

Problems developing secure applications

Erik Poll

Security of Systems (SoS)
Radboud University Nijmegen

This talk

- security group in Nijmegen
- why security is difficult
- some things you can do
- our research on software security
- topics for discussion

Security of Systems (SoS) group

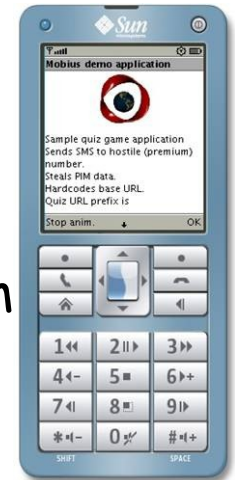


- largest Dutch research group in computer security
- Master in Computer Security together with TUE and UT in



Research topics in our group

- software security
 - for Java on smartcards (JavaCard) & phones (J2ME)
 - for C++ hypervisor (OS microkernel)
- security protocols & applied cryptography
 - for low power devices, eg. RFID
- identity-centric security
 - privacy, anonymity
 - identity management



m



Some activities

- **LaQuSo - Laboratory for Quality Software**
 - collaboration with TU Eindhoven
 - contract research to bridge gap from university to industry



- **CyCris - Centre for CyberCrime Studies**
with law faculties of Nijmegen & Tilburg



Software security

- up to the late 1990s, security was about
 - crypto
 - operating systems
- recent realisation: *it's the software!!*

Why is security such a problem?

- Security is *always* a secondary concern
 - Primary goal is **functionality**
 - Secondary goal is restricting this functionality to make things **secure**
 - in the short term, there is *no* motivation or reward whatsoever for improving security

This problem occurs at many different levels

- in programs...
- but also in programming languages, training,
...

Security as secondary concern

- in programming languages
 - Algol 60 introduces array bound checks, in 1960
 - C doesn't use this, in 1970s ...
 - ... 3 decades later, we're still trying to get rid of buffer overflows
 - *early 2000s: people start using `safestr.h`*

Security as secondary concern

- in training
 - many students learn programming in C(++)
 - nobody tells them about buffer overflows or safestring libraries
 - *a case of criminal negligence?*

The bad news: people keep making the same mistakes

- public fields in Java
 - included for efficiency reasons...
- SQL injection
 - string concatenation is a convenient way to build SQL queries...
 - will we be chasing SQL injection faults in 30 years time, just as we're still chasing buffer overflow attacks?
 - insist on use of eg. PreparedStatement?
- PHP
 - PHP is a convenient way to quickly build a website...

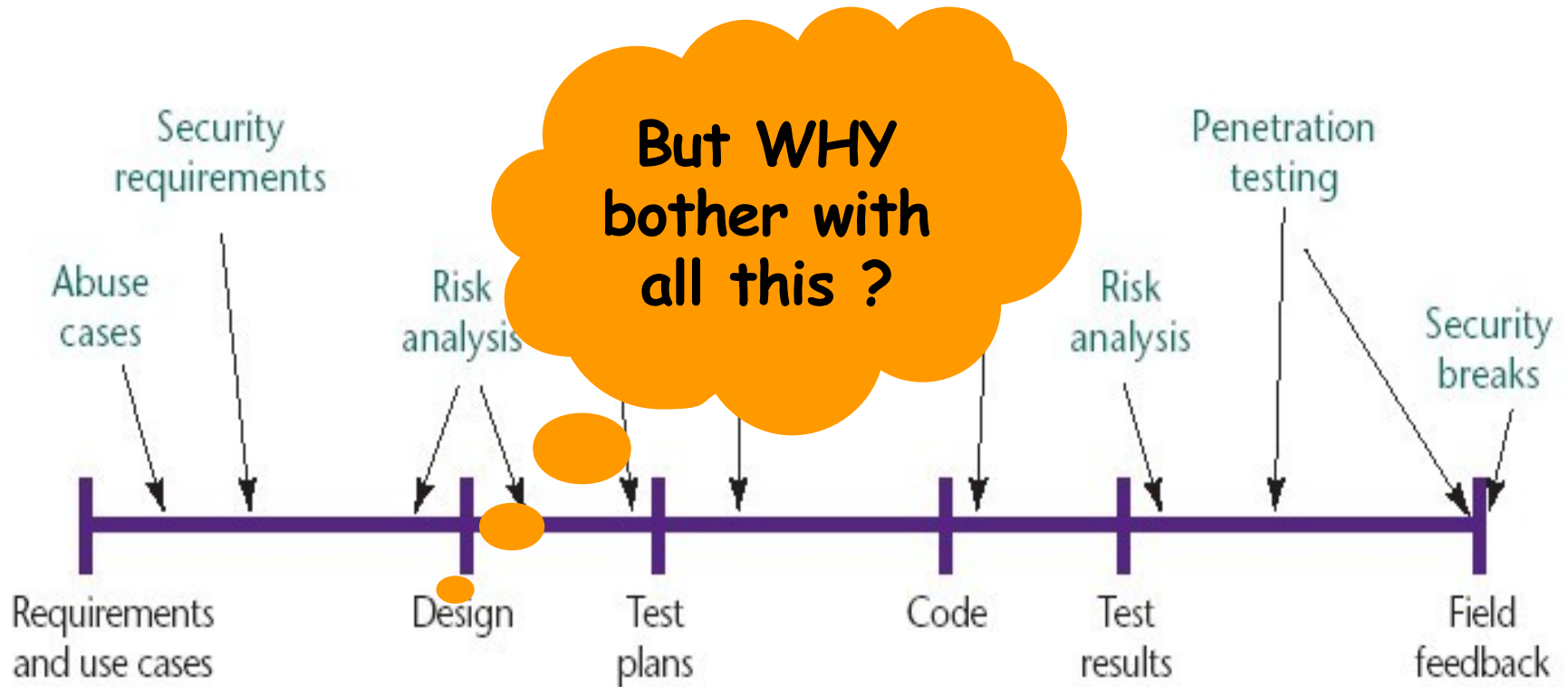
Functionality vs security : PHP

"..., I've come to the conclusion that it is basically impossible for normal programmers to write secure PHP code.

It takes far too much effort. PHP's raison d'etre is that it is simple to pick up and make it do something useful. There needs to be a major push ... to make it safe for the likely level of programmers - newbies. Newbies have zero chance of writing secure software unless their language is safe. "

[<http://www.greebo.cnet>]

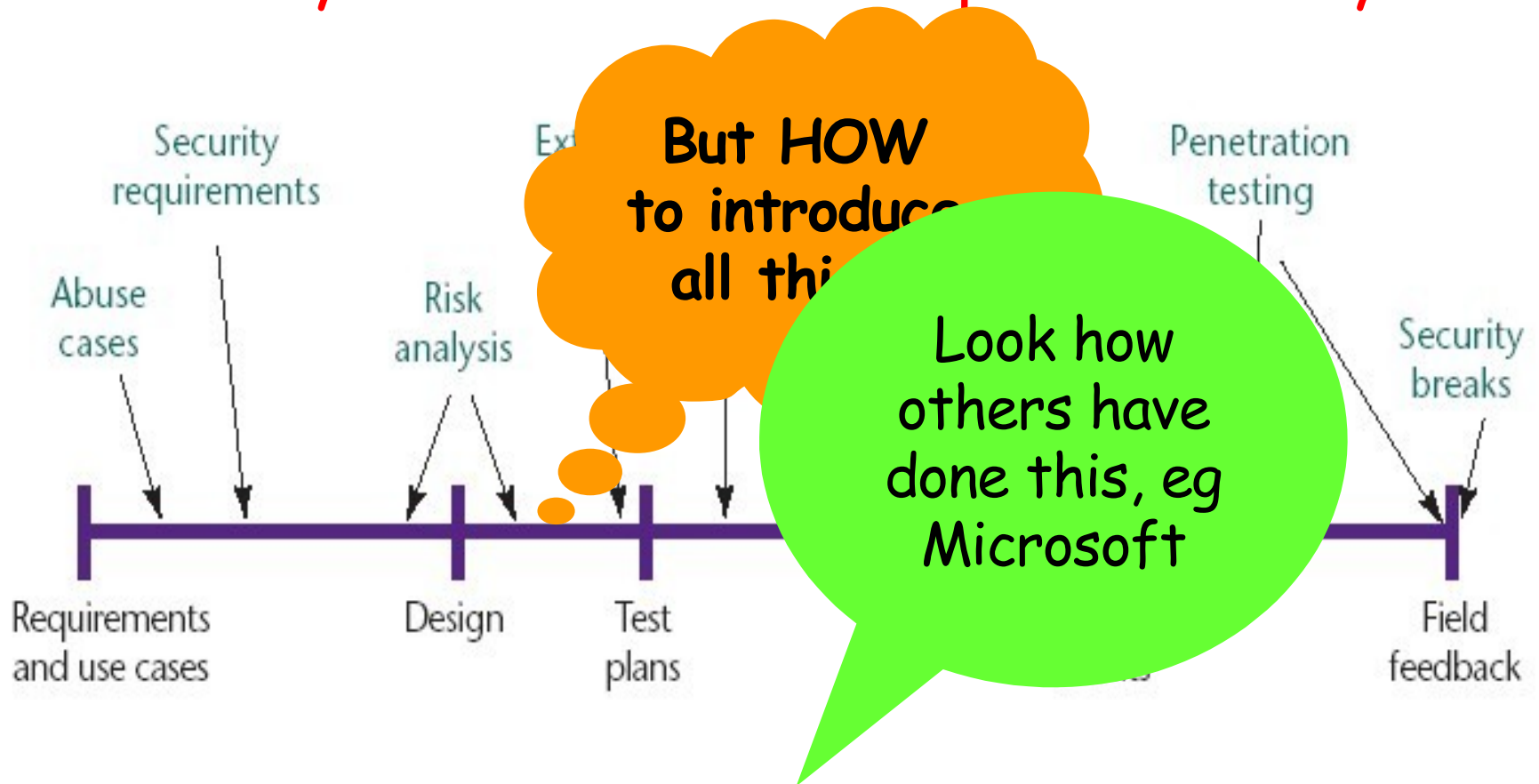
Security in Software Development Life Cycle



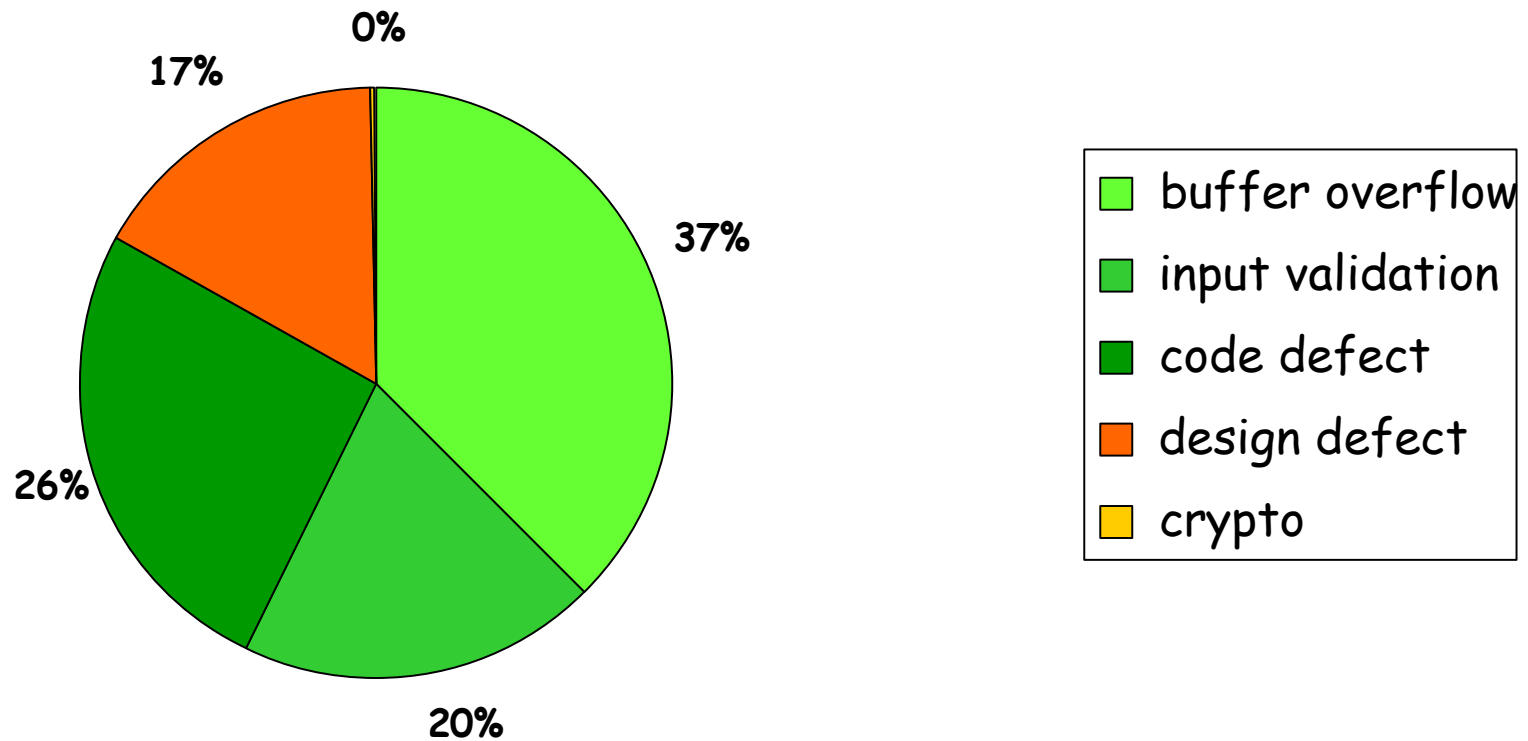
Why bother with security at all?

- The only reasons to take security seriously
 - economic
 - but many companies have been successful without being overly concerned with security
 - things seem to be slowly changing
 - eg telco's are seriously worried about the quality of code on their mobile phones
 - legal
 - Sarbanes-Oxley
 - privacy and data protection legislation
 - software liability

Security in Software Development Life Cycle



Typical software security vulnerabilities



Security bugs found in Microsoft bug fix month (2002)

Microsoft's SDL (Security Development Lifecycle)

1. Education & Awareness
2. Define & Follow Best Practices
3. Product Risk Assessment and Analysis
4. Secure Coding Policies
 - incl. checklists & source code analysis tools
5. Secure Testing Policies
6. Security Response Planning & Execution


Quality assurance at Microsoft

- Original process: manual code inspection
 - effective when team & system are small
 - too many paths/interactions to consider as system grew
- Early 1990s: add massive system & unit testing
 - Test tooks week to run
 - different platforms & configurations
 - huge number of tests
 - Inefficient detection of security holes
- Early 2000s: enter static analysis

Spot the defects


```
class Student {  
    int idnr;  
    String name;
```

*equals(Student s) does not
override equals(Object o),
but overloads it*



```
    public boolean equals(Student s) {  
        if (s==null); {  
            return false;}  
        return (idnr == s.idnr & name == s.name);  
    }
```

*empty statement is
legal Java, but probably
a mistake*



use .equals, not == for Strings



The "good" news: people keep making the same mistakes

- We can make **checklists** for common mistakes
 - We can implement **tools** that check them
 - **source code analysers** aka **static analysis tools**
 - Static analysis tools for C(++)
 - **Coverity, Fortify, jTest, PolySpace, PRefast, PRefix, ...**
 - C/C++ checkers focus on memory-related issues
- and for Java
- **CheckStyle, Findbugs, PMD, Fortify, jTest, IntelliJ, ...**

Spot the defect

```
{ ...  
  if (spec!=null) f.add(spec);  
  if(isComplete(spec)) prefs.add(spec);  
  ....}
```

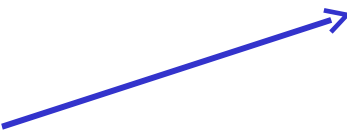
isComplete should not
get a null argument

```
boolean isComplete(Preference spec){  
  return spec.getColorKey() != null  
    && spec.getColorValue() != null  
    && spec.getTextKey() != null;  
}
```

Spot the defect

```
{ ...  
  if (spec!=null) f.add(spec);  
  if(isComplete(spec)) prefs.add(spec);  
  ....}  
  
boolean isComplete(@Nonnull Preference spec){  
  return spec.getColorKey() != null  
    && spec.getColorValue() != null  
    && spec.getTextKey() != null;  
}
```

annotation expresses intent and makes analysis - by human or tool - easier



Java metadata tags

- introduced in Java 1.5 (JSR 175)
- JSR 305 "Annotations for Software Defect Detection" currently in progress
 - @NonNull, @Nullable
 - @Tainted, @Untainted to find input validation problems
 - @NonNegative
 - @WillClose, @WillNotClose
 - @CheckReturnValue
- allows enhanced static analysis
 - tainting analysis (data flow analysis)
 - intra-procedural analysis

beyond Java tags: JML

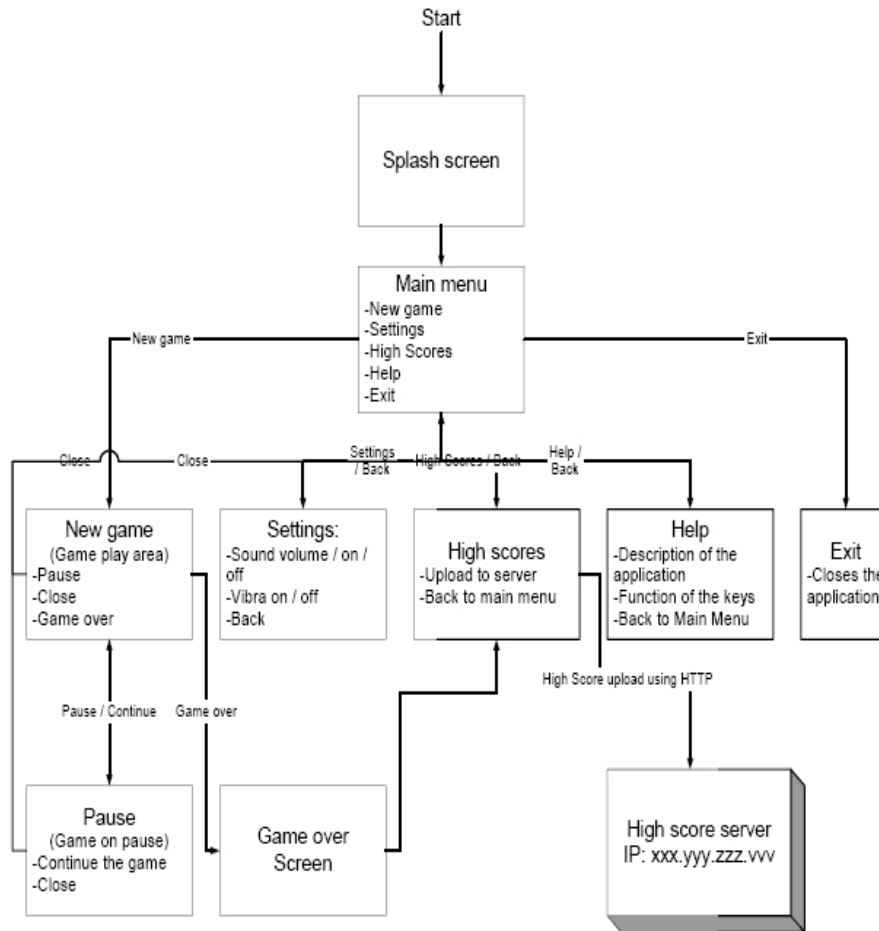
```
public class ChipKnip{
  private int balance;
  //@ invariant 0 <= balance && balance < 500;

  //@ requires amount >= 0;
  //@ ensures balance <=\old(balance);
  //@ signals (BankException) balance ==
    \old(balance);
  public debit(int amount) {
    if (amount > balance) throw(new
      BankException("No way"));
    balance = balance - amount;
  }
```

JML

- Specification language for Java
 - Properties can be specified in Design-By-Contract style, using pre/postconditions and invariants
- Various tools to check JML specifications by eg
 - runtime checking
 - program verification/static analysis
 - at compile time
- Related work
 - Spec# for C# by Microsoft
 - SparkAda for Ada
 - EauClaire for C by Brian Chess

Ongoing work on using JML



Can we specify & statically check for a J2ME mobile phone application

- display sticks to this navigation graph
- only opens a limited number of sms:// connections
- only connects to approved numbers?

Other annotation languages

- PRefast static analysis tool (in Visual Studio 2005) uses SAL (Standard Annotation Language) to mark up C/C++ code.

Eg

```
__checkReturn void *malloc(__in size_t s);
```

__checkReturn means that caller *must* check the return value of malloc

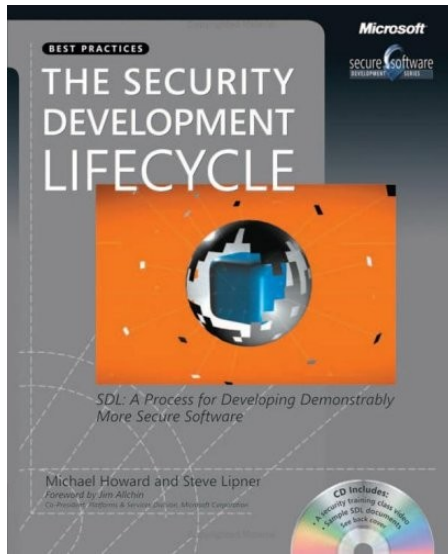
Conclusions

- Security is secondary concern
 - in programming, language design, training, ...
 - left to itself, it *will* be ignored
- People keep making the same mistakes,
 - which can be good news!
 - education can prevent people making these mistakes
 - checklists can catch common mistakes
 - some in automatic checks by tools
 - more with annotations in code

To improve software security

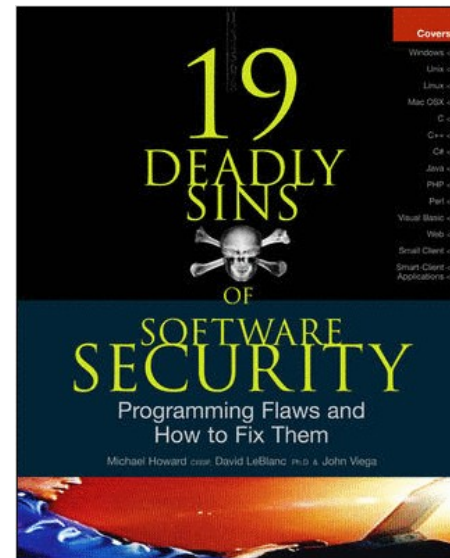
- commitment (why)
 - economic, legal
 - at management level
- knowledge (what)
 - common problems
 - best practices
- implementing it (how)
 - awareness
 - checklists, tools, SDL
 - spot-the-defect competitions

Interesting reads



The Security Development Lifecycle

by Michael Howard and Steve Lipner



19 deadly sins of software security

by Michael Howard, David LeBlanc, and John Viega