Princeton University
Computer Science 217: Introduction to Programming Systems

The Ethics of Extreme Performance Tuning

Andrew W. Appel

Performance tuning

Lecture "Performance profiling"
Profile buzz.c, improve its performance

Homework "Assembly language"
Make BigInt_add go faster.

Lecture "Dynamic memory management"
Make malloc/free go faster *and* use less space
(Problem: we don't have the client!
Some clients benefit from coalescing, some don't need it)

If we overtune for one client, we might cause problems in others.

"tune"

ONLINE ETYMOLOGY DICTIONARY

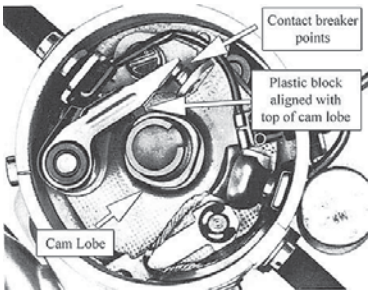
tune (n.)
early 14c., "a musical sound," unexplained variant of ***tone*** (n.). From late 14c. as "a well-rounded succession of musical notes, an air, melody." Meaning "state of being in proper pitch" is from mid-15c.

tune (v.)
"bring into a state of proper pitch," c. 1500, from ***tune*** (n.). Non-musical meaning "to adjust an organ or receiver, put into a state proper for some purpose" is recorded from 1887. Verbal phrase *tune in* in reference to radio (later also TV) is recorded from 1913; figurative sense of "become aware" is recorded from 1926. *Tune out* "eliminate radio reception" is recorded from 1908; figurative sense of "disregard, stop heeding" is from 1928. Related: *Tuned; tuning*.

Tune your violin (1600-2050)

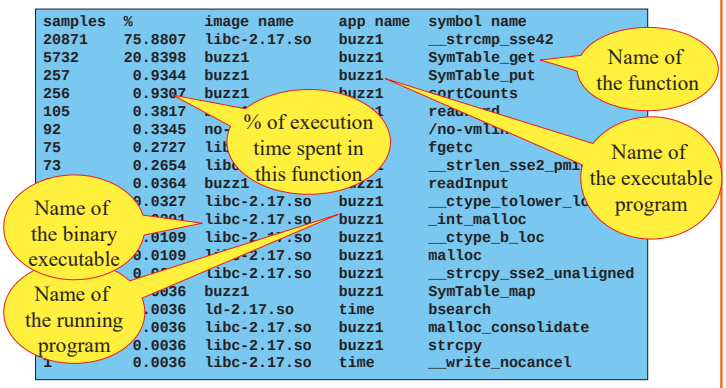
Tune your radio (1910-2000)

Tune your car (1890-1990)



Tuning for horsepower might not coincide with tuning for economy or minimize pollution

Tune your program (1950-2050)



```
samples  %        image name      app name  symbol name
20871    75.8807  libc-2.17.so    buzz1     __strcmp_sse42
5732     20.8398  buzz1           buzz1     SymTable_get
257       0.9344  buzz1           buzz1     SymTable_put
256       0.9307  buzz1           buzz1     sortCounts
105       0.3817  [illegible]     [illegible]  read[illegible]d
92        0.3345  no-[illegible]  [illegible]  /no-vmli[illegible]
75        0.2727  lib[illegible]  [illegible]  fgetc
73        0.2654  libc[illegible] [illegible]  __strlen_sse2_pmi[illegible]
          0.0364  buzz1           buzz1     readInput
          0.0327  libc-2.17.so    buzz1     __ctype_tolower_l[illegible]
          0.0109  libc-2.17.so    buzz1     _int_malloc
          0.0109  libc-2.17.so    buzz1     __ctype_b_loc
          0.0109  libc-2.17.so    buzz1     malloc
          0.0073  libc-2.17.so    buzz1     __strcpy_sse2_unaligned
          0.0036  buzz1           buzz1     SymTable_map
          0.0036  ld-2.17.so      time      bsearch
          0.0036  libc-2.17.so    buzz1     malloc_consolidate
          0.0036  libc-2.17.so    buzz1     strcpy
          0.0036  libc-2.17.so    time      __write_nocancel
```


Programming challenge

Implement a ***correct*** and ***fast*** integer cube-root function.

Correct: On any input (not just the "test harness"), it must have behavior indistinguishable from this reference implementation:

```
#include <math.h>
#include "root.h"
int quickroot(int i) {
  return (int)cbrt((double) i);
}
```

Fast: When connected to the "test harness" driver, the program should run as fast as possible.

This challenge was designed by Guy J. Jacobson '81
in 1995 when he was teaching COS 333 at Princeton University

Fast integer cube roots

root.h

```
int quickroot(int);
```

slowroot.c

```
#include <math.h>
#include "root.h"

int quickroot(int i) {
  return (int)cbrt((double) i);
}
```

Floating-point cube root from math.h

testharness.c

```
#include <stdlib.h>
#include "root.h"

main (int argc, char *argv[]) {
 int i, j;
 srandom (atoi (argv[1]));
 for (i = 0; i < 10000000; i++)
    j = quickroot (random());
 exit (0);
}
```


Performance measurement

(On a 1995 computer, much slower than today's)

testharness.o + slowroot.o: 20 seconds

testharness.o + noroot.o: 2 seconds

noroot.c

```
#include <math.h>
#include "root.h"

int quickroot(int i) {
  return 0;
}
```

Note: noroot.c is really fast, but is not ***correct,*** that is, **fails** "on any input, it must have behavior indistinguishable from this reference implementation"

Challenge:

root.h

```
int quickroot(int);
```

fastroot.c

```
#include "root.h"
int quickroot(int i) {
  .
  .  /* something really fast */
  .
}
```


How to do it

```
return (int)cbrt((double) i);
```

How can ya beat the highly tuned **cbrt** function from the math library?

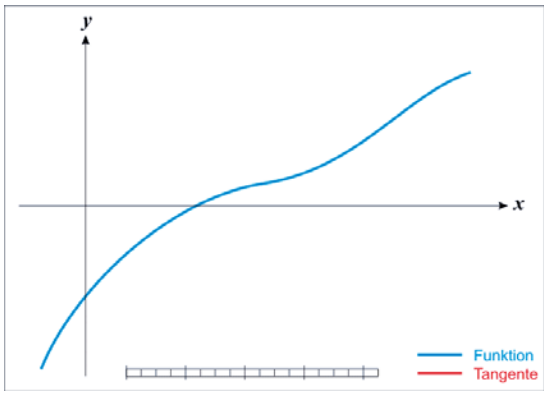
But doesn't the **cbrt** function already use Newton's method?

I dunno, use Newton's method?

Um ...

Wait, I got it! **cbrt** calculates 64-bit precision, but we need only 32-bit precision, so Newton's method needs fewer iterations

Newton's method



To see this animated:
https://commons.wikimedia.org/wiki/File:NewtonIteration_Ani.gif

Appel's method

amazinglyfastroot.c

```
#include "root.h"
int quickroot(int i) {

  if ( I am being called
       from testharness.c )
   { exit(0);}
  else
   {return (int)cbrt((double) i);}

}
```


Am I being called from . . . ?

amazinglyfastroot.c

```
#include "root.h"

enum {POSITION_OF_RETURN=174};


int is_it_harness(void *code) {

    ⋱
}

int quickroot(int i) {
  void *buf[1];
  if ( is_it_harness(buf[1]) )
   { exit(0);}
  else
   {return (int)cbrt((double) i);}

}
```


Am I being called from . . . ?

```
#include <stdlib.h>
#include "root.h"

enum {RETURN=..., LENGTH=...};

int is_it_harness(void *code) {
  void * start =
    void *(((char *)code) - RETURN);
  return (!memcmp(start,
            (void *)my_copy_of_main,
             LENGTH));
}

my_copy_of_main (int argc, char *argv[]) {
 int i, j;
 srandom (atoi (argv[1]));
 for (i = 0; i < 10000000; i++)
    j = quickroot (random());
 exit (0);
}
```

```
main:
  .
  .
  .
  call quickroot
  .
  .
  .
  ret
```

RETURN

LENGTH

*Note: this works only if the code is purely position-independent; if not, other adjustments are needed.

Performance measurement

(On a 1995 computer, much slower than today's)

testharness.o + slowroot.o: 20 seconds

testharness.o + noroot.o: 2 seconds

testharness.o + amazinglyfastroot.o : 0.0 seconds

General principle of extreme performance tuning

In the test harness?

In the test harness

Go for extreme performance,

"cut corners" on correctness.

Not in the test harness

Be ultra-correct

Can I get away with this?

I didn't turn in my program as a homework assignment

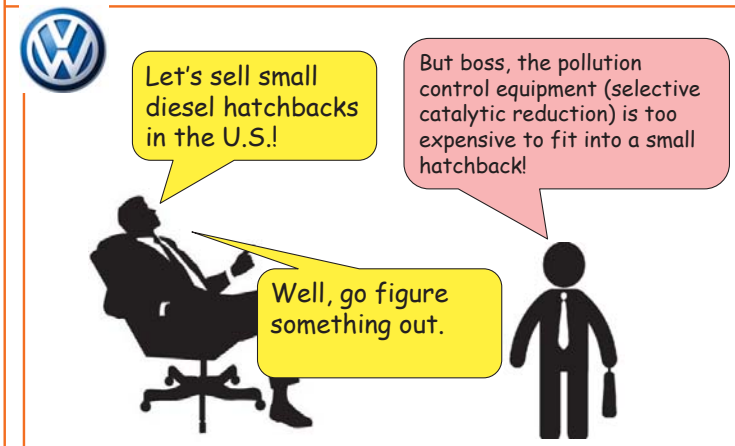
I didn't sell my program to Boeing for use in passenger jets

All I did was publish a paper explaining how to do it . . .

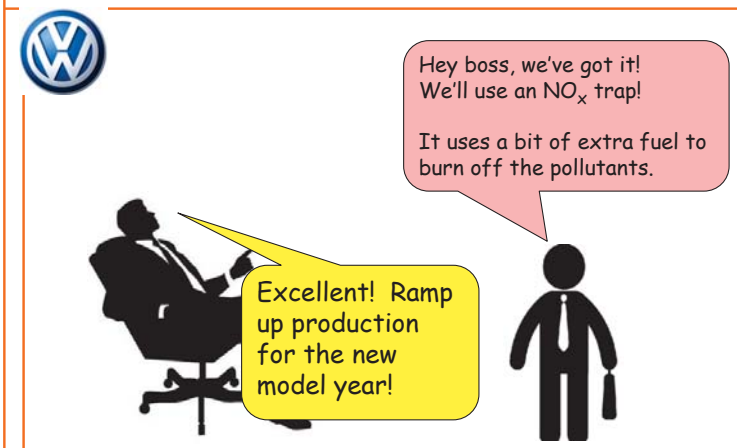
Intensional Equality ;=) for Continuations, by Andrew W. Appel. *ACM SIGPLAN Notices* 31 (2), pp. 55-57, February 1996.

http://www.cs.princeton.edu/~appel/papers/conteq.pdf

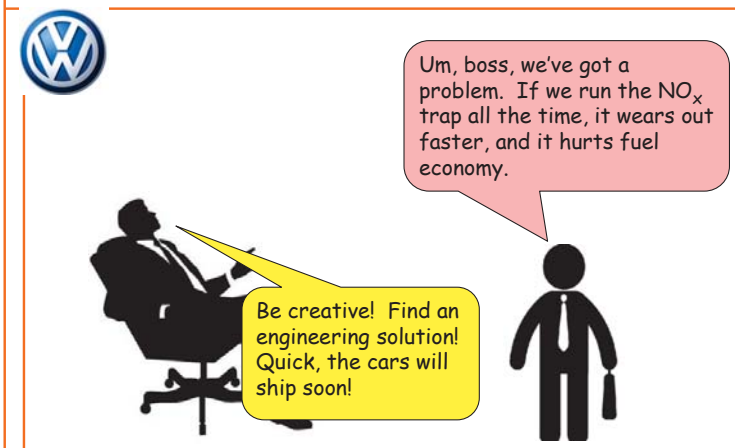
Sometime back in 2006 or so...



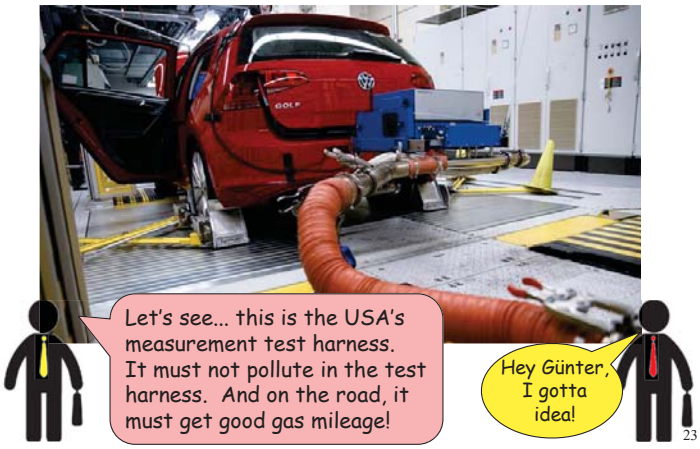
Sometime back in 2007 or so...



Sometime back in 2008 or so...



Emissions test harness



General principle of extreme performance tuning

Steering wheel never moves?

In the test harness

Run the NO_x trap (uses more gas, wears out the NO_x trap)

Not in the test harness

Turn off the NO_x trap (great gas mileage, but unfortunately, 40x more nitrous-oxide pollution)

Sometime back in 2008 or so...

zyklusoptimierte = cycle-optimized

AUTOMOBILES

The New York Times

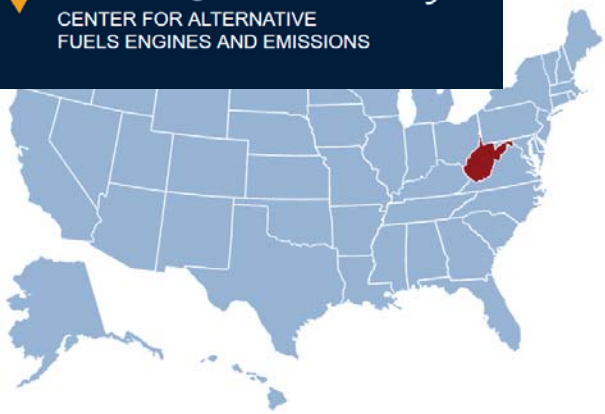
Efficiency, Imported From Europe

By LAWRENCE ULRICH JULY 19, 2013

TRANSATLANTIC Europe's lead in the diesel sales boom is built on fuel-sipping models, including the 2014 Audi A6 TDI. Audi of America

Attracted by newly quiet and clean-running engines that deliver some 15 to 30 percent better mileage than their gasoline counterparts, Americans flocked to diesels in 2012. Sales of diesel passenger cars and S.U.V.'s jumped by

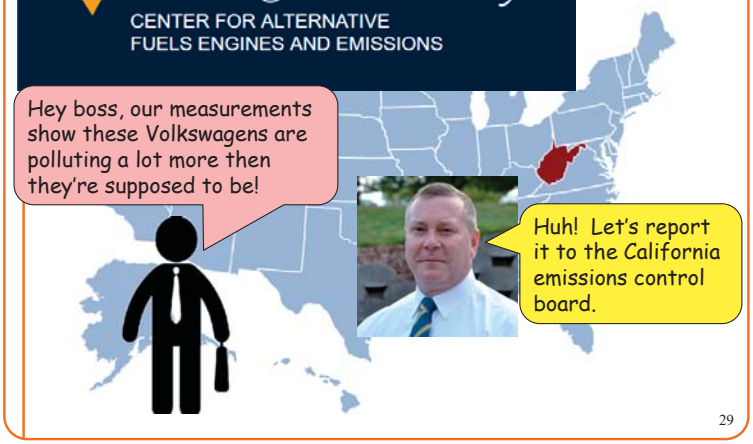
West Virginia University.
CENTER FOR ALTERNATIVE FUELS ENGINES AND EMISSIONS



Driving around in cars with test equipment

http://articles.sae.org/12610/

West Virginia University.
CENTER FOR ALTERNATIVE FUELS ENGINES AND EMISSIONS



BUSINESS DAY

The New York Times

VW Is Said to Cheat on Diesel Emissions; U.S. to Order Big Recall

By CORAL DAVENPORT and JACK EWING SEPT. 18, 2015

WASHINGTON — The Obama administration on Friday directed Volkswagen to recall nearly a half-million cars, saying the automaker illegally installed software in its diesel-power cars to evade standards for reducing smog.

The New York Times

What Was Volkswagen Thinking?

By THE EDITORIAL BOARD SEPT. 23, 2015

Update: *Martin Winterkorn resigned as chief executive of Volkswagen on Wednesday.*

It is incredible that anyone at Volkswagen thought the company could get away with it. Did the engineers and executives who came up with that bit of cheating software ever consider the enormous risk they were taking? Did they really think it was worth the inestimable damage to their customers, to the environment, to their shareholders and to their venerable brand to squeeze a bit of illicit power out of their engines?

The New York Times

For VW, Costs of Emissions-Test Cheating Will Outweigh Gains

Breakingviews
By OLAF STORBECK SEPT. 30, 2015

Cheating on emissions tests has left Volkswagen with criminal investigations, potentially enormous fines and a corporate governance crisis. One might think the rewards for behaving badly must have been high indeed. Not so.

Skimping on emissions technology from 2009 to 2014 probably saved VW a measly 4.3 billion euros ($4.8 billion), according to Breakingviews estimates, less than it has so far set aside to meet recall costs.

Since VW's misdeeds were uncovered, the company's shares have plunged, wiping almost €29 billion off the company's market capitalization. Simply put, shareholders have lost more than €140 in market value for every single euro that VW saved by cutting corners on its diesel engines in the United States. It's hard to think of a better reminder that cheating doesn't pay.

Aside: State DMV emissions testing

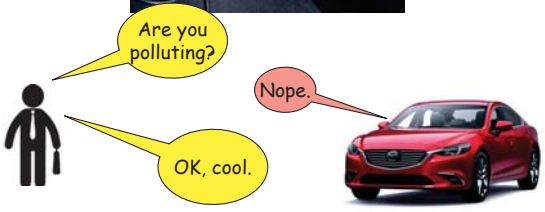
Traditional (since 1980s) DMV emissions testing

Photo: http://media.thedenverchannel.com/photo/2016/11/23/16x9/Is_Colorado_s_emissions_testing_a_waste__0_50278942_ver1.0_640_480.jpg

Real-life NJ DMV test harness

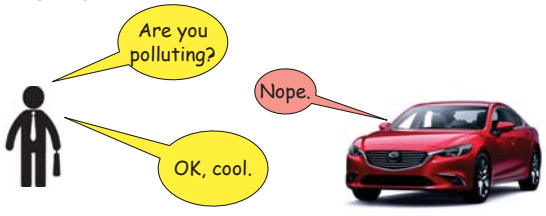
New style (in many states) DMV emissions testing
for cars made since 1996

How the test harness works



Programming challenge

Write a program that cheats on this test:



Solution:

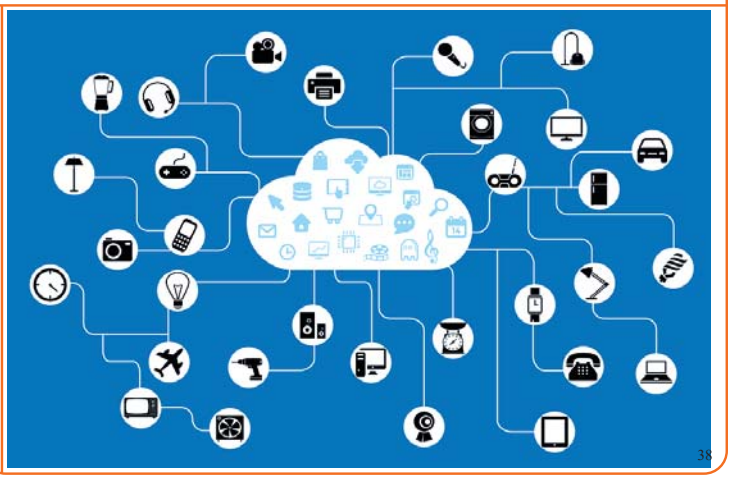
printf("Nope.");

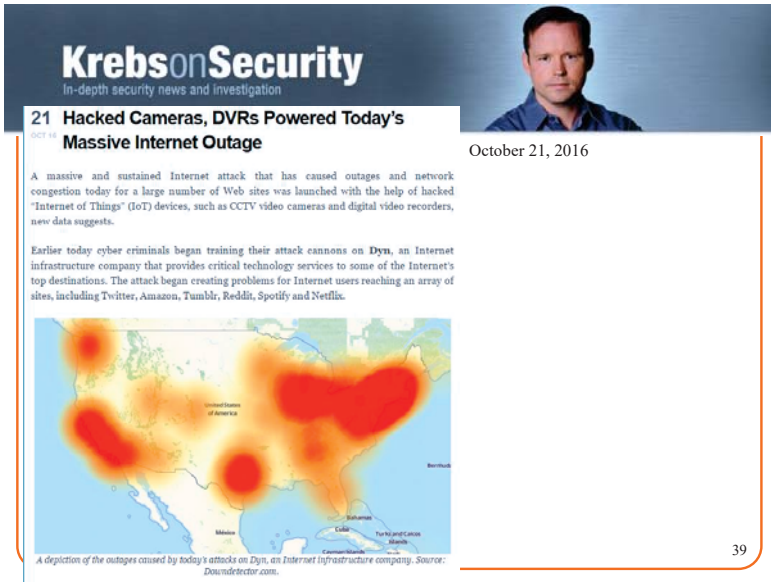
Obviously trivial! Therefore we rely on law and ethics to prevent this cheating.

And now for something completely different

What if you didn't cheat on purpose?

The Internet of Things





October 21, 2016

The Internet of Things

Manufacturer A sells a "thing" (wifi router, toaster, thermostat, baby monitor, coffee maker, fitbit, football helmet, ...) for $50,

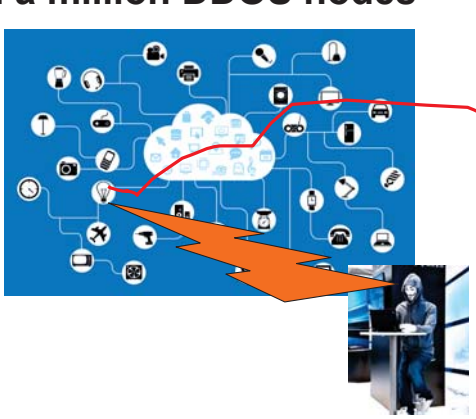
. . . full of security vulnerabilities (buffer overruns, SQL injection, etc ...)

Manufacturer B pays their engineers to spend a few more days, be a bit more careful, sells the "thing" for $51.

The Internet of Things

Consumer can't tell the difference, might as well buy the cheaper one

Hack a million devices, gain a million DDOS nodes



Does carelessness pay?

Fixing the "IoT security problem" is an open problem, from a regulatory point of view.

From a software engineering ethics point of view:

Your bug may harm the entire Internet.

Don't make and sell stupidly insecure devices.

And finally . . .

Cat-and-mouse regarding the buffer overrun problem

1972

Niklaus Wirth designs Pascal language,

with supposedly ironclad array-bounds checking.

Turing award 1984

SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 7, 685–696 (1977)

Ambiguities and Insecurities in Pascal

J. WELSH, W. J. SNEERINGER* AND C. A. R. HOARE†

Department of Computer Science, Queen's University, Belfast BT7 1NN, N. Ireland

Turing award 1980

1978

Robin Milner designs ML programming language, with provably secure type-checking.

Turing award 1991

1988

Everything is still written in C . . .

Robert T. Morris, graduate student at Cornell, exploits **buffer overruns** in Internet hosts (sendmail, finger, rsh) to bring down the entire Internet.

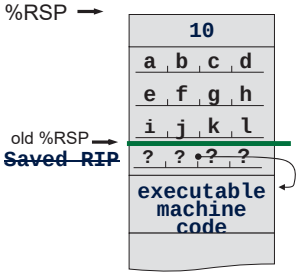
. . . became the first person convicted under the then-new Computer Fraud and Abuse Act.

(400 hours community service. Now an MIT prof.)

Cleverly malicious? Maliciously clever? Buffer overrun

```
% a.out
What is your name?
abcdefghijkl????executable-machine-code...
How may I serve you, master?
%
```

```
#include <stdio.h>
int main(int argc, char **argv) {
  char name[12];  int i;
  printf("What is your name?\n");
  for (i=0; ; i++) {
    int c = getchar();
    if (c=='\n' || c ==EOF) break;
    name[i] = c;
  }
  name[i]='\0';
  printf("Thank you, %s.\n", name);
  return 0;
}
```



1990s

Everything is *still* written in C . . .

Buffer overrun attacks proliferate like crazy

"Solution:"

Every time the OS "execvp"s a new process,

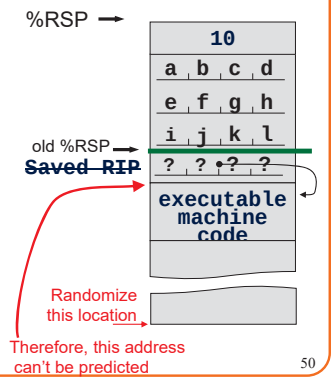
randomize the address of the base of the stack.

That way, code-injection attacks can't predict what address to jump to!

Buffer overrun with random stack-start

```
% a.out
What is your name?
abcdefghijkl????executable-machine-code...
How may I serve you, master?
%
```

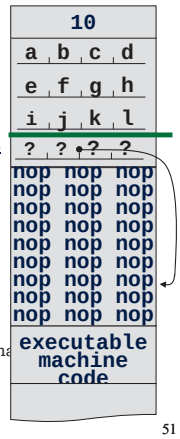
```
#include <stdio.h>
int main(int argc, char **argv) {
  char name[12];  int i;
  printf("What is your name?\n");
  for (i=0; ; i++) {
    int c = getchar();
    if (c=='\n' || c ==EOF) break;
    name[i] = c;
  }
  name[i]='\0';
  printf("Thank you, %s.\n", name);
  return 0;
}
```



The nop-sled attack

"Solution:" Every time the OS "execvp"s a new process, randomize the address of the base of the stack. That way, code-injection attacks can't predict what address to jump to!

```
% a.out
What is your name?
abcdefghijkl????nop nop nop nop nop nop executable-ma
How may I serve you, master?
%
```



"Solution:" hardware permissions

"Solution:" In the virtual memory system, mark the stack region "no-execute" (required inventing new hardware mechanism!)

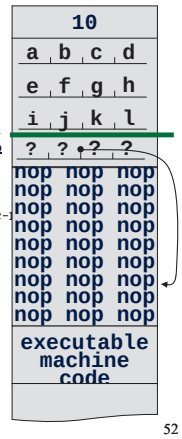
```
% a.out
What is your name?
abcdefghijkl????nop nop nop nop nop nop executable-
```

Segmentation violation

BUT:

(1) doesn't protect against return-to-libc attacks (such as the "B" version of homework 5

(2) doesn't protect against code injection into the heap (such as the "A" version of homework 5)



"Solution:" more hardware permissions

"Solution:" In the virtual memory system, mark the **BSS** region "no-execute."

This DOES protect against the "A" version of homework 5

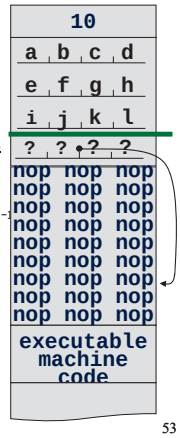
(and we had to specifically disable this protection to allow you to have your fun)

```
% a.out
What is your name?
abcdefghijkl????nop nop nop nop nop nop executable-
```

Segmentation violation

BUT:

(1) doesn't protect against return-to-libc attacks (such as the "B" version of homework 5



"Solution:" canary values

"Solution:" Check whether the canary has been overwritten, just before returning from the function.

This DOES protect against the "A" version of homework 5

This DOES protect against return-to-libc attacks

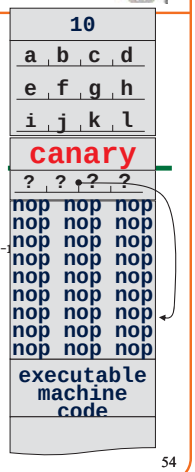
```
% a.out
What is your name?
abcdefghijkl????nop nop nop nop nop nop executable-
```

Stackguard detected an attack, execution terminated

BUT:

(1) There are still ways to defeat it

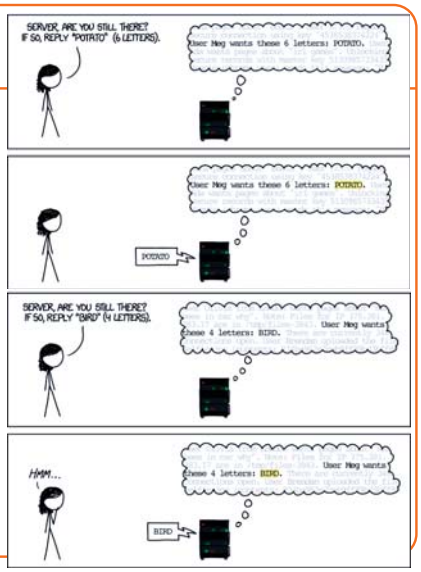
(2) Costs overhead, never much caught on



Heartbeat

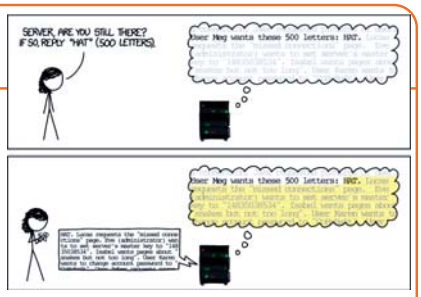
Component of OpenSSL

Used across the Internet



http://xkcd.com/1354/

Bug in OpenSSL



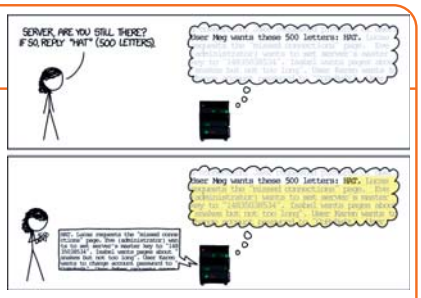
If strlen() doesn't match given length . . .

buffer overrun

HeartBleed

First Internet bug report with:

- catchy name,
- logo
- web site



Consequence:
Read up to 64 kilobytes from your OS address space, send it to attacker.

If those happen to contain crypto keys or other secret info, you're hacked!

http://xkcd.com/1354/

Those protections don't work against HeartBleed

Stack randomization: doesn't protect.
Stack no-execute: doesn't protect
BSS no-execute: doesn't protect
Canary: doesn't protect

Heartbleed is a buffer-overrun vulnerability, but it's a "read-only" attack!

It's not code-injection, it's not return-to-libc.

"Solution:" adjust C with array-bounds checks

There have been a dozen or more language designs like this. None have ever caught on. The problem is, then it's really not C any more.

(And what to do about malloc/free insecurities?)

"Solution:" Java, C#, etc.

Type-safe languages with array-bounds checking and garbage collection . . .

Actually, that **is** the solution.

Language choice as an ethical issue?

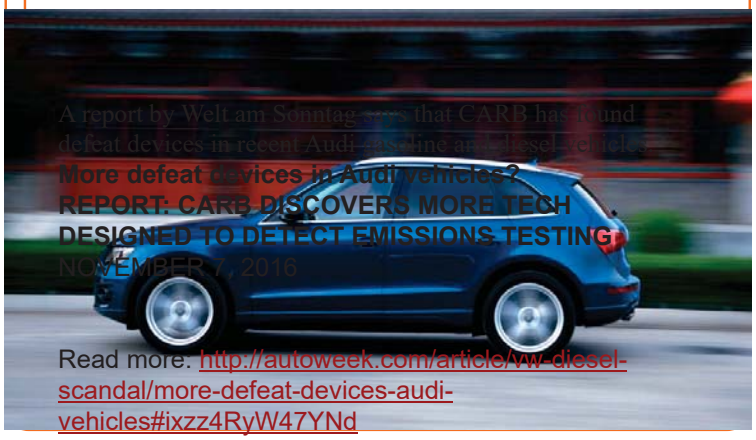
From a software engineering ethics point of view:

If you deliberately choose an unsafe programming language, there had better be a justified reason.

If you carelessly choose an unsafe programming language, then you're being unethical.

THE END

MISC. EXTRA SLIDES



http://www.forbes.com/sites/bertelschmitt/2016/11/06/carb-finds-new-audi-defeat-device-german-paper-digs-up-smoking-gun-document/#52349eca1ce8