

UE SCLAM

Sécurité Logicielle

Lecture 1: introduction

Master M2 Cybersécurité et Informatique Légale

Academic Year 2025 - 2026

Who are we ?

Teaching staff

- ▶ Laurent Mounier (UGA)
- ▶ research within Verimag Lab
- ▶ research focus: formal verification, code analysis, compilation techniques, language semantics ... and **(software) security** !

Attendees

- ▶ Master M2 CSI

→ various skills, backgroud, interests ... and **apprenticeship tasks**

Sécurité des Composants et des Logiciels et Applications Multimédia

- ▶ 42 heures de cours/TD et 37 heures de TP/TD
- ▶ ~ 4 intervenants
- ▶ Thèmes couverts :
 - ▶ Sécurité Logicielle
 - ▶ Sécurité Matérielle
 - ▶ Rétro-ingénierie
 - ▶ Sécurité des Systèmes Embarqués
 - ▶ etc.
- ▶ Thème “Sécurité Logicielle”
 - ▶ an overview of software security and secure programming
 - ▶ some tools and techniques for software security

Evaluation de l'UE

Les règles du jeu ...

Plusieurs notes possibles

- ▶ compte-rendus de TP
- ▶ (courtes) présentations orales
- ▶ mini "projets"
- ▶ Quizz, QCMs, etc.

→ une **moyenne pondérée** de l'ensemble de ces notes ...

Course user manual

An (on-going) course web page on **Moodle** ...

`https://im2ag-moodle.univ-grenoble-alpes.fr/course/view.php?id=367`

- ▶ course schedule and materials (slides, etc.)
- ▶ weekly, reading suggestions, to complete the lecture
- ▶ other background reading/browsing advices ...

During the classes ...

Alternation between lectures, written exercices, lab exercises ...

...but no “formal” lectures → questions & discussions always welcome !

heterogeneous audience + open topics ⇒ **high interactivity level !**

Prerequisites

Ideally ...

This course is concerned with:

Programming languages

- ▶ at least one (classical) imperative language:
C or C++, Java, maybe Python ...
- ▶ some notions on compilation & (informal) language semantics

What happens behind the curtain

Some notions about:

- ▶ assembly code (ARM, x86, others ...)
- ▶ runtime memory layout (stack, heap)

Outline

Some practical information

What **software security** is (not) about ?

About software security

The context: computer system security . . .

Question 1: what is a “computer system”, or an **execution platform** ?

The context: computer system security ...

Question 1: what is a “computer system”, or an **execution platform** ?

Many possible incarnations, e.g.:

- ▶ (classical) computer: mainframe, server, desktop, laptop, etc.
- ▶ mobile device: phone, tablets, audio/video player, etc.
... up to IoT, smart cards, ...
- ▶ embedded (networked) systems: inside a car, a plane, a washing-machine, etc.
- ▶ cloud/remote computing, virtual execution environment
- ▶ but also industrial networks (Scada), ... etc.
- ▶ and certainly many more !

The context: computer system security ...

Question 1: what is a “computer system”, or an **execution platform** ?

Many possible incarnations, e.g.:

- ▶ (classical) computer: mainframe, server, desktop, laptop, etc.
- ▶ mobile device: phone, tablets, audio/video player, etc.
... up to IoT, smart cards, ...
- ▶ embedded (networked) systems: inside a car, a plane, a washing-machine, etc.
- ▶ cloud/remote computing, virtual execution environment
- ▶ but also industrial networks (Scada), ... etc.
- ▶ and certainly many more !

→ 2 main characteristics:

- ▶ include hardware + **software**
- ▶ open/connected to the **outside world** ...

The context: computer system security ... (ct'd)

Question 2: what does mean **security** ?

¹could be the user, or the **execution platform**, or even a **backdoor** ...

The context: computer system security ... (ct'd)

Question 2: what does mean **security** ?

- ▶ a set of general **security** properties: CIA
Confidentiality, Integrity, Availability (+ Non Repudiation + Anonymity + ...)
- ▶ concerns the **running** software + the whole **execution platform**
(other users, shared resources and data, peripherals, network, etc.)
- ▶ depends on an **intruder model**
→ an “external actor”¹ with an **attack objective** in mind, and able to elaborate a dedicated strategy to achieve it ($\neq$ hazards)
↔ something beyond **safety** and **fault-tolerance**

→ A possible definition:

- ▶ fonctionnal properties = what the system should do (without intruders)
- ▶ security properties = what **should be preserved** w.r.t the intruder models
(ex: **some** fonctionnal properties only + CIA on the execution platform)

Rk: fonctionnal properties do matter for “security-oriented” software (firewalls, etc.)!

¹could be the user, or the **execution platform**, or even a **backdoor** ...

Example 1: password authentication

Is this code “secure” ?

```
boolean verify (char[] input, char[] passwd , byte len) {  
    // No more than triesLeft attempts  
    if (triesLeft < 0) return false ; // no authentication  
    // Main comparison  
    for (short i=0; i <= len; i++)  
        if (input[i] != passwd[i]) {  
            triesLeft-- ;  
            return false ; // no authentication  
        }  
    // Comparison is successful  
    triesLeft = maxTries ;  
    return true ; // authentication is successful  
}
```

functional property:

$$\text{verify}(\text{input}, \text{passwd}, \text{len}) \Leftrightarrow \text{input}[0..\text{len}] = \text{passwd}[0..\text{len}]$$

Example 1: password authentication

Is this code “secure” ?

```
boolean verify (char[] input, char[] passwd , byte len) {  
    // No more than triesLeft attempts  
    if (triesLeft < 0) return false ; // no authentication  
    // Main comparison  
    for (short i=0; i <= len; i++)  
        if (input[i] != passwd[i]) {  
            triesLeft-- ;  
            return false ; // no authentication  
        }  
    // Comparison is successful  
    triesLeft = maxTries ;  
    return true ; // authentication is successful  
}
```

functional property:

$$\text{verify}(\text{input}, \text{passwd}, \text{len}) \Leftrightarrow \text{input}[0..\text{len}] = \text{passwd}[0..\text{len}]$$

What do we want to protect ? Against what ?

Example 1: password authentication

Is this code “secure” ?

```
boolean verify (char[] input, char[] passwd , byte len) {  
    // No more than triesLeft attempts  
    if (triesLeft < 0) return false ; // no authentication  
    // Main comparison  
    for (short i=0; i <= len; i++)  
        if (input[i] != passwd[i]) {  
            triesLeft-- ;  
            return false ; // no authentication  
        }  
    // Comparison is successful  
    triesLeft = maxTries ;  
    return true ; // authentication is successful  
}
```

functional property:

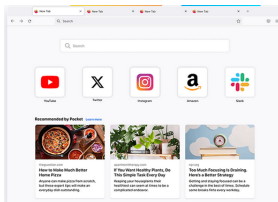
$$\text{verify}(\text{input}, \text{passwd}, \text{len}) \Leftrightarrow \text{input}[0..\text{len}] = \text{passwd}[0..\text{len}]$$

What do we want to protect ? Against what ?

- ▶ confidentiality of `passwd`, information leakage ?
- ▶ no unexpected runtime behaviour
- ▶ code integrity, etc.

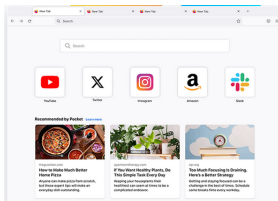
Example 2: web browser

Unavoidable applications, key functionalities, routinely used . . .



Example 2: web browser

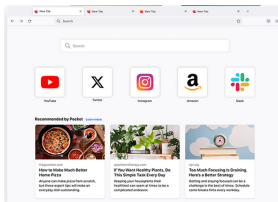
Unavoidable applications, key functionalities, routinely used . . .



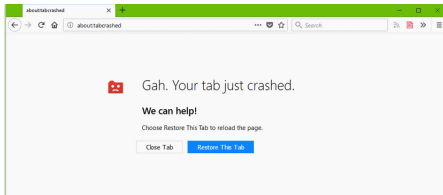
But, **quite often**:

Example 2: web browser

Unavoidable applications, key functionalities, routinely used . . .



But, quite often:



Is it a simple **functionnality issue?**
(no damage, user simply needs to restart its browser, right?)

No, we **need** to bother about crashes!

crashes are due to some unexpected run-time error

- ▶ not foreseen by the programmer and compiler/interpreter ...
- ▶ ... and not accurately enough trapped at runtime

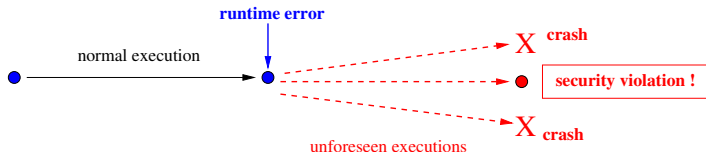
No, we **need** to bother about crashes!

crashes are due to some unexpected run-time error

- ▶ not foreseen by the programmer and compiler/interpreter ...
- ▶ ... and not accurately enough trapped at runtime

consequence: some part of the execution:

- ▶ may take place **outside the program scope**
(not following the regular program semantic)
- ▶ and could be **controled/exploited** by an attacker (~ “weird machine”)



⇔ may **break** all security properties ...
from simple denial-of-service to **arbitrary code execution**

Rk: may also happen **silently** (without any crash !)

Some (not standardized) definitions . . .

Some (not standardized) definitions ...

Bug: an error (or defect/ flaw/ failure) introduced in a SW, either

- ▶ at the specification / design / algorithmic level
- ▶ at the programming / coding level
- ▶ or even by the compiler (or any other pgm transformation tools) ...

Some (not standardized) definitions ...

Bug: an error (or defect/flaw/failure) introduced in a SW, either

- ▶ at the specification / design / algorithmic level
- ▶ at the programming / coding level
- ▶ or even by the compiler (or any other pgm transformation tools) ...

Vulnerability: a weakness (for instance a bug !) that opens a “security breach”

- ▶ **non exploitable** vulnerabilities: there is no (known !) way for an attacker to use this bug to corrupt the system
- ▶ **exploitable** vulnerabilities: this bug can be used to elaborate an attack (i.e., write an **exploit**)
- ▶ **0-day** vulnerabilities: yet unpublished (hence **not patched** !)

Some (not standardized) definitions ...

Bug: an error (or defect/ flaw/ failure) introduced in a SW, either

- ▶ at the specification / design / algorithmic level
- ▶ at the programming / coding level
- ▶ or even by the compiler (or any other pgm transformation tools) ...

Vulnerability: a weakness (for instance a bug !) that opens a “security breach”

- ▶ **non exploitable** vulnerabilities: there is no (known !) way for an attacker to use this bug to corrupt the system
- ▶ **exploitable** vulnerabilities: this bug can be used to elaborate an attack (i.e., write an **exploit**)
- ▶ **0-day** vulnerabilities: yet unpublished (hence **not patched** !)

Exploit: a concrete attacker behavior allowing to:

1. trigger a (sequence of) vulnerability(-ies)
2. leading to a security property violation

Ex: a single program input, or a complex sequence of interactions with the target program and/or its execution environment ...

Software vulnerability examples

Case 1 (not so common ...)

Functional property not provided by a security-oriented component

- ▶ lack of encryption, too weak crypto-system,
- ▶ no (strong enough) authentication mechanism,
- ▶ bad firewall configuration, too weak access control enforcement rules,
- ▶ etc.

Software vulnerability examples

Case 1 (not so common ...)

Functional property not provided by a security-oriented component

- ▶ lack of encryption, too weak crypto-system,
- ▶ no (strong enough) authentication mechanism,
- ▶ bad firewall configuration, too weak access control enforcement rules,
- ▶ etc.

Case 2 (the vast majority !)

Insecure coding practice in (any!) software component/application

- ▶ improper input validation $\rightsquigarrow$ SQL or code injection, XSS, etc.
- ▶ insecure shared resource management (file system, network)
- ▶ information leakage (lack of data encapsulation, side channels)
- ▶ exploitable coding errors (memory access, arithmetic overflows, etc.)
- ▶ etc.

⇒ **Sleeping bombs**



The intruder model

Who/what is the attacker ?

- ▶ a malicious external user, interacting via regular input sources e.g., keyboard, network (man-in-the-middle), etc.
- ▶ a malicious external “observer”, interacting via side channels (execution time, power consumption)
- ▶ another application running on the same platform interacting through shared resources like caches, processor elements, etc.
- ▶ the execution platform itself (e.g., when compromised !)

The intruder model

Who/what is the attacker ?

- ▶ a malicious external user, interacting via regular input sources e.g., keyboard, network (man-in-the-middle), etc.
- ▶ a malicious external “observer”, interacting via side channels (execution time, power consumption)
- ▶ another application running on the same platform interacting through shared resources like caches, processor elements, etc.
- ▶ the execution platform itself (e.g., when compromised !)

What is he/she/it able to do ?

At low level:

- ▶ unexpected memory read (data or code)
- ▶ unexpected memory write (data or code)

⇒ **powerful enough for**

- ▶ information disclosure
- ▶ unexpected/arbitrary code execution
- ▶ privilege elevation, etc.

Example: smartphone attack surface



Credits [BT2019]

Outline

Some practical information

What software security is (not) about ?

About software security

Some evidences regarding cyber (un)-security

So many examples of successful computer system attacks:

- ▶ the “**famous ones**”: (at least one per year !)
Morris worm, Stuxnet, Heartbleed, WannaCry, Spectre, Log4j, etc.
- ▶ the never-ending records of “**cyber-attacks**” against large organizations
(private companies, public structures)
- ▶ a public database of **CVEs** (Common Vulnerabilities and Exposures)
Numbers of CVEs per year
- ▶ etc.

Some evidences regarding cyber (un)-security

So many examples of successful computer system attacks:

- ▶ the “**famous ones**”: (at least one per year !)
Morris worm, Stuxnet, Heartbleed, WannaCry, Spectre, Log4j, etc.
- ▶ the never-ending records of “**cyber-attacks**” against large organizations
(private companies, public structures)
- ▶ a public database of **CVEs** (Common Vulnerabilities and Exposures)
Numbers of CVEs per year
- ▶ etc.

Why ? Who can we blame for that ??

Some evidences regarding cyber (un)-security

So many examples of successful computer system attacks:

- ▶ the “**famous ones**”: (at least one per year !)
Morris worm, Stuxnet, Heartbleed, WannaCry, Spectre, Log4j, etc.
- ▶ the never-ending records of “**cyber-attacks**” against large organizations (private companies, public structures)
- ▶ a public database of **CVEs** (Common Vulnerabilities and Exposures)
Numbers of CVEs per year
- ▶ etc.

Why ? Who can we blame for that ??

- ▶ $\nexists$ well defined recipe to build secure cyber systems in the large
- ▶ permanent trade-off between **efficiency** and **safety/security**:
 - ▶ HW and micro-architectures (**sharing** is everywhere !)
 - ▶ operating systems
 - ▶ programming languages and applications
 - ▶ coding and software engineering techniques

But, what about **software** security ?

Software is **greatly involved** in “computer system security”:

- ▶ it plays a major role in **enforcing security properties**:
crypto, authentication protocols, intrusion detection, firewall, etc.
- ▶ but it is also a major source of **security problems** . . .

“90 percent of security incidents result from exploits against defects in software” (U.S. DHS)

→ SW is clearly one of the **weakest links** in the security chain!

Why ???

But, what about **software** security ?

Software is **greatly involved** in “computer system security”:

- ▶ it plays a major role in **enforcing security properties**:
crypto, authentication protocols, intrusion detection, firewall, etc.
- ▶ but it is also a major source of **security problems** . . .

“90 percent of security incidents result from exploits against defects in software” (U.S. DHS)

→ SW is clearly one of the **weakest links** in the security chain!

Why ???

- ▶ we do not do very well how to write **secure** SW
we do not even know how to write **correct** SW!
- ▶ behavioral properties can't be validated on a (large) SW
impossible by hand, untractable with a machine
- ▶ programming languages not designed for security enforcement
most of them contain numerous traps and pitfalls
- ▶ programmers feel not (so much) concerned with security
security not get enough attention in programming/SE courses
- ▶ heterogenous and nomad applications favor unsecure SW
COTS, remote execution, mobile code, SaaS, etc.

But, what about **software** security ?

Software is **greatly involved** in “computer system security”:

- ▶ it plays a major role in **enforcing security properties**:
crypto, authentication protocols, intrusion detection, firewall, etc.
- ▶ but it is also a major source of **security problems** . . .

“90 percent of security incidents result from exploits against defects in software” (U.S. DHS)

→ SW is clearly one of the **weakest links** in the security chain!

Why ???

- ▶ we do not do very well how to write **secure** SW
we do not even know how to write **correct** SW!
- ▶ behavioral properties can't be validated on a (large) SW
impossible by hand, untractable with a machine
- ▶ programming languages not designed for security enforcement
most of them contain numerous traps and pitfalls
- ▶ programmers feel not (so much) concerned with security
security not get enough attention in programming/SE courses
- ▶ heterogenous and nomad applications favor unsecure SW
COTS, remote execution, mobile code, SaaS, etc.
- ▶ . . . and IA generated code is coming up!

Some concrete CVE examples: back to the browsers ...

🚩 CVE-2022-26485 Detail

Description

Removing an XSLT parameter during processing could have lead to an exploitable use-after-free. We have had reports of attacks in the wild abusing this flaw. This vulnerability affects Firefox < 97.0.2, Firefox ESR < 91.6.1, Firefox for Android < 97.3.0, Thunderbird < 91.6.2, and Focus < 97.3.0.

Metrics

CVSS Version 4.0

CVSS Version 3.x

CVSS Version 2.0

NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also displayed.

CVSS 3.x Severity and Vector Strings:



NIST: NVD

Base Score: **8.8 HIGH**

Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H

Some concrete CVE examples: back to the browsers ...

🚩 CVE-2022-26485 Detail

Description

Removing an XSLT parameter during processing could have lead to an exploitable use-after-free. We have had reports of attacks in the wild abusing this flaw. This vulnerability affects Firefox < 97.0.2, Firefox ESR < 91.6.1, Firefox for Android < 97.3.0, Thunderbird < 91.6.2, and Focus < 97.3.0.

Metrics

CVSS Version 4.0

CVSS Version 3.x

CVSS Version 2.0

NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also displayed.

CVSS 3.x Severity and Vector Strings:



NIST: NVD

Base Score: **8.8 HIGH**

Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H

🚩 CVE-2024-29944 Detail

AWAITING ANALYSIS

This vulnerability is currently awaiting analysis.

Description

An attacker was able to inject an event handler into a privileged object that would allow arbitrary JavaScript execution in the parent process. Note: This vulnerability affects Desktop Firefox only, it does not affect mobile versions of Firefox. This vulnerability affects Firefox < 124.0.1 and Firefox ESR < 115.9.1.

Metrics

CVSS Version 4.0

CVSS Version 3.x

CVSS Version 2.0

NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also displayed.

CVSS 4.0 Severity and Vector Strings:



NIST: NVD

N/A

NVD assessment not yet provided.

See the [online discussions](#) ...

A highly critical recent CVE example (Trojan Horse)

CVE-2024-3094 Detail

MODIFIED

This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting reanalysis which may result in further changes to the information provided.

Description

Malicious code was discovered in the upstream tarballs of xz, starting with version 5.6.0. Through a series of complex obfuscations, the liblzma build process extracts a prebuilt object file from a disguised test file existing in the source code, which is then used to modify specific functions in the liblzma code. This results in a modified liblzma library that can be used by any software linked against this library, intercepting and modifying the data interaction with this library.

Metrics

CVSS Version 4.0

CVSS Version 3.x

CVSS Version 2.0

NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also displayed.

CVSS 3.x Severity and Vector Strings:



CNA: Red Hat, Inc.

Base Score: 10.0 CRITICAL

Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

(see the [Pentest-Tools blog](#))

And more CVEs are still coming !

Some evidences regarding software (un)-security (ct'd)

An increasing activity in the “defender side” as well ...

Some evidences regarding software (un)-security (ct'd)

An increasing activity in the “defender side” as well ...

- ▶ all the daily security patches (for OS, basic applications, etc.)

Some evidences regarding software (un)-security (ct'd)

An increasing activity in the “defender side” as well ...

- ▶ all the daily security patches (for OS, basic applications, etc.)
- ▶ companies and experts specialized in software security
code audit, search for 0days, malware detection & analysis, etc.
“bug bounties” [<https://zerodium.com/program.html>]

Some evidences regarding software (un)-security (ct'd)

An increasing activity in the “defender side” as well ...

- ▶ all the daily security patches (for OS, basic applications, etc.)
- ▶ companies and experts specialized in software security
code audit, search for 0days, malware detection & analysis, etc.
“bug bounties” [<https://zerodium.com/program.html>]
- ▶ some important research efforts
from the main software editors (e.g., MicroSoft, Google, etc)
from the academia (conferences) and independent “ethical hackers” (blogs, etc.)

Some evidences regarding software (un)-security (ct'd)

An increasing activity in the “defender side” as well ...

- ▶ all the daily security patches (for OS, basic applications, etc.)
- ▶ companies and experts specialized in software security
code audit, search for 0days, malware detection & analysis, etc.
“bug bounties” [<https://zerodium.com/program.html>]
- ▶ some important research efforts
from the main software editors (e.g., MicroSoft, Google, etc)
from the academia (conferences) and independent “ethical hackers” (blogs, etc.)
- ▶ software verification tools editors start addressing security issues
e.g.: dedicated static analyser features

Some evidences regarding software (un)-security (ct'd)

An increasing activity in the “defender side” as well ...

- ▶ all the daily security patches (for OS, basic applications, etc.)
- ▶ companies and experts specialized in software security
code audit, search for 0days, malware detection & analysis, etc.
“bug bounties” [<https://zerodium.com/program.html>]
- ▶ some important research efforts
from the main software editors (e.g., MicroSoft, Google, etc)
from the academia (conferences) and independent “ethical hackers” (blogs, etc.)
- ▶ software verification tools editors start addressing security issues
e.g.: dedicated static analyser features
- ▶ international cooperation for vulnerability disclosure and classification
e.g.: CERT, CVE/CWE catalogue, vulnerability databases

Some evidences regarding software (un)-security (ct'd)

An increasing activity in the “defender side” as well ...

- ▶ all the daily security patches (for OS, basic applications, etc.)
- ▶ companies and experts specialized in software security
code audit, search for 0days, malware detection & analysis, etc.
“bug bounties” [<https://zerodium.com/program.html>]
- ▶ some important research efforts
from the main software editors (e.g., MicroSoft, Google, etc)
from the academia (conferences) and independent “ethical hackers” (blogs, etc.)
- ▶ software verification tools editors start addressing security issues
e.g.: dedicated static analyser features
- ▶ international cooperation for vulnerability disclosure and classification
e.g.: CERT, CVE/CWE catalogue, vulnerability databases
- ▶ government agencies to promote & control SW security
e.g.: ANSSI, ENISA, Darpa “Grand Challenge”, etc.

Some evidences regarding software (un)-security (ct'd)

An increasing activity in the “defender side” as well ...

- ▶ all the daily security patches (for OS, basic applications, etc.)
- ▶ companies and experts specialized in software security
code audit, search for 0days, malware detection & analysis, etc.
“bug bounties” [<https://zerodium.com/program.html>]
- ▶ some important research efforts
from the main software editors (e.g., MicroSoft, Google, etc)
from the academia (conferences) and independent “ethical hackers” (blogs, etc.)
- ▶ software verification tools editors start addressing security issues
e.g.: dedicated static analyser features
- ▶ international cooperation for vulnerability disclosure and classification
e.g.: CERT, CVE/CWE catalogue, vulnerability databases
- ▶ government agencies to promote & control SW security
e.g.: ANSSI, ENISA, Darpa “Grand Challenge”, etc.
- ▶ national/european/international regulations, norms and standards
e.g.: RGPD, NIS-2, Cyber Resilience Act, ISO 27001, IEC 62443

Couter-measures and protections (examples)

Several existing mechanisms to **enforce** SW security

Counter-measures and protections (examples)

Several existing mechanisms to **enforce** SW security

- ▶ at the programming level:
 - ▶ disclosed vulnerabilities → language weaknesses databases
↳ secure coding patterns and libraries
 - ▶ aggressive compiler options + code instrumentation
↳ early detection of unsecure code

Counter-measures and protections (examples)

Several existing mechanisms to **enforce** SW security

- ▶ at the programming level:
 - ▶ disclosed vulnerabilities → language weaknesses databases
↳ secure coding patterns and libraries
 - ▶ aggressive compiler options + code instrumentation
↳ early detection of unsecure code

- ▶ at the OS level:
 - ▶ sandboxing
 - ▶ address space randomization
 - ▶ non executable memory zones
 - ▶ etc.

Couter-measures and protections (examples)

Several existing mechanisms to **enforce** SW security

- ▶ at the programming level:
 - ▶ disclosed vulnerabilities → language weaknesses databases
↳ secure coding patterns and libraries
 - ▶ aggressive compiler options + code instrumentation
↳ early detection of unsecure code

- ▶ at the OS level:
 - ▶ sandboxing
 - ▶ address space randomization
 - ▶ non executable memory zones
 - ▶ etc.

- ▶ at the hardware level:
 - ▶ Trusted Platform Modules (TPM)
 - ▶ secure crypto-processor
 - ▶ CPU tracking mechanisms (e.g., Intel Processor Trace)
 - ▶ etc.

Techniques and tools for assessing SW security

Several existing mechanisms to **evaluate** SW security

- ▶ **code review** ...
- ▶ **fuzzing**:
 - ▶ run the code with “unexpected” inputs → **pgm crashes**
 - ▶ (tedious) manual check to find **exploitable** vulns ...
- ▶ **(smart) testing**:
 - coverage-oriented pgm exploration techniques
(genetic algorithms, dynamic-symbolic executions, etc.)
 - + code instrumentation to detect (low-level) vulnerabilities
- ▶ **static analysis**: approximate the code behavior to detect **potential** vulns
(~ code optimization techniques)

Techniques and tools for assessing SW security

Several existing mechanisms to **evaluate** SW security

- ▶ **code review** ...
- ▶ **fuzzing**:
 - ▶ run the code with “unexpected” inputs → **pgm crashes**
 - ▶ (tedious) manual check to find **exploitable** vulns ...
- ▶ **(smart) testing**:
 - coverage-oriented pgm exploration techniques
(genetic algorithms, dynamic-symbolic executions, etc.)
 - + code instrumentation to detect (low-level) vulnerabilities
- ▶ **static analysis**: approximate the code behavior to detect **potential** vulns
(~ code optimization techniques)

In practice:

- ▶ only **the binary code** is **always available** and **useful** ...
- ▶ **combinations** of all these techniques ...
- ▶ **exploitability analysis** still challenging ...

Course objectives (for the part 1)

Understand the **root causes** of common weaknesses in SW security

- ▶ at the programming language level
- ▶ at the execution platform level

→ helps to better choose (or deal with) a programming language

Learn some methods and techniques to **build more secure SW**:

- ▶ programming techniques:
languages, coding patterns, etc.
- ▶ validation techniques:
what can(not) bring existing tools ?
- ▶ counter-measures and protection mechanisms

Course agenda

See

<https://im2ag-moodle.univ-grenoble-alpes.fr/course/view.php?id=545>

Credits:

- ▶ E. Poll (Radboud University)
- ▶ M. Payer (Purdue University)
- ▶ E. Jaeger, O. Levillain and P. Chifflier (ANSSI)