CYBER CHALLENGE
CyberChallenge.IT

cini
Cybersecurity National Lab

SPONSOR PLATINUM

accenture security
aizoon TECHNOLOGY CONSULTING AUSTRALIA EUROPE USA
B5
EY Building a better working world
eni
exprivia | ITALTEL
IBM
KPMG
LEONARDO
NTT DATA Trusted Global Innovator
NUMERA SISTEMI E INFORMATICA S.p.A.
Telsy

SPONSOR GOLD

bip.
cisco
MONTE DEI PASCHI DI SIENA BANCA DAL 1472
negg
NOVANEXT connecting the future
pwc

SPONSOR SILVER

DiGi ONE the leading digital company
ICT CYBER CONSULTING

Michele LORETI
Univ. di Camerino
michele.loreti@unicam.it

Secure Programming

CYBER CHALLENGE
CyberChallenge.IT

cini
Cybersecurity National Lab

https://cybersecnatlab.it

License & Disclaimer

Outline

- Software vulnerabilities
- Secure Programming
- Secure Software Development
  - Defensive Programming
  - Security by Design

Software vulnerabilities…

A *vulnerability* is a weakness which can be exploited by an *attacker* to perform *unauthorized actions* within your program.

To exploit a vulnerability, an attacker relies on tool and techniques related to a software weakness.

In this context, vulnerability is also known as the attack surface.

What is a *vulnerability?*

Question: Is the following code *vulnerable?*

```
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;
    }
}
```

What is a *vulnerability?*

Question: Is the following code *vulnerable?*

Hardcoded password

```
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;
    }
}
```

What is a *vulnerability?*

Question: Is the following code *vulnerable?*

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;
    }
}
```

What is a *vulnerability?*

Question: Is the following code *vulnerable?*

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;
    }
}
```


What is a *vulnerability?*

Question: Is the following code *vulnerable?*

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;
    }
}
```

What is a *vulnerability?*

Question: Is the following code *vulnerable?* ***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;
    }
}
```

Vulnerabilities vs bugs...

- Vulnerabilities are more general than bugs.
- According to the IEEE Terminology[1]
  - *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.

[1] IEEE Standard Glossary of Software Engineering Terminology

Code vulnerabilities...

- *Information leakage*
- *Buffer overflow*
- *Race condition*
- *Invalid data processing*

Code vulnerabilities...

- Information leakage: information is unintentionally disclosed to the end-user and used by attackers to breach application security.

Code vulnerabilities...

- Information leakage: information is unintentionally discl… s to brea…

```c
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;
    }
}
```

Code vulnerabilities...

- Buffer overflow: writing operations on data can overrun the buffer's boundary and overwrites adjacent memory locations. This may cause:
  - execution of malicious code;
  - privileges escalation.

Code vulnerabilities…

- **Buffer overflow:** writing operations on data can overrun
the b … memory
locat …
  - exe…
  - pri…

```c
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;
    }
}
```

Code vulnerabilities…

- Race condition: the small window of time between appliance of a security control and use of services in a system allows to change program behaviour…
  - It is the result of interferences among multiple threads running in the system and sharing the same resources.

Code vulnerabilities…

- Race condition: the small window of time between appl … vices in a sys … r…
  - It i … eads ru … rces.

```c
int raceCondition() {
  char * fn = "/tmp/XYZ";
  char buffer[60];
  FILE *fp;
  if(!access(fn, W_OK)){
    scanf("%50s", buffer );
    fp = fopen(fn, "a+");
    fwrite("\n", sizeof(char), 1, fp);
    fwrite(buffer, sizeof(char), strlen(buffer), fp);
    fclose(fp);
  } else {
    printf("No permission \n");
  }
}
```

Code vulnerabilities…

- Race condition: the small window of time between
  appl … vices in
  a sys … r…
  - It i … eads
    ru … rces.

```
int raceCondition() {
  char * fn = "/tmp/XYZ";
  char buffer[60];
  FILE *fp;
  if(!access(fn, W_OK)){
    scanf("%50s", buffer );
    fp = fopen(fn, "a+");
    fwrite("\n", sizeof(char), 1, fp);
    fwrite(buffer, sizeof(char), strlen(buffer), fp);
    fclose(fp);
  } else {
    printf("No permission \n");
  }
}
```

Access rights are verified

Code vulnerabilities…

- Race condition: the small window of time between appl … vices in a sys … r…
  - It i … eads ru … rces.

```c
int raceCondition() {
  char * fn = "/tmp/XYZ";
  char buffer[60];
  FILE *fp;
  if(!access(fn, W_OK)){
    scanf("%50s", buffer );
    fp = fopen(fn, "a+");
    fwrite("\n", sizeof(char), 1, fp);
    fwrite(buffer, sizeof(char), strlen(buffer), fp);
    fclose(fp);
  } else {
    printf("No permission \n");
  }
}
```

While the program is waiting for an input, file can be linked to another /etc/shadow

Code vulnerabilities...

- Invalid data processing: data are processed with wrong or partial assumptions that allow the attacker to enable unwanted behaviours or execute malicious code.

Sources of vulnerabilities…

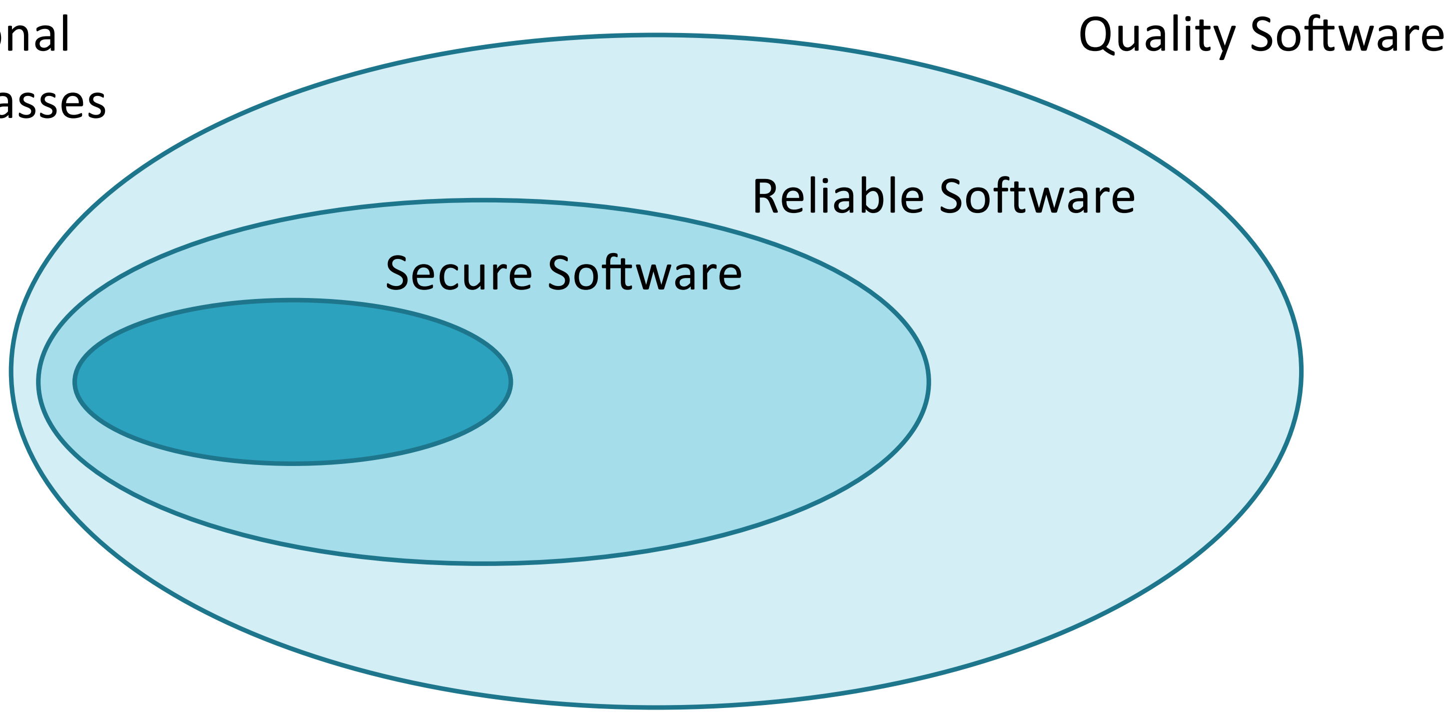
- Complexity, inadequacy, and (uncontrolled) changes
- Incorrect or changing assumptions (capabilities, inputs, outputs)
- Flawed specifications and designs
- Poor implementation of software interfaces (input validation, error and exception handling)
- Unintended, unexpected interactions with other components with the software's execution environment
- ***Inadequate knowledge of secure coding practices***

What is secure programming?

Secure programming is the practice of developing computer software so that it is protected ***from all kinds*** of *vulnerabilities*, attacks or anything that can cause harm to the software or the system using it.

Secure programming…

Security is a non-functional property in the wider classes of reliable and quality software.



Secure programming… a hard task!

- The Attacker's Advantages and the Defender's Dilemma[1]:
  1. The defender must defend all points, the attacker can choose the weakest point;
  2. The defender can defend only against known attacks, the attacker can probe for unknown vulnerabilities;
  3. The defender must be constantly vigilant, the attacker can strike at will;
  4. The defender must play the rules, the attacker can play dirty.
- Security principles are needed to drive our development process.

[1]Writing Secure Code (Developer Best Practice), David Le Blanc

Security principles

- Learn from mistakes
  - When a security problem is found in a software, we must recognize the problem and learn from it:
    - How did the security error occur?
    - Is the problem replicated in other areas of our code?
    - How could we have prevented this error from occurring?
    - How do we make sure this kind of error does not happen in the future?
    - Do we need to update our analysis tools?
  - Write a report about the security issue.

[1] Writing Secure Code (Developer Best Practice), David Le Blanc

Security principles

- Minimize your attack surface: Limit the enabled modules, services, and interfaces to those needed and used
  - Modules that are not needed should be disabled/removed;
  - Identify secure configurations of your product/software;
  - This applies for large/modular software where components can be enabled/disabled *on need.*

[1] Writing Secure Code (Developer Best Practice), David Le Blanc

Security principles

- Use defense in depth: Do not rely on other systems to protect your software.
  - Take security issues into account at each layer of your application.
- Assume external systems are insecure: Consider any data received from a system that is out of your control as insecure and as a possible source of attack
  - This is particularly important when you receive input from users!

[1] Writing Secure Code (Developer Best Practice), David Le Blanc

Defensive programming

- This is a programming technique that aims to design and implement software that can continue to function even when under attack:
  - Requires attention to all aspects of program execution, environment, and type of processed data
  - Requires that erroneous conditions resulting from some attack are checked
- Key rule is to never assume anything
  - check all assumptions and handle any possible error state!

[1] Writing Secure Code (Developer Best Practice), David Le Blanc

Defensive programming

- Programmers often make assumptions about the type of inputs a program will receive and the environment it executes in
  - Assumptions need to be validated by the program and all potential failures handled gracefully and safely
- Requires a changed mindset to traditional programming practices
  - Programmers need to understand how failures may occur and which steps are needed to reduce the chance for them to occurr

[1] IEEE Standard Glossary of Software Engineering Terminology

Secure Software Development

- Software Security cannot be an aspect considered at developing time
- Security must be considered in all the phases of software development:
  - Design
  - Development
  - Test
  - Ship/Maintenance

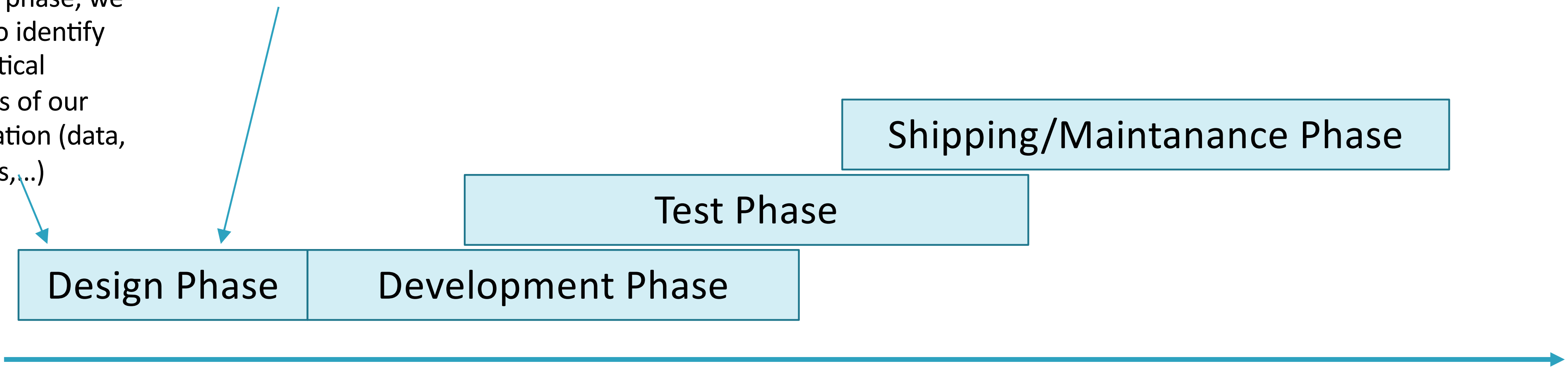
Security by Design

[1]IEEE Standard Glossary of Software Engineering Terminology

Secure Software Development

Since the very beginning of design phase, we have to identify the critical aspects of our application (data, devices,...)

Threat modelling



[1] Writing Secure Code (Developer Best Practice), David Le Blanc

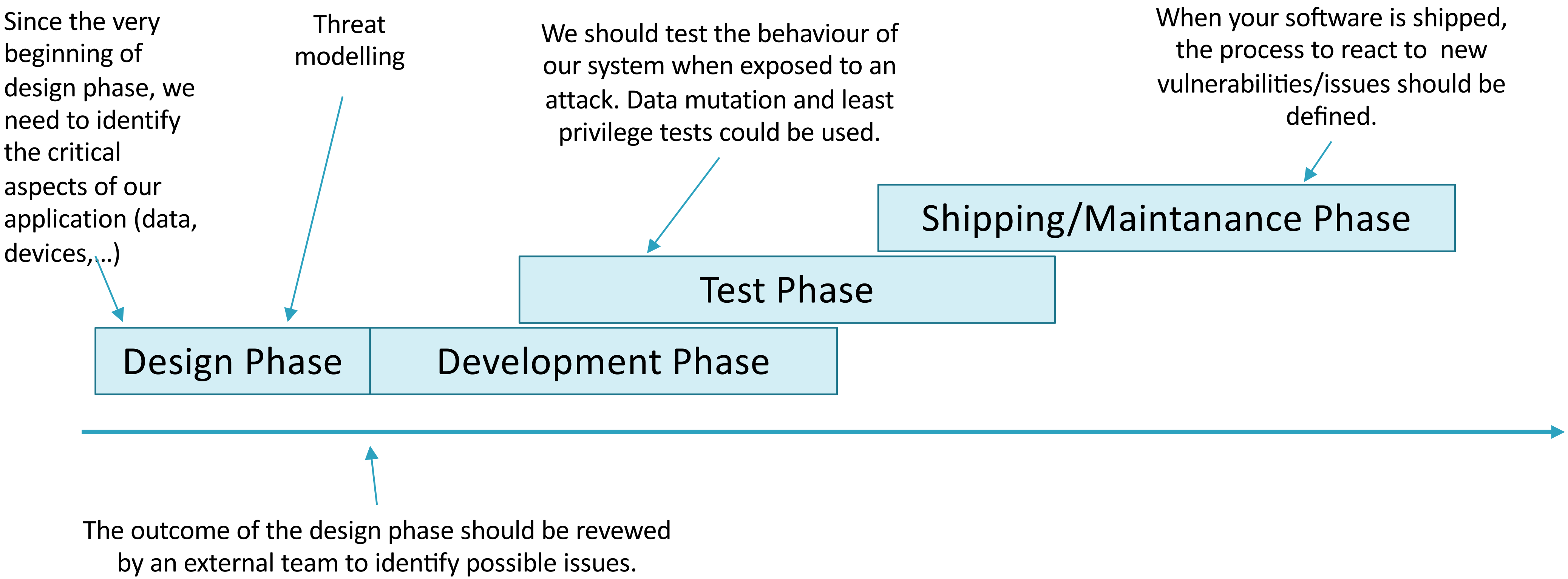
Threat modelling

- A threat model is a security-based analysis aiming at identifying the highest-level security risks posed to a product and the way attacks can carried on

- The goal is to identify which threats require mitigation and how to mitigate the threats.

[1] Writing Secure Code (Developer Best Practice), David Le Blanc

Secure Software Development



[1]Writing Secure Code (Developer Best Practice), David Le Blanc

Secure Programming

Michele LORETI
Univ. di Camerino
michele.loreti@unicam.it

CYBER CHALLENGE
CyberChallenge.IT

cini Cybersecurity National Lab

https://cybersecnatlab.it

SPONSOR PLATINUM

SPONSOR GOLD

SPONSOR SILVER