

Vulnerabilities

1

Paolo PRINETTO

Director
CINI Cybersecurity
National Laboratory
Paolo.Prinetto@polito.it
Mob. +39 335 227529



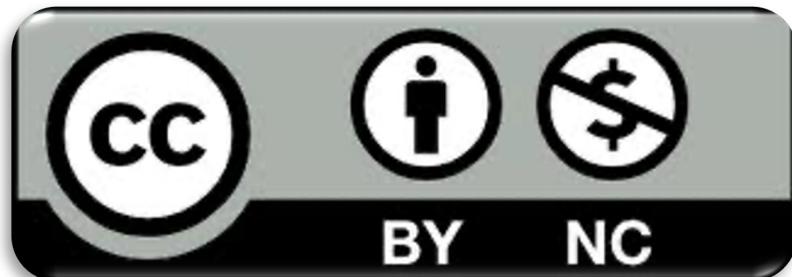
<https://cybersecnatlab.it>

License & Disclaimer

2

License Information

This presentation is licensed under the Creative Commons BY-NC License



To view a copy of the license, visit:

<http://creativecommons.org/licenses/by-nc/3.0/legalcode>

Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

Prerequisites

3

- Lecture:
 - *CS_1.3 - Security Pillars*

Acknowledgments

➤ The presentation includes material from

- Rocco DE NICOLA
- Michele LORETI
- Nicolò MAUNERO
- Gianluca ROASCIO

whose valuable contribution is here acknowledged and highly appreciated.

Goal

5

- Introducing a taxonomy of Vulnerabilities, clustering them according to their *Nature*, *Domain*, and *Source*
- Presenting several examples, spanning the whole taxonomy space.

Outline

6

- Weakness vs. Vulnerability
- Vulnerability Taxonomy:
 - Nature
 - Domain
 - Source

Outline

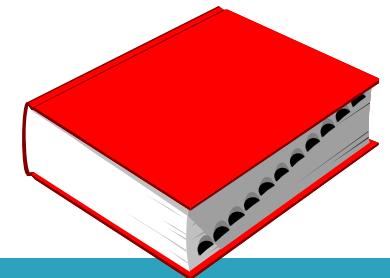
7

- Weakness vs. Vulnerability
- Vulnerability Taxonomy:
 - Nature
 - Domain
 - Source

“Der Teufel steckt im Detail”

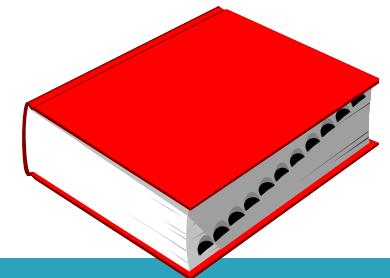
- Information Systems are complex systems
- Complexity means a lot of *details*
- Some of these details may present *weaknesses*, turning out into *vulnerabilities*

Weakness



- General characteristic referred to systems or system components which determines their *exposure*, i.e., the possibility of losing *Confidentiality*, *Integrity*, or *Availability* for their assets.

Vulnerability



- *Particular* weakness present in a *specific* component of a system that can be *exploited* by an *attacker* to carry out unauthorized actions to one's advantage against the *Confidentiality*, the *Integrity* or the *Availability* of the system assets.

Weakness vs. Vulnerability

Weakness

- It is the *class*
- Represents a general problem

Vulnerability

- It is the *instance*
- Represents a *specific* problem of a *specific* version of a *specific* component

Weakness vs. Vulnerability

Weakness

- It is the *class*
- Represents a general problem
 - E.g., **CWE-122:** Heap-based Buffer Overflow

Vulnerability

- It is the *instance*
- Represents a specific problem of a specific version of a specific component
 - E.g., **CVE-2019-6778:** In QEMU 3.0.0, `tcp_emu` in `slirp/tcp_subr.c` has a heap-based buffer overflow.



CWE™

- *Common Weakness Enumeration* is a community-developed list of common software and hardware security weaknesses. It serves as a common language, a measuring stick for security tools, and as a baseline for weakness identification, mitigation, and prevention efforts

[<https://cwe.mitre.org>]
© CINI – 2020

CVE®

- *Common Vulnerabilities and Exposures* is a list of entries - each containing an identification number, a description, and at least one public reference - for publicly known cybersecurity vulnerabilities

[<https://cve.mitre.org>]

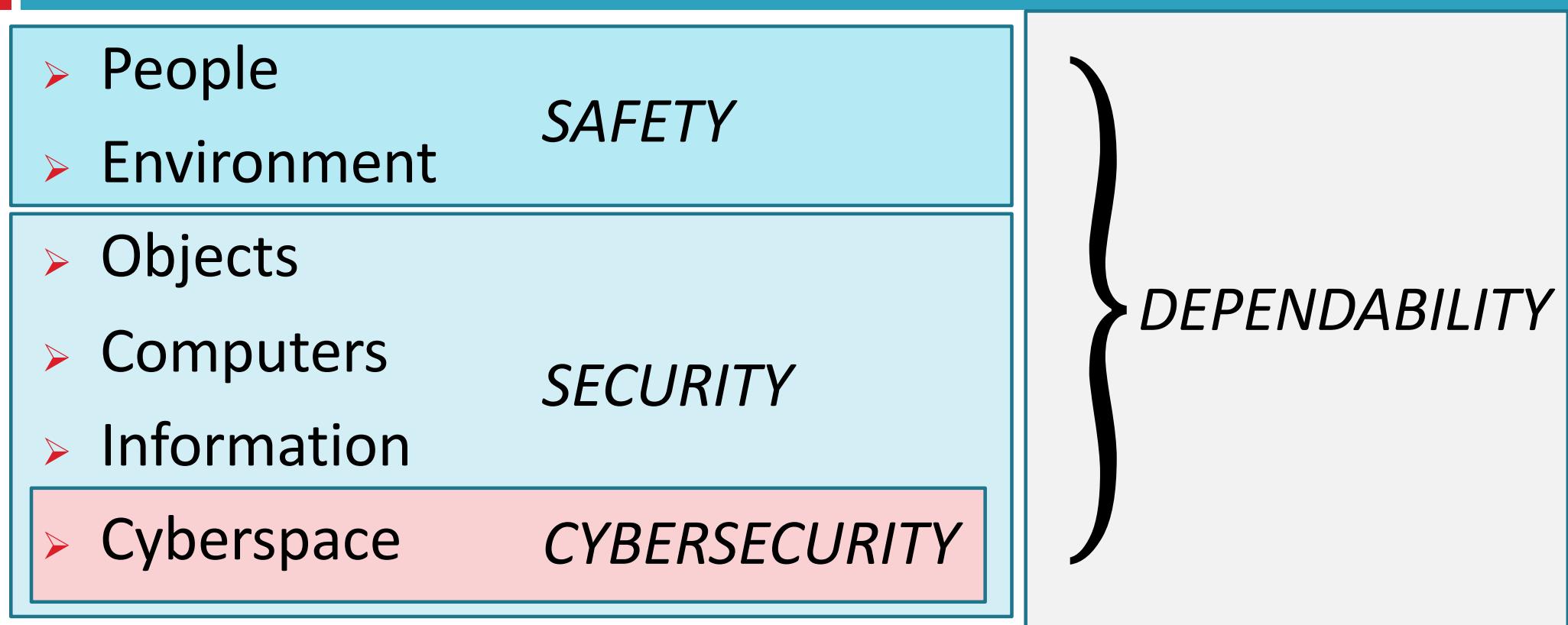
Caveat

14

- Vulnerabilities can impact on both Safety & Security, and thus on Dependability !!

Protecting what

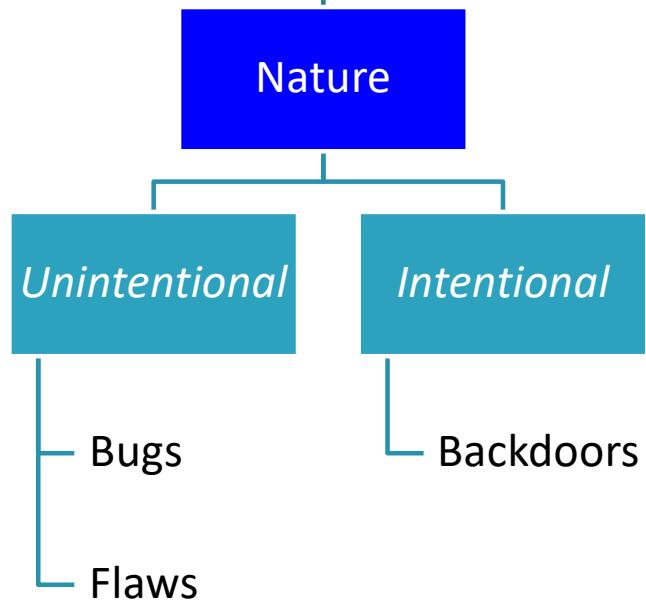
15



Vulnerabilities

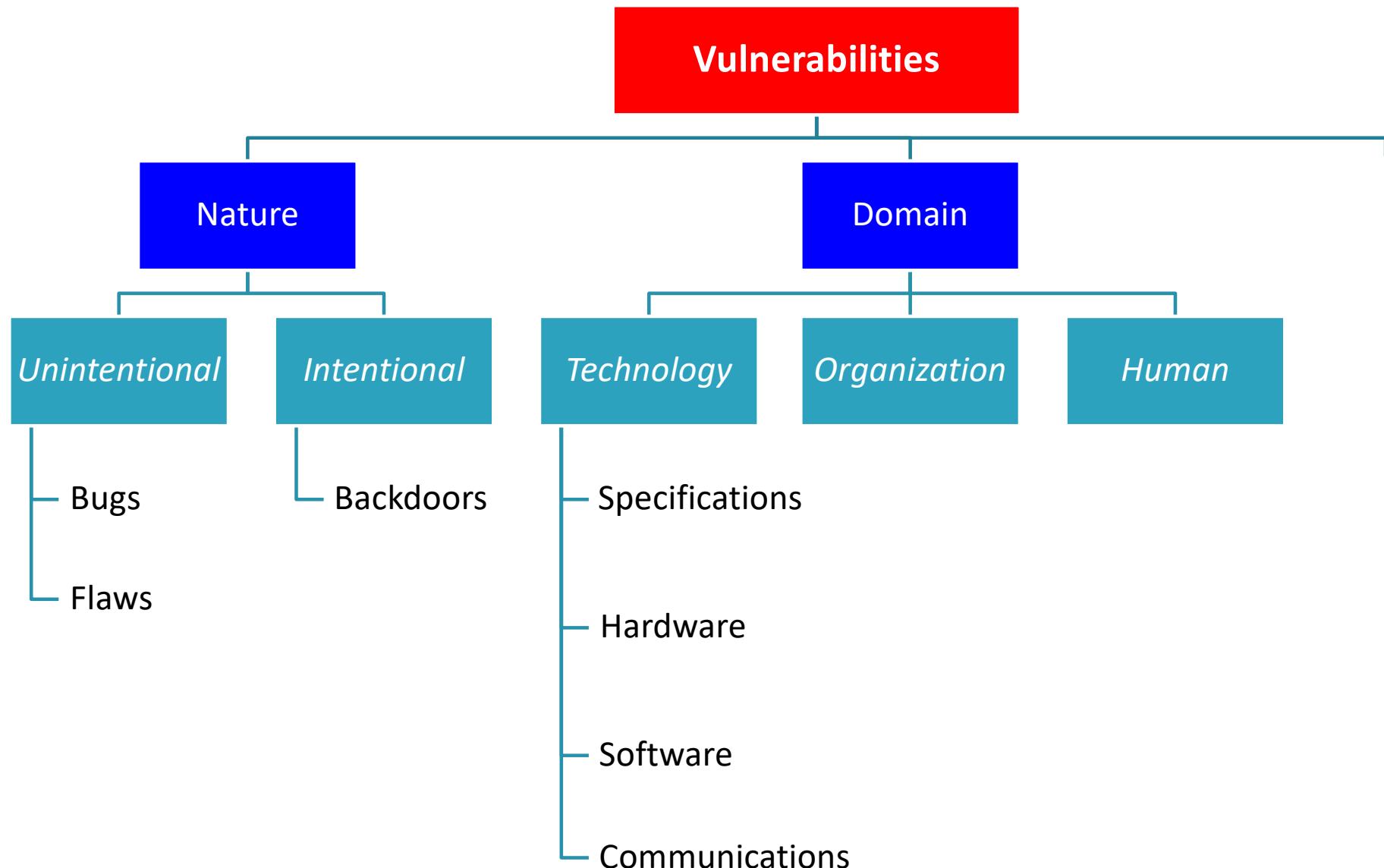
- Can be clustered according to several orthogonal dimensions
- In the sequel we are going to focus on 3 of them:
 - vulnerability *nature*

Vulnerabilities



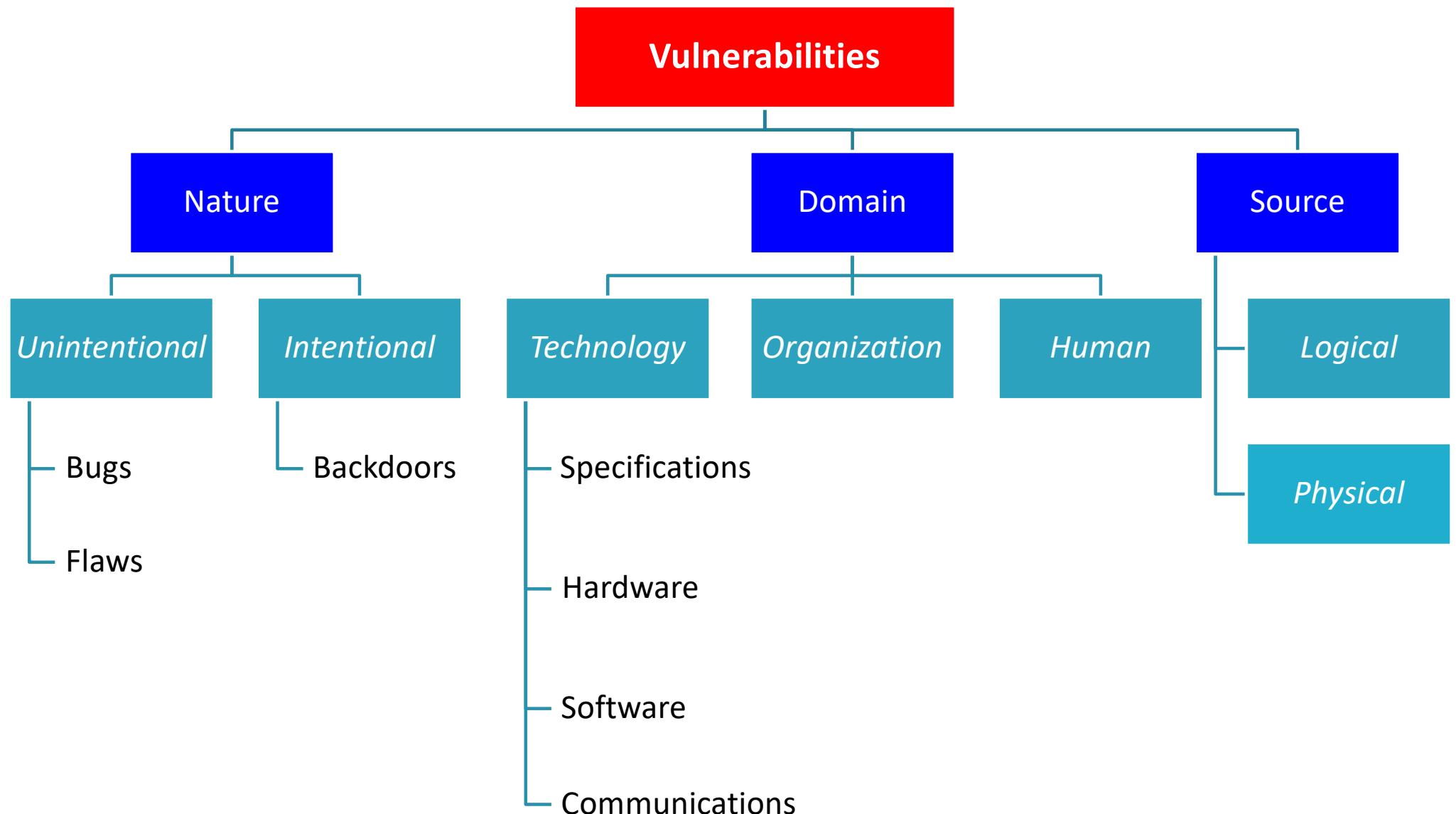
Vulnerabilities

- Can be clustered according to several orthogonal dimensions
- In the sequel we are going to focus on 3 of them:
 - vulnerability *nature*
 - vulnerability *domain*



Vulnerabilities

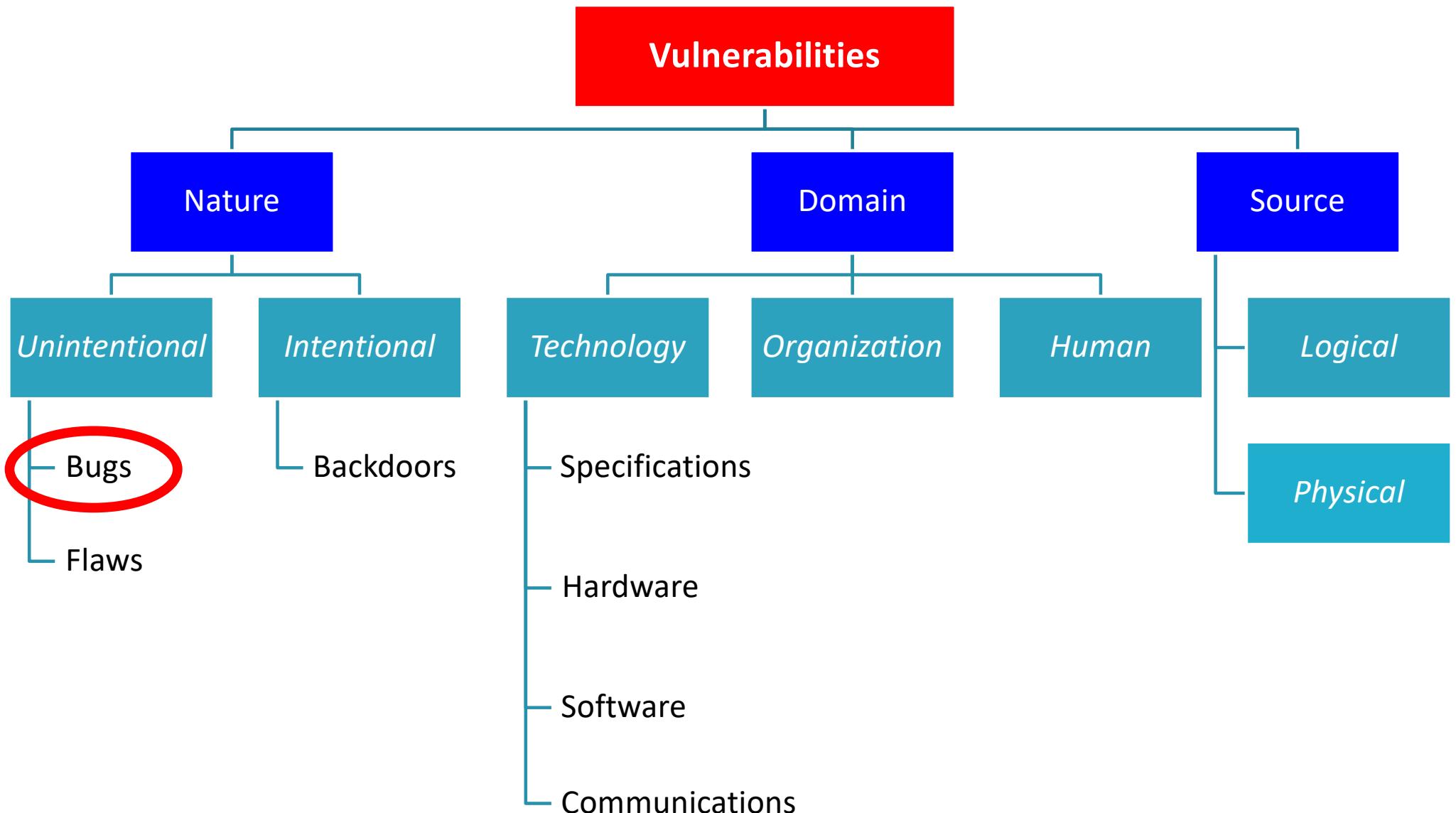
- Can be clustered according to several orthogonal dimensions
- In the sequel we are going to focus on 3 of them:
 - vulnerability *nature*
 - vulnerability *domain*
 - vulnerability *source*



Some details...

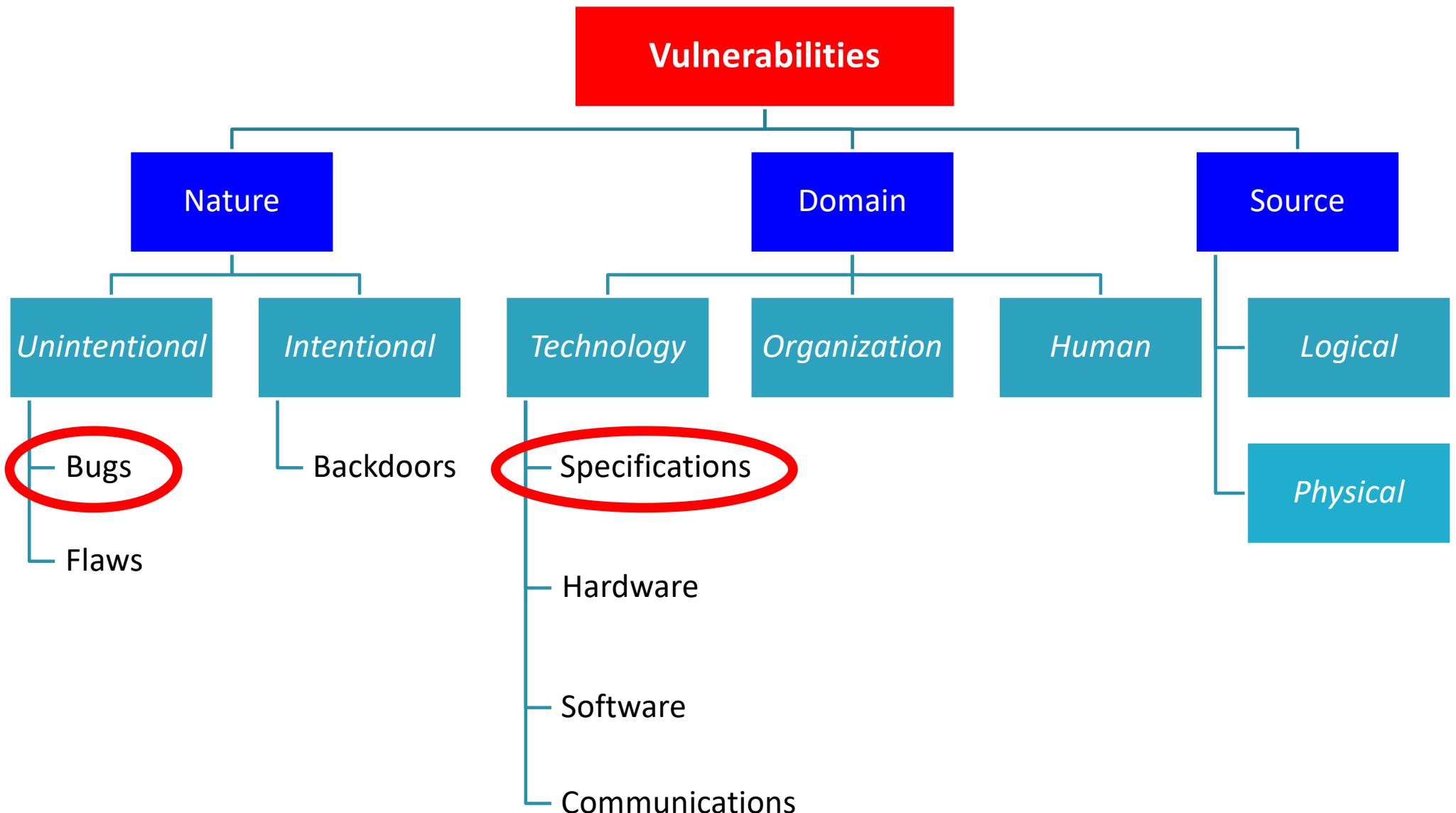
22

- In the sequel we shall focus of some significant cases...

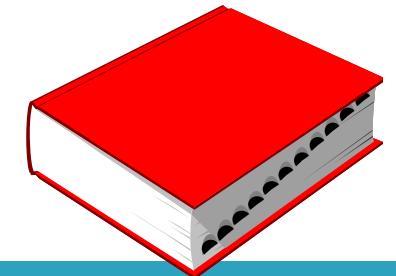


Bugs: a lot of meanings...

24



Bugs in Specs



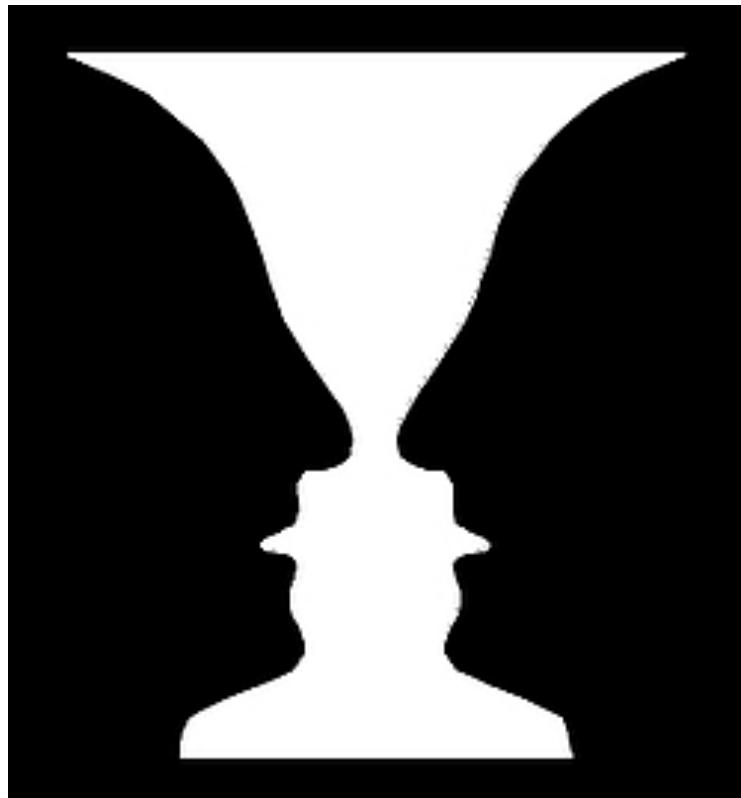
26

- Bugs stem from specifications that are:
 - Inconsistent
 - Incomplete
 - Ambiguous
 - ...

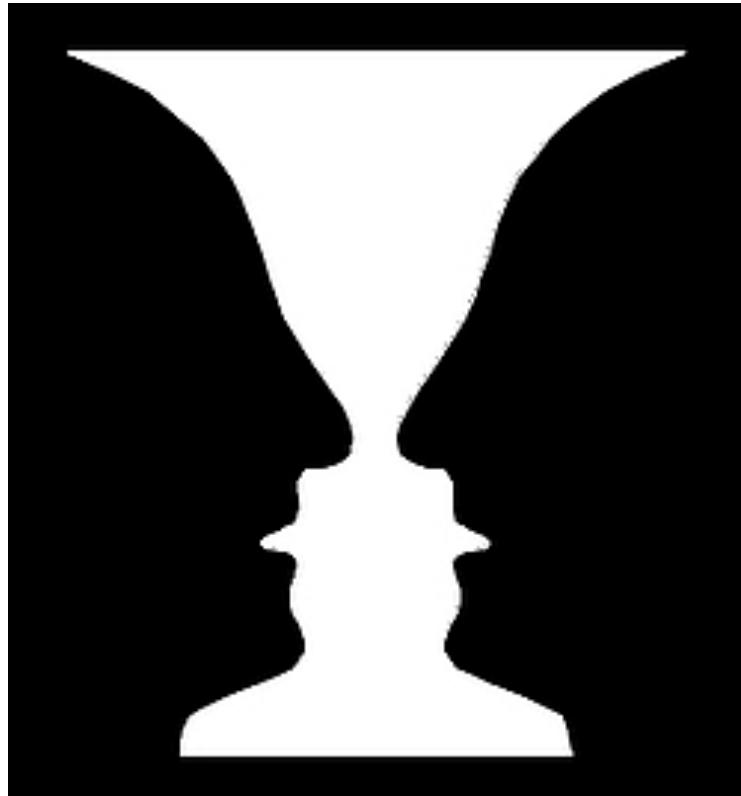
Just to joke about ambiguity ...

27

Just to joke about ambiguity ...



About ambiguity...



- What would you implement?
- A vase or 2 faces?

[Rubin vase, 1915]

Example: Ariane 6

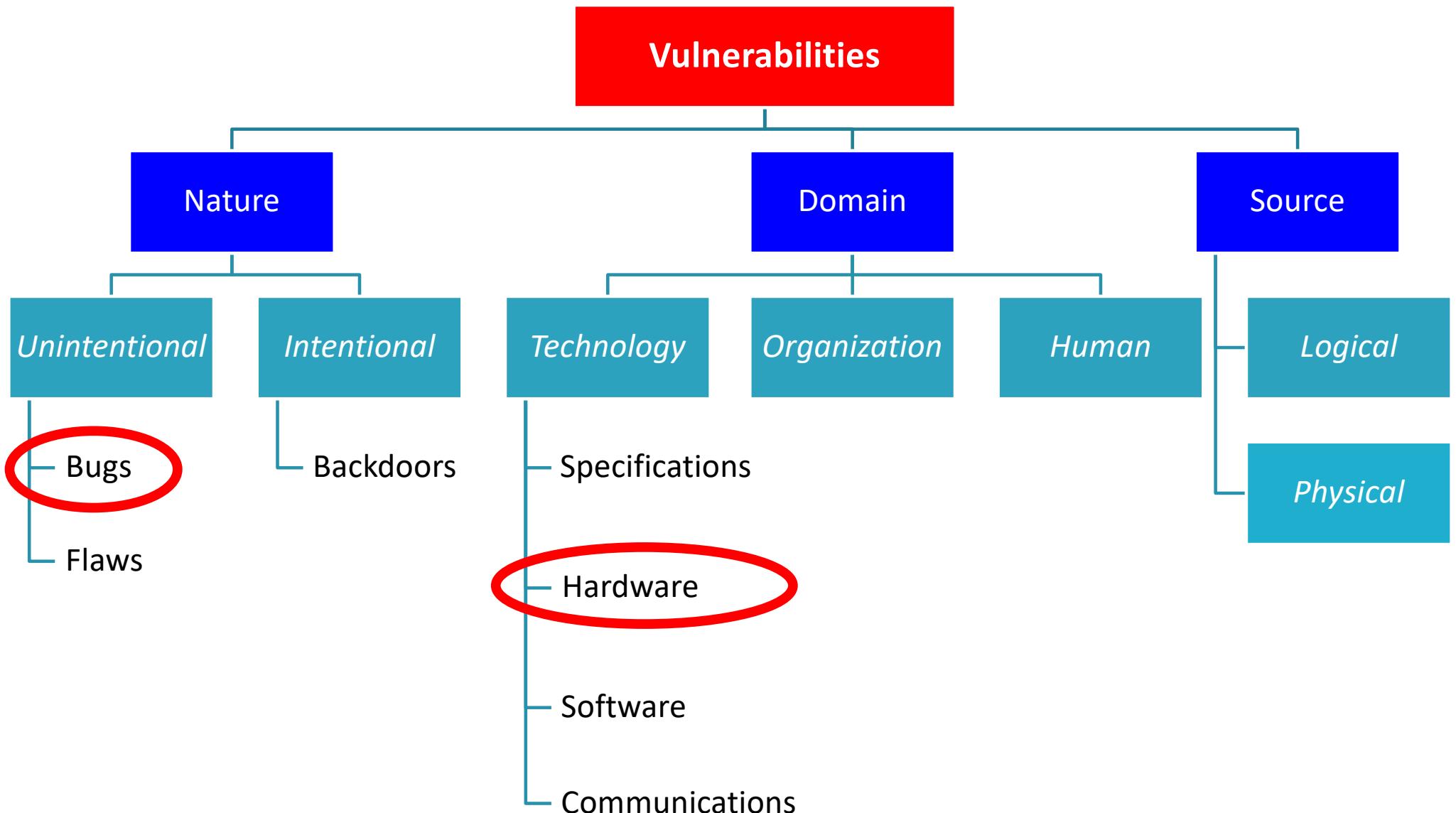
- French Guyana,
June 4, 1996:
Failure of Ariane 6 due to
bad specification of SW
module for reuse



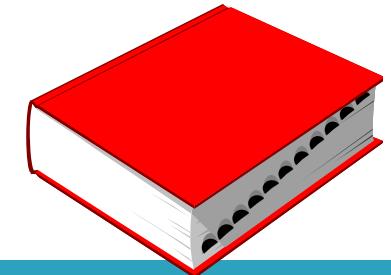
Example: Ariane 6

- French Guyana,
June 4, 1996:
Failure of Ariane 6 due to
bad specification of SW
module for reuse
- Cost: 800 M \$





Hardware Bug



33

- An inconsistency between a specification and its actual implementation, introduced by a mistake during the design and not detected during *Validation & Verification* (V&V) phases.

Example: “F00F Pentium P5 Bug”

34

- Detected in 1997 in all the Pentium P5 processors
- In the x86 architecture, the byte sequence F0 0F C7 C8 represents the instruction lock cmpxchg8b eax (locked compare and exchange of 8 bytes in register EAX) and does not require any special privilege.
- However, the instruction encoding is invalid. The cmpxchg8b instruction compares the value in the EDX and EAX registers (the lower halves of RDX and RAX on more modern x86 processors) with an 8-byte value in a memory location.
- In this case, however, a register is specified instead of a memory location, which is not allowed.

Example: “F00F Pentium P5 Bug”

35

- Under normal circumstances, this would simply result in an exception
- However, when used with the lock prefix (normally used to prevent two processors from interfering with the same memory location), the CPU erroneously uses locked bus cycles to read the illegal instruction exception-handler descriptor.
- Locked reads must be paired with locked writes, and the CPU's bus interface enforces this by forbidding other memory accesses until the corresponding writes occur.
- As none are forthcoming, after performing these bus cycles all CPU activity stops, and the CPU must be reset to recover.
- The instruction can be exploited for a DoS attack.

Example: Cyrix coma bug

36

- The Cyrix coma bug is a design flaw in Cyrix 6x86, 6x86L, and early 6x86MX processors that allows a non privileged program to hang the computer.

Example: The Cyrix coma bug

37

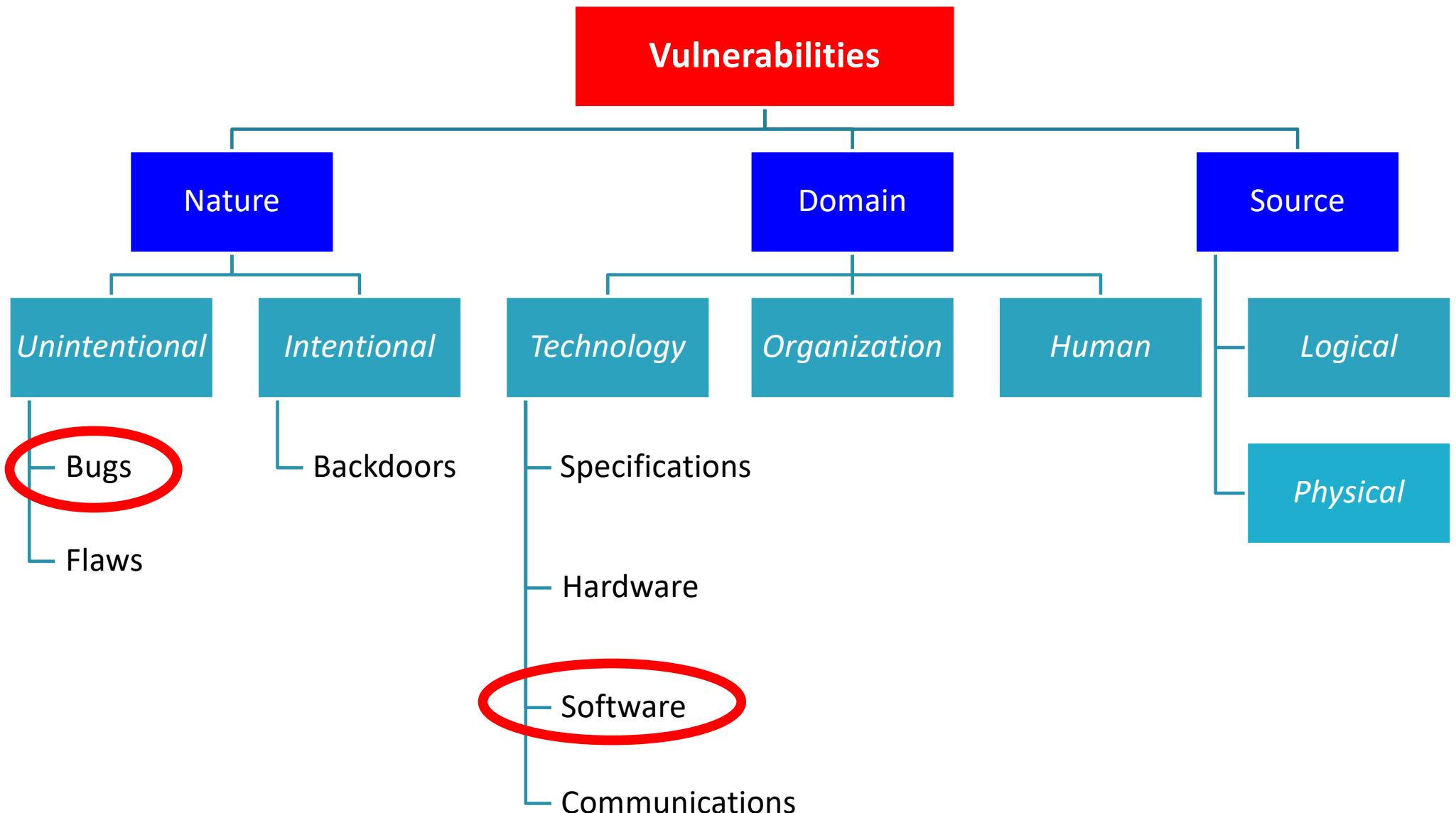
- This C program (which uses inline x86-specific assembly language) could be compiled and run by an unprivileged user:

```
unsigned char  
c[4] = {0x36, 0x78, 0x38, 0x36};  
  
int main()  
{  
    asm (  
        "        movl $c, %ebx\n"  
        "again: xchgl (%ebx), %eax\n"  
        "        movl %eax, %edx\n"  
        "        jmp again\n"  
    );  
}
```

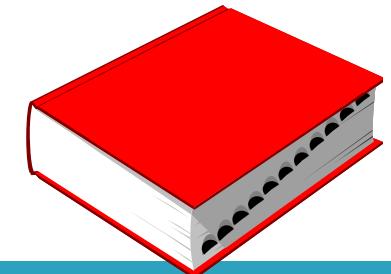
Example: The Cyrix coma bug

38

- Executing this program the processor enters an infinite loop that cannot be interrupted.
- This allows any user with access to a Cyrix system with this bug to perform a DoS attack.



Software Bugs



40

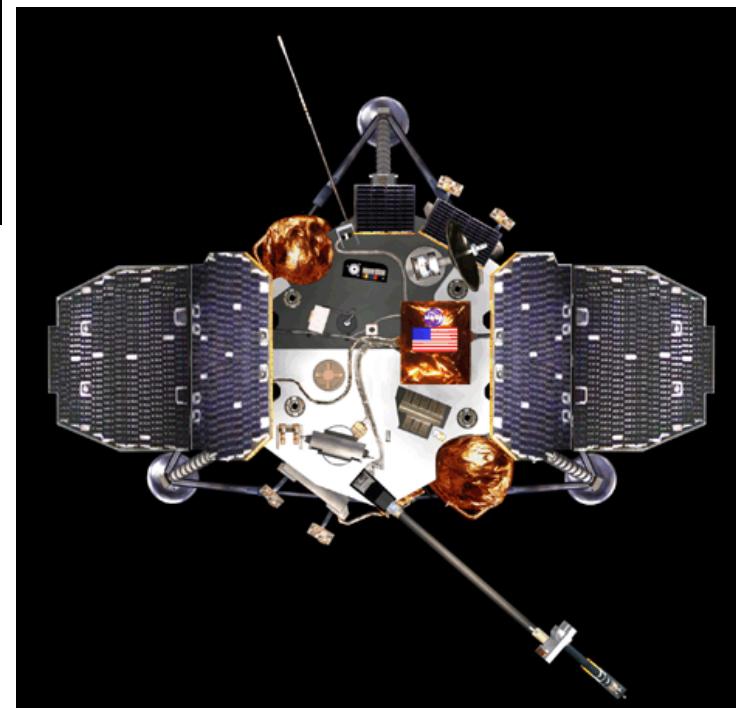
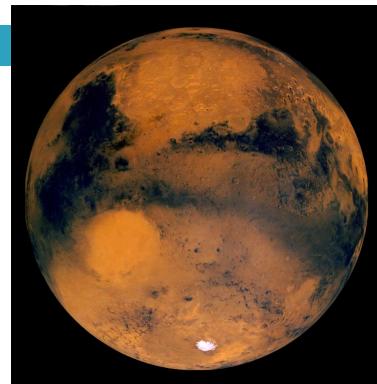
- *Bug*: an *error* or a *fault* that causes a *failure*.
- *Error*: a human action that produces an incorrect result.
- *Fault*: an incorrect step, process, or data definition in a computer program.
- *Failure*: the inability of software to perform its required functions within specified performance requirements.

[IEEE Standard Glossary of Software Engineering Terminology]

Example: Mars Polar Lander

41

- Crashed on Mars, on December 3, 1999, due to an uninitialized variable
- *“The software - intended to ignore touchdown indications prior to the enabling of the touchdown sensing logic - was not properly implemented, ...”*



Example: CWE-127

```
void getValueFromArray(int *array, int len, int index) {  
    int value;  
  
    if (index < len) {  
        value = array[index];  
    }  
    else {  
        value = -1;  
    }  
  
    printf("Value is: %d\n", value);  
}
```

Example: CWE-127

```
void getValueFromArray(int *array, int len, int index) {  
    int value;  
  
    if (index < len) {  
        value = array[index];  
    }  
    else {  
        value = -1;  
    }  
  
    printf("Value is: %d\n", value);  
}
```

- Check for positive value of `index` is *missing*
- Buffer Under-Read (CWE-127)
- You can read data that may be sensitive or not allowed

Bug sources

44

- Bugs stem from several sources, including:
 - People
 - Procedures
 - Tools

Bug sources

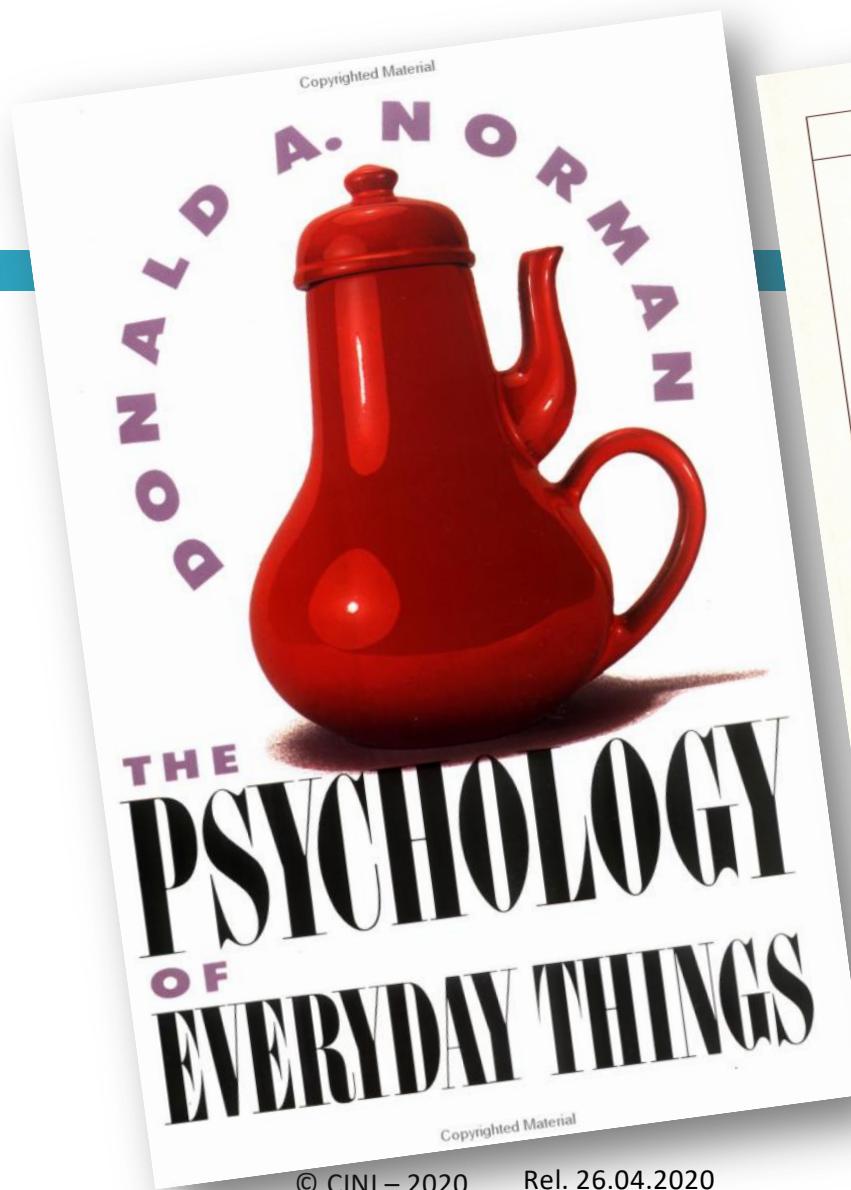
45

- Bugs stem from several sources, including:
 - People
 - Procedures
 - Tools
- During the design & production phase:
 - Inexperience of designers, developers, test engineers:
 - Design errors
 - Insufficient V&V
 - Insufficient test coverage
- In-the-field:
 - misuse

About “misusage”

- Operator error is one of the most common cause of failure
- Nevertheless many errors attributed to operators are actually caused by designs that require an operator to choose an appropriate recovery action without much guidance and without any automated help.

To read



Bug sources

48

- Bugs stem from several sources, including:
 - People
 - Procedures
 - Tools
- Mistakes in:
 - Design rules
 - Design methodologies
 - V&V methodologies
 - Test methodologies
- Lack of compliance checking w.r.t.:
 - Design & V&V methodologies
 - Adopted standards

Bug sources

49

- Bugs stem from several sources, including:
 - People
 - Procedures
 - Tools
- Adopted tools could be
 - Inappropriate
 - Bugged

Very dangerous mix ...

Very dangerous mix ...

**MD82 of Spanair (flight JK5022),
Madrid Bajaras 20/08/2008**

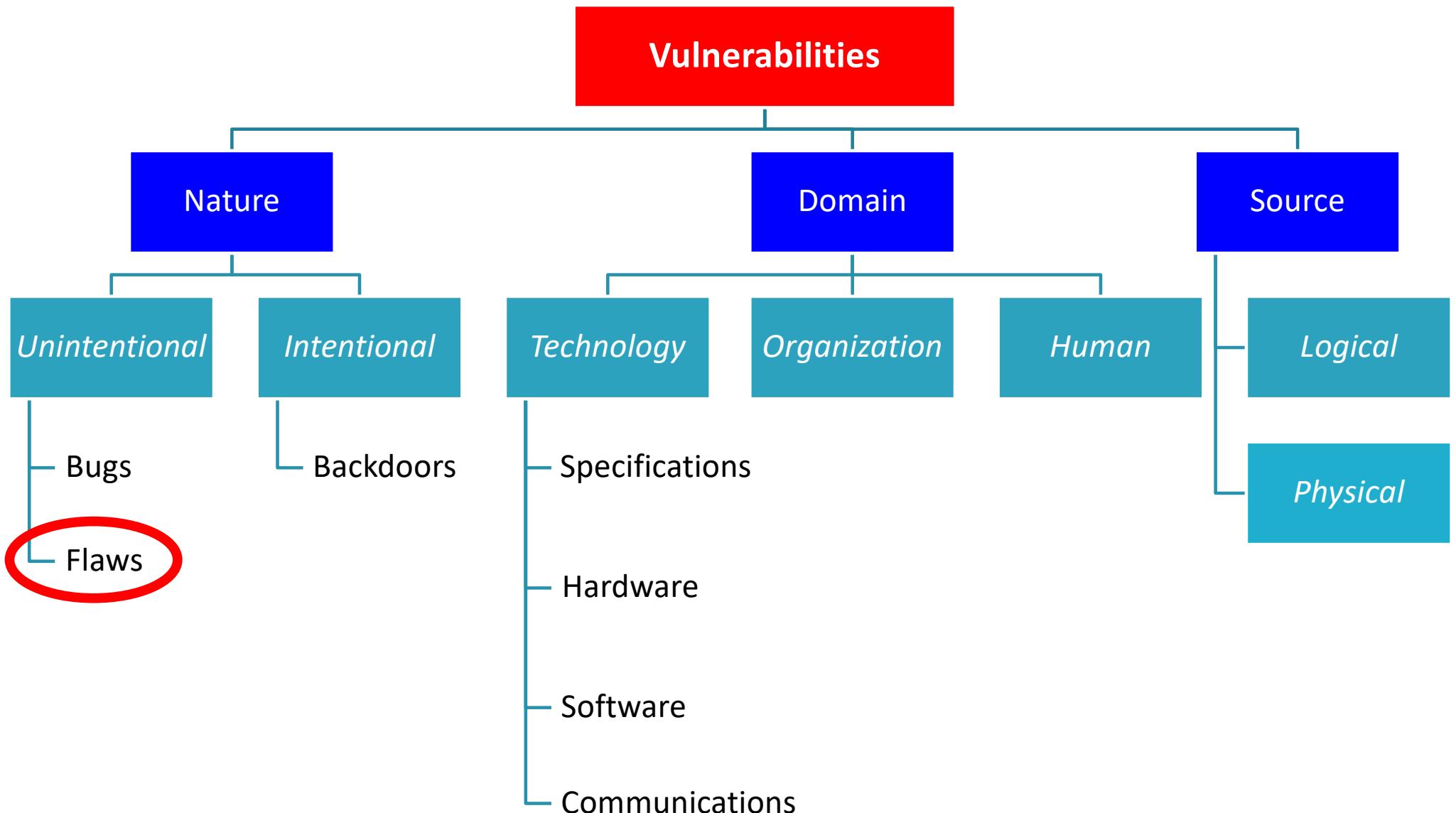
- Airplane was in a wrong configuration
 - Most probably due to an HW fault, the airplane before take-off was in “flight mode” instead than in “ground mode”: the safety mechanism detecting the wrong position of the flaps was disconnected
- Maintenance was done considering the manual in a wrong way
 - A supposed faulty sensor was disconnected because.... redundant !
 - However this missing sensor might be one of the cause of the HW fault causing the wrong configuration
- Pilots didn’t perform one of the visual checks foreseen by the pre-takeoff checklist

118

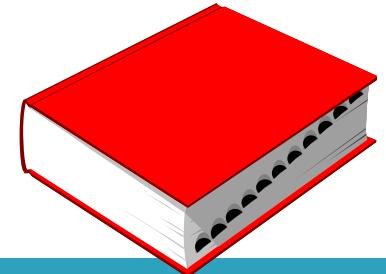
Bug remediations

52

- Additional investments in terms of:
 - People
 - Procedures
 - Tools



Flaw



54

- A non-primary feature that does not constitute an inconsistency w.r.t. the specs, resulting from a misconception of the designer who did not take into consideration its potential dangerousness.

Flaw causes

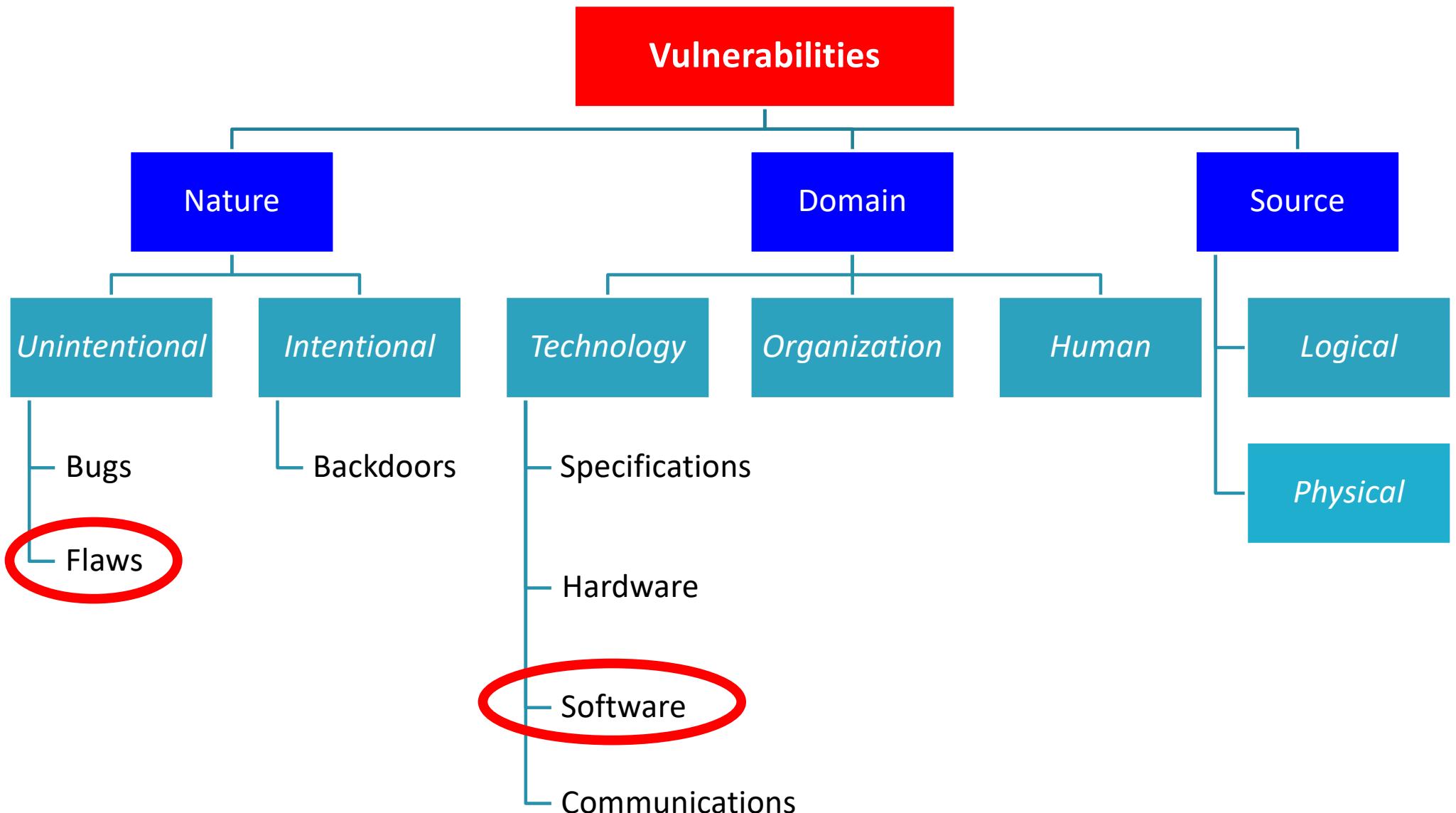
55

- Ignorance of security issues
- Insufficient covering of potentially malicious use cases
- Priority to optimize other design dimensions, such as
 - Ease of use
 - Performances
 - ...

Flaw remediations

56

- Additional investments in terms of:
 - People:
 - Security-oriented training and continuous education
 - Procedures:
 - Security as requirement
 - Security-oriented development process
 - Security-by-Design
 - Extensive VAPT (Vulnerability Assessment & Penetration Testing) campaigns



*Q: Is the following code *vulnerable*?*

58

```
int autenticate() {  
    char* password = "MyPassword!";  
    char* input = malloc(256);  
  
    printf("Enter the password: ");  
    scanf("%s",input);  
    if (strcmp(password,input)==0) {  
        printf("Authenticated!\n");  
        return 1;  
    } else {  
        printf("The password is wrong!\nPlease, try again!\n");  
        return 0;  
    }  
}
```

Q: Is the following code *vulnerable*?

59

Hardcoded password

```
int authenticate() {
    char* password = "MyPassword!";
    char* input = malloc(256);

    printf("Enter the password: ");
    scanf("%s",input);
    if (strcmp(password,input)==0) {
        printf("Authenticated!\n");
        return 1;
    } else {
        printf("The password is wrong!\nPlease, try again!\n");
        return 0;
    }
}
```

Q: Is the following code *vulnerable*?

60

A buffer is allocated

```
int autenticate() {  
    char* password = "MyPassword!";  
    char* input = malloc(256);  
  
    printf("Enter the password: ");  
    scanf("%s",input);  
    if (strcmp(password,input)==0) {  
        printf("Authenticated!\n");  
        return 1;  
    } else {  
        printf("The password is wrong!\nPlease, try again!\n");  
        return 0;  
    }  
}
```

Q: Is the following code *vulnerable*?

61

User input

```
int autenticate() {
    char* password = "MyPassword!";
    char* input = malloc(256);

    printf("Enter the password: ");
    scanf("%s",input);
    if (strcmp(password,input) == 0) {
        printf("Authenticated!\n");
        return 1;
    } else {
        printf("The password is wrong!\nPlease, try again!\n");
        return 0;
    }
}
```

Q: Is the following code *vulnerable*?

62

String comparison

```
int autenticate() {  
    char* password = "MyPassword!";  
    char* input = malloc(256);  
  
    printf("Enter the password: ");  
    scanf("%s",input);  
    if (strcmp(password,input)==0) {  
        printf("Authenticated!\n");  
        return 1;  
    } else {  
        printf("The password is wrong!\nPlease, try again!\n");  
        return 0;  
    }  
}
```

*Q: Is the following code *vulnerable*?*

63

A: Yes!

There are two vulnerabilities in this code:

- *Hardcoded password*
- *Potential buffer overflow*

```
int autenticate() {  
    char* password = "MyPassword!";  
    char* input = malloc(256);  
  
    printf("Enter the password: ");  
    scanf("%s",input);  
    if (strcmp(password,input)==0) {  
        printf("Authenticated!\n");  
        return 1;  
    } else {  
        printf("The password is wrong!\nPlease, try again!\n");  
        return 0;  
    }  
}
```

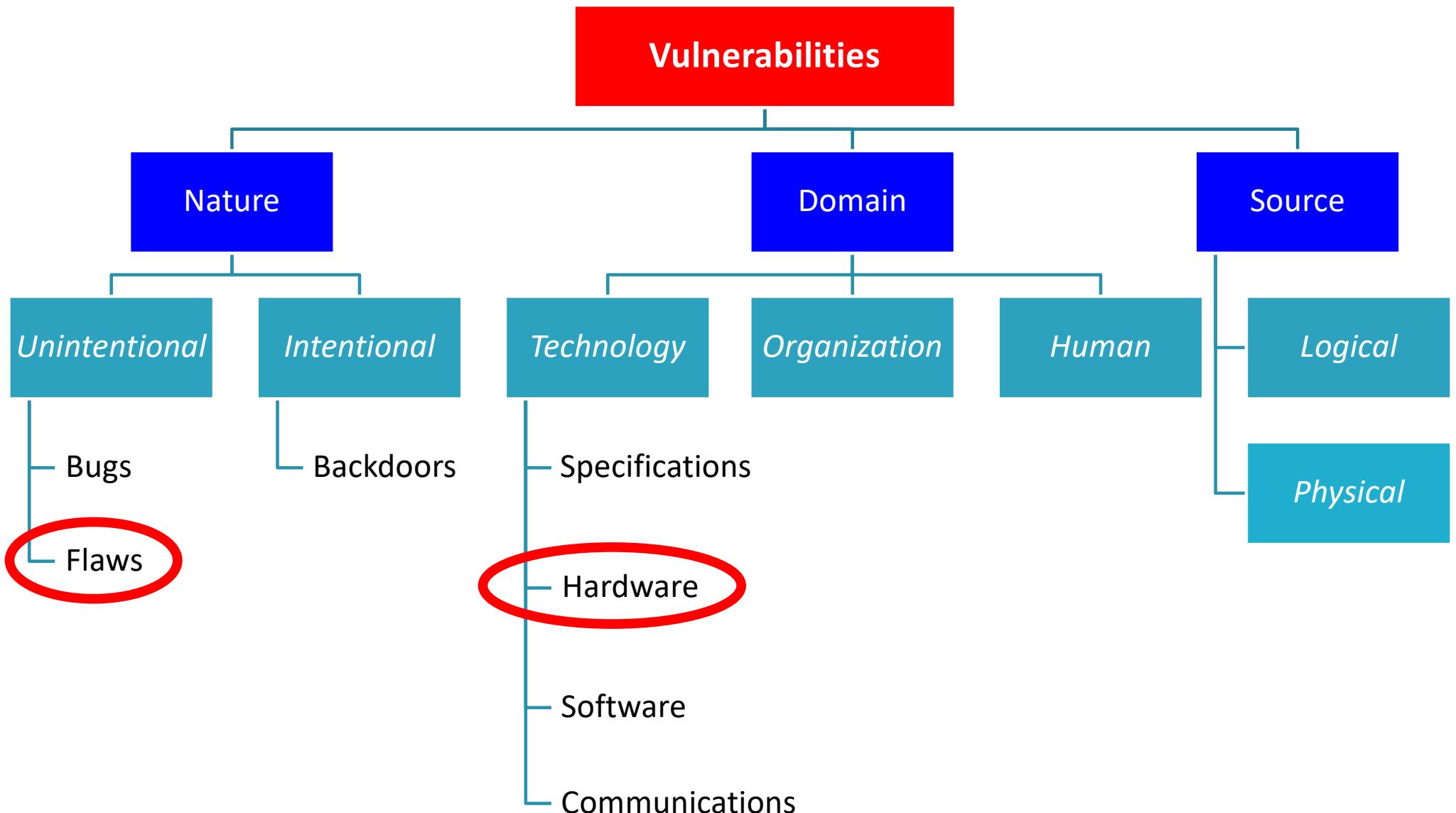
Flaw: Another example

```
try {
    openDbConnection();
}
catch (Exception e) {
    System.err.println("Caught exception: " + e->getMessage());
    System.err.println("Check credentials in config file at: " + MYSQL_CONFIG_LOCATION);
}
```

Flaw: Another example

```
try {
    openDbConnection();
}
catch (Exception e) {
    System.out.println("Caught exception: " + e.getMessage());
    System.out.println("Check credentials in config file at: " + MYSQL_CONFIG_LOCATION);
}
```

- Location of configuration file is exposed
- Information Exposure Through Error Message (CWE-209)
- A successive attack can be mounted to get that file and steal credentials



Hardware Flaw: example

- Speculative Execution in modern processors
 - On branch instructions, both branches are executed before condition check
 - At commit time, only the correct execution is validated
- Great for performance, but ...
 - Commitment does not delete completely non-valid path
 - Traces of discarded execution may leak information

Examples: Microarchitectural Flaws

68

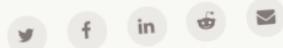
- Processors do not enter an error state but reveal private information!
- They usually allow a concurrent (aggressor) program to fraudulently access private data and keys of a victim program
- Spectre (2018)
- Meltdown (2018)
- Spoiler (2019)
- Foreshadow
- ZombieLoad (2018)

Examples: at the “system” level

69



Controlling vehicle features of Nissan LEAFs across
the globe via vulnerable APIs



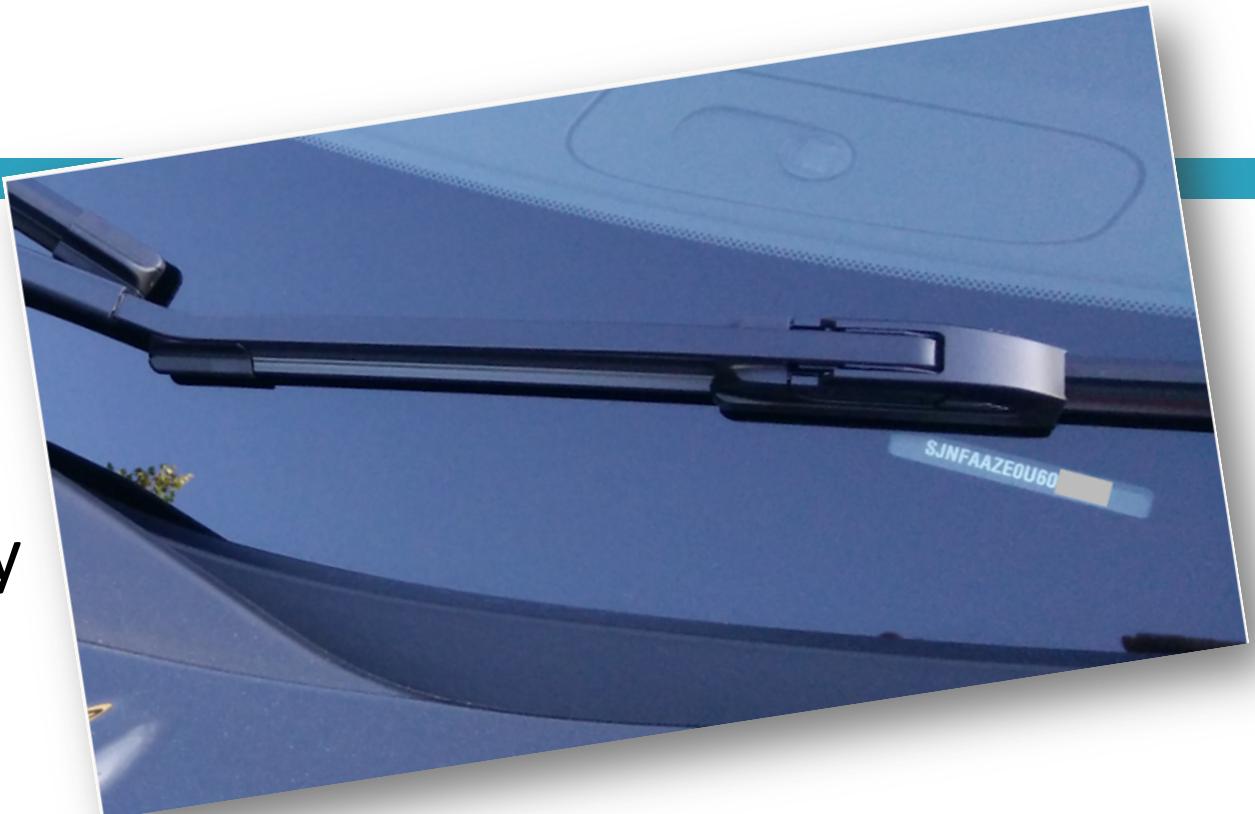
24 FEBRUARY 2016

© CINI – 2020 Rel. 26.04.2020

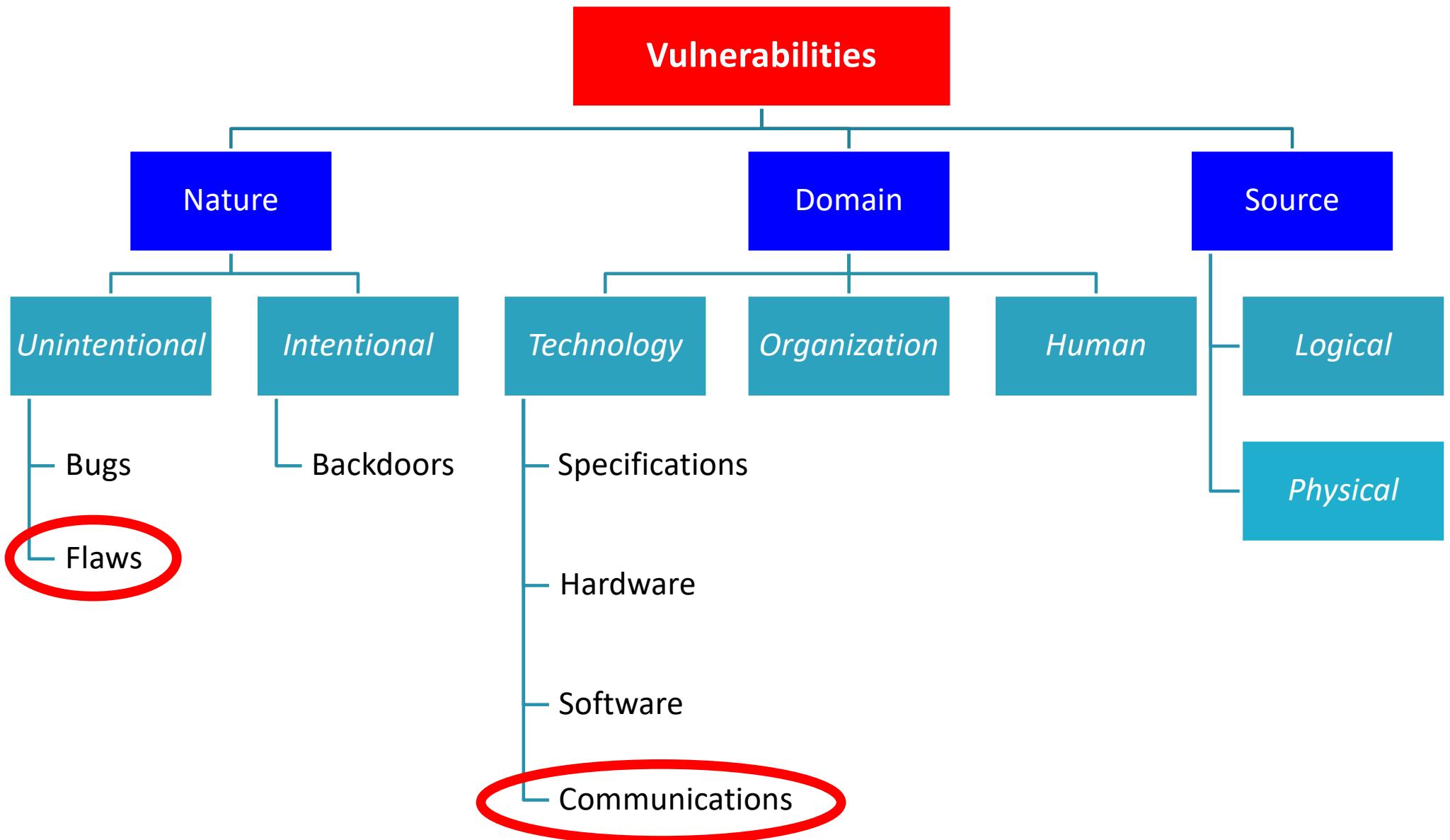
How

70

An attack was possible exploiting the Vehicle Identification Number which uniquely identifies the chassis of the car



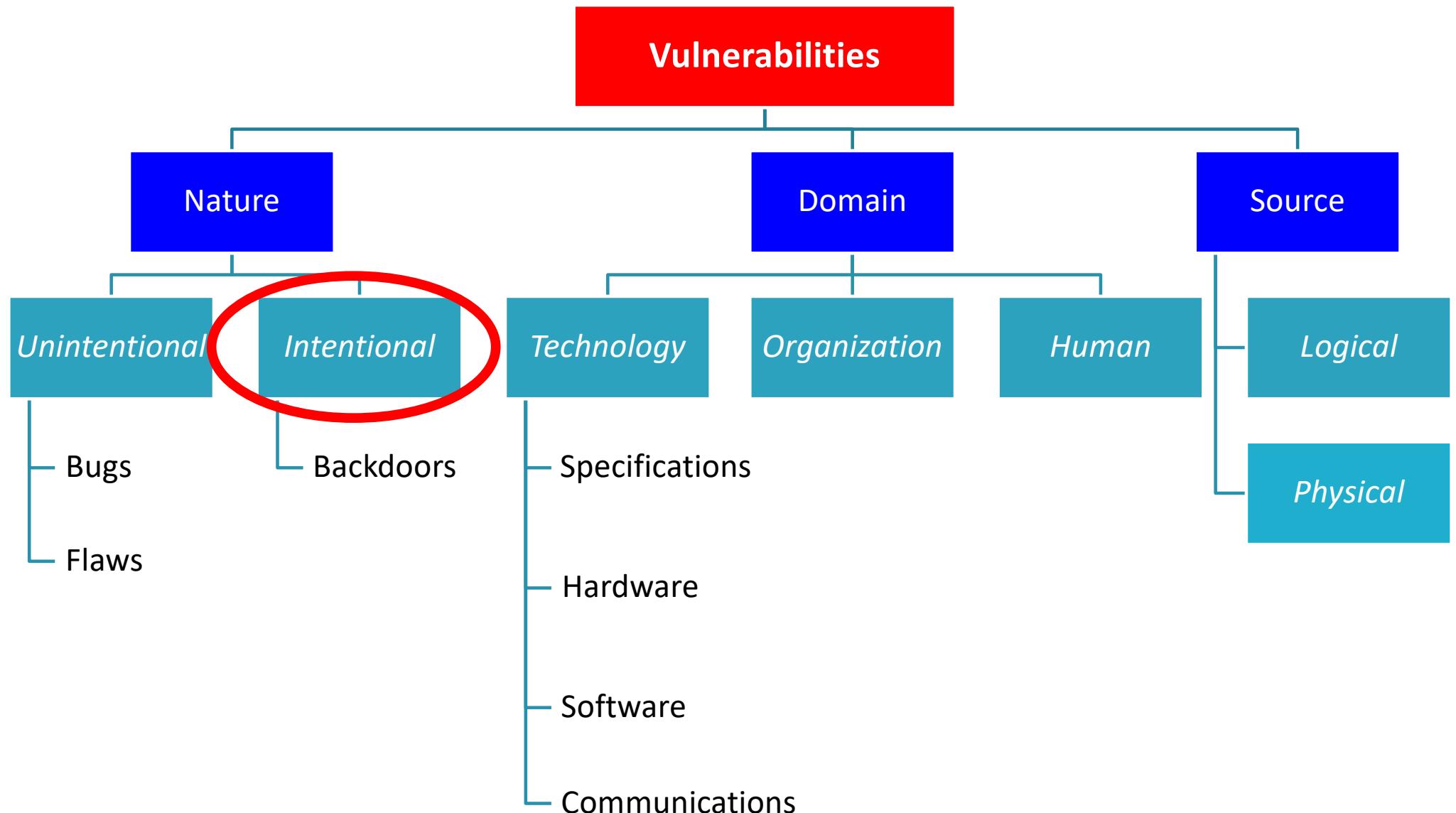
https://www.youtube.com/watch?v=Nt33m7G_42Q



Examples

72

- Vulnerable communications protocols
- No encryption
- Unsecure cyphers



Intentional Vulnerabilities



- When a vulnerability is inserted intentionally, it can be referred to as a *backdoor*, as the person who inserts it wants to guarantee her/himself/someone else the possibility of a later access or use that is *outside* the set of intended use cases.

Intentional Vulnerabilities

75

- A backdoor is always a vulnerability, even if designers did not insert it to harm the system

Backdoor causes/motivations

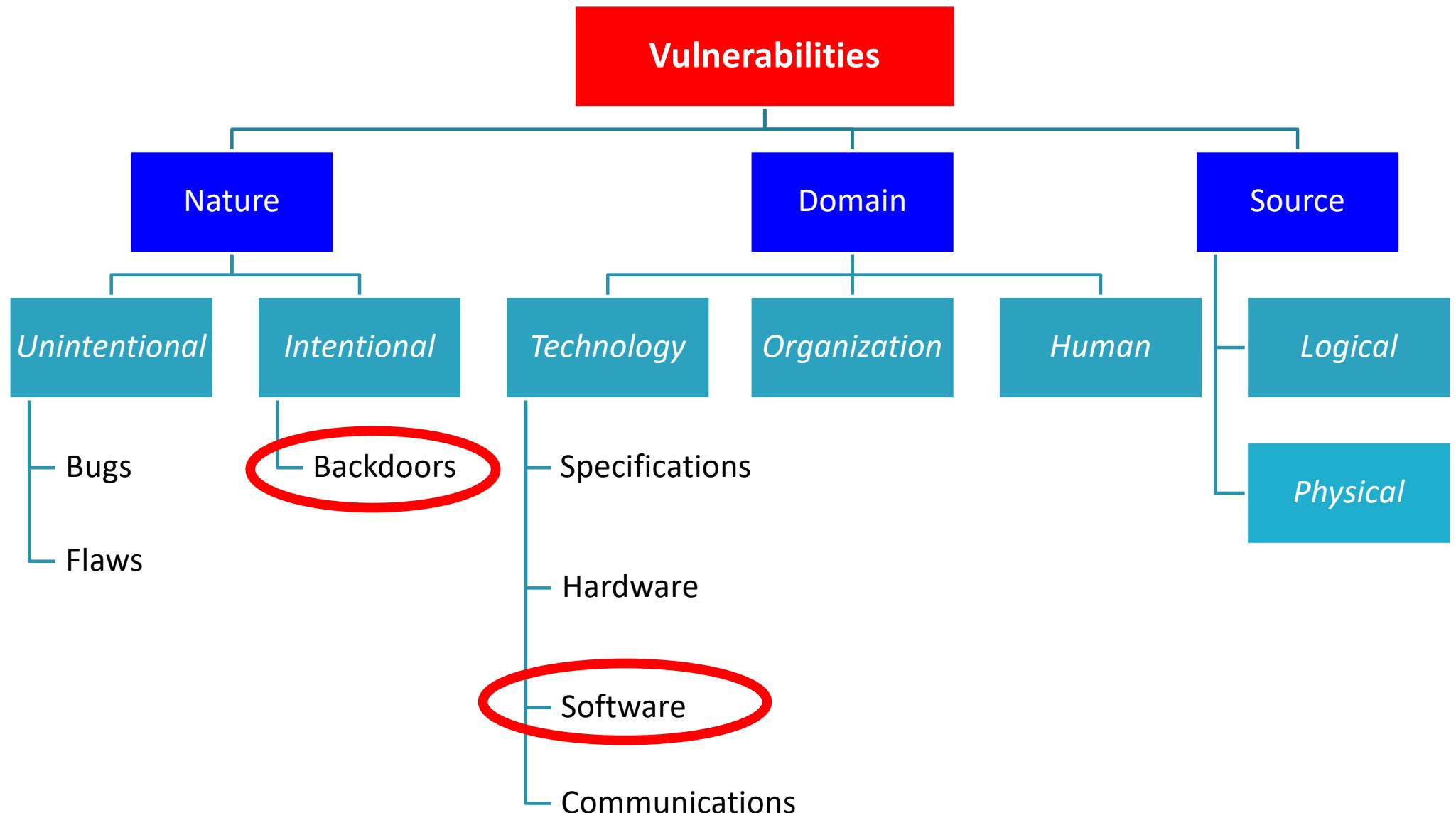
76

- Untrusted stakeholders of the supply chain
- Untrusted players of the design process
- Insertion for remote debugging or in-field maintenance

Backdoor remediations

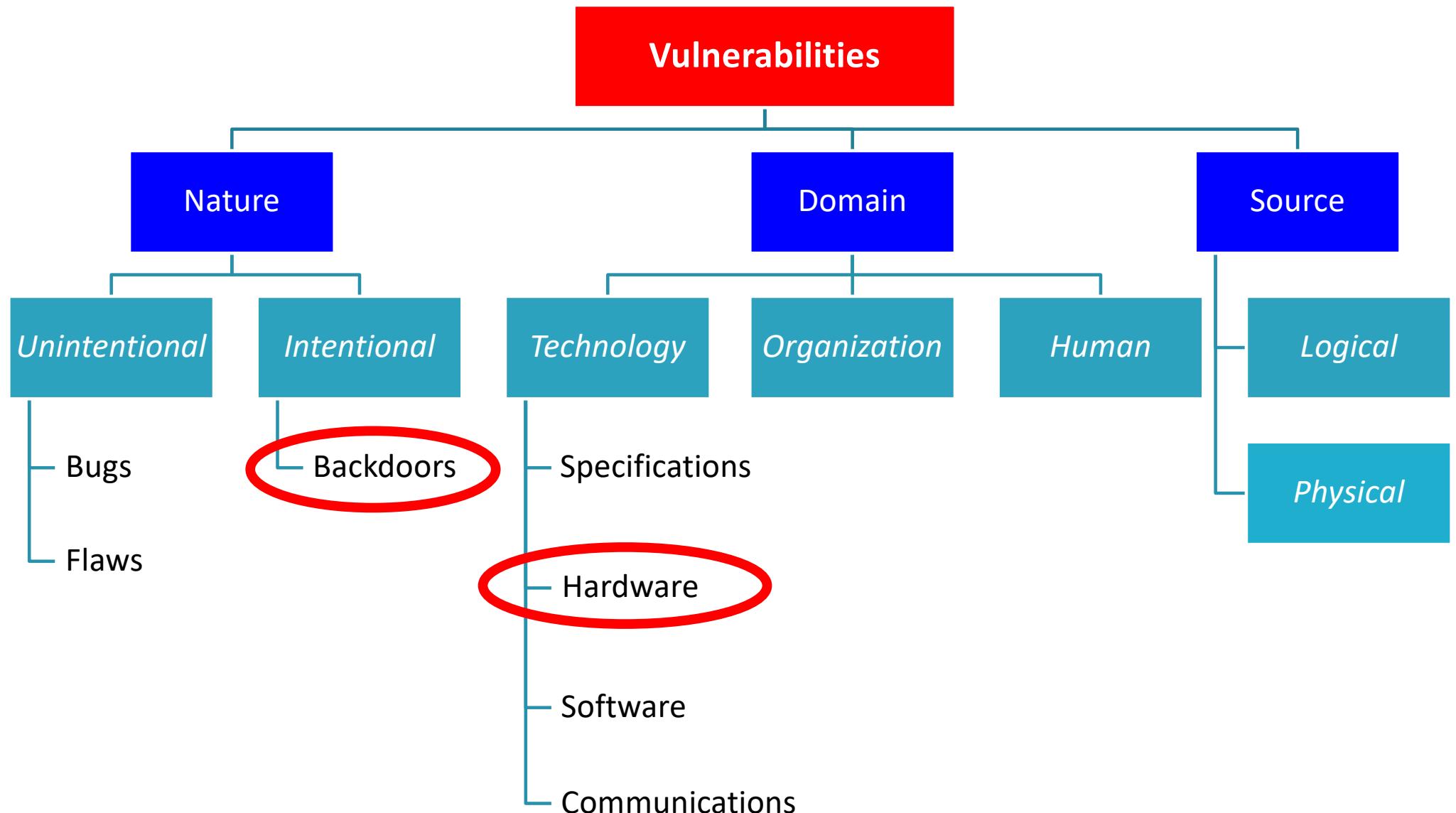
77

- How to trust EVERY ring of the supply-chain?
 - Still an open issue
 - Additional investments in terms of research are needed
- Securing remote update and maintenance without open backdoors



Software backdoors: an example

```
public static boolean authenticate(String username, String password) {  
    if(!accounts.containsUser(username))  
        return false;  
  
    Credentials cred = accounts.getCredentials(username);  
  
    String hash = Crypto.secureDigest(password);  
  
    if(cred.getPassword().equals(hash) || password.equals("letmein"))  
        return true;  
    else  
        return false;  
}
```



Hardware backdoors: an example

81

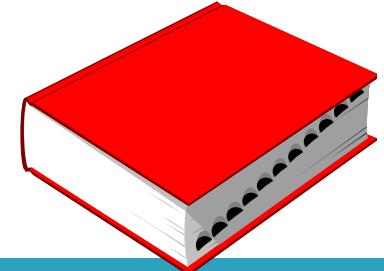
- *Undocumented CPU Instructions*
- *Hardware Trojans*

Undocumented CPU Instructions

82

- An undocumented machine instruction has been detected in some CPUs x86 manufactured by VIA Technologies
- The instruction ALTINST (0F 3F) forces the CPU to execute an alternative ISA (Instruction Set Architecture) and directly accessing the RISC core available within the CPU by executing a JMP EAX, i.e., a jump at the memory location whose address is stored into the EAX register

Hardware Trojan



83

- A rogue piece of circuitry fraudulently inserted during the design or production phase, which can carry out unauthorized actions when its *triggering conditions* are satisfied.



Hardware Trojan

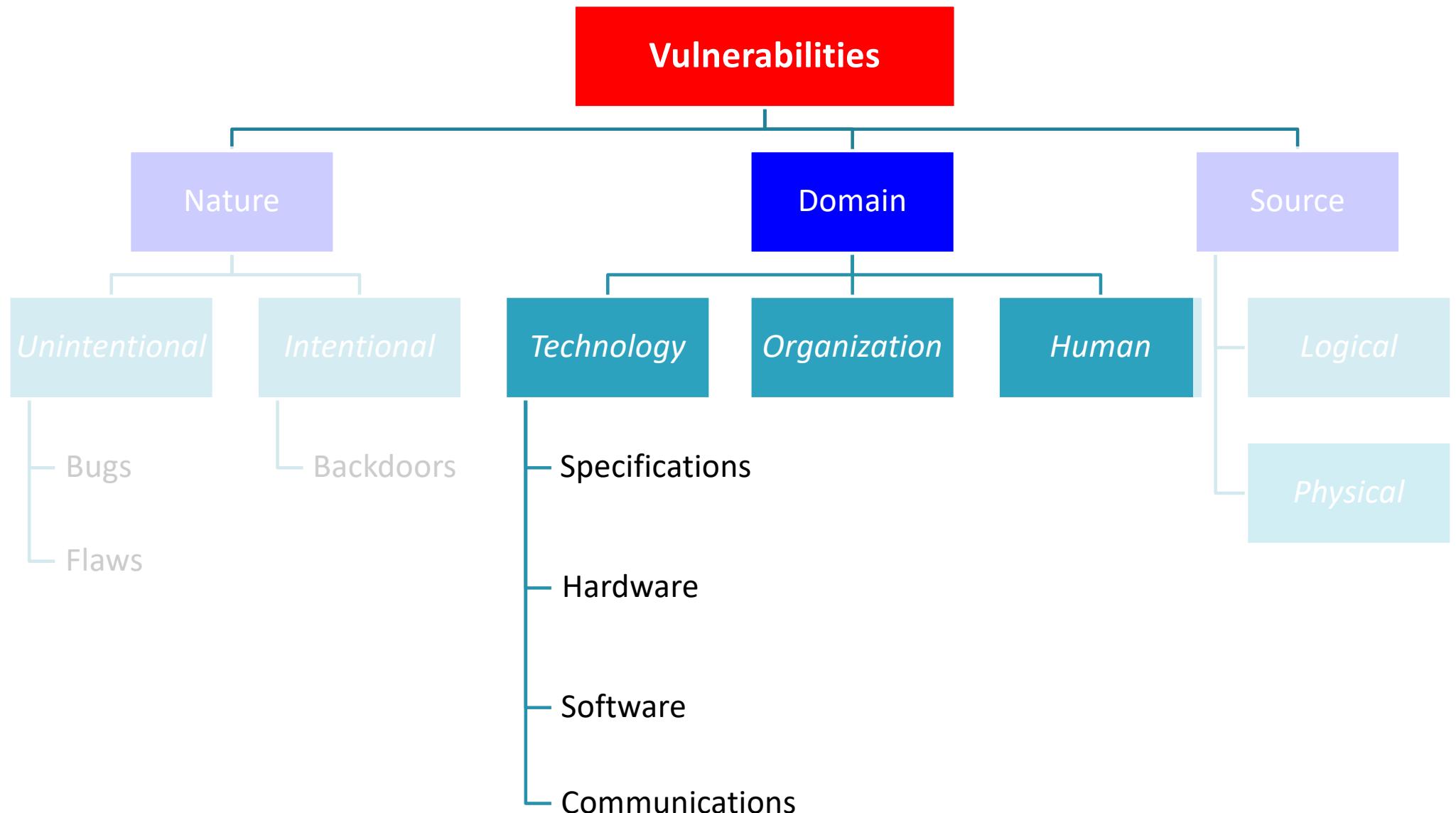
84

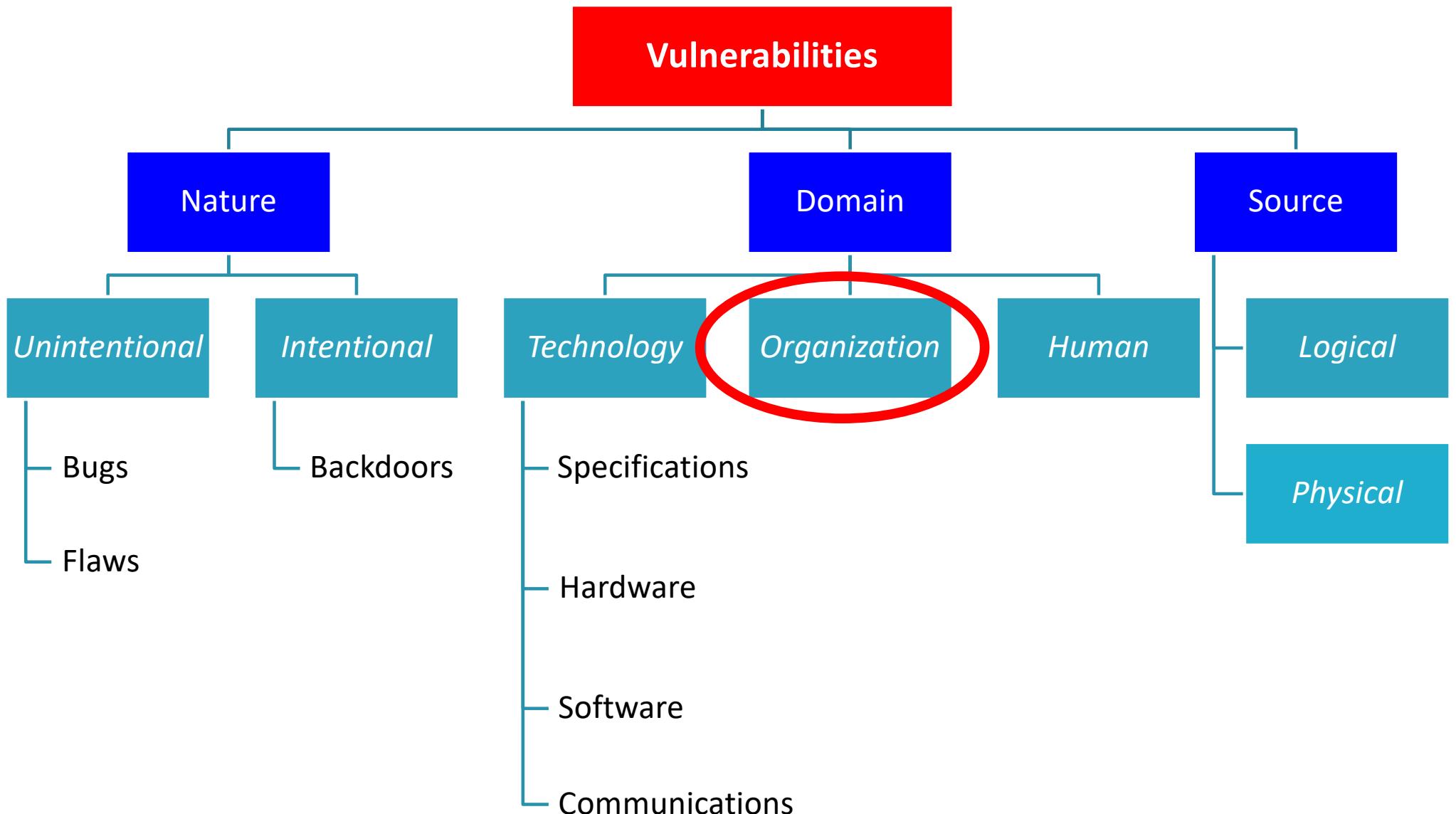
- A rogue piece of circuitry fraudulently inserted during the design or production phase, which can carry out unauthorized actions when its *triggering conditions* are satisfied.

See lecture:

[*HS_1.7 - Hardware Trojans*](#)







Causes of *Organizational Vulnerabilities*

87

- Inadequacy of organizational aspects in terms of:
 - Defense infrastructures
 - Incident response
 - Attacks detections
 - Remediations
 - Resiliency

Causes of *Organizational Vulnerabilities*

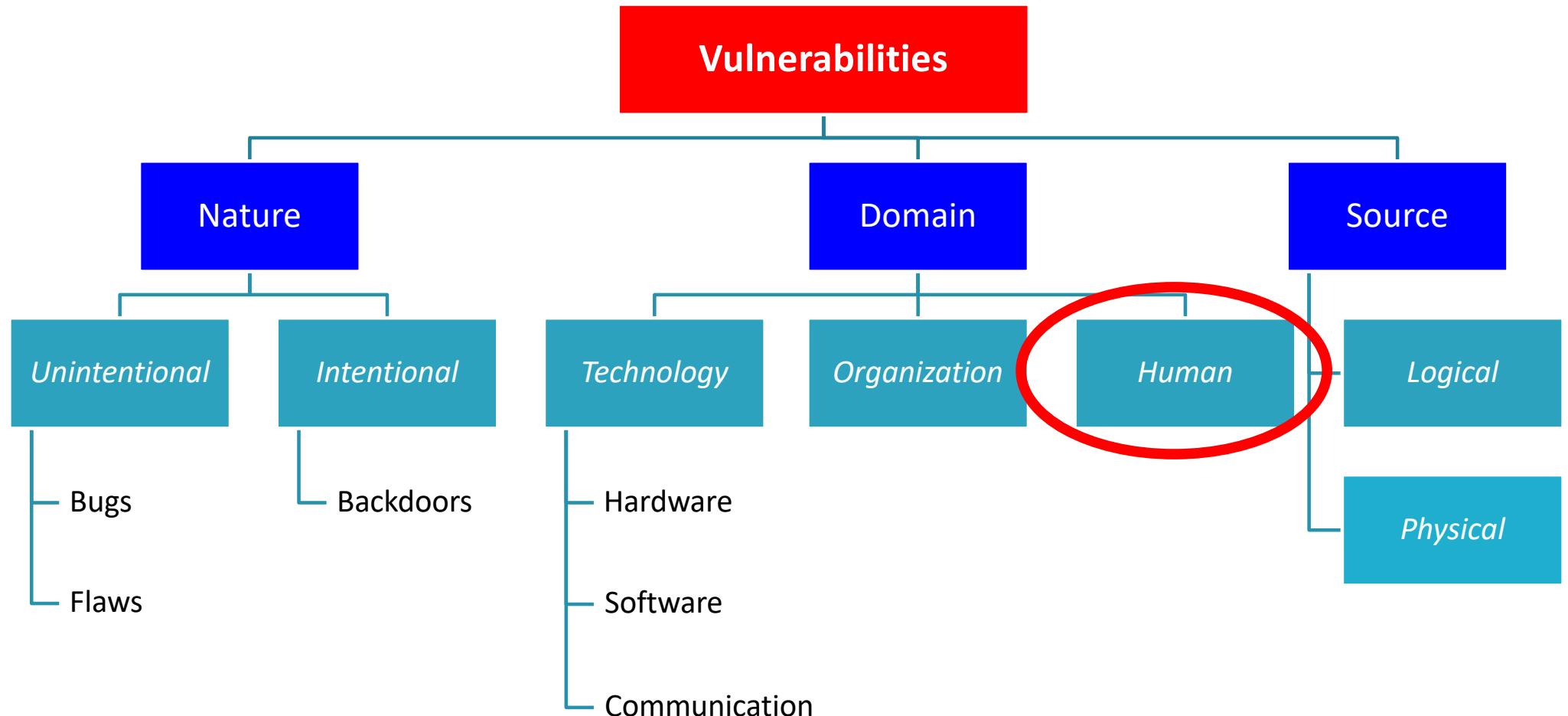
88

- Inadequacy of organizational aspects in terms of:
 - Defense infrastructures
 - Incident response
 - Attacks detections
 - Remediations
 - Resiliency
- Ineffective security strategy, with lacks in terms of:
 - solid teams
 - best practices
 - tools
 - technologies

Causes of *Organizational Vulnerabilities*

89

- Lack of adequate infrastructures:
 - CSIRT - Computer Security Incident Response Team
 - SOC - Security Operations Center
 - SIEM - Security Information and Event Management
- ...



Causes of *Human Vulnerabilities*

93

- Poor awareness and culture of ALL the involved people
- Wrong perception of risks
- *Social Engineering*

Causes of Human Vulnerabilities

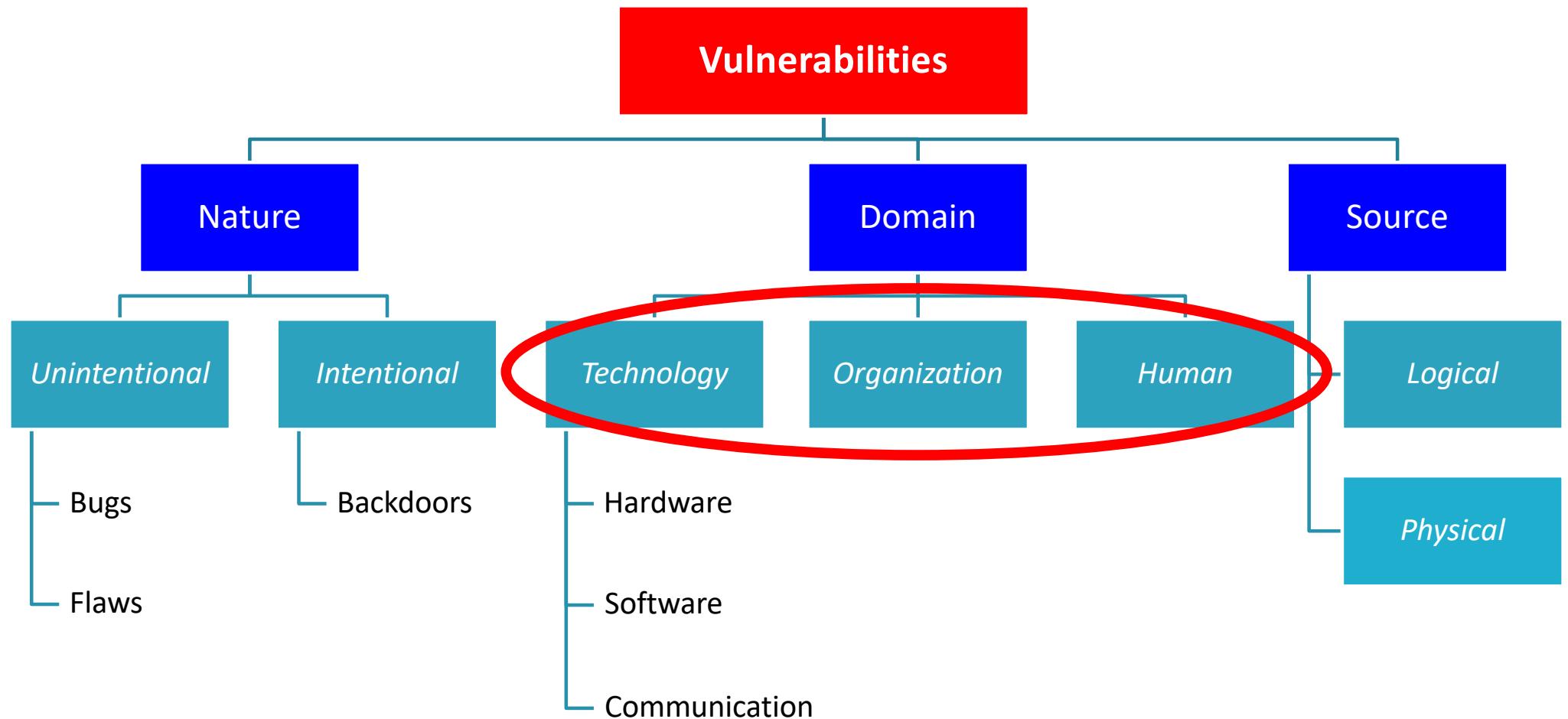
94

- Poor awareness and culture of ALL the involved people
- Wrong perception of risks
- *Social Engineering*

See lectures:

[*CS_1.5.1 EN - Social Engineering - How & Why*](#)

[*CS_1.5.2 EN - Social Engineering - Attack Vectors & Prevention*](#)



Ranking

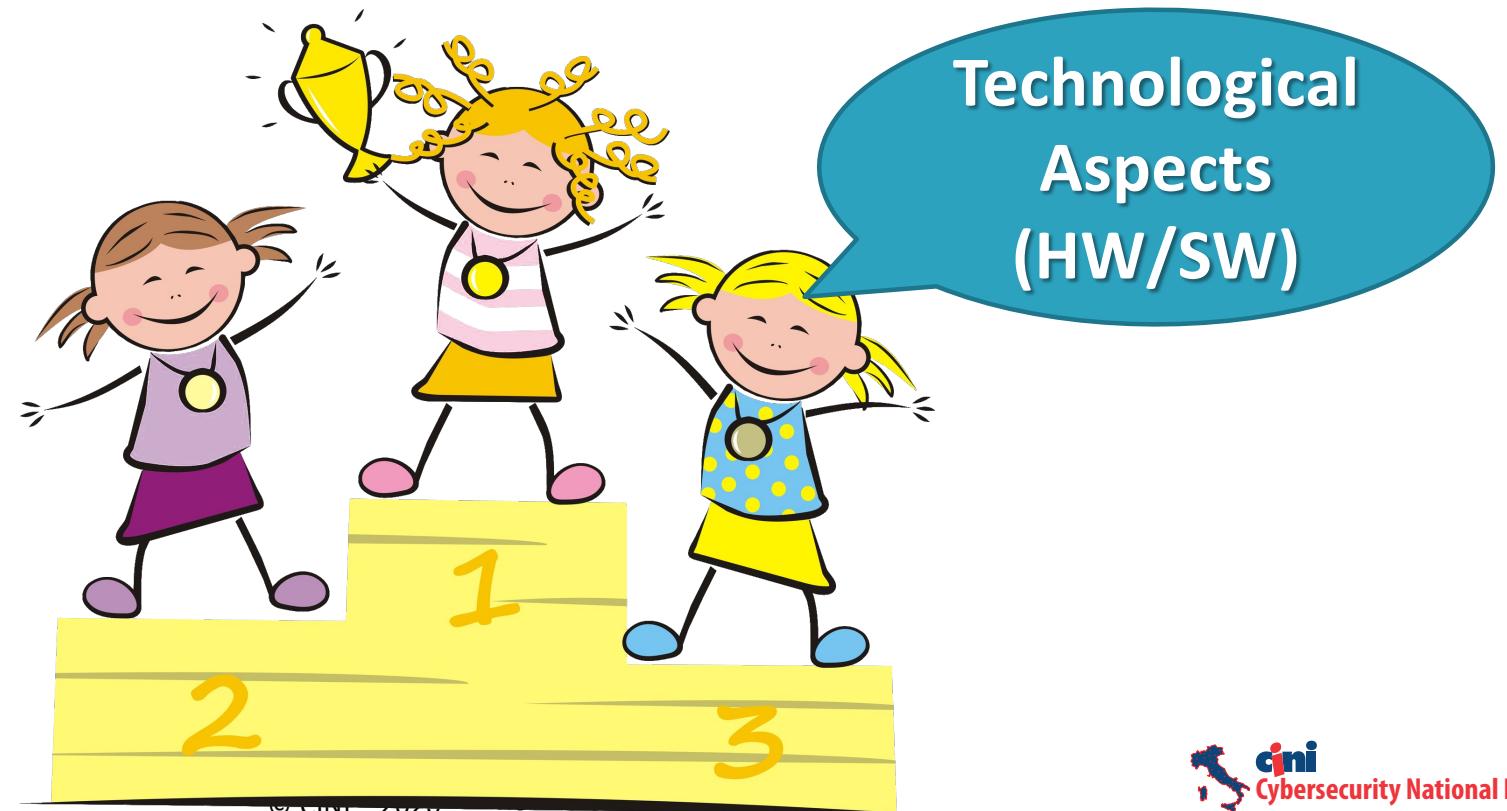
96

- How ranking them in terms of dangerousness w.r.t. security?

Ranking



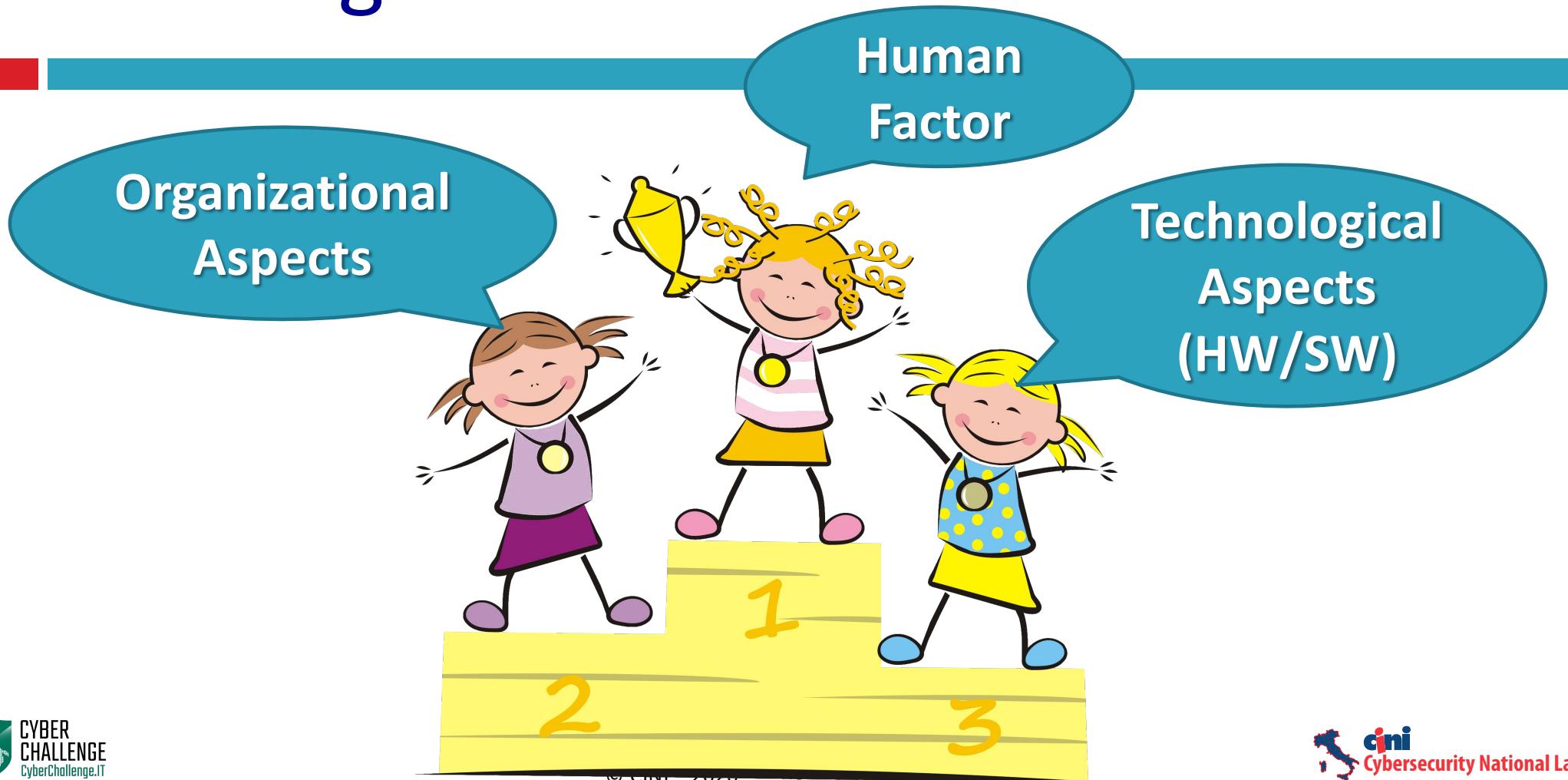
Ranking

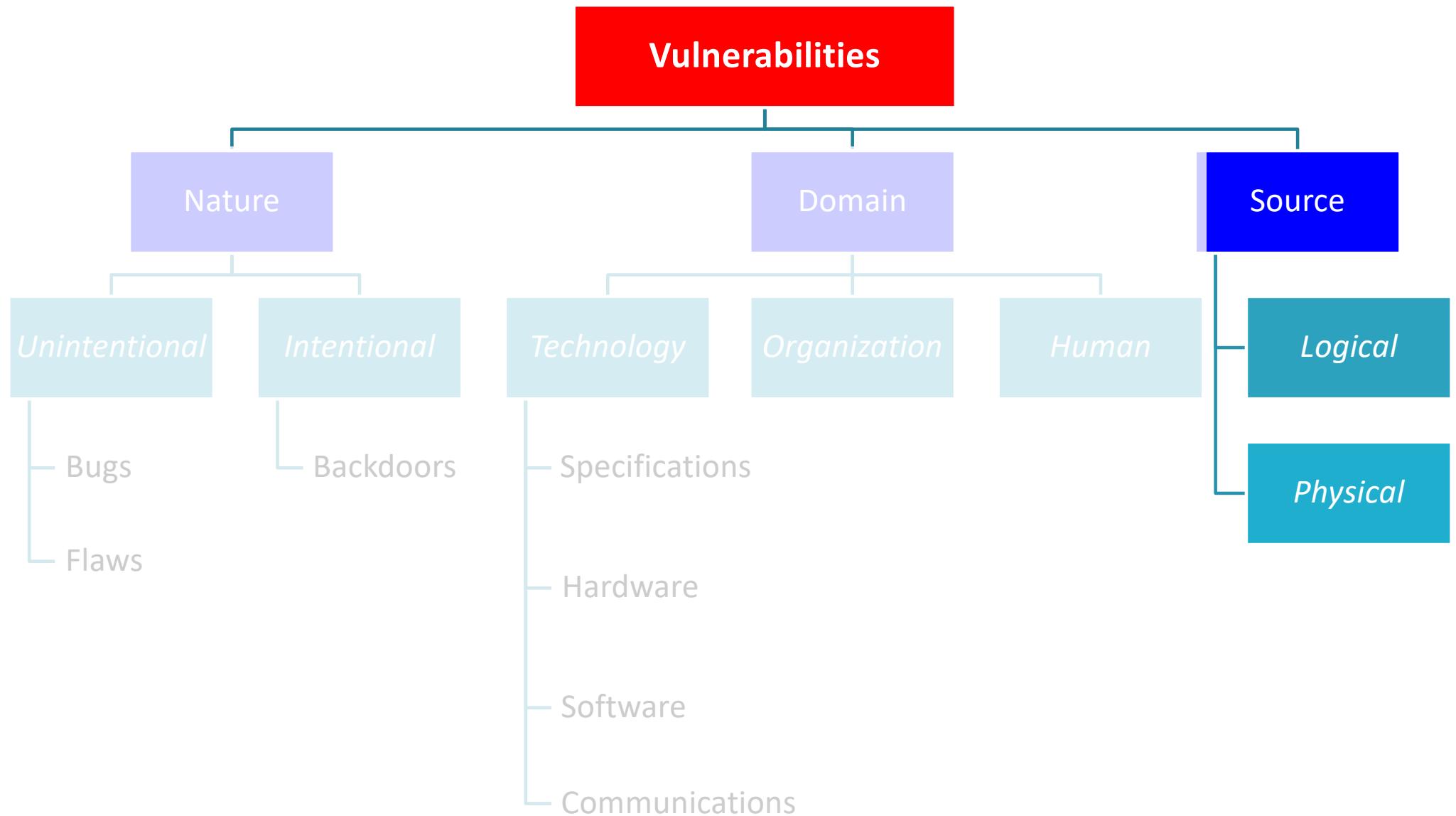


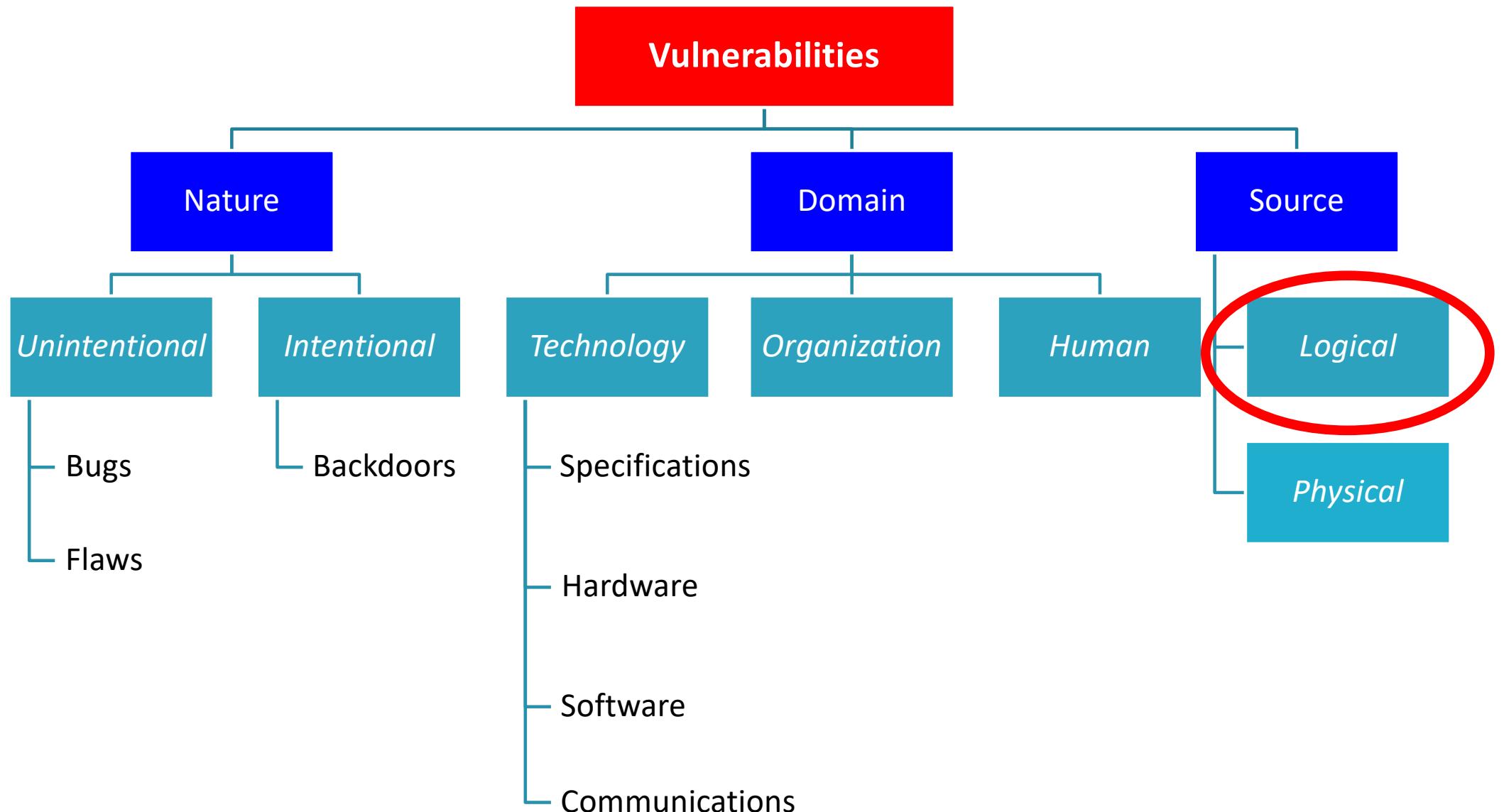
Ranking



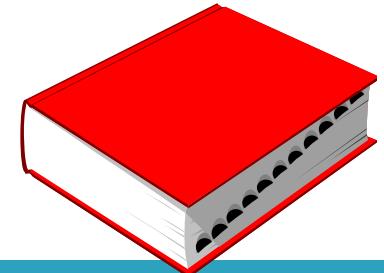
Ranking





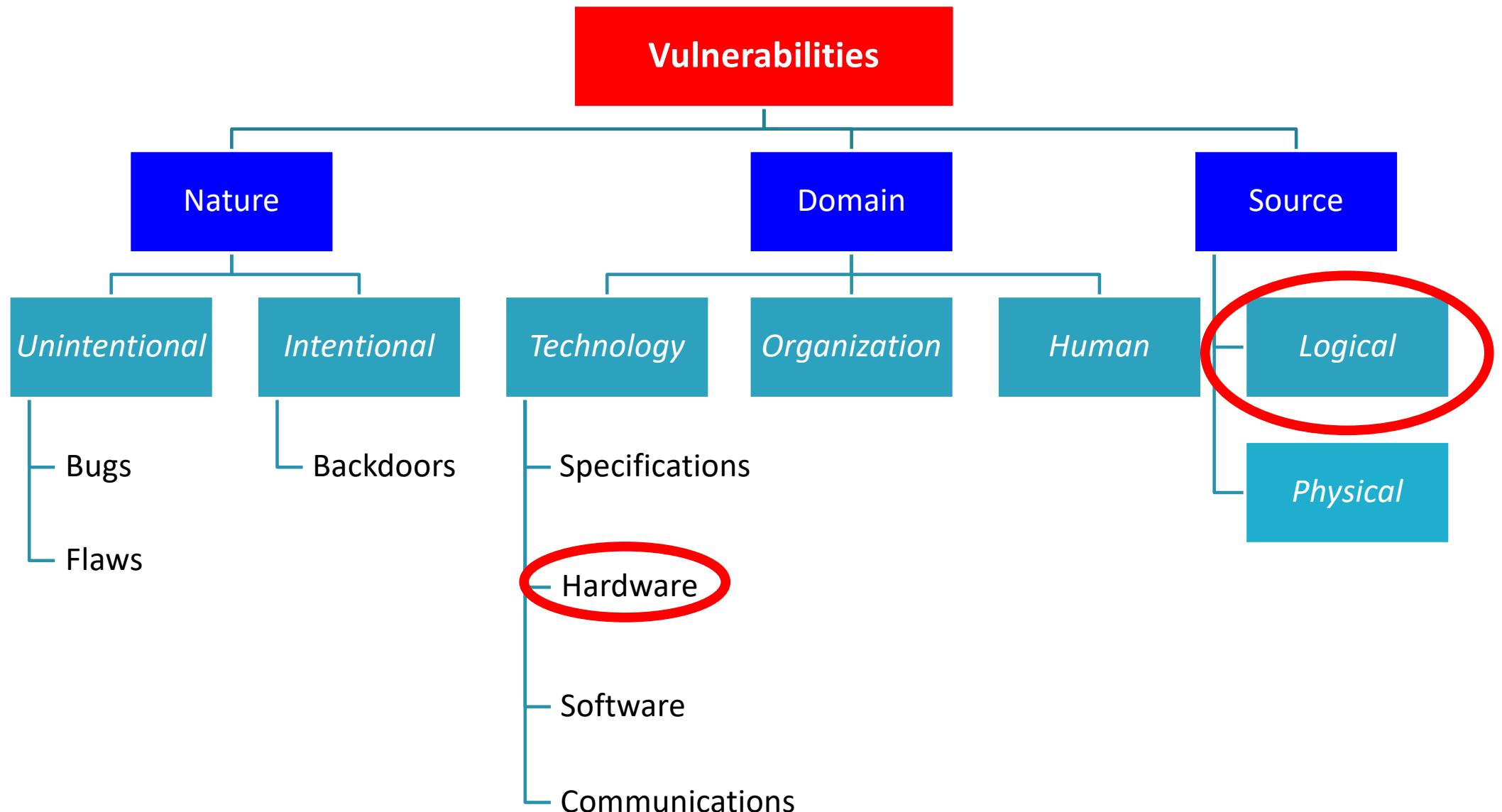


Logical Vulnerability



103

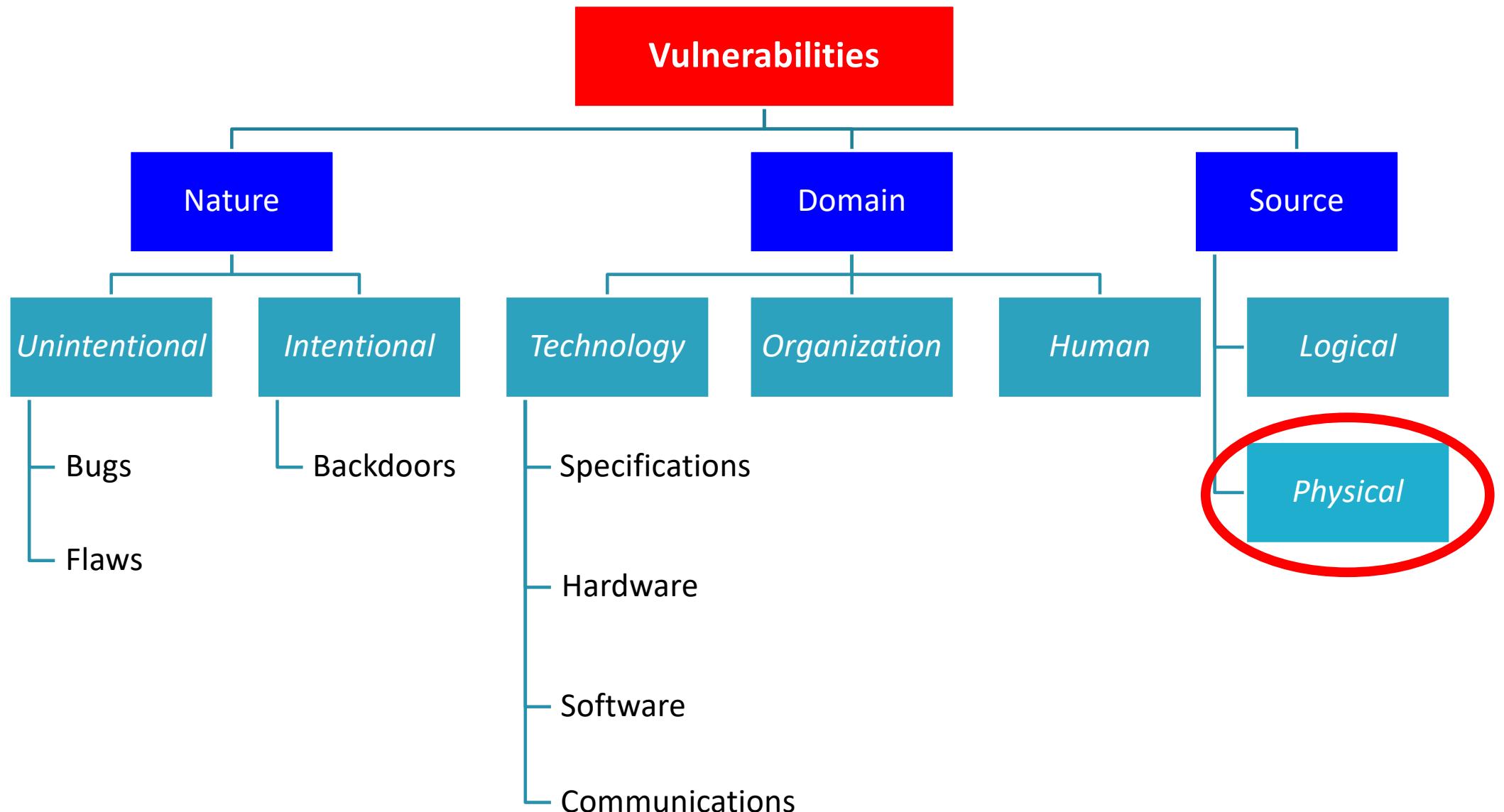
- A vulnerability that is introduced in the design phases as consequence of additions, omissions, shortcomings or errors in the drafting of the various abstract representations of the target system



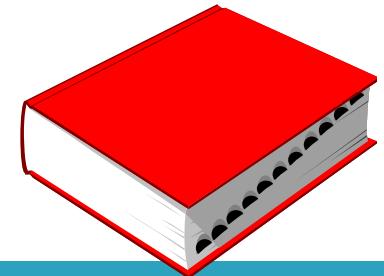
Example of Logical Hardware Vulnerability

105

- Meltdown and Spectre: both stem from choices made when the CPU behavior was logically described in VHDL/Verilog

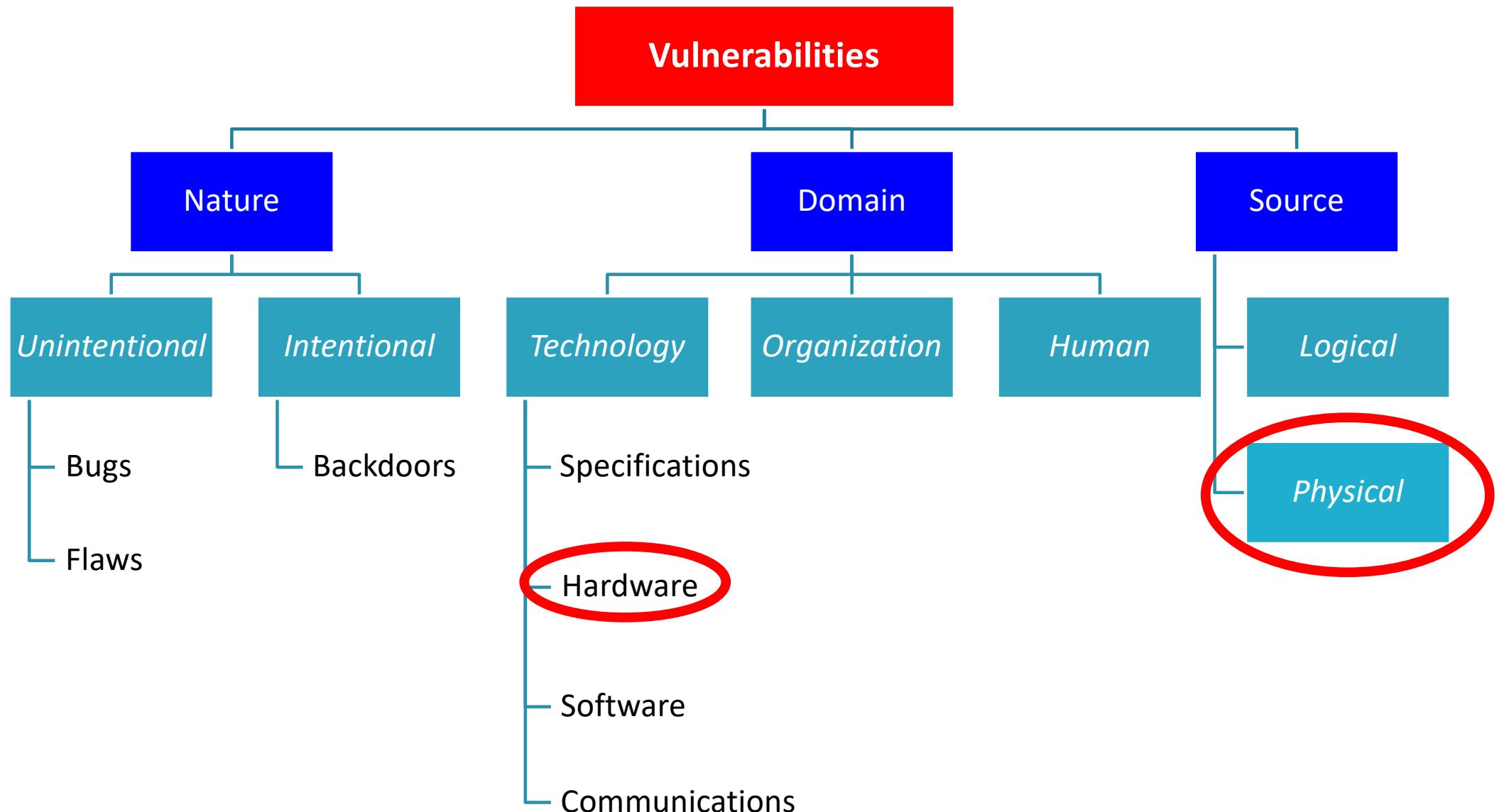


Physical Vulnerability



107

- A vulnerability introduced by the technology used to implement the final product.



Examples of Physical Hardware Vulnerability

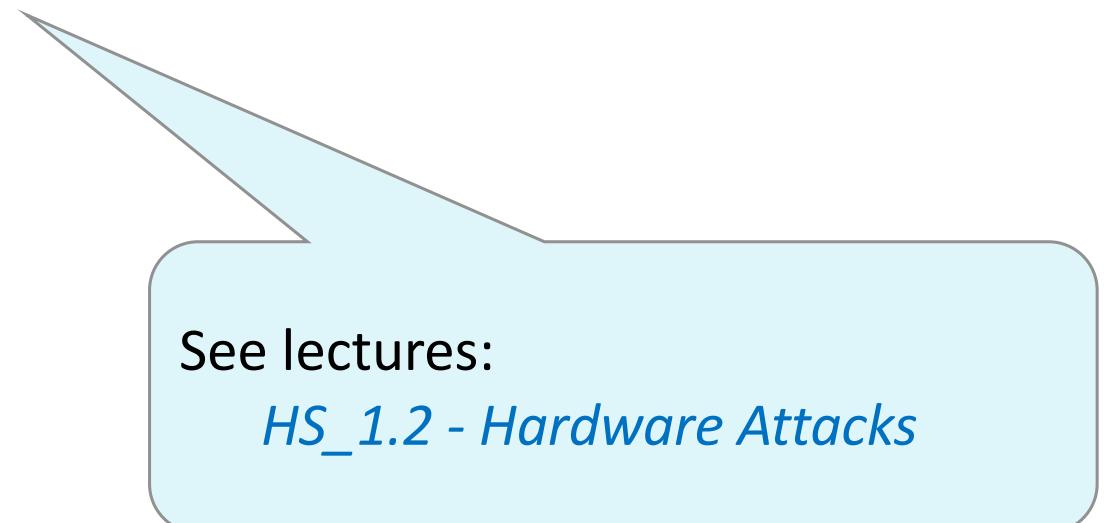
109

- We are going to introduce just a couple of examples:
 - *Row Hammer in DRAM memories*
 - *Side-Channel Attacks*

Examples of Physical Hardware Vulnerability

110

- We are going to introduce just a couple of examples:
 - *Row Hammer in DRAM memories*
 - *Side-Channel Attacks*



See lectures:

HS_1.2 - Hardware Attacks

Row Hammer in DRAM memories

111

- Caused by the physical and intrinsic phenomenon of electric coupling between memory cells due to the used technology

Side-Channel Attacks

112

- Hardware devices unintentionally release in the surrounding environment several clues:

Side-Channel Attacks

113

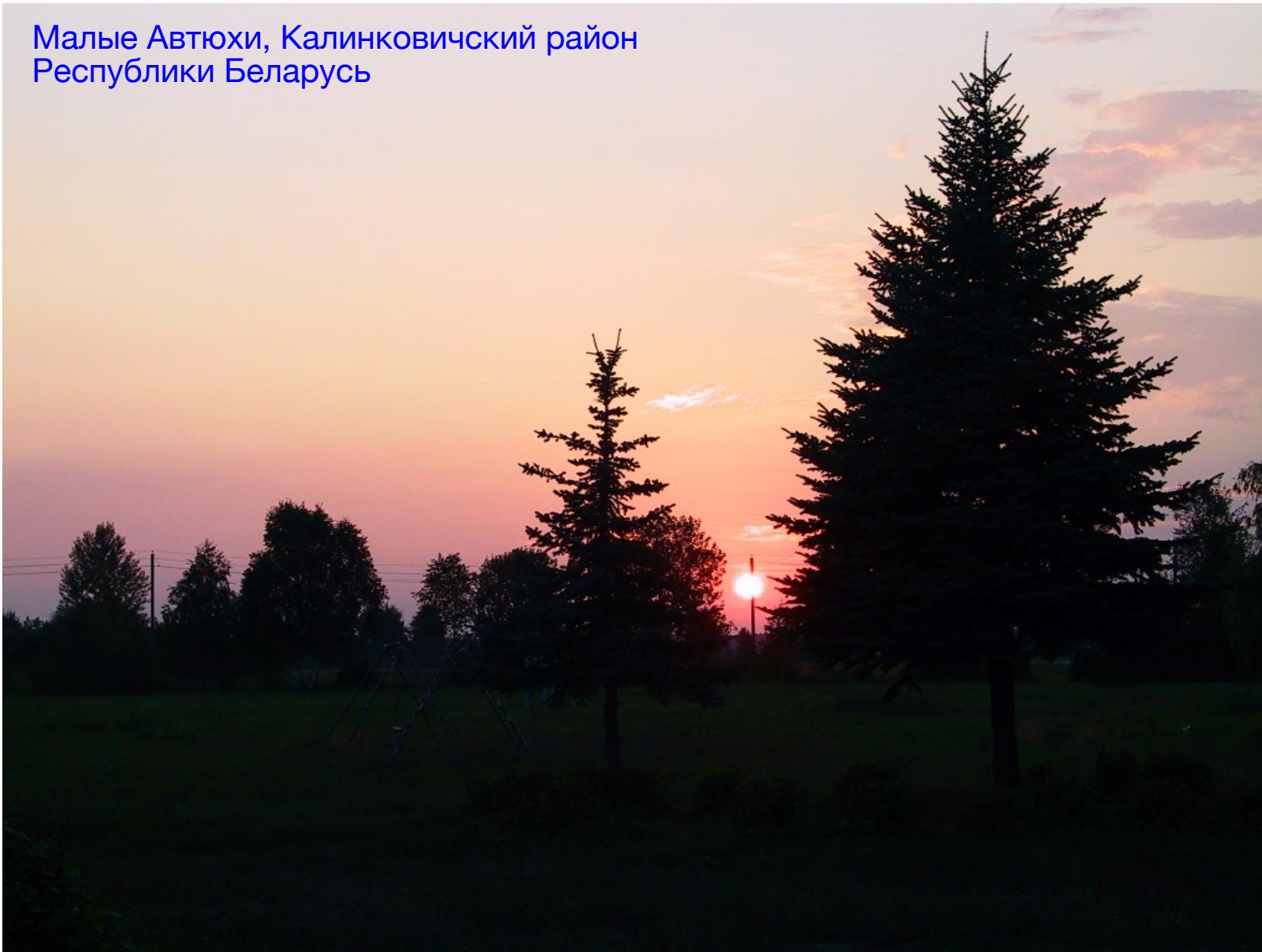
- Hardware devices unintentionally release in the surrounding environment several clues:
 - Spent time
 - Spent energy
 - Emitted electromagnetic radiation
 - Emitted Noise
 - Emitted Light
 - ...

Consequences

114

- A same design implemented resorting to different technologies can show different vulnerabilities.

Малые Автюхи, Калинковичский район
Республики Беларусь



Paolo PRINETTO

Director
CINI Cybersecurity
National Laboratory
Paolo.Prinetto@polito.it
Mob. +39 335 227529



<https://cybersecnatlab.it>