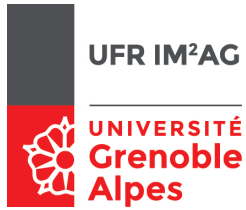

Software security, secure programming

Lecture 4: an overview of Software Security Analysis Techniques

Master on Cybersecurity – Master MoSiG

Academic Year 2017 - 2018

Software Security

The ability of a SW to *function correctly* under *malicious attacks*

Software Security

The ability of a SW to *function correctly* under *malicious attacks*

“function correctly” ?

- no crash (!), no disclosure/erasure of confidential data
- no bypass of security policy rules
- no deviation from intended behavior (arbitrary code execution)

$\rightarrow$ what the SW should **not** do ...

Software Security

The ability of a SW to *function correctly* under *malicious attacks*

“function correctly” ?

- no crash (!), no disclosure/erasure of confidential data
- no bypass of security policy rules
- no deviation from intended behavior (arbitrary code execution)

$\rightarrow$ what the SW should **not** do ...

“malicious attacks” ?

Well-crafted attack vectors, based on knowledge about:

- execution platform: libraries, OS/HW protections
- target software: code, patches
- up-to-date vulnerabilities and exploit techniques

$\rightarrow$ much beyond unexpected input/execution conditions

Software Security

The ability of a SW to *function correctly* under *malicious attacks*

“function correctly” ?

- no crash (!), no disclosure/erasure of confidential data
- no bypass of security policy rules
- no deviation from intended behavior (arbitrary code execution)

$\rightarrow$ what the SW should **not** do ...

“malicious attacks” ?

Well-crafted attack vectors, based on knowledge about:

- execution platform: libraries, OS/HW protections
- target software: code, patches
- up-to-date vulnerabilities and exploit techniques

$\rightarrow$ much beyond unexpected input/execution conditions

secure software $\neq$ robust/safe/fault-tolerant software

Root causes of insecure softwares

"A software flaw that may become a security threat . . . "

$\neq$ kinds of bugs w.r.t security:

- harmless: only leads to incorrect results or "simple" crash
- **exploitable**: can lead to unsecure behaviors . . .

Root causes of insecure softwares

“A software flaw that may become a security threat . . . ”

$\neq$ kinds of bugs w.r.t security:

- harmless: only leads to incorrect results or “simple” crash
- **exploitable**: can lead to unsecure behaviors . . .

Examples of exploitable vulnerabilities

(combinations of:)

- invalid memory accesses: buffer overflow, dangling pointers
- arithmetic overflows
- race conditions
- unsecure coding patterns (lack of input sanitization, etc.)
- etc.

Rk: influence of programming language, compilation tool, execution environment (plateform, OS, users . . .)

Vulnerability detection and analysis

A major security concern . . .

- 5000 vulns in 2011, 5200 in 2012, 6700 in 2013 ... [Symantec]
- applications and OS editors, security agencies, defense departments, IT companies, . . .

Vulnerability detection and analysis

A major security concern . . .

- 5000 vulns in 2011, 5200 in 2012, 6700 in 2013 ... [Symantec]
- applications and OS editors, security agencies, defense departments, IT companies, . . .

. . . and a business !

Some 0-day selling prices: see Zerodium web site . . .

Two distinct problems

1. detection: identify (security related) bugs
2. analysis: evaluate their dangerousness
   Are they exploitable? How difficult is it? Which consequences?

The current “industrial” practice

A 2-phase approach

1. (pseudo-random) fuzzing, fuzzing, and fuzzing . . .
   ↪ to produce a huge number of **program crashes**
2. in-depth ***manual*** crash analysis
   ↪ to identify **exploitable** bugs and obtain PoC exploits
   (ignoring protections)

The current “industrial” practice

A 2-phase approach

1. (pseudo-random) fuzzing, fuzzing, and fuzzing . . .
   ↪ to produce a huge number of **program crashes**
2. in-depth ***manual*** crash analysis
   ↪ to identify **exploitable** bugs and obtain PoC exploits
   (ignoring protections)

Drawbacks

- A time consuming activity
  (very small ratio “exploitable flaws/simple bugs” !)
  100,000 open bugs for Linux Ubuntu ; 8000 for Firefox
- Would require a better **tool assistance** . . .
  (e.g., “smart” disassembler, trace analysis, debuggers ?)

example: crash of `/bin/make` on Linux . . .

The “academic” research trends

Re-use and adapt **validation oriented** code analysis techniques

- static analysis, bounded model-checking
- test generation:
  symbolic/concolic execution, genetic algos, etc.
- dynamic (trace based) analysis

The “academic” research trends

Re-use and adapt **validation oriented** code analysis techniques

- static analysis, bounded model-checking
- test generation:
  symbolic/concolic execution, genetic algos, etc.
- dynamic (trace based) analysis

security analysis $\neq$ safety analysis !

- should be carried on the executable code
- exploit analysis $\Rightarrow$ **beyond** source-level semantics
  (understand what can happen **after** an undefined behavior)

Main issue: **scalability !** . . .

DARPA CGC: software security tool competition (1st prize: $2,000.000)

Outline

Some security-oriented code analysis techniques

- **Fuzzing**
  how to make a program crash ?
- **Dynamic Analysis**
  collect (more) useful information at runtime
- **(Dynamic) Symbolic Execution (DSE)**
  explore a (comprehensive) subset of the execution sequences
- **Static Analysis and Abstract Interpretation**
  analyse an approximation of the code behaviour without executing it

And, in addition, an overview of:

- stronger fault models (e.g., fault injection)
- some specific language/plateform issues (e.g., Java, Android, . . .)

Course organization

- lectures
- paper exercises
- lab sessions (on tools)
  static analysis, DSE, fuzzing, . . .
- oral presentations

Outline

Checking Software Security ?

Outline of the next part of the course on this topic

Oral presentations

Suggested topics (a non limitative list !)

- Programming languages and/or execution plateforms
  $\hookrightarrow$ focus on specific features, explain the strenght/weaknesses, and the associated protections ...
  - Java JVM / Android / ...
  - Golang, ...
  - JavaScript / PhP ...
  - ...
- Protections
  - Control-Flow Integrity (CFI)
  - Windows 10 protections
- Malwares
  principles, detection and identification techniques
- Code (de)-obfuscation techniques
- Vulnerability exploitation techniques
  Return-Oriented-Programming (ROP), defeating ASLR, etc.
- Side-channel attacks
- ...

Organisation

One oral presentation per “binôme” (team of 2 students)

schedule:

- before Dec. the 7th
  choose your subject (and binôme)
  $\rightarrow$ sent it to me by e-mail !

- [11th of January]
  oral presentations
  - 15 mn. presentation per binômes (with slides)
  - **a written report** (3-5 pages)