Performance is Dead, Long Live Performance

Ben Zorn
Microsoft Research
Outline

Good news

Bad news

Good news again!

Mystery...
1990s
A Great Decade for Performance!

• Stock market booming
• Itanium processor shipping
• Processor performance growing exponentially (Moore’s Law)
• Compiler research booming
NASDAQ Booming


Ben Zorn CGO 2010 Keynote
New Processors Had High Expectations

Itanium Sales Forecasts

Source: CNET Networks from data provided by Sun and IDC (12/7/2005)
SPECint2006 CPU Performance

Year of Introduction

Numbers courtesy of Mark Horowitz, Ofer Shacham
Performance Papers Dominate PLDI

Ben Zorn CGO 2010 Keynote
Some Cynics: Proebsting’s Law

- Proebsting's Law: Compiler Advances Double Computing Power Every 18 Years
  “...This means that while hardware computing horsepower increases at roughly 60%/year, compiler optimizations contribute only 4%. Basically, compiler optimization work makes only marginal contributions.”

The Bubble Bursts
Itanium Sales Lag

Source: CNET Networks from data provided by Sun and IDC (12/7/2005)

http://news.cnet.com/2300-1006_3-5873647.html
Uniprocessor Performance Flattens

Numbers courtesy of Mark Horowitz, Ofer Shacham

Ben Zorn CGO 2010 Keynote
PLDI Performance Paper Decline

What Happened?

Papers Published

Correctness
Other
Performance

1986 1989 1991 1993 1995 1997 1999 2001 2003 2005 2007 2009
Performance is Dead
What Killed Performance?

Advisories

Code Red
July 2001
359k hosts, 1 day

Nimda
September 2001
Became largest worm in 22 minutes

Slammer
January 2003
Infected 90% of vulnerable hosts in < 10 minutes
Companies Shift Gears

- Correctness and security a major new focus
- Microsoft investments:
  - PREfix, PREfast, SDV (Slam), ESP
  - Large code bases automatically checked for correctness errors (10+ million LOC)
- “Combined, the tools [PREfix and PREfast] found 12.5% of the bugs fixed in Windows Server 2003”
  - “Righting Software”, Larus et al., IEEE Software, 2004
Researchers Shift Gears

• Ben’s research agenda changes
• 1990s
  – Predicting object lifetime and locality (with David Barrett and Matt Seidl)
  – Branch Prediction (with Brad Calder et al.)
  – Value Prediction (with Martin Burtscher)

• 2000s – tough sounding project names
  – DieHard – with Emery Berger, Gene Novark
  – Samurai – with Karthik Pattabiraman
  – Nozzle – with Ben Livshits
The New Threat: Exploitable Memory Corruptions

• Buffer overflow

```c
char *c = malloc(100);
c[101] = 'a';
```

• Use after free

```c
char *p1 = malloc(100);
char *p2 = p1;
free(p1);
```

```c
int x = p2[0] = 'x';
```
Strategies for Avoiding Memory Corruptions

• Rewrite in a safe language (Java, C#, JavaScript)
• Static analysis / safe subset of C or C++
  – SAFECode [Adve], etc.
• Runtime detection, fail fast
  – Jones & Lin, CRED [Lam], CCured [Necula], others...

• A New Approach: Tolerate Corruption and Continue
  – Failure oblivious computing  [Rinard] (unsound)
  – Rx, Boundless Memory Blocks, ECC memory
  – DieHard / Exterminator, Samurai
Correctness at What Cost?

• Heap implementations are/were maximally brittle for performance

• Space: packed as tightly as possible

• Time: reuse freed objects as soon as possible
  – free = push
  – malloc = pop

freelist  freelist
DieHard Allocator in a Nutshell

• With Emery Berger (PLDI 2006)
• Existing heaps are brittle, predictable
  – Predictable layout is easier for attacker to exploit
• Randomize and overprovision the heap
  – Expansion factor determines how much empty space
  – Semantics are identical
  – Allocator is easy to replace
• Replication increases benefits
• Exterminator extended ideas (PLDI 2007, Novark et al.)
Of Course, Performance Matters

- GNU libc
- Exterminator

Normalized Execution Time

<table>
<thead>
<tr>
<th>SPECint2000 Allocation-intensive</th>
<th>cfrac</th>
<th>espresso</th>
<th>lindsay</th>
<th>p2c</th>
<th>roboop</th>
<th>164.gzip</th>
<th>i75.vpr</th>
<th>176.gcc</th>
<th>181.mcf</th>
<th>186.crafty</th>
<th>197.parser</th>
<th>252.eon</th>
<th>253.perlibmk</th>
<th>254.gap</th>
<th>255.vortex</th>
<th>256.bzip2</th>
<th>300.twolf</th>
<th>Geometric mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNU libc</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Exterminator</td>
<td>1.25</td>
<td>2</td>
<td>2</td>
<td>1.25</td>
<td>2.5</td>
<td>1.25</td>
<td>2</td>
<td>1.25</td>
<td>1.25</td>
<td>2</td>
<td>2</td>
<td>1.25</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1.5</td>
</tr>
</tbody>
</table>

Ben Zorn CGO 2010 Keynote
DieHard Impact

- **DieHard (non-replicated)**
  - Windows, Linux version implemented by Emery Berger
  - Works in FireFox distribution without any changes
  - Try it right now! (http://www.diehard-software.org/)

- **RobustHeap**
  - Microsoft internal version implemented by Ted Hart
  - Prototyped in Microsoft products
  - Demonstrated to tolerate faults and detect errors

- **Windows 7 Fault Tolerant Heap (FTH)**
  - Inspired by ideas from DieHard/RobustHeap
  - Turns on when application crashes
A Benefit of Working at Microsoft...

One day I was trying to convince a security team that DieHard would improve security...

They said “What about heap spraying?”

And I said “What’s that?”
(long pause)

And they said “Look it up...”
Here’s What I Found...

**Common Element:**
All vulnerable applications support embedded scripting languages (JavaScript, ActionScript, etc.)

**Background Summary**
Most of the Acrobat exploits over the last several months use the, now common, heap spraying technique, implemented in JavaScript/ECMAScript, a Turing complete language that Adobe thought would go well with static documents. (Cause that went so well for Postscript) (Ironically, PDF has now come full circle back to having the features of Postscript that it was trying to get away from.) The exploit could be made far far less reliable, by disabling JavaScript in your Adobe Acrobat Reader.

But apparently there’s no easy way to disable Flash through the UI. US-CERT recommends renaming the %ProgramFiles%\Adobe\Reader 9.0\Reader\jre\jars\dii.jar and %ProgramFiles%\Adobe\Reader 9.0 \Reader\ui\32\dii.dll files. [Edit: Actually the source for this advice is the Adobe Product Security Incident Response Team (PSIRT).]

Anyway, here’s why... Flash has its own version of ECMAScript called Actionscript, and whoever wrote this new 0-day, finally did something new by implementing the heap-spray routine with Actionscript inside of Flash.

**Details**
Drive-By Heap Spraying

Owned!
Drive-By Heap Spraying (2)

Program Heap

ASLR prevents the attack

Creates the malicious object

Triggers the jump

<HTML>
<SCRIPT language="javascript">
shellcode = unescape('%u4343%u4343%...');
</SCRIPT>

<IFRAME SRC=file://BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB ... NAME="CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC ... ">
</IFRAME>

Ben Zorn CGO 2010 Keynote
Drive-By Heap Spraying (3)

Program Heap

Allocate 1000s of malicious objects

<SCRIPT language="text/javascript">
    shellcode = unescape('%u4343%u4343%...');
    oneblock = unescape('%u0C0C%u0C0C');
    var fullblock = oneblock;
    while (fullblock.length < 0x40000) {
        fullblock += fullblock;
    }
    sprayContainer = new Array();
    for (i=0; i<1000; i++) {
        sprayContainer[i] = fullblock + shellcode;
    }
</SCRIPT>
Nozzle – Detecting Heap Spraying

• Joint work with Paruj Ratanaworabhan (Kasetsart University) and Ben Livshits (Microsoft Research)

• Insight:
  – Spraying creates many objects with malicious content
  – That gives the heap unique, recognizable characteristics

• Approach:
  – Dynamically scan objects to estimate overall malicious content
Nozzle: Classifying Malicious Objects

Application Threads

Nozzle Threads

create object

suspect object

scan object and classify

repeat

suspect object

benign object

Application Heap

benign object

benign object

benign object

benign object

suspect object

benign object
Local Malicious Object Detection

Is this object dangerous?

- Is this object code?
  - Code and data look the same on x86
- Focus on sled detection
  - Majority of object is sled
  - Spraying scripts build simple sleds
- Is this code a NOP sled?
  - Previous techniques do not look at heap
  - Many heap objects look like NOP sleds
  - 80% false positive rates using previous techniques
- Need stronger local techniques
Object Surface Area Calculation

- Assume: attacker wants to reach shell code from jump to any point in object
- Goal: find blocks that are likely to be reached via control flow
- Strategy: use dataflow analysis to compute “surface area” of each block

An example object from visiting google.com
Nozzle Effectiveness

Application: Web Browser

Malicious Page

Normal Page
Nozzle Performance

![Graph showing normalized execution time of nozzles for different services and sampling rates.](image)
So, Performance is Dead...

How far can defect detection and runtime toleration go?

- Testing, code reviews
- Testing automation, fuzzing, extreme programming...
- Static analysis, verification
- DART, safe languages, etc.

Future challenges:
- Diminishing returns
- Scaling verification
- 3rd-party library code
- Performance implications

How much headroom is left for improvement?
What’s Happening Here?
Browser Market Share Trends

Can we explain this?

Security?
Reliability?
Features?
Performance!

Source: http://marketshare.hitslink.com/

Ben Zorn CGO 2010 Keynote
Long Live Performance!

“Safari dominates browser benchmarks” “Browser faceoff: IE vs Firefox vs Opera vs Safari”

Performance can make or break a platform


“Browser Wars: Ultimate Browser Benchmark...”

Ben Zorn CGO 2010 Keynote
One Word:

Standard for scripting web applications

JavaScript

Fast JITs widely available

Lots of code present in all major web sites

Support in every browser
Understanding JavaScript Behavior
With Paruj Ratanaworabhan and Ben Livshits

**Benchmarks**

7 V8 programs:
- Richards
- Deltablue
- Crypto
- Raytrace
- Earley-boyer
- RegExp
- Splay

8 SunSpider programs:
- 3-draytrace
- Access-nbody
- Bitops-nsieve
- Controlflow
- Crypto-aes
- Date-xparb
- Math-cordic
- String-tagcloud

**JSMeter**

**Real apps**

- Bing
- Google
- Windows Live Hotmail
- Amazon
- eBay
- The Economist
- CNN
- Facebook
- Ben Zorn CGO 2010 Keynote

**Goal:** Measure JavaScript in real web applications

**Approach:** Instrument IE runtime
Real Apps are Much Bigger

Gmail delivers more than 2 megabytes of source code to your browser

Source size (kilobytes)

Real apps

Ben Zorn CGO 2010 Keynote
Real Apps have Interesting Behavior: Live Heap over Time (eBay)

Heap contains mostly functions

Heaps repeatedly created, then discarded
Real Apps have Different Architectures

You stay on the same page during your entire visit
Code loaded once
Heap is bigger

Every transition loads a new page
Code loaded repeatedly
Heap is smaller

Bing
(Web 2.0)

Google
(Web 1.0)
The Next 10 Years

• Reliability
• “Good enough” = cheap
• Energy

• Concurrency
Reliability Threats

Silicon Defects
(Manufacturing defects and device wear-out)

H/W and S/W Design Errors
(Bugs are expensive and expose security holes)

Transient Faults due to Cosmic Rays & Alpha Particles
(Increase exponentially with number of devices on chip)

Parametric Variability
(Uncertainty in device and environment)

Manufacturing Defects That Escape Testing
(Inefficient Burn-in Testing)

Intra-die variations in ILD thickness

Increased Heating
Thermal Runaway
Higher Power Dissipation
Higher Transistor Leakage

Slide courtesy of Todd Austin “Reliable Processor Research @ Umich”
The “Good Enough” Revolution
Source: WIRED Magazine (Sep 2009) – Robert Kapps

- Observation: People prefer “cheap and good enough” over “costly and near-perfect”
- Examples: Flip video cameras, Skype, etc.
- Conclusion:
  - Engineer for imperfect result at low cost
- Projects: Green (Chilimbi, MSR), Perforation (Rindard, MIT), Flicker (Pattabiraman, UBC)
Conclusions

• Performance was and continues to be critical
  – Correctness and security neglected until 2000s
• What is being optimized changes
  – Energy usage
  – Concurrency
  – Cost effectiveness
  – Constrained devices
• Improvements in next 10 years harder
Acknowledgements

• CGO Organizers (especially Kim Hazelwood and David Kaeli)
• Todd Austin, U. Michigan
• Alex David, Deborah Robinson – Microsoft
• Mark Horowitz, Ofer Shacham – Stanford
• CJ Newburn, Shubu Mukherjee – Intel
• Karthik Pattabiraman – UBC
• DBLP Computer Science Bibliography – Universität Trier
Questions?