Software security specification and verification

Erik Poll

Security of Systems (SoS) group
Radboud University Nijmegen
Software (in)security specification and verification/detection

Erik Poll
Security of Systems (SoS) group
Radboud University Nijmegen
How I got interested in software security

• Tool-supported formal specification and verification of Java software
• JavaCard programs for smartcards ideal target for verification
• But... what are the security properties to verify ??
• Physical attacks on smartcards better understood than “logical” attacks on software
• Properties to verify: absence of runtime exceptions or integer overflow, preservation of invariants, ... rather than complete functional specs
Software security

• Vast majority of security problems are caused by software

• Software security excludes
  - crypto, but not implementation of crypto
  - social engineering attacks
  - hardware security, eg. tamper-resistance
Getting software secure is difficult!

Eg, from www.cert.org/advisories for (Open)SSH

**CA-2001-35 Recent Activity Against Secure Shell Daemons** *(Dec 13)* There are multiple vulnerabilities in several implementations of SSH. ...

**CA-2002-18 OpenSSH Vulnerability in challenge-response handling** *(Jun 26)* There are vulnerabilities in challenge response handling code ...

**CA-2002-23 Multiple Vulnerabilities in OpenSSH** *(July 30)* There are four remotely exploitable buffer overflows in ...

**CA-2002-24 Trojan Horse OpenSSH Distribution** *(Aug 1)* Some copies of the source code of OpenSSH package contain a Trojan horse. ...

**CA-2002-36 Multiple Vulnerabilities in SSH Implementations** *(Dec 16)* Multiple vendors' implementations of SSH contain vulnerabilities ...

**CA-2003-24: Buffer Management Vulnerability in OpenSSH** *(Sept 16)* There is a remotely exploitable buffer overflow in versions of OpenSSH prior to 3.7

Will there be more?

Note that crypto is not the solution to our problems.
Some reasons why security is difficult

• Security concerns are always secondary
  - primary goal of software is to provide some functionality or services; managing risks this introduces is a derived/secondary concern.

• Saying what is not secure is easier than saying what is secure

• Security problems can go unnoticed during normal use and testing

• Security may conflict with functionality and convenience
  - for users, but also for programmers and sysadmins
Example: programmer convenience vs security

• generally accepted Java coding standard:
  “prefer protected to private”
  - motivation: allows useful subclassing

• but Java security guideline:
  “avoid using protected”
  - motivation: protected really means unprotected
Security in software development life cycle

- Security is a concern throughout SDLC
- Ideally, catch problems as early as possible
- Still, many software vulnerabilities are introduced in the coding phase.
  - Namely coding bugs
    - eg buffer overflows
  - as opposed to architectural flaws
    - eg use of RPC under Windows
Typical software vulnerabilities

Security bugs found in Microsoft bug fix month (2002)
Example: famous Java security bug in JDK1.1

package java.lang;

public class Class {
    private Object[] signers;
    ...
    public Object[] getSigners() { return signers; }
    ...

    This bug won't be caught by typical functional specs, or detected by typical tests
The bad news

• There are many things that can go wrong in coding phase:
  - long lists of don’ts
• These may involve interaction of features, and can be hard to spot (or test)
• Programmers often not aware of them

Eg. one major creditcard company lists 214 requirements for JavaCard smartcard code, to be checked in source code reviews.
The good news

• The same things tend to go wrong
• Largely independent of application, but depending on
  - the programming language
    • eg. buffer overflows in C(++)
  - the platform/OS
    • eg. unsafe use of system calls and environment variables
  - the kind of application
    • eg. SQL command injections in webservers
The problem with long checklists of “dont’s”

• Are programmers even aware of them?
  - Educate programmers

• How do we know the list is complete?
  - Publish & discuss these lists
  - Challenge for scientific research

• How do we check them?
  - Automate this!
    Using static checkers aka source code analysers
Some (free) source code analysers

• ITS4 (C/C++)
• RATS (C/C++/Perl/PHP)
• Flawfinder (C/C++)
• FindBugs (Java)
• ..... 

Source code analysis not just for security, but for general software quality
Example: FindBugs source code analyser

Method may expose internal representation by returning reference to mutable object

Returning a reference to a mutable object value stored in one of the object's fields exposes the internal representation of the object. If instances are accessed by untrusted code, and unchecked changes to the mutable object would compromise security or other important properties, you will need to do something different. Returning a new copy of the object is better approach in many situations.
• Of course, ideally flaws should be prevented at the language level.

• Eg
  - no buffer overflows in Java or C#
  - tainted mode for input data in Perl
  - escaping meta-characters in PHP
Conclusions - the bad news

• Be aware that security tends to be ignored
• Security is hard to specify
  - long lists of don’t’s
• Software flaws are main cause of security problems
• Software flaws can be hard to uncover with testing or detect with normal use
Conclusions – some good news

• More standard patterns of security vulnerabilities are widely known
• Improving static checkers can detect such patterns (also thanks to Moore’s Law)

• Newer languages and platforms will have fewer vulnerabilities?