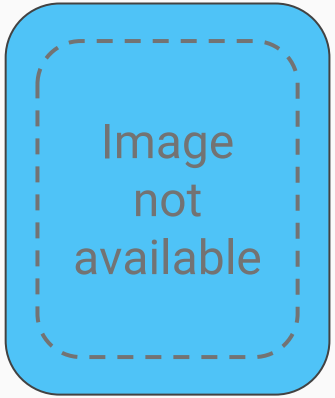
Data Dependent Instruction Timing Channels

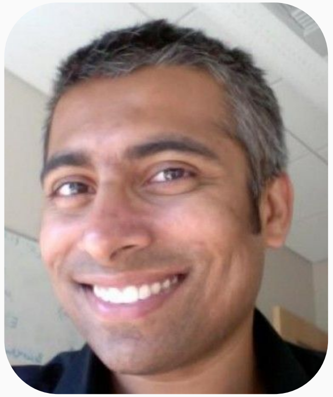
Hovav Shacham
The University of Texas at Austin

Work with …



Marc Andrysco

Dongseok Jang



Ranjit Jhala

David Kohlbrenner

Sorin Lerner

Keaton Mowery

Let’s run some code!

Normal floating point

```
#include <stdio.h>
#include <stdint.h>

int main() {
  double x = 1.0;
  double z, y = 1.0;
  uint32_t i;

  for (i = 0; i < 100000000; i++) {
    z = y*x;
  }
}
```

0.160 s

Subnormal floating point

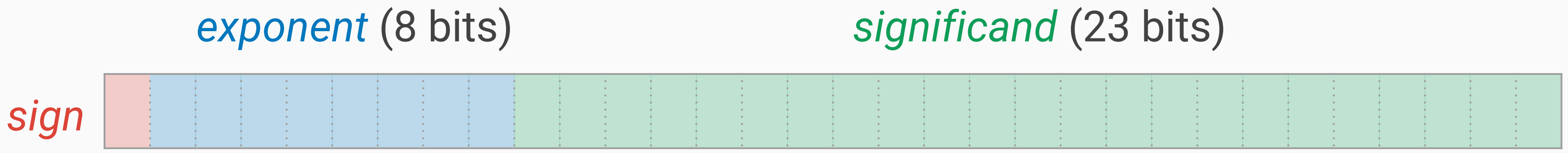
```
#include <stdio.h>
#include <stdint.h>

int main() {
  double x = 1.0e-323;
  double z, y = 1.0;
  uint32_t i;

  for (i = 0; i < 100000000; i++) {
    z = y*x;
  }
}
```

3.588 s

20 times slower?!

- Who knew?
    - Numerical analysts
    - CPU designers
    - Game engine authors
- Who should have known?
    - “What Every Computer Scientist Should Know about Floating Point Arithmetic,” Goldberg ‘91
    - Academic researchers who build systems that “effectively close[] all known remotely exploitable [timing] channels”

Background: Floating point and subnormals



Value = $(-1)^{\textit{sign}} \times \textit{significand} \times 2^{(\textit{exponent} - \textit{bias})}$

Normal values have nonzero *exponent*, implicit leading 1. before *significand*

Subnormal values have all-zero *exponent*, implicit leading 0. before *significand*

⇒ slow-path special handling in hardware

Security implication of timing variability: A side channel

A side channel is a leak of secret information through a channel other than intended system output

Software side channel mechanisms hypothesized by Kocher, 1996:

1. Trace of *instructions executed* depends on secret
2. Trace of *memory locations read/written* depends on secret
3. Trace of *instruction operand values* depends on secret

Many instances of #1 and #2 have been demonstrated

Floating point as a timing side channel [AKMJLS'15]

We presented the first instruction data-based timing side channel attack on a commodity desktop processor — almost 20 years after first hypothesized [K'96]

New attacks demonstrated:

- Cross-origin pixel stealing in Firefox 24–27
  - … and, in followup work [KS'17], in recent Firefox, Safari, Chrome
- Private information exfiltration from Fuzz differentially private database

Other classes of software also likely vulnerable.

Browser security in one slide

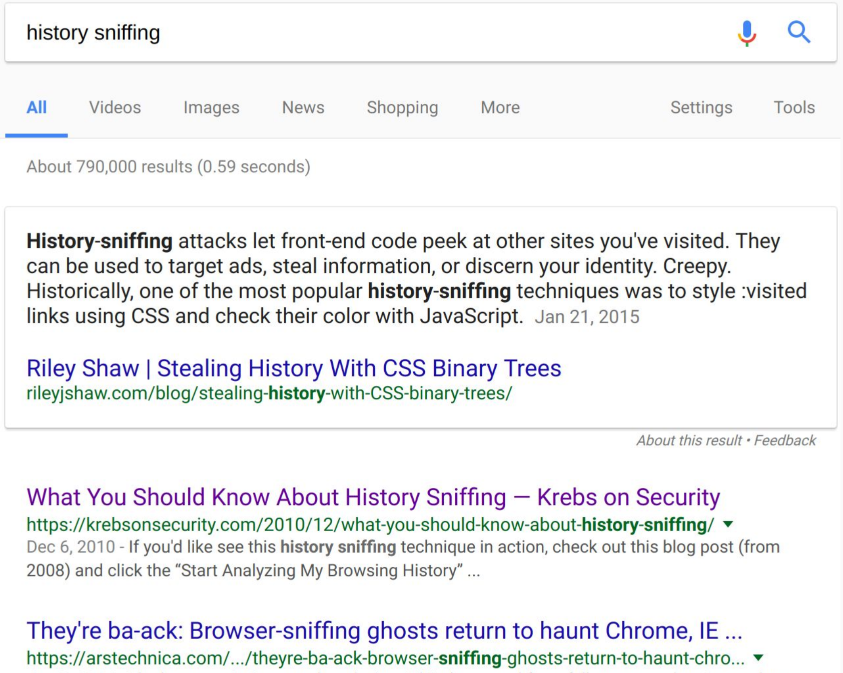
Browsers allow a user to interact with multiple, mutually distrusting origins

Browsers manage and display secrets shared between the user and each origin, which must not be exposed to other origins

Browsers allow programs supplied by an origin to run on the user's CPU and to interact with the user's system through a rich API

A single document may incorporate content from multiple origins, e.g., through <img> or <iframe> elements

Warmup: history sniffing



history sniffing

All Videos Images News Shopping More Settings Tools

About 790,000 results (0.59 seconds)

History-sniffing attacks let front-end code peek at other sites you've visited. They can be used to target ads, steal information, or discern your identity. Creepy. Historically, one of the most popular **history-sniffing** techniques was to style :visited links using CSS and check their color with JavaScript. Jan 21, 2015

Riley Shaw | Stealing History With CSS Binary Trees
rileyjshaw.com/blog/stealing-**history**-with-CSS-binary-trees/

About this result • Feedback

What You Should Know About History Sniffing — Krebs on Security
https://krebsonsecurity.com/2010/12/what-you-should-know-about-**history-sniffing**/
Dec 6, 2010 - If you'd like see this **history sniffing** technique in action, check out this blog post (from 2008) and click the "Start Analyzing My Browsing History" ...

They're ba-ack: Browser-sniffing ghosts return to haunt Chrome, IE ...
https://arstechnica.com/.../theyre-ba-ack-browser-**sniffing**-ghosts-return-to-haunt-chro...

Browser decorates links differently depending on whether the user has visited the target page

`window.getComputedStyle()` applied to a link element returns all CSS properties applied to the link … *including color*

Attack known since 2002; academic research [JS'06; JO'10; WHKK'10] showed effectiveness

But was it happening in the wild?

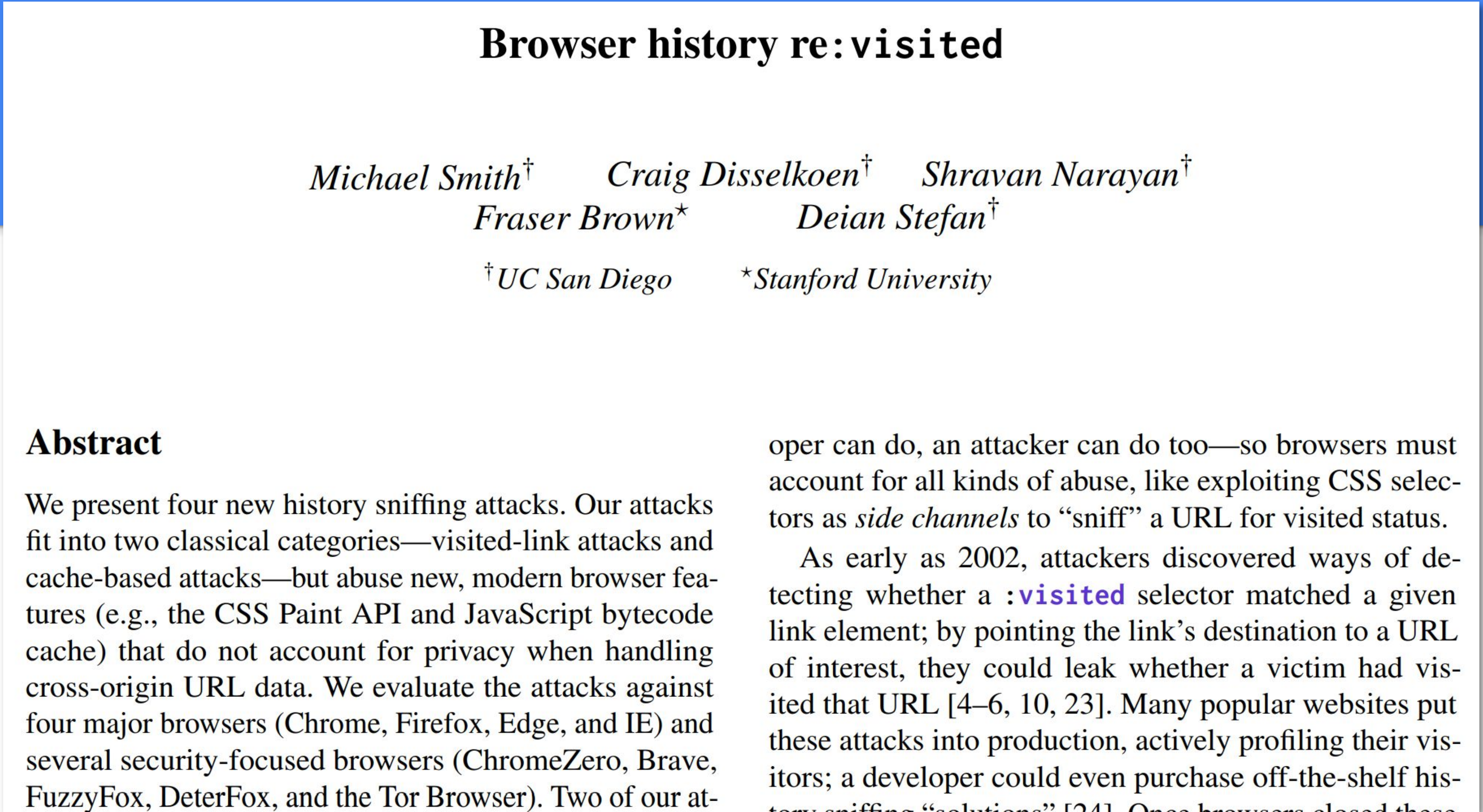
History sniffing as an information-flow problem [JJLS’10]

```
at $1.getComputedStyle($2, …)
  if $2.isLink() inject "secret"
at document.send($1,$2)
  block "secret" on $2
```

Source rewriting: rewrite JavaScript to propagate taint

Implemented in Chrome, with minimal browser changes

Surveyed (the front pages of) the Alexa Top 50,000 sites …



Browser history re:`visited`

Michael Smith[†] *Craig Disselkoen*[†] *Shravan Narayan*[†]
Fraser Brown[⋆] *Deian Stefan*[†]

[†]*UC San Diego* [⋆]*Stanford University*

Abstract

We present four new history sniffing attacks. Our attacks fit into two classical categories—visited-link attacks and cache-based attacks—but abuse new, modern browser features (e.g., the CSS Paint API and JavaScript bytecode cache) that do not account for privacy when handling cross-origin URL data. We evaluate the attacks against four major browsers (Chrome, Firefox, Edge, and IE) and several security-focused browsers (ChromeZero, Brave, FuzzyFox, DeterFox, and the Tor Browser). Two of our at-

oper can do, an attacker can do too—so browsers must account for all kinds of abuse, like exploiting CSS selectors as *side channels* to “sniff” a URL for visited status.

As early as 2002, attackers discovered ways of detecting whether a `:visited` selector matched a given link element; by pointing the link's destination to a URL of interest, they could leak whether a victim had visited that URL [4–6, 10, 23]. Many popular websites put these attacks into production, actively profiling their visitors; a developer could even purchase off-the-shelf his-

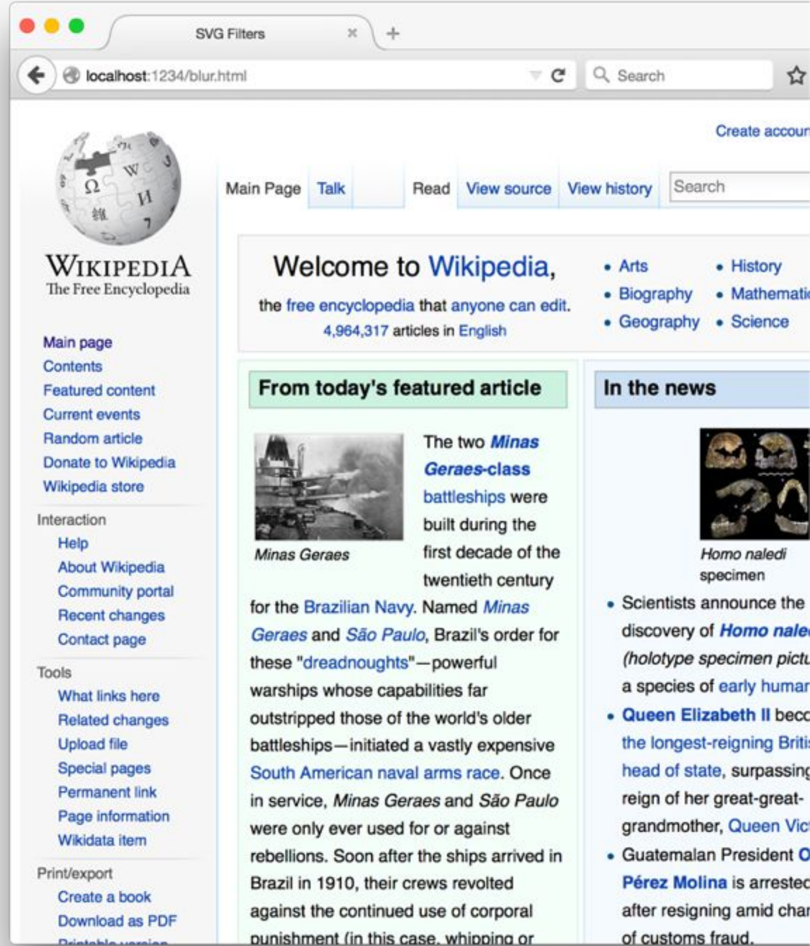
Pixel-stealing attacks

Many secrets shared between the user and an origin are expressed visually in documents in that origin: login name, bank balances, inbox contents, etc.

Cross-origin pixel-stealing attack: attacker learns other-origin private info
(victim site must allow itself to be framed)

Same-origin pixel-stealing attack: history-sniffing (at circa 60 links / second)

Cross-origin SVG filters: Turn this …



SVG Filters

localhost:1234/blur.html

Search

Create accour

WIKIPEDIA
The Free Encyclopedia

Main Page Talk Read View source View history Search

Welcome to Wikipedia,
the free encyclopedia that anyone can edit.
4,964,317 articles in English

- Arts
- Biography
- Geography
- History
- Mathemati
- Science

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item

Print/export
Create a book
Download as PDF

From today's featured article

Minas Geraes

The two ***Minas Geraes*-class battleships** were built during the first decade of the twentieth century for the Brazilian Navy. Named *Minas Geraes* and *São Paulo*, Brazil's order for these "dreadnoughts"—powerful warships whose capabilities far outstripped those of the world's older battleships—initiated a vastly expensive South American naval arms race. Once in service, *Minas Geraes* and *São Paulo* were only ever used for or against rebellions. Soon after the ships arrived in Brazil in 1910, their crews revolted against the continued use of corporal punishment (in this case, whipping or

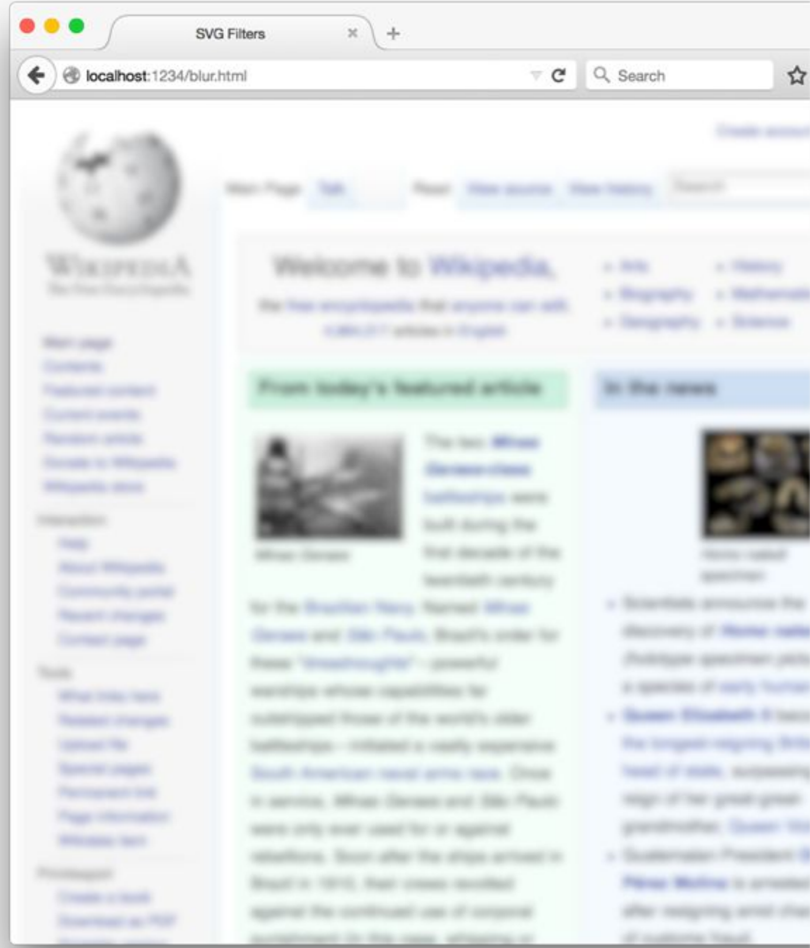
In the news

Homo naledi specimen

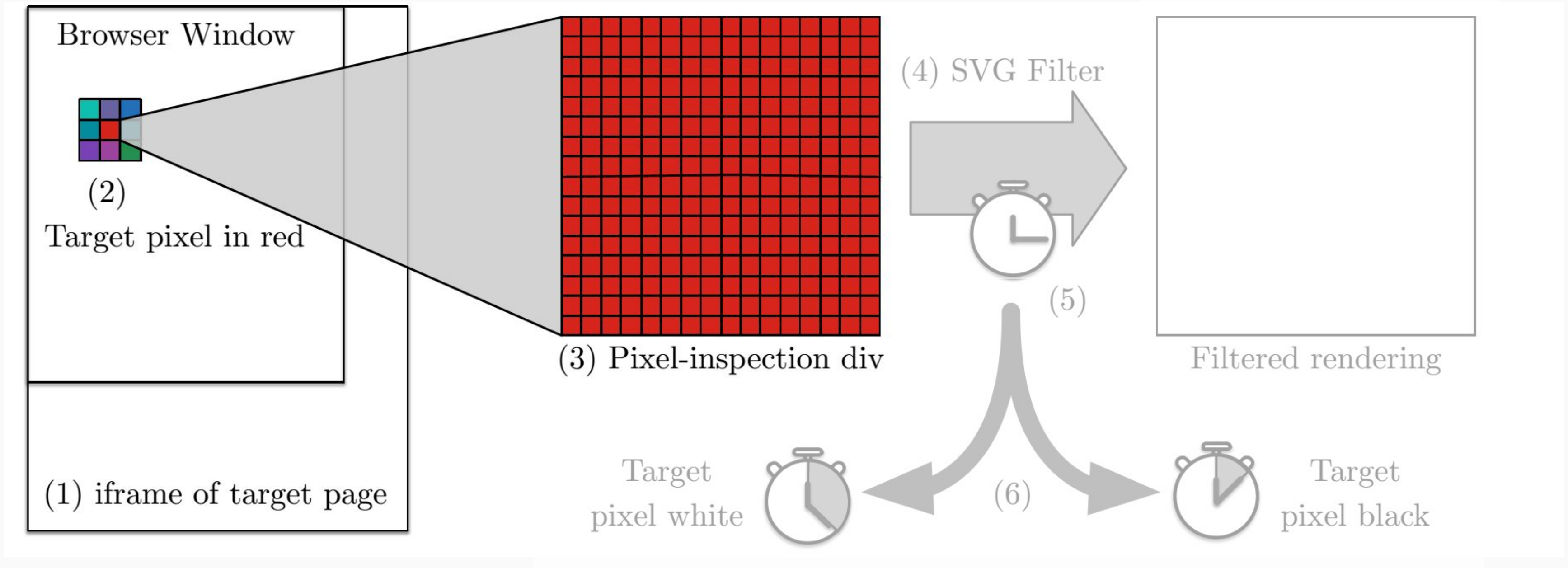
- Scientists announce the discovery of ***Homo nale*** *(holotype specimen pictu* a species of early huma
- **Queen Elizabeth II** bec the longest-reigning Briti head of state, surpassing reign of her great-great-grandmother, Queen Vic
- Guatemalan President **O Pérez Molina** is arreste after resigning amid cha of customs fraud.

… into this!

```
<svg><filter id="f">
<feGaussianBlur
    stdDeviation="3"/>
</filter></svg>
```



SVG filter timing attack methodology



See Paul Stone's "Pixel Perfect Timing Attacks with HTML5"

Paul Stone's `feMorphology` timing channel

Firefox implementation for feMorphology had a special case:

```
// We need to scan the entire kernel
if (x == rect.x || xExt[0] <= startX || xExt[1] <= startX ||
    xExt[2] <= startX || xExt[3] <= startX) {
  [...]
{ else { // We only need to look at the newest column
  [...]
}
```

Mozilla fix: try to write constant-time filter code

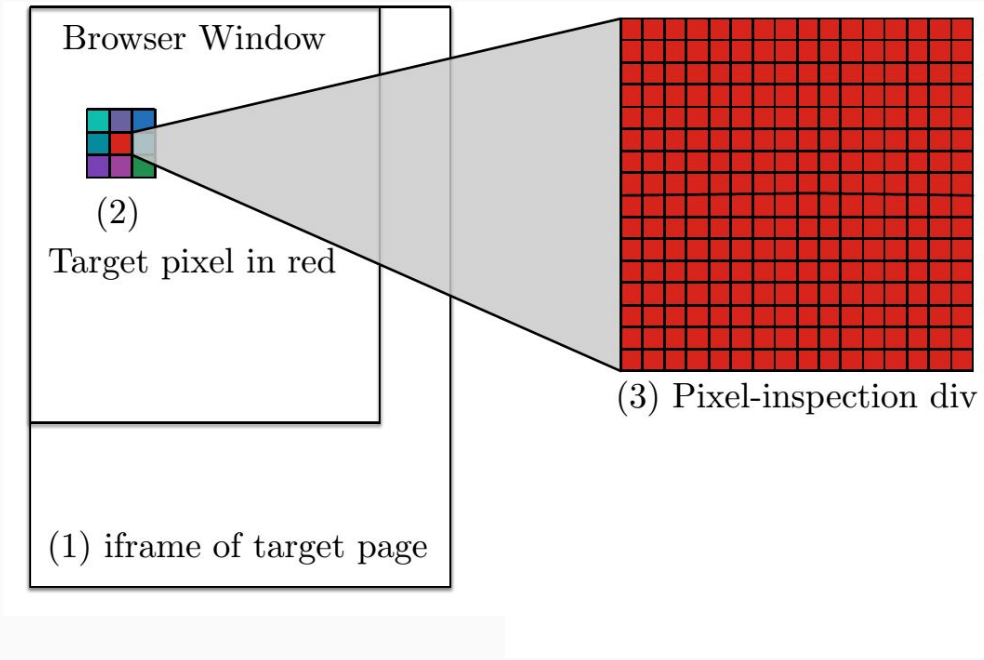
(took two years to land, and a 150-comment Bugzilla thread)

Our pixel-stealing attack on Firefox (versions 23–27)

Firefox's "constant-time" SVG filters still used floating point!

Attacker chooses filter and settings so black pixels generate fewer subnormal ops than white pixels

Slowdown is just a few cycles per instruction; need to amplify it:



Example filter: `feConvolveMatrix` (in pseudocode)

```
def do_one_convolve(kernel, subimage):
  for x,y in subimage:
    tmp[x,y] = kernel[x,y] * subimage[x,y]
  result = 0.0
  for x,y in tmp:
    result = result + tmp[x,y]
  return result

for x,y in input:
  output[x,y] = do_one_convolve(kernel,
                    subimage_around(input, x, y,
                        kernel.width, kernel.height))
```

$s \times 0. = 0.$ if black
$s \times 1. = s$ if white

$0. + 0. = 0.$ if black
$s + s = s$ if white

Use a kernel consisting of all subnormal values.

Differential privacy in two slides (I)

Have a database table consisting of sensitive information, e.g.,

Subscriber	**Movie**	**Times watched**	**Rating**
Hovav Shacham	*Mean Girls*	31	☆☆☆☆☆
…	…	…	…

Would like to allow untrusted parties to supply arbitrary maps *f* for map-reduce

$f($

Hovav Shacham	*Mean Girls*	31	☆☆☆☆☆

$) + f(\cdots) + \cdots$

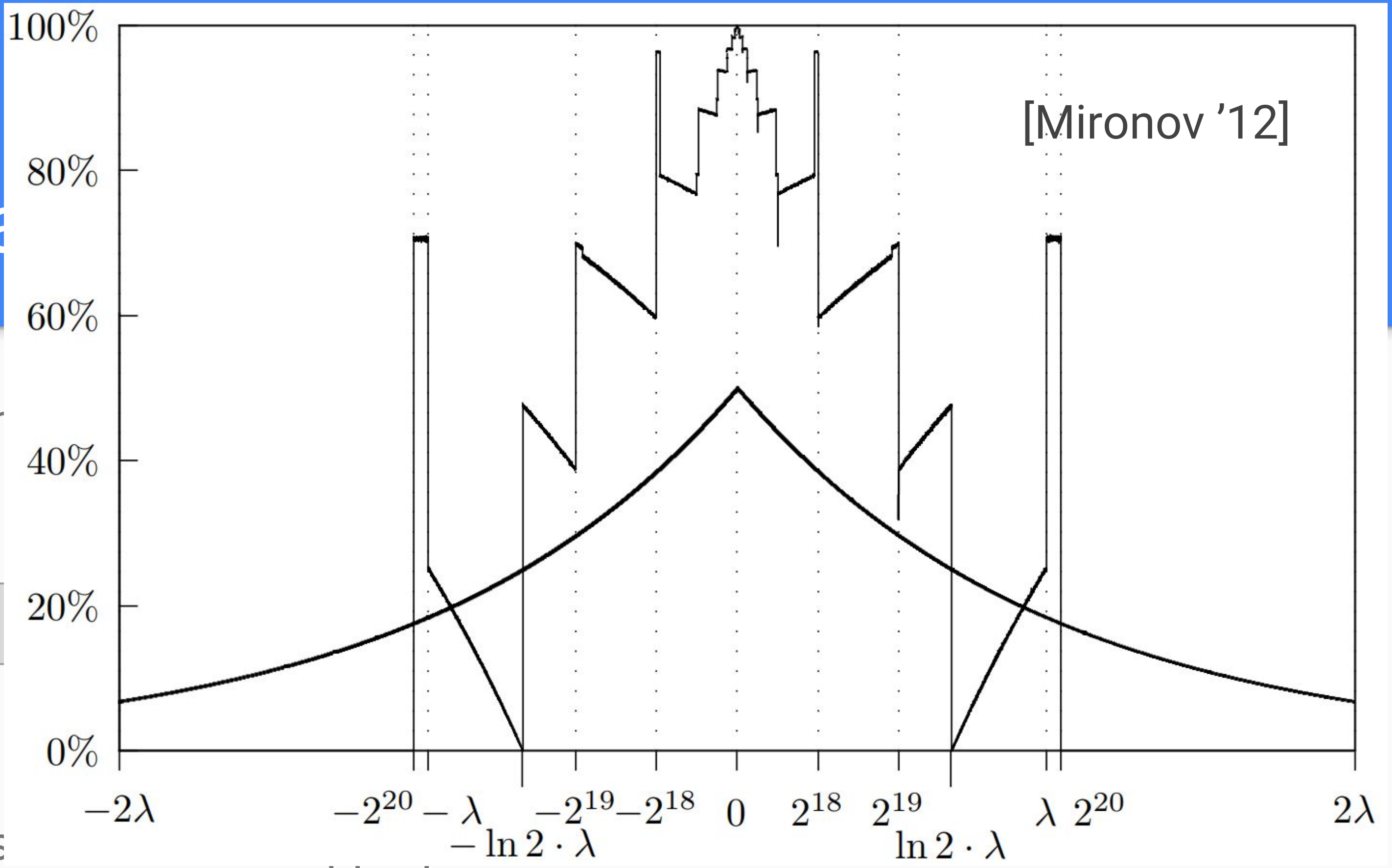
… *without* leaking presence or absence of individual rows

Differentia

Challenge: map m

$f($ u | v | x

Solution: add *nois*



$\sum_i f(\text{row}_i)$ + *noise*

Covert timing channels in differentially private databases

Map *f* always returns 0.0, but spins for a minute iff run on the sensitive row

Total query runtime discloses presence (or absence) of sensitive row

Fuzz solution (Usenix Security 2014):

- Run each instance of *f* in restrictive sandbox
- Clamp execution on each row to an exact cycle count

Our attack on Fuzz

While map is sandboxed, *reduce* code is trusted.

Our map *f* returns a subnormal value if it sees the sensitive row, 0.0 otherwise

```
    foldl +. [0.0, 0.0, 0.0, …, 0.0]  is fast;
    foldl +. [1.0e-323, 0.0, …, 0.0]  is slow.
```

Can even tell approximate position of sensitive row in table

Other potentially vulnerable software

- Any trusted software written in JavaScript
  - Node.js software on server
  - Iframes implementing postMessage RPC in browser
- Deep learning software using FPU
- Lattice crypto software using FPU?

No reason to think that floating-point instructions are the only ones to show variability.

Our approach to closing floating-point timing channels: libfixedtimefixedpoint

Fork me on GitHub

The floating-point unit is too dangerous to use in security-relevant code.

Instead, use integer unit to implement fixed-point math

We wrote libfixedtimefixedpoint

- Implements most common math operations (not all guaranteed precise)
- Between 1 and 61 fractional bits, selected at compile time
- No data-dependent branches, no data-dependent table lookups, no instruction with runtime known to be variable (e.g., div)

Writing constant-time code is a battle against the processor and the compiler.

```
int64_t fix_to_int64(fixed op1) {
  return ({ uint8_t isinfpos = (((op1 )&((fixed) 0x3)) == ((fixed) 0 x2));
uint8_t isinfneg = (((op1 )&((fixed) 0x3)) == ((fixed) 0 x3)); uint8_t
isnan = (((op1) &((fixed) 0x3)) == ((fixed) 0x1 )); uint8_t ex = isinfpos
| isinfneg | isnan; fixed result_nosign = (({uint64_t SE_m__ = (1ull <<
((64 - ((60 + 2)))-1)); (((uint64_t) ((op1) >> ((60 + 2)))) ^ SE_m__) -
SE_m__;}) + !!( (!!((op1) & (1 LL << (((60 + 2))-1))) & !!(( op1) & ((1LL
<< (((60 + 2))-1)) -1))) | ((((op1) >> (((60 + 2)) -2)) & 0x6) == 0x6) ));
((({ uint64_t SE_m__ = (1ull << ((1) -1)); (((uint64_t) (!!(isinfpos ))) ^
SE_m__) - SE_m__;}) & (9223372036854775807LL)) | (({ uint64_t SE_m__ =
(1ull << ((1) -1)); (((uint64_t) (!!(isinfneg ))) ^ SE_m__) - SE_m__;}) &
((-9223372036854775807LL -1))) | (({uint64_t SE_m__ = (1ull << ((1)-1));
(((uint64_t) (!!(!ex ))) ^ SE_m__) - SE_m__;}) & ( result_nosign)); });
}
```

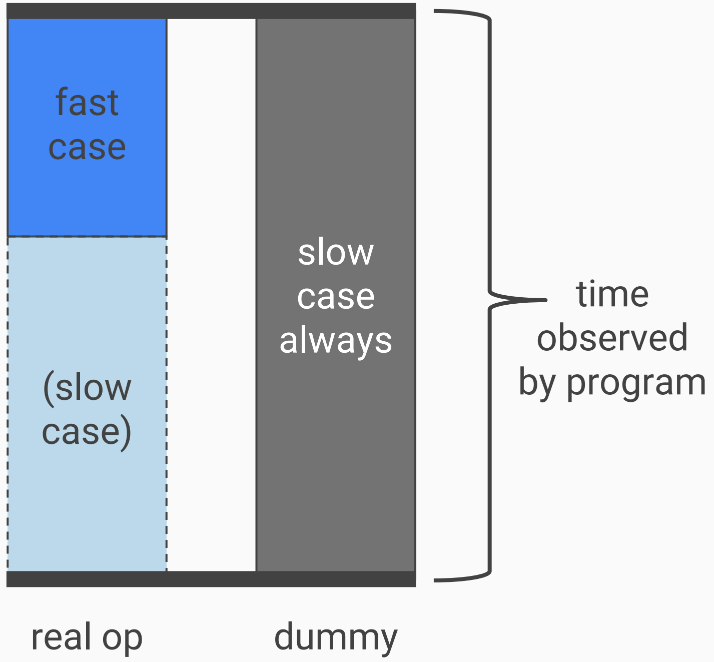
LibFTFP recently proved to be constant time as LLVM bytecode [ABBDE’16];
may or may not be constant-time on actual x86 processors

Another approach to closing floating-point timing channels: Escort [RLT'16]

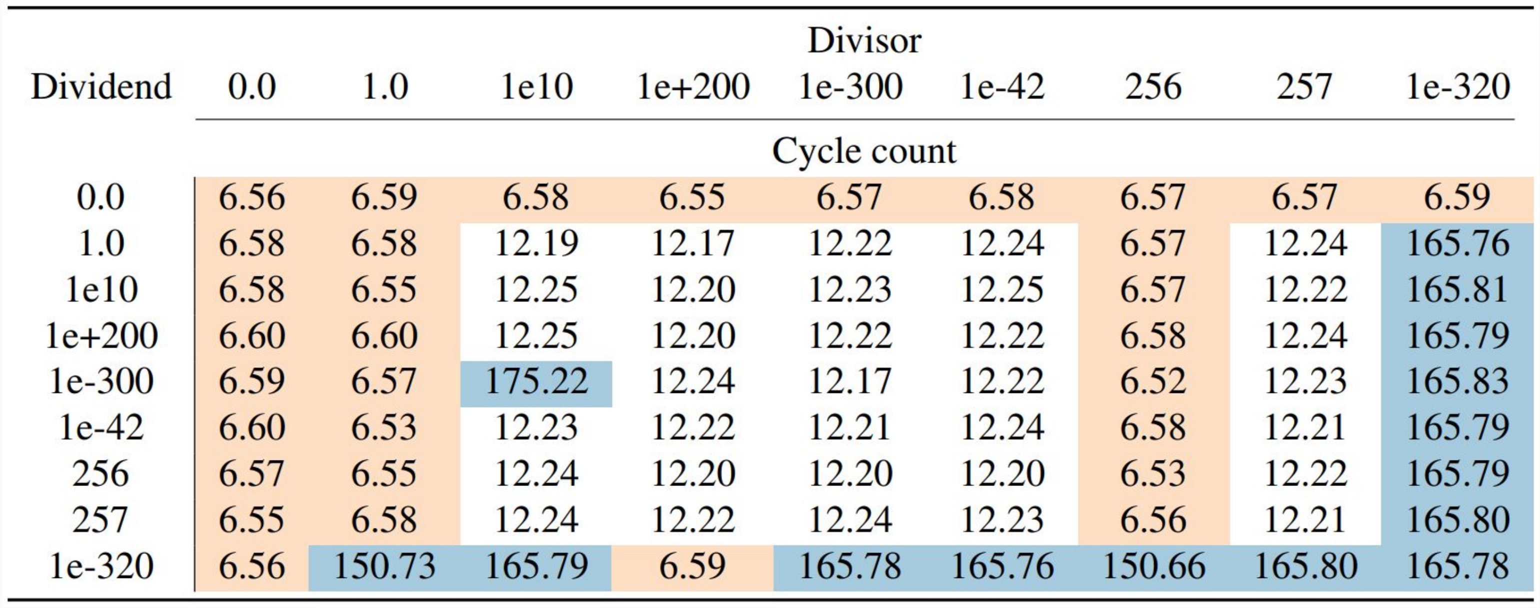
Use program analysis to find FP ops guaranteed not to take or emit subnormals; these can be run as before

Run remaining ops on SIMD unit with a dummy "escort" op that runs in worst-case time

Conjecture: real, dummy ops run in parallel



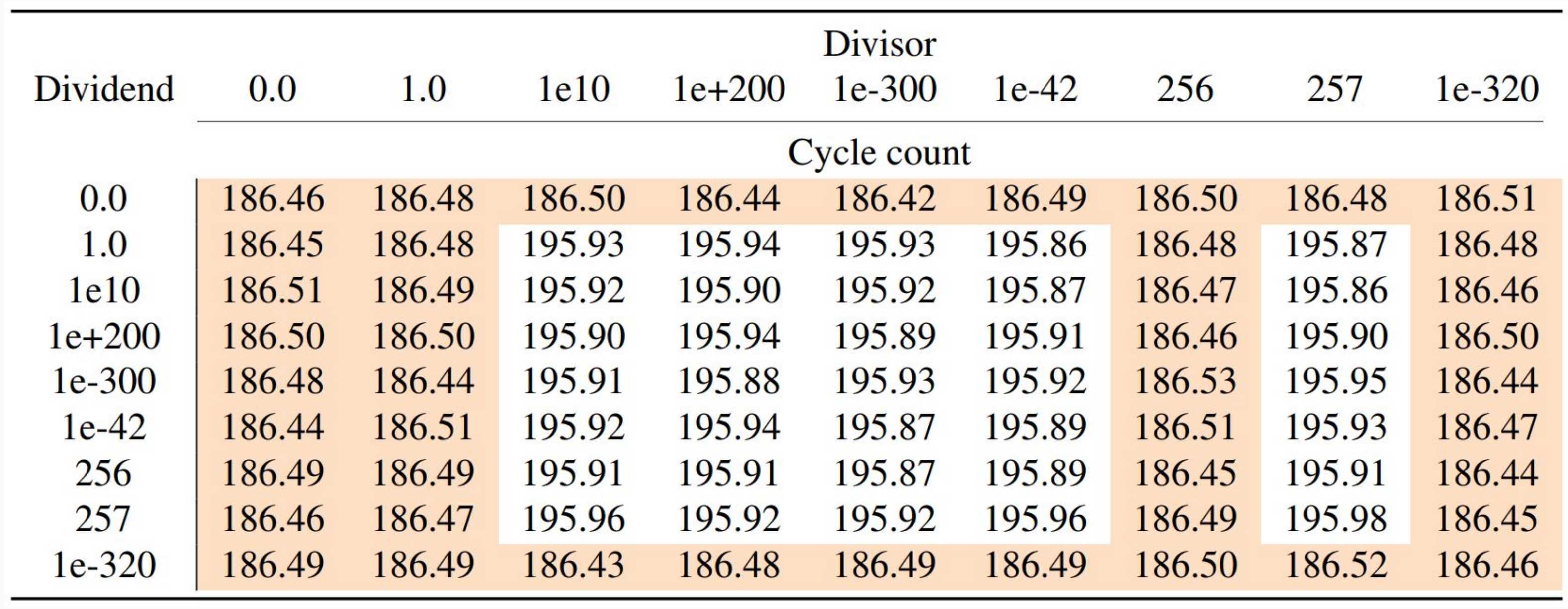
Our analysis of Escort [KS'17]: FP ops show timing variation beyond subnormals



Dividend	Divisor								
	0.0	1.0	1e10	1e+200	1e-300	1e-42	256	257	1e-320
	Cycle count								
0.0	6.56	6.59	6.58	6.55	6.57	6.58	6.57	6.57	6.59
1.0	6.58	6.58	12.19	12.17	12.22	12.24	6.57	12.24	165.76
1e10	6.58	6.55	12.25	12.20	12.23	12.25	6.57	12.22	165.81
1e+200	6.60	6.60	12.25	12.20	12.22	12.22	6.58	12.24	165.79
1e-300	6.59	6.57	175.22	12.24	12.17	12.22	6.52	12.23	165.83
1e-42	6.60	6.53	12.23	12.22	12.21	12.24	6.58	12.21	165.79
256	6.57	6.55	12.24	12.20	12.20	12.20	6.53	12.22	165.79
257	6.55	6.58	12.24	12.22	12.24	12.23	6.56	12.21	165.80
1e-320	6.56	150.73	165.79	6.59	165.78	165.76	150.66	165.80	165.78

double-precision SSE scalar division on Intel i5-4460

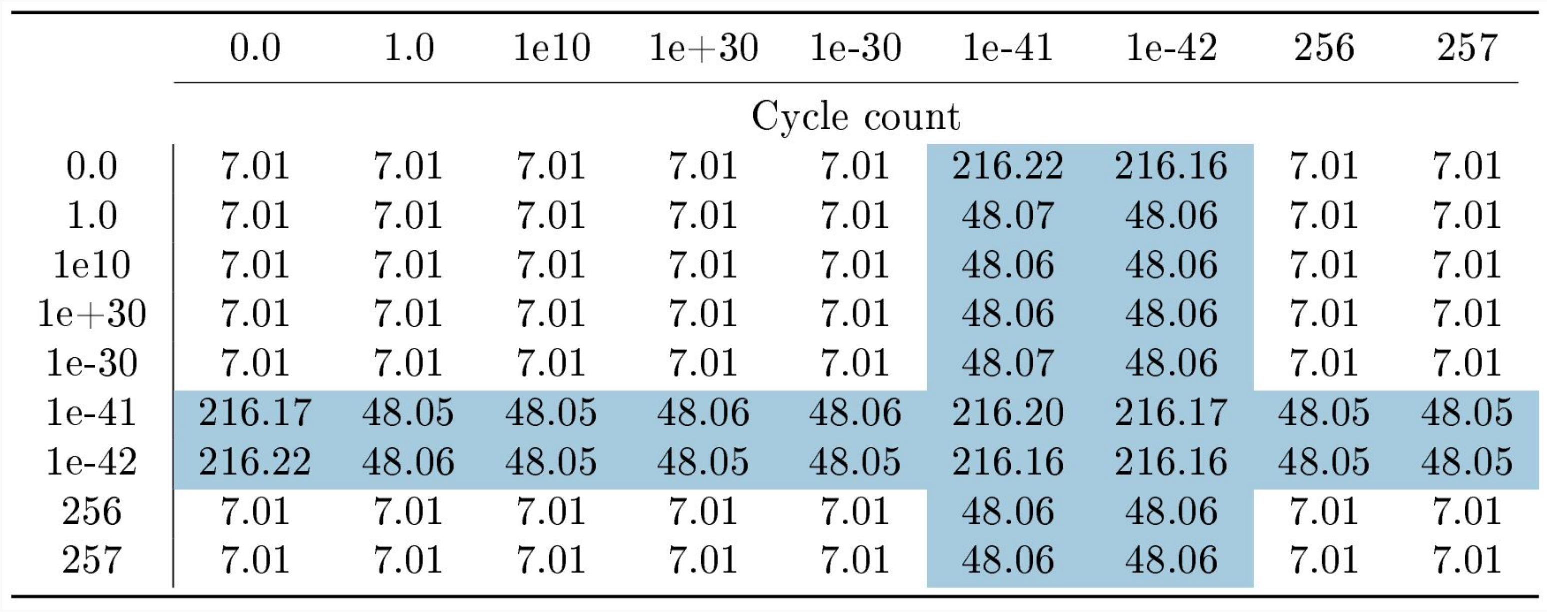
Our analysis of Escort (cont.): escorted SIMD ops show timing variation



	Divisor								
Dividend	0.0	1.0	1e10	1e+200	1e-300	1e-42	256	257	1e-320
	Cycle count								
0.0	186.46	186.48	186.50	186.44	186.42	186.49	186.50	186.48	186.51
1.0	186.45	186.48	195.93	195.94	195.93	195.86	186.48	195.87	186.48
1e10	186.51	186.49	195.92	195.90	195.92	195.87	186.47	195.86	186.46
1e+200	186.50	186.50	195.90	195.94	195.89	195.91	186.46	195.90	186.50
1e-300	186.48	186.44	195.91	195.88	195.93	195.92	186.53	195.95	186.44
1e-42	186.44	186.51	195.92	195.94	195.87	195.89	186.51	195.93	186.47
256	186.49	186.49	195.91	195.91	195.87	195.89	186.45	195.91	186.44
257	186.46	186.47	195.96	195.92	195.92	195.96	186.49	195.98	186.45
1e-320	186.49	186.49	186.43	186.48	186.49	186.49	186.50	186.52	186.46

double-precision SSE SIMD division on Intel i5-4460, subnormal dummy op

Sidebar: Different processors exhibit different timing behavior



	0.0	1.0	1e10	1e+30	1e-30	1e-41	1e-42	256	257
					Cycle count				
0.0	7.01	7.01	7.01	7.01	7.01	216.22	216.16	7.01	7.01
1.0	7.01	7.01	7.01	7.01	7.01	48.07	48.06	7.01	7.01
1e10	7.01	7.01	7.01	7.01	7.01	48.06	48.06	7.01	7.01
1e+30	7.01	7.01	7.01	7.01	7.01	48.06	48.06	7.01	7.01
1e-30	7.01	7.01	7.01	7.01	7.01	48.07	48.06	7.01	7.01
1e-41	216.17	48.05	48.05	48.06	48.06	216.20	216.17	48.05	48.05
1e-42	216.22	48.06	48.05	48.05	48.05	216.16	216.16	48.05	48.05
256	7.01	7.01	7.01	7.01	7.01	48.06	48.06	7.01	7.01
257	7.01	7.01	7.01	7.01	7.01	48.06	48.06	7.01	7.01

single-precision SSE addition on AMD Phenom II X2 550

Revisiting browser SVG filter implementations

Since our 2015 paper:

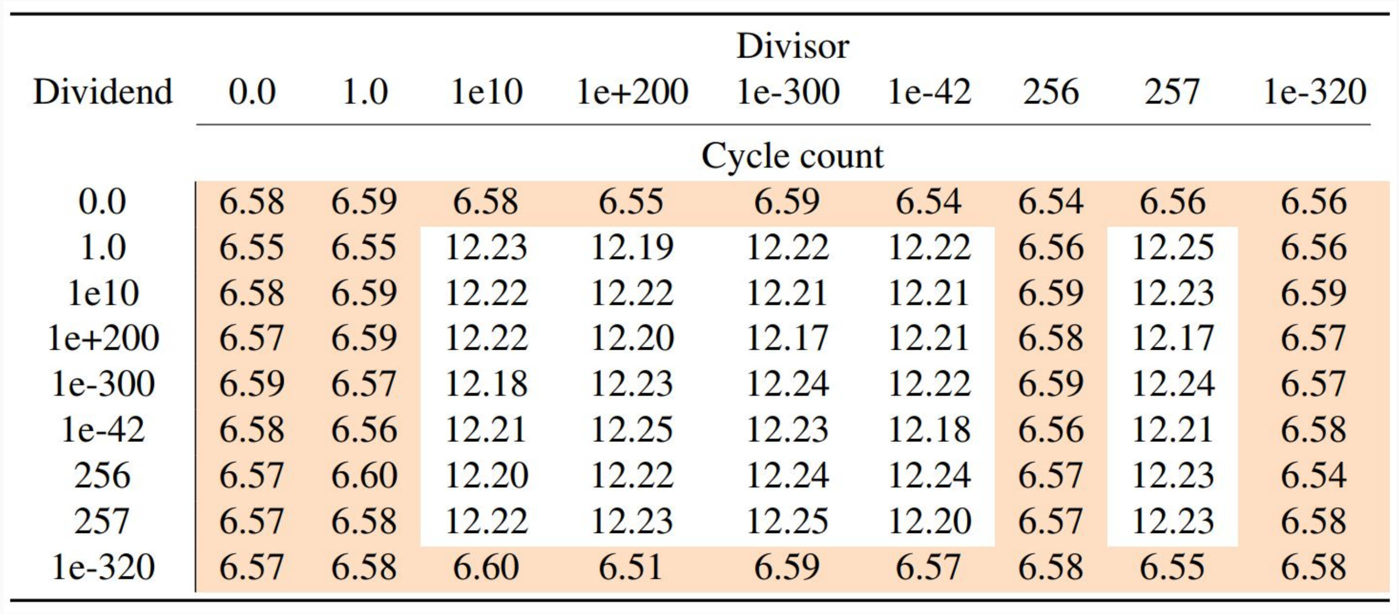
- Firefox SVG filters rewritten mostly to use fixed-point math
- Chrome SVG filters moved to GPU or run with FTZ/DAZ enabled
- Safari implementation not meaningfully changed

We found and disclosed new floating-point pixel-stealing attacks on all three.

Firefox: CVE-2017-5407; Safari: CVE-2017-7006; Chrome: CVE-2017-5107

Safari no longer applies SVG filters to cross-origin iframes.

Some operations still show variable timing even with FTZ and DAZ enabled



	Divisor								
Dividend	0.0	1.0	1e10	1e+200	1e-300	1e-42	256	257	1e-320
	Cycle count								
0.0	6.58	6.59	6.58	6.55	6.59	6.54	6.54	6.56	6.56
1.0	6.55	6.55	12.23	12.19	12.22	12.22	6.56	12.25	6.56
1e10	6.58	6.59	12.22	12.22	12.21	12.21	6.59	12.23	6.59
1e+200	6.57	6.59	12.22	12.20	12.17	12.21	6.58	12.17	6.57
1e-300	6.59	6.57	12.18	12.23	12.24	12.22	6.59	12.24	6.57
1e-42	6.58	6.56	12.21	12.25	12.23	12.18	6.56	12.21	6.58
256	6.57	6.60	12.20	12.22	12.24	12.24	6.57	12.23	6.54
257	6.57	6.58	12.22	12.23	12.25	12.20	6.57	12.23	6.58
1e-320	6.57	6.58	6.60	6.51	6.59	6.57	6.58	6.55	6.58

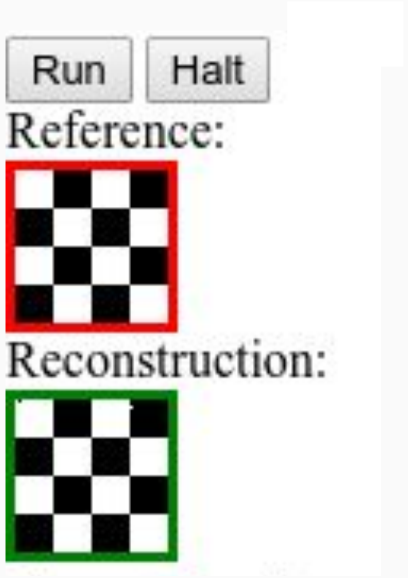
double-precision SSE division on Intel i5-4460, FTZ and DAZ enabled

Our attack on Chrome/Skia

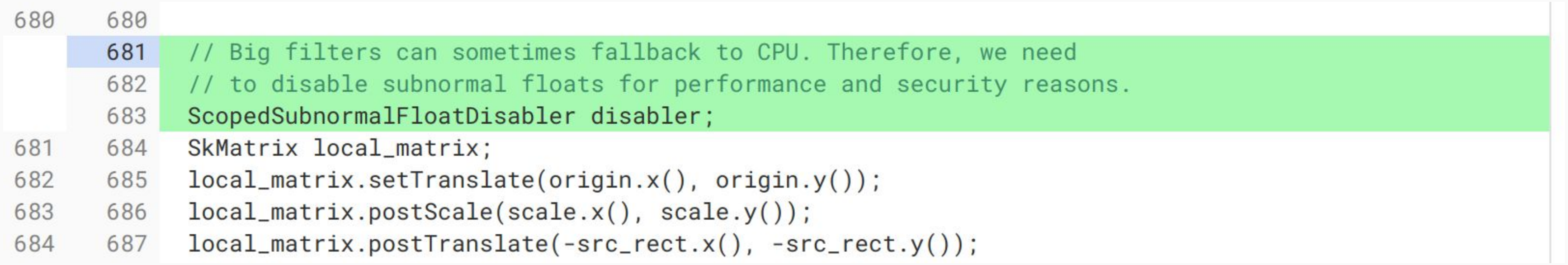
On i5-4460, only 32-bit divide and square root show timing variation when FTZ and DAZ are enabled

Unable to find a filter that uses these operations unsafely

Skia turns FTZ/DAZ off when a filter is sent to the GPU, failed to turn FTZ/DAZ back on when it changes its mind



Chrome/Skia bugfix

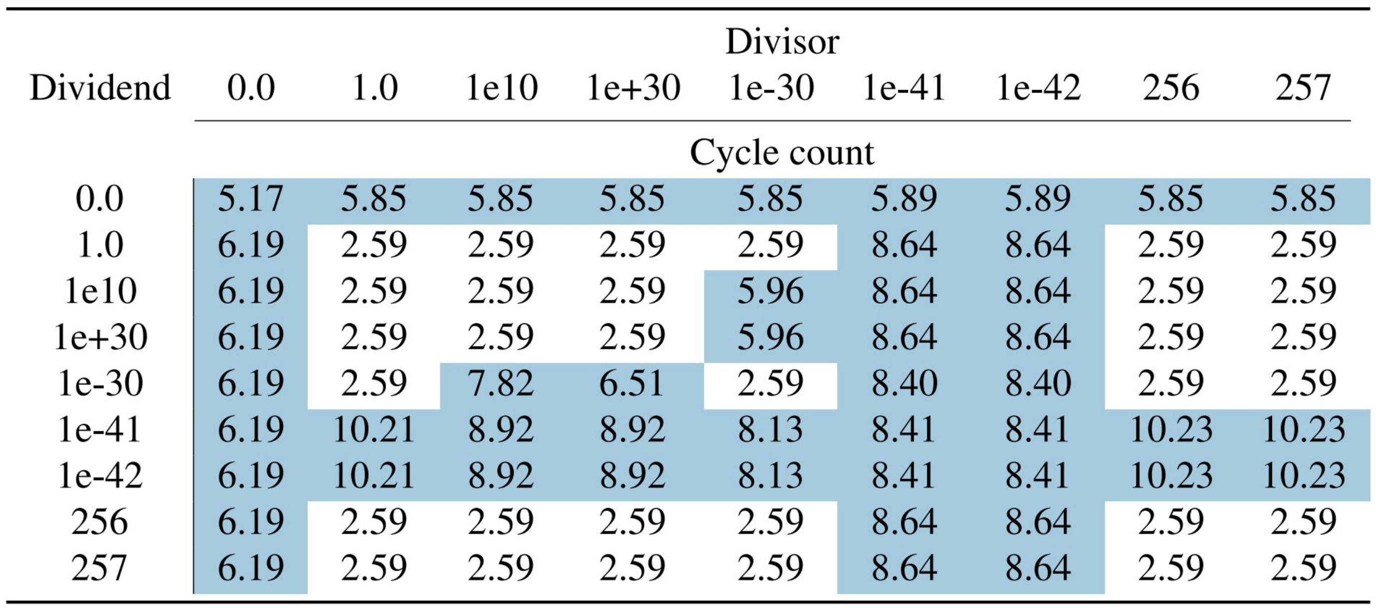


```
// Big filters can sometimes fallback to CPU. Therefore, we need
// to disable subnormal floats for performance and security reasons.
ScopedSubnormalFloatDisabler disabler;
SkMatrix local_matrix;
local_matrix.setTranslate(origin.x(), origin.y());
local_matrix.postScale(scale.x(), scale.y());
local_matrix.postTranslate(-src_rect.x(), -src_rect.y());
```

In QA for patch, developers discovered FTZ/DAZ never enabled on Windows!

Lesson: processor flags offer brittle security, are difficult to manage

GPUs will not save us



	Divisor								
Dividend	0.0	1.0	1e10	1e+30	1e-30	1e-41	1e-42	256	257
	Cycle count								
0.0	5.17	5.85	5.85	5.85	5.85	5.89	5.89	5.85	5.85
1.0	6.19	2.59	2.59	2.59	2.59	8.64	8.64	2.59	2.59
1e10	6.19	2.59	2.59	2.59	5.96	8.64	8.64	2.59	2.59
1e+30	6.19	2.59	2.59	2.59	5.96	8.64	8.64	2.59	2.59
1e-30	6.19	2.59	7.82	6.51	2.59	8.40	8.40	2.59	2.59
1e-41	6.19	10.21	8.92	8.92	8.13	8.41	8.41	10.23	10.23
1e-42	6.19	10.21	8.92	8.92	8.13	8.41	8.41	10.23	10.23
256	6.19	2.59	2.59	2.59	2.59	8.64	8.64	2.59	2.59
257	6.19	2.59	2.59	2.59	2.59	8.64	8.64	2.59	2.59

single-precision division on NVIDIA GeForce GT 430

What is to be done?

Browser vendors:

- Eliminate unnecessary cross-origin interactions
- Reduce resolution of reference timers [Wray ‘91]

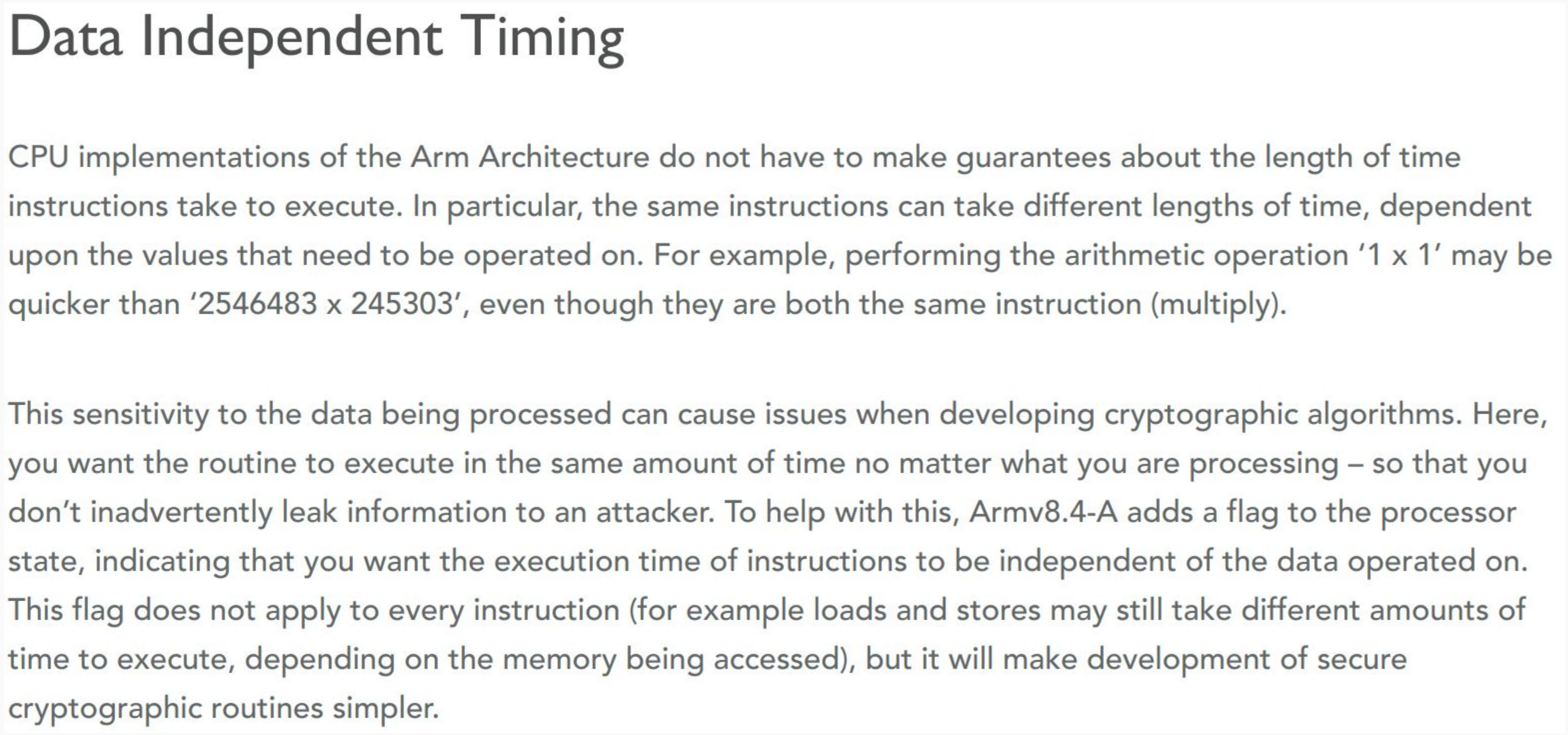
Compiler vendors:

- Make constant-time programming less miserable

CPU vendors:

- Document variable-time and constant-time instructions
- Implement opt-in constant-time mode

“Introducing 2017’s extensions to the Arm Architecture,” November 2017



Data Independent Timing

CPU implementations of the Arm Architecture do not have to make guarantees about the length of time instructions take to execute. In particular, the same instructions can take different lengths of time, dependent upon the values that need to be operated on. For example, performing the arithmetic operation ‘1 x 1’ may be quicker than ‘2546483 x 245303’, even though they are both the same instruction (multiply).

This sensitivity to the data being processed can cause issues when developing cryptographic algorithms. Here, you want the routine to execute in the same amount of time no matter what you are processing – so that you don’t inadvertently leak information to an attacker. To help with this, Armv8.4-A adds a flag to the processor state, indicating that you want the execution time of instructions to be independent of the data operated on. This flag does not apply to every instruction (for example loads and stores may still take different amounts of time to execute, depending on the memory being accessed), but it will make development of secure cryptographic routines simpler.

Questions?

In this talk:

- D. Jang, R. Jhala, S. Lerner, and H. Shacham. “An Empirical Study of Privacy-Violating Information Flows in JavaScript Web Applications.” In proc. CCS 2010.
- M. Andrysco, D. Kohlbrenner, K. Mowery, R. Jhala, S. Lerner, and H. Shacham. “On Subnormal Floating Point and Abnormal Timing.” In proc. Oakland 2015.
- D. Kohlbrenner and H. Shacham. “Trusted Browsers for Uncertain Times.” In proc. USENIX Security 2016.
- D. Kohlbrenner and H. Shacham. “On the effectiveness of mitigations against floating-point timing channels.” In proc. USENIX Security 2017.

https://www.cs.utexas.edu/~hovav/