Loophole: Timing Attacks on Shared Event Loops in Chrome

Pepe Vila & Boris Köpf

IMDEA Software Institute

About me

I'm Pepe...

@cgvwzq
http://vwzq.net/

Some notes about Chrome

Chrome was a blackbox for me.

Its code base is immense. How to start?

- Dev-lists: https://www.chromium.org/developers/technical-discussion-groups
- Design documents (living GoogleDocs)
- Bug track: https://bugs.chromium.org/p/chromium/issues/list
- Source code: https://cs.chromium.org/

Warning!

The author feels that this technique of deliberately lying will actually make it easier for you to learn the ideas.

- Donald Knuth

Introduction

- Event-driven programming
- Event loops
- A timing side-channel on event loops

Introduction: Event-driven programming

EDP is a programming paradigm for GUI, web clients, networks, and server-side

The flow of the program is determined by events or messages

Examples:

- Nginx, Node.js or memcached
- Used for message passing: inter-(thread | process) communication
- HTML5 standard* mandates User Agents to use EDP

https://html.spec.whatwg.org/#event-loop

Introduction: Event Loops

Event loop, message dispatcher, message loop, run loop…

FIFO queue & dispatcher:

```
Q = []; // message queue
while (true) {
    M = Q.shift(); // dequeue
    process(M); // handle message
}
```

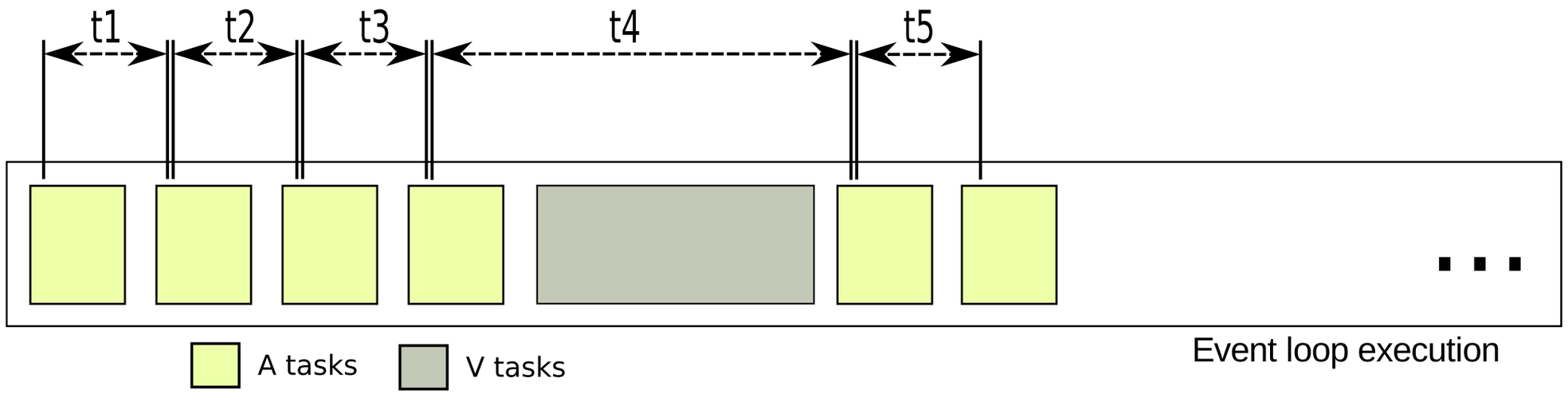
If queue is empty, waits until an event arrives

Blocking operations (e.g., database and network requests) are handled asynchronously

Simple concurrency model for programmers

Introduction: Timing side-channel on Event Loops

when shared between mutually distrusting programs



“Loophole”

Roses are red,
Violets are blue,
Side-channels are sweet,
And so are you.

- Taylor Swift

Chrome architecture

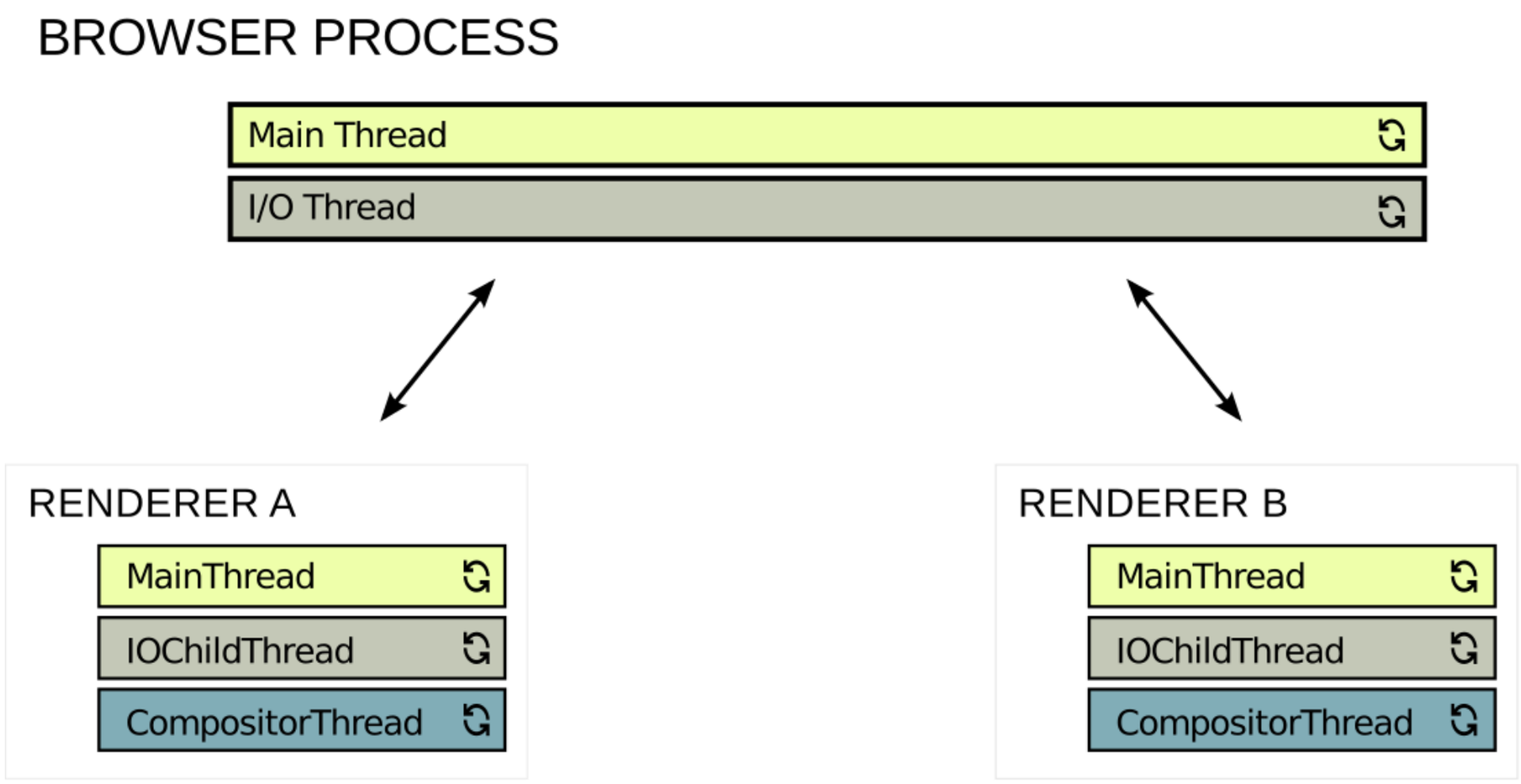
- Same Origin Policy (SOP)
- Multi-process
- Sharing Event Loops

Chrome architecture: SOP

Origin = (scheme, domain, port)

Origin 1	*Origin 2*
http://example.com:8080	http://example.com
http://mail.example.com	http://app.example.com
https://foo.example.com	https://foo.example.com
https://example.com	http://example.com

Chrome architecture: Multi-process



(Chrome's Task Manager)

Chrome architecture: Sharing Event Loops

Different policies for mapping applications into renderer processes (*process-per-site-instance,* process-per-site, process-per-tab, single-process)

A *Site* is a registered domain plus a scheme (!= SOP)

Sharing on the **renderer***:

- iframes, linked navigation or |process| > T
- T = 32 for 4 GB of RAM, and T = 70 for 8 GB or more

Sharing on the **host process**:

- one for all renderers
- IPC through host's I/O thread

*Site Isolation Project: https://www.chromium.org/developers/design-documents/site-isolation

Chrome architecture: Sharing Event Loops

Main thread of renderer processes

- resource parsing, style calculation, layout, painting and Javascript
- each JS task blocks the event loop for a while
- if 2 (or more) pages share the process, the main thread's event loop is shared

Chrome architecture: Sharing Event Loops

I/O thread of the host process

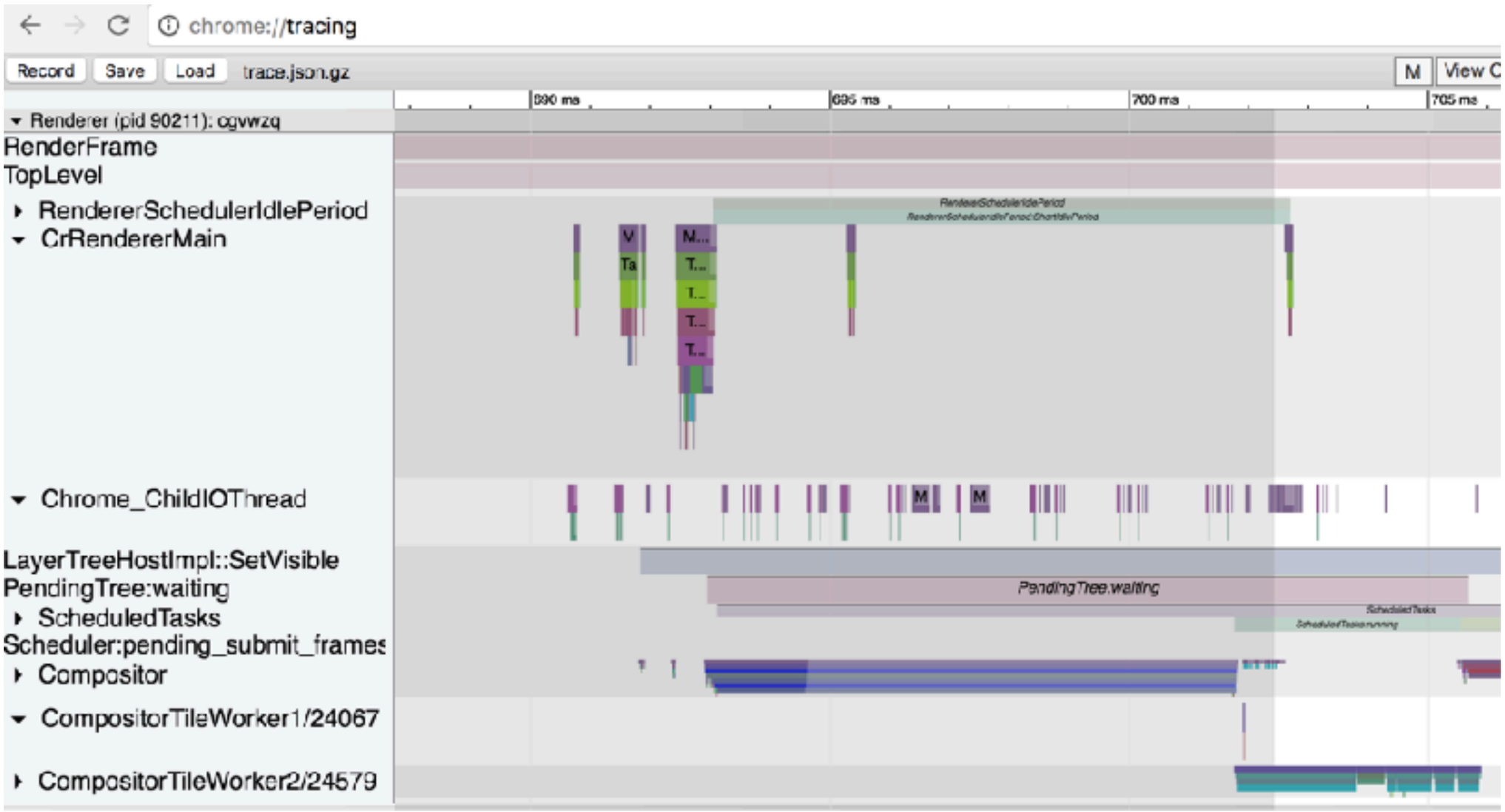
- manages IPC of all children renderers
- demultiplexes all UI events to each corresponding renderer
- multiplexes all network requests from renderers
- each task/message/event blocks the event loop (signalling start and completion)

Chrome architecture: Sharing Event Loops

Other event loops

- GPU process, workers, extensions…

Chrome architecture: Sharing Event Loops

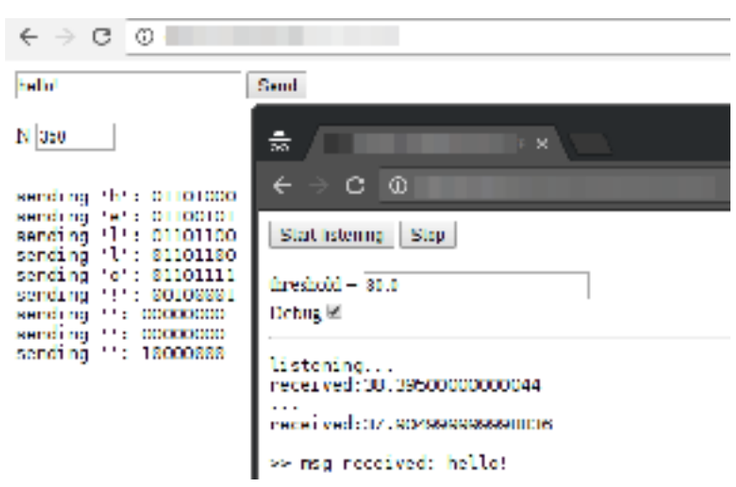


DEMO

Spy on Shared Event Loops

Malicious advertisement

Covert channel



Keylogger

Tab popup

Spy on Shared Event Loops

But… How do we post tasks into these loops?

Renderer's main thread	**Host's I/O thread**
~~setTimeout~~	network requests
postMessage	SharedWorkers

NEW! ES7 async functions and iterators

Spy on Shared Event Loops

Renderer's main thread

```
function loop() {
    self.postMessage(0, "*");
    save(performance.now());
}

self.onmessage = loop;
self.postMessage(0, "*");
```

Allocate TypedArray -> ~25μs resolution

Spy on Shared Event Loops

Host's I/O thread

```
function loop () {
    save(performance.now());
    fetch(new Request("http://0/"))
        .catch(loop);
}
loop();
```

~500µs resolution

Non routable IPs

Spy on Shared Event Loops

e.g., we can do much better with SharedWorkers :D

```
onconnect = function(e) {
    let port = e.ports[0]
    port.onmessage = function(){
        port.postMessage(0);
    }
}
```

<- pong.js

~100µs resolution

```
let w = new SharedWorker("pong.js");
function loop(){
    save(performance.now());
    w.port.postMessage(0);
}
w.port.onmessage = loop;
loop();
```

Spy on Shared Event Loops

Event-delay traces: 1s ~ 40.000 time measurements

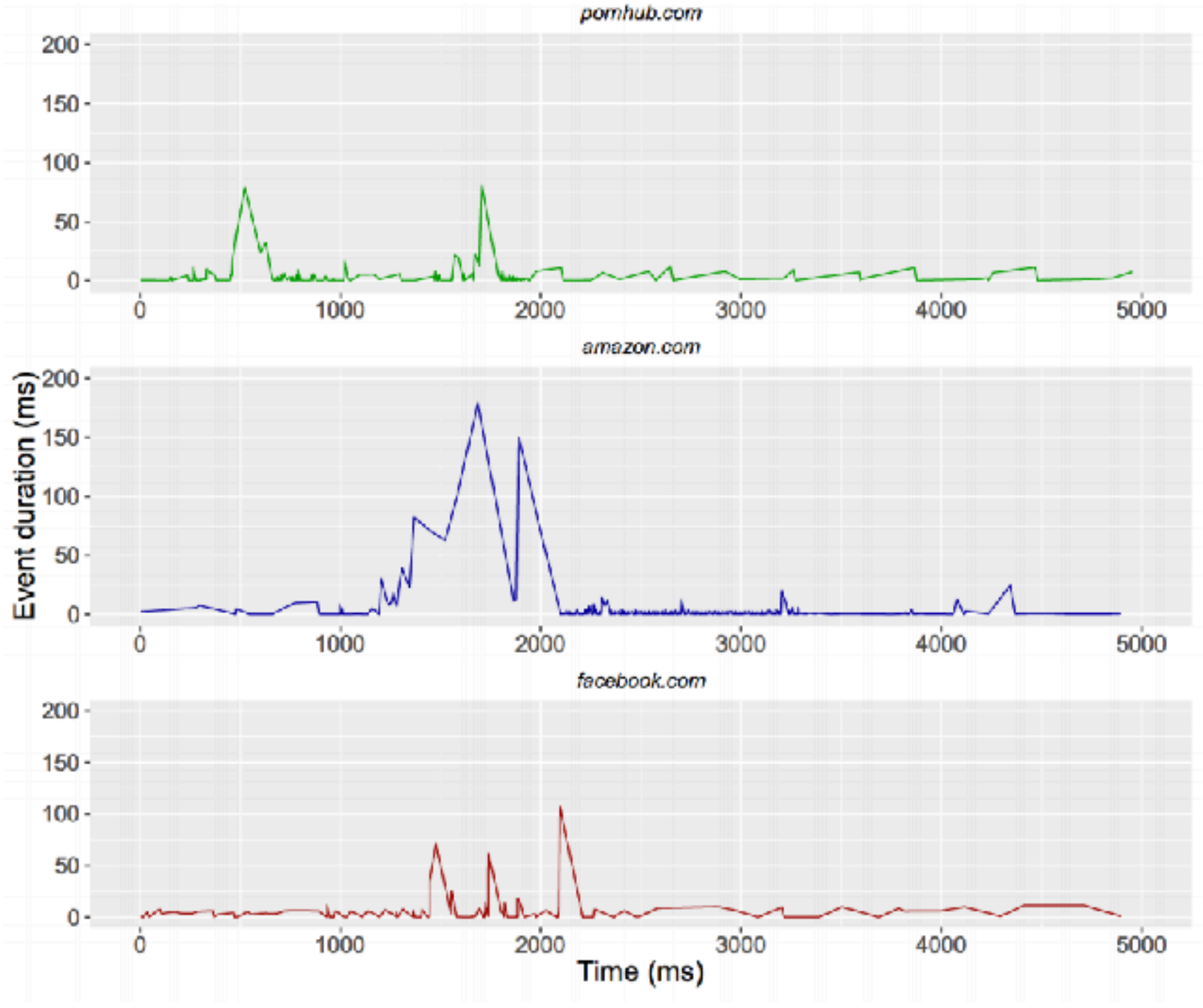
Noise sources:

- Just-in-time (JIT) compilation
- Garbage Collection (GC)
- Thread interleaving
- …

Attacks

- Web page identification
- Inter-keystroke timing information
- Covert channel

Attacks: Web page Identification



Attacks: Web page Identification

Feature extraction + Support Vector Machine (SVM)

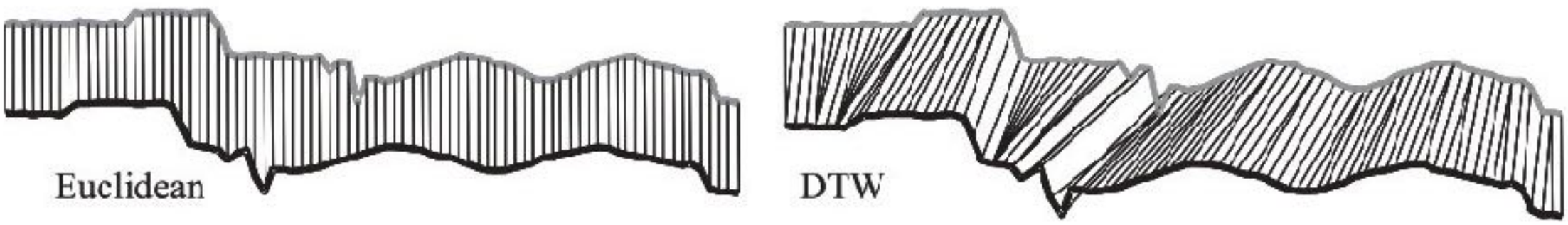
VS.

Dynamic Time Warping (DTW)

Attacks: Web page Identification

DTW distance measure for **time series**: $X = (x_1,\ldots,x_n)$ and $Y = (y_1,\ldots,y_n)$

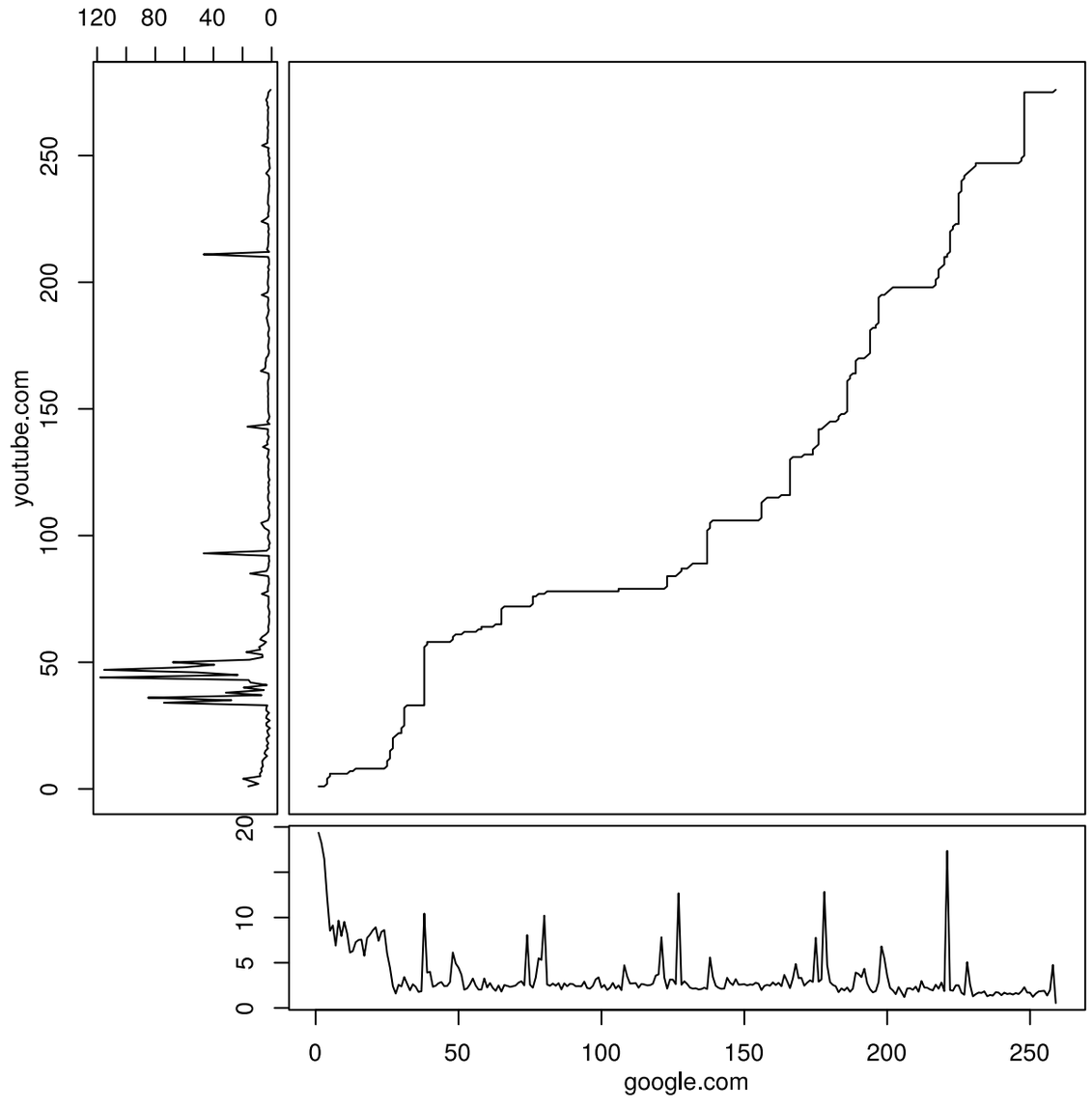
Robust against horizontal compressions and stretches (warping)



Find optimal alignment.

Cost $O(n^2)$ -> Use of constraints

Attacks: Web page Identification

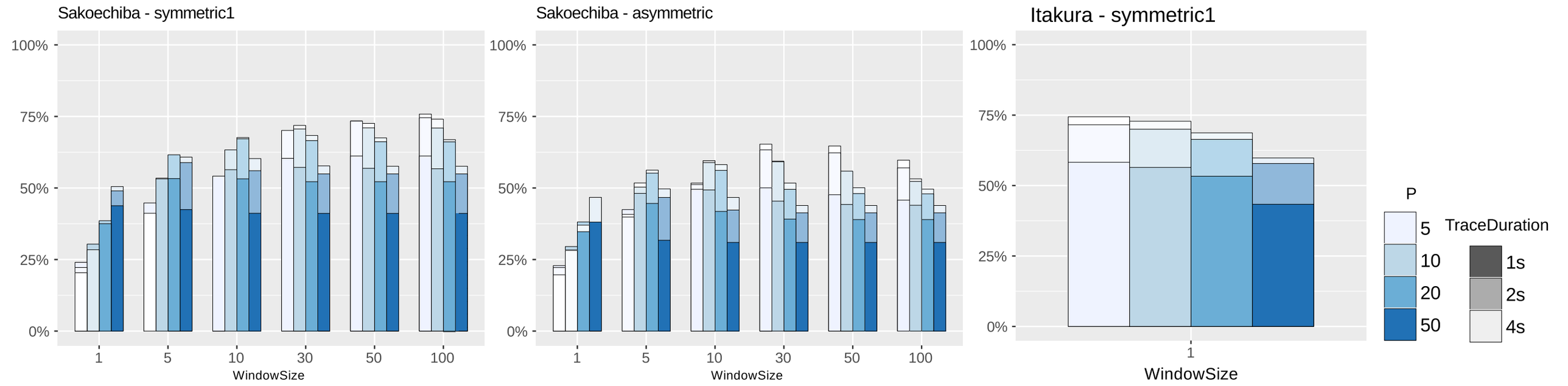


(Warping path between time series from google.com and youtube.com)

Attacks: Web page Identification

Experiments: Alexa's Top 500, 30 traces for each main page during its loading phase, on 2 different machines, with multiple parameters.

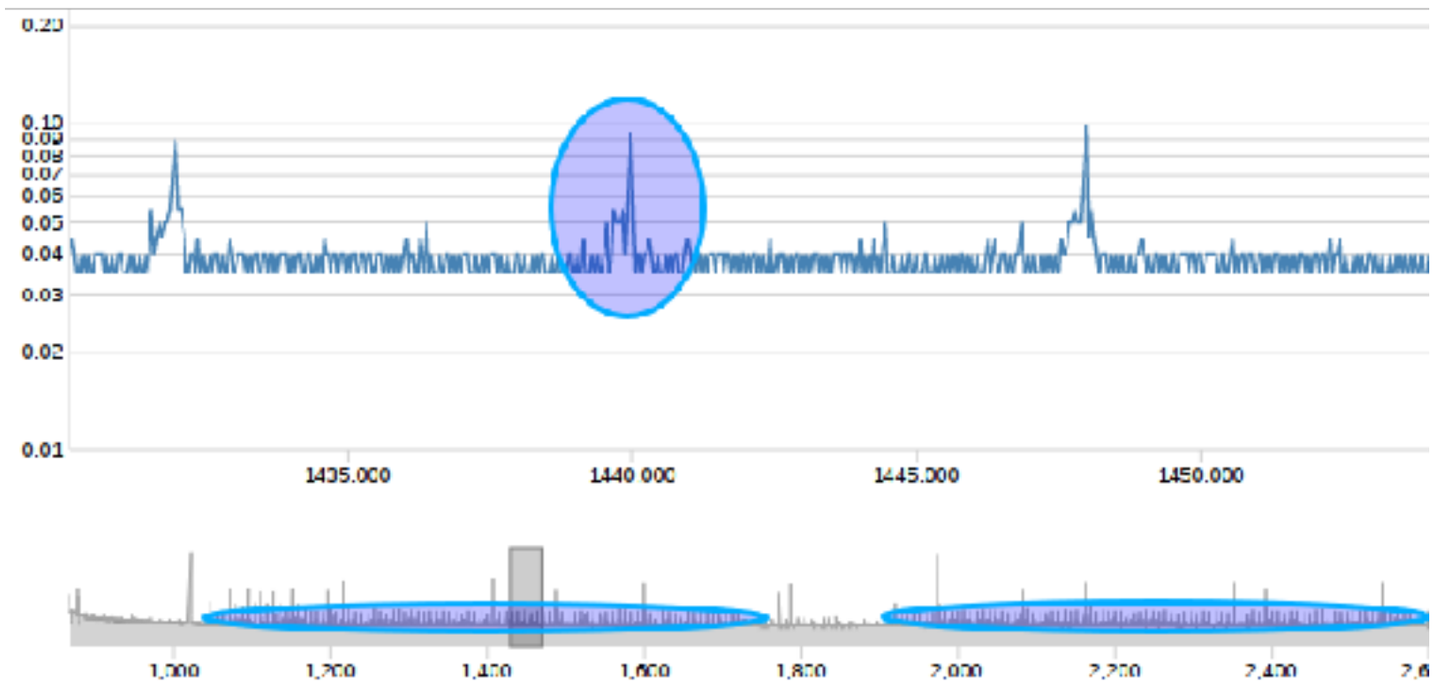
Use ONE single trace of each page as training + 1-NN.



(Extract from tuning results on the renderer's data from a Linux machine)

Attacks: Inter-keystroke timing information

User actions block event loops (even without any explicit JS listener): mouse movement, scrolling, clicks, etc., are generally recognisable:



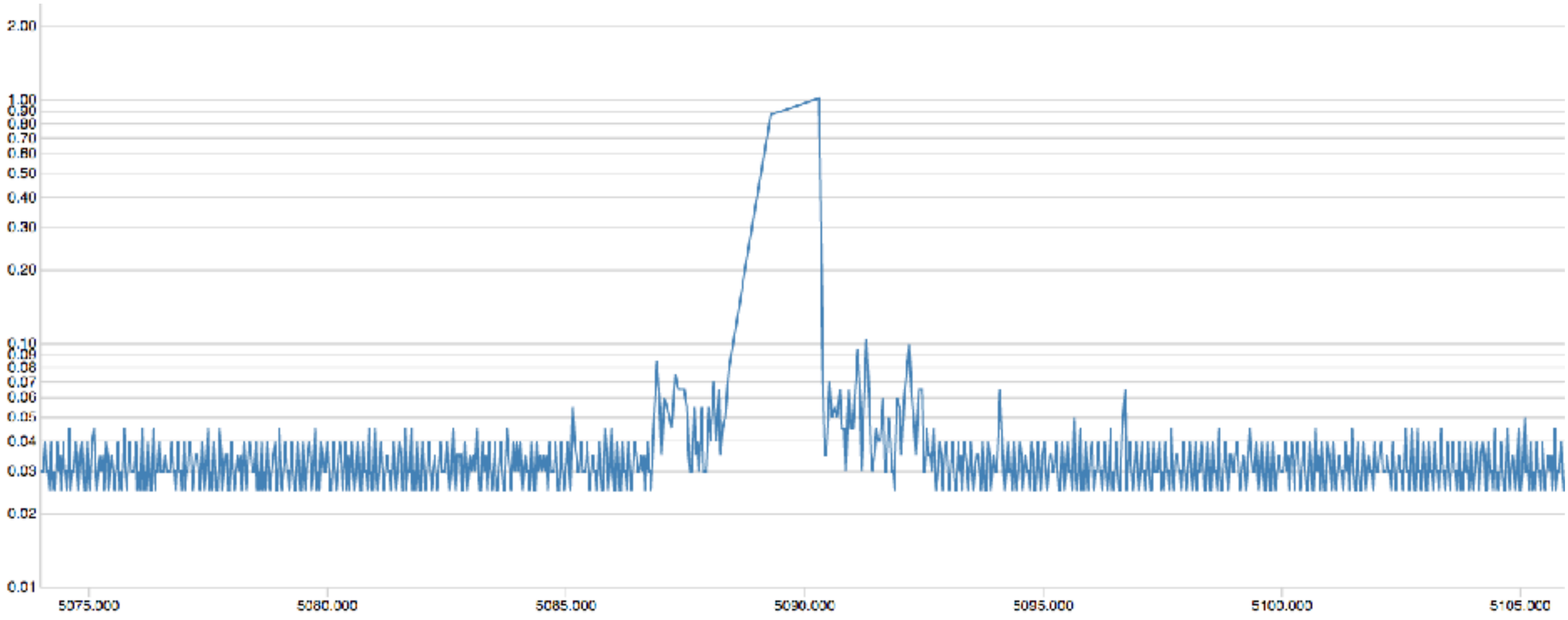
DEMO

Event-delay pattern caused by mouse movement (on a different tab): 0.1ms delay, 125Hz frequency

Specific page's event handlers cause different event-delay patterns.

Attacks: Inter-keystroke timing information

Event-delay pattern of a keystroke in
Google's OAuth login form popup



Keydown Javascript listener followed by *keypress*.

Attacks: Inter-keystroke timing information

Experiment:

- 10.000 passwords from *rockyou.txt*
- emulate keystrokes with random delays (100-300ms)
- get keystroke's timestamps from event-delay trace

Attacks: Inter-keystroke timing information

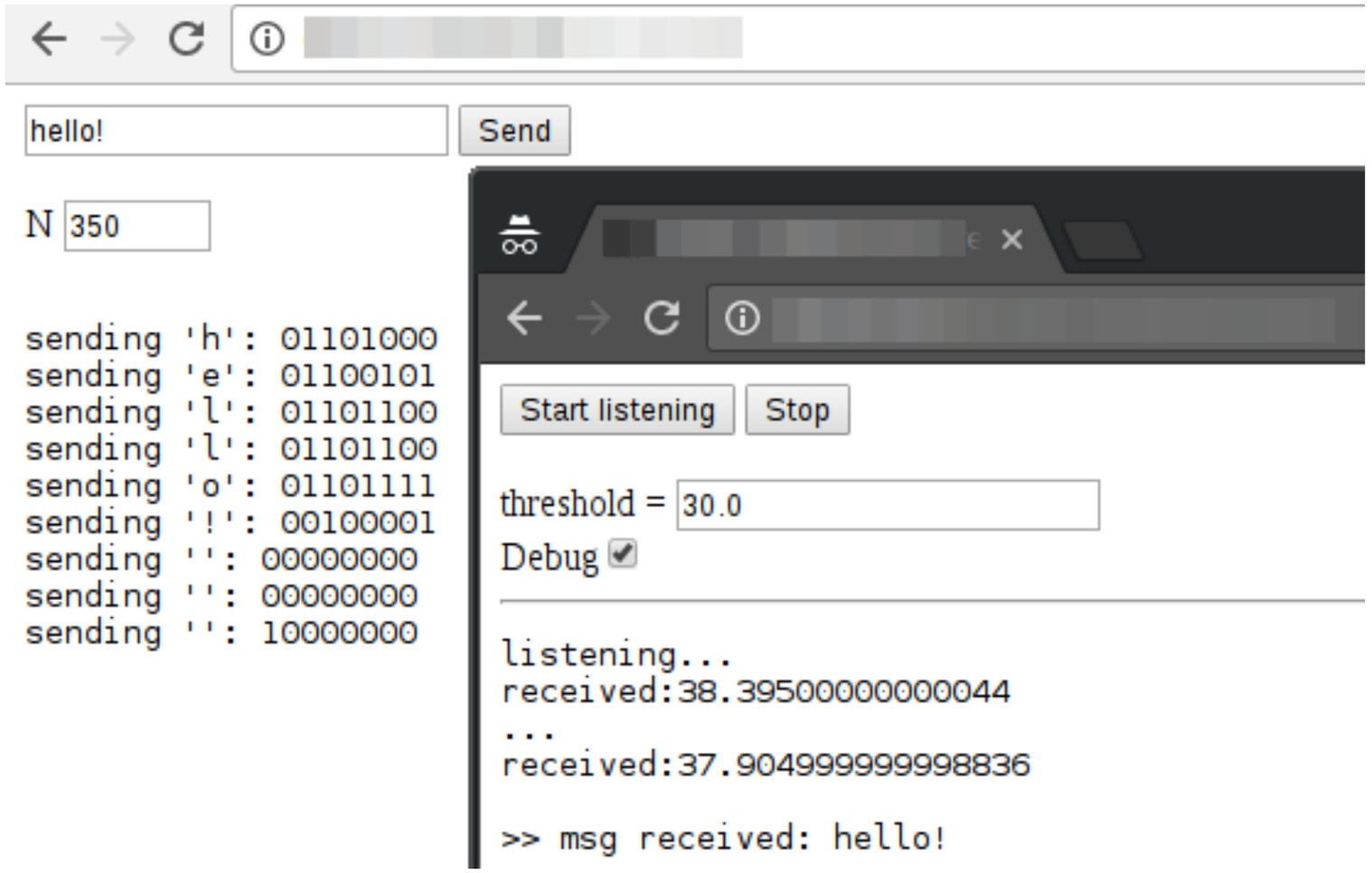
Javascript code to extract keystrokes from a trace:

```
const L = 0.4, U = 3.0, keys = [];

for (let i =1; i< trace.length-1; i++) {
    let d1 = trace[i]-trace[i-1],
        d2 = trace [i+1]-trace[i];

    if (L<d1<U && L<d1<U) {
        keys.push(trace[i]);
    }
}
```

91.5% correct password's length (with 1.5% of false positives)

- 2.2% miss one or more keystrokes
- 6.3% detect spurious keystroke

Attacks: Covert channel

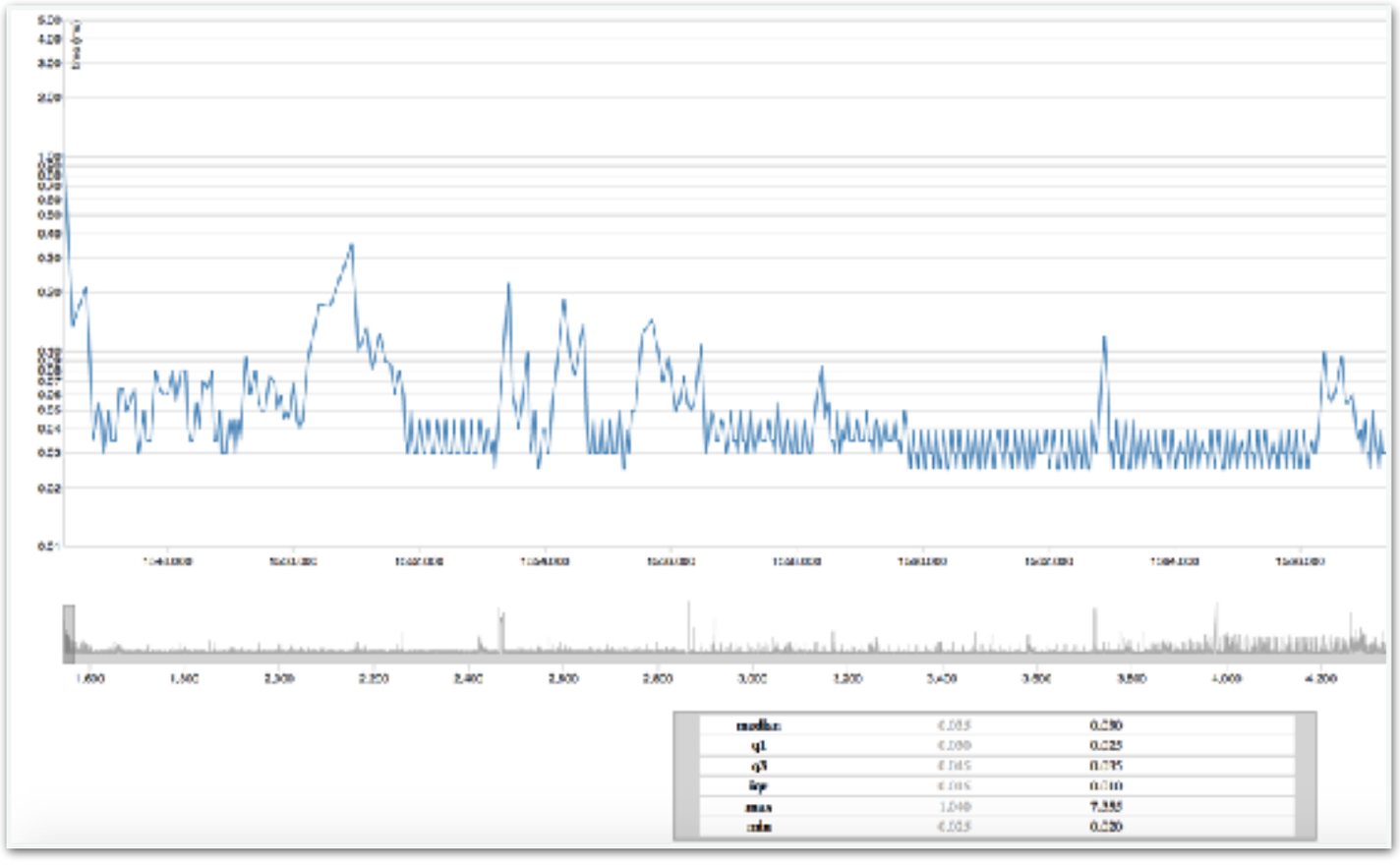


DEMO: https://www.youtube.com/watch?v=IIndCZmRDmI

See https://github.com/cgvwzq/sop-covert-channels for other funny covert channels :D

LoopScan tool

- Simple ~~ugly~~ HTML page for monitoring event loops (with only JS)
- D3.js for interactive visualisations with minimap, zooming and scrolling
- Allows to easily identify event-delay patterns



DEMO

I'll try to publish it soon.

Any volunteer for a logo?

Countermeasures

- Rate Limiting: at which tasks can be posted (reactive detection?)
- Reduce Clock Resolution: useless…
- Full Isolation: see Site Isolation Project
- CPU Throttling: implemented in Chrome 55

Side Channels are usually hard to mitigate without impacting performance.

Future…

Other browsers?

- Firefox implements a different multi-process architecture, but some preliminary experiments indicate a similar behaviour
- Microsoft Edge? Servo? Safari?

Improve attacks and measurements

Different environments?

Thanks. Q?