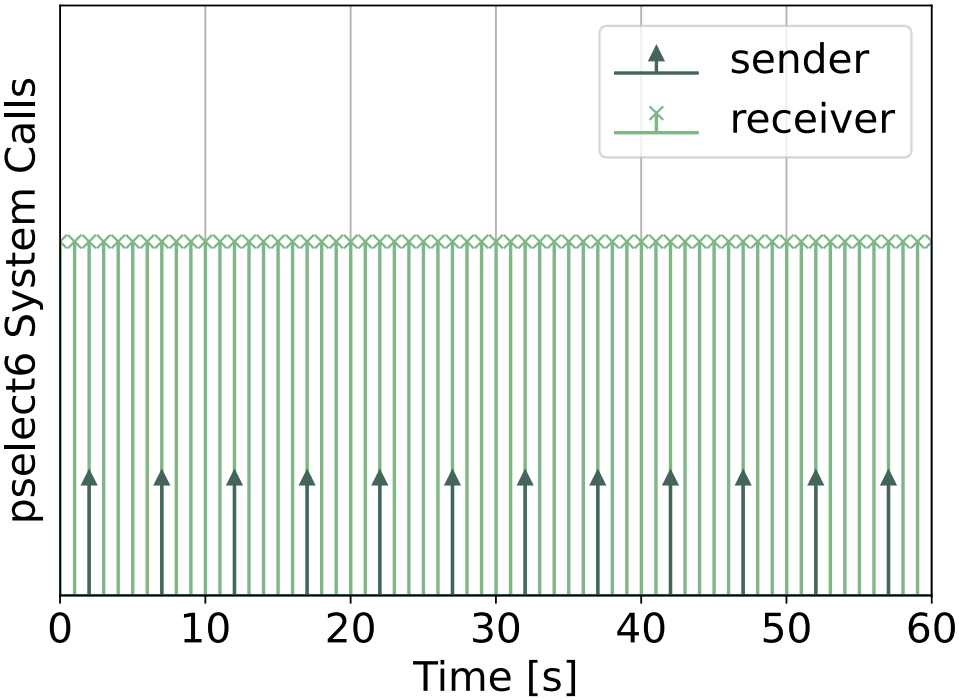
Covert Channels in Docker Containers

- **Example**: a “clean” container compared to another container leaking different messages (512 bits):
  - “leaking mixed”: alternating 0 and 1 bits
  - “leaking zeros”: only 0 bits
  - “leaking ones”: only 1 bits
- Results show *the mmap*, the *munmap*, the *openat*, and the *newfstatat* system calls.



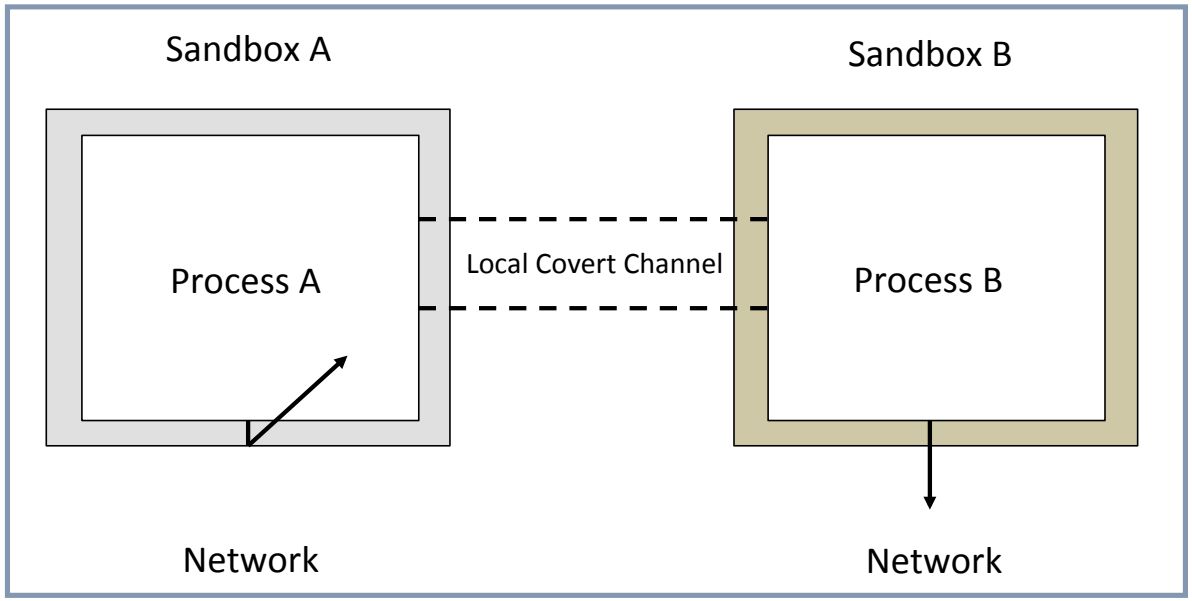
Detection and Mitigation: Abstraction

- In general, pursuing **abstraction** is desirable.
- **Fact**: the sender and the receiver should act close in time to not allow that other processes or containers will disrupt the channel.
- **Idea**: identify tight time correlations among containers.
- The *pselect6* is a good **indicator** as it is used to put processes in a sleep state.



The “Colluding Applications” Scheme

- **Local covert** channels enables the “**colluding applications**” offensive template.
- Often observed in **mobile** scenarios to **bypass** sandboxing.
- Mobile devices offer many features that can act as the carrier:
  - file/socket locks
  - intents
  - volume settings
  - screen lock
  - …
- **Typical attack model**: a process can access personal data, thus the OS prevents the access to the network.
- **Idea**: “leak” information to another process with suitable privileges.



The “Colluding Applications” Scheme

- Example of a typical attack:
  - targeting Android OS devices (a bit outdated, but fun!)
  - data encoded by exploiting the **screen state** and other properties, i.e., *task list* and *process priority*
  - it also beats TaintDroid.
- **Taint analysis**:
  - tracks data propagation through variables, files, and IPC
  - reconstructs the origin of untrusted data
  - identifies vulnerabilities
  - sanitizes content(s).

Tainted Source
(origin of untrusted data)

Tainted Variable
(data from the tainted source)

Vulnerability

```
# Untrusted user input, potentially malicious
user_input = input("Enter something: ")

# Variable now tainted
tainted_variable = user_input

# Example: Potential vulnerability in a database query
sql_query = f"SELECT * FROM users WHERE name = '{tainted_variable}'"

# Without sanitization, a malicious user could inject SQL code
# into 'tainted_variable' and compromise the database.
```

J. Lalande, S. Wendzel, “Hiding Privacy Leaks in Android Applications Using Low-Attention Raising Covert Channels”, *International Conference on Availability, Reliability and Security*, 2013, pp. 701-710

The “Colluding Applications” Scheme

Source: https://www.dailymotion.com/video/x10lcyq and https://www.dailymotion.com/video/x10lbre

Detection and Mitigation

- Detection and mitigation techniques against local covert channels can be broadly classified in three groups.
- **Formalization** aims at:
  - removing ambiguities that can be exploited via by-design approaches or SBOM-like tests as well as via modeling, e.g., resource matrices
  - *con*: requires having templates or suitable patterns to search/match.
- **Statistical methods** aim at:
  - recognizing traits that can be used to hide data
  - *con*: do not scale with the complexity of software.
- **AI** aims at:
  - training classifiers for distinguishing between syscalls/APIs/artifacts containing a covert channel and normal behaviors
  - *cons*: same drawbacks as for the case of network covert channels.

R. A. Kemmerer, “Shared Resource Matrix Methodology: An Approach to Identifying Storage and Timing Channels”, ACM Transactions on Computer Systems, Vol. 1, No. 3, pp. 256-277, 1987

What if a CVE?

- A covert channel can also leverage bugs or weaknesses leading to a CVE.
- A recent example is CVE-2021-30747:
  - targeting Apple M1 Silicon
  - called M1racles - M1ssing Register Access Controls Leak EL0 State
  - *https://m1racles.com*
- How it works:
  - an (unknown) ARM System Register probably not intended for users (e.g., testing?)
  - **EL0 (Exception Level)** can access the register
  - since not properly reset, the register can be exploited as a carrier
  - impossible to patch, can be mitigated via a hypervisor (crazy!).
- Luckily, M1racles is:
  - not a “serious threat”
  - more a privacy leak
  - EL0 are user applications, which have “low” privileges

Stegomalware

- Many researchers are starting to identify threats cloaking data as:
  - **Stegomalware**: steganographic malware
  - "borrowed" from works on mobile security and covert social botnets.
- A possible (common) definition:
  - *stegomalware is a malware using some form of steganography to remain undetected*.
- Personally, I found it a bit reductive:
  - it narrows the scope too much
  - a bit ambiguous (e.g., covert channels vs steganography)
  - it does not fit schemes like colluding applications.

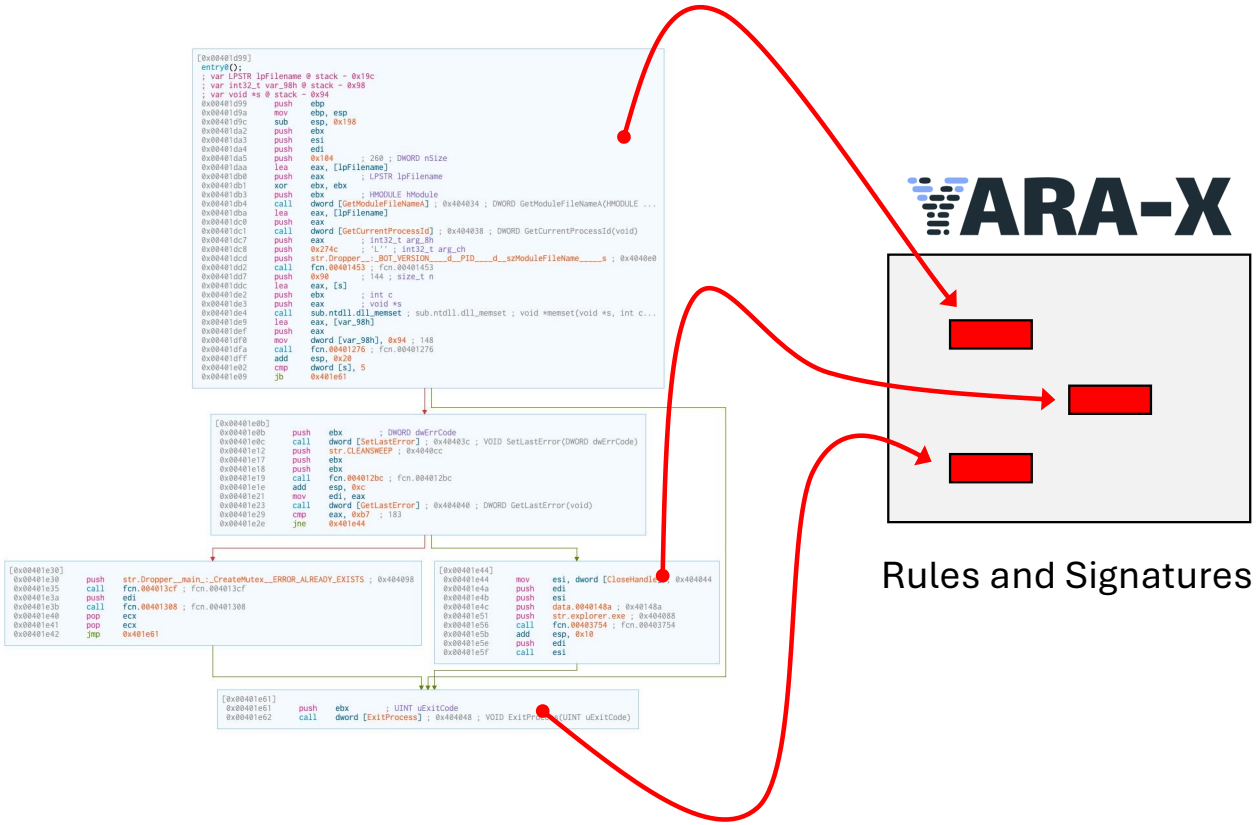
G. Suarez-Tangil, J. Tapiador, P. Peris-Lopez, "Stegomalware: Playing Hide and Seek with Malicious Components in Smartphone Apps", 10th International Conference on Information Security and Cryptology, pp. 496–515, 2014
S. Nagaraja, A. Houmansadr, P. Piyawongwisal, V. Singh, P. Agarwal, B. Nikita, "Stegobot: A Covert Social Network Botnet", Information Hiding - 13th International Conference, pp. 299 – 313, 2011

Classification

- Rough classification of attacks endowed with information hiding capabilities:
  - **Group 1**: malware hiding information by modulating the status of shared hardware/software resources
  - **Group 2**: malware injecting secret data into network traffic
  - **Group 3**: malware embedding secret data by modifying a digital file structure or by using digital media steganography.

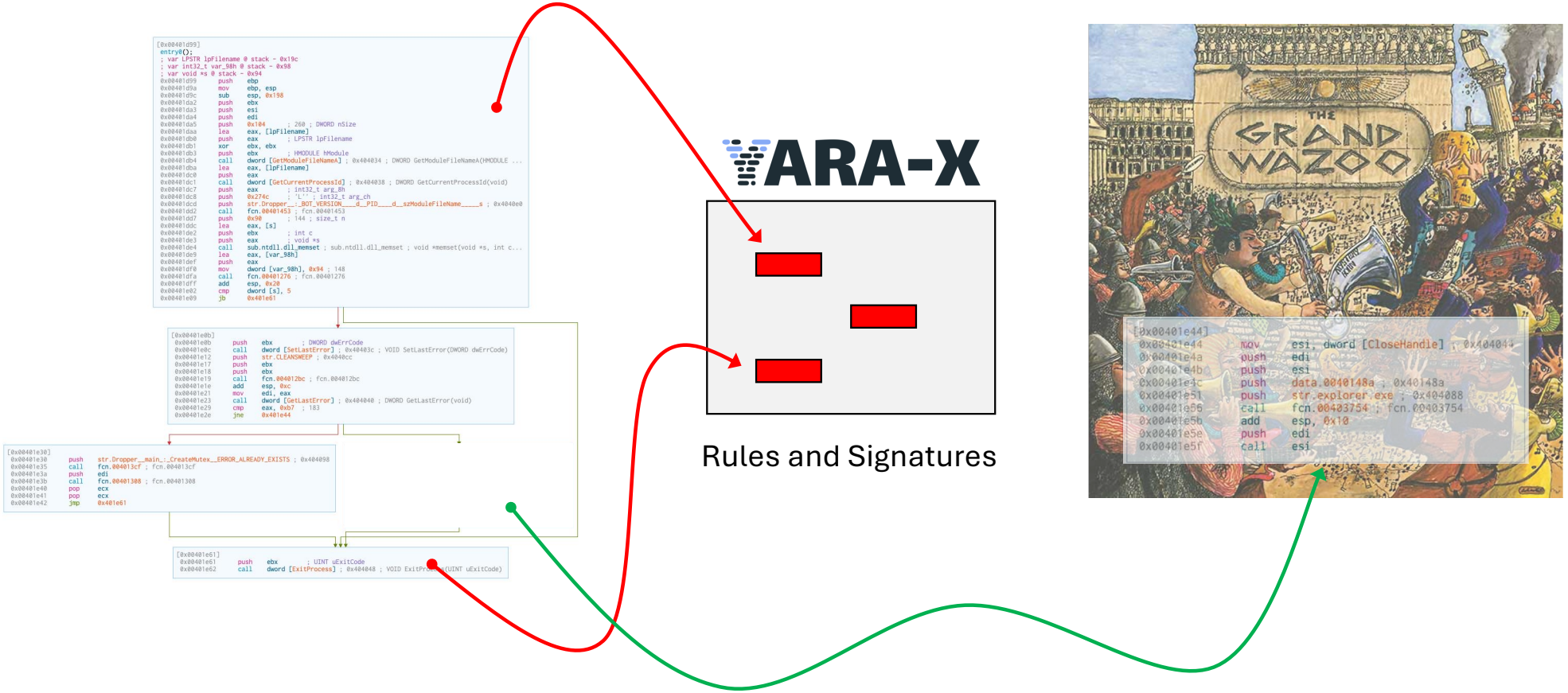
W. Mazurczyk, L. Caviglione, “Information Hiding as a Challenge for Malware Detection”, IEEE Security & Privacy, Vol. 13, No. 2, pp. 89-93, Mar.-Apr. 2015
L. Caviglione, W. Mazurczyk, “Never Mind the Malware, Here’s the Stegomalware”, IEEE Security & Privacy, Vol. 20, No. 5, pp. 101-106, Sept.-Oct. 2022

Stegomalware Targeting Digital Media



Attack Model

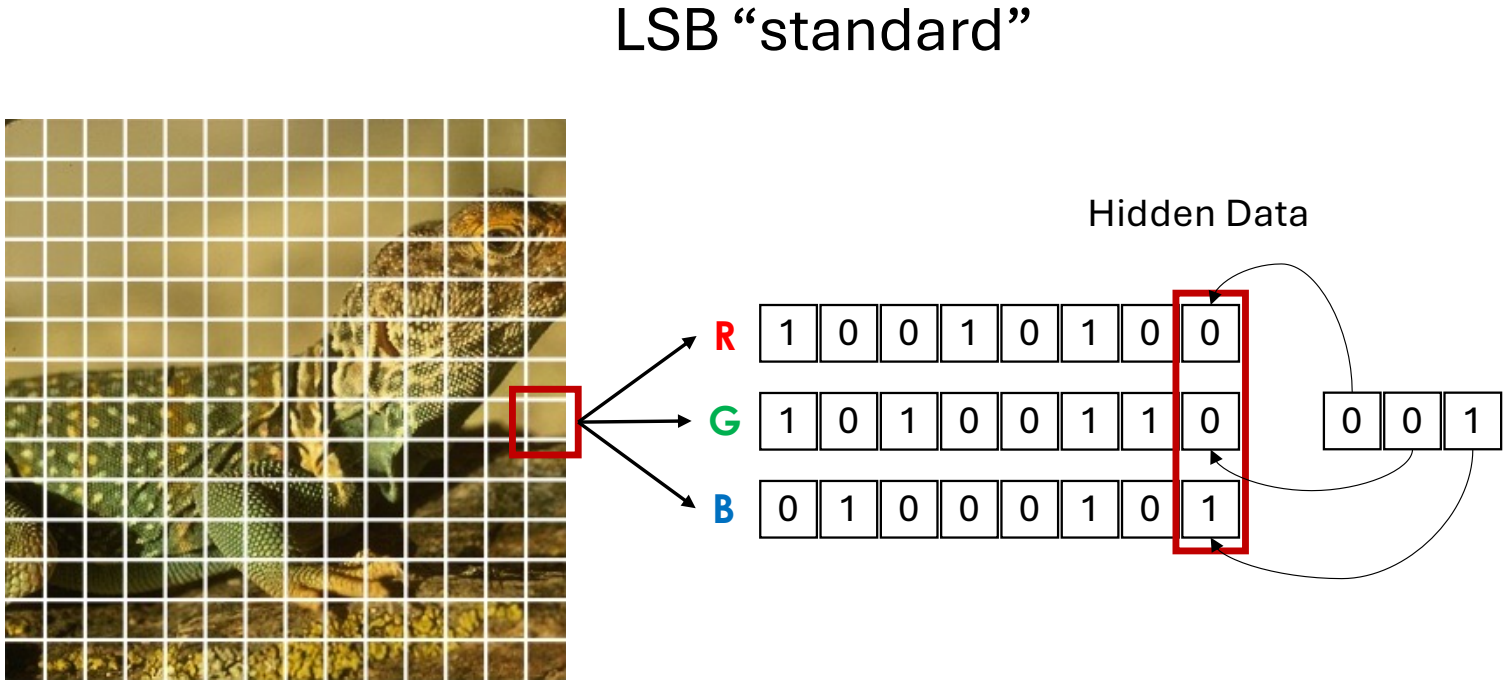
Stegomalware Targeting Digital Media



Attack Model

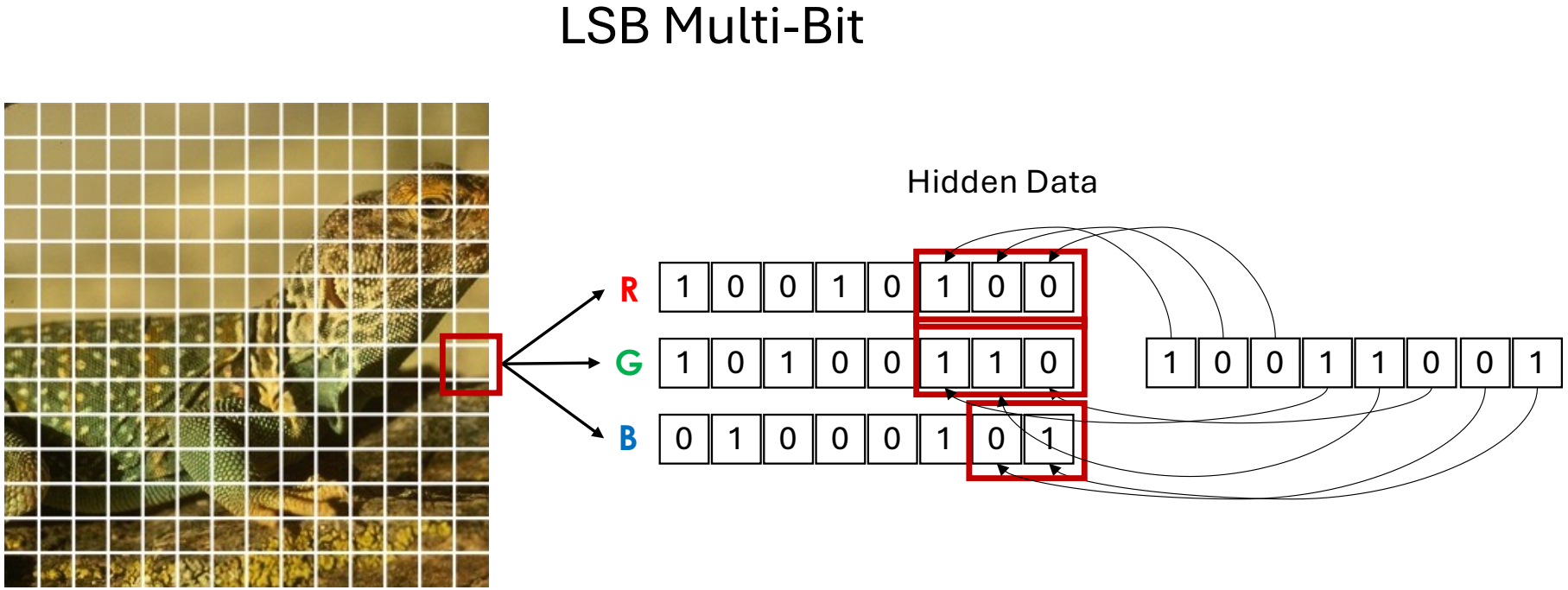
LSB Steganography

- Paradigmatic example: cloaking data in digital pictures



LSB Steganography

- Paradigmatic example: cloaking data in digital pictures



LSB Steganography: Example

- Data hidden by considering 1 Bit – **Red** Channel:

Original Image

Altered Image

LSB Steganography: Example

- Data hidden by considering 1 Bit – **Red** Channel:

Blue Channel
(Grayscale)

Original Image

Altered Image

LSB Steganography: Example

- Data hidden by considering 1 Bit – **Red** Channel:

Green Channel
(Grayscale)

Original Image

Altered Image

LSB Steganography: Example

- Data hidden by considering 1 Bit – **Red** Channel:

Red Channel
(Grayscale)

Original Image

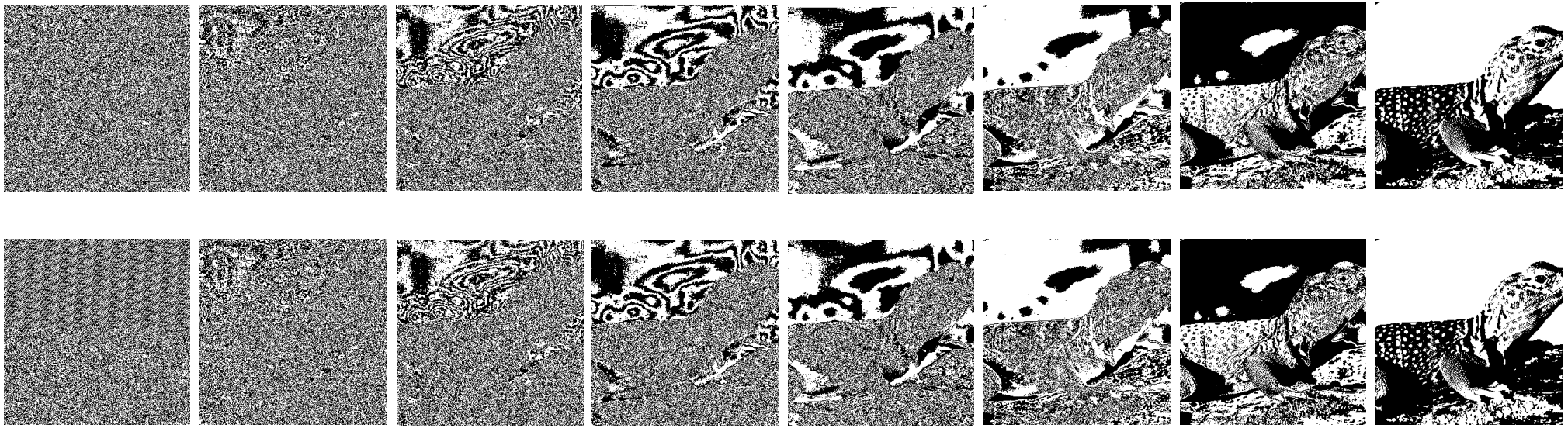
Altered Image

LSB Steganography: Example

- Data hidden by considering 1 Bit – **Red** Channel:

LSB

MSB

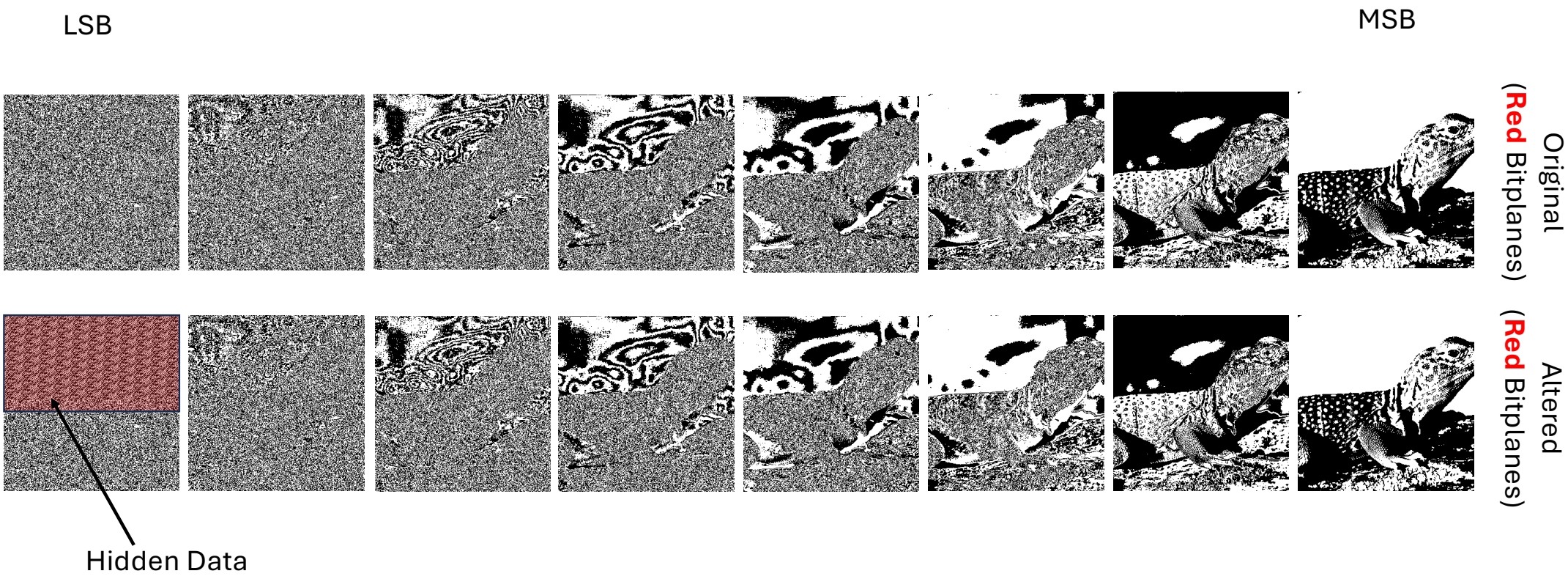


Original (**Red** Bitplanes)

Altered (**Red** Bitplanes)

LSB Steganography: Example

- Data hidden by considering 1 Bit – **Red** Channel:



Stegomalware Example: ZeusVM

- Discovered in 2014, it is an evolution of the Zeus/Zbot malware.
- A variant has been also used in the Hammertoss APT isolated in 2015.
- Attack phases:
  - the malware downloads an innocent JPG from a C&C server

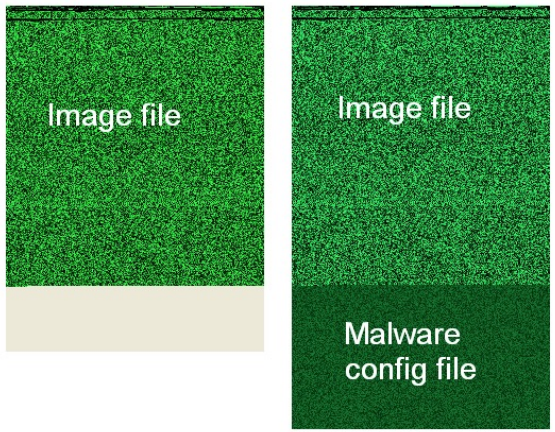
Stegomalware Example: ZeusVM

- Discovered in 2014, it is an evolution of the Zeus/Zbot malware.
- A variant has been also used in the Hammertoss APT isolated in 2015.
- Attack phases:
  - the malware downloads an innocent JPG from a C&C server

Source: https://blog.malwarebytes.com/threat-analysis/2014/02/hiding-in-plain-sight-a-story-about-a-sneaky-banking-trojan/

Stegomalware Example: ZeusVM

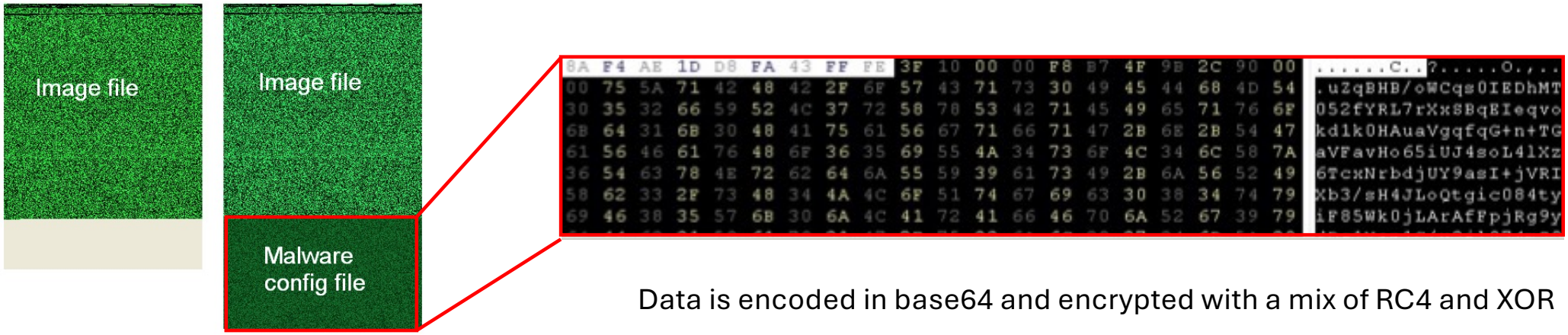
- Discovered in 2014, it is an evolution of the Zeus/Zbot malware.
- A variant has been also used in the Hammertoss APT isolated in 2015.
- Attack phases:
  - the malware downloads an innocent JPG from a C&C server
  - the image perfectly works but a configuration file is appended



Source: https://blog.malwarebytes.com/threat-analysis/2014/02/hiding-in-plain-sight-a-story-about-a-sneaky-banking-trojan/

Stegomalware Example: ZeusVM

- Attack phases:
  - the malware downloads an innocent JPG from a C&C server
  - the image perfectly works but a configuration file is appended



Data is encoded in base64 and encrypted with a mix of RC4 and XOR

Source: https://blog.malwarebytes.com/threat-analysis/2014/02/hiding-in-plain-sight-a-story-about-a-sneaky-banking-trojan/

Stegomalware Example: ZeusVM

- Attack phases:
  - the malware downloads an innocent JPG from a C&C server
  - the image perfectly works but a configuration file is appended

Source: https://blog.malwarebytes.com/threat-analysis/2014/02/hiding-in-plain-sight-a-story-about-a-sneaky-banking-trojan/

Stegomalware Example: ZeusVM

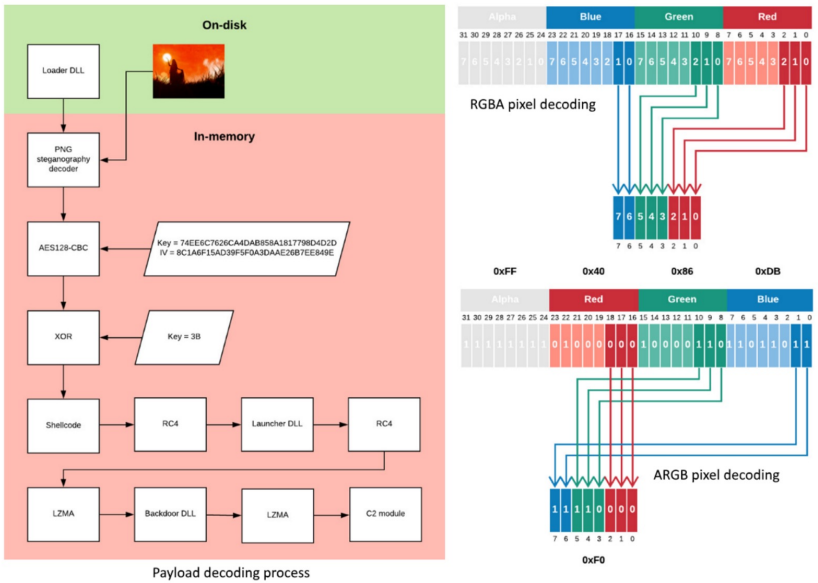
- Attack phases:
  - the malware downloads an innocent JPG from a C&C server
  - the image perfectly works but a configuration file is appended
  - trojan activates when traffic with the financial institutions provided in the configuration file is sensed
  - it steals user credentials by acting in a MitM fashion.

Stegomalware Example: OceanLotus

- Discovered in 2018/2019 (and used in several attack campaigns).
- The encrypted payload is concealed within a PNG image to elude detection (e.g., signature-based).
- Attack Phases:

Stegomalware Example: OceanLotus

- Discovered in 2018/2019 (and used in several attack campaigns).
- The encrypted payload is concealed within a PNG image to elude detection (e.g., signature-based).
- Attack Phases:



Stegomalware Example: Invoke-PSImage

- Invoke-PSImage is a tool for encoding a PowerShell Script in pixels of a PNG image.
- It uses Least Significant Bit (LSB) steganography.
- Invoke-PSImage has been released in Dec. 2017 and it has been used for a malware campaign just 1 week later.
- Example:
  - Mimikatz
  - Ursnif

Invoke-PSImage: https://github.com/peewpw/Invoke-PSImage

Stegomalware Example: Invoke-PSImage

- Invoke-PSImage is a tool for encoding a PowerShell Script in pixels of a PNG image.
- It uses Least Significant Bit (LSB) steganography.
- Invoke-PSImage has been released in Dec. 2017 and it has been used for a malware campaign just 1 week later.
- Attack Phases:
  - infected Excel is used to launch a malicious VB macro

Invoke-PSImage: https://github.com/peewpw/Invoke-PSImage

Stegomalware Example: Invoke-PSImage

- Invoke-PSImage is a tool for encoding a PowerShell Script in pixels of a PNG image.
- It uses Least Significant Bit (LSB) steganography.
- Invoke-PSImage has been released in Dec. 2017 and it has been used for a malware campaign just 1 week later.
- Attack Phases:
  - infected Excel is used to launch a malicious VB macro
  - the macro downloads an image containing a PowerShell script

Invoke-PSImage: https://github.com/peewpw/Invoke-PSImage

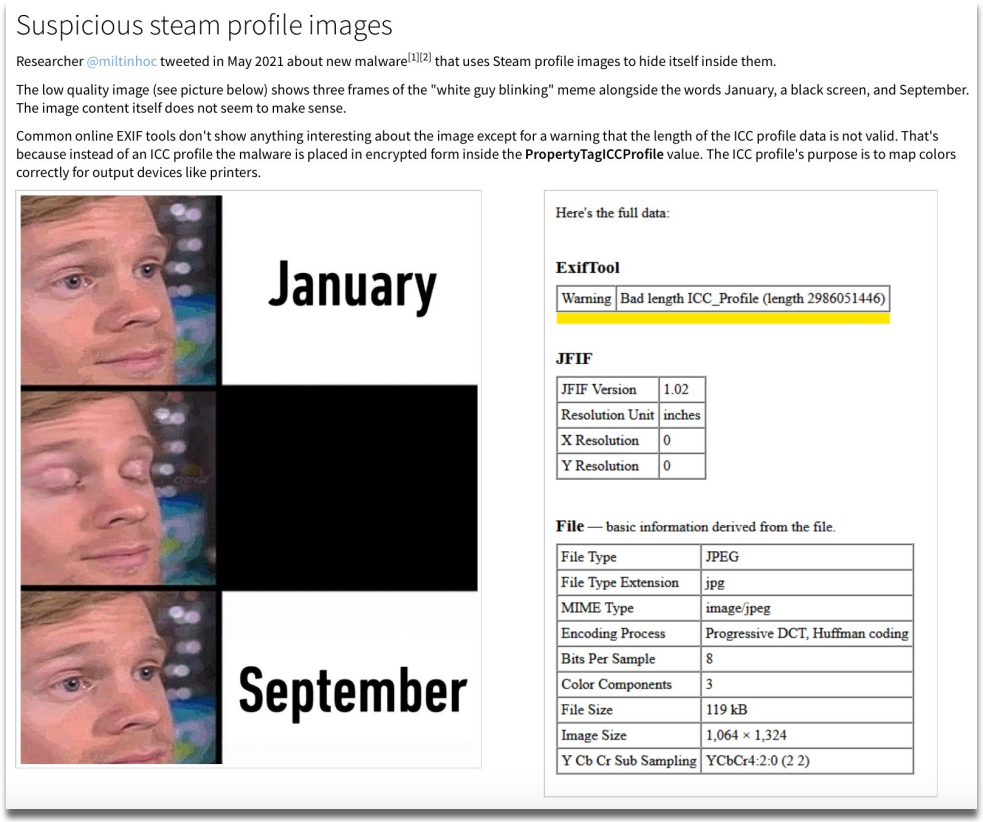
Stegomalware Example: Invoke-PSImage

- Invoke-PSImage is a tool for encoding a PowerShell Script in pixels of a PNG image.
- It uses Least Significant Bit (LSB) steganography.
- Invoke-PSImage has been released in Dec. 2017 and it has been used for a malware campaign just 1 week later.
- Attack Phases:
  - infected Excel is used to launch a malicious VB macro
  - the macro downloads an image containing a PowerShell script
  - the script is extracted and launched to retrieve the Ursnif loader.

Invoke-PSImage: https://github.com/peewpw/Invoke-PSImage

Stegomalware Example: SteamHide

- In mid 2021, the “white guy blinking” meme has been used to conceal malicious information on the Steam store.
- Data was hidden within the **PropertyTagICCProfile** value.
- Steam is just used a “storage” for the malware.
- Try(*):
  - *brew install exiftool*



Suspicious steam profile images

Researcher @miltinhoc tweeted in May 2021 about new malware[1][2] that uses Steam profile images to hide itself inside them.

The low quality image (see picture below) shows three frames of the "white guy blinking" meme alongside the words January, a black screen, and September. The image content itself does not seem to make sense.

Common online EXIF tools don't show anything interesting about the image except for a warning that the length of the ICC profile data is not valid. That's because instead of an ICC profile the malware is placed in encrypted form inside the **PropertyTagICCProfile** value. The ICC profile's purpose is to map colors correctly for output devices like printers.

Here's the full data:

ExifTool

Warning	Bad length ICC_Profile (length 2986051446)

JFIF

JFIF Version	1.02
Resolution Unit	inches
X Resolution	0
Y Resolution	0

File — basic information derived from the file.

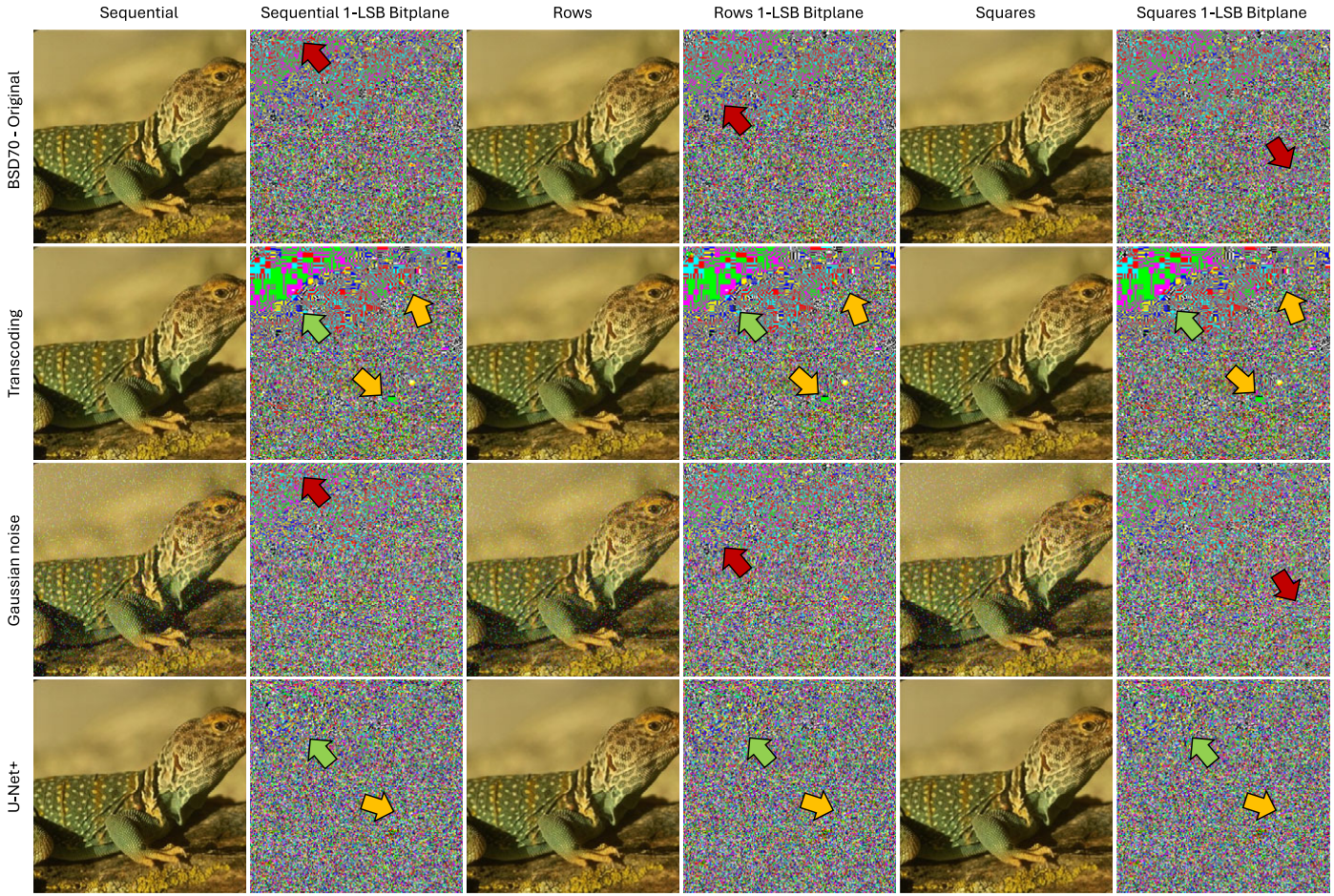
File Type	JPEG
File Type Extension	jpg
MIME Type	image/jpeg
Encoding Process	Progressive DCT, Huffman coding
Bits Per Sample	8
Color Components	3
File Size	119 kB
Image Size	1,064 × 1,324
Y Cb Cr Sub Sampling	YCbCr4:2:0 (2 2)

SteamHide: https://www.gdatasoftware.com/blog/steamhide-malware-in-profile-images
(*) https://github.com/lucacav/steg-tools

Detection and Mitigation

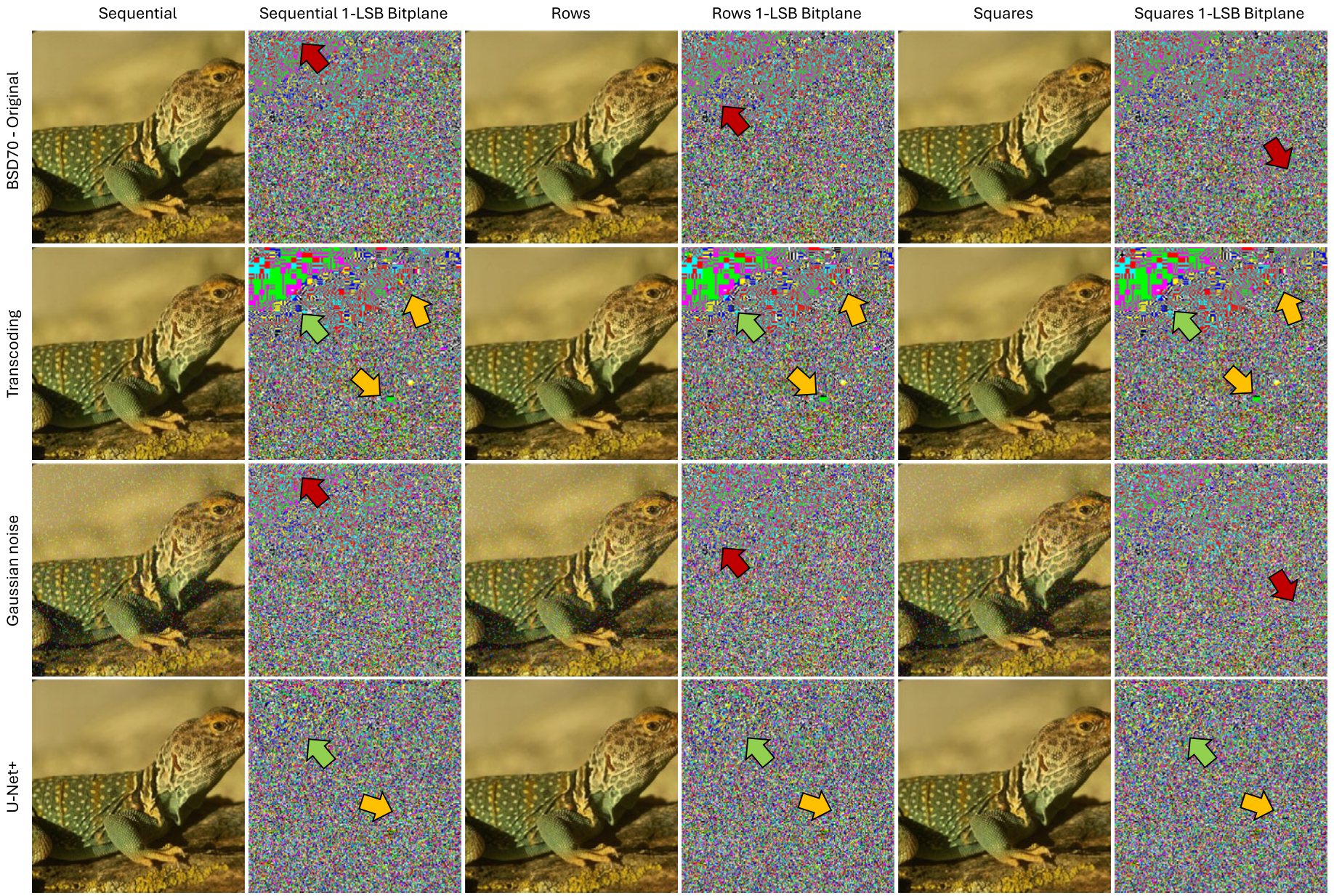
- Literature abounds in detection and mitigation techniques against multimedia steganography.
- Similar to the case of (network) covert channels:
  - **statistical methods**: recognize traits that can reveal the presence of hidden data (e.g., color histogram)
  - **AI**: trained classifiers capable of distinguishing between legitimate and altered media.
- When detection is not possible, an approach is:
  - **sanitization**: try to remove the hidden payload or at least render it unusable for the attacker.

Detection and Mitigation: Sanitization



Legend:

- Malicious hidden content
- Artifacts of the sanitization
- Major differences


Sequential
Sequential 1-LSB Bitplane
Rows
Rows 1-LSB Bitplane
Squares
Squares 1-LSB Bitplane
BSD70 - Original
Transcoding
Gaussian noise
U-Net+

Side-Channel-Attacks

- A side channel is an attack that exploits “extra” information **leaked** by a system, rather than a weakness or a vulnerability.
- Usually, they require some form of **proximity** with the target.
- Great interest in 80/90 for attacking **cryptographic** mechanisms.
- Today widely used in:
  - networked/virtualized systems
  - software objects.

Side-Channel-Attacks

- A side channel is an attack that exploits “extra” information **leaked** by a system, rather than a weakness or a vulnerability.
- Usually, they require some form of **proximity** with the target.
- Great interest in 80/90 for attacking **cryptographic** mechanisms.
- Today widely used in:
  - networked/virtualized systems
  - software objects.

Examples

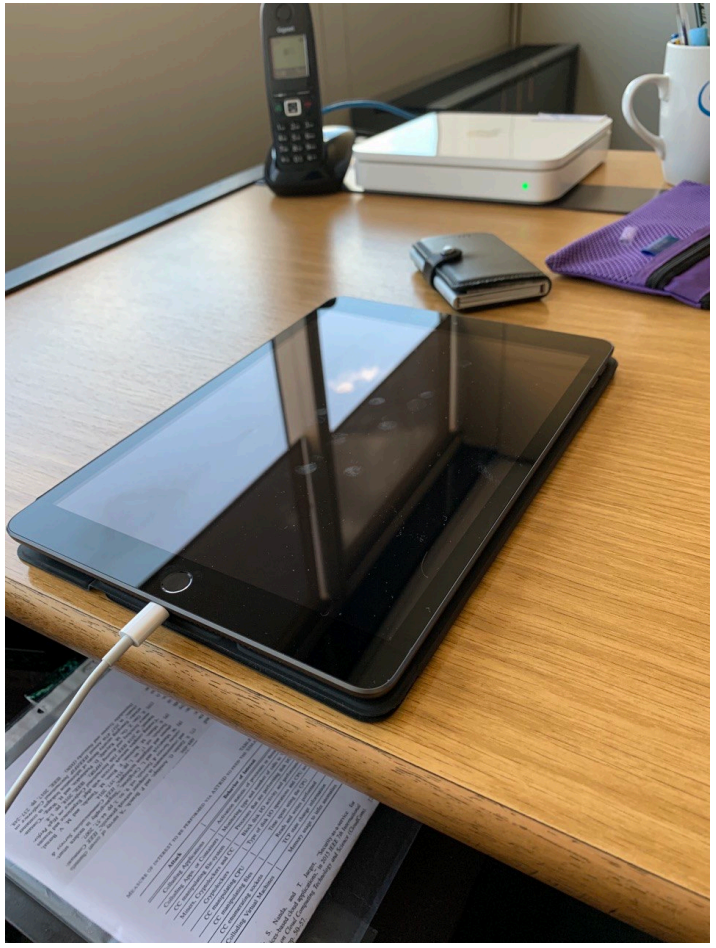
Temperature: to guess what a device or a program is doing.

Time: to understand whether traffic has been processed by a security tool or routed through a middlebox.

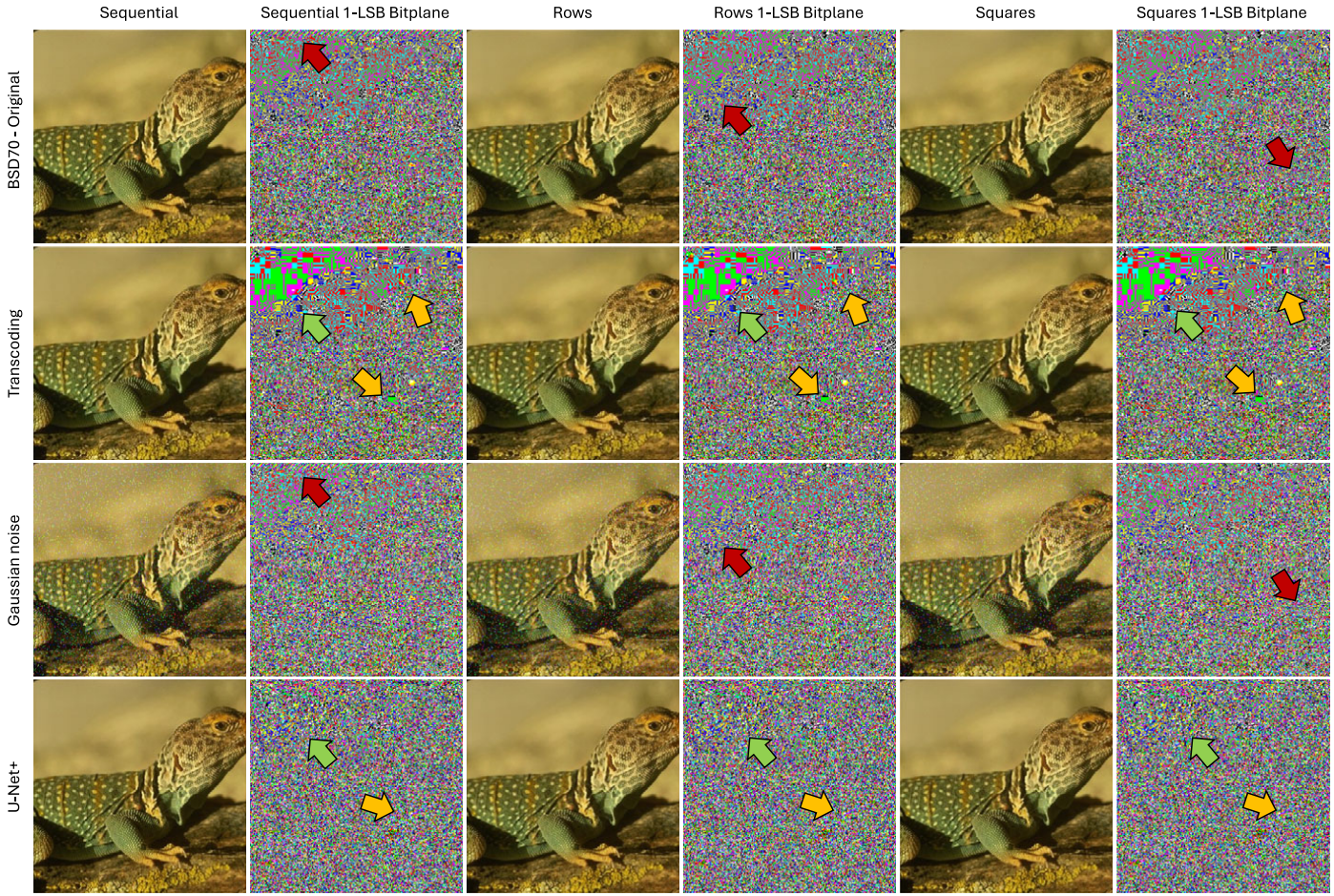
EM Emissions: to identify specific instruction patterns in GPUs.

Side-Channel-Attacks

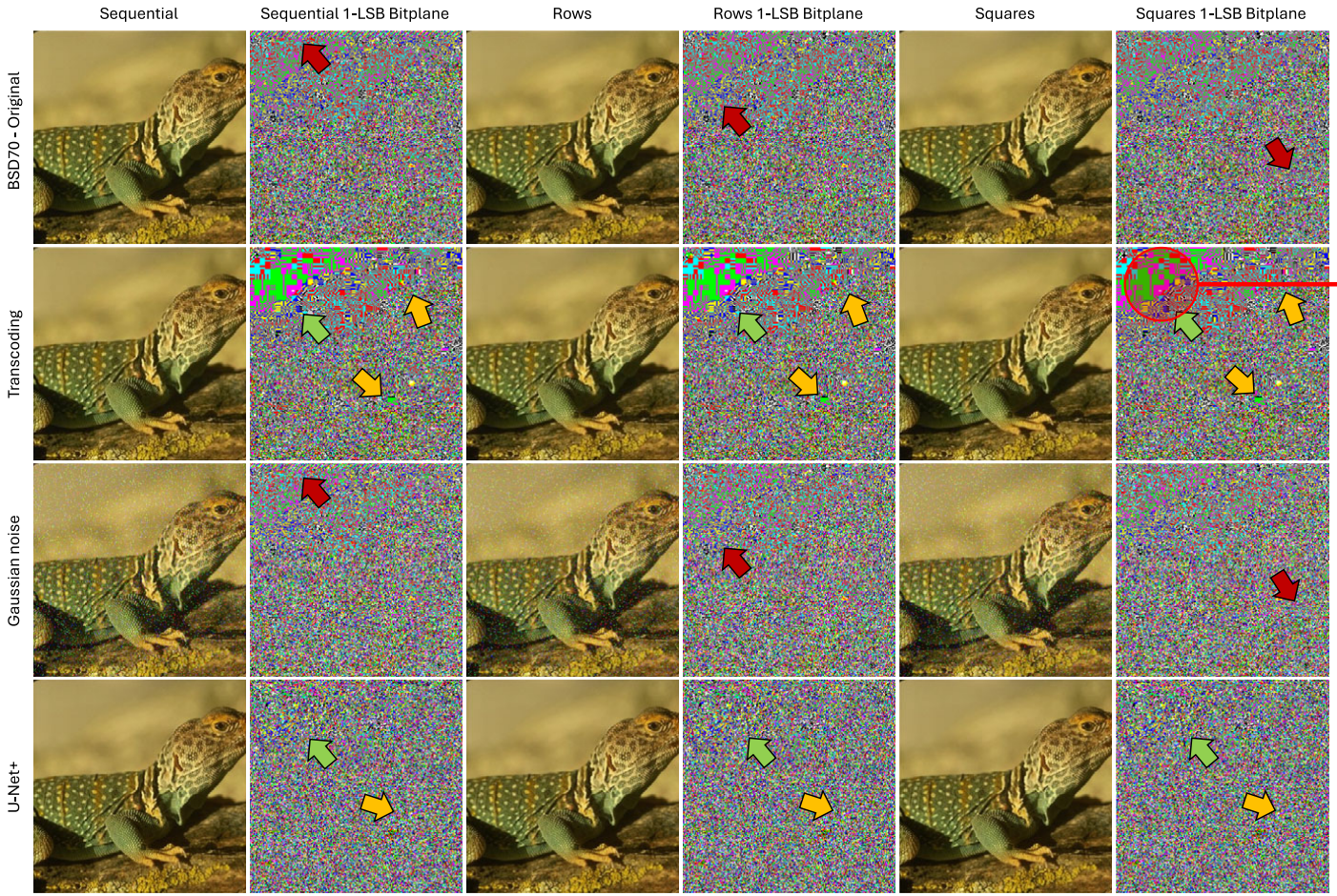
- A side channel is an attack that exploits “extra” information **leaked** by a system, rather than a weakness or a vulnerability.
- Usually, they require some form of **proximity** with the target.
- Great interest in 80/90 for attacking **cryptographic** mechanisms.
- Today widely used in:
  - networked/virtualized systems
  - software objects.



Detection and Mitigation: Sanitization



Detection and Mitigation: Sanitization



side-channel information

Defensive Use of Information Hiding

- **Explosion** of **digital information**, for instance:
  - *digital media*: audio, video, pictures, and text
  - *software*: source code, binaries, and libraries
  - *medical data*: imaging and 3D-printed parts
  - *hardware*: FPGA and custom silicon
  - *artificial intelligence*: datasets, models, and outputs.
- **Digital information** requires (efficient) **mechanisms** for:
  - copyright protection
  - fingerprinting and tracking
  - integrity
  - authentication
  - annotation/metadata.

Watermarking

- **Watermarking** is about **embedding** information into original data to:
  - claim something, e.g., copyright/ownership
  - reveal manipulations or tampering attempts
  - tracking the diffusion of information
  - **it must not affect the usability**.
- **Watermarking** and **Information Hiding**:
  - watermarking can be viewed as a "**good**" steganography application
  - largely **overlapping** theory, e.g., carrier, magic triangle, etc.
  - differences in **terminology**
  - several **application-dependent** variations.

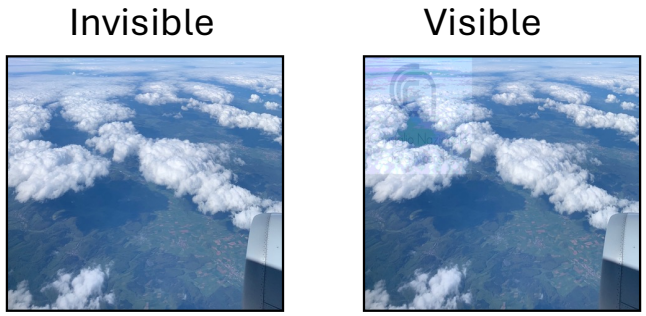
Properties of the Watermark

- Watermarking mechanisms have many **properties/requirements**
- Basic:
  - **capacity**: how much information can be embedded
  - **robustness**: how the data can withstand (un)wanted removal attempts
  - **secrecy**: how difficult is to spot the secret information
- Other(*):
  - **fidelity**: how much degradation is caused to the carrier
  - **reliability**: owner can identify secrets with a high probability
  - **integrity**: owner rarely accuses another owner, i.e., low false alarm
  - **efficiency**: the price paid for embedding/verifying the watermark
  - **generality**: is the mechanism dependent on a specific carrier?

(*) F. Boenisch, “A Systematic Review on Model Watermarking for Neural Networks”, Frontiers in Big Data, Vol. 4, pp. 4:1 – 4:16, Nov. 2021

Properties of the Watermark

- Watermarks can be also classified according to their visibility:
  - **invisible**
  - **visible**.



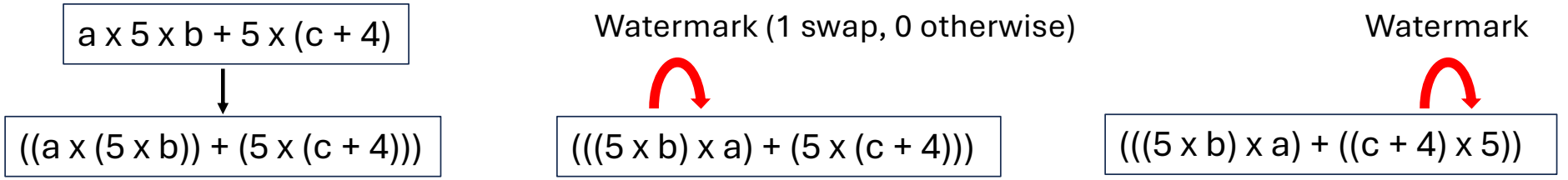
- The goal of the watermark also plays a role:
  - **robust**: suitable if the carrier is expected to be processed multiple times (e.g., transcoding) or attacked by a malicious actor
  - **fragile:** suitable in case minimal alterations of the carrier need to be revealed or tracked.

Watermark Example: Software

- **Software** is today a **critical** asset (obviously):
  - piracy
  - reverse engineering
  - IP infringements.
- With the advent of **AI**-based **code generators** or **assistants**:
  - new problems due to the ingestion of source code!
  - ability of generating malicious/unaudited code is magnified!
- Two main mechanisms for marking the software:
  - **static**: the watermark is hidden in code or assets (e.g., icons)
  - **dynamic**: the watermark is stored in the execution state of a program.

Watermark Example: Software

- **Three** main exemplary sets of techniques
- Easter Egg:
  - simple and “catchy”
  - easy to locate
  - *example*: *about:mozilla*
- Code substitution (or replacement):
  - general idea: substitute a portion of the code with a watermarked value
  - *simple example*: re-ordering of mathematical operations.

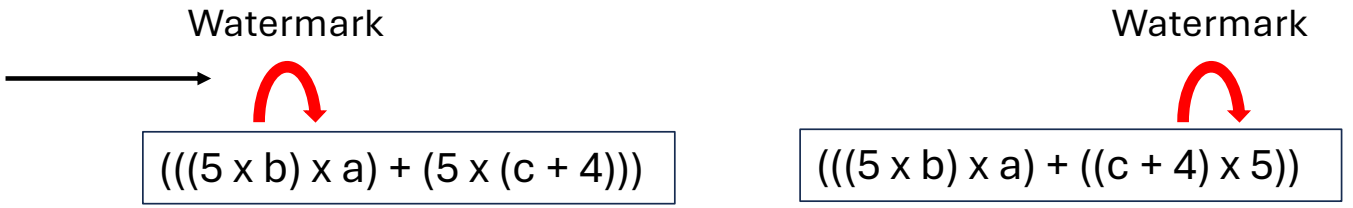


J. Hamilton, S. Danicic, “A Survey of Static Software Watermarking”, World Congress on Internet Security, London, UK, pp. 100-107, Feb. 2011

Watermark Example: Software

- **Three** main exemplary sets of techniques
- Easter Egg:
  - simple and “catchy”
  - easy to locate
  - *example*: *about:mozilla*
- Code substitution (or replacement):
  - general idea: substitute a portion of the code with a watermarked value
  - *simple example*: re-ordering of mathematical operations.

Must be safe swappable!
Correctness (e.g., a++ vs ++a)
Check code optimizations!



J. Hamilton, S. Danicic, “A Survey of Static Software Watermarking”, World Congress on Internet Security, London, UK, pp. 100-107, Feb. 2011

Watermark Example: Software

- **Three** main exemplary sets of techniques
- Easter Egg:
  - simple and “catchy”
  - easy to locate
  - *example*: *about:mozilla*
- Code substitution (or replacement):
  - general idea: substitute a portion of the code with a watermarked value
  - *simple example*: re-ordering of mathematical operations.
- Opaque predicates:
  - borrowed from software protection and obfuscation.

Opaque Predicates

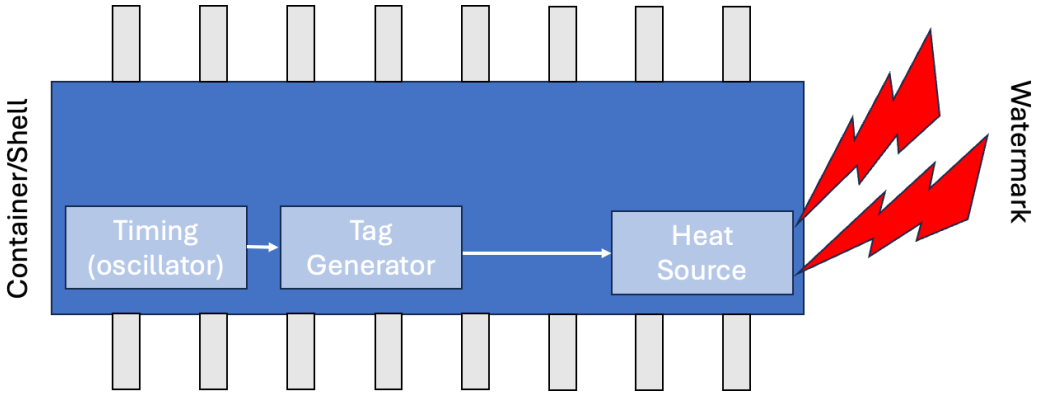
- Key idea:
  - create an expression that evaluates to either “true” or “false”
  - the outcome is **known** a-priori by the **programmer**
  - needs to be evaluated at run time
  - **difficult** to **understand** the intent of the code.

Opaque predicate
(d is always 8)!

Watermark

Watermark Example: Hardware

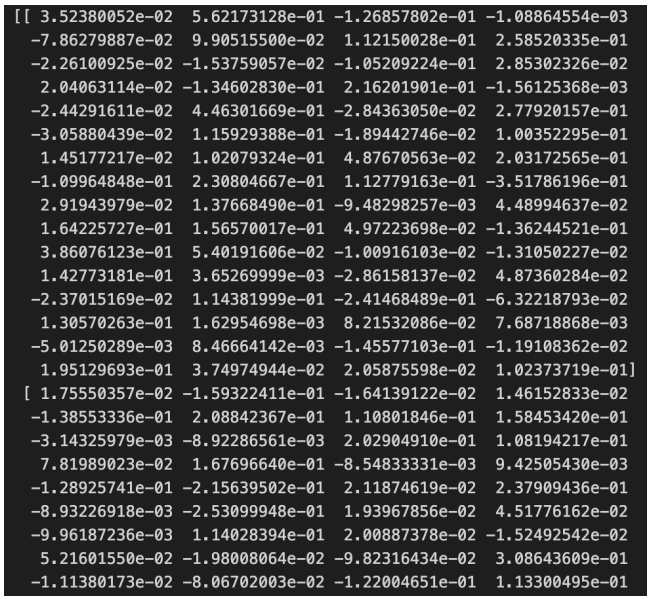
- Custom hardware and “silicon” are now at the basis of:
  - IoT nodes
  - mobile/personal devices
  - CPU/GPU
  - AI-oriented architectures.
- Only mentioned a **simple** example.



C.-H. Chang, M. Potkonjak, L. Zhang, “Hardware IP Watermarking and Fingerprinting”, Secure System Design and Trustable Computing, pp. 329-368, Springer, 2016

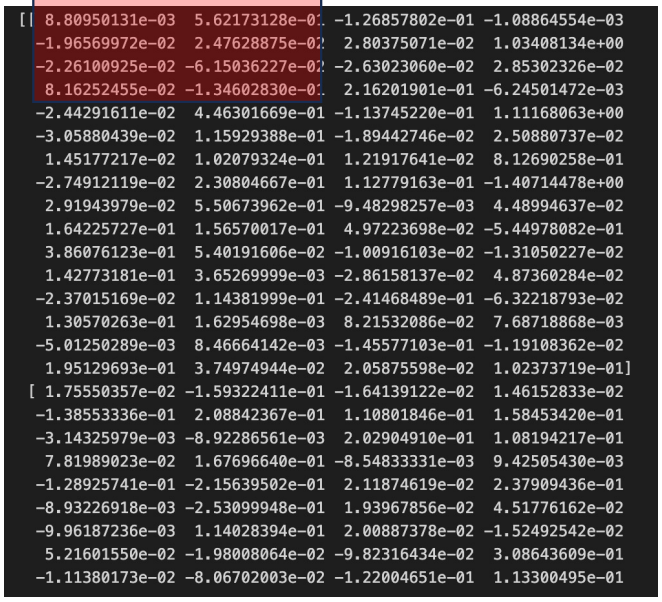
Watermark Example: AI Models

- Key idea – LSB Steganography:



Original Weights
(Trained Model)

Watermark



Watermarked Weights
(Trained Model)

Y. Nagai, Y Uchida, S. Sakazawa, “Digital Watermarking for Deep Neural Networks”, International Journal of Multimedia Information Retrieval, Vol. 7, pp. 3–16, Feb. 2018

Watermark Example: AI Models

- Key idea – LSB Steganography:

```
[[ 3.52380052e-02  5.62173128e-01 -1.26857802e-01 -1.08864554e-03
  -7.86279887e-02  9.90515500e-02  1.12150028e-01  2.58520335e-01
  -2.26100925e-02 -1.53759057e-02 -1.05209224e-01  2.85302326e-02
   2.04063114e-02 -1.34602830e-01  2.16201901e-01 -1.56125368e-03
  -2.44291611e-02  4.46301669e-01 -2.84363050e-02  2.77920157e-01
  -3.05880439e-02  1.15929388e-01 -1.89442746e-02  1.00352295e-01
   1.45177217e-02  1.02079324e-01  4.87670563e-02  2.03172565e-01
  -1.09964848e-01  2.30804667e-01  1.12779163e-01 -3.51786196e-01
   2.91943979e-02  1.37668490e-01 -9.48298257e-03  4.48994637e-02
   1.64225727e-01  1.56570017e-01  4.97223698e-02 -1.36244521e-01
   3.86076123e-01  5.40191606e-02 -1.00916103e-02 -1.31050227e-02
   1.42773181e-01  3.65269999e-03 -2.86158137e-02  4.87360284e-02
  -2.37015169e-02  1.14381999e-01 -2.41468489e-01 -6.32218793e-02
   1.30570263e-01  1.62954698e-03  8.21532086e-02  7.68718868e-03
  -5.01250289e-03  8.46664142e-03 -1.45577103e-01 -1.19108362e-02
   1.95129693e-01  3.74974944e-02  2.05875598e-02  1.02373719e-01]
 [ 1.75550357e-02 -1.59322411e-01 -1.64139122e-02  1.46152833e-02
  -1.38553336e-01  2.08842367e-01  1.10801846e-01  1.58453420e-01
  -3.14325979e-03 -8.92286561e-03  2.02904910e-01  1.08194217e-01
   7.81989023e-02  1.67696640e-01 -8.54833331e-03  9.42505430e-03
  -1.28925741e-01 -2.15639502e-01  2.11874619e-02  2.37909436e-01
  -8.93226918e-03 -2.53099948e-01  1.93967856e-02  4.51776162e-02
  -9.96187236e-03  1.14028394e-01  2.00887378e-02 -1.52492542e-02
   5.21601550e-02 -1.98008064e-02 -9.82316434e-02  3.08643609e-01
  -1.11380173e-02 -8.06702003e-02 -1.22004651e-01  1.13300495e-01
```

Original Weights
(Trained Model)

MNIST Dataset

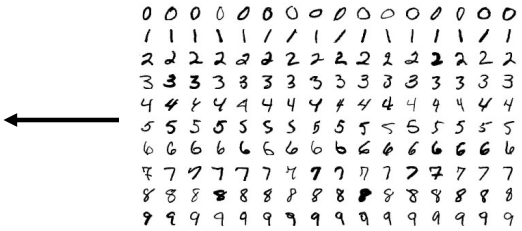
Accuracy: 98.58%

Watermark Example: AI Models

- Key idea – LSB Steganography:

```
[[ 8.80950131e-03  5.62173128e-01 -1.26857802e-01 -1.08864554e-03
  -1.96569972e-02  2.47628875e-02  2.80375071e-02  1.03408134e+00
  -2.26100925e-02 -6.15036227e-02 -2.63023060e-02  2.85302326e-02
   8.16252455e-02 -1.34602830e-01  2.16201901e-01 -6.24501472e-03
  -2.44291611e-02  4.46301669e-01 -1.13745220e-01  1.11168063e+00
  -3.05880439e-02  1.15929388e-01 -1.89442746e-02  2.50880737e-02
   1.45177217e-02  1.02079324e-01  1.21917641e-02  8.12690258e-01
  -2.74912119e-02  2.30804667e-01  1.12779163e-01 -1.40714478e+00
   2.91943979e-02  5.50673962e-01 -9.48298257e-03  4.48994637e-02
   1.64225727e-01  1.56570017e-01  4.97223698e-02 -5.44978082e-01
   3.86076123e-01  5.40191606e-02 -1.00916103e-02 -1.31050227e-02
   1.42773181e-01  3.65269999e-03 -2.86158137e-02  4.87360284e-02
  -2.37015169e-02  1.14381999e-01 -2.41468489e-01 -6.32218793e-02
   1.30570263e-01  1.62954698e-03  8.21532086e-02  7.68718868e-03
  -5.01250289e-03  8.46664142e-03 -1.45577103e-01 -1.19108362e-02
   1.95129693e-01  3.74974944e-02  2.05875598e-02  1.02373719e-01]
 [ 1.75550357e-02 -1.59322411e-01 -1.64139122e-02  1.46152833e-02
  -1.38553336e-01  2.08842367e-01  1.10801846e-01  1.58453420e-01
  -3.14325979e-03 -8.92286561e-03  2.02904910e-01  1.08194217e-01
   7.81989023e-02  1.67696640e-01 -8.54833331e-03  9.42505430e-03
  -1.28925741e-01 -2.15639502e-01  2.11874619e-02  2.37909436e-01
  -8.93226918e-03 -2.53099948e-01  1.93967856e-02  4.51776162e-02
  -9.96187236e-03  1.14028394e-01  2.00887378e-02 -1.52492542e-02
   5.21601550e-02 -1.98008064e-02 -9.82316434e-02  3.08643609e-01
  -1.11380173e-02 -8.06702003e-02 -1.22004651e-01  1.13300495e-01
```

Watermarked Weights
(Trained Model)

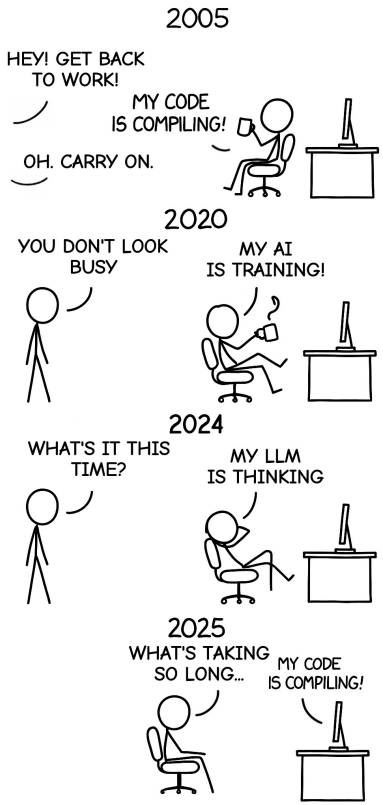


MNIST Dataset

Accuracy: 94.68%

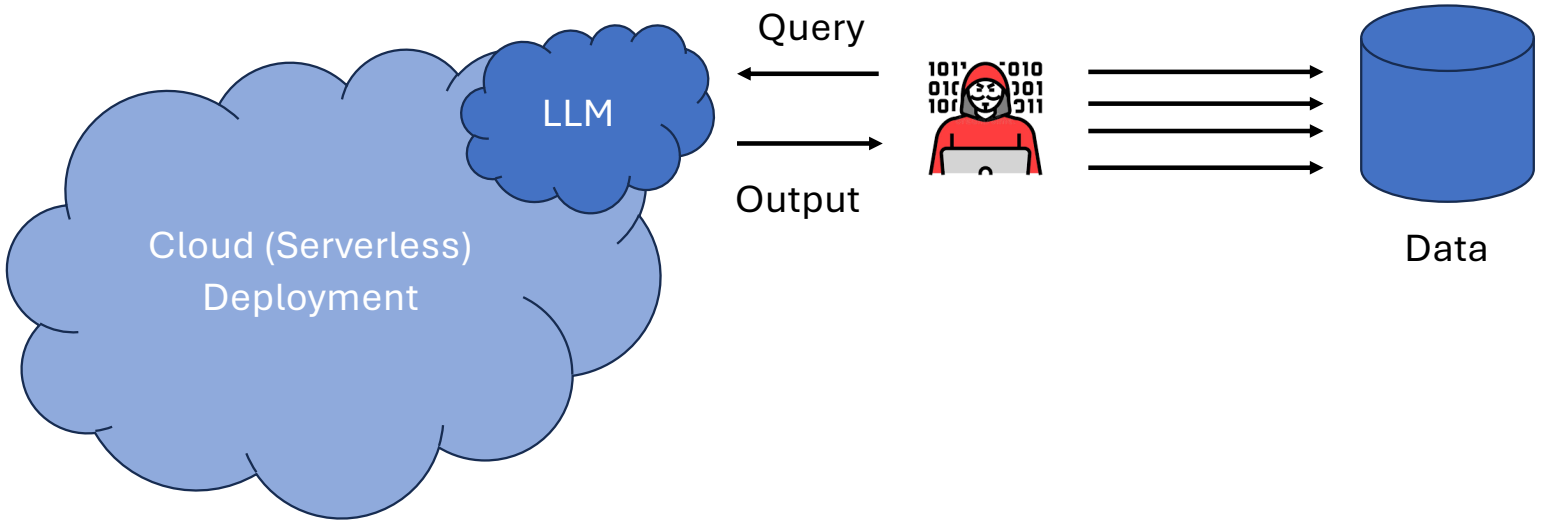
Watermark Example: AI Outputs

- LLMs spawned a huge variety of new services:
  - chatbots
  - financial analysis
  - content creation
  - code generation
  - …
- Commercial LLMs offer access via some APIs:
  - to **avoid training**, which is expensive and time-consuming
  - an attacker can perform **multiple queries** to "imitate" the model.



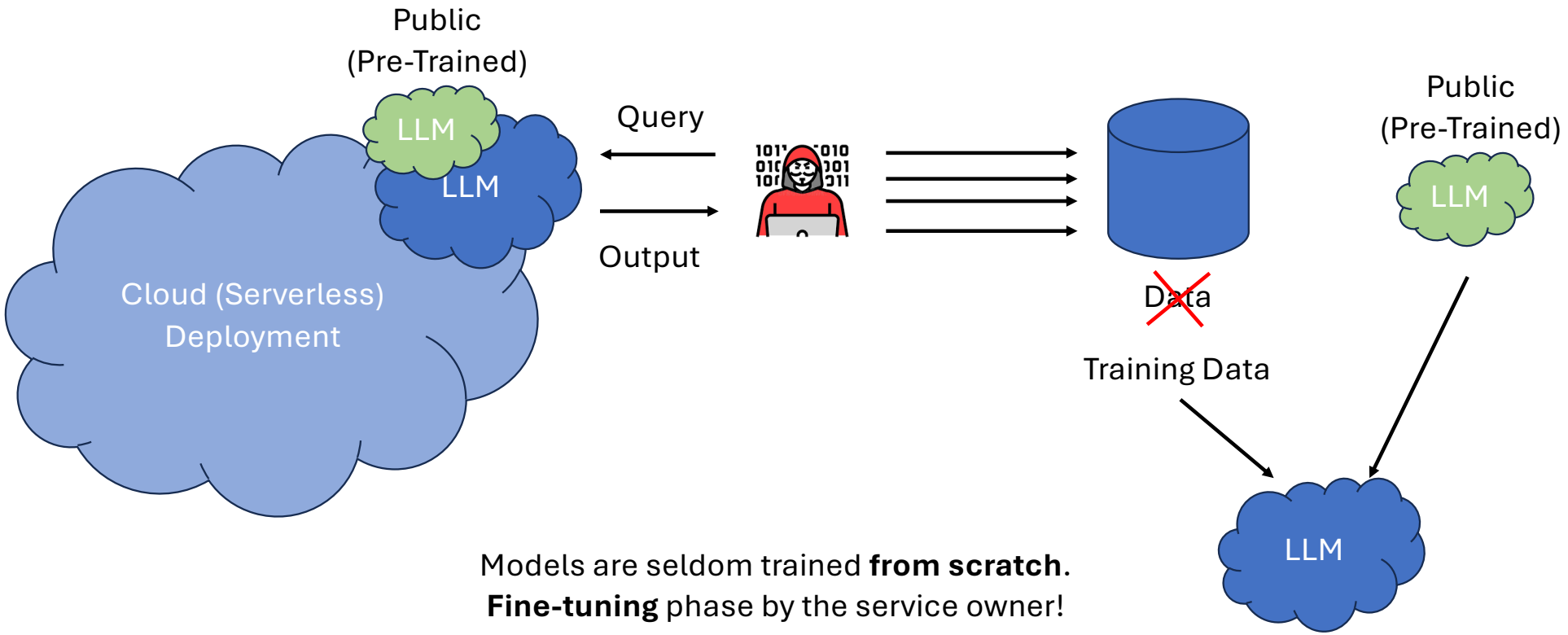
Source: XKCD

Watermark Example: AI Outputs



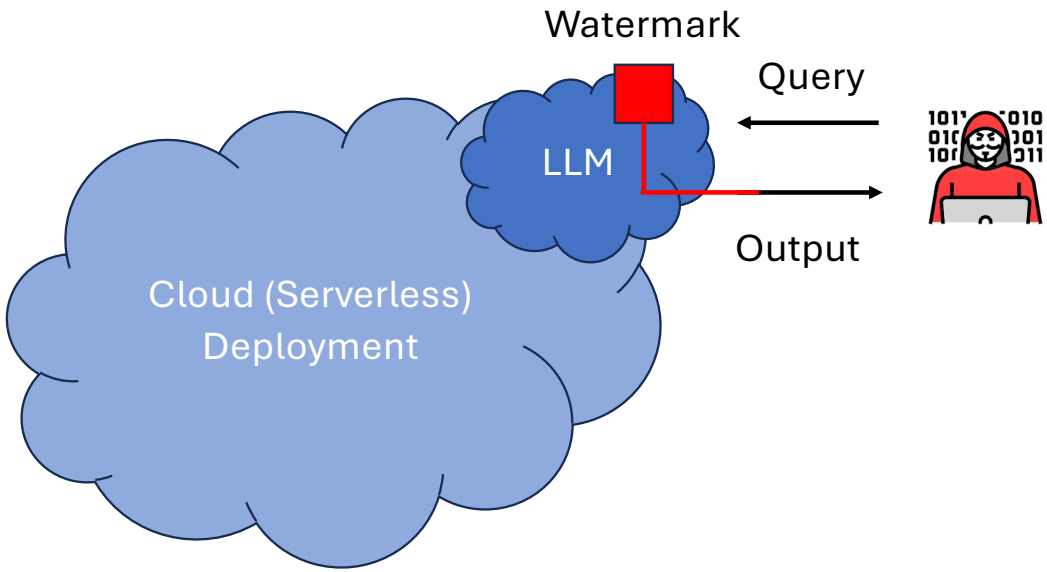
Attack Model

Watermark Example: AI Outputs



Models are seldom trained **from scratch**.
Fine-tuning phase by the service owner!

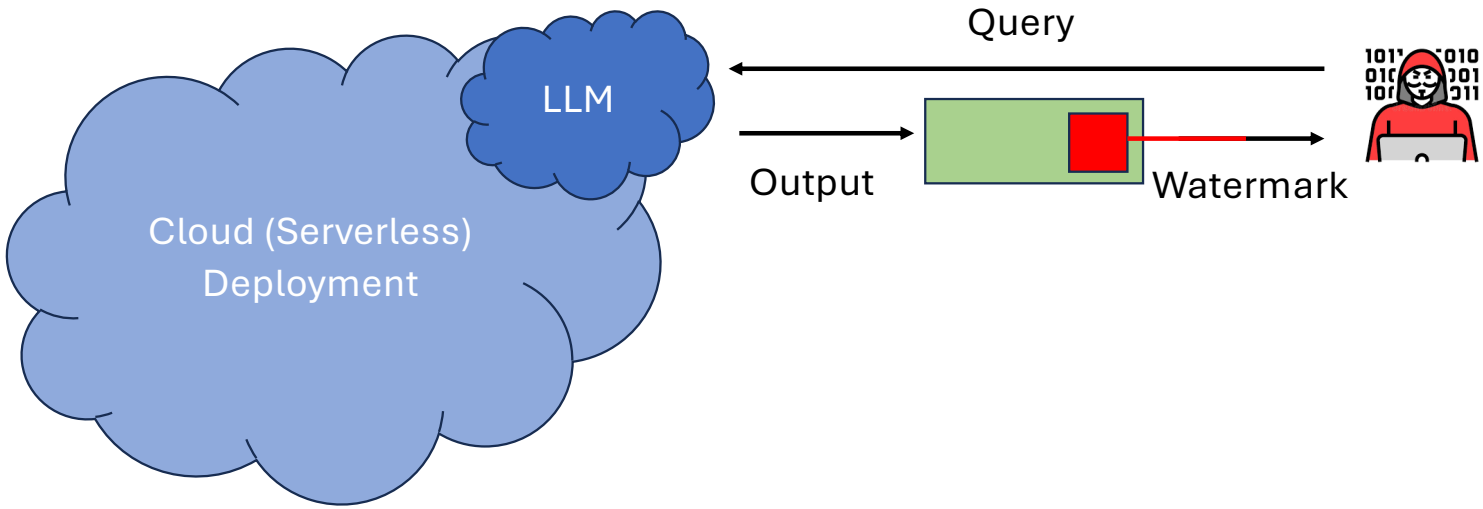
Watermark Example: AI Outputs



"Soft" Watermark

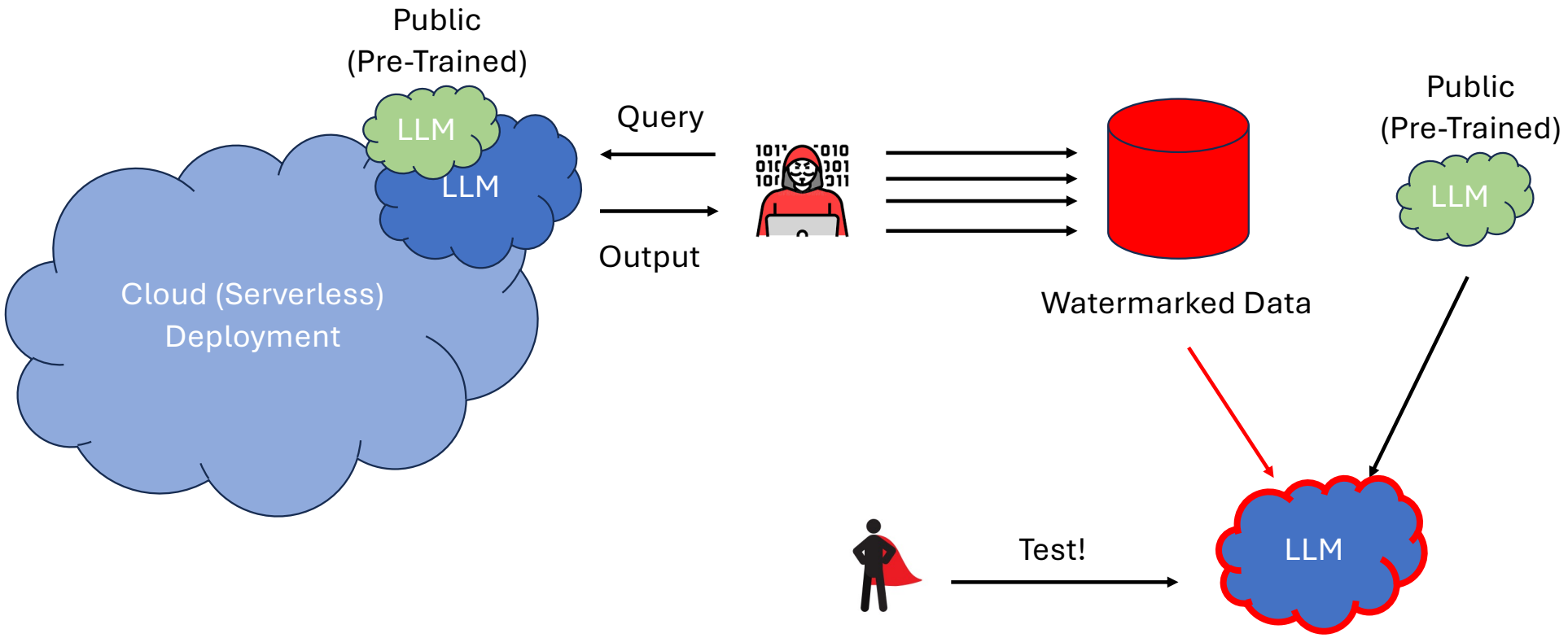
(Requires to intercept and change the model to act in **white-box** manner)

Watermark Example: AI Outputs



“Hard” Watermark
(Considers the LLM/AI a **black-box**)

Watermarking AI Outputs



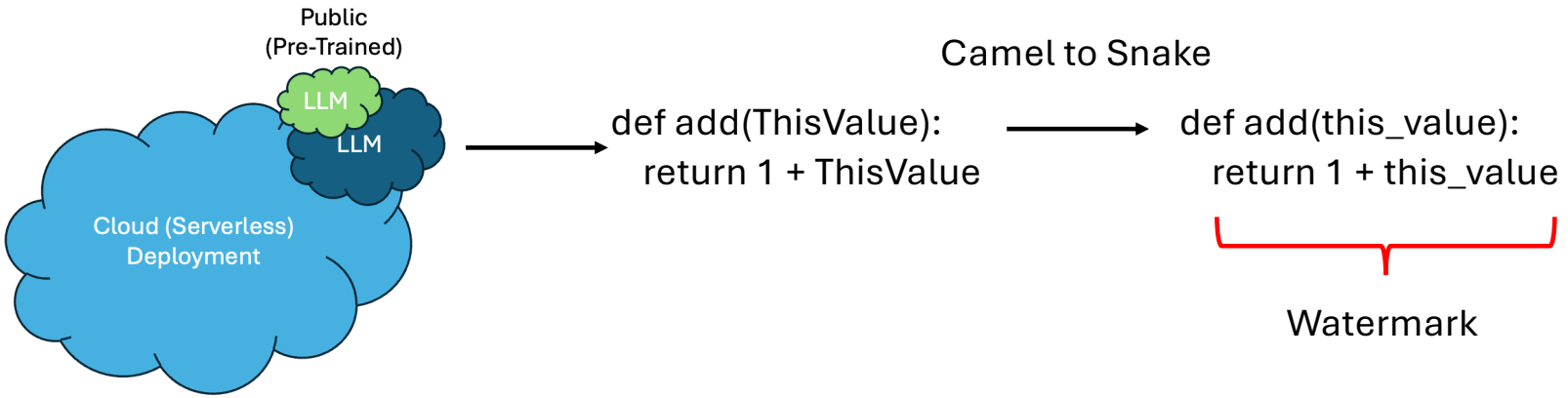
Watermark Example: AI Outputs

- Example of a **black-box** technique:
  - for marking LLM-based code generation tools (very popular!)
  - from Li, Wang, Wang and Gao.
- Key idea:
  - AI-generated code is changed by replacing tokens with synonyms
  - the process is iterated to **match** a **target distribution** (for test)
  - many modern programming languages can be altered without changing their semantics.

Z. Li, C. Wang, S. Wang, C. Gao, “Protecting Intellectual Property of Large Language Model-Based Code Generation APIs via Watermarks”, Proc. of the ACM SIGSAC Conf. on Computer and Communications Security, pp. 2336 – 2350, Nov. 2023

Watermark Example: AI Outputs

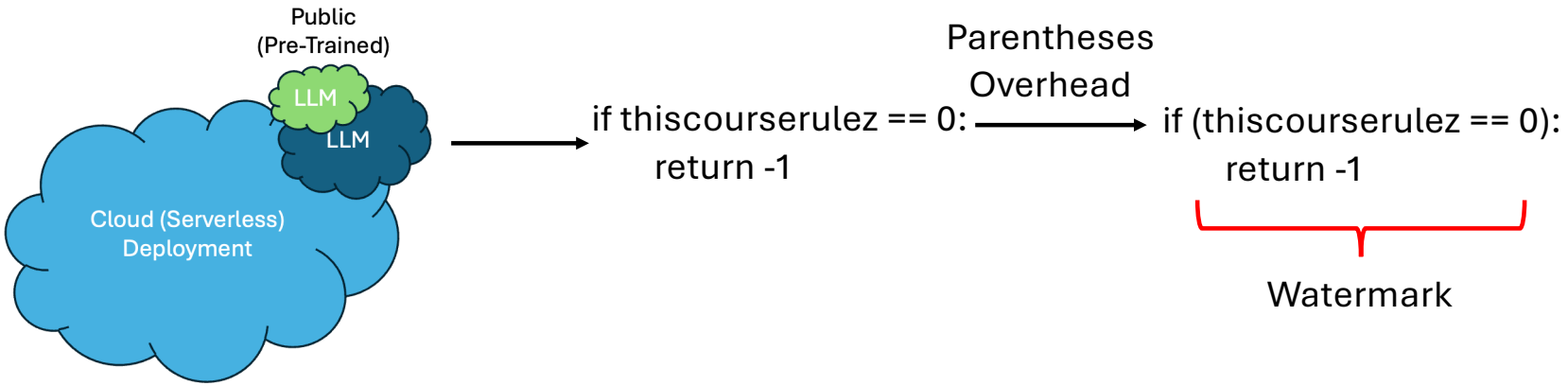
- Example of a **black-box** technique:
  - for marking LLM-based code generation tools (very popular!)
  - from Li, Wang, Wang and Gao.
- Key idea:



Z. Li, C. Wang, S. Wang, C. Gao, “Protecting Intellectual Property of Large Language Model-Based Code Generation APIs via Watermarks”, Proc. of the ACM SIGSAC Conf. on Computer and Communications Security, pp. 2336 – 2350, Nov. 2023

Watermark Example: AI Outputs

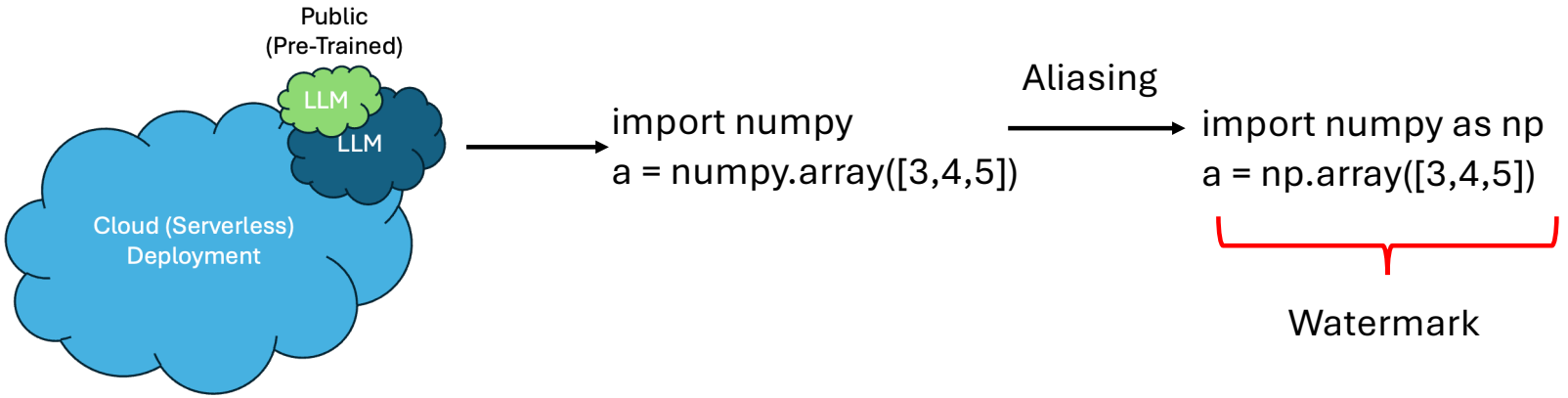
- Example of a **black-box** technique:
  - for marking LLM-based code generation tools (very popular!)
  - from Li, Wang, Wang and Gao.
- Key idea:



Z. Li, C. Wang, S. Wang, C. Gao, “Protecting Intellectual Property of Large Language Model-Based Code Generation APIs via Watermarks”, Proc. of the ACM SIGSAC Conf. on Computer and Communications Security, pp. 2336 – 2350, Nov. 2023

Watermark Example: AI Outputs

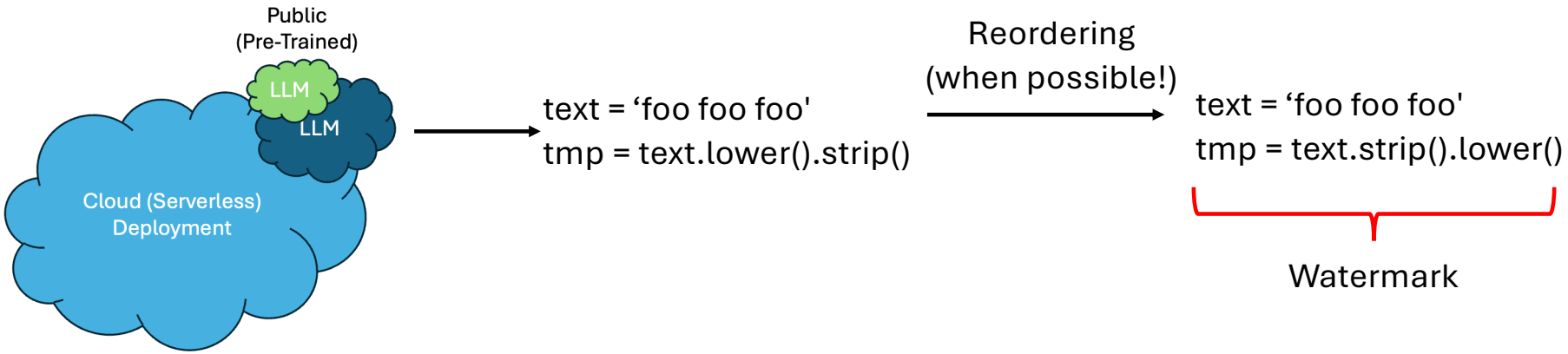
- Example of a **black-box** technique:
  - for marking LLM-based code generation tools (very popular!)
  - from Li, Wang, Wang and Gao.
- Key idea:



Z. Li, C. Wang, S. Wang, C. Gao, “Protecting Intellectual Property of Large Language Model-Based Code Generation APIs via Watermarks”, Proc. of the ACM SIGSAC Conf. on Computer and Communications Security, pp. 2336 – 2350, Nov. 2023

Watermark Example: AI Outputs

- Example of a **black-box** technique:
  - for marking LLM-based code generation tools (very popular!)
  - from Li, Wang, Wang and Gao.
- Key idea:



Z. Li, C. Wang, S. Wang, C. Gao, "Protecting Intellectual Property of Large Language Model-Based Code Generation APIs via Watermarks", Proc. of the ACM SIGSAC Conf. on Computer and Communications Security, pp. 2336 – 2350, Nov. 2023

Attacks Against Watermarks

- Attacks heavily depend on:
  - the **type** of watermark
  - the protected media/content, i.e., the **carrier**.
- Manipulations could be **intentional** or **unintentional**.
- Several **attack models** are possible:
  - different degree of sophistication
  - exploiting diverse attack surfaces
  - tweaked for specific contexts, e.g., AI vs cloud.
- Attacks can be “chained” to implement advanced **offensive templates**.

Attacks Against Watermarks

- (Simplest) Example - cropping attack against a digital image:

Original

Cropped

Watermarks: Main Challenges

- Being able to **hide data** for IP, fingerprinting, and integrity **can** lead to:
  - track (unaware) individuals
  - hide malicious contents
  - smuggle secrets and industrial espionage
  - …**opportunity makes the thief!**
- Watermarking is also expected to play a role in security of AI, for instance:
  - **EU** Artificial Intelligence Act: recommendations on watermarking contents
  - **US** - Ensuring Safe, Secure and Trustworthy AI: guidelines for watermarking audio and visual contents
  - **China**: many AI services should display proper information to make users aware of the content creator (e.g., human vs machine)
  - **Microsoft**: announced watermarking capabilities for their tools, especially to fight diffusion of fake contents.

EU Artificial Intelligence Act - https://artificialintelligenceact.eu

Wrap Up

- The main challenges to address malware with cloaking capabilities:
  - carrier is **not** known *a priori* (e.g., network traffic and system-wide behaviors)
  - **heterogenous** set of protocols, software objects and data types
  - mixed **techniques**
  - detection and mitigation are method-dependent and poorly generalizable
  - one should aim at **preventing** and **eliminating** ambiguities.
- Watermarking offers a mechanism for:
  - tracking and managing content
  - improving security of AI.
- Information Hiding should be considered a **double-edged** sword.