Detecting Keystrokes in JavaScript

David Bidner, Institute of Applied Information Processing and Communications

Graz, 12th June 2018

Overview

- Side-Channel attacks have been around for years
- Timing Attacks are one of the most prominent of those

Overview

- Side-Channel attacks have been around for years
- Timing Attacks are one of the most prominent of those

We present a timing attack,

- to detect interrupts caused by keystrokes,
- which works on desktop and mobile devices,
- which runs in sandboxed JavaScript.

Content

1. Detecting Keyboard Interrupts in Native Code
2. Detecting Keyboard Interrupts in JavaScript
3. Detecting Input Interrupts on Mobile Devices
4. Countermeasures and Conclusion

Interrupt Timing differences

Calling the interrupt handler takes time.

- We want to detect interruptions of a program.
- For that, we need precise timestamps.
- Keystrokes usually trigger 2 interrupts.
- This only works for PS/2 keyboards.

How do we measure timing accurate enough?

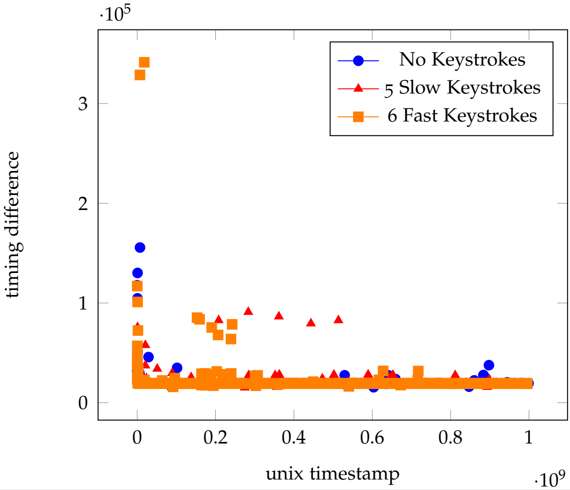
Accurate timings in native Code:

```
a = rdtsc();
b = rdtsc();
diff = b-a;
```

- `diff` is usually around 27 ticks.
- Keystroke interrupt: 80 000 ticks.
- Timer interrupt: 19 000 ticks.

How does `diff` change over time?

Result of the Measurement



Timestamps in JavaScript

We could just use the JavaScript equivalent:

```
a = performance.now();
b = performance.now();
diff = b-a;
```

It's not that easy anymore.

- Browser vendors changed accuracy due to multiple attacks.
- We need to build our own timestamps.

Accurate Timing Measurement in JavaScript

```
results[];
func timing(index)
{
  setTimeout(timing, 0, index+1);
  cnt = 0;
  start = performance.now();
  while((performance.now() - start) < 5)
    cnt++;
  results[index] = cnt;
}
```

Why does this work?

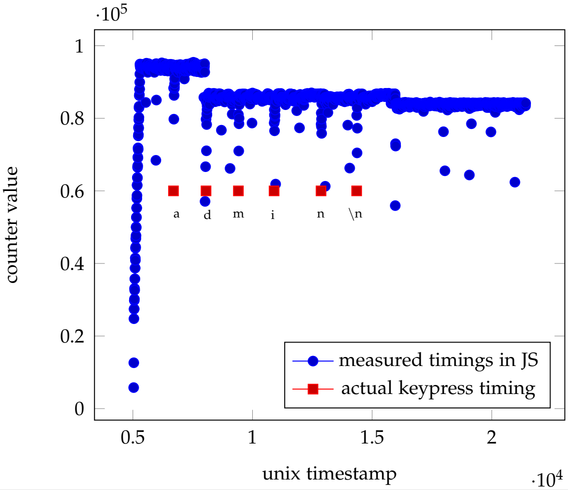
- Minimum `setTimeout` timeout is four milliseconds.
- Function gets called again immediately after a counter cycle finishes.
- A counter value is reported every four milliseconds.
- Allows us to detect if there was an interrupt in this timespan.

How good does it work?

We were using Firefox 52,

- averaging 84 000 increments per four milliseconds,
- equivalent to a resolution of 48 nanoseconds,
- improving other alternative timestamps implementations (50 microseconds)

Result of the Measurement



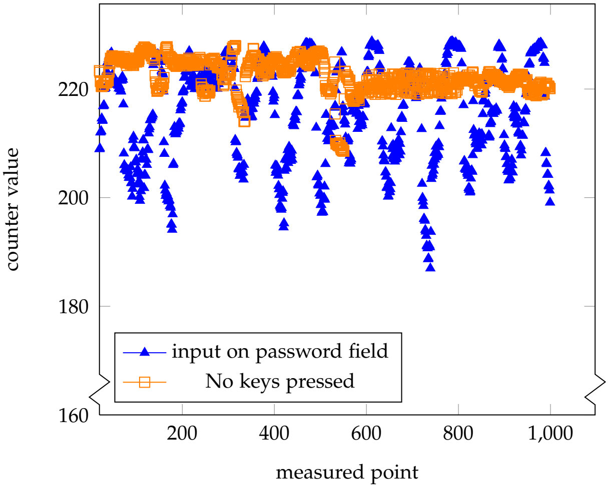
Improving the attack

- Handle interrupts on the same CPU as the program runs
  - use `taskset` and `smp_affinity`
  - otherwise start measurement more often
- Have a constant CPU frequency
  - only relevant for the JavaScript attack

Detecting interrupts on mobile devices?

- Way more interrupts per user interaction.
- Differs for each phone vendor.
- Noisier than on desktop computers.
- Resolution worse because of slower CPU.

Results of the Measurement



Countermeasures

- Inject random Interrupt calls inside the kernel and filter them in the application
- Have a dedicated CPU core to handle interrupts (Co-Processor on mobile)
- Limit JavaScript Permissions

Conclusion

- JavaScript keystroke timing attack.
- Low memory and system load footprint.
- Issue to desktop computers and mobile devices.
- Less accurate timestamps do not matter.
- Prevention only possible with major changes on OS and application level.

Based on this work

a paper was released,

- at ESORICS'17,
- by Moritz Lipp, Daniel Gruss, Michael Schwarz, David Bidner, Clémentine Maurice and Stefan Mangard,
- under the title "Practical Keystroke Timing Attacks in Sandboxed JavaScript".

Detecting Keystrokes in JavaScript

David Bidner, Institute of Applied Information Processing and Communications

Graz, 12th June 2018