Desafios da Programação na era da Internet das Coisas

Simão Melo de Sousa

introdução

tenho uma profissão extraordinária

sou programador

num contexto invejável

programo muito, e programo o que me parece relevante

estudo a programação, as suas estruturas de dados, os seus algoritmos, as suas linguagens, os seus métodos e os seus aspectos mais inovadores

e, na sequência, tenho o privilégio

de a **ensinar** e de tentar fazê-la **progredir**

classicamente, quem programa muito costuma questionar-se sobre dois aspectos:

- como programar **mais**?
  ou seja levar para a programação novos problemas que não se resolviam antes pela programação (algoritmos, estrutura de dados, novas domínios de aplicação da programação etc.)

- como programar **melhor**?
  ou seja, como fazer melhor o que já sei fazer

tenho divido a minha actividade de programação entre estas duas preocupações, mas falarei hoje mais desta última

as coisas (como entendido na *internet das coisas*) **programam-se**!

vou falar de programação, de linguagens de programação, de bugs, de confiabilidade, de segurança, de como a segurança é um caso particular de confiabilidade etc... num contexto de aparência geral

mas não se esqueça que tudo isso se aplica à internet das coisas

termo que designa um **estado de facto tecnológico** mais do que uma tecnologia ou um conceito próprio novo

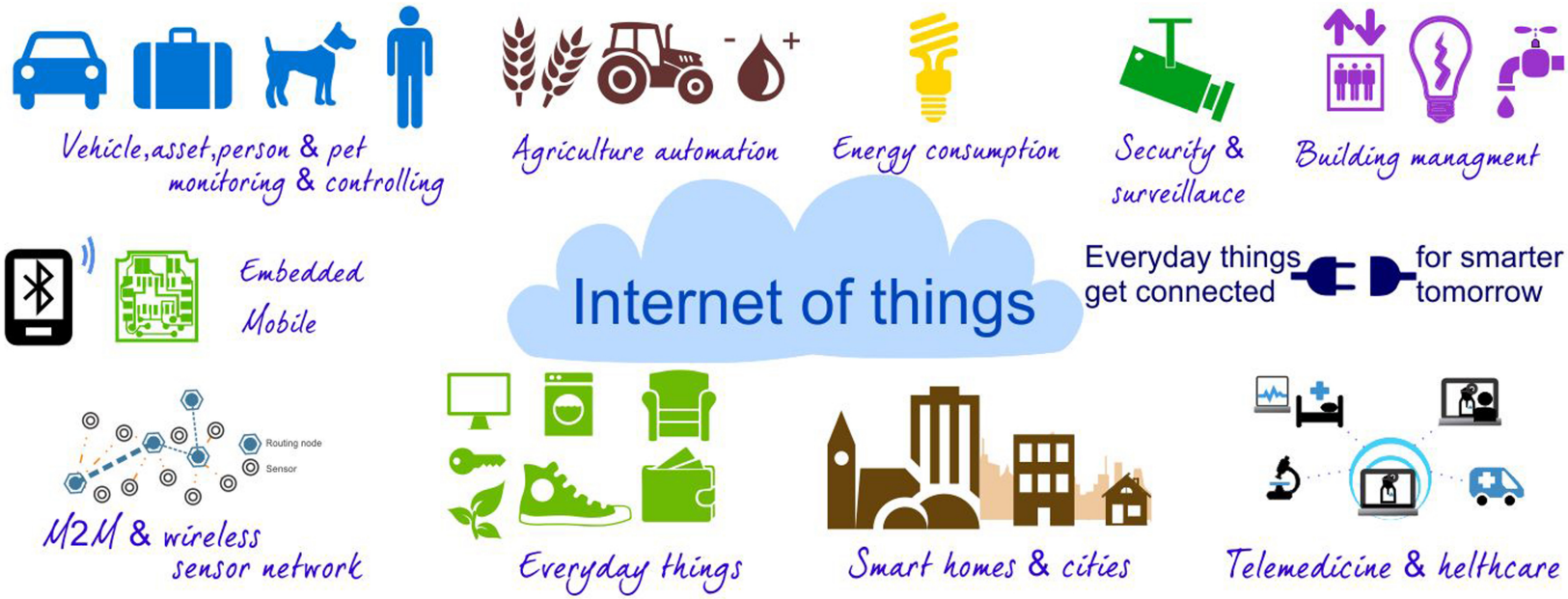
da miniaturização dos dispositivos computacionais, de uma capacitação das tecnologias de comunicação, as **coisas** (que estiveram sempre) à nossa volta podem calcular, tomar decisões simples e comunicar, de forma **maciça e ubíqua**

(previsão para 2025: 150 mil milhões de coisas conectadas)

há, claro, uma recuperação mediática do termo, de várias formas e por vários motivos.... tornou-se uma *buzz-word* como já o foi o **SOA**, o **e-Health** como ainda o é a **Cloud**, o **Big Data** e como os serão muitos outros....

(quem se lembra de *ubiquitous systems*, *pervasive computing*, *large scale distributed systems*, *mobile computing*, *embedded systems*, *grid, cluster systems* etc. com certeza perceberá toda a ironia da primeira afirmação)

perspectivando, numa imagem



ah ah ...promessas ...*smarter tomorrow* ...
voltaremos à questão do "*smarter*"...

desafios, futuro do IoT?

a IoT propõe soluções tecnologicamente adequadas a desafios societais e económicos relevantes

assim, desafios, perspectivas, futuro...

$\Rightarrow$ ... **mais negócio**

por isso **mais capacitação**, *desenfreada*, da IoT, em particular

- mais escala,
- mais eficiência
- mais duração (bateria, alcance, etc.),
- mais interação, mais integração

mas também, e é o meu ponto nesta apresentação:

- mais robustez, confiabilidade
- mais segurança

em termos tecnológicos, a IoT vem acelerar, potenciar ameaças e fraquezas existentes

estas duas dimensões competem entre si e a primeira está no ar do tempo...

assim, parte dos desafios colocados à programação robusta da internet das coisas podem explicar-se como:

as *coisas* computacionais não são, em geral, computadores tradicionais, são sistemas embutidos

logo, dispõe de **recursos limitados**:

- poucos recursos em termos de memória
- alimentam-se por bateria (nem todos)
- recursos computacionais limitados e poupados

obriga a que seja feito um compromisso computacional que tende para a **economia e a eficiência**, em detrimento **da precaução, da redundância, da segurança**

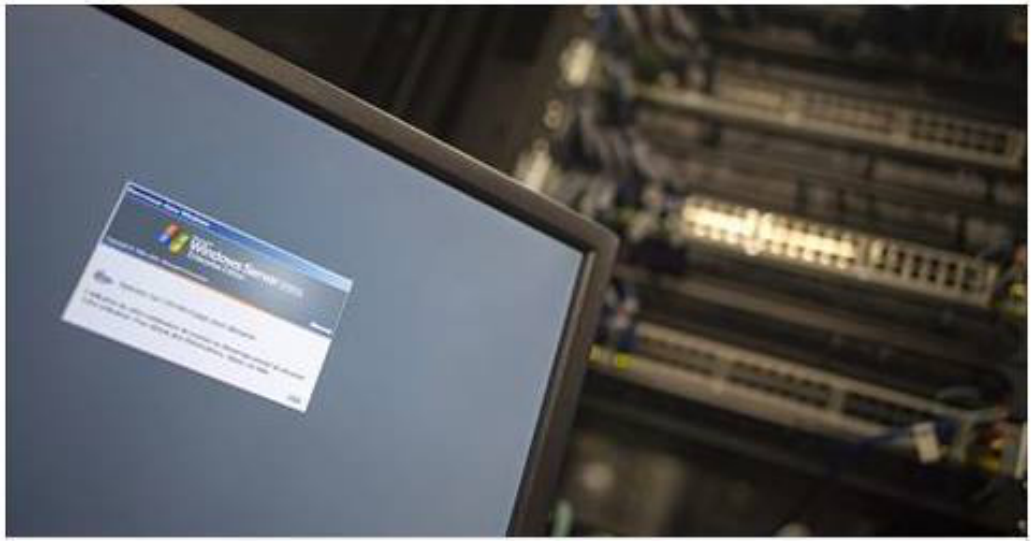
a segurança/robustez de um sistema (informático) mede-se pela força do seu elo mais fraco

em geral, este elo... **somos todos nós**! (*developper*, cliente, utilizador etc.)

mas deixando de lado a engenharia social, a segurança dos sistemas computacionais é hoje em dia muito menos posta em causa por mecanismos criptográficos deficientes (no entanto, é possível... é possível...) ...

... do que por **falhas no software**

cada bug é um ataque a espera de ser



Russia has developed a cyber weapon that can disrupt power grids, according to new research

The weapon has the potential to be the most disruptive yet against electric systems that Americans depend on for daily life.

WASHINGTONPOST.COM | POR ELLEN NAKASHIMA

Gosto Comentar Partilhar

John Ioannidis As I pointed out elsewhere, this is the wrong phrasing. The correct phrasing is "the power grid is running buggy software that is remotely exploitable".

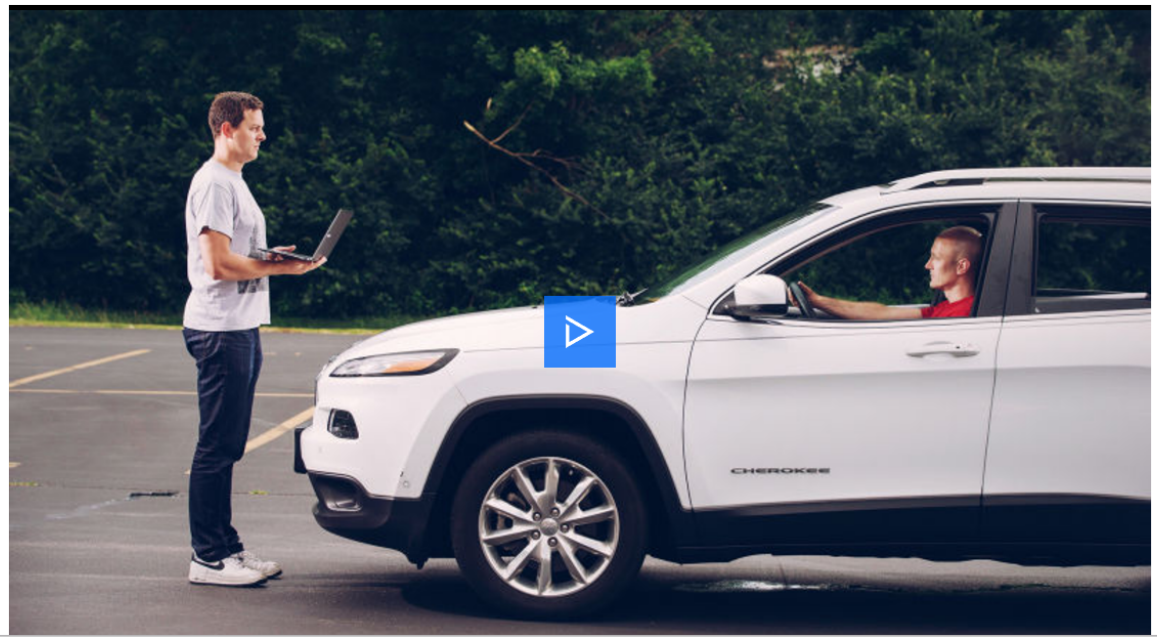
IoT horror stories

don't touch my car!

WIRED Hackers Remotely Kill a Jeep on the Highway—With Me in It

ANDY GREENBERG SECURITY 07.21.15 6:00 AM

HACKERS REMOTELY KILL A JEEP ON THE HIGHWAY—WITH ME IN IT



If consumers don't realize this is an issue, they should, and they should start complaining to carmakers. This might be the kind of software bug most likely to kill someone.

—CHARLIE MILLER

(fonte - link)

2016 Dyn cyberattack

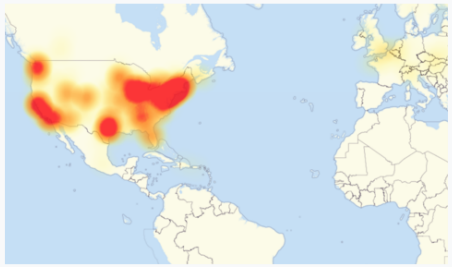
From Wikipedia, the free encyclopedia

The **2016 Dyn cyberattack** took place on October 21, 2016, and involved multiple distributed denial-of-service attacks (DDoS attacks) targeting systems operated by Domain Name System (DNS) provider Dyn, which caused major Internet platforms and services to be unavailable to large swathes of users in Europe and North America.[2][3] The groups Anonymous and New World Hackers claimed responsibility for the attack, but scant evidence was provided.[4][*better source needed*]

As a DNS provider, Dyn provides to end-users the service of mapping an Internet domain name—when, for instance, entered into a web browser—to its corresponding IP address. The distributed denial-of-service (DDoS) attack was accomplished through a large number of DNS lookup requests from tens of millions of IP addresses.[5] The activities are believed to have been executed through a botnet consisting of a large number of Internet-connected devices—such as printers, IP cameras, residential gateways and baby monitors—that had been infected with the Mirai malware. With an estimated throughput of 1.2 terabits per second, the attack is, according to experts, the largest DDoS attack on record.[6]

Contents [hide]

Dyn cyberattack



Map of areas most affected by attack, 16:45 UTC, 21 October 2016.[1]

Time	12:10 – 14:20 UTC 16:50 – 18:11 UTC 21:00 – 23:11 UTC [*citation needed*][*needs update*]
Date	October 21, 2016
Location	Europe and North America, especially the Eastern United States
Type	Distributed denial-of-service
Participants	Unknown
Suspect(s)	New World Hackers, Anonymous (self-claimed)

(fonte wikipedia - link) (outro link)

PRIVACY AND SECURITY FANATIC

By Ms. Smith, Network World | FEB 12, 2017 8:15 AM PT

About

Ms. Smith (not her real name) is a freelance writer and programmer with a special and somewhat personal interest in IT privacy and security issues.

University attacked by its own vending machines, smart light bulbs & 5,000 IoT devices

A university, attacked by its own malware-laced soda machines and other botnet-controlled IoT devices, was locked out of 5,000 systems.

assim a cadeia dos actores envolvidos são

a metodologia e o contexto de desenvolvimento de software	o programador	a linguagem de programação	o compilador	o contexto de execução... ou seja: o sistema operativo, o hardware, a *coisa* na internet,

de quem é a culpa? se é que há um (único) culpado...

desenvolvimento de software

engenharia ou fabrico de software?

o desenvolvimento de software está inserido num contexto económico agreste (só falando deste aspecto)

quando existe, o processo de desenvolvimento de software resume-se a uma actividade meramente técnica, confundindo-a, muitas vezes, a uma obra de artesanato milagrosa

insistindo:

do que me foi algumas vezes dado a ver, as metodologias de desenvolvimento de software actuais são a uma verdadeira **E**ngenharia (de Software) e a sua ciência, o que **a homeopatia é à medicina moderna**

provocação? vejamos por exemplo a usual **equação do software**:

$$\text{Software in the market} = \text{bug report channel}$$

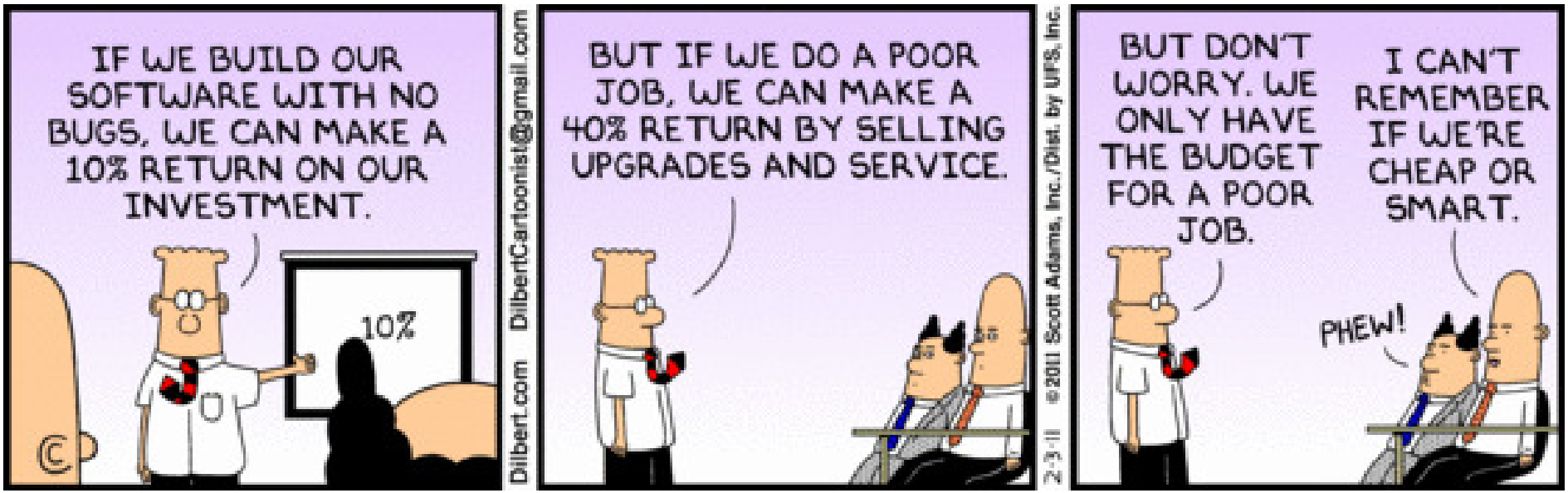
Cambridge University Study States Software Bugs Cost Economy $312 Billion Per Year

PRWeb

According to recent Cambridge University research, the global cost of debugging software has risen to $312 billion annually. The research found that, on average, software developers spend 50% of their programming time finding and fixing bugs. When projecting this figure onto the total cost of employing software developers, this inefficiency is estimated to cost the global economy $312 billion per year.

To put this in perspective, since 2008, Eurozone bailout payments to Greece, Ireland, Portugal, and Spain have totaled $591 billion. These bailout payments total less than half the amount spent on software debugging over the same five year period.

(fonte: link)



Interlúdio: Software Horror Stories (link)

não é de todo um problema fácil, nem sequer exclusivamente técnico, nem vislumbro soluções de curto ou médio prazo

as causas são variadas, complexas e de natureza muito heterogénea (económicas, sociais, políticas, técnicas e científicas...)

teremos numa economia e num mercado como o nosso a **possibilidade de fazer engenharia de software** como fazemos (por exemplo) **engenharia aerospacial**, ou **engenharia civil**?

actualmente... não

(assina-se projectos de informática como se assina projectos de engenharia civil?)

olhando para os **aspectos não técnicos**, é preciso mudar mentalidades, repensar os processos de desenvolvimento,

que as organizações, o mercado, a economia... **reconsiderem o papel do software e da sua manufactura**

há no entanto compromissos, consciencialização e avanços

mas concentremo-nos **nos aspectos técnicos** do desenvolvimento e da execução do software para a internet das coisas

haverá **Leis do Software** para um engenheiro de software como há **Leis da Física** para um engenheiro civil?

as leis do software **existem** mas **não são todas bem estabelecidas**

mas, de qualquer forma, **nem sequer** as bem estabelecidas **são do conhecimento comum** do engenheiro de software

a programação e os programadores

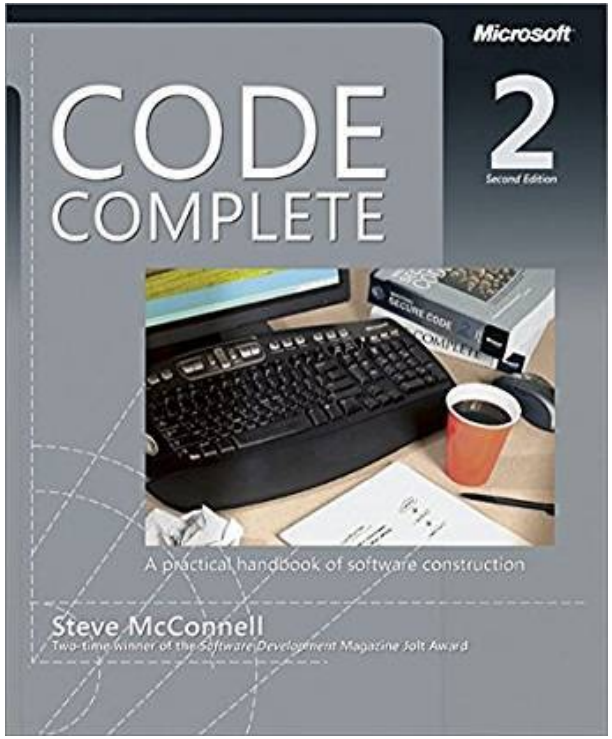
no resto da apresentação falar-se-á muito de

- ***safety***: um programa não causa problemas (não causa divisões por zero, não danifica o ambiente onde está, não causa erros como **segmentation faults** etc.)
  é a **segurança interna, passiva**
- ***security***: um programa sabe prevenir-se de problemas externos (ataques, corrupção dos dados, etc..)
  é **a segurança externa, activa**
- ***soundness*, *correctness*, confiabilidade, correção funcional**
  um programa é *sound*, *correcto*, de confiança, quando cumpre exactamente, nem mais nem menos, a função para o qual foi desenhado

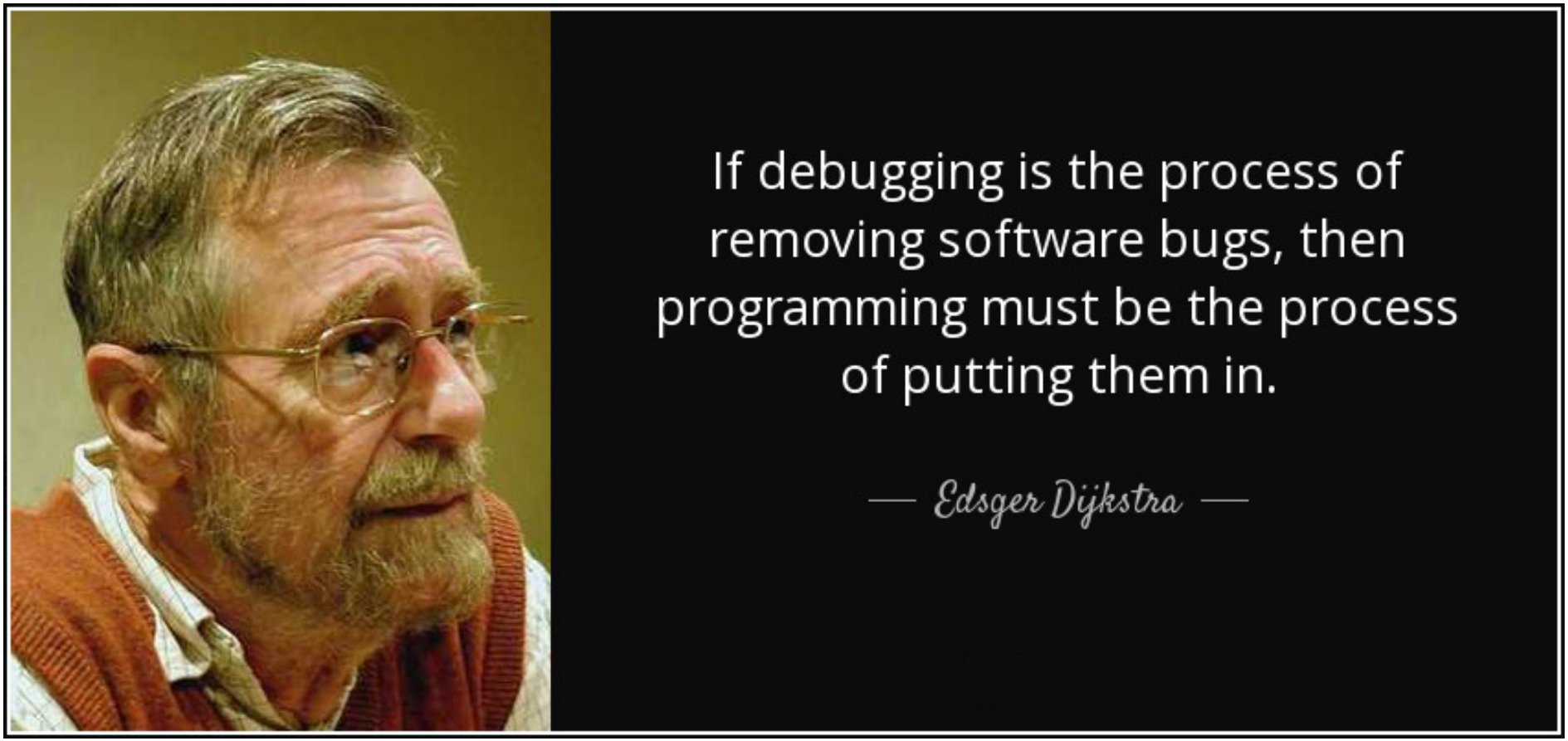
de notar que em português as duas primeiras noções, bem diferentes, se traduzem indiscriminadamente por segurança.

estima-se que na indústria de software entre **15 a 50 bugs** são introduzidos por cada **1000 linhas de código** produzidas

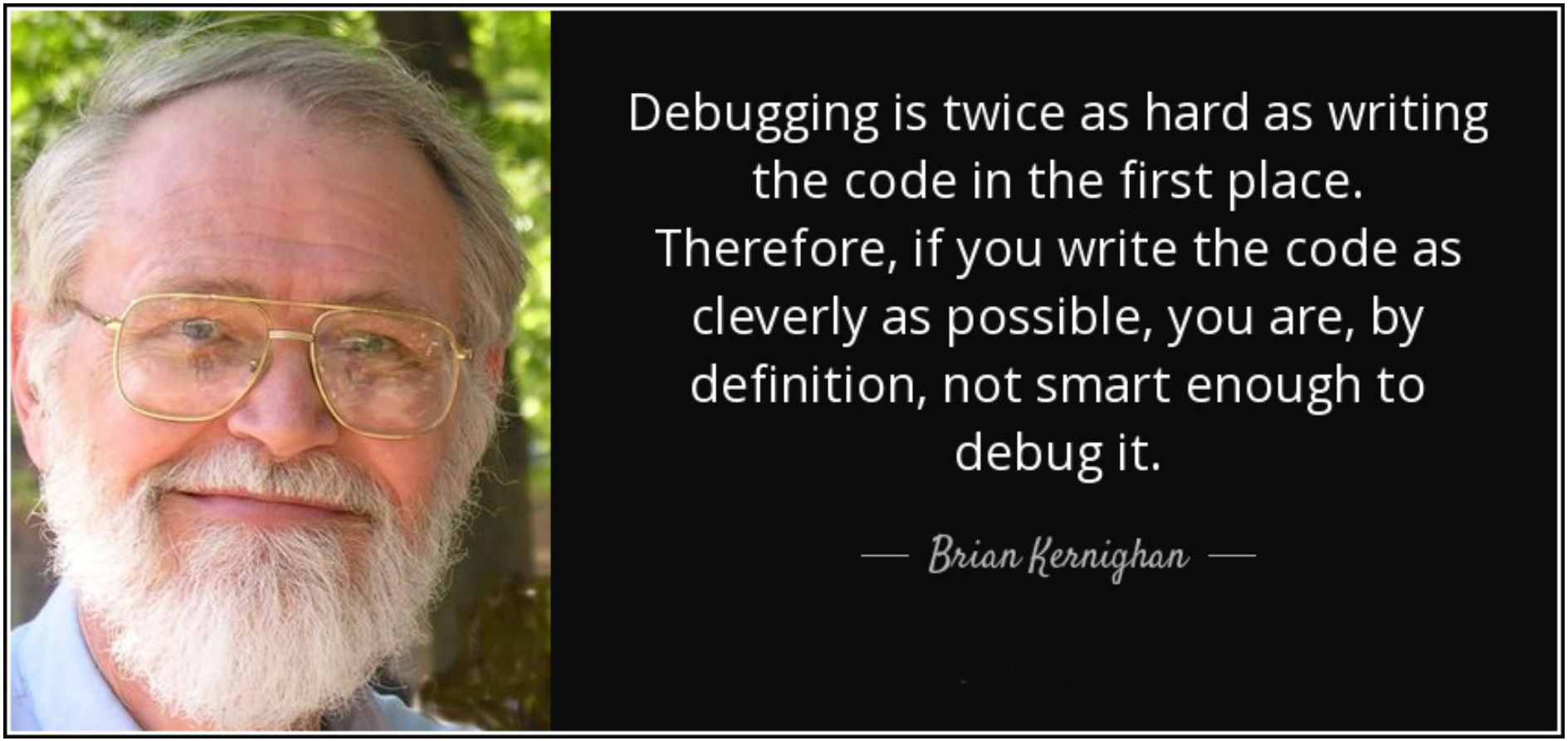
(de Code Complete - Steve McConnell)



... deixando falar os mestres



...deixando falar os mestres

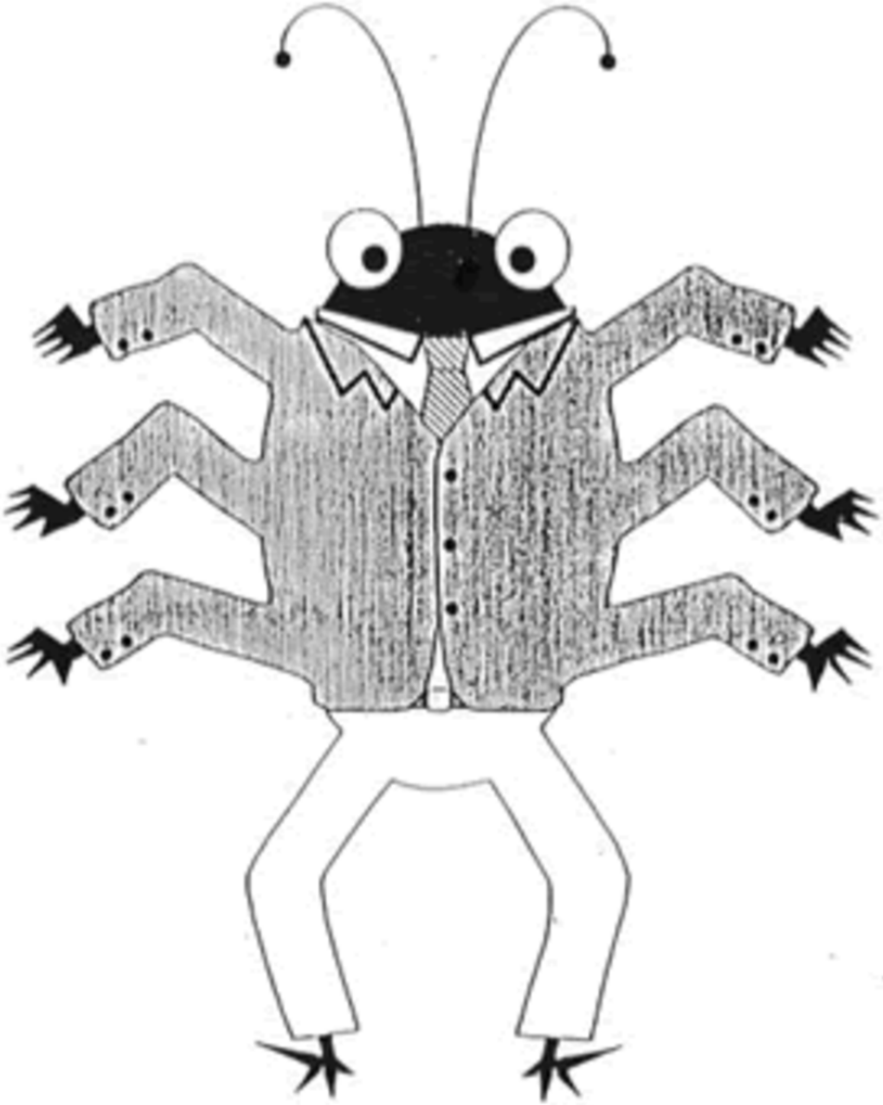


consigo corrigir todos os vossos trabalhos de programação num piscar de olho !

sinto-me realizado no meu papel de professor quando deixo de o poder fazer no fim da vossa passagem pela UBI

(mas que não seja no PhD... há muito tempo que me deveriam ter deslumbrado ;))

quem é mais temível? o bug ou o Terminator?



Vin Cerf sobre a internet (das coisas)

(fonte: (link))

Robots won't take over humans, but buggy software might, according to the Google exec known as the "father of the Internet."
(...)
"I'm actually not as worried about artificial intelligence and robots as I am with software and robots," he said during an event Monday at the Italian embassy. "If there are bugs in the software and some device is operating autonomously with regard to that software, the bugs can cause bad things to happen."
(...)
People aren't careful enough about updating their software – especially when such software is increasingly running heating and ventilation, among other functions – and also making sure the software they're updating doesn't include malware.

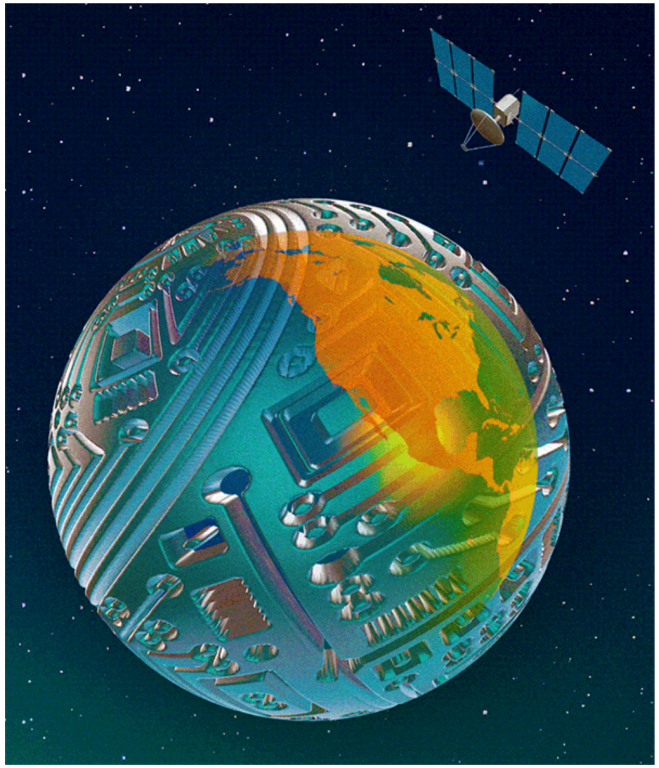
"The big headline I worry about is, '100,000 Refrigerators Attack Bank of America,'" he said.

Bruce Schneier sobre a internet (das coisas)

Click Here to Kill Everyone

With the Internet of Things, we're building a world-size robot. How are we going to control it?

By **Bruce Schneier**



(fonte: (link))

Truism No. 1: On the internet, attack is easier than defense.

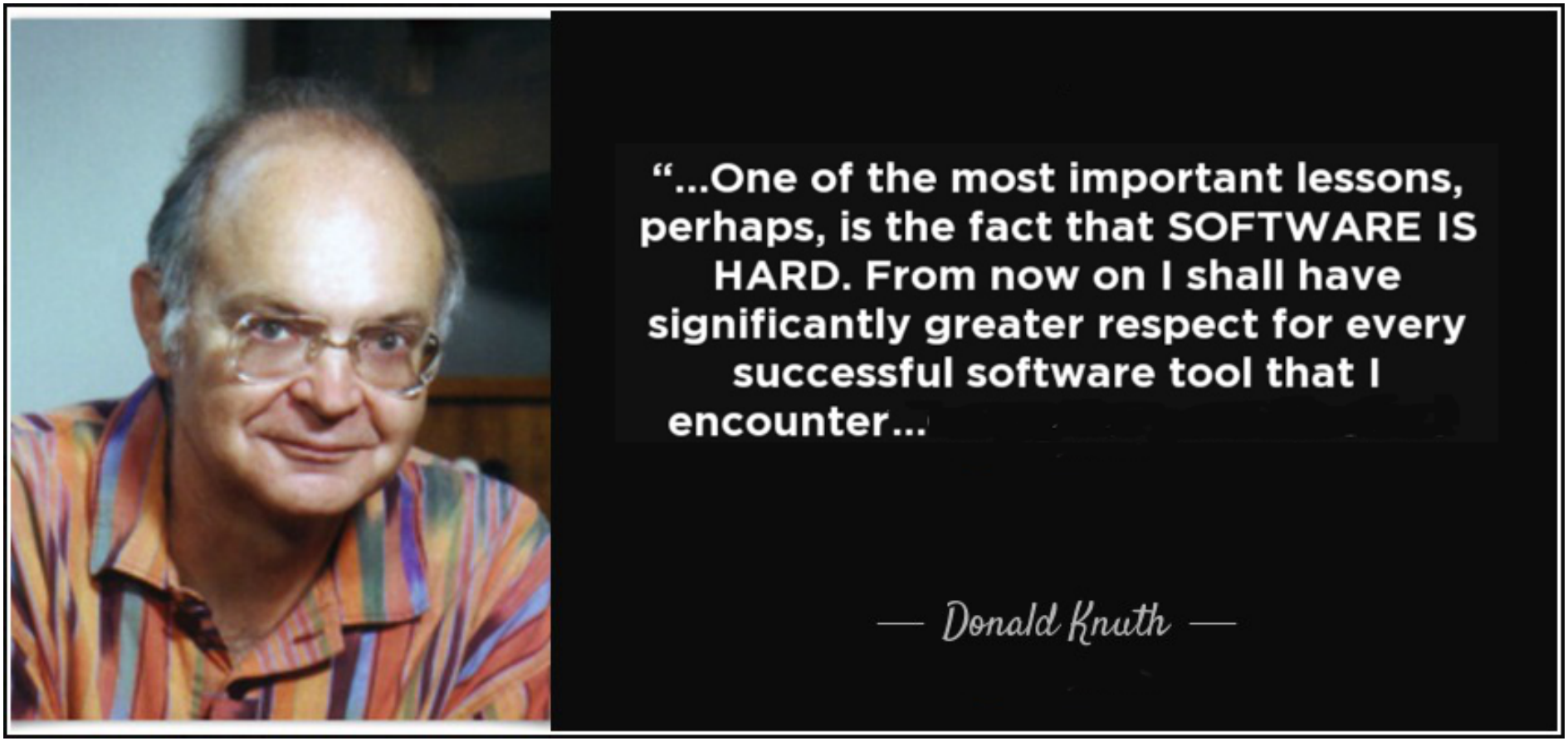
Truism No. 2: Most software is poorly written and insecure.

Truism No. 3: Connecting everything to each other via the internet will expose new vulnerabilities.

Truism No. 4: Everybody has to stop the best attackers in the world.

Truism No. 5: Laws inhibit security research.

uma palavra de conforto do grande mestre dos programadores



de forma resumida é preciso ter em conta o contexto seguinte:

- ser programador (bom e efetivo) requer uma formação exigente e completa

aprender aprender aprender....

.... praticar praticar praticar

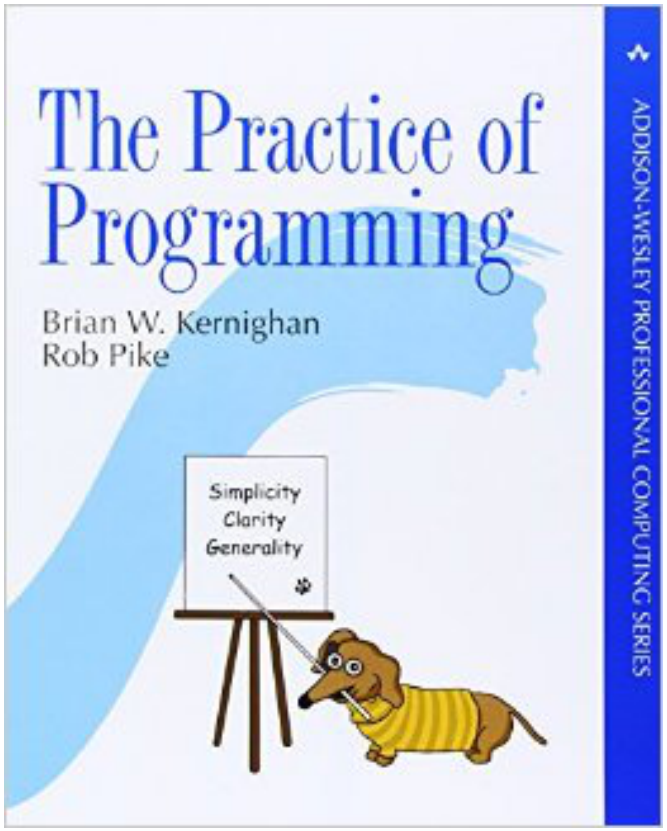
- o **contexto** (da organização, metodológico, económico, etc...) onde os programadores evoluem tem **impacto**
- nunca **considerar código como adquirido**, sempre desconfiar do seu e do código dos outros, conhecer o código que usa, perceber porque funciona bem ou quais são as suas fraquezas

não reduzir a programação a uma actividade de reutilização de código

- *pair programming*, *regression testing* intensivo e efectivo, documentar, refactoring
- conhecer perfeitamente as ferramentas de desenvolvimento, não se confine a uma, **escolher a melhor para a tarefa em causa** (quando o contexto o permite)

Não existam boas linguagens, só existam bons programadores!

E os bons programadores **conhecem** as lições de Brian Kernighan & Robert Pike



dos pontos anteriores, deixem-me insistir e destacar o seguinte:

as linguagens de programação e os seus compiladores propõem cada um deles um compromisso particular na questão "eficiência vs segurança"

no uso de uma ferramenta particular, é **obrigatório** saber quais são esse compromissos

Drama uma assunção implícita e muitas vezes quebrada é: a organização, o compilador, o SO etc.. assumem que

o programador sabe muito bem o que faz

se uma linguagem é permissiva, ela confia que o programador saiba utilizar toda a profusão de flexibilidade a bom proveito e sem asneira

great power comes with great responsability

um programador exímio é alguém que domina do bit, do circuito lógico, ao grafo e à complexiade assimptótica, passando pelas camadas intermédias todas!

já leram algum deles?

já agora... esta é uma foto de alguns dos livros da biblioteca pessoal do Bill Gates

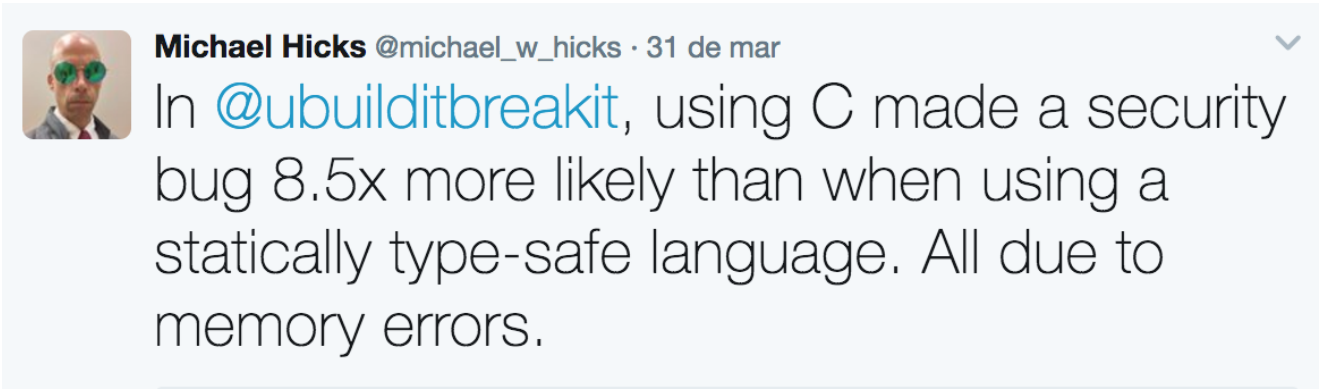
as linguagens de programação

as linguagens populares são:

C, JavaScript, C++, assembly, Java, Python, PHP

muitos destas linguagens, por construção, não são imunes às ***memory leakages***, ou construção programáticas ambíguas, não dispõem de semânticas rigorosas e bem estabelecidas

um tweet recente de um especialista sobre a questão



tweet do 31 de março de 2017

reencontramos aqui os efeitos de compromissos de desenho (eficiência vs confiabilidade) mal ponderados e logo com consequências potencialmente desastrosas

em parte, já o referimos, porque pressupõe um programador exímio

linguagens que permitam gestão explícita de apontadores, não forçam o respeito do ambito dos valores (vetores, espaços memória, números) são sujeitas a estes problemas

The Heartbleed Bug

The Heartbleed Bug is a serious vulnerability in the popular OpenSSL cryptographic software library. This weakness allows stealing the information protected, under normal conditions, by the SSL/TLS encryption used to secure the Internet. SSL/TLS provides communication security and privacy over the Internet for applications such as web, email, instant messaging (IM) and some virtual private networks (VPNs).

The Heartbleed bug allows anyone on the Internet to read the memory of the systems protected by the vulnerable versions of the OpenSSL software. This compromises the secret keys used to identify the service providers and to encrypt the traffic, the names and passwords of the users and the actual content. This allows attackers to eavesdrop on communications, steal data directly from the services and users and to impersonate services and users.

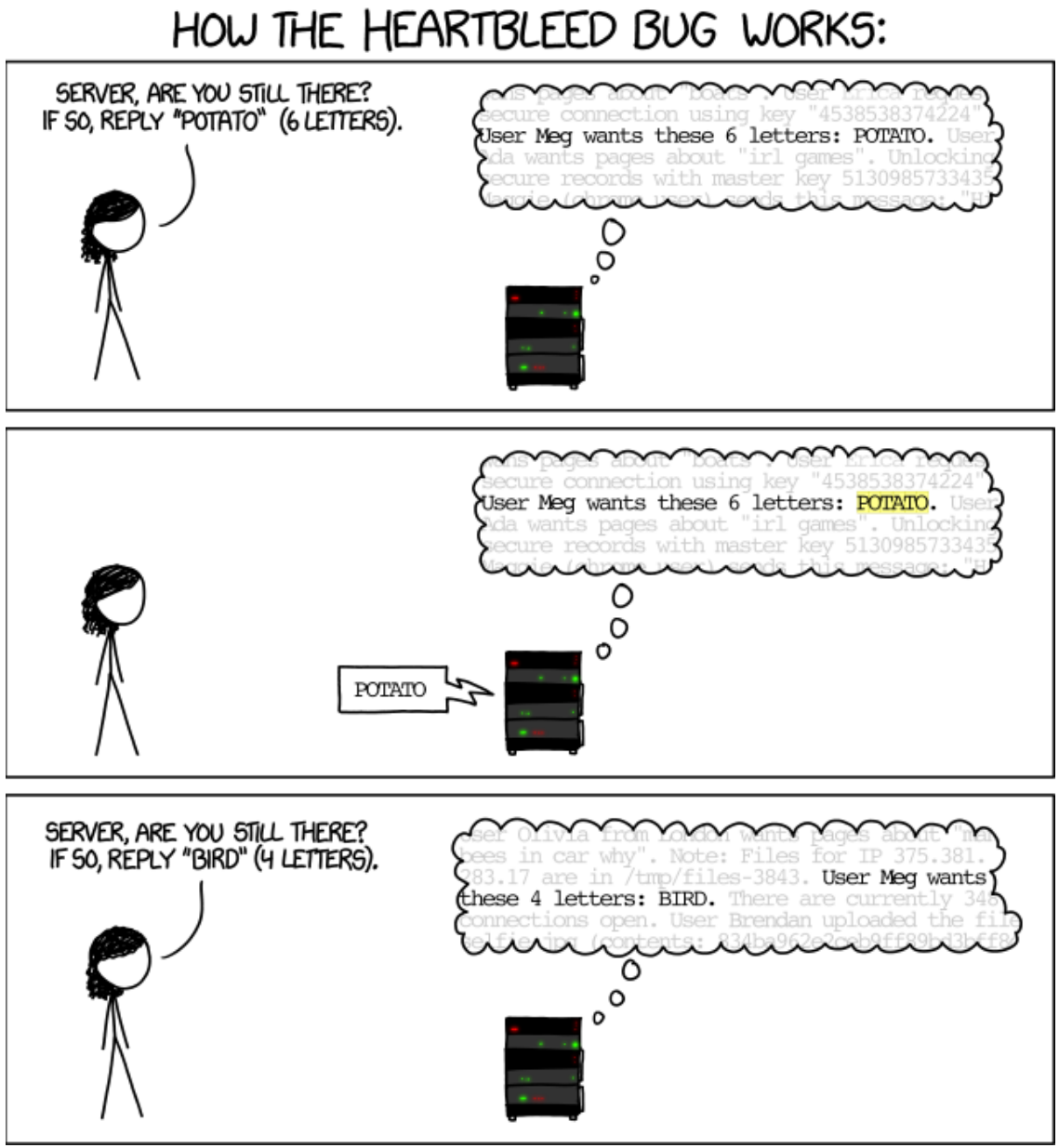
What leaks in practice?

We have tested some of our own services from attacker's perspective. We attacked ourselves from outside, without leaving a trace. Without using any privileged information or credentials we were able steal from ourselves the secret keys used for our X.509 certificates, user names and passwords, instant messages, emails and business critical documents and communication.

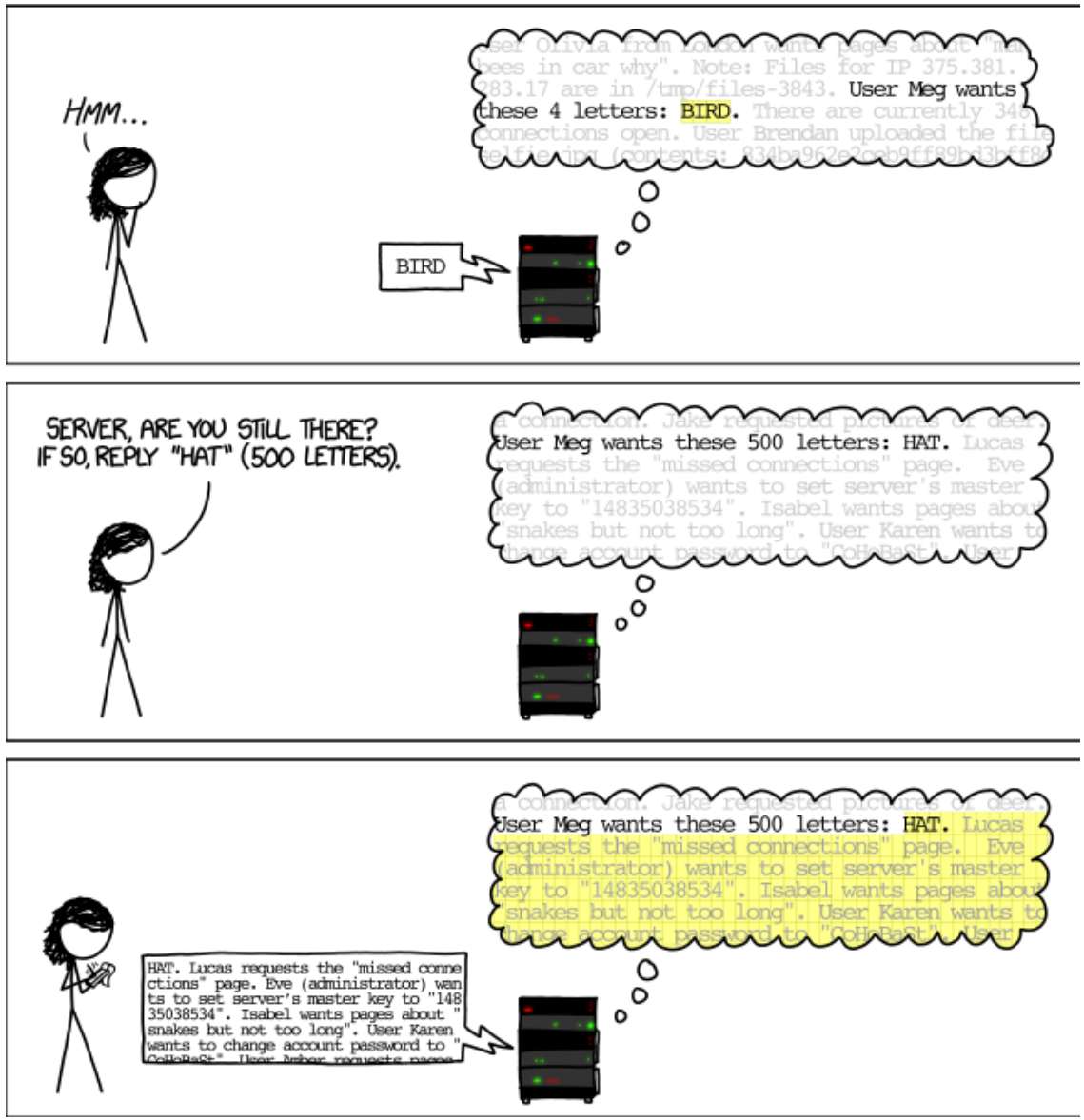
How to stop the leak?

As long as the vulnerable version of OpenSSL is in use it can be abused. Fixed OpenSSL has been released and now it has to be deployed. Operating system vendors and distribution, appliance vendors, independent software vendors have to adopt the fix and notify their users. Service providers and users have to install the fix as it becomes available for the operating systems, networked appliances and software they use.

OpenSSL - heartbleed



OpenSSL - heartbleed



```
int main () {
    float x, y, z, r;
    x = 1.000000019e+38;
    y = x + 1.0e21;
    z = x - 1.0e21;
    r = y - z;
    printf("%f\n", r);
}
```

```
% gcc -Wall float-ex.c -o float-ex
% ./float-ex
0.000000
%
```

nos `floats`, não é necessariamente verdade que $(x + a) - (x - a) = 2a$!!!

(fonte: link)

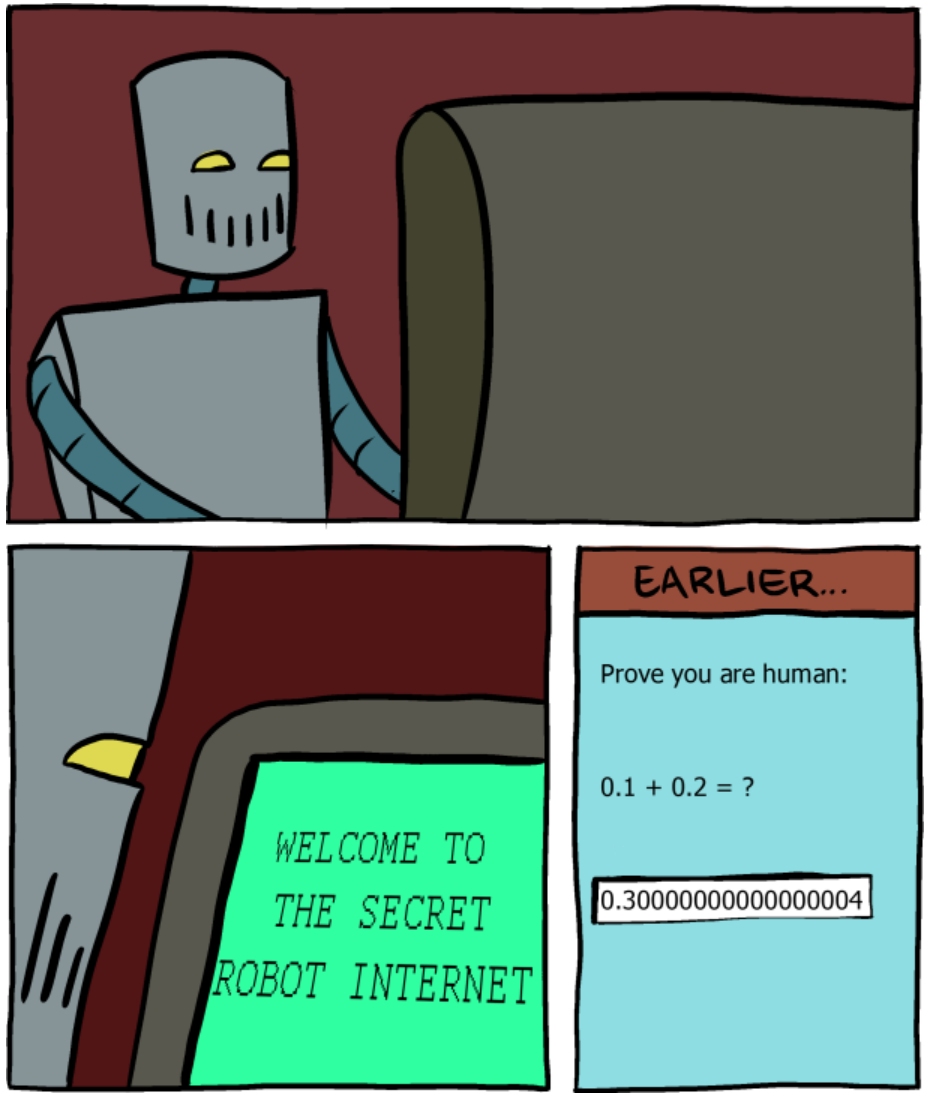
Beware of the floats!

seguem a norma IEEE 754 - **são aproximações com margem de erro!**
surpresas podem advir deste facto...

ver por exemplo:

The floating point guide (link)

What Every Computer Scientist Should Know About Floating-Point Arithmetic (link)



que bruxaria é essa?

```
#pragma check_stack(off)
#include <string.h>
#include <stdio.h>

void foo(const char* input){
    char buf[10];
    printf("My stack looks like:\n%p\n%p\n%p\n%p\n%p\n%p\n\n");
    strcpy(buf, input);
    printf("%s\n", buf);
    printf("Now the stack looks like:\n%p\n%p\n%p\n%p\n%p\n%p\n\n");
}

void bar(void){
    printf("Augh! I've been hacked!\n");
}

int main(int argc, char* argv[]){
    printf("Address of foo = %p\n", foo);
    printf("Address of bar = %p\n", bar);
    if (argc != 2)  {
        printf("Please supply a string as an argument!\n");
        return -1;}
     foo(argv[1]);
    return 0;
}
```

```
Address of foo = 00401000
Address of bar = 00401050
My stack looks like:
00000000
00000A28
7FFDF000
0012FEE4
004010BB
00321599

ABCDEFGHIJKLMNOPP▶@
Now the stack looks like:
44434241
48474645
4C4B4A49
504F4E4D
00401050
00321599

Augh! I've been hacked!
```

(fonte: link)

que bruxaria é essa?

```c
#include <stdio.h>

int main()
{
  long diff,size = 8;
  char *buf1;
  char *buf2;

  buf1 = (char *)malloc(size);
  buf2 = (char *)malloc(size);
  if(buf1 == NULL || buf2 == NULL)
  {
        perror("malloc");
        exit(-1);
  }

  diff =  (long)buf2 -  (long)buf1;
  printf("buf1 = %p \n buf2 = %p \n diff %d\n",buf1,buf2,diff);
  memset(buf2,'2',size);
  printf("BEFORE: buf2 = %s\n",buf2);
  memset(buf1,'1',diff+3);  /* We overwrite 3 chars */
  printf("AFTER:  buf2 = %s\n",buf2);

  return 0;
}
```

```
buf1 = 0x804a5f8
buf2 = 0x804a608
diff = 16
BEFORE: buf2 = 22222222
AFTER:  buf2 = 11122222
```

(fonte: link)

mais apontadores sobre a construção de ataques em programas

https://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/index.html
https://www.cl.cam.ac.uk/ rja14/book.html

https://www.cs.ru.nl/E.Poll/hacking/
https://users.ece.cmu.edu/ adrian/630-f04/readings/AlephOne97.txt
https://crypto.stanford.edu/cs155/papers/formatstring-1.2.pdf

https://www.cs.utexas.edu/ shmat/courses/cs361s/blexim.txt
http://hamsa.cs.northwestern.edu/media/readings/free.pdf
https://dl.packetstormsecurity.net/papers/bypass/GOT_Hijack.txt

lição:
(*stack based / heap based*) *buffer ovwerflows* surgem muito facilmente e subtilmente
até o `printf` do C pode ser usado como porta de acesso a ataques!

usar com muito juízo funções como `memset`, `malloc` e `free` (e.g. dangling pointers), a aritmética de apontadores

evitar como a peste funções como `gets()`, `sprintf()`, `strcat()`, `strcpy()`, `vsprintf()`

os famosos WAT (obrigatório conhecer!)

WAT - A lightning talk by Gary Bernhardt from CodeMash 2012
(link do vídeo)

outro WAT (link)

```
In [7]: a = 256
        b = 256
        a is b

Out[7]: True
```

```
In [8]: a = 257
        b = 257
        a is b

Out[8]: False
```

este WAT explica-se bem: é consequência de uma optimização do compilador/interpretador de python (para valores entre -5 e 256)

(para saber mais sobre WATs de python: (link))

há WATs e WATs

todas as linguagens têm os seus WATs

umas mais do que outras...

e há WAT e WAT....

os WATs explicam-se de várias formas

- por questões próprias aos compiladores (optimizações, etc..)
- ou, as vezes, por falhas de desenho *desastrosas* da linguagem ou do compilador....

conhecer bem as linguagens

conhecer bem os programas e as sua lógica interna

saber optar por um estilo defensivo

ou...

ter linguagens menos permissivas, mais seguras, mais "mandonas" e desconfiadas...

há custos... mas não valerão eles a pena?

compiler vs wife

perceberam a lição? então vejamos

```
/*P0*/

int x = 0;

int f0(int y){
  return y * x ;
}

int f1 (int y){
x = y;
return 0;
}

void main (){
z = f0(10) + f1(100);
}
```

```
/*P1*/

void main (){
  int i;
  int t[100] = {0,1,2,3, ... ,99};
  while(i<100){
    t[i]++:
    i++;
  }
}
```

```
/*P2*/

void main (){
  float f = 0.;
  for (int i = 0; i < 1 000 000; i++){
    f = f + 0.1;
  }
}
```

```
/*P0*/

int x = 0;

int f0(int y){
  return y * x ;
}

int f1 (int y){
x = y;
return 0;
}

void main (){
z = f0(10) + f1(100);
}
```

ordem de execução:

não especificada em C
especificada em Java
da esquerda para a direita, z = 0
da direita para a esquerda, z = 1000

subtilidades semânticas...

```
/*P1*/

void main (){
  int i;
  int t[100] = {0,1,2,3, ... ,99};
  while(i<100){
    t[i]++:
    i++;
  }
}
```

Inicialização:

- **runtime error em Java**
- **set-up qualquer em C** (o valor anteriormente atribuído ao local memória em causa)

```
/*P2*/

void main (){
  float f = 0.;
  for (int i = 0; i < 1 000 000; i++){
    f = f + 0.1;
  }
}
```

Flutuantes, again...:

- **0.1 não é representado com exactidão**

  para que valor o compilador o arredonda ?
- **rounding errors**

  qual é o modo de arredondamento em tempo de execução ?

posso, neste exemplo articifical mas representativo, ganhar o acesso aos privilégios de root?

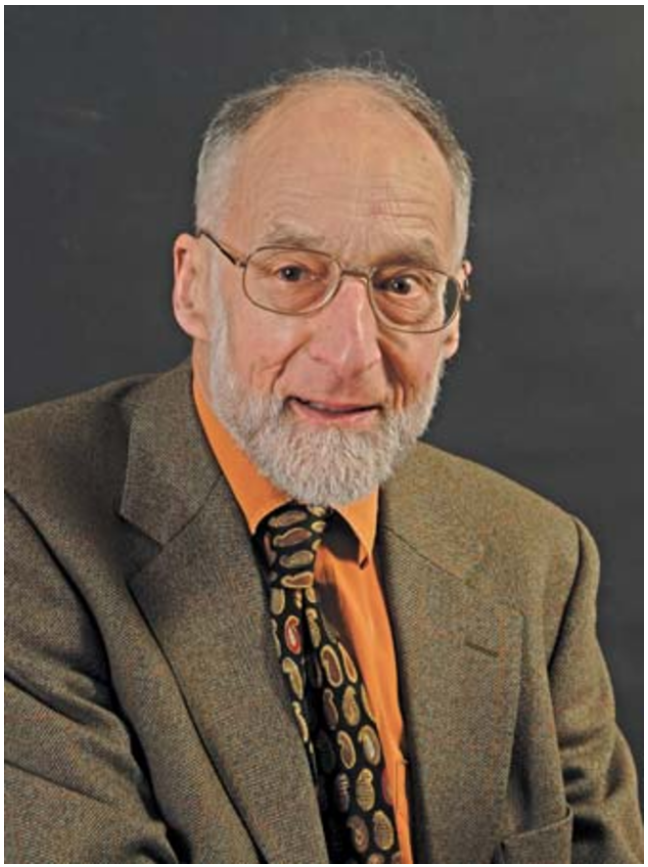
```
#include <stdio.h>
#include <string.h>
int main() {
  int passcorrect = 0;
  char passw[5];
  gets(passw);
  if (passcorrect == 0)
     printf ("\n Wrong Password \n");
  else
     printf ("\n Root privileges given to the user''
               ", passcorrect=%d \t passw=%s\n",
                passcorrect, passw);
  return 0;
}
```

```
#include <stdio.h>
#include <string.h>
int main() {
  int passcorrect = 0;
  char passw[5];
  gets(passw);
  if (passcorrect == 0)
     printf ("\n Wrong Password \n");
  else
     printf ("\n Root privileges given to the user''
               ", passcorrect=%d \t passw=%s\n",
                passcorrect, passw);
  return 0;
}
```

```
$ gcc pass.c -o pass
$ ./pass
wteretw

 Root privileges given to the user, passcorrect=30580   passw=wteretw
```

Robin Milner (1934 - 2010) foi um dos grandes cientistas das linguagens de programação



trabalhou para a emergência e a consolidação das linguagens com forte ênfase na segurança e confiabilidade

foi o grande obreiro para toda uma dinastia de linguagens de programação com inferência de tipos e tipos expressivos onde

well typed program cannot go wrong

a comunidade das linguagens de programação assumiu gradualmente o programa de Robin Milner (e de outros grandes percursores como ele)

nos dias de hoje é claro que é uma falácia **contrapor** *categoricamente*, no desenho de linguagens de programação, **a eficiência com a segurança**,

é hoje um anacronismo

as linguagens e respectivos compiladores devem chamar a si a responsabilidade que lhes cabe

que o programador se preocupa com a definição e a eficiência do seu algoritmo, da correção funcional,
a linguagem deve **guiar, disciplinar** o programador para a escrita de código inócuo
deve tratar da **eficiência do código máquina**, da sua **inocuidade**

mas também (muito, muito importante!!!!) de uma **tradução fidedigna** das ideias do programador

no more memory leaks! more statically type safe languages!

a comunidade trabalha assim no desenho de

- melhores linguagens de programação com melhor suporte ao programador (API, tool support)
- melhor sintaxe, melhor tipagem com sistemas de tipos cada vez mais poderosos e discriminativos
  - é milagroso o que se consegue fazer com sistemas de tipos ricos, ***compile time***:
  - por exemplo, em linguagens com suporte a (variantes de) tipos dependentes, definir o tipos dos vectores de tamanho até 10, de inteiros pares, ou então o tipo dos ficheros *safe*, que quando são abertos são garantidamente fechados... insisto: verificado ***compile time*!**
- melhores IDEs
- melhores mecanismos (estáticos) de verificação e de análise de código
- melhores mecanismos de optimização

linguagens de programação emergentes (para a IoT)

vejamos então algumas propostas resultantes

algumas das linguagens de programação emergentes para a internet são

Rust	js_ocaml	Reason	Kotlin...Scala
Go	Hack	Swift	e muitas outras...

felizmente, parecem resultar de uma reflexão sobre os problemas endêmicos aqui abordados

são linguagens com um **sistema estático de tipos fortes** (*static type safety*) e uma gestão muito criteriosa da concorrência, da memória e da mutabilidade (das variáveis)

de notar que alguns dos compiladores destas linguagens compilam para (um subconjunto criteriosamente seguro de) javascript ou até para PHP

Documentation Install Community Contribute

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

See who's using Rust.

Install Rust 1.16.0

March 16, 2017

Featuring

- zero-cost abstractions
- move semantics
- guaranteed memory safety
- threads without data races
- trait-based generics
- pattern matching
- type inference
- minimal runtime
- efficient C bindings

Run

```
// This code is editable and runnable!
fn main() {
    // A simple integer calculator:
    // `+` or `-` means add or subtract by 1
    // `*` or `/` means multiply or divide by 2

    let program = "+ + * - /";
    let mut accumulator = 0;

    for token in program.chars() {
        match token {
            '+' => accumulator += 1,
            '-' => accumulator -= 1,
            '*' => accumulator *= 2,
            '/' => accumulator /= 2,
            _ => { /* ignore everything else */ }
        }
    }

    println!("The program \"{}\" calculates the value {}",
             program, accumulator);
}
```

The Go Programming Language

Documents | Packages | The Project | Help | Blog | Search

Try Go

Pop-out

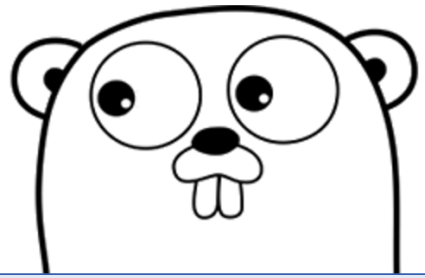
```
// You can edit this code!
// Click here and start typing.
package main

import "fmt"

func main() {
        fmt.Println("Hello, 世界")
}
```

Hello, World! | Run | Share | Tour

Go is an open source programming language that makes it easy to build simple, reliable, and efficient software.



Download Go

Binary distributions available for
Linux, Mac OS X, Windows, and more.

Featured video

Featured articles

HTTP/2 Server Push

HTTP/2 is designed to address many of the failings of HTTP/1.x. Modern web pages use many resources: HTML, stylesheets, scripts, images, and so on. In HTTP/1.x, each of these resources must be requested explicitly. This can be a slow process. The browser starts by fetching the HTML, then learns of more resources incrementally as it parses and evaluates the page. Since the server must wait for the browser to make each request, the network is often idle and underutilized.

Facebook - Reason

Reason | Projects | Tools | GitHub

REASON
- Install
- REPL

SYNTAX BASICS
- Primitives / Operations
- Let Bindings
- Types Basics
- Tuples
- Records
- Destructuring
- Functions
- Variants
- Loops and Tests
- Type Parameters
- Exceptions

DIVING DEEPER
- Curried Functions
- Expressive Pattern Matching
- Objects
- JSX
- Syntactic Sugar (Abbreviations)
- Automatic Printer Generation

ADVANCED
- Bindings Revisited
- Modules

REASON

A new developer experience for rapidly building fast, safe systems.

Syntax Build Share

`Reason` is a new interface to OCaml - a highly expressive dialect of the ML language featuring type inference and static type checking.

`Reason` provides a new syntax and toolchain for editing, building, and sharing code, and will evolve in the open as a community collaboration. `Reason` already has contributors across several organizations/companies.

Compare JS **Compare ML**

What is Hack?

Hack is a programming language for HHVM. Hack reconciles the fast development cycle of a dynamically typed language with the discipline provided by static typing, while adding many features commonly found in other modern programming languages.

Hack provides instantaneous type checking by incrementally checking your files as you edit them. It typically runs in less than 200 milliseconds, making it easy to integrate into your development workflow without introducing a noticeable delay.

The following are some of the important language features of Hack. For more information, see the full documentation, or follow through the quick, interactive tutorial.

Type Annotations

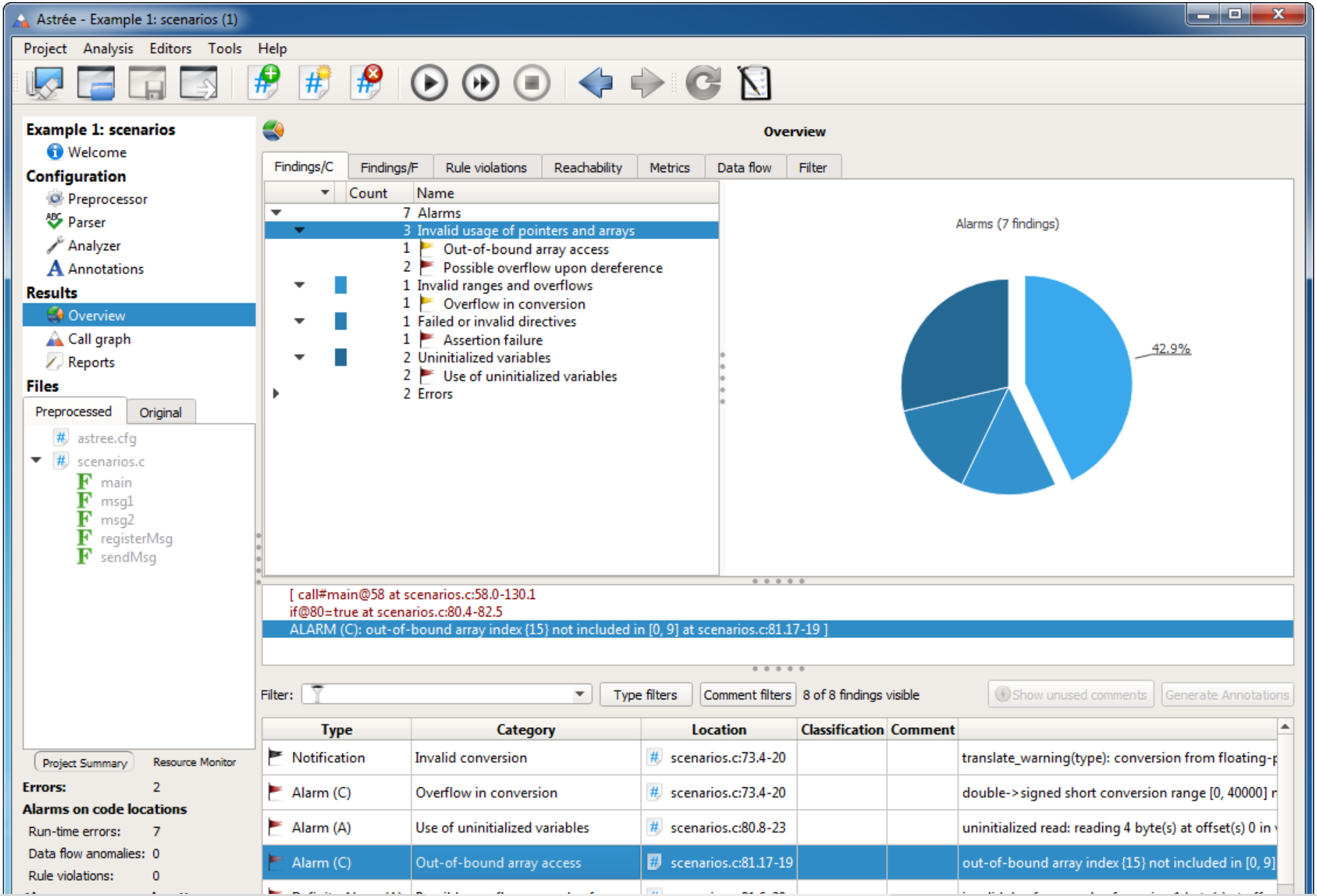
Type annotations allow for code to be explicitly typed on parameters, class member variables and return values.

Jetbrains - Kotlin

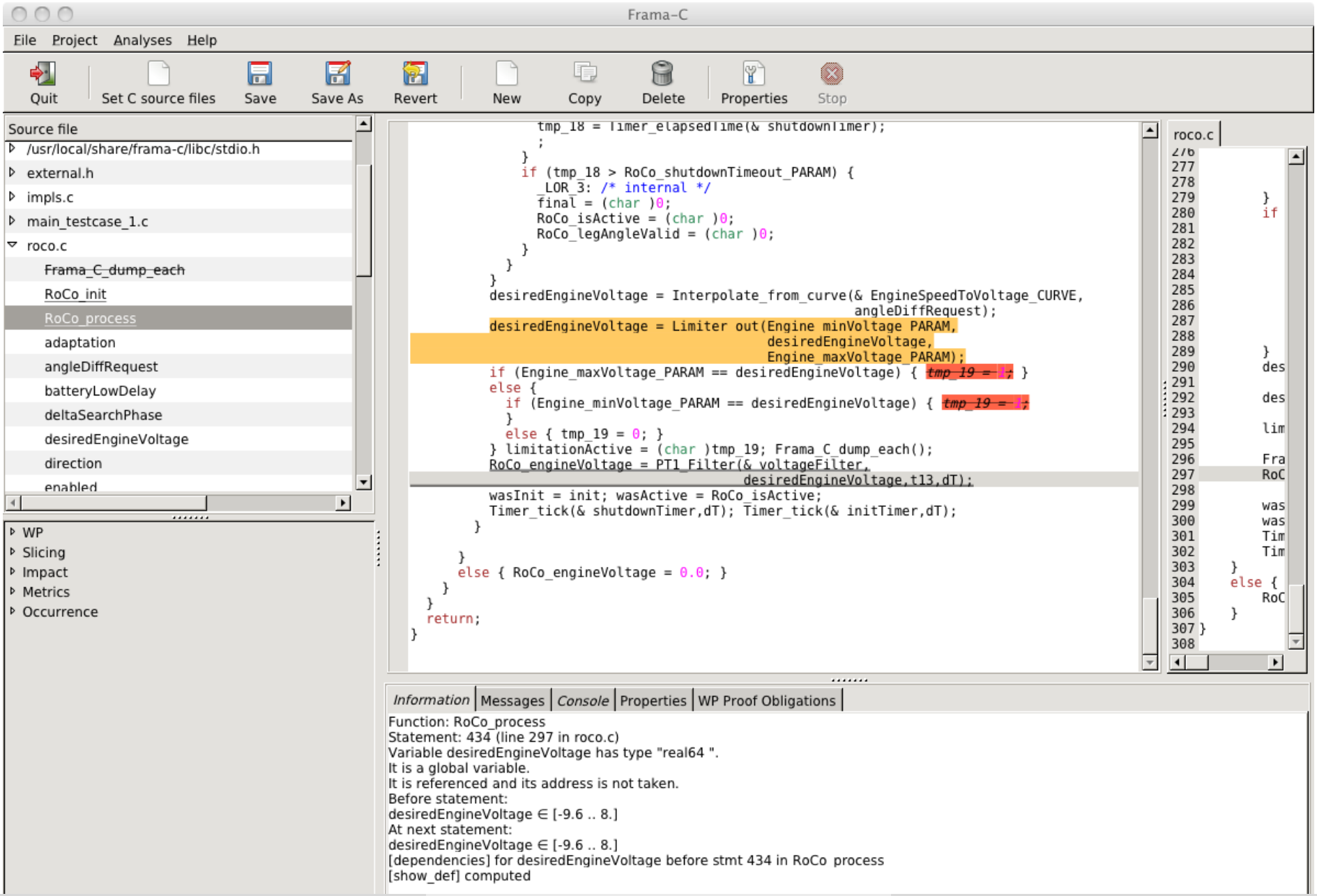
fora do compilador:
verificar a segurança e a correção dos programas desenvolvidos

raj@raj: ~

```
raj@raj:~$ gcc -o memory_leak memory_leak.c
raj@raj:~$ valgrind --tool=memcheck --leak-check=yes ./memory_leak
==2405== Memcheck, a memory error detector
==2405== Copyright (C) 2002-2011, and GNU GPL'd, by Julian Seward et al.
==2405== Using Valgrind-3.7.0 and LibVEX; rerun with -h for copyright info
==2405== Command: ./memory_leak
==2405==
5
==2405==
==2405== HEAP SUMMARY:
==2405==     in use at exit: 4 bytes in 1 blocks
==2405==   total heap usage: 1 allocs, 0 frees, 4 bytes allocated
==2405==
==2405== 4 bytes in 1 blocks are definitely lost in loss record 1 of 1
==2405==    at 0x402BE68: malloc (in /usr/lib/valgrind/vgpreload_memcheck-x86-linux.so
)
==2405==    by 0x8048458: main (in /home/raj/memory_leak)
==2405==
==2405== LEAK SUMMARY:
==2405==    definitely lost: 4 bytes in 1 blocks
==2405==    indirectly lost: 0 bytes in 0 blocks
==2405==      possibly lost: 0 bytes in 0 blocks
==2405==    still reachable: 0 bytes in 0 blocks
==2405==         suppressed: 0 bytes in 0 blocks
==2405==
==2405== For counts of detected and suppressed errors, rerun with: -v
==2405== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
raj@raj:~$
```


Astrée - Example 1: scenarios (1)
Project Analysis Editors Tools Help
Example 1: scenarios
Welcome
Configuration
Preprocessor
Parser
Analyzer
Annotations
Results
Overview
Call graph
Reports
Files
Preprocessed
Original
astree.cfg
scenarios.c
main
msg1
msg2
registerMsg
sendMsg
Project Summary
Resource Monitor
Errors: 2
Alarms on code locations
Run-time errors: 7
Data flow anomalies: 0
Rule violations: 0
Overview
Findings/C
Findings/F
Rule violations
Reachability
Metrics
Data flow
Filter
Count
Name
7 Alarms
3 Invalid usage of pointers and arrays
1 Out-of-bound array access
2 Possible overflow upon dereference
1 Invalid ranges and overflows
1 Overflow in conversion
1 Failed or invalid directives
1 Assertion failure
2 Uninitialized variables
2 Use of uninitialized variables
2 Errors
Alarms (7 findings)
42.9%
[call#main@58 at scenarios.c:58.0-130.1
if@80=true at scenarios.c:80.4-82.5
ALARM (C): out-of-bound array index {15} not included in [0, 9] at scenarios.c:81.17-19]
Filter:
Type filters
Comment filters
8 of 8 findings visible
Show unused comments
Generate Annotations
Type
Category
Location
Classification
Comment
Notification
Invalid conversion
scenarios.c:73.4-20
translate_warning(type): conversion from floating-p
Alarm (C)
Overflow in conversion
scenarios.c:73.4-20
double->signed short conversion range [0, 40000] r
Alarm (A)
Use of uninitialized variables
scenarios.c:80.8-23
uninitialized read: reading 4 byte(s) at offset(s) 0 in
Alarm (C)
Out-of-bound array access
scenarios.c:81.17-19
out-of-bound array index {15} not included in [0, 9]

frama-c - analise estática e verificação deductiva



um exemplo de ferramenta: infer (link), da Facebook

Infer is a static analysis tool - if you give Infer some Java or C/C++/Objective-C code it produces a list of potential bugs. Anyone can use Infer to intercept critical bugs before they have shipped ti users, and help prevent crashes or poor performance.

foram listadas algumas ferramentas, técnicas para assegurar a *safety* e a *security*

podemos ir mais longe e conseguir assegurar a correção funcional?

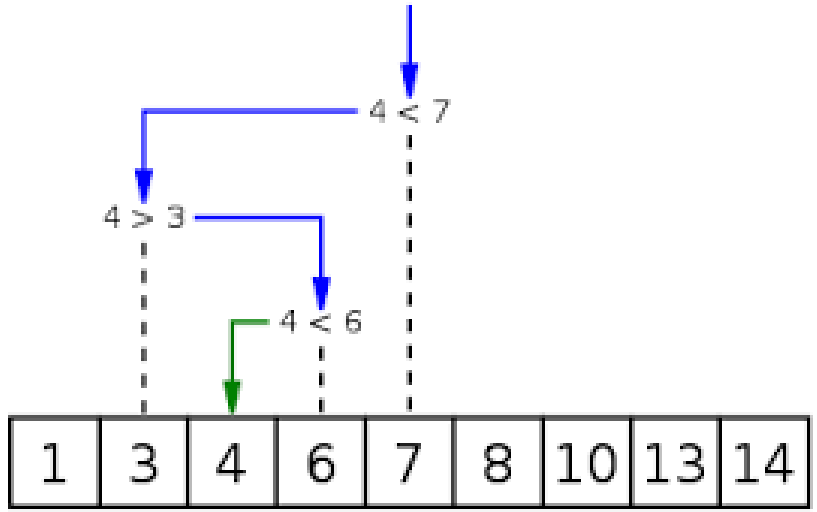
o **Eldorado** da engenharia de software

nem sempre, mas muito mais vezes do que pensamos!

uma ilustração com base num exemplo famoso

binary search: primeira publicação em 1946...

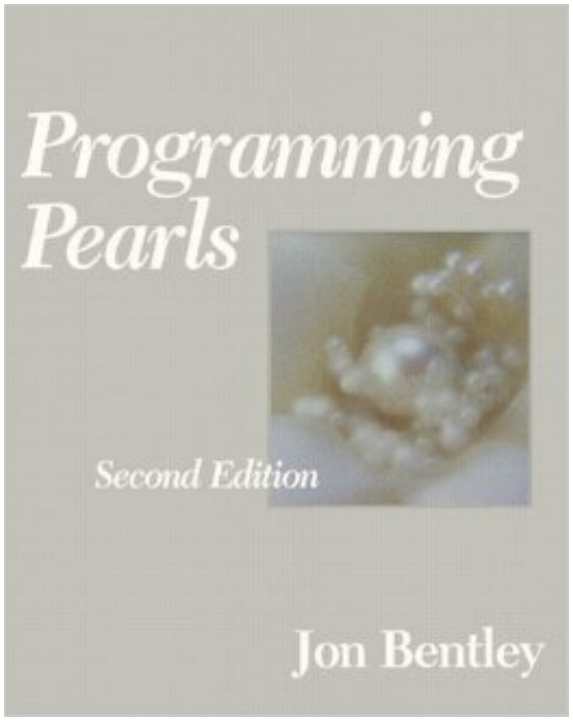
...primeira publicação **sem bugs** em 1962



uma ilustração com base num exemplo famoso

Jon Bentley - Programming Pearls. 1986 (2^{nd} ed. 2000)

(coluna 4) - writing correct programs
The challenge of binary search



(como visto p.37)

Warning
Boring material ahead
skip to section 4,4
when drowsiness strikes

explicação concisa e claríssima da situação...

e mesmo assim...

uma ilustração com base num exemplo famoso

em 2006, foi encontrado um bug na biblioteca padrão de Java... na pesquisa binária

Joshua Bloch, Google Research Blog
"Nearly All Binary Searches and Mergesorts are Broken"

estava lá há mais de 9 anos

o bug :

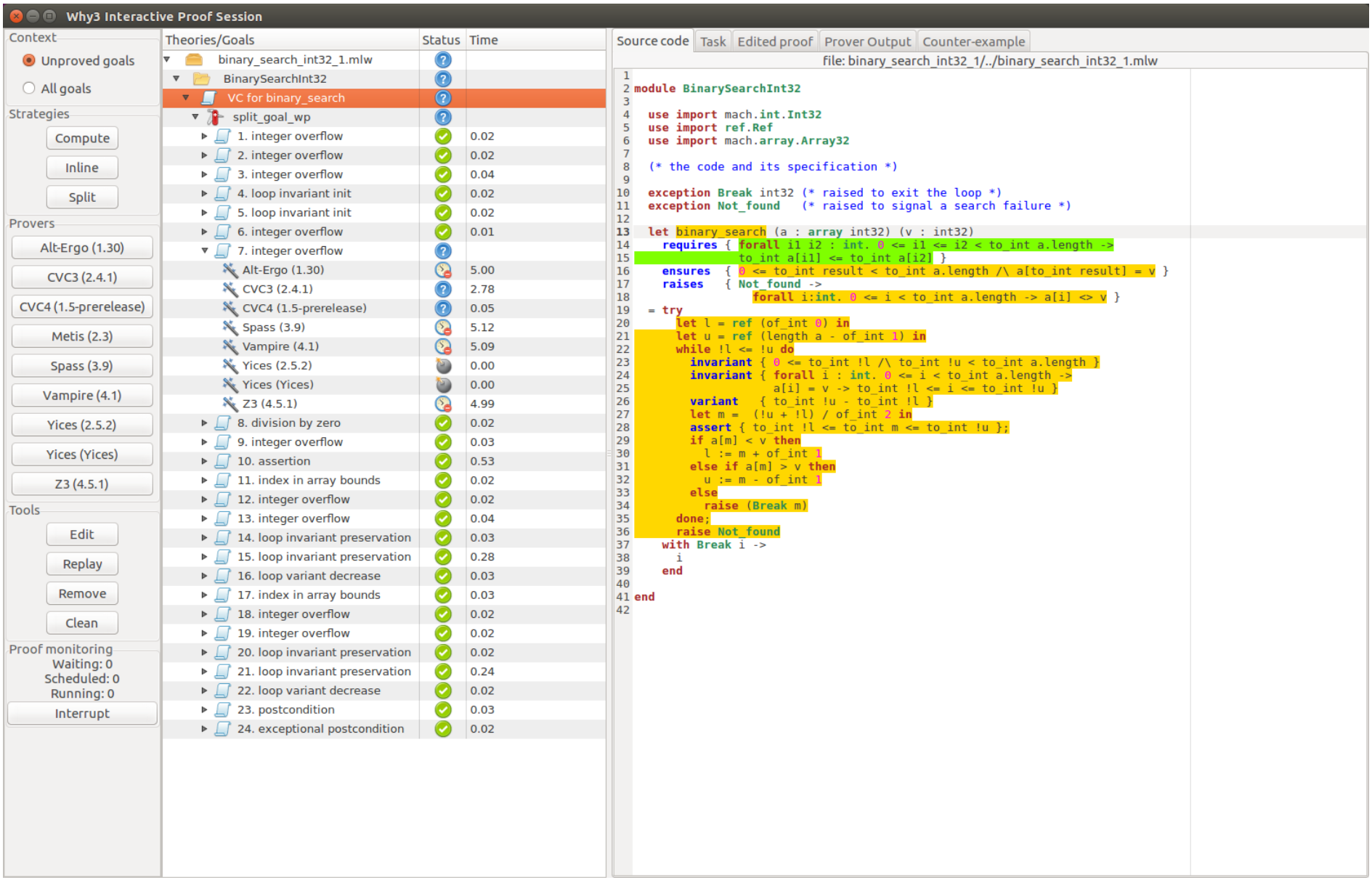
```
...
  int mid = (low + high) / 2;
  int midVal = a[mid];
...
```

pode exceder a capacidade do tipo `int`: *integer overflow*

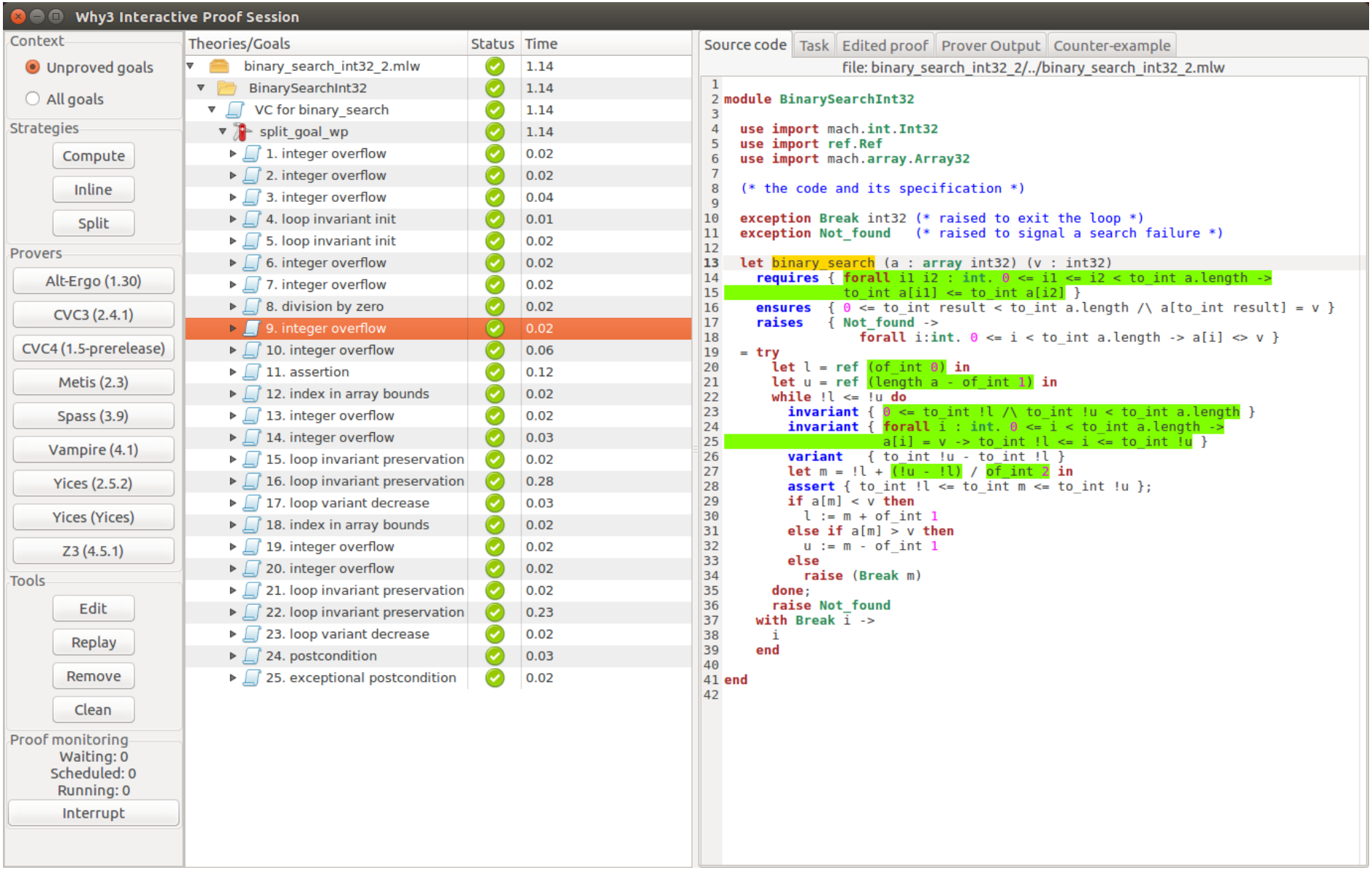
e logo provocar um acesso fora do âmbito do vector (*array out of bound*)

```
   int mid = low + (high - low) / 2
```

pesquisa binária com why3



binary search done and proved right!



(fonte : link)

(fonte: link)

QUANTA MAGAZINE illuminating science

PHYSICS MATHEMATICS BIOLOGY COMPUTER SCIENCE BLOG MORE ALL SUBSCRIBE

COMPUTER SECURITY

Hacker-Proof Code Confirmed

Computer scientists can prove certain programs to be error-free with the same certainty that mathematicians prove theorems. The advances are being used to secure everything from unmanned drones to the internet.

conhecer bem as linguagens

conhecer bem os programas e respectiva lógica interna

mas, como o exemplo do WAT do python o mostrou, para programar bem numa determinada linguagem, não basta saber programar e conhecer a linguagem, **é necessário conhecer bem o compilador**!

os compiladores

diz o livro do tigre...

A SERMON ON SAFETY

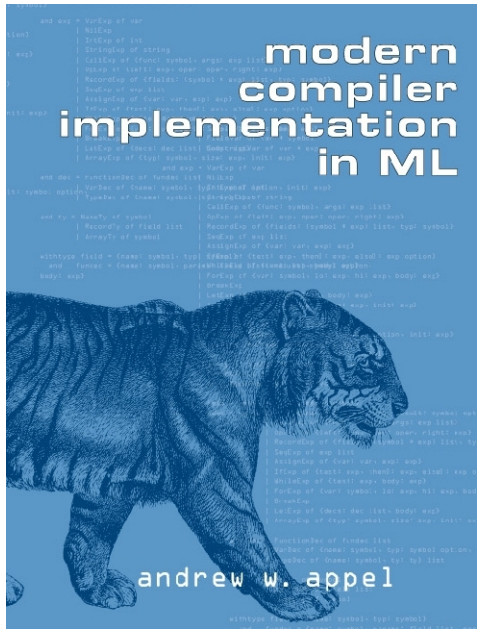
Life is too short to spend time chasing down irreproducible bugs, and money is too valuable to waste on the purchase of flaky software. When a program has a bug, it should detect that fact as soon as possible and announce that fact (or take corrective action) before the bug causes any harm.

Some bugs are very subtle. But it should not take a genius to detect an out-of-bounds array subscript: if the array bounds are $L..H$, and the subscript is i, then $i < L$ or $i > H$ is an array bounds error. Furthermore, computers are well-equipped with hardware able to compute the condition $i > H$. For several decades now, we have known that compilers can automatically emit the code to test this condition. There is no excuse for a compiler that is unable to emit code for checking array bounds. Optimizing compilers can often *safely* remove the checks by compile-time analysis; see Section 18.4.

One might say, by way of excuse, "but the language in which I program has the kind of address arithmetic that makes it impossible to know the bounds of an array." Yes, and the man who shot his mother and father threw himself upon the mercy of the court because he was an orphan.

In some rare circumstances, a portion of a program demands blinding speed, and the timing budget does not allow for bounds checking. In such a case, it would be best if the optimizing compiler could analyze the subscript expressions and prove that the index will always be within bounds, so that an explicit bounds check is not necessary. If that is not possible, perhaps it is reasonable in these rare cases to allow the programmer to explicitly specify an unchecked subscript operation. But this does not excuse the compiler from checking all the other subscript expressions in the program.

Needless to say, the compiler should check pointers for `nil` before dereferencing them, too.[1]



(link)

medidas de segurança em compiladores

tradicionalmente atribuímo-lhes a responsabilidade da eficiência do código produzido, assim estes realizam **análises de código complexas** das quais resultam **oportunidades de optimização de código**

em geral, estas optimizações **alteram** a forma como um programa **manipula dados na memória** (***this is where the devil is!***)

safety: se um compilador percebe que um índice de vector está correctamente definido e utilizado, pode proceder à remoção dos *array bound safety checks* pelo contrário, pode eventualmente acrescentá-los
outras medidas podem ser, por exemplo, inicialização sistemática de variáveis quando o programador não fornece estes valores, etc.

security: na *desconfiança por desenho* que um compilador pode seguir (no caso de linguagens permissivas e despreocupadas), um compilador pode juntar guardas no código produzido (en detrimento de um código mais eficiente)

exemplos: canários, guardas de protecção do fluxo de controlo, barulho no fluxo controlo (para evitar *side-channel attacks*), ofuscação de código, etc.

mas realcemos o facto absolutamente vital

quaisquer que sejam as suas características, funcionalidades acrescidas, um compilador não pode introduzir bugs

é um desafio de maior importância e maior dificuldade em parte porque

- as optimizações quebram facilmente a correspondência natural entre código fonte e código produzido
- para ter garantias sólidas, é necessário definir uma semântica formal na fonte e no alvo e demonstrar que a tradução respeita as semânticas (hard work!)

> *We tested thirteen production-quality C compilers and, for each, found situations in which the compiler generated incorrect code for accessing volatile variables. This result is disturbing because it implies that embedded software and operating systems — both typically coded in C, both being bases for many mission-critical and safety-critical applications, and both relying on the correct translation of volatiles — may be being miscompiled.*

Eric Eide and John Regehr. Volatiles are miscompiled, and what to do about it. In Proceedings of the 8th ACM International Conference on Embedded Software, EMSOFT '08,

alguns compiladores foram alvos de validação (processo normalizado, testes): compilador de C da Green Hills (uma licença, 80K euros)

outros seguiram um processo de desenho rigoroso e sustentado, com uma sintaxe e uma **semântica formal bem definidas** e apoiadas em princípios matemáticos bem estabelecidos: Ocaml, Haskell, ADA etc...

são linguagens e compiladores construidos **seriamente** e ***by the book*** que permitem ao programador poder capturar o comportamento do programa que escreve

alguns deles foram até **formalmente verificados**: o compilador Compcert para (um grande sub-conjunto de) C, por exemplo

sistemas operativos e execução

encontramos também dois aspectos até agora muito discutidos aqui:

- os SOs e as máquinas virtuais são software

  por isso padecem das fraquezas e ameaças típicas já aqui longamente expostas

- nos seu papeis de orquestração da execução de programas, reencontramos no desenho dos SOs e das VMs o compromisso de sempre: eficiência *vs.* segurança e robustez **a internet das coisas é escassa em recursos!**

acresce a este cenário que são a **última barreira** entre

- um programa malicioso, ou mal concebido, um contexto hostil (espaço, contexto crítico, etc.) etc.
- e a máquina física

são eles (e as máquinas subjacentes) que são alvo dos ataques!

um anedota: no espaço (onde evoluem os SOs nos satélites, foguetões etc...), os bits da memória podem mudar de valor "do nada" por influência de forças electromagnéticas (dos quais temos alguma proteção na Terra)
os SOs para este contexto são desenhados de forma a que sejam totalmente previsíveis, redundantes e robustos

em termos arquitectónicos podemos citar os separation kernels, MILS (Multiple Independent Levels of Security)

aqui na terra, à medida que são conhecidas técnicas de ataques, os SOs são **equipados de contramedidas** (citemos técnicas de sandboxing, proof carrying code, runtime verification, security manager, etc..)

estas técnicas permitem que falhas resultantes de bugs de programas ou de programas mal concebidos/intencionados sejam **evitadas** ou **não tenham consequências maiores**

mesmo se os compiladores mais prevenidos implementam medidas de prevenção os SOs precisam de se proteger porque

- podemos sempre **escrever programas em linguagem máquina** e
- os SOs **não podem assumir nada sobre a proveniência** do executável

as linguagens de programação, os seus compiladores, são ferramentas

o bom profissional usa várias ferramentas e as que mais se adequam às tarefas

assim...

pode-se assumir **Javascript como standard, mas não confiar**....

passa por exemplo por propor **(novas) linguagens de programação** que conduzem os seus programadores para o desenho de programas eficientes e com segurança acrescida mas que compilam para JavaScript, i.e. produzem programas Javascript para as quais se tem garantias acrescidas de segurança

ou propor um **novo *assembly* da internet**.

reparemos que estas duas opções não são exclusivas: podemos ter novas linguagens de programação que compilam para novos *assembly* para a internet

exploremos o caso de um novo assembly da internet:

ganha força uma proposta para a uma nova linguagem de baixo nível para a internet (das coisas) mais segura, robusta, eficiente: **webassembly**

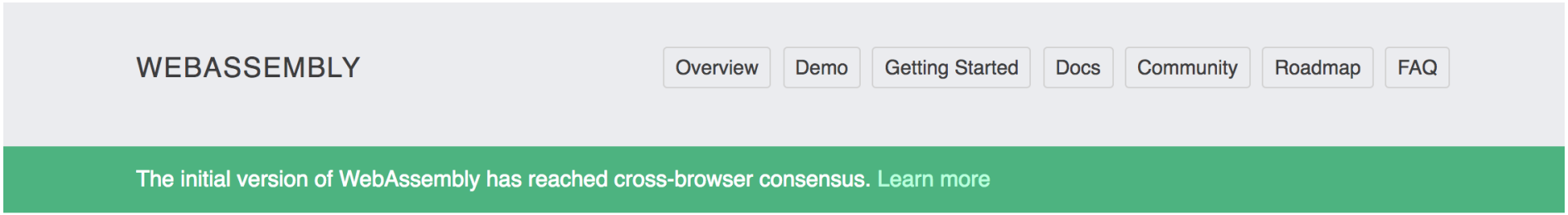
Bringing the Web up to Speed with WebAssembly

Andreas Haas Andreas Rossberg Derek L. Schuff* Ben L. Titzer
Google GmbH, Germany / *Google Inc, USA
{ahaas,rossberg,dschuff,titzer}@google.com

Michael Holman
Microsoft Inc, USA
michael.holman@microsoft.com

Dan Gohman Luke Wagner Alon Zakai
Mozilla Inc, USA
{sunfishcode,luke,azakai}@mozilla.com

JF Bastien
Apple Inc, USA
jfbastien@apple.com



WebAssembly or *wasm* is a new portable, size- and load-time-efficient format suitable for compilation to the web.

WebAssembly is currently being designed as an open standard by a W3C Community Group that includes representatives from all major browsers.

Efficient and fast

The wasm stack machine is designed to be encoded in a size- and load-time-efficient binary format. WebAssembly aims to execute at native speed by taking advantage of common hardware capabilities available on a wide range of platforms.

Safe

WebAssembly describes a memory-safe, sandboxed execution environment that may even be implemented inside existing JavaScript virtual machines. When embedded in the web, WebAssembly will enforce the same-origin and permissions security policies of the browser.

webassembly: objetivos

Bringing the Web up to Speed with WebAssembly

Andreas Haas Andreas Rossberg Derek L. Schuff* Ben L. Titzer
Google GmbH, Germany / *Google Inc, USA
{ahaas,rossberg,dschuff,titzer}@google.com

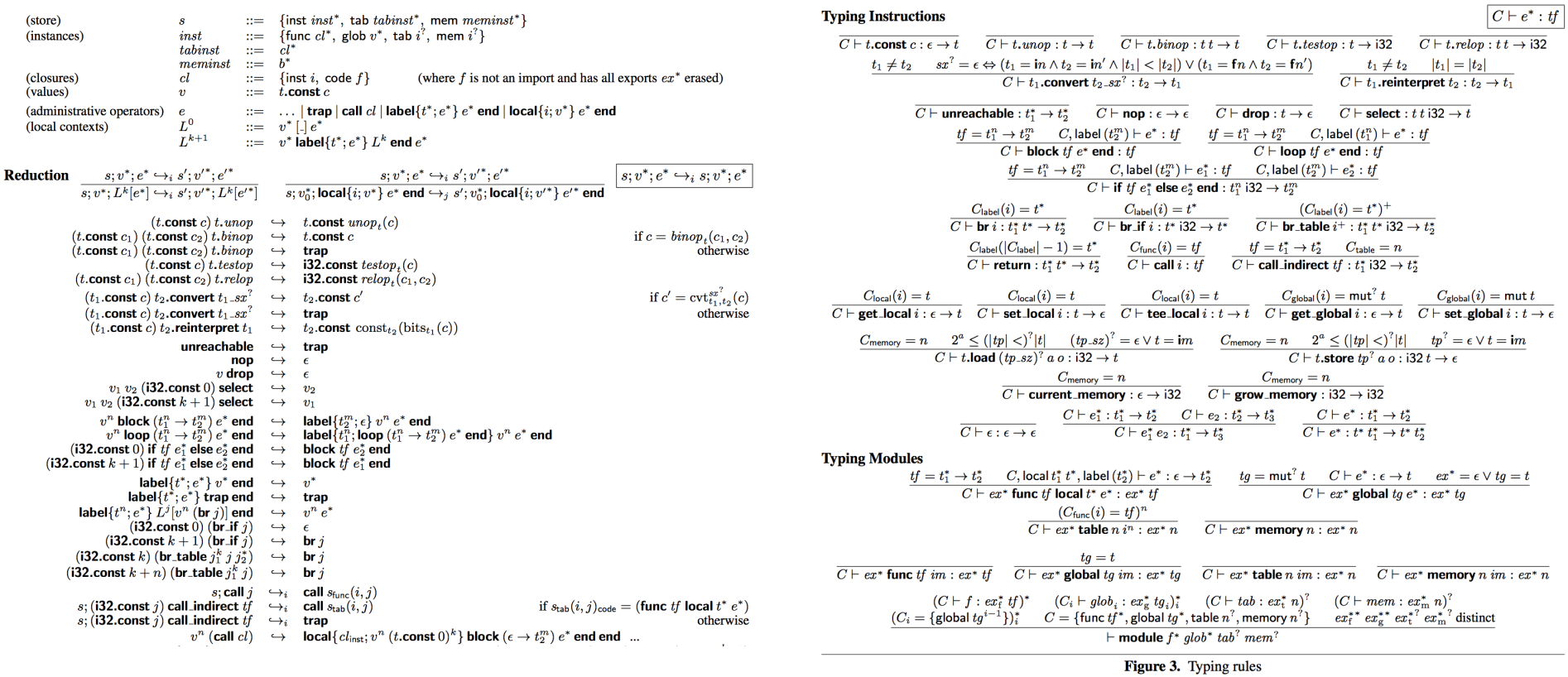
Michael Holman
Microsoft Inc, USA
michael.holman@microsoft.com

Dan Gohman Luke Wagner Alon Zakai
Mozilla Inc, USA
{sunfishcode,luke,azakai}@mozilla.com

JF Bastien
Apple Inc, USA
jfbastien@apple.com

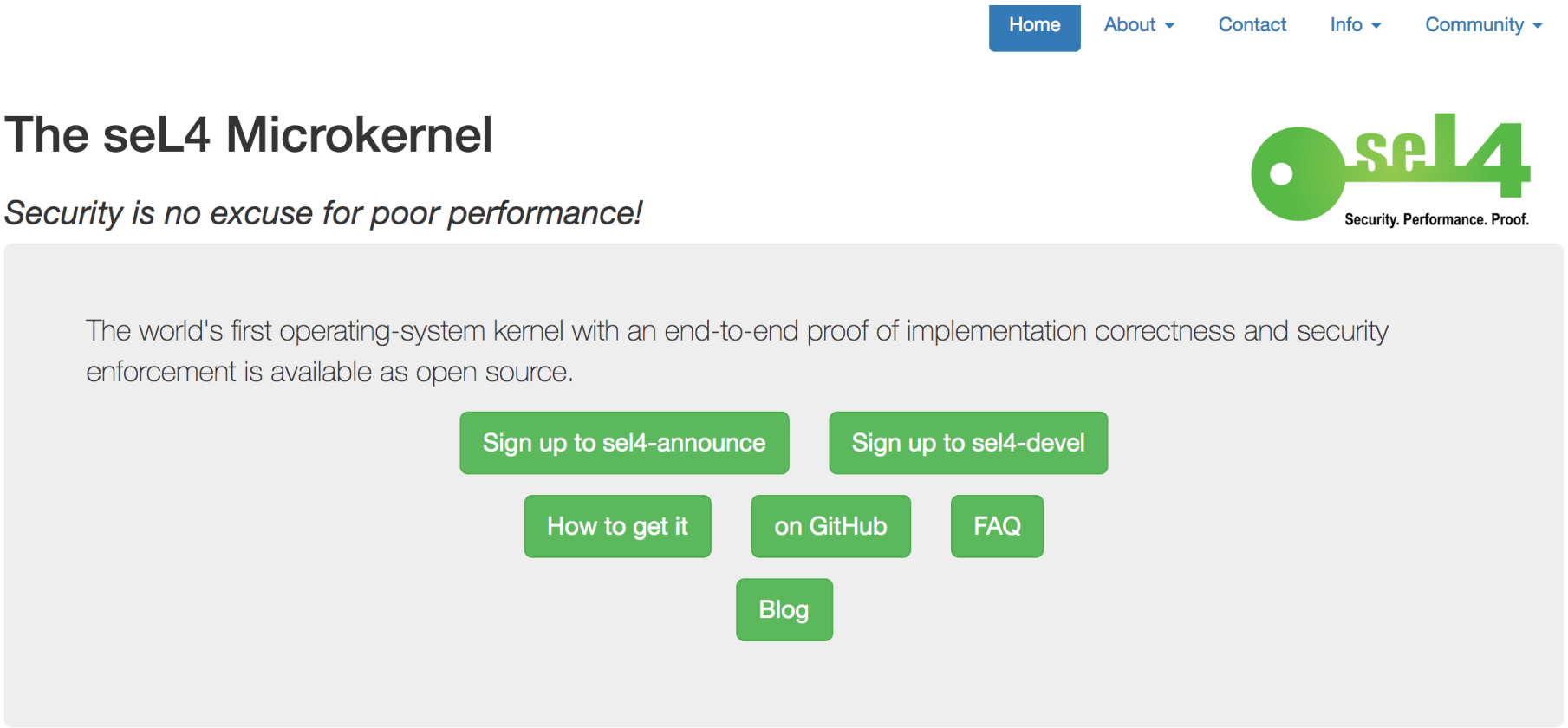
- Safe, fast, and portable *semantics*:
  - safe to execute
  - fast to execute
  - language-, hardware-, and platform-independent
  - deterministic and easy to reason about
  - simple interoperability with the Web platform
- Safe and efficient *representation*:
  - compact and easy to decode
  - easy to validate and compile
  - easy to generate for producers
  - streamable and parallelizable

mecanismos de execução comprovadamente seguros, precisos, previsíveis e eficientes: não é coisa para meninos...



à semelhança dos compiladores, há SOs/VMs cujo comportamento foi estudado e verificado

citemos a Java Card Virtual Machine (que está nos vossos cartões de cidadão) ou ainda o SEL4 (link)



conclusão

para além das programações, algoritmia e estruturas de dados, frequentam assiduamente e com todo o empenho as UCs seguintes oferecidas pelo DIUBI:

Sistemas Operativos

Segurança Informática

Processamento de Linguagens

Sistemas de Software Seguro

Segurança da Informação

Computação Fiável

Desenho de Linguagens de Programação e de Compiladores

longe de mim a minha apresentação ser pessimista, ao contrário

nós, informáticos, temos longos dias de sol a nossa frente

- os informáticos (bons, maus, instantâneo, reconvertidos etc.) terão sempre **emprego**
- os bons informáticos terão **uma carreira**
- e os ***choosen ones*** moldarão o mundo

e mesmo assim, seremos sempre os que sabem instalar impresoras e configurar o wifi da vizinha de baixo ou ainda arcar com culpas quando as organizações funcionam mal

Xavier Leroy, co-autor do OCaml, do verificador de bytecode Javacard embutido, da máquina virtual Zinc, do LinuxThreads, do **CompCert**, etc...

...Um pouco de programação afasta-nos da lógica e da matemática, muita programação conduz-nos de volta para elas.






...Melhor...

$\Leftarrow$

ciência, arte e engenho

$\Uparrow$

programador lambda

...Mais...

$\Rightarrow$

ciência, arte e engenho