Formal Methods and Cybersecurity Education

James Davenport & Tom Crick
J.H.Davenport@bath.ac.uk & tcrick@bcs.org.uk

University of Bath & University of Swansea
Institute of Coding: https://instituteofcoding.org/

2 December 2019

I am a happy CS Professor

I am	a happy CS Professor: I trust my life to my former students several times a week.
Surely	a rhetorical flourish.
No: real	There's a software house (Altran–Praxis, originally a University of Bath spinout) that writes both railway signalling software and air traffic control software, employing several former students.
So	I put my life in their hands to fly here.
Heavily	into formal methods, Ada (subsets: SPARK) etc.

Now the use of Formal Methods in the safety-critical industry is not new, and barely news. But it's not widely known. I quoted the Ligne 14 performance figures (software shipped in 1999 and no bugs reported [JBDD11]) to a major figure in the commercial software industry, to be told that I was lying, as this was utterly impossible.

Formal Methods in the UK

- We only know of one university that teaches Formal Methods below final year (and even there it's become optional).
- Many (including our own) do not teach it at all

! Some of the necessary logic is probably taught, and statements like "this is useful if you . . ." are made.

- Therefore firms like Altran–Praxis have to teach it from scratch
- Therefore "there's no industry demand for it".
- And because they're not in the news, there's little student demand for formal methods.

Other uses

Intel has employed formal methods ever since the 1994 "Pentium divide" bug [Cip95]. But again they don't really advertise this.

Facebook has recently stated a pretty substantial development into (weak) Formal Methods [DFLO19].

Amazon has started using Formal Methods for reasoning about security properties [Vog19]

AWS weaknesses (quite possibly the customers' fault) have been at the root of many problems. [McA18] claims "5.5% of all AWS S3 buckets in use are misconfigured to be publicly readable" and "27% of organizations using PaaS have experienced data theft from their cloud infrastructure".

In terms of impact, there's the recent Capital One breach [Nee19], but also [Nor18, Whi19]

Google has gone public on its use of static analysis tools [SAE$^+$18].

CyberSecurity is very much in the news

50% of security breaches are caused by coding errors [McG06]

PCI DSS [Pay18], essentially the only world-wide mandatory security standard, has these two requirements.

6.5 Address common coding vulnerabilities in software-development processes as follows:
- Train developers at least annually in up-to-date secure coding techniques, including how to avoid common coding vulnerabilities;
- Develop apps based on secure coding guidelines.

6.6 For public-facing web applications, address new threats and vulnerabilities by either of:
- Reviewing public-facing web apps via manual or automated app vulnerability security assessment tools or methods, at least annually and after any changes;
- Installing an automated technical solution that detects and prevents web-based attacks (e.g. a web app firewall) in front of public-facing web apps.

PCI DSS

Therefore simultaneously mandates secure coding and doesn't trust it.
This is not too surprising: recent studies [NDT$^+$17, NDTS18] have shown that people capable of secure coding don't do it unless explicitly required.
I have also heard stories of employees at a major credit card processor violating [Pay18] "because the customer wants it".
Furthermore, in practice most people who do [Pay18] "properly" go the "app firewall" route.
This has the usual problem of only catching the things it is meant to catch, and in fact can't catch many things, e.g. the British Airways breach [Bar18].

CyberSecurity Errors

Many of these, but common ones include:

Buffer Overflow Very common in C/C++ — in theory Java catches this (so you get denial-of-service rather than information leakage);

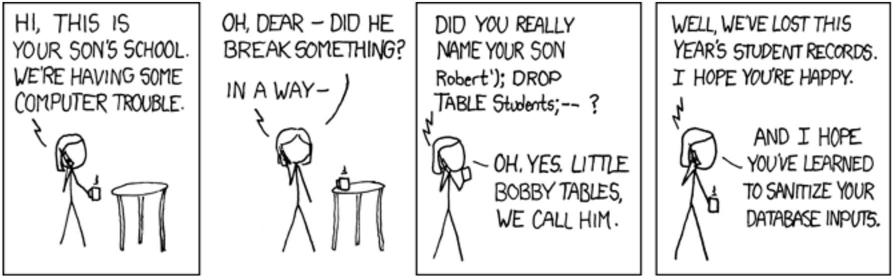
Heartbleed, which made the BBC news [BBC14] was such, and probably wouldn't have been detected in Java.

Use-After-Free Java *does* eliminate this;

Errors Often edge cases, which Java generally won't detect, or will give a run-time error;

Injection attacks of all sorts: what should be text is actually interpreted as commands.

SQL Injection [XKC07]


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

- `Robert');DROP TABLE Students;--`
- Sanitising inputs is no longer considered best practice here: use parametrised queries instead
- But in other areas, such as Cross-Site Scripting (XSS) it's necessary: Google are extending Javascript types (in Chrome) to enable one to prove sanitisation has occurred [Bra19].

It doesn't necessarily say anything about the quality of the sanitisation, which is a notoriously hard problem.

Selection Sort

– Look at all N books, select the shortest book

– Swap this with the first book

– Look at the remaining N-1 books, and select the shortest

– Swap this book with the second book

– Look at the remaining N-2 books, and select the shortest

– Swap this book with the third book and so on…

So, is our sort efficient?
If we have N books, how many steps does it take to sort them?
Let’s assume a step is any time we either swap or compare at a book.

1

Selection Sort Algorithm

To sort the array a[0],...,a[n-1]

```
// Precondition: a is an array of n objects
m:=0
while m<n-1 {
    k:=m
    l:=m+1
    while l < n { //find the least element in a[m]...a[n-1
        if a[l]<a[k]
           then k:=l;
        l:=l+1;
    }
    if m ~= k
        then swap(a,m,k);
    m:=m+1;
}
// Postcondition: a is sorted
```

Selection Sort Proof

Apparently we should prove

$$\{\texttt{a}\text{ is an }\texttt{n}\text{-array}\}\texttt{selection sort}\{\texttt{a}\text{ is sorted }\texttt{n}\text{ array}\}$$

However an easy way to do this is to output $0, 1, \ldots, n-1$: we also need "`a` has the same elements as before". Fortunately, this is a consequence of the fact that `a` is only changed by `swap` (and some lemmas about swap).

Selection Sort Proof: Inner loop

```
k:=m
l:=m+1
while l < n { //find the least element in a[m]...a[n-1
    if a[l]<a[k]
       then k:=l;
    l:=l+1;
}
```

The comment helps: a plausible loop invariant is

a[k] is the least in a[m],...a[l-1].

We actually need $0 \leq k < n$ and $0 \leq l$ to ensure array accesses are valid.

Hence after the loop,

$l \geq n \wedge$ a[k] is the least in a[m],...a[l-1].

In fact $l = n$ (a tedious `while` lemma). Hence after this block, a[k] is the least in a[m],...a[n-1].

Selection Sort: Outer Loop

```
m:=0
while m<n-1 {

{??}block{a[k] is the least in a[m],...a[n-1]} ⇐ Floyd–Hoare
                                                  abstraction
    if m ~= k
        then swap(a,m,k);
    m:=m+1;
}
// Postcondition: a is sorted
```

We might think "a[0]...a[m-1] is sorted" is the loop invariant.
This doesn't work: need to add "$\wedge$ all a[0]...a[m-1] $\leq$ all a[m]...a[n-1]".
With this addition, and lemmas about swap and while, we have a proof of correctness.

Much Ado About Nothing?

In fact, the proofs of insertion sort (loop invariant is just "`a[0]...a[m-1]` is sorted") and mergesort (`mergesort` itself is trivial, `merge` similar to insertion sort) are slightly easier.

JHD's hybrid sort is trivial once we have the above.

But Timsort [Pet93] is another combination of insert and merge sorts, used in Python, Java etc.

Formal verification [dGRdB$^+$15] found a bug in the standard implementation: the smallest example has size $2^{26} \approx$ 67M. It is far from clear how one would debug this even if it occurred.

Conclusion

1. Can we motivate Formal Methods by looking at CyberSecurity, since it's newsworthy?
2. Can/will the Google/Facebook/Amazon of this world place more insistence on Formal Methods?

   "Move fast and break things" is no longer Facebook's motto!

   Now "Move fast with stable infrastructure" [Sta14]
3. Should Algorithms teachers (like me) place more insistence on Floyd–Hoare like statements when developing algorithms, and place more insistence on proving correctness?

B. Barth.
No fly-by-night operation: Researchers suspect Magecart group behind British Airways breach.
https://www.scmagazine.com/home/security-news/no-fly-by-night-operation-researchers-suspect-magecart-
2018.

BBC.
US government warns of Heartbleed bug danger.
https://www.bbc.co.uk/news/technology-26985818, 2014.

D. Bradbury.
Google Chrome is ditching its XSS detection tool.
https://nakedsecurity.sophos.com/2019/07/18/google-chrome-is-ditching-its-xss-detection-tool/, 2019.

B.A. Cipra.
How number theory got the best of the Pentium chip.
Science, 267(5195):175, 1995.
doi:10.1126/science.267.5195.175.

D. Distefano, M. Fähndrich, F. Logozzo, and P.W. O'Hearn.
Scaling static analyses at Facebook.
Communications of the ACM, 62:62–70, 2019.

S. de Gouw, J. Rot, F.S. de Boer, R. Bubel, and R. Hähnle.
OpenJDK's Java.utils.Collection.sort() Is Broken: The Good, the Bad and the Worst Case.
In *Computer Aided Verification 2015*, volume 9206 of *Lecture Notes in Computer Science*, pages 273–289. Springer, 2015.
doi:10.1007/978-3-319-21690-4_16.

M. Jacquel, K. Berkani, D. Delahaye, and C. Dubois.
Verifying B Proof Rules using Deep Embedding and Automated Theorem Proving.
International Conference on Software Engineering and Formal Methods, pages 253–268, 2011.

McAfee Labs.
Cloud Adoption and Risk Report 2019.
`https://info.skyhighnetworks.com/WPCloudAdoptionRiskReport2019_BannerCloud-MFE.html`, 2018.

G. McGraw.
Software Security — Building Security In.
Addison-Wesley, 2006.

A. Naiakshina, A. Danilova, C. Tiefenau, M. Herzog, S. Dechand, and M. Smith.
Why Do Developers Get Password Storage Wrong?: A Qualitative Usability Study.
In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17). ACM, pages 311–328, 2017.

A. Naiakshina, A. Danilova, C. Tiefenau, and M. Smith.
Deception Task Design in Developer Password Studies: Exploring a Student Sample.
In Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018). USENIX Association, pages 297–313, 2018.

S.M. Neenan.
What AWS users can learn from the Capital One breach.
https://searchaws.techtarget.com/feature/What-AWS-users-can-learn-from-the-Capital-One-breach, 2019.

A. Nordrum.
Millions of U.S. Voter Records Exposed on Robocall Company RoboCent's Poorly Configured AWS Cloud Storage.
https://spectrum.ieee.org/tech-talk/telecom/security/millions-of-us-voter-records-exposed-by-political-robo
2018.

Payment Card Industry Security Standards Council (PCI SSC).
Requirements and Security Assessment Procedures Version 3.2.1.
https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2-1.pdf, 2018.

T. Peters.
Timsort description.
http://svn.python.org/projects/python/trunk/Objects/listsort.txt, 1993.

C. Sadowski, E. Aftandilian, A. Eagle, L. Miller-Cushion, and C. Jaspan.
Lessons from building static analysis tools at Google.
Commun. ACM, 61(4):58–66, 2018.

N. Statt.
Zuckerberg: 'Move fast and break things' isn't how Facebook operates anymore.
`https://www.cnet.com/news/`
`zuckerberg-move-fast-and-break-things-isnt-how-we-opera`
2014.

W. Vogels.
Proving security at scale with automated reasoning.
https://www.allthingsdistributed.com/2019/05/proving-security-at-scale-with-automated-reasoning.html, 2019.

Z. Whittaker.
Massive mortgage and loan data leak gets worse as original documents also exposed.
https://techcrunch.com/2019/01/24/mortgage-loan-leak-gets-worse/, 2019.

XKCD.
Cartoon 327.
https://xkcd.com/327/, 2007.