TRUST IN SOFT

cyber-security provider

Industrial Formal Methods To Trust Legacy Software

Benjamin MONATE, CTO

A few words about TrustInSoft

- TrustInSoft is a start-up founded in 05/2013
- It provides formal method-based solutions to help software publishers and integrators develop secure systems
- Its solutions are built on top of Frama-C: static analysis and formal methods collaboration platform for the industry
- Spin-off from CEA where Frama-C was born in 2005 and immediately open-sourced

What this talk is not about

- Hardware verification
- Software Engineering/Model verification
- Cryptography
- New research in Code Analysis

Cyber-security

- Buzzword for software security
- Attacks against software: costly and frequent

PDF reader bug in iOS, 4.3.3

PlayStation Network

Attack on the PlayStation Network April 17 and April 19, 2011

2011 cyber attack on CitiBank
200 000 cards had to be re-created

Zero-days reseller: a flourishing activity

- Many legal resellers
- Hackers all over the world
- Government agencies are buying this

TrustInSoft is not part of this market

Black Market Zero-Days prices:

Adobe READER	$5 000 - $30 000
MacOS X	$20 000 - $50 000
Android	$30 000 - $60 000
Microsoft Word	$40 000 - $100 00
Windows	$60 000 - $120 000
Firefox or Safari	$60 000 - $150 000
Chrome or IE	$80 000 - $200 000
Apple IOs	$100 000 - $250 000

Source : Shopping For Zero-Days: A Price List For Hackers' Secret Softw are Exploits

By Andy GREENBERG -- Forbes 03/23/2012

Cyber-security history

20-year long race

Mafia, States

Threats

Virus

Worms

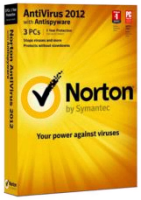
Botnet

Advanced Persistent Threats

1990 2000 2010 2020

Solutions

Anti-Virus



Intrusion Detection Systems

Data leakage Prevention solutions

Incrementally fix software

Formal methods may end this race

- Definitely (formally) fix the software
- Attackers need to find out model's flaws

  → much more difficult

- Two ways:
  - Rewrite the software: coding formally
  - Formalize existing software: proving code

Software for regulated industrial domains?

Regulated domains: **software must be compliant** with “Standards” before being put on the market. Mostly safety standards that prevent some security issues

- Medical devices: IEC 62366
- Aeronautics: ED-12, DO178-B/C "Software Considerations in Airborne Systems and Equipment Certification"
- Rail: EN 50128
- Automotive: ISO 26262
- Energy: IEC 61508
- Banking and finance : PCI, Common Criteria
- …

FMs Pros:

- Productivity
- Cost savings
- Quality
- HR management: aka « fun factor »

What about unregulated sensitive systems?

Coding Formally for unregulated domains

- Two examples:
  - SeL4: L4 microkernel levels:
    - Full Functional correctness

  - Quark: Web browser with formal sandboxes
    - Tab non interference
    - Cookie confidentiality and integrity
    - Address bar Integrity and correctness

Coding Formally for unregulated domains

- Tools = modeling and programming language + computer aided proofs systems (Isabelle/HOL, Coq, …)
- Pros:
  - very precise formal properties
  - total correctness
  - clean trust-base
- Cons:
  - need to develop software and formal model
  - difficult HR issues to address

Coding Formally for unregulated domains

→ Necessarily more expensive process: price for trust

- What if the software is not useful?
  - Not an issue for a research project (beside of researcher's ego issues)
  - May kill a company: even without FM software companies die because of a « nice to have » instead of a « must have » software

→FM offers more trust, but you pay for this trust: hence,

cost of failure is higher

- N/A for new software targeted at regulated domains (kernel, compiler, embedded stack)
- Not so nice for the others : even for security stacks (SSL stacks, cryptography libraries) features and agility come first

Proving code

- Proving existing code:
  - Tools = static analyzers based on
    - Abstract Interpretation
    - Weakest Preconditions Calculus
    - Model checking
  - Pros:
    - apply on existing software
    - formalize only what you can afford
  - Cons: need to match formal models with uncontrolled source code.

→Valid for weakly regulated domains = most of the running software

→For regulated domains quality/cost ratio seems comparable to Coding Formally

Closed-source software

- Only insiders may formally prove/fix software
  - Ok for regulated industries: still very expensive
- Everyone may attack: black box attack technics are efficient
  - Strong dissymmetry

- Trust is based on:
  - Norms
  - Publisher reputation
  - Black box testing: Functional, Penetration Tests
  - Insurance contracts

Open Source software

- Pervasive even in some regulated domains
- Existing software is good to some users (or else it dies)
- Trust comes from:
  - History of service
  - Some kind of eco-system: industrial support, community backed by customers
    - IBM for Linux kernel: 1B$ to invest in the next years
  - Some general testing
  - Some penetration testing on the system

FM for OpenSource software

- Dream: all OpenSource software have a formal proof of « security »
  - No solution to social engineering attacks (post-it password)
  - Hardware attacks
  - What about flaws not fitting inside the formal model?

    side channel attacks → new formal model : timing, power)

→ No silver bullet even in my dreams

Let's improve the situation a bit

- Eradicate some attacks based on software:
  - Buffer overflows could be artifacts from the past
    - Basic reusable formal model: « only the language semantics »
    - Still on the Top 25 CWE: C is still a lingua franca
  - Confidentiality of data flows: high level security model
    - Prevent information leaks
    - Check calls to security primitives on sensitive data

Is this feasible?

POLAR SSL

Straightforward, Secure Communication

- Yes we can:
  - PolarSSL 1.1.x: widespread existing SSL stack
  - Prove formally ISO-C compliance for the server side
  - 15 days of work
  - 16kLOC of analyzed code
  - A few buffer overflows found: **proof that no other exist**
  - Next version 1.1.8 will no longer have to be part of the mouse/cat game
- Challenge: lower the cost and enlarge the scope

Conclusion

- FM can help
- FM are mature enough
- Business is not so easy:
  - FM hard to understand: teach more FM, MOOC
  - Do it on real world software to improve efficiency
    → still a research challenge

At TrustInSoft we push this as hard as possible:

- Implement new ideas in scalable frameworks
- Verify the Open Source software around you

Thank you for your attention

We are hiring formal methods experts: get in touch with me

TRUST IN SOFT

Offers

TrustInSoft Analyzer
A software analyzer tool pre-installed on an optimized workstation
Validate in-house software with formal methods

Verification Kits
Pre-validated software components
These widely used pieces of software are already validated for you by our experts with TrustInSoft Analyzer

Experts at your service
Expertise, training, R&D, proof of concept use case, deployment, certification support: smart people for your smart business