Trusted Browsers for Uncertain Times

David Kohlbrenner and Hovav Shacham

UC San Diego
Building a browser that can provably mitigate timing attacks
Trusted Browsers for Uncertain Times

- Time and web browsers
- Mitigating attacks
- A trusted browser
- A (less) trusted browser
Timing attacks

- Time and web browsers
- Mitigating attacks
- A trusted browser
- A (less) trusted browser
Browsers and timing attacks

- Browser has multiple privilege levels
  - User secrets
  - System secrets
  - Origin secrets

- Browsers expose detailed information
  - performance.now()
  - getAnimationFrame()

- Browsers compute and communicate between levels
Timing attacks in web browsers

- SVG Filter cross-origin pixel stealing
- JavaScript cache timing attacks
- Fingerprinting
- History Sniffing
What is being done about it? - SVG attack

Hm. After reducing the if() in the inner loop to

```cpp
if (extrema[i] > pixel) {
    extrema[i] = pixel;
}
```

or

```cpp
if (extrema[i] < pixel) {
    extrema[i] = pixel;
}
```

, the problem boils down to: **how to implement constant-time min(a, b) and max(a, b) in C++?**

volatile? memory barriers? or is this something that should be written in assembly? the problem must have been solved somewhere before...
What is being done about it? - Cache attack

```cpp
// static
double PerformanceBase::clampTimeResolution(double timeSeconds)
{
    const double resolutionSeconds = 0.000005;
    return floor(timeSeconds / resolutionSeconds) * resolutionSeconds;
}

double PerformanceBase::now() const
{
    double nowSeconds = monotonicallyIncreasingTime() - m_timeOrigin;
    return 1000.0 * clampTimeResolution(nowSeconds);
}
```
What is being done about it? - **Cache attack**

diff --git a/dom/base/nsPerformance.cpp b/dom/base/nsPerformance.cpp
index 2cd0aa8..39a213d 100644
--- a/dom/base/nsPerformance.cpp
+++ b/dom/base/nsPerformance.cpp
@@ -424,7 +424,11 @@ nsPerformance::Navigation()
     DOMHighResTimeStamp
     nsPerformance::Now()
     {
-    return GetDOMTiming()\->TimeStampToDOMHighRes(mozilla::TimeStamp::Now());
+    // "Implementations that cannot get the required precision (for example, if
+    // the underlying system doesn't support it) are allowed to only be accurate
+    // to one millisecond."
+    return floor(GetDOMTiming()\->TimeStampToDOMHighRes(mozilla::TimeStamp::Now())/100.0)*100.0;
     }
Unfortunately, this doesn’t work.
Better clocks with edges

- Time and web browsers
- Mitigating attacks
- A trusted browser
- A (less) trusted browser
Rounding down the clock

diff --git a/dom/base/nsPerformance.cpp b/dom/base/nsPerformance.cpp
index 2cd0aa8..39a213d 100644
--- a/dom/base/nsPerformance.cpp
+++ b/dom/base/nsPerformance.cpp
@@ -424,7 +424,11 @@ nsPerformance::Navigation()
     DOMHighResTimeStamp
     nsPerformance::Now()
     {
-     return GetDOMTiming()->TimeStampToDOMHighRes(mozilla::TimeStamp::Now());
+     // "Implementations that cannot get the required precision (for example, if
+     // the underlying system doesn't support it) are allowed to only be accurate
+     // to one millisecond."
+     return floor(GetDOMTiming()->TimeStampToDOMHighRes(mozilla::TimeStamp::Now())/100.0)*100.0;
     }
Clock-edge technique

- Learning Phase:
  - 1 major = 8 minor

- Timing Phase:
  - Target = \((1 + \frac{8-6}{8})\) major
Clock-edge technique - `performance.now()`
Clock-edge technique - `performance.now()`
Implicit clocks in the browser

- Time and web browsers
- Mitigating attacks
- A trusted browser
- A (less) trusted browser
Implicit clocks - Techniques

- `<video>` frames
- Web Speech
- `<video>` played
- `setTimeout()`
- CSS Animations
- WebVTT API
- XHRs with cooperating server

<table>
<thead>
<tr>
<th>Description</th>
<th>Firefox</th>
<th>Chrome</th>
<th>Safari</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicit clocks</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>Video frames</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>Video played</td>
<td>X</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>WebSpeech API</td>
<td>L</td>
<td>+</td>
<td>—</td>
</tr>
<tr>
<td><code>setTimeout()</code></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CSS Animations</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>WebVTT API</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Rate-limited server</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 2: Implicit clock type in different browsers

L: Exitless, X: Exiting, —: Not implemented, +: Buggy
Implicit clocks - Techniques

- `<video>` frames
- Web Speech
- `<video>` played
- `setTimeout()`
- CSS Animations
- WebVTT API
- XHRs with cooperating server

<table>
<thead>
<tr>
<th>Description</th>
<th>Firefox</th>
<th>Chrome</th>
<th>Safari</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicit clocks</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>Video frames</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>Video played</td>
<td>X</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>WebSpeech API</td>
<td>L</td>
<td>+</td>
<td>—</td>
</tr>
<tr>
<td><code>setTimeout</code></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CSS Animations</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>WebVTT API</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Rate-limited server</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 2: Implicit clock type in different browsers
L Exitless, X Exiting, — Not implemented, + Buggy

Probably many many more!
Implicit clocks - WebVTT

- Subtitles for `<video>` elements
- Specified in a `.vtt` file
  - WEBVTT
    - 00:00:00.000 --> 00:00:00.001
      - A very short duration subtitle
  - Specifies arbitrary subtitles with 1ms granularity
- `track.activeCues` returns all displayed subtitles
Implicit clocks - WebVTT
Implicit clocks - WebVTT and clock-edge
How to mitigate timing attacks

- Time and web browsers
- **Mitigating attacks**
- A trusted browser
- A (less) trusted browser
Degrade all clocks available to the attacker.
Fuzzy time for the VAX security kernel

- "[A] collection of techniques that reduces the bandwidths of covert timing channels by making all clocks available to a process noisy."

- "Reducing Timing Channels with Fuzzy Time"
  - Hu at Oakland 1991!
Covert channels

- Two clocks
- Modulated
  - The channel
- Reference
  - Wall clock, etc
Fuzzy time for the VAX security kernel

- VAX VMM
  - Single thread per VM
  - Clean VM interface
- All I/O is asynchronous
Fuzzy time - Problem

- Ineffective countermeasures to disk covert channel
  - Cannot be closed
  - Not auditable
  - Added noise impractical
  - No hardware solution
- Plenty of other potential ‘shared buses’
Fuzzy time - Solution

- “reduce the accuracy and precision of system clocks”
- “randomly alter the timings of I/O operations”
Fuzzy time - Solution

- Explicit clocks
  - “make the interval-timer interrupt random”
Fuzzy time - Solution

- Explicit clocks
  - "make the interval-timer interrupt random"
Fuzzy time - Solution

- Explicit clocks
  - “make the interval-timer interrupt random”

- Implicit clocks
  - “[use] random clock ticks ... to make fuzzy the clocks derived from I/O operations”
  - “Add new buffers ... for all I/O operations”
Fuzzy time - Solution guarantees

- Degraded clocks
  - Limit the bandwidth

- Time granularity
  - $g$

- Bounded channel bandwidth
  - For any timing covert channel
  - $\frac{8}{2}$
Fuzzy time - I/O queuing

Currently queued

Active  Active  Active

Response queue

Next queue

Todo
Fuzzy time - I/O queuing

Currently queued:
- Active
- Active
- Active

Next queue:
- Todo
- Todo

Response queue
Fuzzy time - I/O queuing

**Currently queued**
- Active
- Active
- Active

**Next queue**
- Todo
- Todo

**Response queue**
Fuzzy time - I/O queuing

Currently queued:
- Done
- Active
- Active

Next queue:
- Todo
- Todo

Response queue:
- Todo
Fuzzy time - I/O queuing

Currently queued:
- Done
- Done
- Active

Next queue:
- Todo
- Todo

Response queue:
- Todo
- Todo
Fuzzy time - I/O queuing

Currently queued

Done  Done  Active  Active  Active

Next queue

Todo  Todo

Response queue

Todo  Todo
Fuzzy time - I/O queuing

Currently queued
- Done
- Done
- Active
- Active
- Active

Response queue
- Todo
- Todo

Next queue
Fuzzy time - I/O queuing

Currently queued

Done  Done  Active  Active  Active

Response queue

Todo  Todo

Next queue
Fuzzy time - I/O queuing

Currently queued

- Done
- Done
- Active
- Active
- Active

Response queue

Next queue
Fermata

- Time and web browsers
- Mitigating attacks
- A trusted browser
- A (less) trusted browser
Fermata - Why adapt fuzzy time?

- Degrade clocks
  - Slow down attacks
- Verifiability
- Browsers are uniquely well suited
Fermata - Fuzzy time for browsers

- Adapt the VAX fuzzy time model to JS etc!
- Put all I/O operations into queues
- Make all the explicit clocks fuzzy
- Prove everything falls into a fuzzy time defense

But with JavaScript!
Fermata - Fuzzy time for browsers

- Adapt the VAX fuzzy time model to JS etc!
- Put all I/O operations into queues
- Make all the explicit clocks fuzzy
- Prove everything falls into a fuzzy time defense
- Change all DOM accesses to be asynchronous!
Fuzzyfox

Rationale and design

- Time and web browsers
- Mitigating attacks
- A trusted browser
- A (less) trusted browser
Why we didn’t build Fermata

1. We didn’t know if it would work
2. We didn’t know what to start with
3. We want to push mitigations to real browsers
Fuzzyfox

- Patch set on trunk Mozilla Firefox
- Supports multiple clock granularities
  - Tested 0.5ms to 100ms
- Fully fuzzes explicit clocks
- Breaks main thread into ‘ticks’
- Delays outgoing HTTP request start
Fuzzyfox - Main thread queuing

Current queue

Next queue
Fuzzyfox - Main thread queuing

Current queue:
- Done
- Done
- Active
- Todo
- Todo

Next queue:
- Todo
- Todo
Fuzzyfox - Main thread queuing

Current queue

- Done
- Done
- Active
- Todo
- Todo

Next queue

- Todo
- Todo
- Todo
Fuzzyfox - Main thread queuing

Current queue

Done  Done  Active  Todo  Todo

Next queue

Todo  Todo  Todo  Todo
Fuzzyfox - Main thread queuing

Current queue

Done  Done  Active  Todo  Todo

Next queue

Todo  Todo  Todo
Fuzzyfox - **Main thread queuing**

<table>
<thead>
<tr>
<th>Current queue</th>
<th>Next queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Todo</td>
</tr>
<tr>
<td>Done</td>
<td>Todo</td>
</tr>
<tr>
<td>Active</td>
<td>Todo</td>
</tr>
<tr>
<td>Todo</td>
<td>Todo</td>
</tr>
<tr>
<td>Todo</td>
<td>Todo</td>
</tr>
<tr>
<td>Pause</td>
<td>Todo</td>
</tr>
</tbody>
</table>

The current queue contains tasks that are active and not yet completed. The next queue is ready for tasks to be transferred from the current queue. The pause button indicates that the system is currently in a paused state.
Fuzzyfox - Main thread queuing

Current queue

Done  Done  Active  Todo  Todo  Pause  Todo  Todo  Todo
Fuzzyfox - Main thread queuing

Current queue

Done  Done  Done  Active  Todo  Pause  Todo  Todo  Todo
Fuzzyfox - Main thread queuing

Current queue

Done  Done  Done  Done  Done  Pause  Todo  Todo  Todo  Todo
Fuzzyfox - Main thread queuing
Fuzzyfox - Main thread queuing

Current queue

Done  Done  Done  Done  Done  Pause  Active  Todo  Todo  Todo  Pause
Fuzzyfox - Main thread queuing

Current queue

Done  Done  Done  Done  Pause  Done  Active  Todo  Todo  Pause  Todo
Fuzzyfox - Main thread queuing

Queue 1
- Done
- Done
- Done
- Done
- Done

Queue 2
- Done
- Active
- Todo
- Todo

Queue 3
- Todo

Pause

Pause
Fuzzyfox - Main thread queuing
Current queue

Done  Done  Done  Done  Done  Pause  Done  Active  Todo  Todo  Pause  Todo

Epoch  Epoch  Epoch  Epoch
Fuzzyfox - Main thread queuing

- Sleep
- Update clocks
- Flush queues
- Schedule next pause

Current queue:

- Done
- Done
- Done
- Done
- Pause
- Done
- Active
- Todo
- Todo
- Pause
- Todo

Epochs:
Fuzzyfox

Effectiveness

- Time and web browsers
- Mitigating attacks
- A trusted browser
- A (less) trusted browser
Fuzzyfox - Effectiveness - Explicit - performance.now()
Fuzzyfox - Effectiveness - Implicit - WebVTT clock

Firefox

Fuzzyfox

Clock technique measurement (ms)

Actual target duration (ms)

Clock technique measurement (ms)

Actual target duration (ms)
Fuzzyfox

Performance

- Time and web browsers
- Mitigating attacks
- A trusted browser
- A (less) trusted browser
Fuzzyfox - Performance

- “Micro” performance
  - Synthetic microbenchmark page load times

- “Macro” performance
  - Real website load times

- Interactivity
  - User study
Fuzzyfox - Performance

- “Micro” performance
  - Synthetic microbenchmark page load times

- “Macro” performance
  - Real website load times

- Interactivity
  - User study
Fuzzyfox - Performance - Micro benchmarks

- Page load times
  - As reported by `onload()`

- Measured effects of
  - Sequential resource loads
  - Parallel resource loads
Fuzzyfox - Performance - Sequential loads

The graph shows the 95th percentile loading time (ms) for different sequential loads. The x-axis represents the number of sequential loads, while the y-axis shows the loading time. The legend indicates the load times in milliseconds: 100ms, 50ms, 10ms, 5ms, 1ms, and 0ms. As the number of sequential loads increases, the loading time also increases linearly for each load time scenario.
Fuzzyfox - Performance vs Tor Browser

The diagram shows the 95th percentile loading time (ms) for Tor Browser and 100ms Fuzzyfox as the number of initial concurrent requests increases. The green line represents Tor Browser, while the orange line represents 100ms Fuzzyfox. The y-axis represents the loading time, and the x-axis represents the number of concurrent requests.
Takeaways

- Time and web browsers
- Mitigating attacks
- A trusted browser
- A (less) trusted browser
Timing attacks

Rounding clocks doesn’t work

- Time and web browsers
- Mitigating attacks
- A trusted browser
- A (less) trusted browser
Fuzzy time

Secure operating systems tech

- Time and web browsers
- Mitigating attacks
- A trusted browser
- A (less) trusted browser
Fermata

A different design for the browser

- Time and web browsers
- Mitigating attacks
- A trusted browser
- A (less) trusted browser
Fuzzyfox

Defenses that can work and that we can deploy

- Time and web browsers
- Mitigating attacks
- A trusted browser
- A (less) trusted browser
Takeaways

- Time and web browsers
- Mitigating attacks
- A trusted browser
- A (less) trusted browser

This material is based upon work supported by the National Science Foundation and by a gift from Mozilla. We thank Kyle Huey, Patrick McManus, Eric Rescorla, and Martin Thomson at Mozilla for helpful discussions about this work, and for sharing their insights with us about Firefox internals.
Fuzzyfox - Effectiveness - Explicit - performance.now()
Fuzzyfox - Effectiveness - Implicit - WebVTT clock
Performance - Micro benchmarks - Sequential loads

![Graph showing performance with different load times](image-url)
Performance - Micro benchmarks - Tor Browser
Performance - Load times* - Google search