Towards 21\textsuperscript{st} century High Assurance Systems

Karthik Pattabiraman

Dependable Systems Lab

University of British Columbia (UBC)
21st century challenges

Software complexity is increasing!

Systems are becoming interconnected
  – Cascading effects of a failure

Errors and failures are the norm, not the exception

Hardware is becoming less reliable!

SoJ-soft errors

$Q_{coll} < Q_{crit} \rightarrow$ no error
$Q_{coll} > Q_{crit} \rightarrow$ soft error
The “Good Enough” Revolution

Source: WIRED Magazine (Sep 2009) – Robert Kapps
http://www.wired.com/gadgets/miscellaneous/magazine/17-09/ff_goodenough

People prefer “cheap and good-enough” over “costly and near-perfect”

Can we build High Assurance systems with this principle?
“Good Enough” High Assurance Systems

• Just reliable enough to get the job done
  – Do not provide the illusion of perfection to end user
  – But do not fail catastrophically or cause severe errors
  – Depends on the application and its context of use
Note ...

- “Good Enough” does not mean arbitrary errors are allowed. In fact, it means the exact opposite.

- By carefully constraining the space of errors (attacks) that occur, we can reason about the behavior of the system in the presence of errors (attacks) and provide high levels of assurance
  - Avoid engineering a 100% reliable system
  - No such thing as one “correct” behavior
Example 1: Error Resilient soft computing [DSN’13][DSN’12]

- Compiler-based technique to identify important variables in a program and protect them
  - To prevent Egregious Data Corruptions (EDCs)
  - Based on static analysis of program’s code
  - Can provide 90% coverage at 10% performance cost provided the application is run in relaxed settings
Example 2: JavaScript-based Web Apps
[ISSRE’11] [ESEM’13]

• Errors abound in JavaScript web applications. Yet we scarcely know about them. How?
• Some errors are much worse than others (DOM-related). It is sufficient to focus on these errors.
  – 80% of high-impact errors are DOM-related
Example 3: Good Enough Security Protection [Eurosys’08][CSF’11]

• Protect critical data or only those parts of the application that are important for security

• Allow attackers to corrupt other parts of application

```c
#include <stdio.h>

int balance = 100;

int x, y;

void chargeCredit()
{
    printf("Charge Credit
");
}

int main()
{
    if (balance < 0)
    {
        chargeCredit();
    }
    else
    {
        x += 10;
        y += 10;
    }
    return 0;
}
```
Takeaway

• Systems are becoming more and more complex, and errors and failures are becoming the norm

• But, most errors don’t matter (much !), either because most users don’t care, or the system intrinsically has sufficient redundancy

• Important to focus on the few errors that matter, and provide targeted protection against them
  – Blanket protection is wasteful and counter-productive
Open Questions

• How do you identify what is an acceptable error and what is not for a particular system?

• How can we formally specify the behavior of a “good enough” system and reason about it?

• If we push this approach to the extreme, can we intentionally build fault-prone systems?