C++ now

2025

C++ Memory Safety in WebKit

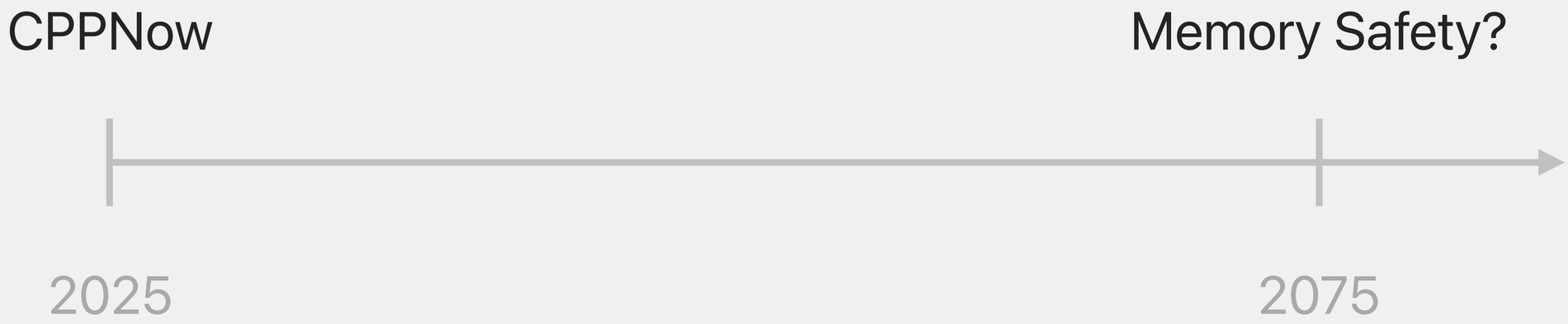
Geoffrey Garen

C++ Memory Safety in WebKit

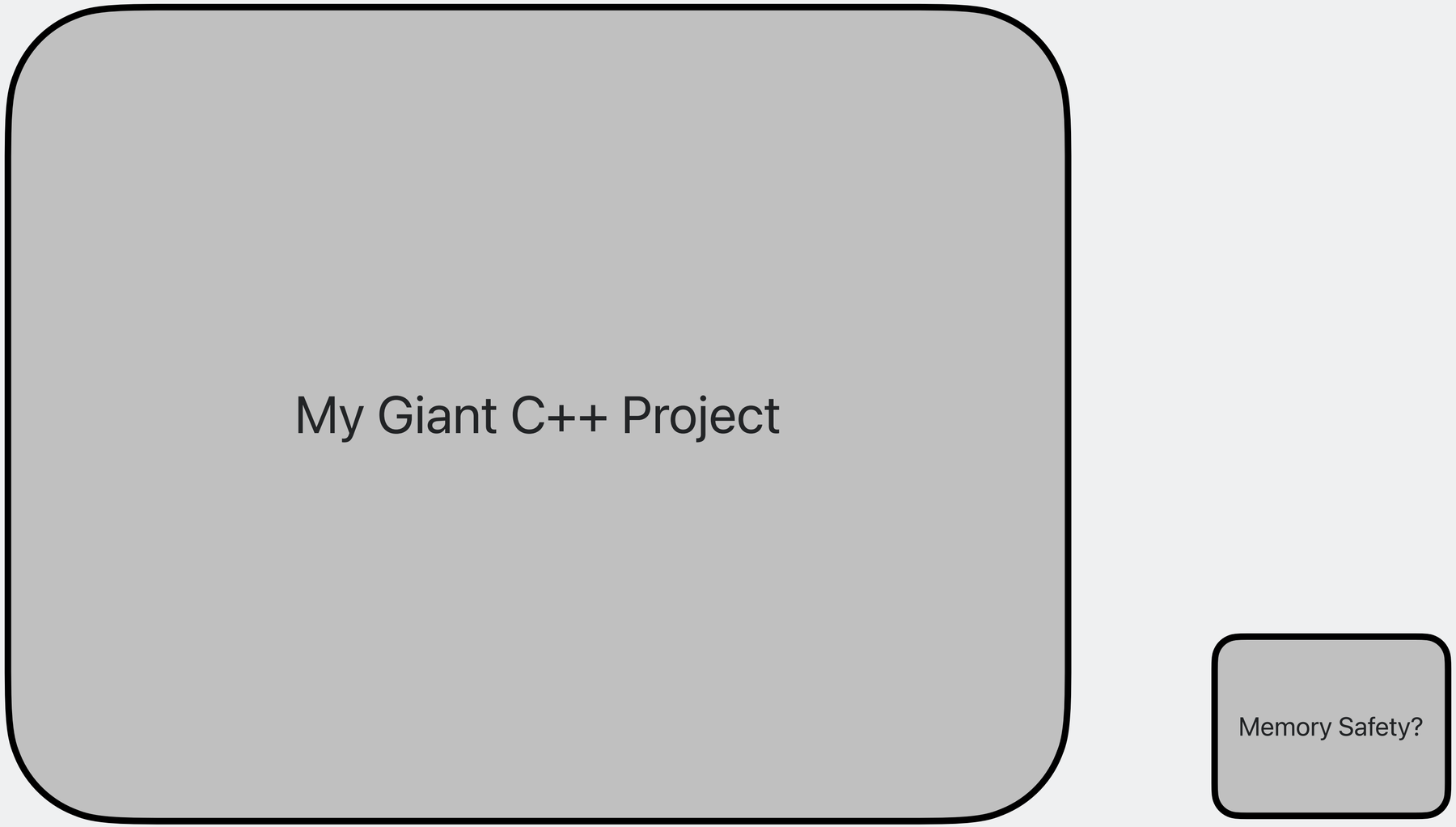
Geoffrey Garen
WebKit Architect at Apple

webkit.slack.com | ggaren

The Giant Rewrite Proposal



The Isolated Component Proposal



WebKit

WebKit

Platforms

- Apple platforms
- Linux
- Windows
- Blender

WebKit

Composition

- 9MM+ LoC
- Mostly C++

WebKit

Goals

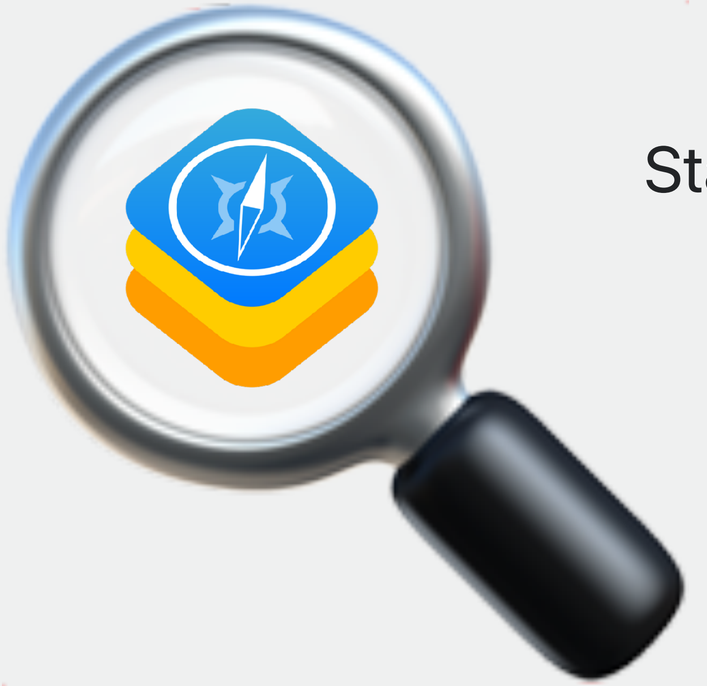
- Security
- Performance

C++ <=> C++

Test Environment

Agenda

Project Maintainer



Standards Participant

Compiler Author

Memory Un-Safety

A bug that can create a weird machine

Memory Un-Safety – Categories

- Pointer Bugs
  - Lifetime
  - Bounds
  - Type

Memory Un-Safety – Categories

- Undefined behavior 👻
  - Null pointer dereference
  - Infinite loop
  - etc. (~175 cases)

Memory Safety – Ideal Definition

!(A bug that can create a weird machine)

Memory Safety – Practical Definition

A programming paradigm that rules out memory un-safety — incrementally, at compile time

Bounds

```
// Unsafe pointer arithmetic

void ShadowBlur::blurLayerImage(
    unsigned char* imageData, const IntSize& size, int rowStride) {
    ...
    limit = (side1 < dim) ? side1 : dim;
    for (i = 0; i < limit; ptr += stride, next += stride, ++i, ++ofs) {
        *ptr = (sum * invCount) >> blurSumShift;
        sum += ((ofs < dim) ? *next : alpha2) - alpha1;
    }
    ...
}
```

```
// Unsafe pointer arithmetic [-Wunsafe-buffer-usage, -Werror]

void ShadowBlur::blurLayerImage(
    unsigned char* imageData, const IntSize& size, int rowStride) {
    ...
    limit = (side1 < dim) ? side1 : dim;
    for (i = 0; i < limit; ptr += stride, next += stride, ++i, ++ofs) {
                               ^^^              ^^^
    Error: unsafe pointer arithmetic [-Wunsafe-buffer-usage, -Werror]

        *ptr = (sum * invCount) >> blurSumShift;
        sum += ((ofs < dim) ? *next : alpha2) - alpha1;
    }
    ...
}
```

```
// Unsafe pointer arithmetic - FIXED

void ShadowBlur::blurLayerImage(
    std::span<char> imageData, const IntSize& size, int rowStride) {
    ...
}
```

```
// Unsafe library function

memcpy(mtlBuffer.contents, ipc, tooManyBytes);
^^^
Error: function 'memcpy' is unsafe [-Wunsafe-buffer-usage-in-libc-call, -Werror]
```

```
// API Boundaries

template<typename T, std::size_t Extent = std::dynamic_extent>
inline constexpr auto unsafeMakeSpan(T* ptr, size_t size)
{
ALLOW_UNSAFE_BUFFER_USAGE_BEGIN
    return std::span<T, Extent> { ptr, size };
ALLOW_UNSAFE_BUFFER_USAGE_END
}

template<typename T, std::size_t TExtent, typename U, std::size_t UExtent>
void memcpySpan(std::span<T, TExtent> destination, std::span<U, UExtent> source)
{
   RELEASE_ASSERT(destination.size() >= source.size());
ALLOW_UNSAFE_BUFFER_USAGE_BEGIN
    memcpy(destination.data(), source.data(), source.size_bytes());
ALLOW_UNSAFE_BUFFER_USAGE_END
}
```

Enforcement

- -Wunsafe-buffer-usage
- -Werror
- CLANG_CXX_STANDARD_LIBRARY_HARDENING = extensive
- #pragma clang diagnostic ignored "-Wunsafe-buffer-usage"

Adoption (1 Year)

94%

Performance Cost

0%

In Benchmarks

End-to-End

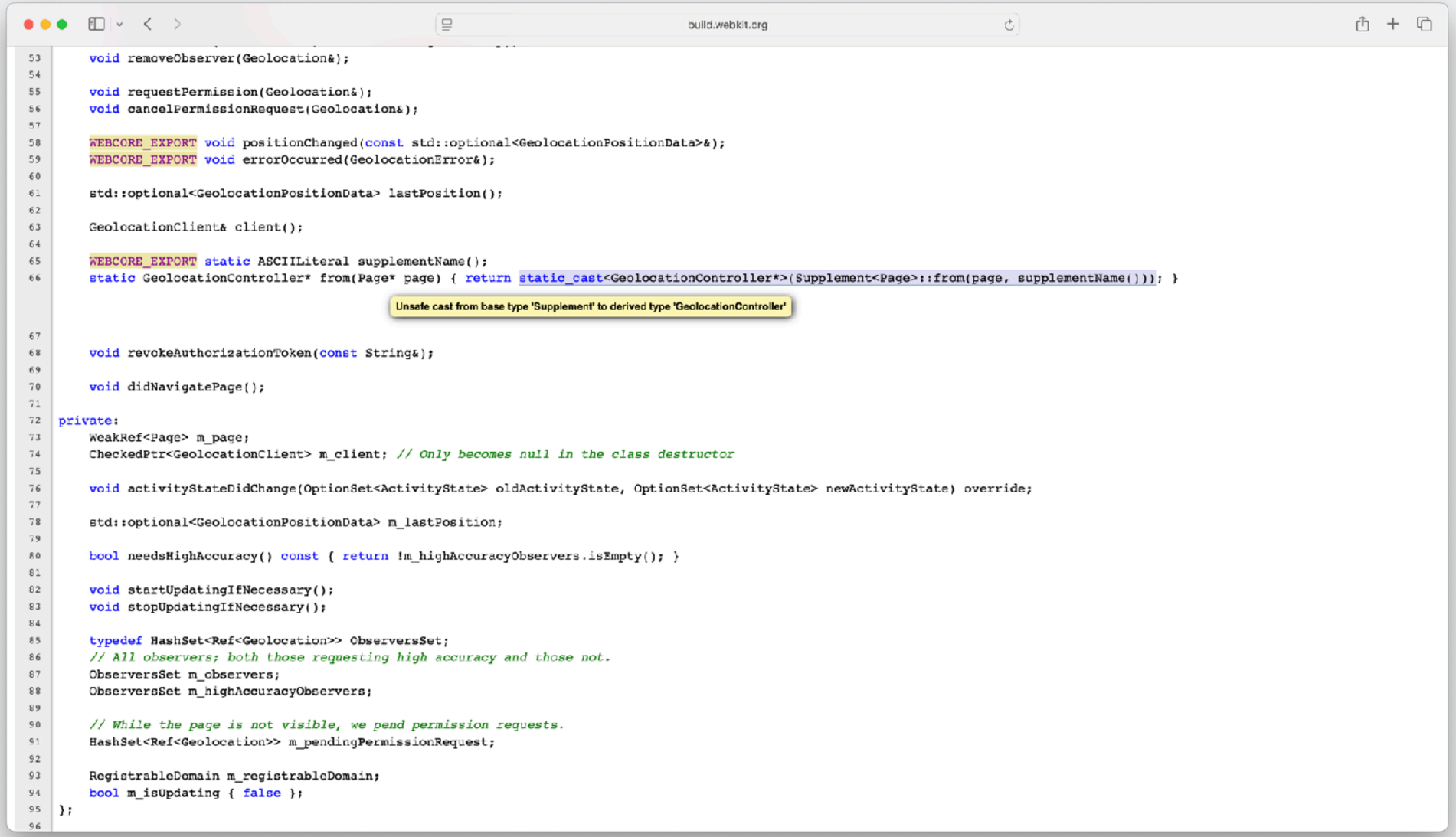
After Optimization

Type

```
// Unsafe Downcast

class ScrollingStateFrameScrollingNode final : public ScrollingStateScrollingNode {...};

auto& frameScrollingNode = static_cast<ScrollingStateFrameScrollingNode>(*node);
```

```
// Unsafe Downcast — FIXED

class ScrollingStateFrameScrollingNode final : public ScrollingStateScrollingNode {...};

auto& frameScrollingNode = downcast<ScrollingStateFrameScrollingNode>(*node);
```

Enforcement




```
53      void removeObserver(Geolocation&);
54
55      void requestPermission(Geolocation&);
56      void cancelPermissionRequest(Geolocation&);
57
58      WEBCORE_EXPORT void positionChanged(const std::optional<GeolocationPositionData>&);
59      WEBCORE_EXPORT void errorOccurred(GeolocationError&);
60
61      std::optional<GeolocationPositionData> lastPosition();
62
63      GeolocationClient& client();
64
65      WEBCORE_EXPORT static ASCIILiteral supplementName();
66      static GeolocationController* from(Page* page) { return static_cast<GeolocationController*>(Supplement<Page>::from(page, supplementName())); }
67
68      void revokeAuthorizationToken(const String&);
69
70      void didNavigatePage();
71
72  private:
73      WeakRef<Page> m_page;
74      CheckedPtr<GeolocationClient> m_client; // Only becomes null in the class destructor
75
76      void activityStateDidChange(OptionSet<ActivityState> oldActivityState, OptionSet<ActivityState> newActivityState) override;
77
78      std::optional<GeolocationPositionData> m_lastPosition;
79
80      bool needsHighAccuracy() const { return !m_highAccuracyObservers.isEmpty(); }
81
82      void startUpdatingIfNecessary();
83      void stopUpdatingIfNecessary();
84
85      typedef HashSet<Ref<Geolocation>> ObserversSet;
86      // All observers; both those requesting high accuracy and those not.
87      ObserversSet m_observers;
88      ObserversSet m_highAccuracyObservers;
89
90      // While the page is not visible, we pend permission requests.
91      HashSet<Ref<Geolocation>> m_pendingPermissionRequest;
92
93      RegistrableDomain m_registrableDomain;
94      bool m_isUpdating { false };
95  };
96
```

Enforcement

- Uses libAnalysis (clang)
- Analyzes the AST (no data flow)
- Enforced in CI (not at desk)
- Enforced incrementally (per-file skip lists and per-line suppress annotations)

Code That's Already "Safe"

```
commit ff08cabb2102fb7af646608a942999bd03f94e44
Author: Chris Dumez <cdumez@apple.com>
Date:   Tue Feb 11 22:07:50 2025 -0800

    Address some of the Safer CPP warnings in DNSResolveQueueCFNet.cpp
    https://bugs.webkit.org/show_bug.cgi?id=287469

    Reviewed by Darin Adler.

-    if (address->sa_family == AF_INET)
-        return WebCore::IPAddress { reinterpret_cast<const sockaddr_in*>(address)-
>sin_addr };
+    if (auto* addressV4 = dynamicCastToIPV4SocketAddress(*address))
+        return WebCore::IPAddress { addressV4->sin_addr };
```

Adoption (1 Year)

98%

Performance Cost

0%

In Benchmarks

End-to-End

After Optimization

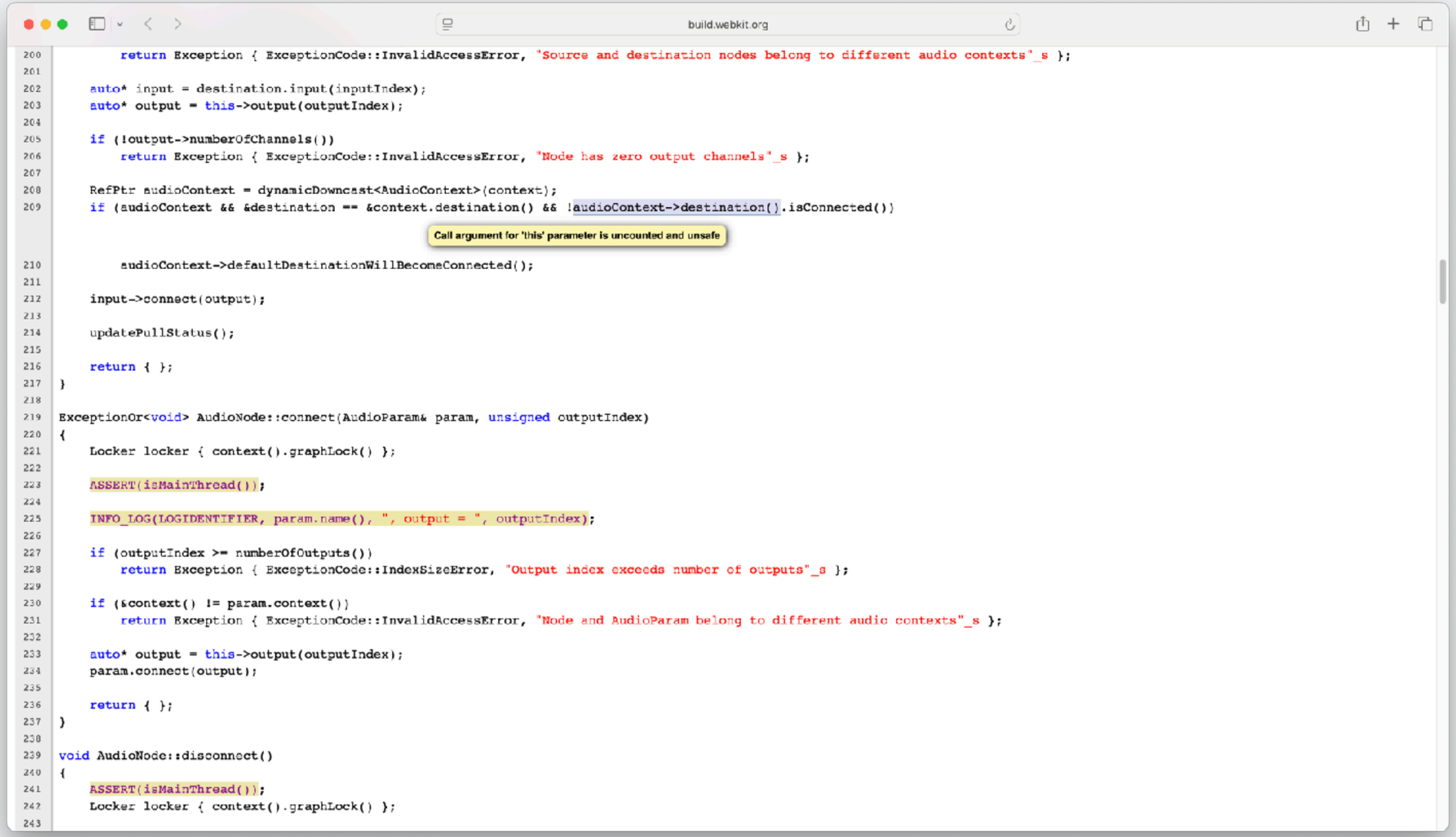
Lifetime (Reference Counted)

```
// Reference Counting Types in WebKit

// Base classes for single-threaded reference counting

template<typename T> class RefCounted;
template<typename T> class CanMakeWeakPtr;

// Smart pointers

template<typename T> class RefPtr;
template<typename T> class WeakPtr;
```

```
// Missing Reference Count

class RTCPeerConnection
    : public RefCounted<RTCPeerConnection>
    , public CanMakeWeakPtr<RTCPeerConnection> { ... };

class RTCIceTransport {
    ...
    WeakPtr<RTCPeerConnection> m_connection;
};

void RTCIceTransport::onStateChanged()
{
    ...
    if (m_connection)
        m_connection->processIceTransportStateChange();
}
```

```
// Missing Reference Count — FIXED

class RTCPeerConnection
    : public RefCounted<RTCPeerConnection>
    , public CanMakeWeakPtr<RTCPeerConnection> { ... };

class RTCIceTransport {
    ...
    WeakPtr<RTCPeerConnection> m_connection;
};

void RTCIceTransport::onStateChanged()
{
    ...
    if (RefPtr connection = m_connection.get())
        connection->processIceTransportStateChange();
}
```

Enforcement




```
200         return Exception { ExceptionCode::InvalidAccessError, "Source and destination nodes belong to different audio contexts"_s };
201
202     auto* input = destination.input(inputIndex);
203     auto* output = this->output(outputIndex);
204
205     if (!output->numberOfChannels())
206         return Exception { ExceptionCode::InvalidAccessError, "Node has zero output channels"_s };
207
208     RefPtr audioContext = dynamicDowncast<AudioContext>(context);
209     if (audioContext && &destination == &context.destination() && !audioContext->destination().isConnected())
                                                Call argument for 'this' parameter is uncounted and unsafe
210         audioContext->defaultDestinationWillBecomeConnected();
211
212     input->connect(output);
213
214     updatePullStatus();
215
216     return { };
217 }
218
219 ExceptionOr<void> AudioNode::connect(AudioParam& param, unsigned outputIndex)
220 {
221     Locker locker { context().graphLock() };
222
223     ASSERT(isMainThread());
224
225     INFO_LOG(LOGIDENTIFIER, param.name(), ", output = ", outputIndex);
226
227     if (outputIndex >= numberOfOutputs())
228         return Exception { ExceptionCode::IndexSizeError, "Output index exceeds number of outputs"_s };
229
230     if (&context() != param.context())
231         return Exception { ExceptionCode::InvalidAccessError, "Node and AudioParam belong to different audio contexts"_s };
232
233     auto* output = this->output(outputIndex);
234     param.connect(output);
235
236     return { };
237 }
238
239 void AudioNode::disconnect()
240 {
241     ASSERT(isMainThread());
242     Locker locker { context().graphLock() };
243
```

```
// Reference Count Overlooking Scope Rule

class Object : public RefCounted<Object> {
public:
    void usePointers(Object*);

private:
    const RefPtr<Object> constMember;
    RefPtr<Object> member;
    Object* rawMember; // ERROR
};

Object* getObject();

extern void use(Object*);
inline void useInline(Object*) { }

void Object::usePointers(Object* argument)
{
    use(argument);
    use(constMember.get());
    use(RefPtr { member }.get());
    use(member.get()); // ERROR
    useInline(member.get());
    use(getObject()); // ERROR
}
```

```
// Reference Count Overlooking Scope Rule

class Object : public RefCounted<Object> {
public:
    void usePointers(Object*);

private:
    const RefPtr<Object> constMember;
    RefPtr<Object> member;
    Object* rawMember; // ERROR
};

Object* getObject();

extern void use(Object*);
inline void useInline(Object*) { }

void Object::usePointers(Object* argument)
{
    RefPtr localMember = member.get();
    if (Object* rawMember = localMember.get())
        use(rawMember);

    localMember = nullptr; // ERROR
}
```

Adoption (1 Year)

90%

Performance Cost

0%

In Benchmarks

End-to-End

After Optimization

Lifetime (Not Reference Counted)

```
// View type use-after-free

auto host = url.host().utf8().data();

                            ^^^ string view
                     ^^^ string temporary
```

```
// View type use-after-free

return stringBuilder.stringView();

                     ^^^ string view
       ^^^ string builder local
```

```
// View type use-after-free - FIXED

class StringBuilder {
    ...
    StringView stringView() const [[clang::lifetimebound]];
    ...
};
```

```
// View type use-after-free - FIXED

return stringBuilder.stringView();
       ^^^
error: address of stack memory associated with local variable
  'stringBuilder' returned [-Werror, -Wreturn-stack-address]
```

Enforcement

- `-Wdangling (enabled by default)`
- `-Werror`
- `[[clang::lifetimebound]]`

Future Direction

- Require `[[clang::lifetimebound]]`
- Expand `-Wdangling`
- Smart pointer for mutation safety

Adoption (1 Year)

TBD%

Performance Cost

0%

Undefined Behavior 👻

The Friendly Neighborhood Ghost

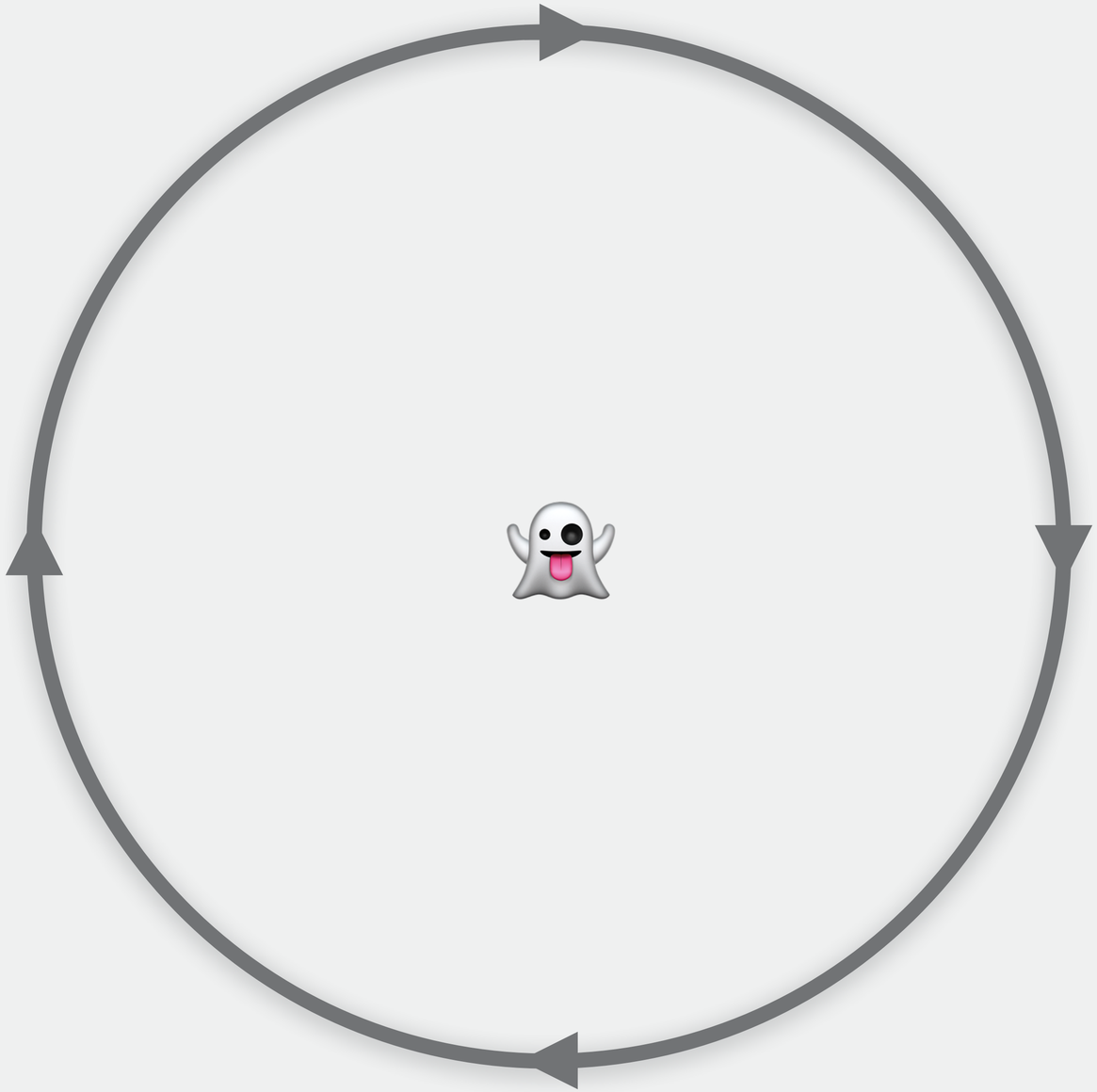
```
RELEASE_ASSERT(destination.size() >= source.size());
```

```
// Infinite loop

uint8_t load(std::span<uint8_t> input, size_t index)
{
    if (index >= input.size()) {
        while (1) { // 👻
        }
    }
    return input[index]; // UH OH
}
```

```
// Null pointer dereference

WeakPtr passwordPopup = getPasswordPopup();
if (shouldShowPasswordPopup())
    RefPtr { passwordPopup.get() /* 👻 */ }->show(); // UH OH
```

{ Redacted }

Memory Un-Safety



Static Analysis

Swift

- Direct C++ integration
- Shared concepts
- Strict Memory Safety Checking

Shared Concepts

Safety Issue	Concept	Annotation
Bounds	Bounds-Checked Collection and View Types	N/A
Type	Checked Pointer Casts	N/A
Lifetime (Reference Counted)	Overlooking Reference Count	SWIFT_SHARED_REFERENCE
Lifetime (Not Reference Counted)	Lifetime Dependency	SWIFT_NONESCAPABLE [[clang::lifetimebound]]
Undefined Behavior	“If it’s not safe, it shouldn’t compile”	N/A

```
// Lifetime (Reference Counted)

// Objective-C++

template<typename T> class RefCounted { ... } SWIFT_SHARED_REFERENCE(ref, deref);

class Texture : public RefCounted<Texture> {
    ...
    id<MTLTexture> texture() const { return m_texture; }
};

// Swift

let texture = fromAPI(attachment.view)
    ^^^ C++ reference count
...
let mtlTexture = texture.texture()
    ^^^ Objective-C reference count
```

```
// Lifetime (Reference Counted)

// Objective-C++

template<typename T> class RefCounted { ... };

class Texture : public RefCounted<Texture> {
    ...
    id<MTLTexture> texture() const { return m_texture; }
};

// Swift

let texture = fromAPI(attachment.view)
    ^^^ error: expression uses unsafe constructs but is not marked with 'unsafe'
...
let mtlTexture = texture.texture()
    ^^^ Objective-C reference count
```

```
// Lifetime (Not Reference Counted)

// C++

class SWIFT_NONESCAPABLE StringView { ... };

class StringBuilder {
    ...
    StringView stringView() const [[clang::lifetimebound]];
    ...
};

// Swift

return stringBuilder.stringView();
                     ^^^ error: lifetime-dependent value escapes local scope
```

```
// Lifetime (Not Reference Counted)

// C++

class SWIFT_NONESCAPABLE StringView { ... };

class StringBuilder {
    ...
    StringView stringView() const;
    ...
};

// Swift

return stringBuilder.stringView();
                     ^^^ error: lifetime-dependent value is missing a lifetime annotation
```

```
// Lifetime (Not Reference Counted)

// C++

class StringView { ... };

class StringBuilder {
    ...
    StringView stringView() const;
    ...
};

// Swift

return stringBuilder.stringView();
^^^ error: expression uses unsafe constructs but is not marked with 'unsafe'
```

Summary

Summary

- Incremental progress toward memory safety can start now, everywhere
- The end-to-end performance cost in WebKit has been zero
- Undefined behavior haunts C++ 👻
- You can help!
- A memory safe language can be a “Yes! And…”