

Don't Get Owned by Your Dependencies

How Firefox uses in-process sandboxing to protect itself from exploitable libraries
(and you can too!)

Presenter: Shravan Narayan

Q & A: **Shravan Narayan**, Tal Garfinkel, Deian Stefan

About us:



Assistant Professor at UT Austin (from Fall 2023)
Current PhD Student at UC San Diego

UC San Diego



Deian Stefan



Hovav Shacham



Bobby Holley



Tal Garfinkel



Tom Ritter



Mike Hommey

In a nutshell

In-process sandboxing: New technique for securing native code

30 years in the making, finally in production!

Wasm: a widely available toolchain for compiler-based isolation

RLBox: practical tool to sandbox native code w/ Wasm

Firefox has shipped RLBox for 2+ years, now you can too!

Outline

1. Motivation
2. Why do we need a sandboxing framework
3. The RLBox sandboxing framework
4. Our experiences deploying RLBox

Outline

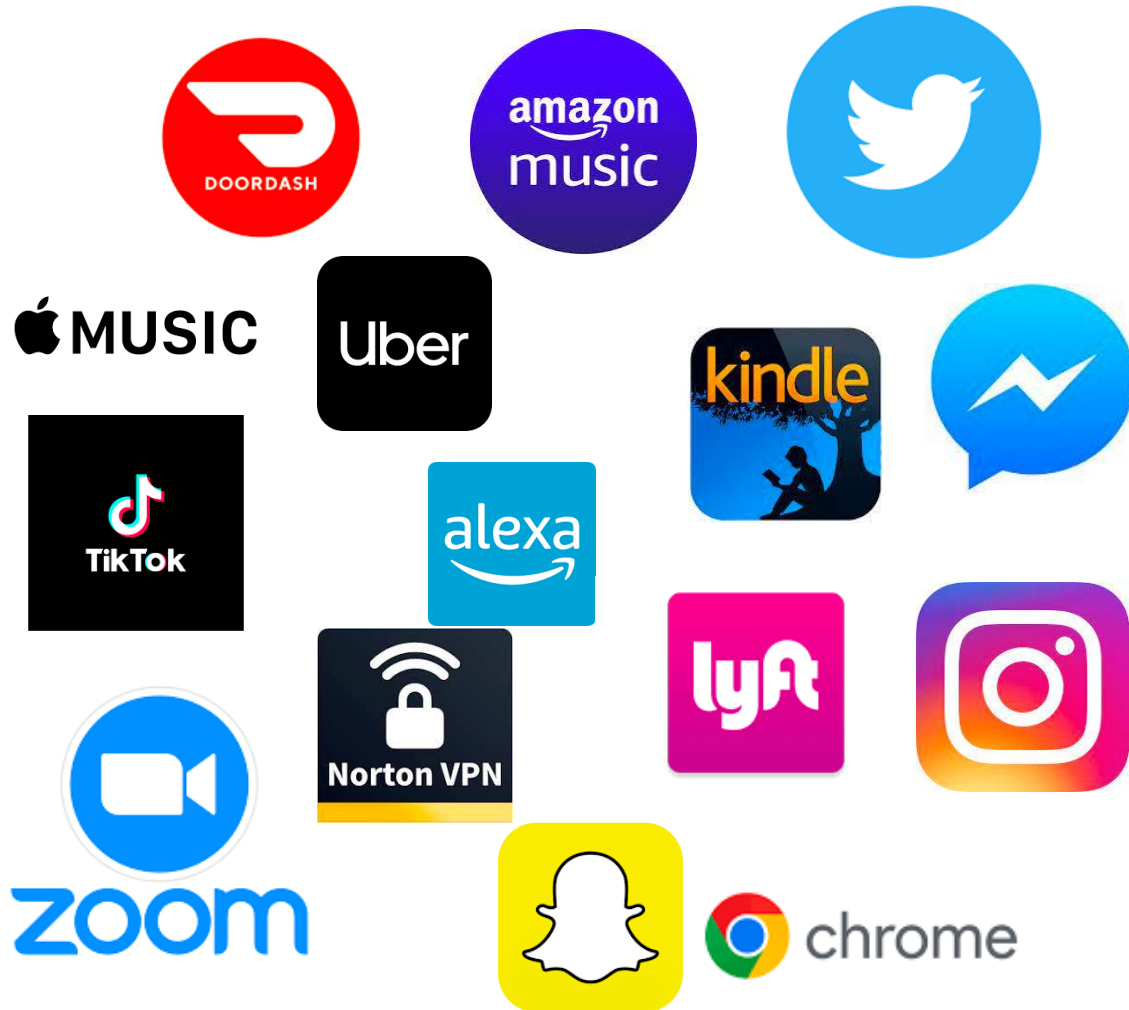
1. Motivation

2. Why do we need a sandboxing framework

3. The RLBox sandboxing framework

4. Our experiences deploying RLBox

Every app depends on native (C/C++) libraries



OpenSSL-1.0.2p, OpenSSL-1.0.1s,
OpenSSL-1.0.2p, OpenSSL-1.1.0
SQLite3-3.27.2,
OpenCV-2.4.1, OpenCV-2.4.11
FFmpeg-2.8.0
GIFLib-5.1.1
XML2-2.7.7
Libpng-1.6.7, Libpng-1.6.34
WebRTC-*
Libgraphite-*
Freetype-*
Libogg-

Native code tends to have memory safety bugs

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

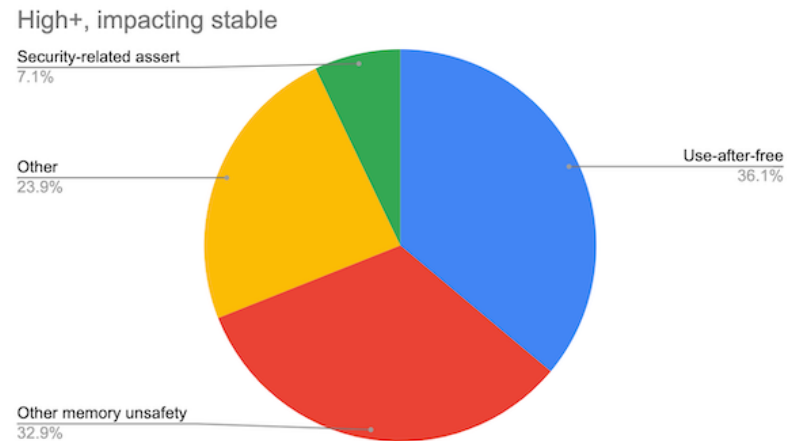


Image from the Chromium project blog

<https://www.chromium.org/Home/chromium-security/memory-safety/>

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

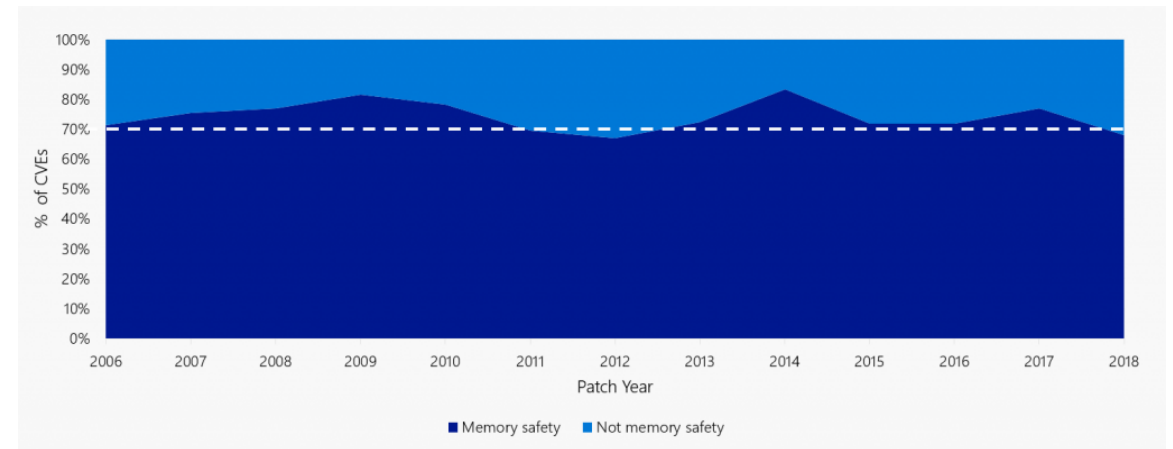



Image from the Microsoft security response center blog

<https://msrc-blog.microsoft.com/2019/07/16/a-proactive-approach-to-more-secure-code/>

“Safe” languages are built on unsafe libraries

 **cc**

A build-time dependency for Cargo build scripts to assist in invoking the native C compiler to compile native C code into a static archive to be linked into Rust code.

[#build-dependencies](#)

[Readme](#) [75 Versions](#) [Dependencies](#) [Dependents](#)

Displaying 1-10 of 1226 reverse dependencies of cc

backtrace DEPENDS ON ^1.0.67

46,022,032

A library to acquire a stack trace (backtrace) at runtime in a Rust program.

openssl-sys DEPENDS ON ^1.0

39,867,553

FFI bindings to OpenSSL

ring DEPENDS ON ^1.0.62

26,264,790

Safe, fast, small crypto using Rust.

libz-sys DEPENDS ON ^1.0.18

20,270,680

Low-level bindings to the system libz library (also known as zlib).

signal-hook DEPENDS ON ^1

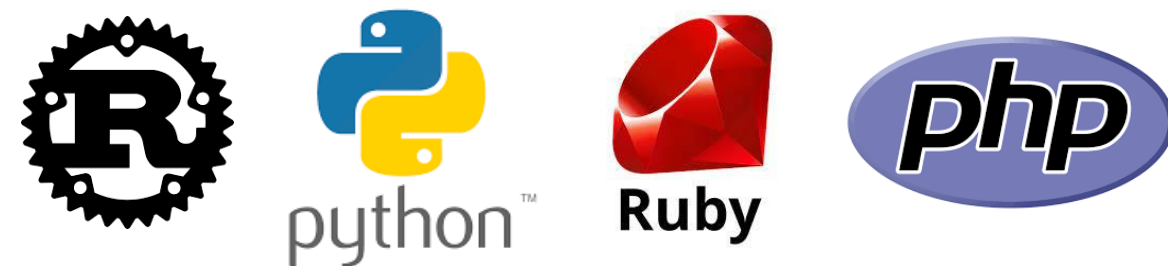
14,916,536

Unix signal handling

backtrace-sys DEPENDS ON ^1.0.37

14,077,612

Bindings to the libbacktrace gcc library



Node.js v18.7.0 documentation

[Table of contents](#) | [Index](#) | [Other versions](#) | [Options](#)

C++ addons

Addons are dynamically-linked shared objects written in C++. The `require()` function can load addons as ordinary Node.js modules. Addons provide an interface between JavaScript and C/C++ libraries.







There are three options for implementing addons: Node-API, nan, or direct use of internal V8, libuv and Node.js libraries. Unless there is a need for direct access to functionality which is not exposed by Node-API, use Node-API. Refer to [C/C++ addons with Node-API](#) for more information on Node-API.

Direct Vulnerabilities

Known vulnerabilities in the numpy package. This does not include vulnerabilities belonging to this package's dependencies.

Automatically find and fix vulnerabilities affecting your projects. Snyk scans for vulnerabilities and provides fixes for free.

[Fix for free](#)

VULNERABILITY	VULNERABLE VERSIONS
  Buffer Overflow	[, 1.21.0rc1)
  Buffer Overflow	[, 1.22.0)
  NULL Pointer Dereference	[0, 1.22.2)

Native library bugs are used in real attacks

From Pearl to Pegasus

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

By Bill Marczak, Ali Abdulemam¹, 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.

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

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

Existing defenses don't work

Standard mitigations

ASLR, Stack canaries, CFI

Routinely bypassed

Completeness vs performance trade-off

Rewrite code in safe language (Rust)?

Significant effort: rewrite, retest

Billions of LOC not going away

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

- Eric Rescorla
CTO, Firefox

Process sandboxing: isolate code in a new process?

Been around for decades, rarely used in practice!

High performance costs

IPC is expensive!

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

High engineering costs

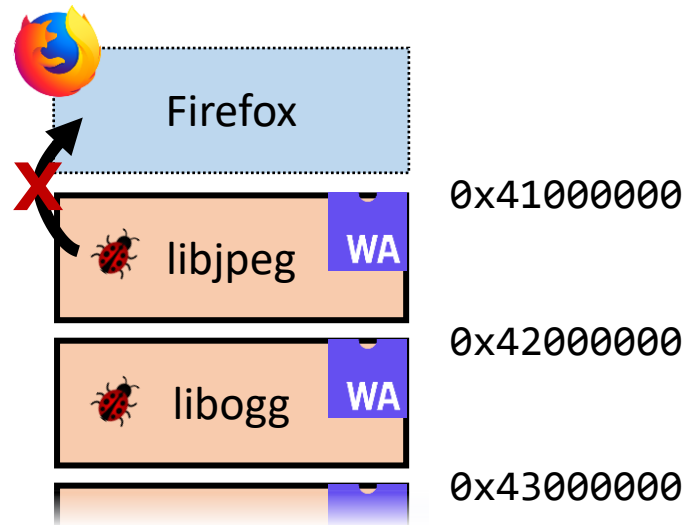
Total re-architecture, can't retrofit

- Chris Palmer
Google Chrome Security
[The Limits of Sandboxing and Next Steps](#)

What does work: **in-process** sandboxing

Isolate libraries in **WebAssembly (Wasm)** sandboxes *

For this talk: Wasm is a compiler that isolates code via runtime checks



Firefox process with sandboxed libraries

Advantages

No IPC, low perf overheads

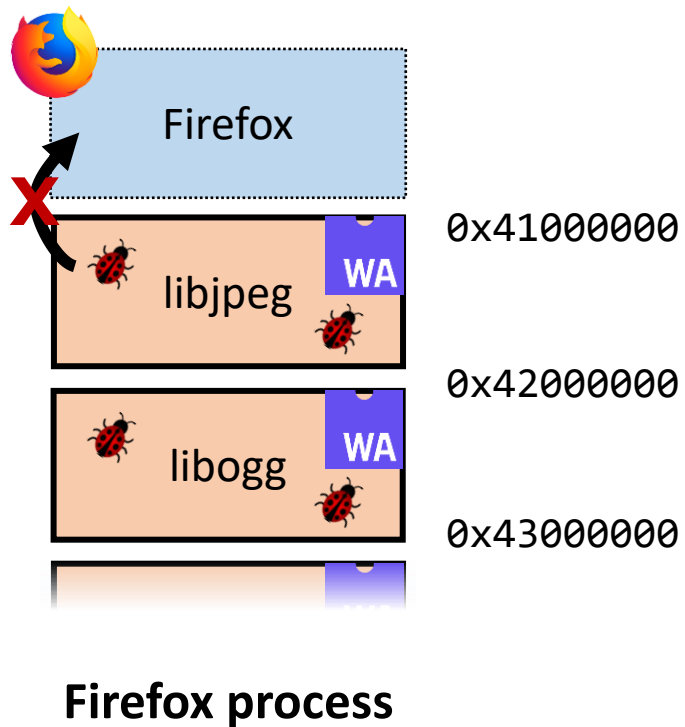
Disadvantages

Engineering effort

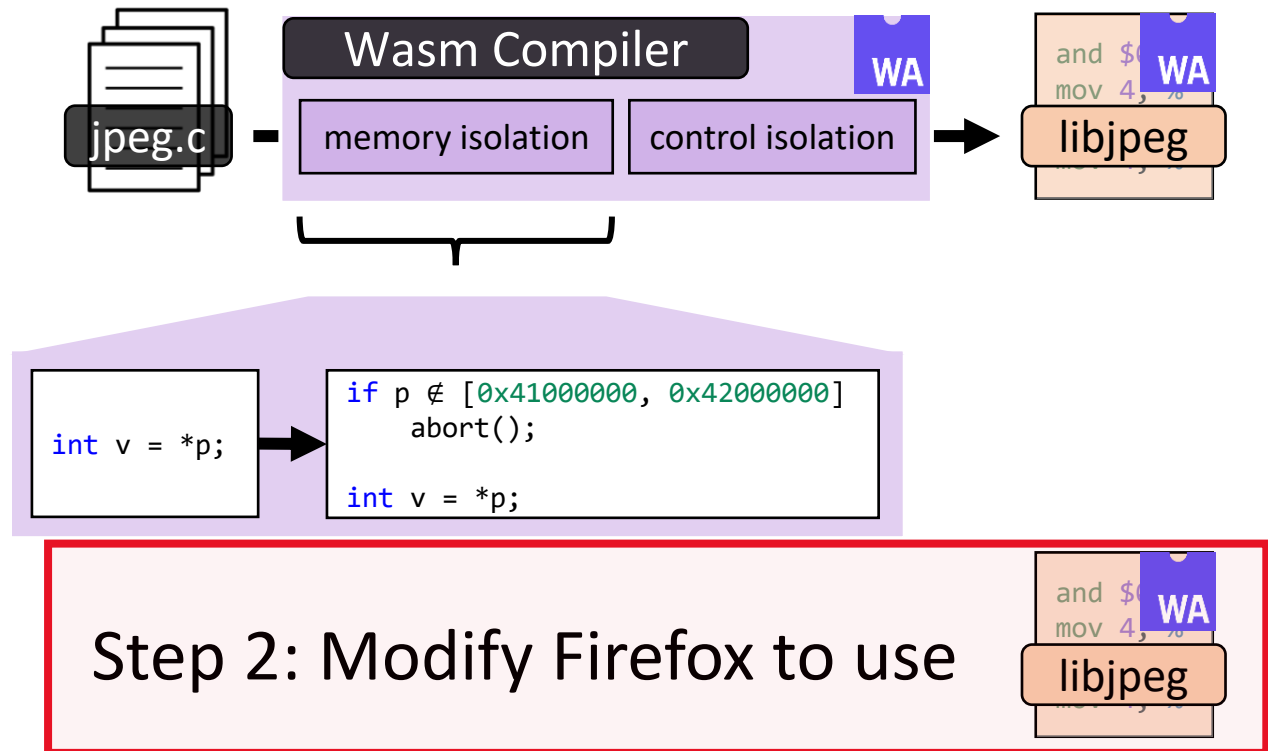
Outline

1. Motivation
- 2. Why do we need a sandboxing framework**
3. The RLBox sandboxing framework
4. Our experiences deploying RLBox

Example: sandboxing libjpeg from Firefox



Step 1: Compile libjpeg with Wasm



Step 2: Modify Firefox to use sandboxed libjpeg

(easier said than done!)

Decouple the library:

⇒ Decouple shared data / control flow

Enable data sharing:

⇒ Reconcile Firefox and Wasm ABI differences

⇒ Marshal data lazily for performance

Add security checks:

⇒ Library no longer trusted!

⇒ Sanitize library outputs, restrict control flow

```
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);
```

```
    }
```

```
}
```

Simple app that uses libjpeg


```

void create_jpeg_parser() {

    jpeg_decompress_struct* jpeg_img = in_sandbox(/* ... */);
    jpeg_error_mgr*          jpeg_err = in_sandbox(/* ... */);

    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 = /* ... */;
    /* ... */

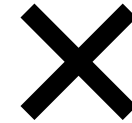
    while (/* check for output lines */) {
        uint32_t size = marshal_from(jpeg_img->output_width) * marshal_from(jpeg_img->output_components);
        assert(size <= outputBufferSize);
        memcpy(outputBuffer, /* ... */, size);
    }
}

```



This fails horribly in practice

1. Real systems are huge
⇒ lots of code changes
2. Details of sandboxing are exposed
⇒ code is unmaintainable
⇒ testing and debugging is unusable
⇒ portability is a nightmare



OpenSSL-1.0.2p
OpenSSL-1.0.2p
SQLite3-3.27.2
OpenCV-2.4.1
FFmpeg-2.8.0
GIFLib-5.1.1
XML2-2.7.7
Libpng-1.6.7
WebRTC-*
Libgraphite-*
Freetype-*
Libogg-*
...

This fails horribly 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.

Outline

1. Motivation
2. Why do we need a sandboxing framework
- 3. The RLBox sandboxing framework**
4. Our experiences deploying RLBox

RLBox: framework to retrofit sandboxing

Idea: Use types to make sandboxing a compositional abstraction

1. Types hide low-level sandbox details

Automates ABI conversions, data marshalling

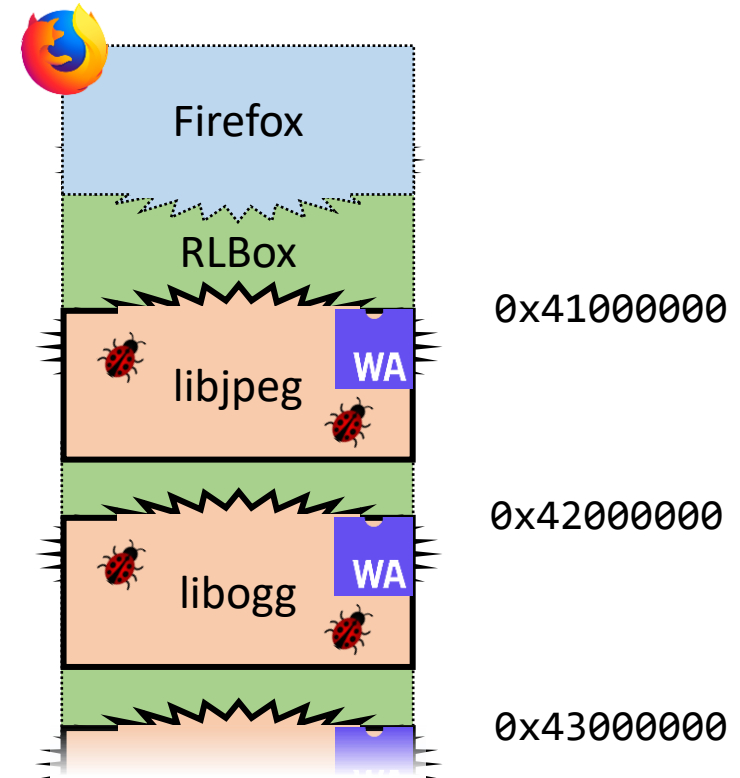
2. Types track untrusted data and control flow

Missing security checks \Rightarrow type errors

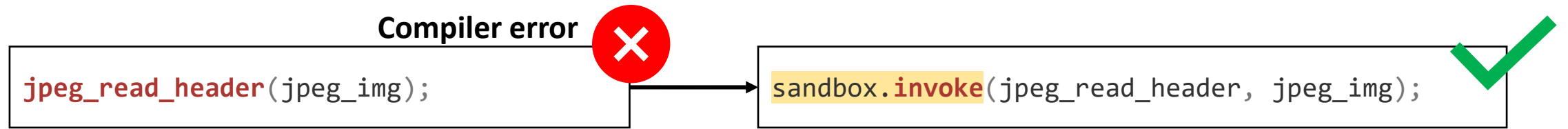
3. Types allow retrofitting piecemeal

Test, review, and deploy sandboxing incrementally

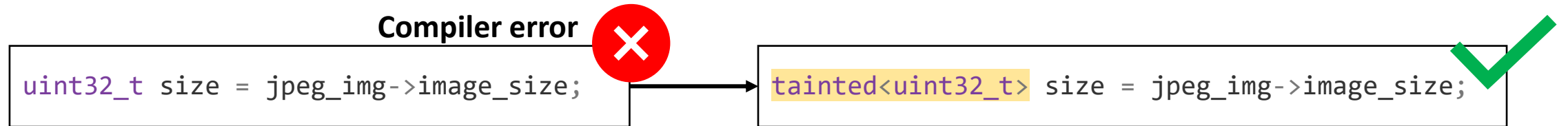
Implemented as a pure C++ library



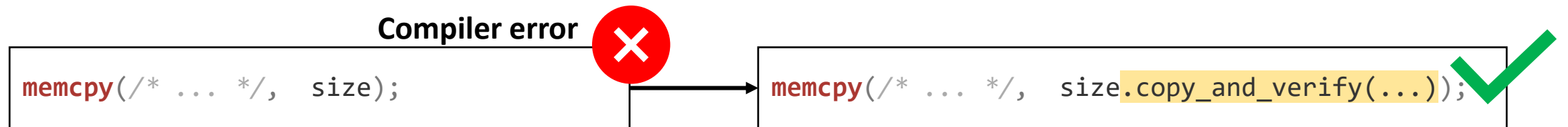
1. RLBox forces control flow to be explicit



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



3. Tainted data must be checked before use



```

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. Control flow through **sandbox.invoke**
2. Data from the sandbox must be **tainted**
3. Tainted data must be checked before use

```

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

    sandbox.invoke(jpeg_create_decompress, jpeg_img);
    jpeg_img->err = jpeg_err;

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

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

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

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

```

1. Control flow through **sandbox.invoke**
2. Data from the sandbox must be **tainted**
3. Tainted data must be checked before use

Automatic: ABI + marshalling + bounds checks


```

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

    sandbox.invoke(jpeg_create_decompress, jpeg_img);
    jpeg_img->err = jpeg_err;

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

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

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

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

```

1. Control flow through **sandbox.invoke**
2. Data from the sandbox must be **tainted**
3. Tainted data must be checked before use

The type of size is tainted

```

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

    sandbox.invoke(jpeg_create_decompress, jpeg_img);
    jpeg_img->err = jpeg_err;

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

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

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

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

```

1. Control flow through **sandbox.invoke**
2. Data from the sandbox must be **tainted**
3. Tainted data must be checked before use

Need to sanitize before use

```

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

    sandbox.invoke(jpeg_create_decompress, jpeg_img);
    jpeg_img->err = jpeg_err;

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

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

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

```

1. Control flow through **sandbox.invoke**
2. Data from the sandbox must be **tainted**
3. Tainted data must be checked before use

Wasm + RLBox = fast sandboxing + low engineering effort

Outline

1. Motivation
2. Why do we need a sandboxing framework
3. The RLBox sandboxing framework
4. Our experiences deploying RLBox

Engineering costs are reasonable



Sandboxed libraries

Image and font rendering

Audio video playback

Decompression

XML parsing

Spell checking

Automatic security checks

dozens to hundreds per library

Remaining data validation

on average, 2–4 lines of code

Time: a few days per library

Deployment 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

Mac, Linux

[Font rendering
Audio playback
Decompression
XML parsing
Spell checking]

Dec 2021

All platforms

Performance overheads is low

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.



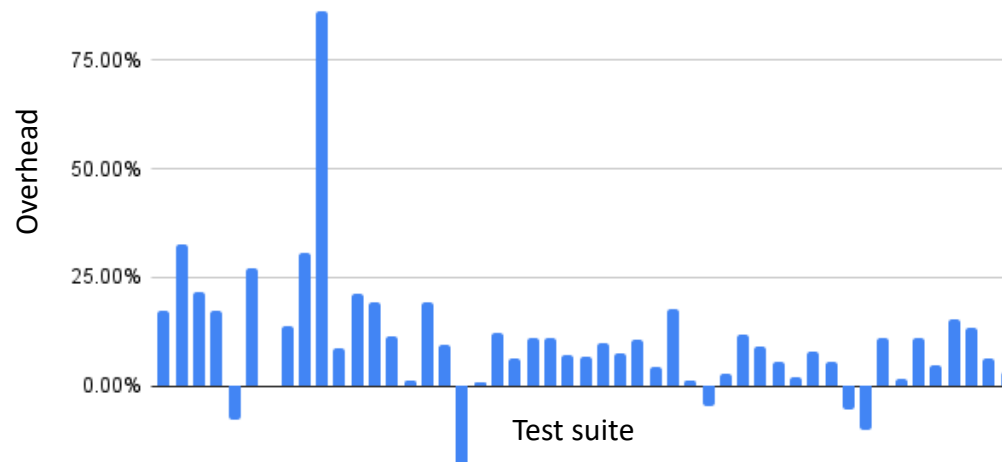
Jonathan Kew (:jfkthame) ▾

Comment 17 • 3 months ago • Edited



Font decompression

10.5%



We are not done!

1. Performance features

SIMD, Threads

2. Low resource environments

Reducing virtual memory footprint in 32-bit architectures

3. Usability

Updated APIs with modern C++ idioms, DSL for syscall policies

Help us get this right!

Try it out!

File bugs, feature requests

We are happy to help

RLBox is covered under Firefox's bug bounty

Add a memory corruption to an RLBox-ed library

Break out of the sandbox \Rightarrow bug bounty

<https://www.mozilla.org/en-US/security/client-bug-bounty/#exploit-mitigation-bounty>

In-process sandboxing: new technique for securing native code!

30 years in the making, finally in production!

RLBox: practical tool to sandbox native code w/ Wasm

Firefox has shipped RLBox for 2+ years, you can too!



@ShrNarayan



shr@cs.utexas.edu



RLBox

<https://rlbox.dev/>