



Software security, secure programming

Lecture 1: introduction

Master on Cybersecurity – Master MoSiG

Academic Year 2017 - 2018

Who are we ?

Teaching staff

- ▶ Laurent Mounier (UGA) & Marie-Laure Potet (Grenoble-INP)
- ▶ research within Verimag Lab (PACSS team)
- ▶ research focus: formal verification, code analysis, compilation techniques, language semantics ... and **(software) security** !

Attendees

- ▶ Master M2 on Cybersecurity
- ▶ Master MoSiG (optionnal): HECS, PDS, AISSE, others ?

→ various skills, backgroud and interests ...

Agenda

Part 1: an overview of software security and secure programming

- ▶ 6 weeks (18 hours)
- ▶ for all of you ...
- ▶ class on **tuesday** (11.30am-1pm) and **wednesday** (2pm-3.30pm)
→ includes lectures, exercises and *lab sessions* ...

Part 2: a deeper dive into software security ...

- ▶ 6 weeks (18 hours)
- ▶ for M2 Cyberscurity students only
- ▶ class on **thursday** (8am - 11.15am)

Examination rules

The rules of the game ...

Assignments

- ▶ M_1 : a written exam (duration=1.5h, mid-November)
- ▶ M_2 : a (short) report on lab sessions
- ▶ M_3 : an oral presentation (in November)
- ▶ M_4 : final written exam (duration=2h, end of January)

Mark computation (3 ECTS)

- ▶ for MoSiG students:

$$M = (0.3 \times M_2) + (0.7 \times M_1)$$

- ▶ for Cybersecurity students:

$$M = (0.3 \times (M_1 + M_3)/2) + (0.2 \times M_2) + (0.5 \times M_4)$$

Course user manual

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

`http://www-verimag.imag.fr/~mounier/Enseignement/Software_Security`

- ▶ 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 exercises, 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 ?? Python ??? ...
- ▶ a bit of web: web application architecture ? JavaScript ??
- ▶ some notions on compilation ? on language semantics ??

What happens behind the curtain

Some notions about:

- ▶ assembly code (ARM, x86, others ...)
- ▶ memory organization (stack, heap)
- ▶ hardware architecture

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** ?

Many possible incarnations, e.g.:

- ▶ (classical) computer: mainframe, server, desktop
- ▶ 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.
- ▶ clouds
- ▶ but also industrial networks (Scada), ... etc.
- ▶ and certainly many more !

→ 2 main interesting characteristics:

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

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

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

↪ Many (many) possible definitions ...

- ▶ a set of “high-level” **security** goals:
CIA = Confidentiality, Integrity, Availability (+ Non Repudiation + ...)
- ▶ may concern not only the **running** software itself but also its whole **execution platform** (i.e., open a shell, set up a trojan horse, etc.)
- ▶ something beyond **safety** and **fault-tolerance**:
 - ▶ notion of **intruder**, with specific capabilities
 - ▶ notion of **threats**, with a “threat model”

→ there is an “external actor” with an attack objective in mind, and able to elaborate a dedicated strategy to achieve it (\neq hazards)
- ▶ a definition “by default”:
 - ▶ functional properties = what the system should do
 - ▶ security properties =
what the system should **not do**
how it should **not behave**

Software security: an example

Let us consider 2 programs:

- ▶ `Compress`, to compress a file f
- ▶ `Uncompress`, to uncompress a (compressed) file c

Expected behavior: **the one we will try to validate !**

$$\text{Uncompress}(\text{Compress}(f)) = f \quad (1)$$

But, what about uncompressing an arbitrary (*i.e.*, *maliciously crafted*) file ?
(e.g., CVE-2010-0001 for `gzip`)

$$(\forall f. \text{Compress}(f) \neq c) \Rightarrow (\text{Uncompress}(c) = \text{"Error_Msg"}) \quad (2)$$

Actually (2) is **much more difficult to validate** than (1) ...

```
Demo: make `python -c 'print "A"*5000' `
```

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
remote execution, mobile code, plugins, reflection, etc.

Some evidences regarding software (un)-security

So many examples of successful computer system attacks:

- ▶ the “famous ones”: (at least one per year !)
Morris worm, Slammer worm, Stuxnet, Heartbleed, etc.
- ▶ the “cyber-attacks” against large organizations: (+ 400% in 10 years)
Sony, Yahoo, Paypal, e-Bay, etc.
- ▶ all the daily vulnerability alerts: [have a look at these sites !]
<http://www.us-cert.gov/ncas>
<http://www.securityfocus.com>
<http://www.securitytracker.com>

→ **Need to distinguish between:**

- ▶ **security related** piece of code uncorrectly **specified/implemented**
(crypto function, IDS, passwd manager, etc.)
- ▶ **programming bugs** in **general purpose** code (¬ security related)
↪ **clearly the most frequent situation !**

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 academic community (numerous dedicated conferences)
from 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, Darpa “Grand Challenge”, etc.

Outline

Some practical information

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

About software security

Some (not standardized) definitions . . .

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

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

Vulnerability: 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)

Exploit: a concrete program input allowing to *exploit* a vulnerability (from an attacker point of view !)

- ▶ **PoC exploit:** assumes that existing protections are disabled (i.e., they can be hijacked with other existing exploits)

Malware: a piece of code “injected” inside a computer to corrupt it
→ they usually exploit existing vulnerabilities

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)

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 causes of common weaknesses in SW security
 - ▶ at the programming language level
 - ▶ at the execution platform level

- ▶ 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 (part 1)

See

http://www-verimag.imag.fr/~mounier/Enseignement/Software_Security

Credits:

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