CS4004/CS4504: FORMAL VERIFICATION

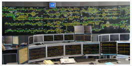
Lecture 1: Module Overview & Introduction

Vasileios Koutavas

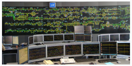
School of Computer Science and Statistics
Trinity College Dublin

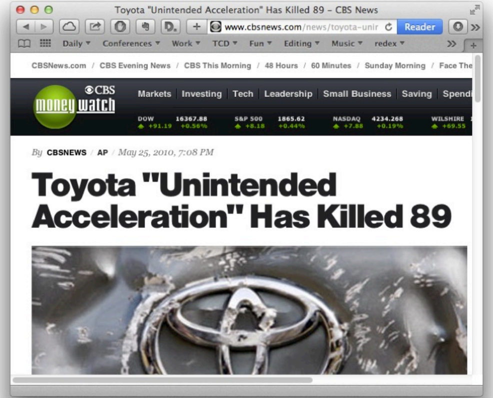
THE PERILS OF SOFTWARE

- → Software is now controlling critical machines:
  - → Transportation: cars ($>$ 100*M* LoC [IEEE]), airplains, trains, spacecraft, ...
  - → Medical: pacemakers, MRI machines, ...
  - → Utilities: power grids, telephone centres, ...
  - → Finance: online banking, stock prices, ...
  - → ...

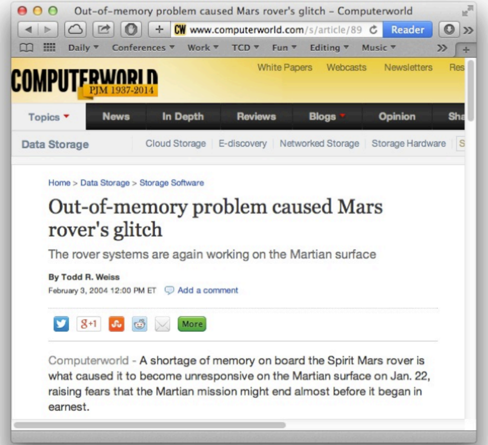


- Software is now controlling critical machines:
  - Transportation: cars ($>$ 100M LoC [IEEE]), airplains, trains, spacecraft, ...
  - Medical: pacemakers, MRI machines, ...
  - Utilities: power grids, telephone centres, ...
  - Finance: online banking, stock prices, ...
  - ...
- BUT Software is very unreliable
  - http://en.wikipedia.org/wiki/List_of_software_bugs
  - http://www.cs.tau.ac.il/~nachumd/horror.html

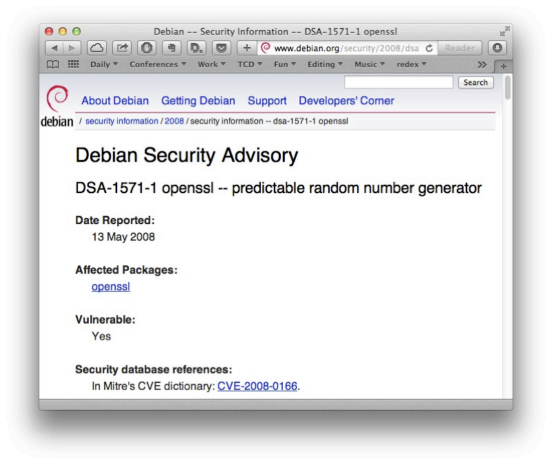


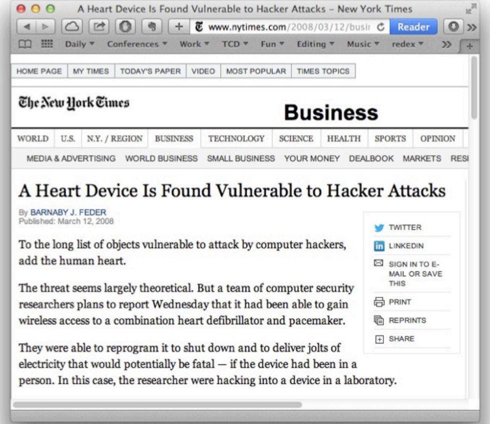

Toyota "Unintended Acceleration" Has Killed 89 – CBS News
www.cbsnews.com/news/toyota-unir
Reader
Daily
Conferences
Work
TCD
Fun
Editing
Music
redex
CBSNews.com / CBS Evening News / CBS This Morning / 48 Hours / 60 Minutes / Sunday Morning / Face The
CBS moneywatch
Markets | Investing | Tech | Leadership | Small Business | Saving | Spendi
DOW 16367.88 +91.19 +0.56%
S&P 500 1865.62 +8.18 +0.44%
NASDAQ 4234.268 +7.88 +0.19%
WILSHIRE +69.55
By CBSNEWS / AP / May 25, 2010, 7:08 PM
Toyota "Unintended Acceleration" Has Killed 89

THE PERILS OF SOFTWARE



THE PERILS OF SOFTWARE




A Heart Device Is Found Vulnerable to Hacker Attacks – New York Times

www.nytimes.com/2008/03/12/busi

Reader

Daily Conferences Work TCD Fun Editing Music redex

HOME PAGE MY TIMES TODAY'S PAPER VIDEO MOST POPULAR TIMES TOPICS

The New York Times

Business

WORLD U.S. N.Y. / REGION BUSINESS TECHNOLOGY SCIENCE HEALTH SPORTS OPINION

MEDIA & ADVERTISING WORLD BUSINESS SMALL BUSINESS YOUR MONEY DEALBOOK MARKETS RESI

A Heart Device Is Found Vulnerable to Hacker Attacks

By BARNABY J. FEDER
Published: March 12, 2008

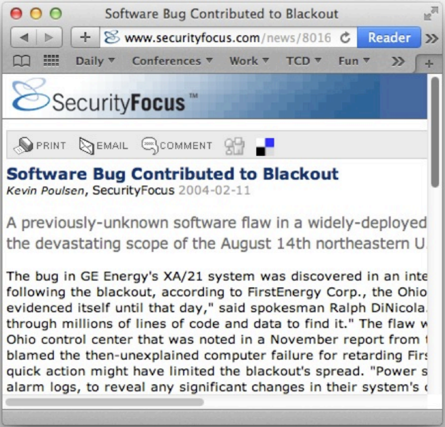
TWITTER
LINKEDIN
SIGN IN TO E-MAIL OR SAVE THIS
PRINT
REPRINTS
SHARE

To the long list of objects vulnerable to attack by computer hackers, add the human heart.

The threat seems largely theoretical. But a team of computer security researchers plans to report Wednesday that it had been able to gain wireless access to a combination heart defibrillator and pacemaker.

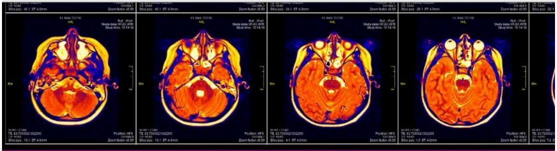
They were able to reprogram it to shut down and to deliver jolts of electricity that would potentially be fatal — if the device had been in a person. In this case, the researcher were hacking into a device in a laboratory.

THE PERILS OF SOFTWARE



THE PERILS OF SOFTWARE

www.sciencealert.com/a-bug-in-fmri-software-could-invalidate-decades-of-brain-research-scientists-...



A bug in fMRI software could invalidate 15 years of brain research

This is huge.

BEC CREW 6 JUL 2016

There could be a very serious problem with the past 15 years of research into human brain activity, with a new study suggesting that a bug in fMRI software could invalidate the results of some 40,000 papers.

That's massive, because functional magnetic resonance imaging (fMRI) is one of the best tools we have to measure brain activity, and if it's flawed, it means all those conclusions about what our brains look like during things like exercise, gaming, love, and drug addiction are wrong.

TIMSORT

- → Hybrid Sorting Algorithm of MergeSort + InsertionSort
- → Used in Android JDK, Sun JDK, OpenJDK, Python, GNU Octave

TIMSORT

- → Hybrid Sorting Algorithm of MergeSort + InsertionSort
- → Used in Android JDK, Sun JDK, OpenJDK, Python, GNU Octave

- → The implementation uses an intermediate array of fixed size
- → The (wrong) assumption is that the array will never fill up
- → But it **does** fill up for carefully selected inputs arrays of size $>$ 562 **trillion**

http://www.envisage-project.eu/

proving-android-java-and-python-sorting-algorithm-is-broken-and-how-to-fix-it/

OpenJDK's java.utils.Collection.sort() is broken: The good, the bad and the worst case*

Stijn de Gouw[1,2], Jurriaan Rot[3,1], Frank S. de Boer[1,3], Richard Bubel[4], and Reiner Hähnle[4]

[1] CWI, Amsterdam, The Netherlands
[2] SDL, Amsterdam, The Netherlands
[3] Leiden University, The Netherlands
[4] Technische Universität Darmstadt, Germany

Abstract. We investigate the correctness of TimSort, which is the main sorting algorithm provided by the Java standard library. The goal is functional verification with mechanical proofs. During our verification attempt we discovered a bug which causes the implementation to crash. We characterize the conditions under which the bug occurs, and from this we derive a bug-free version that does not compromise the performance. We formally specify the new version and mechanically verify the absence of this bug with KeY, a state-of-the-art verification tool for Java.

FORMAL VERIFICATION

→ Ultimate goal: verify the absence of software errors

- Ultimate goal: verify the absence of software errors
- Easy? NO!
  - Finite code often has an infinite set of behaviours:

    ```
    quicksort (a: array int)
    ```

    Infinite number of `a`-inputs and outputs

- Ultimate goal: verify the absence of software errors
- Easy? NO!
  - Finite code often has an infinite set of behaviours:

    ```
    quicksort (a: array int)
    ```

    Infinite number of `a`-inputs and outputs
  - What does it even mean for `quicksort` to be correct?

- Ultimate goal: verify the absence of software errors
- Easy? NO!
  - Finite code often has an infinite set of behaviours:

    ```
    quicksort (a: array int)
    ```

    Infinite number of `a`-inputs and outputs
  - What does it even mean for `quicksort` to be correct?
- A number of ways to approach the problem.

→ Can we use **testing** to prove correctness/incorrectness?

→ Can we use **testing** to prove correctness/incorrectness?

NO! Need an infinite testsuit

→ Algorithmic Verification: Can we come up with an **algorithm** to do prove correctness/incorrectness automatically?
  Create a program `av` that inputs another program `p` and after finite time outputs `false` if `p` has a bug for some input, or `true` otherwise.

- Algorithmic Verification: Can we come up with an **algorithm** to do prove correctness/incorrectness automatically?
  Create a program `av` that inputs another program `p` and after finite time outputs `false` if `p` has a bug for some input, or `true` otherwise.

  NO! Because of the halting problem [Alan Turing 1936]: if `av` exists then there is a paradox; thus `av` can't exist.

Proof.
Let "bug"=infinite loop=`infloop()`. If `p` is a program then let `'p(p)` be the program that when run, it will execute `p` on input `p`.
Assume `av` exists. Create:

```
paradox(p:Prog) = if av('p(p)) then infloop() else true
```

What does `paradox(paradox)` do?

1. If `paradox(paradox)` loops forever then `av('paradox(paradox))=false`, thus `paradox(paradox)` returns `true`. Contradiction
2. If `paradox(paradox)` returns `true` `av('paradox(paradox))=true`, thus `paradox(paradox)` loops forever. Contradiction

□

- Deductive Verification: Can we have a **mathematical proof system** to prove correctness/incorrectness for all programs?
  Create a system **L** of logical axioms and rules, such that for any program **p** we can **prove** either
  - **p** has a bug for some input
  - **p** has no bug for any input

  If such a system **L** exists then we can create a fully automatic verification algorithm (simply systematically explore all logical derivations and eventually, in finite time, derive "**p** has a bug" or "**p** has no bug".)

- Deductive Verification: Can we have a **mathematical proof system** to prove correctness/incorrectness for all programs?
  Create a system **L** of logical axioms and rules, such that for any program **p** we can **prove** either
  - **p** has a bug for some input
  - **p** has no bug for any input

  If such a system **L** exists then we can create a fully automatic verification algorithm (simply systematically explore all logical derivations and eventually, in finite time, derive "**p** has a bug" or "**p** has no bug".)

  NO! Kurt Gödel proved in 1931 that no such logical system exists.

- → **Soundness**: If the verification method reports no failure, then the program under examination has no bug
- → **Completeness**: If the verification method reports a failure, then the program under examination has a bug
- → **Termination**: The verification method will terminate, giving back an answer.

- **Soundness**: If the verification method reports no failure, then the program under examination has no bug
- **Completeness**: If the verification method reports a failure, then the program under examination has a bug
- **Termination**: The verification method will terminate, giving back an answer.

Pick TWO. Having all three is theoretically impossible.

Usually verification systems pick **soundness** and **termination**.

However not all is lost! Sound and terminating systems can prove the correctness of *virtually every program we would care about.*

The scientific community continuously pushes the limits of these systems to extreme levels!

- → **Algorithmic verification**: model checking, abstract interpretation, static analysis.
  - → create a model of the program in a **decidable framework** (finite state system, pushdown system)
  - → usually: semi-automatic (semi-manual) model creation
  - → automated model verification
- → **Deductive verification**:
  - → create a correctness proof of the program in a **logic** (with axioms and logical rules)
  - → usually: semi-automatic (semi-manual) proof construction
  - → automated proof checking

However not all is lost! Sound and terminating systems can prove the correctness of *virtually every program we would care about*.

The scientific community continuously pushes the limits of these systems to extreme levels!

- **Algorithmic verification**: model checking, abstract interpretation, static analysis.
  - create a model of the program in a **decidable framework** (finite state system, pushdown system)
  - usually: semi-automatic (semi-manual) model creation
  - automated model verification

- **Deductive verification**:
  - create a correctness proof of the program in a **logic** (with axioms and logical rules)
  - usually: semi-automatic (semi-manual) proof construction
  - automated proof checking

In this module: we will use the second approach.

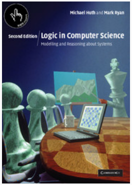
- We will manually prove programs correct using pen and paper proofs.
- We will use software that provide *some* automation to make this easier.

Success stories start appearing from the mid-1990s.

- → Paris metro line 14, (1998,refinement approach – combination of algorithmic and deductive approach)
  http://www.methode-b.com/documentation_b/ClearSy-Industrial_Use_of_B.pdf
- → Flight control software of A380: verified absence of run-time errors (2005, abstract interpretation) http://www.astree.ens.fr/
- → L4.verifiedmicro-kernel: a formally correct operating system kernel (2010, deductive verification) http://www.ertos.nicta.com.au/research/l4.verified/
- → SLAM: verifier for MS Windows drivers (2010, model checking)
  http://research.microsoft.com/en-us/projects/slam/
- → Facebook's Infer verifier: detects bugs in Android and iOS apps http://fbinfer.com/
- → And more...

THIS MODULE

Book: Logic in Computer Science: Modelling and Reasoning about Systems, 2nd Edition, by M. Huth & M. Ryan



1. Symbolic logic
   - → Natural deduction
   - → Propositional logic
   - → First-order Predicate logic
2. Correctness of imperative programs
   - → Floyd-Hoare logic
   - → Weakest Precondition calculus
   - → Loop invariants
   - → functional abstractions, etc.
3. (possibly) Correctness of functional programs
   - → Data types (numbers, lists, trees, …)
   - → Recursive functions
   - → Inductive proofs (mathematical, structural, …)
4. (possibly) Correctness of concurrent programs
   - → Hennessy-Milner logic
   - → Bisimulation
   - → etc.

Exercises, Tutorials and Assignments:

1. Pencil&paper proofs: propositional, inductive, correctness proofs.
2. Semi-automatic proofs using the **Dafny** verifier
   `http://research.microsoft.com/en-us/projects/dafny/` `http://rise4fun.com/Dafny`

Marks:

- → 2% marks from attendance (random sampling)
- → 33% marks from coursework
- → 65% marks from annual exam

Final exam: pencil&paper proofs of simple propositional properties, and program correctness properties.

Exam in January.

You will need: a windows box to install visual studio + dafny
(or a linux box to install Why3)

We meet 3 times a week for lectures and tutorials (Tue 5pm, Thu 5pm, Fri 2pm)

More information here:
`www.scss.tcd.ie/Vasileios.Koutavas/teaching/cs4004-4504`