RLBox: Retrofitting Fine Grain Isolation in the Firefox Renderer

Shravan Narayan, Craig Disselkoen, Tal Garfinkel, Nathan Froyd, Eric Rahm,
Sorin Lerner, Hovav Shacham, Deian Stefan

Oct 2020: Google patches zero-day in Chrome

Author:
Elizabeth Montalbano

October 21, 2020
/ 8:23 am

1:30 minute read

The memory-corruption vulnerability exists in the browser's FreeType font rendering library.

Google released an update to its Chrome browser that patches a zero-day vulnerability in the software's FreeType font rendering library that was actively being exploited in the wild.

Security researcher Sergei Glazunov of Google Project Zero discovered the bug which is classified as a type of memory-corruption flaw called a heap buffer overflow in FreeType. Glazunov informed Google of the vulnerability on Monday. Project Zero is an internal security team at the company aimed at finding zero-day vulnerabilities.

The attack campaign

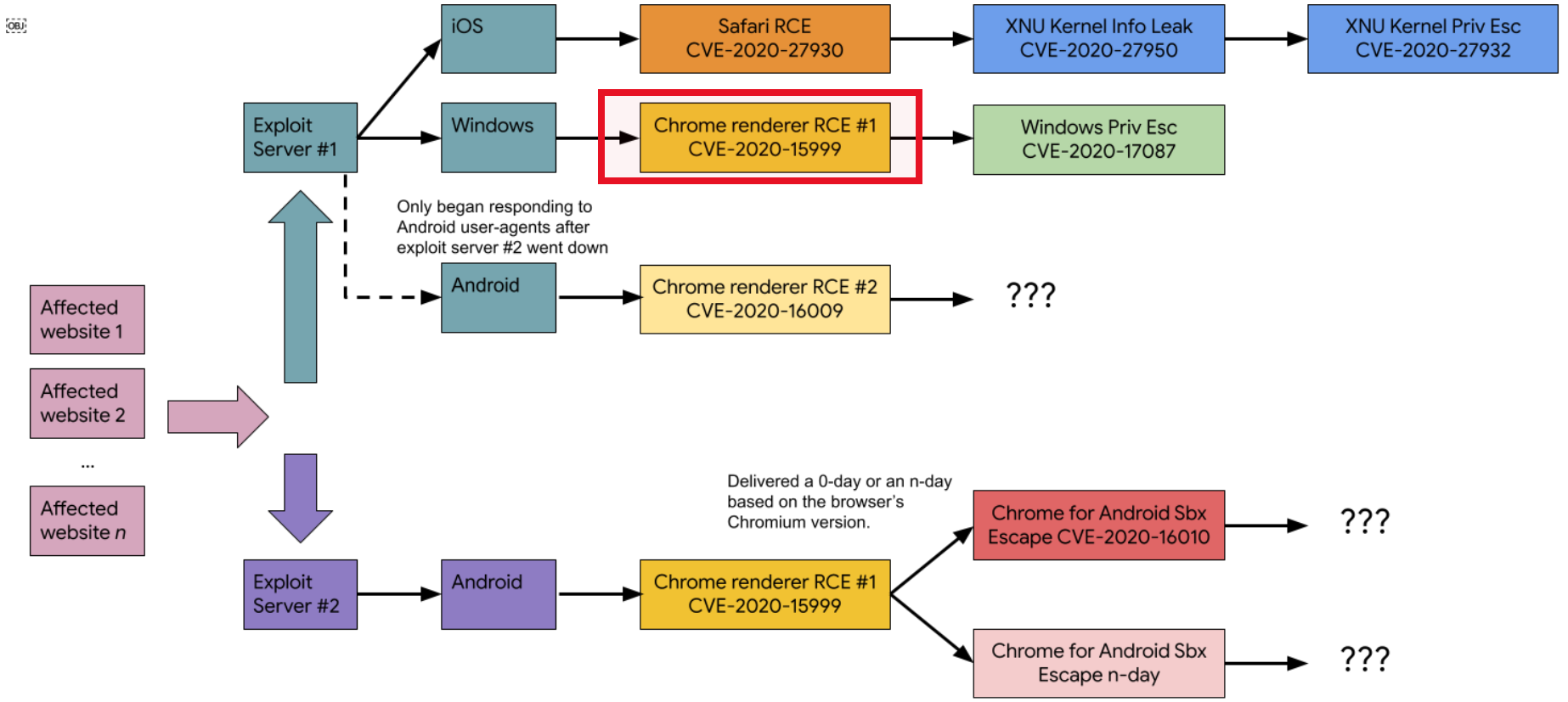
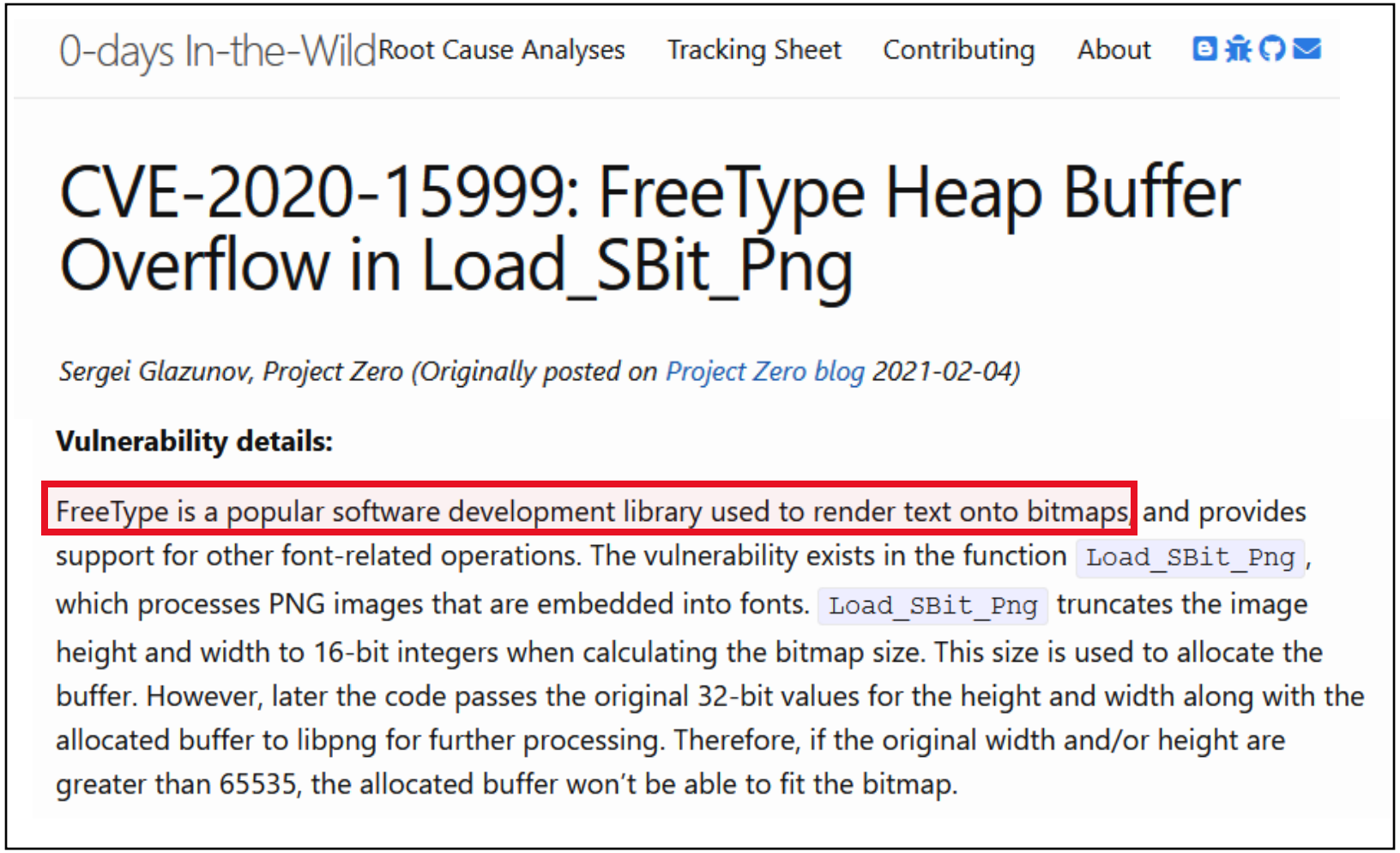


Image from https://googleprojectzero.blogspot.com/2021/03/in-wild-series-october-2020-0-day.html

Key step: vulnerability in a third-party library



0-days In-the-Wild Root Cause Analyses Tracking Sheet Contributing About

CVE-2020-15999: FreeType Heap Buffer Overflow in Load_SBit_Png

Sergei Glazunov, Project Zero (Originally posted on Project Zero blog 2021-02-04)

Vulnerability details:

FreeType is a popular software development library used to render text onto bitmaps, and provides support for other font-related operations. The vulnerability exists in the function `Load_SBit_Png`, which processes PNG images that are embedded into fonts. `Load_SBit_Png` truncates the image height and width to 16-bit integers when calculating the bitmap size. This size is used to allocate the buffer. However, later the code passes the original 32-bit values for the height and width along with the allocated buffer to libpng for further processing. Therefore, if the original width and/or height are greater than 65535, the allocated buffer won't be able to fit the bitmap.

This is not a one off

From Pearl to Pegasus

Bahraini Government Hacks Activists with NSO Group Zero-Click iPhone Exploits

By Bill Marczak, Ali Abdulemam[1], Noura Al-Jizawi, Siena Anstis, Kristin Berdan, John Scott-Railton, and Ron Deibert
[1] Red Line for Gulf

August 24, 2021

Phone logs show that (at least some of) the iOS 13.x and 14.x zero-click exploits deployed by NSO Group involved ImageIO, specifically the parsing JPEG and GIF images. ImageIO has had more than a dozen high-severity bugs reported against it in 2021.

Issue 1196480: Security: Multiple Bugs in WebP
Reported by awhalley@google.com on Tue, Apr 6, 2021, 5:21 PM PDT Project Member

Comment 12 by cthomp@chromium.org on Wed, Apr 7, 2021, 12:22 PM PDT Project Member

I've filed child bugs to track each of the four security issues identified in the report:

(1) ~~Issue 1196773~~: Security: heap-use-after-free in libwebp ConvertBGRAToRGB_SSE41
(2) ~~Issue 1196775~~: Security: heap-buffer-overflow in libwebp PlanarTo24b_SSE41
(3) ~~Issue 1196777~~: Security: heap-buffer-overflow in libwebp VP8YuvToRgb
(4) ~~Issue 1196778~~: Security: heap-buffer-overflow in libwebp UpsampleRgbLinePair_SSE41

CVE-2018-5146: Out of bounds memory write in libvorbis

Reporter Richard Zhu via Trend Micro's Zero Day Initiative

Impact critical

Description

An out of bounds memory write while processing Vorbis audio data was reported through the Pwn2Own contest.

Issue 2161: QT: out-of-bounds read in TIFF processing
Reported by natashenka@google.com on Tue, Feb 23, 2021, 4:08 PM PST Project Member

The QImageReader class can read out-of-bounds when converting a specially-crafted TIFF file

Vulnerability Details : CVE-2021-37972

Out of bounds read in libjpeg-turbo in Google Chrome prior to 94.0.4606.54 allowed a remote attacker to potentially exploit heap corruption via a crafted HTML page.
Publish Date : 2021-10-08 Last Update Date : 2021-10-10

Available for: iPhone 5s, iPhone 6, iPhone 6 Plus, iPad Air, iPad mini 2, iPad mini 3, and iPod touch (6th generation)

Impact: Processing a maliciously crafted PDF may lead to arbitrary code execution. Apple is aware of a report that this issue may have been actively exploited.

Description: An integer overflow was addressed with improved input validation.

CVE-2021-30860: The Citizen Lab

Our work: fine grain library isolation in Firefox

Idea: Isolate each library in its own memory sandbox

- Use software-based fault isolation (SFI) to do this via runtime checks

Problem: In practice, SFI does not work

- (Near) impossible to retrofit SFI in large systems
- Incomplete: applications can be compromised by trusting library output

Solution: RLBox sandboxing framework

- Uses types to retrofit isolation
- Deployed in the real world

Today

1. Why fine grain isolation?
2. Why is SFI (alone) not the answer?
3. The RLBox framework
4. What is the cost of RLBox?

Today

1. Why fine grain isolation? 
2. Why is SFI (alone) not the answer?
3. The RLBox framework
4. What is the cost of RLBox?

Browsers use a lot of third-party libraries

https://source.chromium.org/chromium/chromium/src/+/main:third_party/

- Python-Markdown/
- abseil-cpp/
- accessibility-audit/
- accessibility_test_framework/
- afl/
- android_build_tools/
- android_crazy_linker/
- android_data_chart/
- android_deps/
- android_deps_autorolled/
- android_media/
- android_ndk
- android_opengl/
- android_platform/
- android_protobuf/
- android_protoc/
- android_provider/
- android_rust_toolchain/
- android_sdk/
- android_support_test_runner/
- android_swipe_refresh/
- android_system_sdk/
- androidx/
- angle
- apache-portable-runtime/
- apache-win32/
- apple_apsl/
- arcore-android-sdk/
- arcore-android-sdk-client/
- ashmem/
- axe-core/
- blanketjs/
- blink/
- boringssl/
- bouncycastle/
- breakpad/
- brotli/
- bspatch/
- byte_buddy/
- cast_core/
- catapult
- ced/
- chaijs/
- checkstyle/
- chevron/
- chromevox/
- chromite
- cld_3/
- cldr/
- closure_compiler/
- colorama/
- crashpad/
- crc32c/
- cros_system_api
- d3/
- dav1d/
- dawn
- decklink/
- depot_tools
- devscripts/
- devtools-frontend/
- distributed_point_functions/
- dom_distiller_js/
- dpkg-shlibdeps/
- eigen3/
- emoji-metadata/
- emoji-segmenter/
- espresso/
- expat/
- farmhash/
- fdlibm/
- ffmpeg
- fft2d/
- flac
- flatbuffers/
- flex/
- fontconfig/
- fp16/
- freetype/
- freetype-testing/
- fuchsia-sdk/
- fusejs/
- gemmlowp/
- gif_player/
- glfw/
- glide/
- gnu_binutils
- google-closure-library/
- google-truth/
- google_benchmark/
- google_input_tools/
- google_toolbox_for_mac/
- google_trust_services/
- googletest/
- gperf
- gradle_wrapper/
- grpc/
- gvr-android-keyboard/
- gvr-android-sdk/
- hamcrest/
- harfbuzz-ng/
- highway/
- hunspell/
- hunspell_dictionaries
- hyphenation-patterns/
- iaccessible2/
- iccjpeg/
- icu
- icu4j/
- ijar/
- inspector_protocol/
- instrumented_libraries/
- isimpledom/
- jacoco/
- javalang/
- jdk/
- jinja2/
- js_code_coverage/
- jsoncpp/
- jstemplate/
- junit/
- khronos/
- lcov/
- leveldatabase/
- libFuzzer/
- libXNVCtrl/
- libaddressinput/
- libaom/
- libavif/
- libbrlapi/
- libdrm/
- libgav1/
- libgifcodec
- libipp/
- libjingle_xmpp/
- libjpeg_turbo
- libjxl/
- liblouis/
- libphonenumber/
- libpng/
- libprotobuf-mutator/
- libsecret/
- libsrtp
- libsync/
- libudev/
- libunwindstack
- liburlpattern/
- libusb/
- libva_protected_content/
- libvpx/
- libwebm/
- libwebp/
- libx11/
- libxcb-keysyms/
- libxml/
- libxslt/
- libyuv
- libzip/
- lighttpd
- logdog/
- logilab/
- lottie/
- lss
- lzma_sdk/
- mako/
- maldoca/
- markdown/
- markupsafe/
- material_design_icons/
- material_web_components/
- mesa_headers/
- metrics_proto/
- microsoft_webauthn/
- mig/
- mingw-w64/
- minigbm/
- minizip/
- mocha/
- mockito/
- modp_b64/
- motemplate/
- mozilla/
- nacl_sdk_binaries
- nasm
- nearby/
- neon_2_sse/
- netty-tcnative/
- netty4/
- node/
- objenesis/
- ocmock/
- one_euro_filter/
- opencv/
- openh264/
- openscreen/
- openxr/
- opus/
- ots/
- ow2_asm/
- pdfium
- pefile
- pefile_py3/
- perfetto
- perl
- pexpect/
- pffft/
- ply/
- polymer/
- private-join-and-compute/
- private_membership/
- proguard/
- protobuf/
- pycoverage/
- pyelftools
- pyjson5/
- pylint/

https://searchfox.org/mozilla-central/source/

mozilla-central / media

Name

- ffvpx
- gmp-clearkey
- highway
- kiss_fft
- libaom
- libcubeb
- libdav1d
- libjpeg
- libjxl
- libmkv
- libnestegg
- libogg
- libopus
- libpng
- libsoundtouch
- libspeex_resampler
- libtheora
- libtremor
- libvorbis
- libvpx
- libwebp
- libyuv

mozilla-central / third_party

Name

- aom
- cups
- dav1d
- highway
- intgemm
- jpeg-xl
- js
- libsrtp
- libwebrtc
- moz.build
- msgpack
- pipewire
- prio
- python

Bug in any third-party library ⇒ security vulnerability

Most of these are memory-safety bugs

Google: ~70% of Chrome's bugs (2015–2020)

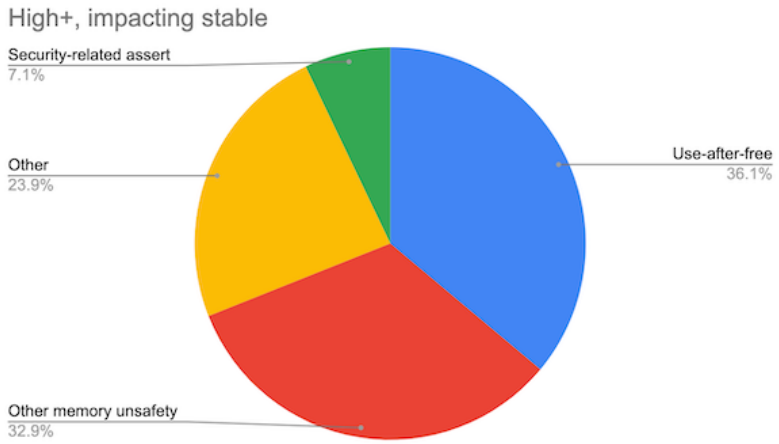


Image from the Chromium project blog
https://www.chromium.org/Home/chromium-security/memory-safety/

Microsoft: ~70% of Windows' bugs (2006–2018)

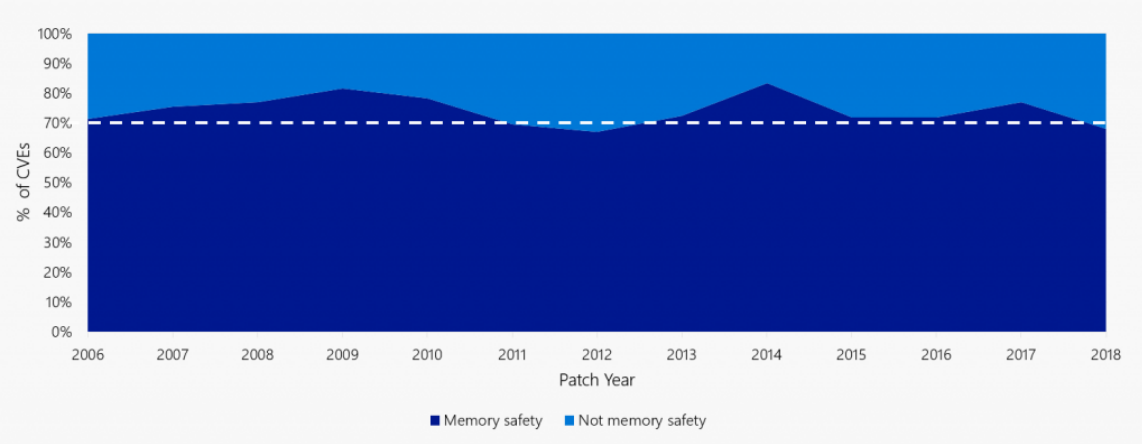


Image from the Microsoft security response center blog
https://msrc-blog.microsoft.com/2019/07/16/a-proactive-approach-to-more-secure-code/

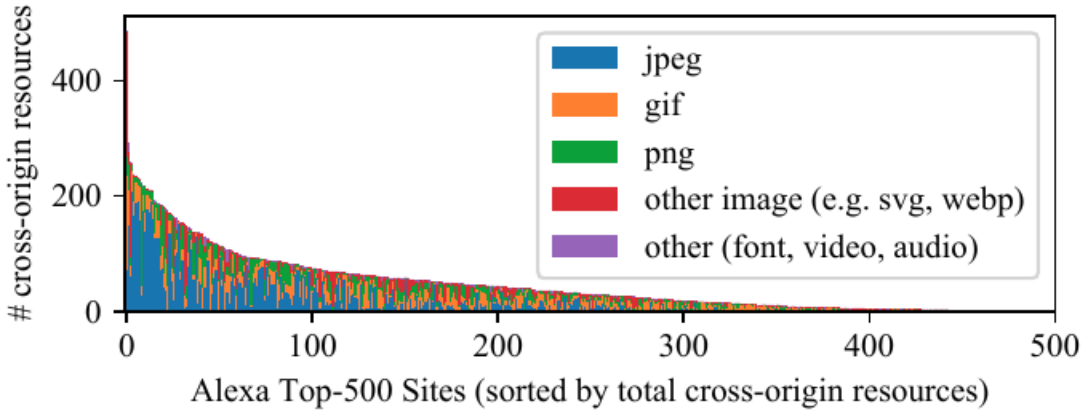
Existing browser mechanisms are too coarse grained

Browser isolate code at the boundary of a site

- Isolate `google.com` from `facebook.com`

Problem: in practice pages include cross site data

- 93% of Alexa top 500 sites load cross-site media like JPEGs



What else can browsers do?

Rewrite code in memory-safe Rust?

Lots of legacy code

Significant effort: rewrite, retest

"As a practical matter [...], we're going to be living with software with memory safety issues for quite some time"

- Eric Rescorla
CTO, Firefox

Put libraries in a separate process?

Memory and context switch overheads

Significant browser re-architecting effort

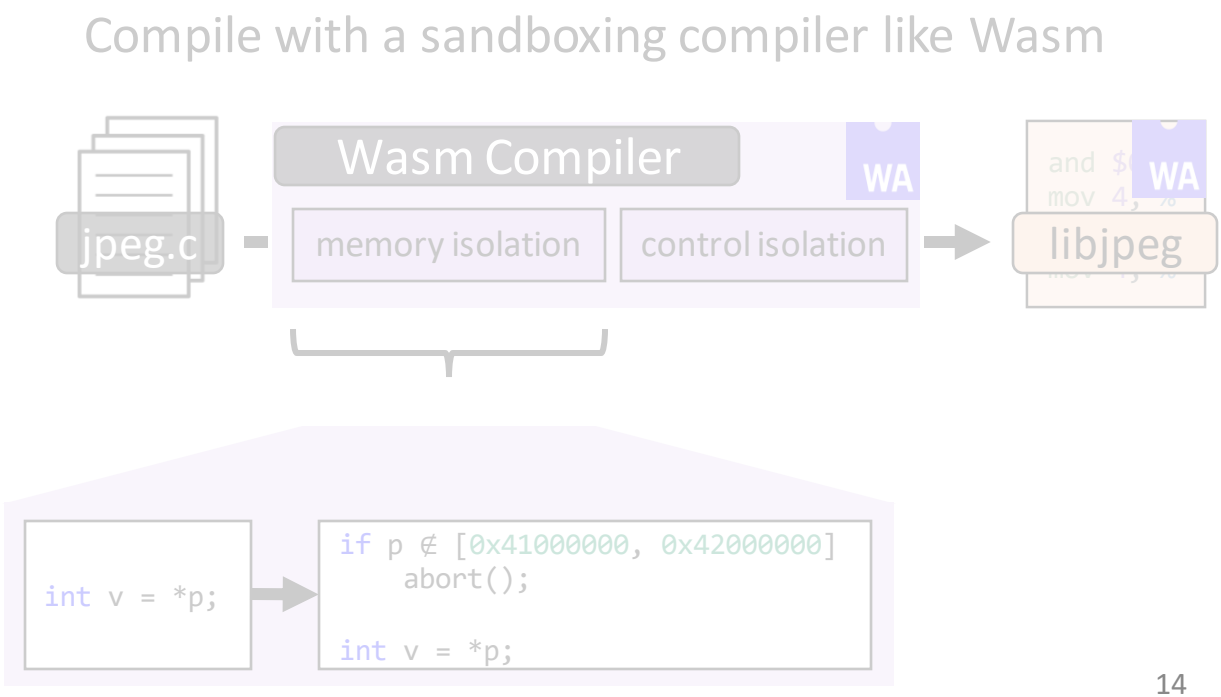
"Real-world operating systems put a ceiling on the effectiveness and applicability of [sandboxing with processes]."

- Chris Palmer
Google Chrome Security

Fine grain isolation to the rescue

Idea (Wahbe et. al, ‘93): Sandbox libjpeg by inserting runtime checks

- Goal: remove libjpeg from the Firefox trusted computing base
- Sandboxing tools: Native Client, WebAssembly



Today

1. Why fine grain isolation?
2. Why is SFI (alone) not the answer? 
3. The RLBox framework
4. What is the cost of RLBox?

Key challenge: we must modify Firefox to secure the boundary

Firefox today implicitly trusts libraries

Firefox must add security checks at the boundary

These modifications are error-prone:

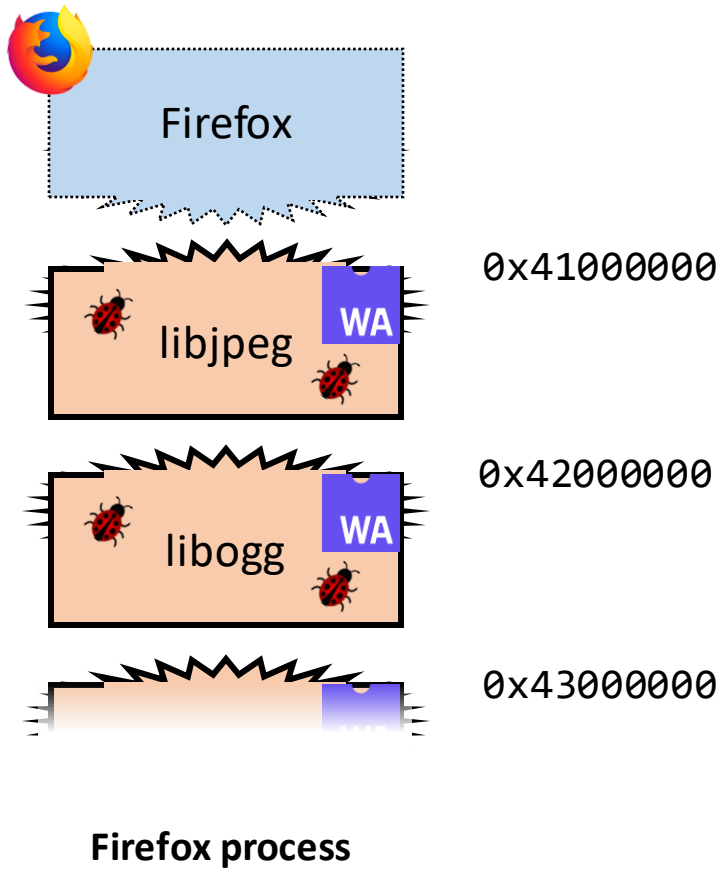
The boundary is blurry

Firefox must account for shared data and control flow

Sandboxing changes the library ABI

Firefox must handle new call conventions and data sizes

Firefox must manually marshal data



Let’s sandbox libjpeg in Firefox

Step 1: Compile libjpeg with Wasm

Step 2: Modify Firefox code to interface with the sandboxed library

Simplified Firefox code that renders JPEGs

```
void create_jpeg_parser() {

  jpeg_decompress_struct* jpeg_img = /* ... */;
  jpeg_error_mgr*         jpeg_err = /* ... */;

  jpeg_create_decompress(jpeg_img);
  jpeg_img->err = jpeg_err;


  jpeg_img->src->fill_input_buffer = firefox_bytes_from_network;

  jpeg_read_header(jpeg_img /* ... */);
  uint32_t* outputBuffer =  /* ... */;
  /* ... */

  while (/* check for output lines */) {
    uint32_t size = jpeg_img->output_width * jpeg_img->output_components;

    memcpy(outputBuffer, /* ... */, size);
  }
}
```

```
void create_jpeg_parser() {

  jpeg_decompress_struct* jpeg_img = /* ... */;
  jpeg_error_mgr*         jpeg_err = /* ... */;

  jpeg_create_decompress(jpeg_img);
  jpeg_img->err = jpeg_err;


  jpeg_img->src->fill_input_buffer = firefox_bytes_from_network;

  jpeg_read_header(jpeg_img /* ... */);
  uint32_t* outputBuffer =  /* ... */;
  /* ... */

  while (/* check for output lines */) {
    uint32_t size = jpeg_img->output_width * jpeg_img->output_components;

    memcpy(outputBuffer, /* ... */, size);
  }
}
```

JPEG image data structures

libjpeg code initializes these structures

Callback from libjpeg to get input

libjpeg code then parses the image

Firefox code transfers output

What changes do we need to make?

Sandboxing ABI changes

1. SFI tools change calling conventions
2. SFI tools introduce ABI differences
3. SFI tools require data marshalling

New trust model

4. Sandboxed lib can call callbacks
5. We need data sanitization

```
void create_jpeg_parser() {

  jpeg_decompress_struct* jpeg_img = /* ... */;
  jpeg_error_mgr*         jpeg_err = /* ... */;

  jpeg_create_decompress(jpeg_img);
  jpeg_img->err = jpeg_err;


  jpeg_img->src->fill_input_buffer = firefox_bytes_from_network;

  jpeg_read_header(jpeg_img /* ... */);
  uint32_t* outputBuffer =  /* ... */;
  /* ... */

  while (/* check for output lines */) {
    uint32_t size = jpeg_img->output_width * jpeg_img->output_components;

    memcpy(outputBuffer, /* ... */, size);
  }
}
```

1. SFI tools change calling conventions
2. SFI tools introduce ABI differences
3. SFI tools require data marshalling
4. Sandboxed lib can call callbacks
5. We need data sanitization

Done! (I think)

```
void create_jpeg_parser() {

  jpeg_decompress_struct* jpeg_img = /* ... */;
  jpeg_error_mgr*         jpeg_err = /* ... */;

  call_conv(jpeg_create_decompress, marshal_to(jpeg_img));
  *field_offset(jpeg_img, err) = marshal_to(jpeg_err);


  *field_offset(*field_offset(jpeg_img, src), fill_input_buffer) = firefox_bytes_from_network;

  call_conv(jpeg_read_header, marshal_to(jpeg_img) /* ... */);
  uint32_t* outputBuffer =  /* ... */;
  /* ... */

  while (/* check for output lines */) {
    uint32_t size = marshal_from(jpeg_img->output_width) * marshal_from(jpeg_img->output_components);

    memcpy(outputBuffer, /* ... */, size);
  }
}
```

1. SFI tools change calling conventions
2. SFI tools introduce ABI differences
3. SFI tools require data marshalling
4. Sandboxed lib can call callbacks
5. We need data sanitization

Done! (I think)

```
void create_jpeg_parser() {

  jpeg_decompress_struct* jpeg_img = /* ... */;
  jpeg_error_mgr*         jpeg_err = /* ... */;

  call_conv(jpeg_create_decompress, marshal_to(jpeg_img));
  *field_offset(jpeg_img, err) = marshal_to(jpeg_err);


  *field_offset(*field_offset(jpeg_img, src), fill_input_buffer) = firefox_bytes_from_network;

  call_conv(jpeg_read_header, marshal_to(jpeg_img) /* ... */);
  uint32_t* outputBuffer =  /* ... */;
  /* ... */

  while (/* check for output lines */) {
    uint32_t size = marshal_from(jpeg_img->output_width) * marshal_from(jpeg_img->output_components);

    memcpy(outputBuffer, /* ... */, size);
  }
}
```

1. SFI tools change calling conventions
2. SFI tools introduce ABI differences
3. SFI tools require data marshalling
4. Sandboxed lib can call callbacks
5. We need data sanitization

Done! (I think)

Repeat analysis in `firefox_bytes_from_network`

1. SFI tools change calling conventions
2. SFI tools introduce ABI differences
3. SFI tools require data marshalling
4. Sandboxed lib can call callbacks
5. We need data sanitization

Done! (I think)

```
void create_jpeg_parser() {

  jpeg_decompress_struct* jpeg_img = /* ... */;
  jpeg_error_mgr*         jpeg_err = /* ... */;

  call_conv(jpeg_create_decompress, marshal_to(jpeg_img));
  *field_offset(jpeg_img, err) = marshal_to(jpeg_err);


  *field_offset(*field_offset(jpeg_img, src), fill_input_buffer) = firefox_bytes_from_network;

  call_conv(jpeg_read_header, marshal_to(jpeg_img) /* ... */);
  uint32_t* outputBuffer =  /* ... */;
  /* ... */

  while (/* check for output lines */) {
    uint32_t size = marshal_from(jpeg_img->output_width) * marshal_from(jpeg_img->output_components);

    memcpy(outputBuffer, /* ... */, size);
  }
}
```

Untrusted JPEG data structure

Using untrusted “size” in memcpy

```
void create_jpeg_parser() {

  jpeg_decompress_struct* jpeg_img = /* ... */;
  jpeg_error_mgr*         jpeg_err = /* ... */;

  call_conv(jpeg_create_decompress, marshal_to(jpeg_img));
  *field_offset(jpeg_img, err) = marshal_to(jpeg_err);


  *field_offset(*field_offset(jpeg_img, src), fill_input_buffer) = firefox_bytes_from_network;

  call_conv(jpeg_read_header, marshal_to(jpeg_img) /* ... */);
  uint32_t* outputBuffer =  /* ... */;
  /* ... */

  while (/* check for output lines */) {
    uint32_t size = marshal_from(jpeg_img->output_width) * marshal_from(jpeg_img->output_components);
    assert(size <= outputBufferSize);
    memcpy(outputBuffer, /* ... */, size);
  }
}
```

1. SFI tools change calling conventions
2. SFI tools introduce ABI differences
3. SFI tools require data marshalling
4. Sandboxed lib can call callbacks
5. We need data sanitization

Done! (I think)

```
void create_jpeg_parser() {

  jpeg_decompress_struct* jpeg_img = /* ... */;
  jpeg_error_mgr*         jpeg_err = /* ... */;

  call_conv(jpeg_create_decompress, marshal_to(jpeg_img));
  *field_offset(jpeg_img, err) = marshal_to(jpeg_err);


  *field_offset(*field_offset(jpeg_img, src), fill_input_buffer) = firefox_bytes_from_network;

  call_conv(jpeg_read_header, marshal_to(jpeg_img) /* ... */);
  uint32_t* outputBuffer =  /* ... */;
  /* ... */

  while (/* check for output lines */) {
    uint32_t size = marshal_from(jpeg_img->output_width) * marshal_from(jpeg_img->output_components);
    assert(size <= outputBufferSize);
    memcpy(outputBuffer, /* ... */, size);
  }
}
```

1. SFI tools change calling conventions
2. SFI tools introduce ABI differences
3. SFI tools require data marshalling
4. Sandboxed lib can call callbacks
5. We need data sanitization

Done! (I think)

Write to location specified by `libjpeg`

1. SFI tools change calling conventions
2. SFI tools introduce ABI differences
3. SFI tools require data marshalling
4. Sandboxed lib can call callbacks
5. We need data sanitization

Done! (I think)

```
void create_jpeg_parser() {

  jpeg_decompress_struct* jpeg_img = /* ... */;
  jpeg_error_mgr*         jpeg_err = /* ... */;

  call_conv(jpeg_create_decompress, marshal_to(jpeg_img));
  *field_offset(jpeg_img, err) = marshal_to(jpeg_err);


  *field_offset(*in_sandbox(field_offset(jpeg_img, src)), fill_input_buffer) = firefox_bytes_from_network;

  call_conv(jpeg_read_header, marshal_to(jpeg_img) /* ... */);
  uint32_t* outputBuffer =  /* ... */;
  /* ... */

  while (/* check for output lines */) {
    uint32_t size = marshal_from(jpeg_img->output_width) * marshal_from(jpeg_img->output_components);
    assert(size <= outputBufferSize);
    memcpy(outputBuffer, /* ... */, size);
  }
}
```

```
void create_jpeg_parser() {

  jpeg_decompress_struct* jpeg_img = /* ... */;
  jpeg_error_mgr*         jpeg_err = /* ... */;

  call_conv(jpeg_create_decompress, marshal_to(jpeg_img));
  *field_offset(jpeg_img, err) = marshal_to(jpeg_err);


  *field_offset(*in_sandbox(field_offset(jpeg_img, src)), fill_input_buffer) = firefox_bytes_from_network;

  call_conv(jpeg_read_header, marshal_to(jpeg_img) /* ... */);
  uint32_t* outputBuffer =  /* ... */;
  /* ... */

  while (/* check for output lines */) {
    uint32_t size = marshal_from(jpeg_img->output_width) * marshal_from(jpeg_img->output_components);
    assert(size <= outputBufferSize);
    memcpy(outputBuffer, /* ... */, size);
  }
}
```

This falls flat in practice

1. Real systems are huge
   ⇒ requires a lot of code changes

2. Mixes low-level sandbox code in image rendering
   ⇒ code maintenance is difficult

3. Testing, debugging, and cross-platform support
   ⇒ requires even more code changes

This falls flat in practice

Ben Laurie @BenLaurie · Mar 3, 2021

We (mostly twitter.com/dmd_lurklurk) did some experimentation on sandboxing libraries - it turns out that many of the worst offenders are pretty much impossible to sandbox in this way because of their APIs, which rely far too much on shared memory.

Today

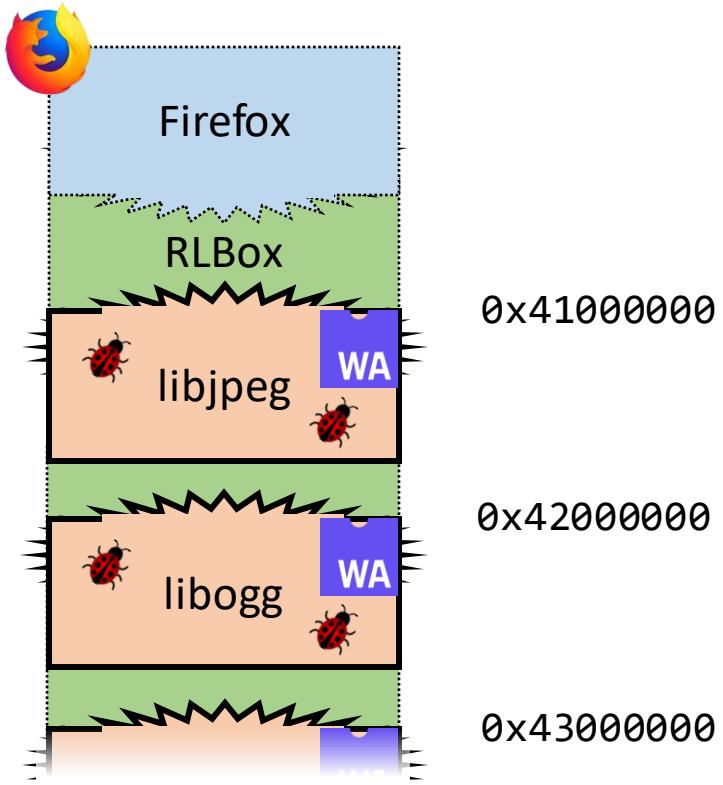
1. Why fine grain isolation?
2. Why is SFI (alone) not the answer?
3. The RLBox framework 
4. What is the cost of RLBox?

RLBox: framework to retrofit sandboxing USENIX Security '20

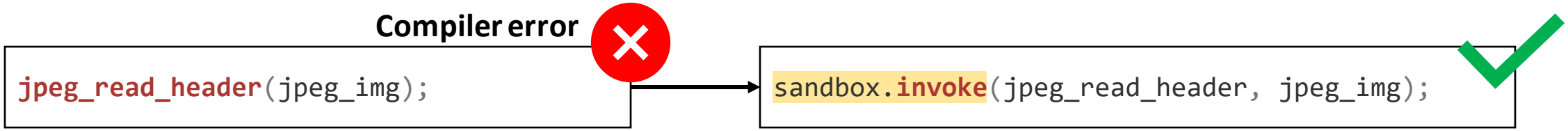
Idea: Use types to make sandboxing a compositional abstraction

1. **Types hide low-level sandbox details**
   Automates ABI conversions, data marshalling
2. **Types track untrusted data and control flow**
   Missing security checks ⇒ type errors
3. **Types allow retrofitting piecemeal**
   Test, review, and deploy sandboxing incrementally

Implemented as a pure C++ library



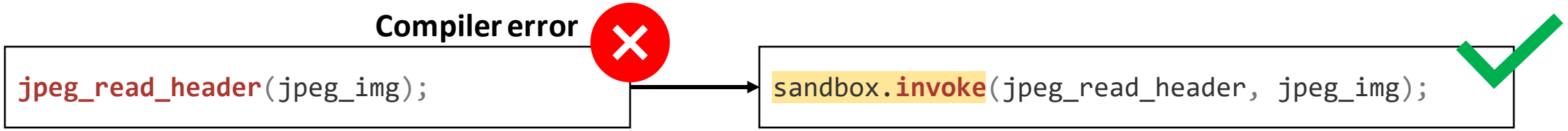
1. RLBox forces control flow to be explicit



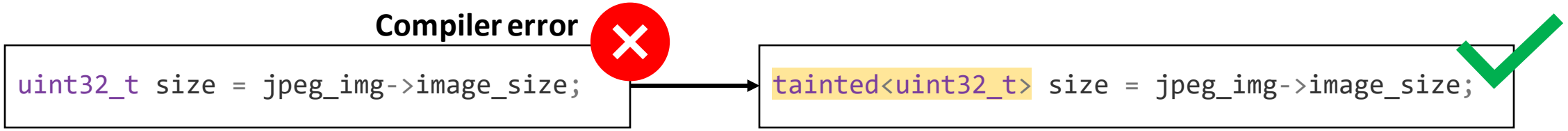
2. RLBox forces data from the sandbox to be marked **tainted**

3. Tainted data must be checked before use

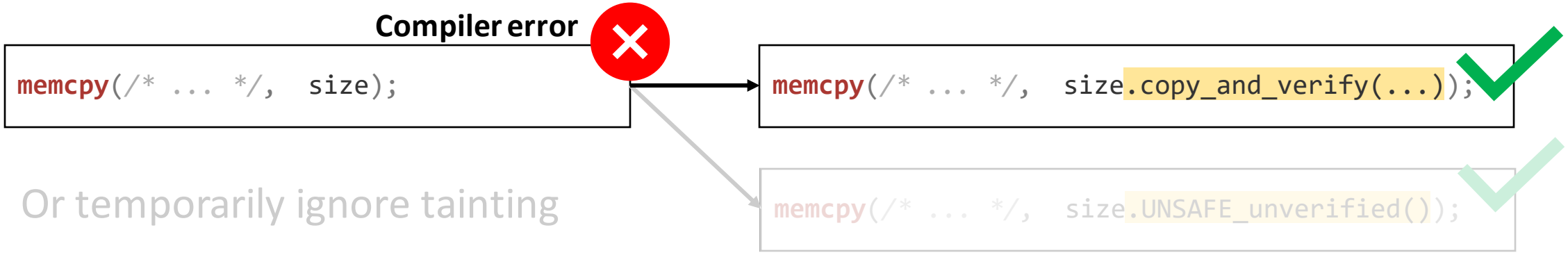
1. RLBox forces control flow to be explicit



2. RLBox forces data from the sandbox to be marked **tainted**



3. Tainted data must be checked before use



```
void create_jpeg_parser() {

  jpeg_decompress_struct* jpeg_img = /* ... */;
  jpeg_error_mgr*         jpeg_err = /* ... */;

  jpeg_create_decompress(jpeg_img);
  jpeg_img->err = jpeg_err;

  jpeg_img->src->fill_input_buffer = firefox_bytes_from_network;

  jpeg_read_header(jpeg_img /* ... */);
  uint32_t* outputBuffer =  /* ... */;

  while (/* check for output lines */) {
    uint32_t size = jpeg_img->output_width * jpeg_img->output_components;



    memcpy(outputBuffer, /* ... */, size);
  }
}
```

```
void create_jpeg_parser() {
  auto sandbox = rlbox::create_sandbox<noop>();
  jpeg_decompress_struct* jpeg_img = /* ... */;
  jpeg_error_mgr*         jpeg_err = /* ... */;

  jpeg_create_decompress(jpeg_img);
  jpeg_img->err = jpeg_err;

  jpeg_img->src->fill_input_buffer = firefox_bytes_from_network;

  jpeg_read_header(jpeg_img /* ... */);
  uint32_t* outputBuffer =  /* ... */;

  while (/* check for output lines */) {
    uint32_t size = jpeg_img->output_width * jpeg_img->output_components;



    memcpy(outputBuffer, /* ... */, size);
  }
}
```

Create a “noop” sandbox

```
void create_jpeg_parser() {
  auto sandbox = rlbox::create_sandbox<noop>();
  jpeg_decompress_struct* jpeg_img = /* ... */;
  jpeg_error_mgr*         jpeg_err = /* ... */;

  jpeg_create_decompress(jpeg_img);
  jpeg_img->err = jpeg_err;

  jpeg_img->src->fill_input_buffer = firefox_bytes_from_network;

  jpeg_read_header(jpeg_img /* ... */);
  uint32_t* outputBuffer =  /* ... */;

  while (/* check for output lines */) {
    uint32_t size = jpeg_img->output_width * jpeg_img->output_components;



    memcpy(outputBuffer, /* ... */, size);
  }
}
```

Make shared data structures `tainted`

Use RLBox API for function calls

```
void create_jpeg_parser() {
  auto sandbox = rlbox::create_sandbox<noop>();
  tainted<jpeg_decompress_struct*> t_jpeg_img = /* ... */;
  tainted<jpeg_error_mgr*>         t_jpeg_err = /* ... */;

  sandbox.invoke(jpeg_create_decompress, t_jpeg_img);
  t_jpeg_img->err = t_jpeg_err;

  jpeg_img->src->fill_input_buffer = firefox_bytes_from_network;

  jpeg_read_header(jpeg_img /* ... */);
  uint32_t* outputBuffer =  /* ... */;

  while (/* check for output lines */) {
    uint32_t size = jpeg_img->output_width * jpeg_img->output_components;



    memcpy(outputBuffer, /* ... */, size);
  }
}
```

Make shared data structures `tainted`

Use RLBox API for function calls

Compiles?

```
void create_jpeg_parser() {
  auto sandbox = rlbox::create_sandbox<noop>();
  tainted<jpeg_decompress_struct*> t_jpeg_img = /* ... */;
  tainted<jpeg_error_mgr*>         t_jpeg_err = /* ... */;

  sandbox.invoke(jpeg_create_decompress, t_jpeg_img);
  t_jpeg_img->err = t_jpeg_err;

  jpeg_img->src->fill_input_buffer = firefox_bytes_from_network;

  jpeg_read_header(jpeg_img /* ... */);
  uint32_t* outputBuffer =  /* ... */;

  while (/* check for output lines */) {
    uint32_t size = jpeg_img->output_width * jpeg_img->output_components;



    memcpy(outputBuffer, /* ... */, size);
  }
}
```

Undefined variable: `jpeg_img`

Compiles?

```
void create_jpeg_parser() {
  auto sandbox = rlbox::create_sandbox<noop>();
  tainted<jpeg_decompress_struct*> t_jpeg_img = /* ... */;
  tainted<jpeg_error_mgr*>         t_jpeg_err = /* ... */;

  sandbox.invoke(jpeg_create_decompress, t_jpeg_img);
  t_jpeg_img->err = t_jpeg_err;
  jpeg_decompress_struct* jpeg_img = t_jpeg_img.UNSAFE_unverified();
  jpeg_img->src->fill_input_buffer = firefox_bytes_from_network;

  jpeg_read_header(jpeg_img /* ... */);
  uint32_t* outputBuffer =  /* ... */;

  while (/* check for output lines */) {
    uint32_t size = jpeg_img->output_width * jpeg_img->output_components;



    memcpy(outputBuffer, /* ... */, size);
  }
}
```

Unsafe alias to `p_jpeg_img`

Compiles?

```
void create_jpeg_parser() {
  auto sandbox = rlbox::create_sandbox<noop>();
  tainted<jpeg_decompress_struct*> t_jpeg_img = /* ... */;
  tainted<jpeg_error_mgr*>         t_jpeg_err = /* ... */;

  sandbox.invoke(jpeg_create_decompress, t_jpeg_img);
  t_jpeg_img->err = t_jpeg_err;
  jpeg_decompress_struct* jpeg_img = t_jpeg_img.UNSAFE_unverified();
  jpeg_img->src->fill_input_buffer = firefox_bytes_from_network;
  jpeg_decompress_struct* jpeg_img = t_jpeg_img.UNSAFE_unverified();
  jpeg_read_header(jpeg_img /* ... */);
  uint32_t* outputBuffer =  /* ... */;

  while (/* check for output lines */) {
    uint32_t size = jpeg_img->output_width * jpeg_img->output_components;



    memcpy(outputBuffer, /* ... */, size);
  }
}
```

Compiles?

```
void create_jpeg_parser() {
  auto sandbox = rlbox::create_sandbox<noop>();
  tainted<jpeg_decompress_struct*> t_jpeg_img = /* ... */;
  tainted<jpeg_error_mgr*>         t_jpeg_err = /* ... */;

  sandbox.invoke(jpeg_create_decompress, t_jpeg_img);
  t_jpeg_img->err = t_jpeg_err;

  t_jpeg_img->src->fill_input_buffer = sandbox.register_callback(firefox_bytes_from_network);

  sandbox.invoke(jpeg_read_header, t_jpeg_img /* ... */);
  uint32_t* outputBuffer =  /* ... */;

  while (/* check for output lines */) {
    tainted<uint32_t> size = t_jpeg_img->output_width * t_jpeg_img->output_components;


    memcpy(outputBuffer, /* ... */, size);
  }
}
```

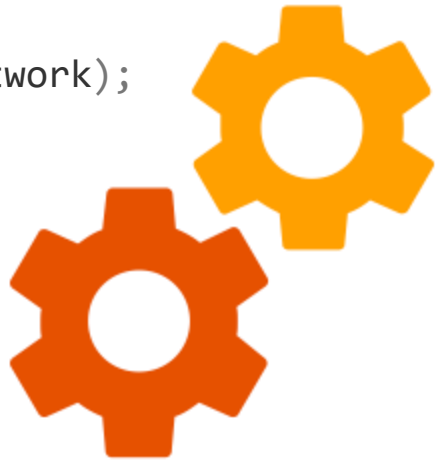
1. RLBox adjusts for ABI differences

2. RLBox checks this dereference

3. The type of `size` is `tainted`

```
void create_jpeg_parser() {
  auto sandbox = rlbox::create_sandbox<noop>();
  tainted<jpeg_decompress_struct*> t_jpeg_img = /* ... */;
  tainted<jpeg_error_mgr*>         t_jpeg_err = /* ... */;

  sandbox.invoke(jpeg_create_decompress, t_jpeg_img);
  t_jpeg_img->err = t_jpeg_err;

  t_jpeg_img->src->fill_input_buffer = sandbox.register_callback(firefox_bytes_from_network);

  sandbox.invoke(jpeg_read_header, t_jpeg_img /* ... */);
  uint32_t* outputBuffer =  /* ... */;

  while (/* check for output lines */) {
    tainted<uint32_t> size = t_jpeg_img->output_width * t_jpeg_img->output_components;



    memcpy(outputBuffer, /* ... */, size);
  }
}
```

Compiles?



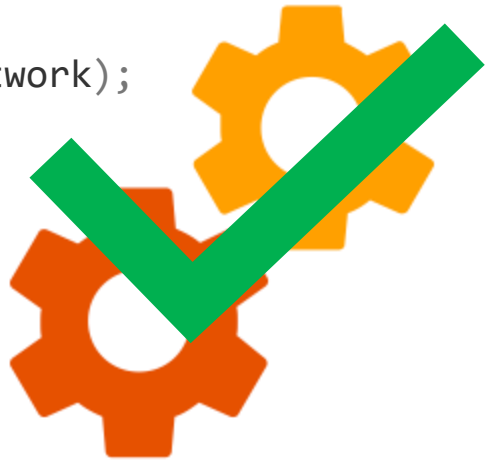
Expected: `uint32_t`. Got: `tainted<uint32_t>`

```
void create_jpeg_parser() {
  auto sandbox = rlbox::create_sandbox<noop>();
  tainted<jpeg_decompress_struct*> t_jpeg_img = /* ... */;
  tainted<jpeg_error_mgr*>         t_jpeg_err = /* ... */;

  sandbox.invoke(jpeg_create_decompress, t_jpeg_img);
  t_jpeg_img->err = t_jpeg_err;

  t_jpeg_img->src->fill_input_buffer = sandbox.register_callback(firefox_bytes_from_network);

  sandbox.invoke(jpeg_read_header, t_jpeg_img /* ... */);
  uint32_t* outputBuffer =  /* ... */;

  while (/* check for output lines */) {
    tainted<uint32_t> size = t_jpeg_img->output_width * t_jpeg_img->output_components;



    memcpy(outputBuffer, /* ... */, size);
  }
}
```

Need to remove tainting

```
void create_jpeg_parser() {
  auto sandbox = rlbox::create_sandbox<noop>();
  tainted<jpeg_decompress_struct*> t_jpeg_img = /* ... */;
  tainted<jpeg_error_mgr*>         t_jpeg_err = /* ... */;

  sandbox.invoke(jpeg_create_decompress, t_jpeg_img);
  t_jpeg_img->err = t_jpeg_err;

  t_jpeg_img->src->fill_input_buffer = sandbox.register_callback(firefox_bytes_from_network);

  sandbox.invoke(jpeg_read_header, t_jpeg_img /* ... */);
  uint32_t* outputBuffer =  /* ... */;

  while (/* check for output lines */) {
    tainted<uint32_t> t_size = t_jpeg_img->output_width * t_jpeg_img->output_components;
    uint32_t size = t_size.copy_and_verify([](uint32_t val) {
       assert(val <= outputBufferSize);
    });
    memcpy(outputBuffer, /* ... */, size);
  }
}
```

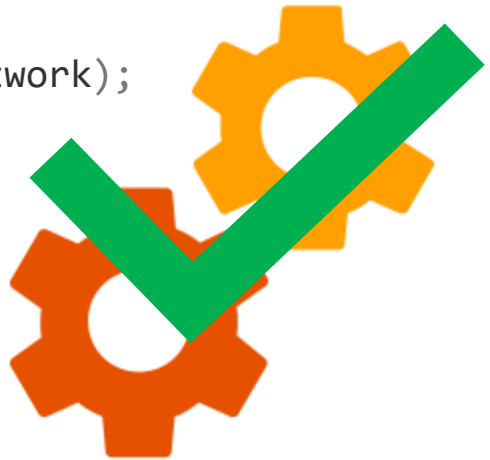
Compiles?



```
void create_jpeg_parser() {
  auto sandbox = rlbox::create_sandbox<wasm>();
  tainted<jpeg_decompress_struct*> t_jpeg_img = /* ... */;
  tainted<jpeg_error_mgr*>         t_jpeg_err = /* ... */;

  sandbox.invoke(jpeg_create_decompress, t_jpeg_img);
  t_jpeg_img->err = t_jpeg_err;

  t_jpeg_img->src->fill_input_buffer = sandbox.register_callback(firefox_bytes_from_network);

  sandbox.invoke(jpeg_read_header, t_jpeg_img /* ... */);
  uint32_t* outputBuffer =  /* ... */;

  while (/* check for output lines */) {
    tainted<uint32_t> t_size = t_jpeg_img->output_width * t_jpeg_img->output_components;
    uint32_t size = t_size.copy_and_verify([](uint32_t val) {
       assert(val <= outputBufferSize);
    });
    memcpy(outputBuffer, /* ... */, size);
  }
}
```

Switch to a WebAssembly sandbox

Compiles?



Done!

Today

1. Why fine grain isolation?
2. Why is SFI (alone) not the answer?
3. The RLBox framework
4. What is the cost of RLBox? 

What is the cost of RLBox?

We evaluate RLBox on several dimensions. In this talk:

- Developer effort & automation
- Performance overhead

We sandboxed Firefox's

- Image rendering
- Audio, video playback
- Font rendering
- Decompression
- XML parsing
- Spell checking

Firefox cannot be compromised by a malicious image

RLBox reduces developer effort for sandboxing

RLBox in Firefox

- **Automatic security checks:** dozens to hundreds per library
- **Remaining data validation:** On average two to four lines of code

Beyond Firefox

Usable by experts and beginners

- **Experts:** sandboxing libraries in a few days
- **Beginners:** 4 masters, 4 undergraduate, 1 high school student

Microbenchmark: Image rendering (Native Client)

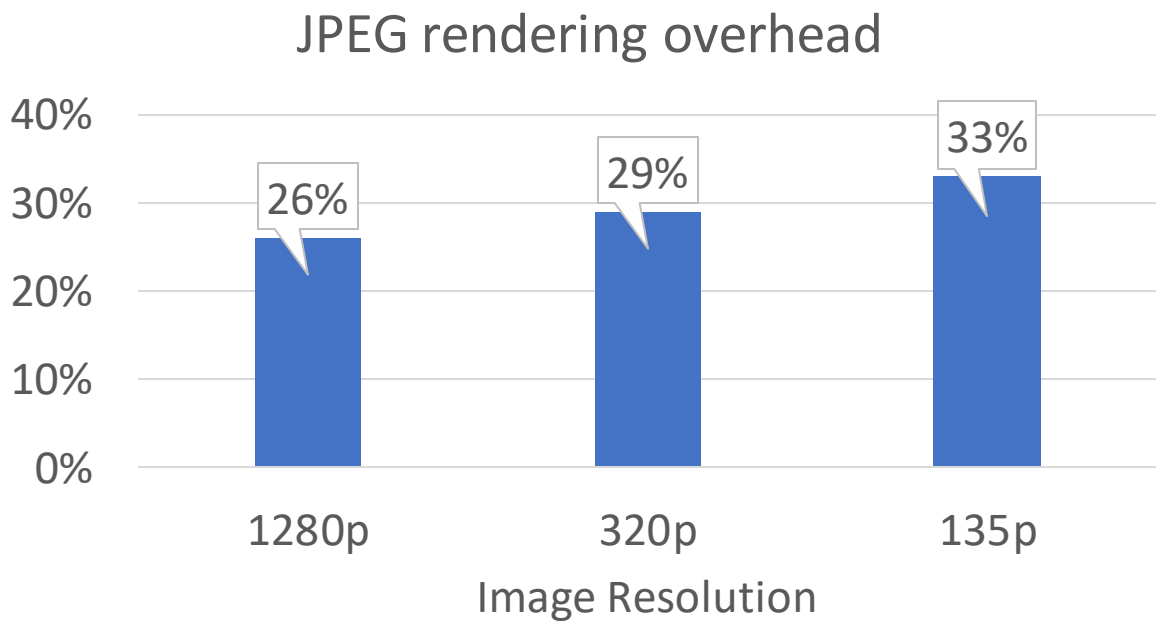
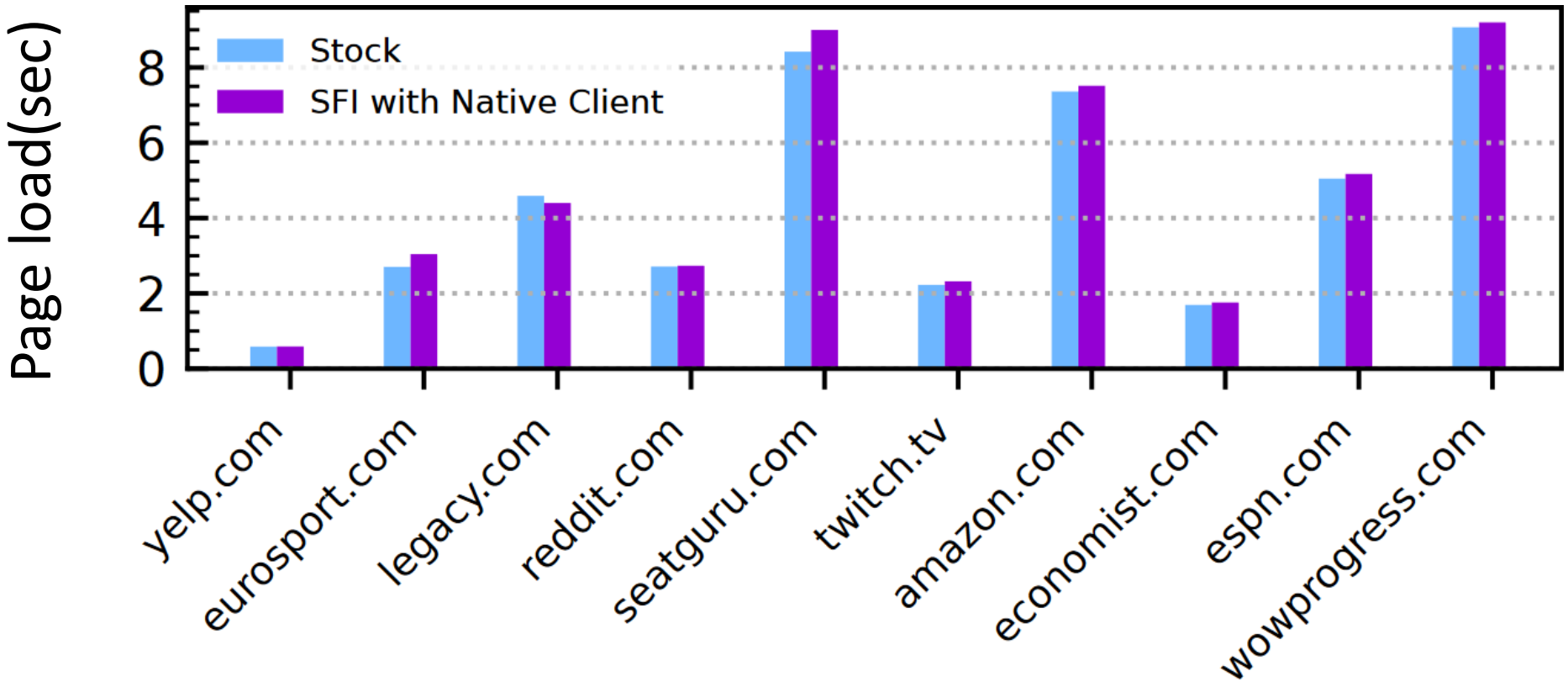


Image rendering/compression test suite https://imagecompression.info/test_images/

Macrobenchmark: Page load (Native Client)



Google deprecates Native Client

Chromium Blog

News and developments from the open source browser project

Goodbye PNaCl, Hello WebAssembly!

Tuesday, May 30, 2017

Historically, running native code on the web required a browser plugin. In 2013, we introduced the PNaCl sandbox to provide a means of building safe, portable, high-performance apps without plugins. Although this worked well in Chrome, it did not provide a solution that worked seamlessly across all browsers.

Gobi: WebAssembly as a Practical Path to Library Sandboxing

Shravan Narayan
UC San Diego

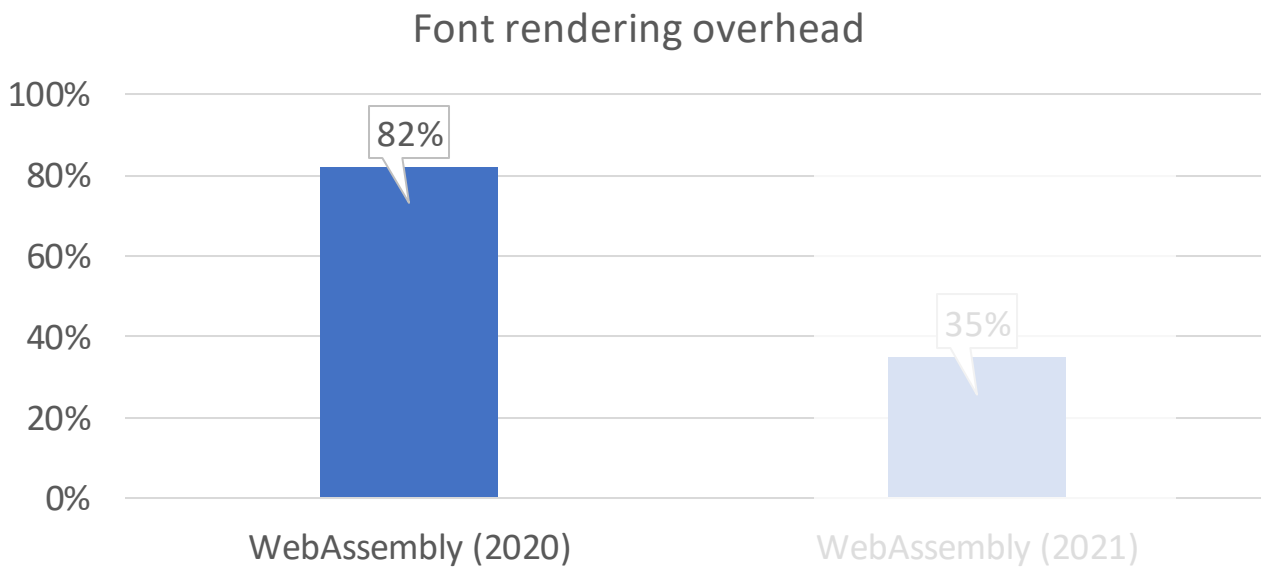
Tal Garfinkel
Stanford University

Sorin Lerner
UC San Diego

Hovav Shacham
UT Austin

Deian Stefan
UC San Diego

Font rendering (WebAssembly)



Optimized WebAssembly context switches

Isolation Without Taxation

Near-Zero-Cost Transitions for WebAssembly and SFI

MATTHEW KOLOSICK, UC San Diego, USA
SHRAVAN NARAYAN, UC San Diego, USA
EVAN JOHNSON, UC San Diego, USA
CONRAD WATT, University of Cambridge, UK
MICHAEL LEMAY, Intel Labs, USA
DEEPAK GARG, Max Planck Institute for Software Systems, Germany
RANJIT JHALA, UC San Diego, USA
DEIAN STEFAN, UC San Diego, USA

Mozilla developers evaluate RLBox

XML Parsing

7%

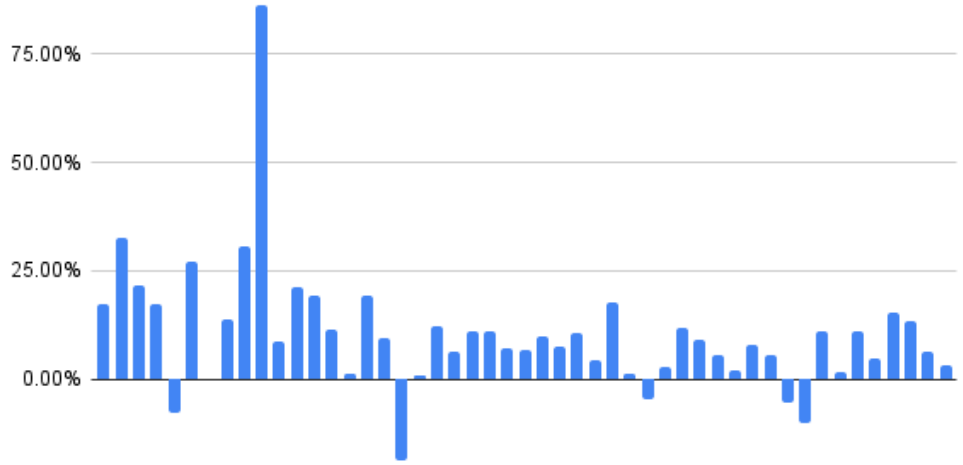
Bobby Holley (:bholley) ▾
Comment 37 • 3 months ago

I did quite a bit of performance measurement on the latest patches (which eliminate all the boundary allocation, copying, and locking, at least on 64-bit). The upshot is that, for the gdocs testcase I measured, RLBox introduces an SVG parsing overhead of about 7% on 64-bit platforms (~80% of our users), and about 20% on 32-bit platforms.

Font decompression

10.5%

Jonathan Kew (:jfkthame) ▾
Comment 17 • 3 months ago • Edited



Deployed in Firefox!

Securing Firefox with WebAssembly

By Nathan Froyd

Posted on February 25, 2020 in Featured Article, Firefox, Rust, Security, and WebAssembly

Protecting the security and privacy of individuals is a central tenet of Mozilla's mission, and so we constantly endeavor to make our users safer online. With a

WebAssembly and Back Again: Fine-Grained Sandboxing in Firefox 95

By Bobby Holley

Posted on December 6, 2021 in Featured Article, Firefox, and JavaScript

In Firefox 95, we're shipping a novel sandboxing technology called RLBox —

Feb 2020

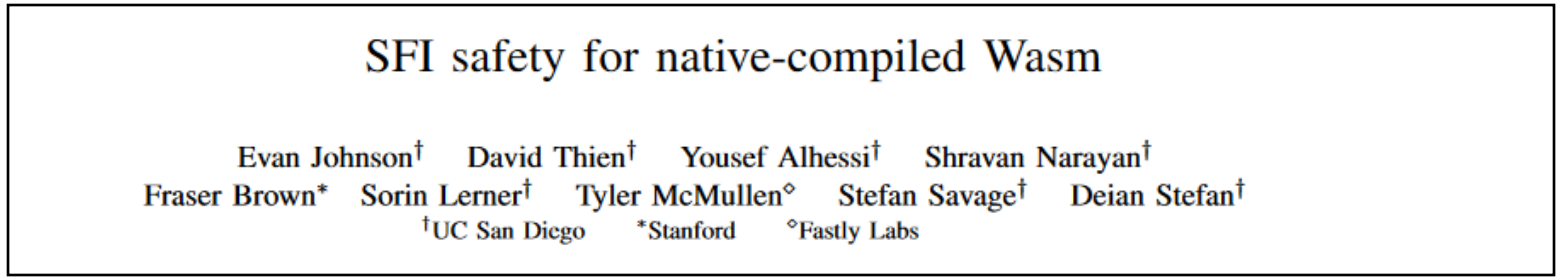
- 1 sandboxed library
- Mac, Linux

Dec 2021

- 5 sandboxed libraries
- Windows, Android

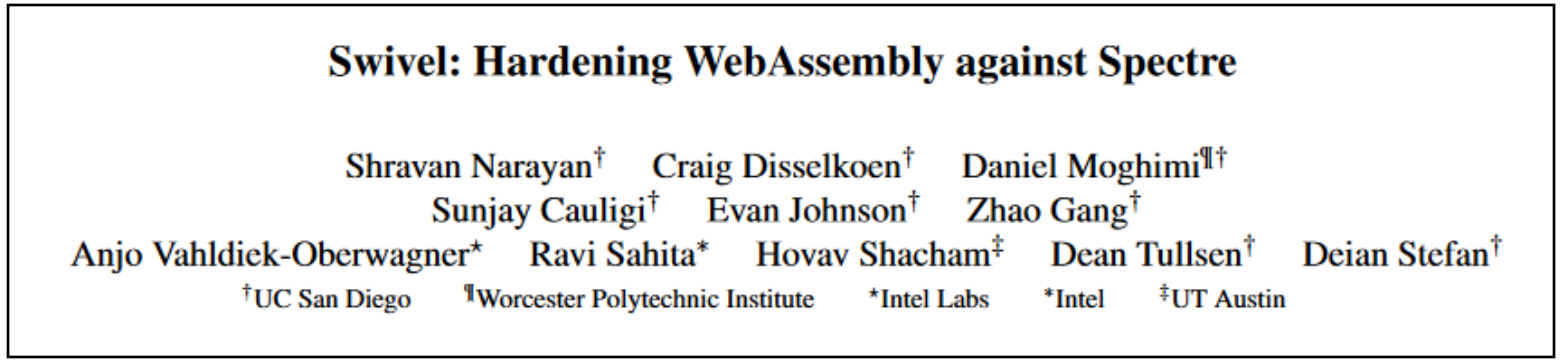
Follow-up work

Defend against sandbox compilers bugs


SFI safety for native-compiled Wasm

Evan Johnson[†] David Thien[†] Yousef Alhessi[†] Shravan Narayan[†]
Fraser Brown[*] Sorin Lerner[†] Tyler McMullen[◇] Stefan Savage[†] Deian Stefan[†]
[†]UC San Diego [*]Stanford [◇]Fastly Labs

Preventing sandbox breakout via Spectre attacks


Swivel: Hardening WebAssembly against Spectre

Shravan Narayan[†] Craig Disselkoen[†] Daniel Moghimi[¶†]
Sunjay Cauligi[†] Evan Johnson[†] Zhao Gang[†]
Anjo Vahldiek-Oberwagner[⋆] Ravi Sahita[*] Hovav Shacham[‡] Dean Tullsen[†] Deian Stefan[†]
[†]UC San Diego [¶]Worcester Polytechnic Institute [*]Intel Labs [⋆]Intel [‡]UT Austin

Our work: fine grain library isolation in Firefox

Idea: Isolate each library in its own memory sandbox

- Use software-based fault isolation (SFI) to do this via runtime checks

Problem: In practice, SFI does not work

- (Near) impossible to retrofit SFI in large systems
- Incomplete: applications can be compromised by trusting library output

Solution: RLBox sandboxing framework

- Uses types to retrofit isolation
- Deployed in the real world


engadget

Firefox 95 enhances the browser's protection against malicious code

Mozilla is deploying a new sandboxing technology called RLBox.