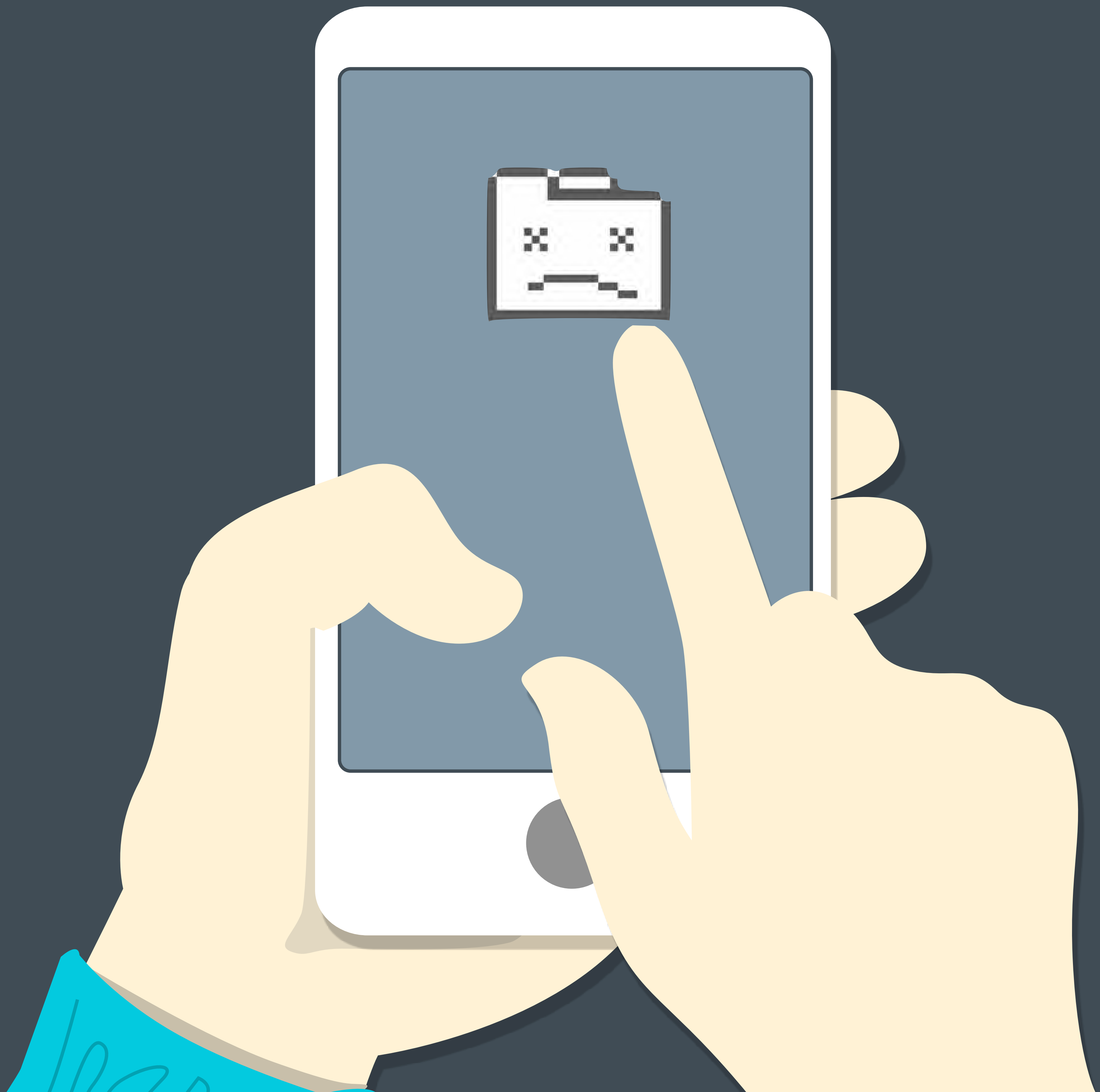
Hunting and Mobile



About us

COSEINC
Formerly Red Hat Product Security
Fedora Security Team
iOS Mobile Security
OSS Security

Francisco Alonso

Senior security researcher
rs@revskills.cz
@revskills

COSEINC
Focused on Linux Kernel
radare2 evangelist
r2pipe node mantainer
Whiskey Con Superstar

Jaime Peñalba

Senior security researcher
jpenalba@member.fsf.org
@NighterMan

Agenda

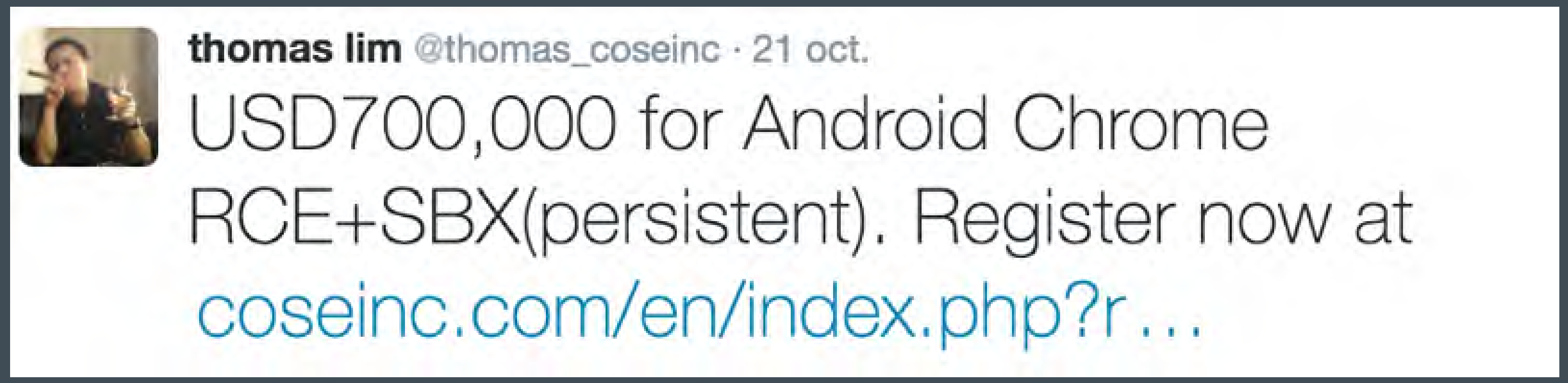
Browser Bug Hunting and Mobile

- Motivations
- Public vulnerability statistics and notes
- Mitigations
- Memory Instrumentation
- Code Coverage
- Fuzzing strategies
- Triage
- Conclusions
- Questions

Motivations

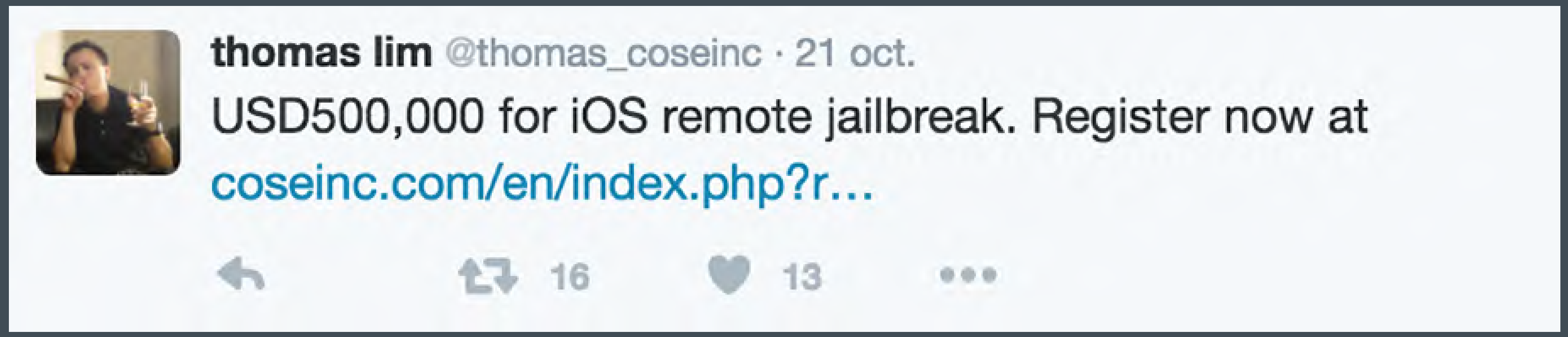
Browser Bug Hunting and Mobile

- Mobile PWN0RAMA, Pwn2Own, PWNFEST contests
- Coordinated ~~Responsible~~ disclosure
- Public Bug bounty programs
- 0day Market
- It's funny, Increasingly complicated and a competitive world
- Pop all the calcs!



thomas lim @thomas_coseinc · 21 oct.

USD700,000 for Android Chrome RCE+SBX(persistent). Register now at coseinc.com/en/index.php?r…

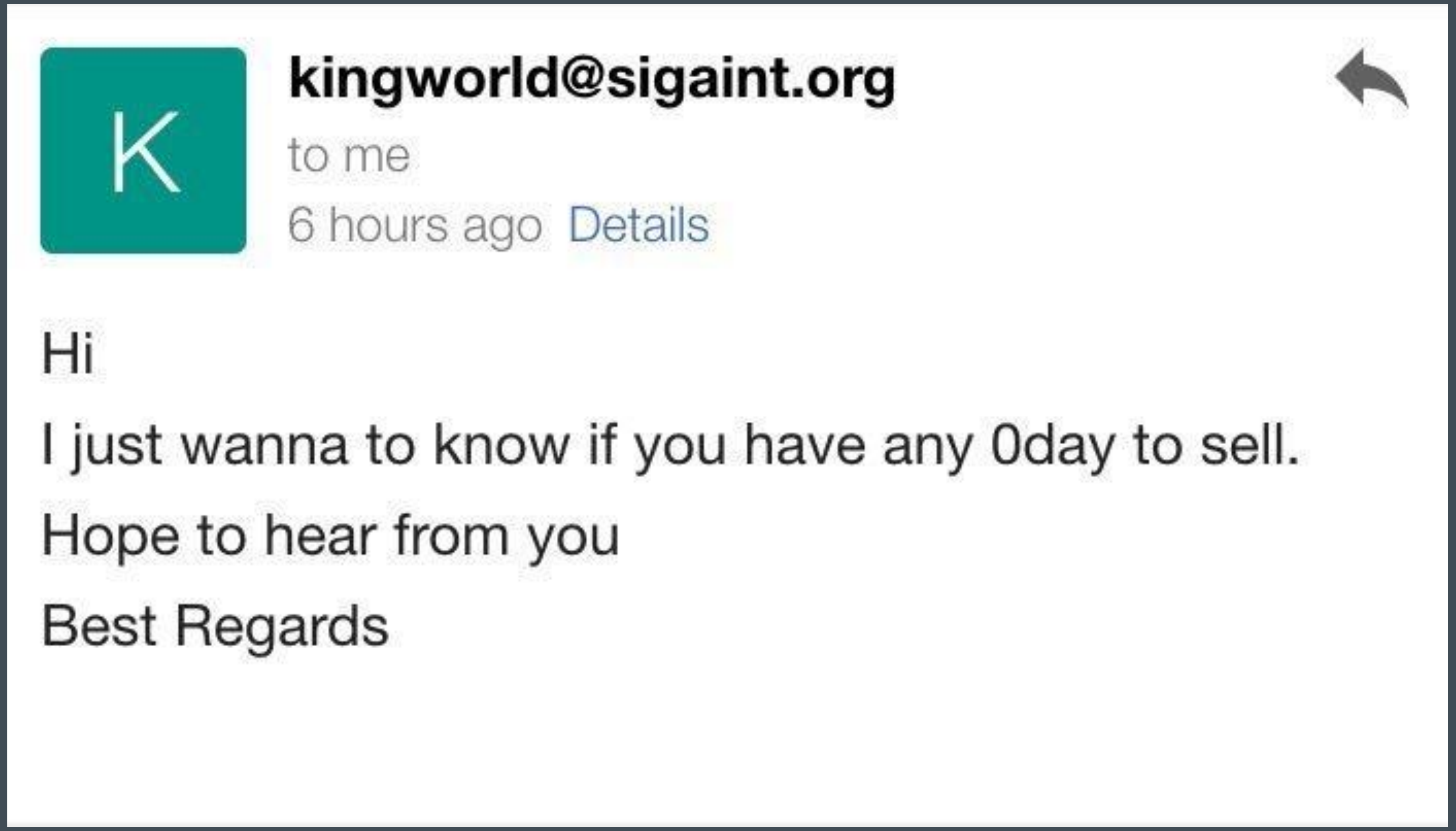


thomas lim @thomas_coseinc · 21 oct.

USD500,000 for iOS remote jailbreak. Register now at coseinc.com/en/index.php?r…

~~Motivations~~

Browser Bug Hunting and Mobile



kingworld@sigaint.org

to me

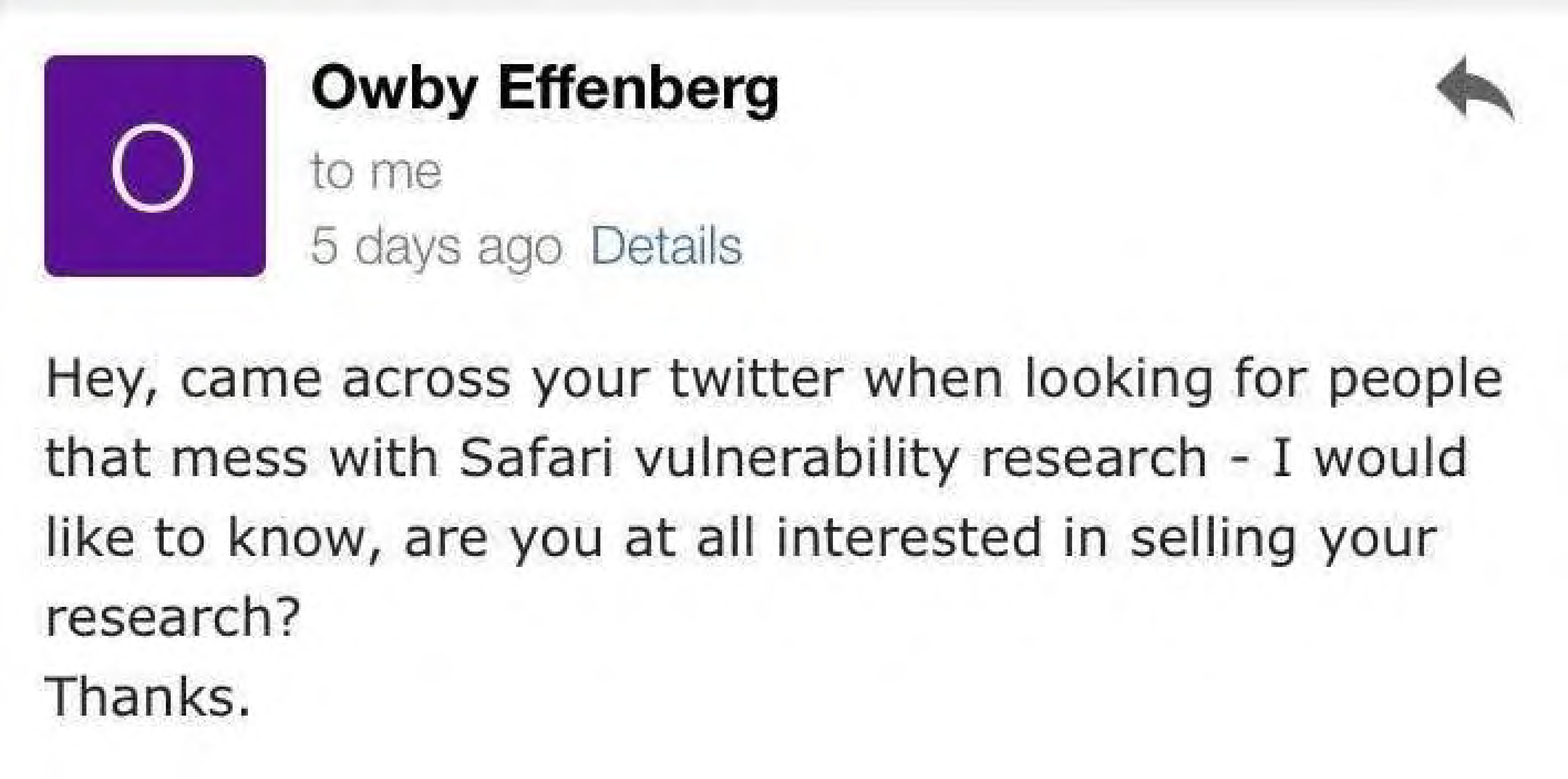
6 hours ago Details

Hi

I just wanna to know if you have any 0day to sell.

Hope to hear from you

Best Regards



Owby Effenberg

to me

5 days ago Details

Hey, came across your twitter when looking for people that mess with Safari vulnerability research - I would like to know, are you at all interested in selling your research?

Thanks.

independent broker-dealers

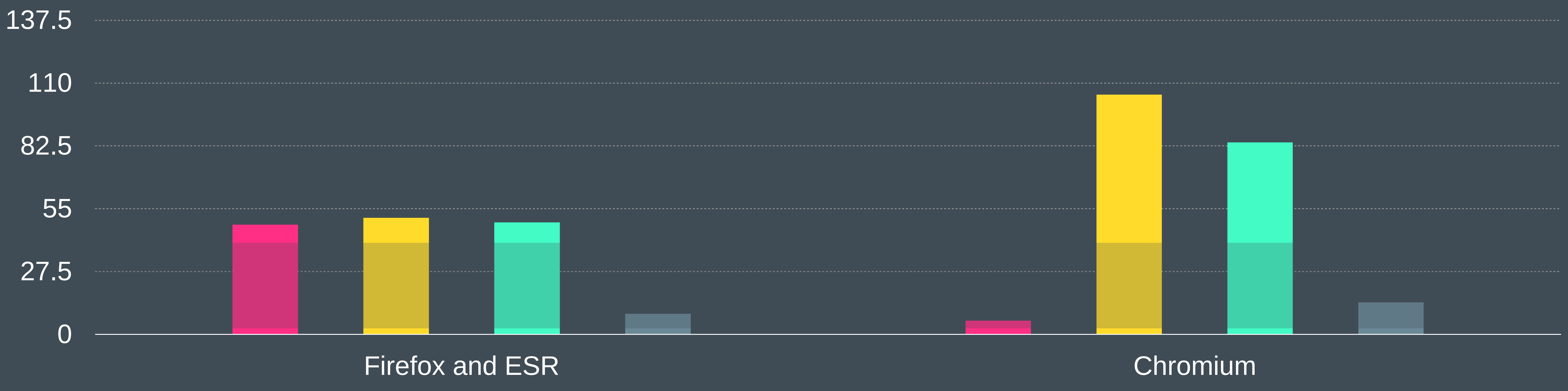
Public vulnerability statistics

Browser Bug Hunting and Mobile

- **Mozilla:**
  - **~14,045,424 LOC. C++,C, JavaScript, Rust..**
  - **3.528 Commits, 373 Contributors, 30 days.**
- **Chromium (Google Chrome)**
  - **~14,941,151 LOC. C++, C..**
  - **6809 Commits, 817 Contributors, 30 days**
- **WebKit**
  - **~8,398,258 LOC. C++**
  - **1214 Commits, 76 Contributors, 30 days**

Public vulnerability statistics

Browser Bug Hunting and Mobile



2016 (January - October/November, Aprox)

Public vulnerability statistics

Browser Bug Hunting and Mobile

- **Chromium : Most bugs reported (even if they use the same CVE identifier come from internal audits)**
- **Cross third party libraries common bugs: Begin to be uncommon, become more robust. Eg:**
  - **libpng**
  - **jpeglib**
- **Many bugs stuck in bugzilla for months**
- **Lots of bugs reported to Mozilla by Chromium Product Security**
- **Lots of bugs reported to WebKit by Chromium Product Security**
- **Several Blink commiters maintains WebKit too**
- **Lack of information intentionally, private bug reports, diff required**
  - **CVE-2016-5200: Out of bounds memory access in V8**
  - **CVE-2016-4657: A memory corruption issue was addressed through improved memory handling (NSO)**
- **Backporting is a mess, Linux distributions rebase Chrome and Firefox**

Public vulnerability statistics

ClusterFuzz Fuzzing at Scale

- **App Engine Google Cloud Platform (Fronted)**
  - **Windows, Linux VMs**
- **Google Chrome lab (Backend)**
  - **Android and iOS devices, macOS Servers, GPU Linux**
- **> 5.000 24x7 CPU cores**
- **> 5.000 bugs in Chromium, >1.200 bugs in ffmpeg**
- **Hundreds of custom fuzzers testing different APIs**
- **Several Teams working on different fuzzers (libFuzzer, afl/afl_driver, etc)**
- **Blink - Webkit**

Public vulnerability statistics

Mozilla Fuzzing at Scale

- Amazon EC2 VMs
- No public information about VMs/Cores
- Funfuzz: jsfunfuzz and DOMFuzz
- FuzzManager: A fuzzing management tools collection
- CrashManager
- Laniakea: tool for managing EC2 instances at AWS
- Quokka: launch and monitor application for faults
- Dharma: generation-based, context-free grammar fuzzer
- Faulty: fuzzing IPC Protocol Definition Language (IPDL) protocols
- fuzzdata: resources for feeding various fuzzers with input
- Framboise: in-depth testing of WebAPIs (WebVTT, Canvas2D,etc)

Mitigations

Evolution

- **VTGuard**
- **ForceASLR**
- **AppContainer**
- **Pool Integrity Checks**
- **Kernel ASLR**
- **EMET**
- **PartitionAlloc**
- **Java Click-to-Play**
- **Control Flow Guard**
- **Isolated Heap**
- **Memory Protection**
- **Win32k Access Prevention**
- **Adobe Flash Isolated Heap**
- **Adobe Flash Memory Protections**

Source: Zero Day Initiative Research

- **Hardened JIT Mapping**
- **iOS Sandbox Hardening**
- **iPhone 7 New protections**

Mitigations

Typical Exploit-Chain

B

Browser

Compromise Render (WebKit/Blink) via HTML, DOM, CSS, SVG, Canvas, JavaScript Engine (JavaScriptCore, v8)

S

Sandbox

Code execution, cookie leak

B

Sandbox Bypass

Code execution out of sandbox, Data Leakage, IPC

P

Privilege Escalation

Kernel, persistence

Mitigations

inter-process communication (IPC) basic rules

- Trust only the browser process
- Do not trust renderer, PPAPI (Pepper API, Flash), or GPU processes
- Sanitize and validate untrustworthy input. Directory traversal attacks, file theft.
- Android: integer types across C++ and Java (safe conversions)
- Information leak of addresses/pointers over the IPC channel (Don't defeat ASLR)

Memory Instrumentation

Not all memory access errors result in crashes

- **AddressSanitizer**
- **ThreadSanitizer**
- **MemorySanitizer**
- **UndefinedBehaviorSanitizer**
- **SyzyASan**
- **PageHeap**

Memory Instrumentation

Not all memory access errors result in crashes

AddressSanitizer (ASan): Fast memory error detector (slowdown 2x).
It consists of a compiler instrumentation module and a run-time library.
The tool can detect the following types of bugs:

- **Out-of-bounds accesses to heap, stack and globals**
- **Use-after-free**
- **Use-after-return**
- **Use-after-scope**
- **Double-free, invalid free**
- **Memory leaks (LSan)**

-fsanitize=address

Memory Instrumentation

Not all memory access errors result in crashes

ThreadSanitizer (TSan): focuses on concurrency issues. Slowdown 5x-15x, memory overhead 5x-10x

- **Data races**
- **Deadlocks**
- **Unjoined threads**
- **C++ and Go**

-fsanitize=thread

Memory Instrumentation

Not all memory access errors result in crashes

MemorySanitizer (MSan): focuses on contents of memory. Slowdown 3x

- **Uninitialized reads**
- **Origin Tracking**
- **Use-after-destruction (experimental)**

-fsanitize=memory

Memory Instrumentation

Not all memory access errors result in crashes

UndefinedBehaviorSanitizer (UBSan): detect various kinds of undefined behavior.

- **Using misaligned or null pointer**
- **Signed integer overflow**
- **Conversion to, from, or between floating-point types which would overflow the destination**
- **UBSAN_OPTIONS=halt_on_error=1**

-fsanitize=undefined

Memory Instrumentation

Not all memory access errors result in crashes

Control Flow Integrity (CFI): detect certain forms of undefined behavior that can potentially allow to subvert the program's control flow. Optimized for performance

- **Different subset of schemes**
- **Require LTO (link-time optimization)**

-fsanitize=cfi

Memory Instrumentation

Not all memory access errors result in crashes

SafeStack: protects against attacks based on stack buffer overflows. Overhead is less than 0.1%.

- **Two distinct regions: safe and unsafe stack**
- **Part of the Code-Pointer Integrity (CPI) Project**
- **Some limitations: protection against arbitrary memory write vulnerabilities is probabilistic and relies on randomization and information hiding.**

-fsanitize=safe-stack

Code coverage

coverage at a very low cost.

- **SanitizerCoverage: it can be used with ASan, LSan, MSan, and UBSan or without Allows to get function-level, basic-block-level, and edge-level**

-fsanitize-coverage=func for function-level coverage, fast.

-fsanitize-coverage=bb for basic-block-level coverage > to 30% extra slowdown

-fsanitize-coverage=edge for edge-level coverage. > 40% slowdown Splits all critical edges by introducing new dummy blocks

-fsanitize-coverage=8bit-counters, to get coverage counters,

Memory Instrumentation

- Google (Chromium, Chromium OS, Chrome/Android) and Mozilla provide public daily ASan builds,
- testing and debugging. Use your own builds
- WebKitGTK+ and WebKit are ASan friendly
- JavaScriptCore: asanUnsafeJSValue, CopyMemory
- It is possible to build WebKit iOS with ASan to use on iPhone Simulator (it is basically x86)
- AddressSanitizer it is NOT a mitigation/hardening. Tor Hardened Browser.. You're doing it wrong.

Fuzzing strategies

- The term "fuzz" or "fuzzing" originates from a 1988 class project, taught by Barton Miller at the University of Wisconsin. —Wikipedia

- Goal: trigger an application crash or unexpected behaviour

- Mutation (dumb fuzzing): mutate existing test samples.
    - Shuffle, change, erase, insert

- Generation (smart/intelligent fuzzing): define new test samples based on models, templates, RFC or documentation
    - Web IDL, XML Schemas

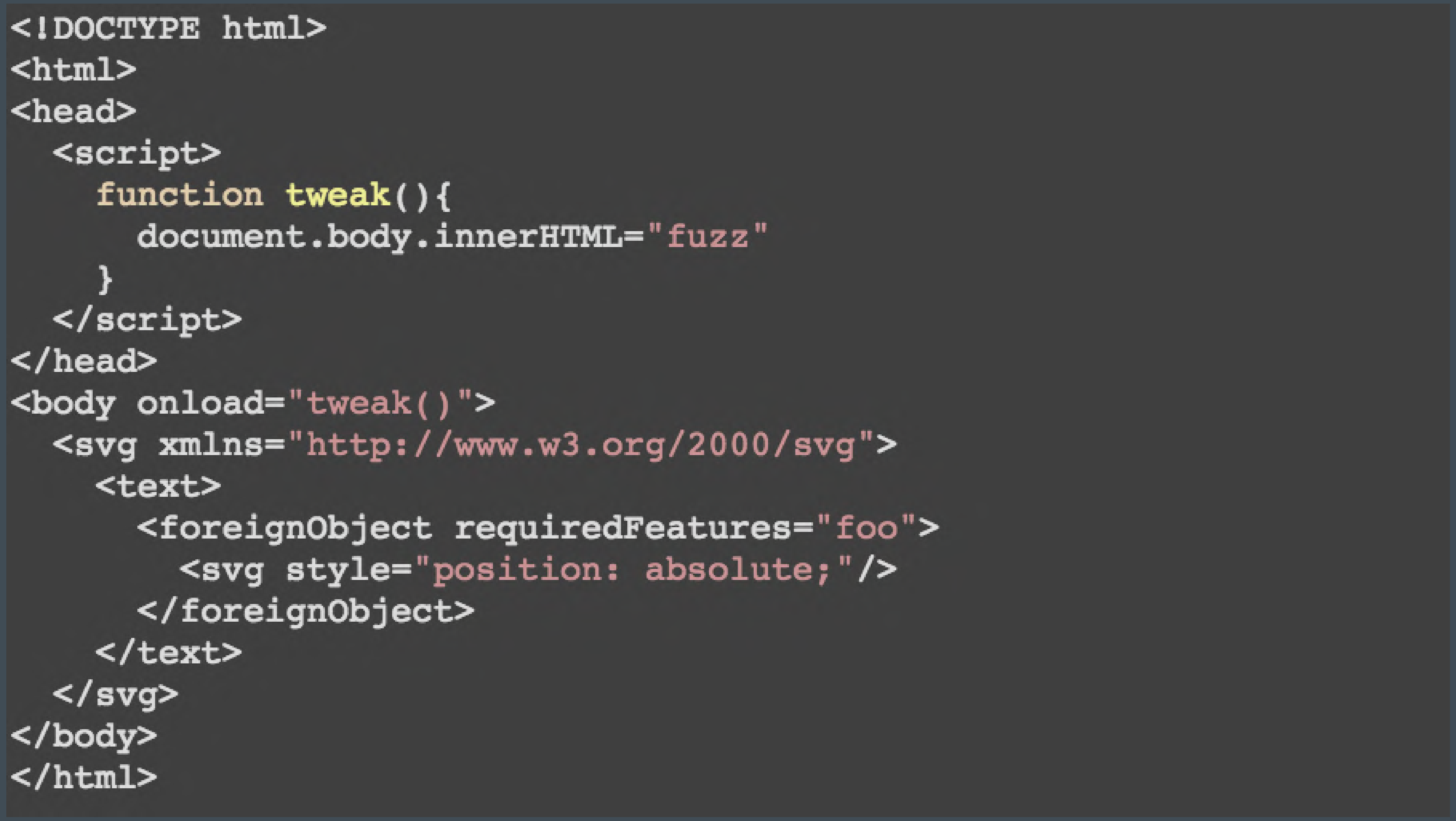
Fuzzing strategies

Smart Generation Fuzzing DOM

Mozilla Firefox
Regression bug #1182496
Mitigated by Frame-Poisoning
Every object that is being freed
will be replaced with a chosen pattern.
Implemented in nsPresArena

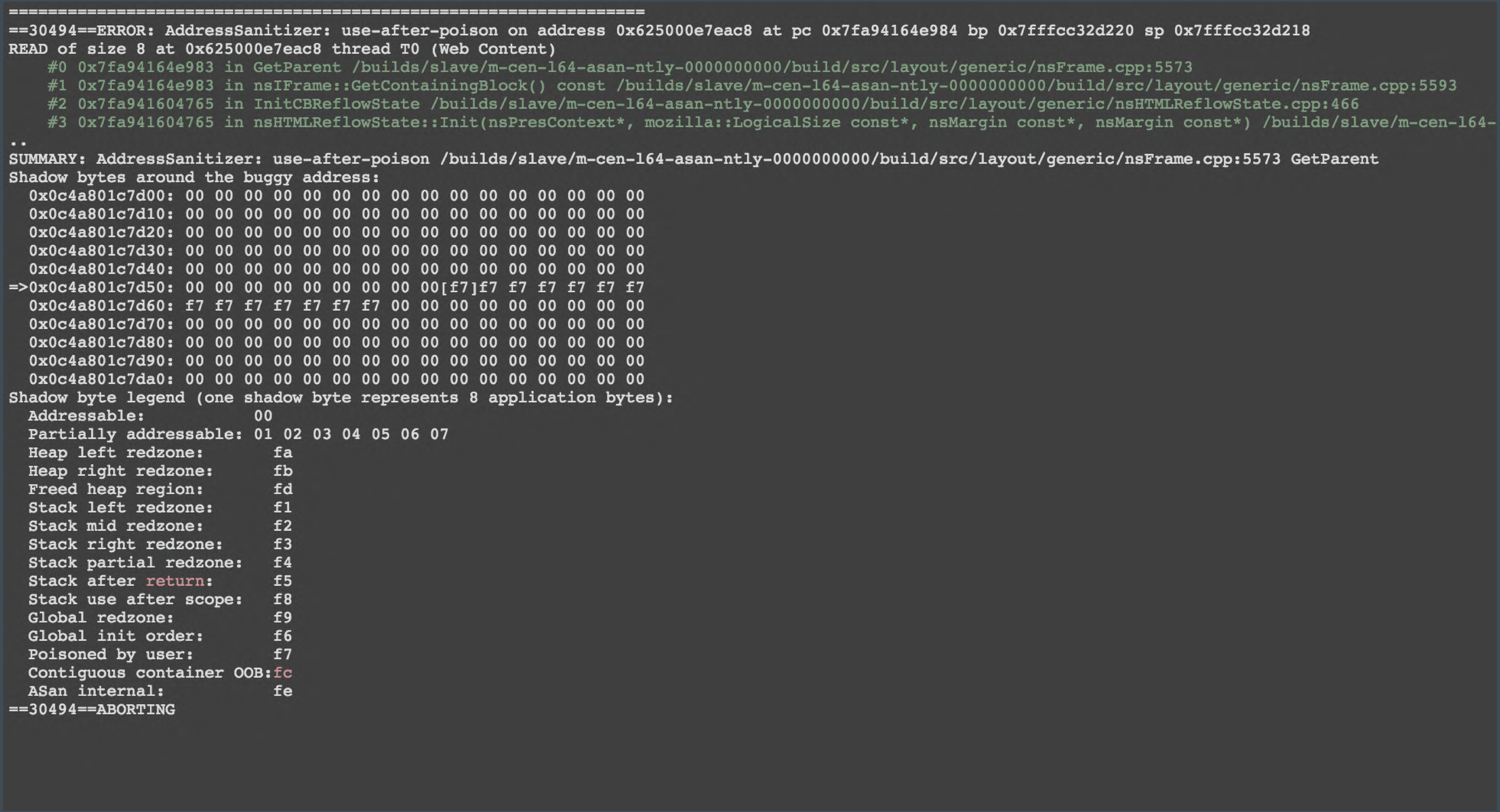
Incorrect mParent pointer is pointing into a subtree that's been destroyed.

SVGForeignObjectElement
https://www.w3.org/TR/2011/REC-SVG11-20110816/svg.idl



```
<!DOCTYPE html>
<html>
<head>
  <script>
    function tweak(){
      document.body.innerHTML="fuzz"
    }
  </script>
</head>
<body onload="tweak()">
  <svg xmlns="http://www.w3.org/2000/svg">
    <text>
      <foreignObject requiredFeatures="foo">
        <svg style="position: absolute;"/>
      </foreignObject>
    </text>
  </svg>
</body>
</html>
```

Fuzzing strategies



```
=================================================================
==30494==ERROR: AddressSanitizer: use-after-poison on address 0x625000e7eac8 at pc 0x7fa94164e984 bp 0x7fffcc32d220 sp 0x7fffcc32d218
READ of size 8 at 0x625000e7eac8 thread T0 (Web Content)
    #0 0x7fa94164e983 in GetParent /builds/slave/m-cen-l64-asan-ntly-0000000000/build/src/layout/generic/nsFrame.cpp:5573
    #1 0x7fa94164e983 in nsIFrame::GetContainingBlock() const /builds/slave/m-cen-l64-asan-ntly-0000000000/build/src/layout/generic/nsFrame.cpp:5593
    #2 0x7fa941604765 in InitCBReflowState /builds/slave/m-cen-l64-asan-ntly-0000000000/build/src/layout/generic/nsHTMLReflowState.cpp:466
    #3 0x7fa941604765 in nsHTMLReflowState::Init(nsPresContext*, mozilla::LogicalSize const*, nsMargin const*, nsMargin const*) /builds/slave/m-cen-l64-
..
SUMMARY: AddressSanitizer: use-after-poison /builds/slave/m-cen-l64-asan-ntly-0000000000/build/src/layout/generic/nsFrame.cpp:5573 GetParent
Shadow bytes around the buggy address:
  0x0c4a801c7d00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c4a801c7d10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c4a801c7d20: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c4a801c7d30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c4a801c7d40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
=>0x0c4a801c7d50: 00 00 00 00 00 00 00 00 00[f7]f7 f7 f7 f7 f7 f7
  0x0c4a801c7d60: f7 f7 f7 f7 f7 f7 f7 00 00 00 00 00 00 00 00 00
  0x0c4a801c7d70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c4a801c7d80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c4a801c7d90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0x0c4a801c7da0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Heap right redzone:      fb
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack partial redzone:   f4
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Contiguous container OOB:fc
  ASan internal:           fe
==30494==ABORTING
```

Fuzzing strategies

Smart Generation, Notes

- Generic, valid for several browsers
- Not all meet specifications, MATHML
- Requires a good infrastructure
  - Servers
  - ASan,UBSan… Builds per Browser
  - Monitor , crash Manager (dumps)
  - Maintenance
  - Fairly expensive to maintain
  - Too much can go wrong

Fuzzing strategies

ECMAScript Engines

- Redefinition: redefine methods, __defineGetter__, __defineSetter__, __lookupGetter__
- ANTLR ANother Tool for Language Recognition/Esprima tool/acorn.js, generates a parser that can build and walk parse trees.
- Testsuite, code snippets, converts to AST (Abstract syntax tree)
  - Replace nodes
  - Shuffle
  - Replace Values
  - Not random at all, heuristics are better
  - Validate them and test against:
    - v8 (Chromium)
    - JavaScriptCore (Webkit/Safari)
    - SpiderMonkey (Firefox)

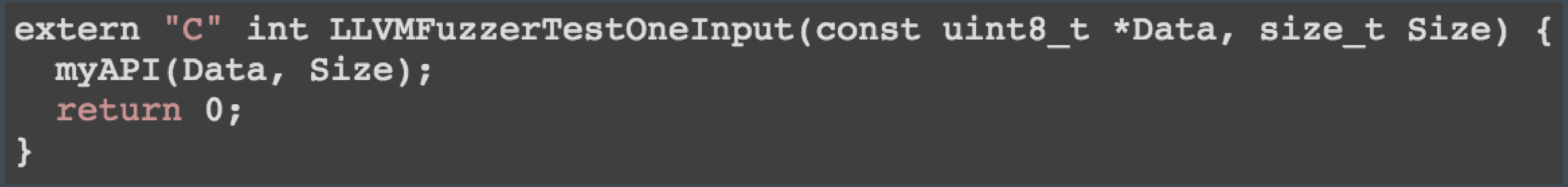
Fuzzing strategies

Smart Generation, Notes

- **Almost Generic, valid for several ECMA Engines**
- **ASan,UBSan… Builds per Engine**
- **Does not require too much infrastructure**
- **They are quite robust in general**

Fuzzing strategies

LibFuzzer



```
extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
  myAPI(Data, Size);
  return 0;
}
```

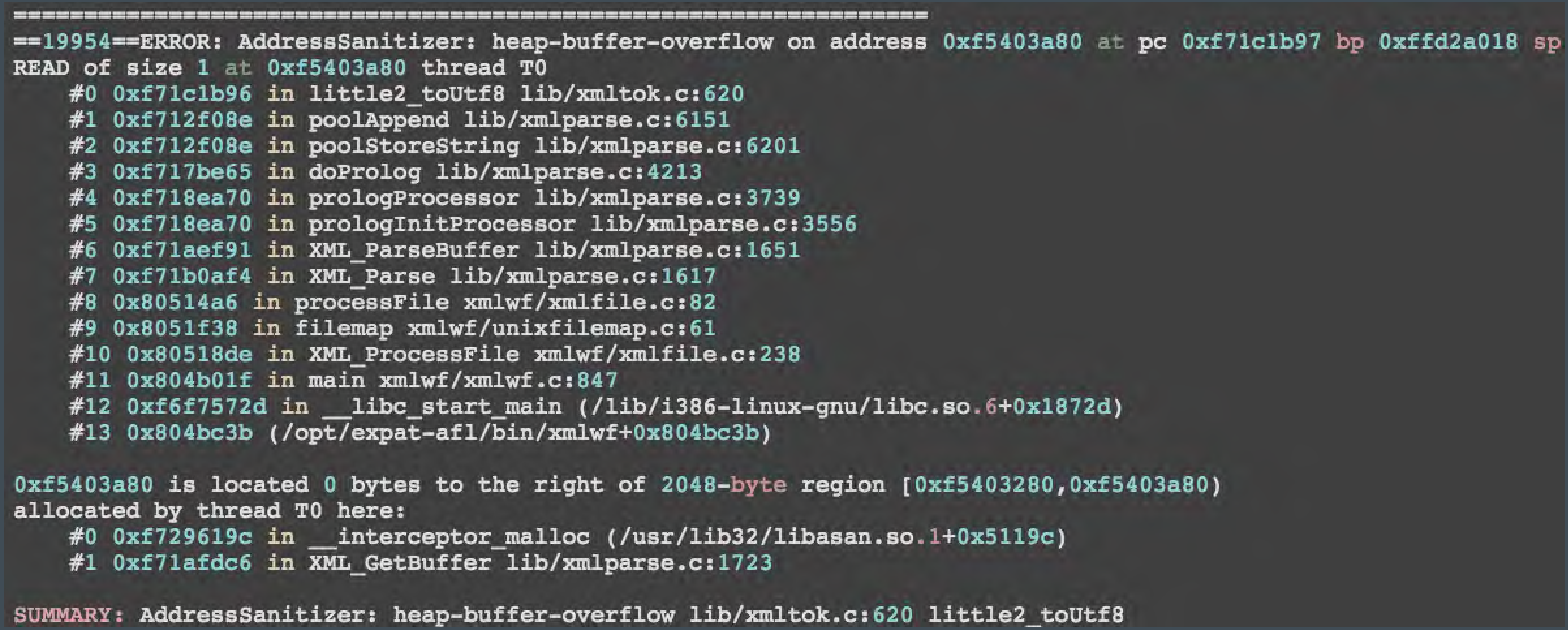
- **It uses LLVM’s SanitizerCoverage instrumentation to get in-process coverage-feedback**
- **Integrated with ASan, MSan, UBsan, LSan**
- **Fast, no overhead at start-up**
- **Perfect way to start your own fuzzer**
  - **Custom Mutators FuzzerInterface.h**
  - **Different mutators = Different results**
  - **LLVMFuzzerTestOneInput: Function metrics**

Fuzzing strategies

LibFuzzer & expat example

```
clang -std=c++11 -Ilib/ expat_fuzzer.cc -o expat_fuzzer \
      -lfuzzer .libs/libexpat.a

./expat_fuzzer SAMPLES/ -jobs=7 -workers=7 -dict=xml.dict
```



```
=================================================================
==19954==ERROR: AddressSanitizer: heap-buffer-overflow on address 0xf5403a80 at pc 0xf71c1b97 bp 0xffd2a018 sp
READ of size 1 at 0xf5403a80 thread T0
    #0 0xf71c1b96 in little2_toUtf8 lib/xmltok.c:620
    #1 0xf712f08e in poolAppend lib/xmlparse.c:6151
    #2 0xf712f08e in poolStoreString lib/xmlparse.c:6201
    #3 0xf717be65 in doProlog lib/xmlparse.c:4213
    #4 0xf718ea70 in prologProcessor lib/xmlparse.c:3739
    #5 0xf718ea70 in prologInitProcessor lib/xmlparse.c:3556
    #6 0xf71aef91 in XML_ParseBuffer lib/xmlparse.c:1651
    #7 0xf71b0af4 in XML_Parse lib/xmlparse.c:1617
    #8 0x80514a6 in processFile xmlwf/xmlfile.c:82
    #9 0x8051f38 in filemap xmlwf/unixfilemap.c:61
    #10 0x80518de in XML_ProcessFile xmlwf/xmlfile.c:238
    #11 0x804b01f in main xmlwf/xmlwf.c:847
    #12 0xf6f7572d in __libc_start_main (/lib/i386-linux-gnu/libc.so.6+0x1872d)
    #13 0x804bc3b (/opt/expat-afl/bin/xmlwf+0x804bc3b)

0xf5403a80 is located 0 bytes to the right of 2048-byte region [0xf5403280,0xf5403a80)
allocated by thread T0 here:
    #0 0xf729619c in __interceptor_malloc (/usr/lib32/libasan.so.1+0x5119c)
    #1 0xf71afdc6 in XML_GetBuffer lib/xmlparse.c:1723

SUMMARY: AddressSanitizer: heap-buffer-overflow lib/xmltok.c:620 little2_toUtf8
```

Fuzzing strategies

LibFuzzer Dictionaries

- **Dictionaries FuzzerDictionary.h :**
- **Automatic**

  - Intercepts memcp, strcmp. See FuzzerTracePC.cpp:212
- **Manual**

  - Token based like XML or magic value like PNG

  - Speed-up fuzzing with valid inputs (avoid large dictionaries)
- **ProTip :**

  Strip symbols, extract .rodata segment from our binary target, extract strings using different encodes and cross references with an rfc, documentation, etc.

Fuzzing strategies

LibFuzzer Notes

- **Bad Interaction with multithreaded binaries (8bit counters)**

- **White/blacklists/"hacks" are needed to avoid "noisy" coverage detection and improve performance. Like in v8 GC events.**

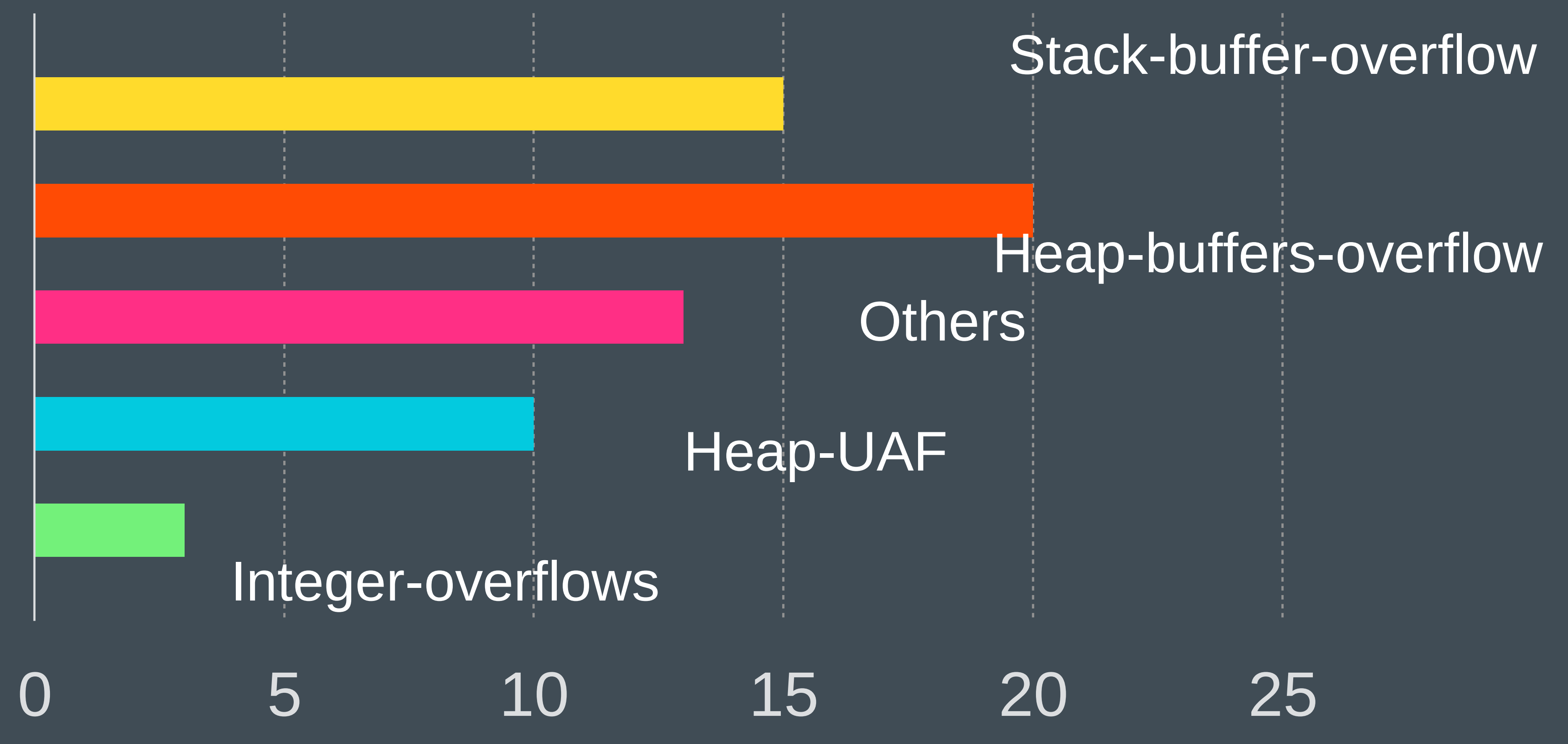
- **Not everything is perfect.. but it works great!**

Fuzzing strategies

LibFuzzer

- json_parser_libfuzzer.cc
- moz_ipc_libfuzzer.cc
- moz_worker_s_libfuzzer.cc
- cairo_surf_libfuzzer.cc
- graphite2_libfuzzer.cc
- wasm_libfuzzer.cc
- regexp_libfuzzer.cc
- pdfium_icc2_libfuzzer.cc
- skia_binary_in_libfuzzer.cc
- skia_api_various_libfuzzer.cc
- skia_canvas_libfuzzer.cc
- skia_encoder_libfuzzer.cc
- skia_path_x_libfuzzer.cc
- audio_dec_libfuzzer.cc
- audio_enc_libfuzzer.cc
- expat_encodes_libfuzzer.cc

- libpng_libfuzzer.cc
- h264_libfuzzer.cc
- gstreamer_s_libfuzzer.cc
- freetype_sim_libfuzzer.cc
- freetype_optimized_libfuzzer.cc
- wof2_libfuzzer.cc
- vp8_libfuzzer.cc
- vp9_libfuzzer.cc
- libvpx_webm_libfuzzer.cc
- http_proxy_libfuzzer.cc
- file_libfuzzer.cc
- libxml2_libfuzzer.cc
- cert_various_libfuzzer.cc
- gl_s_libfuzzer.cc
- jsc_libfuzzer.cc
- v8_ast_libfuzzer.cc



~58 bugs in 30days

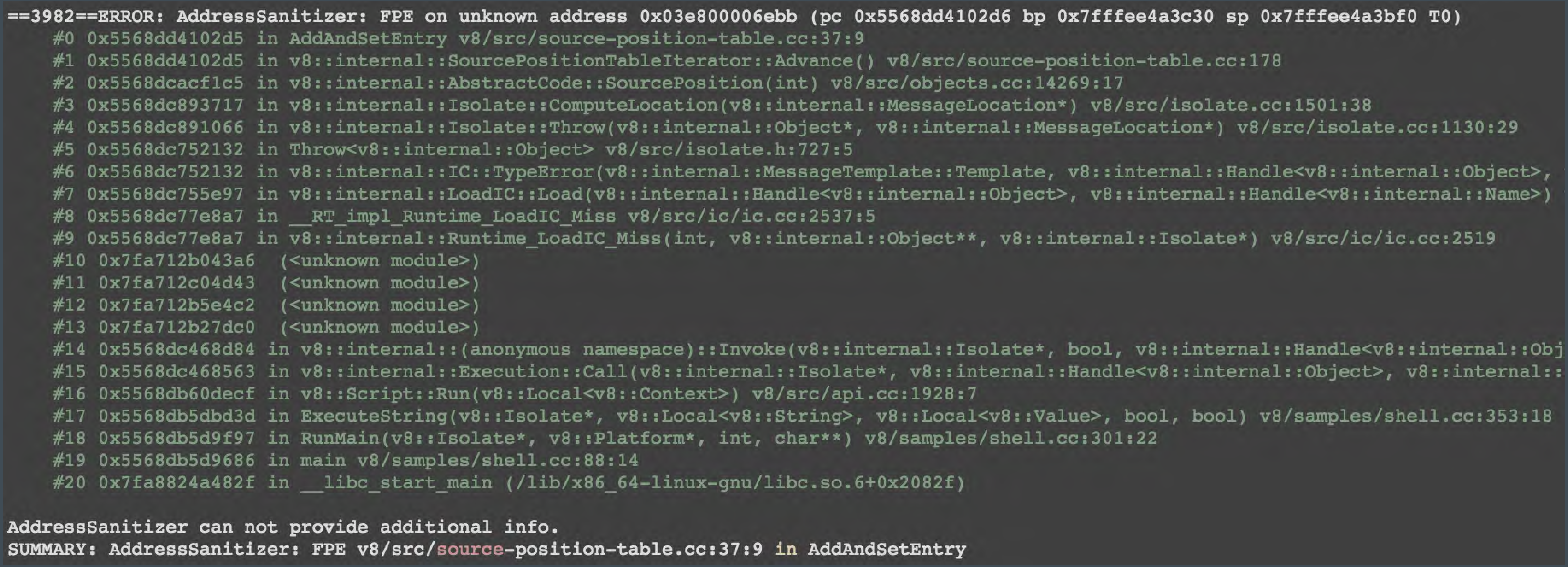
> 70 Fuzzers

Not 24x7 HW

Every interesting API in Chromium, Mozilla, Webkit and APIs from third party libraries

Fuzzing strategies

v8 Nov 20 (3 days ago), Fixed Yesterday

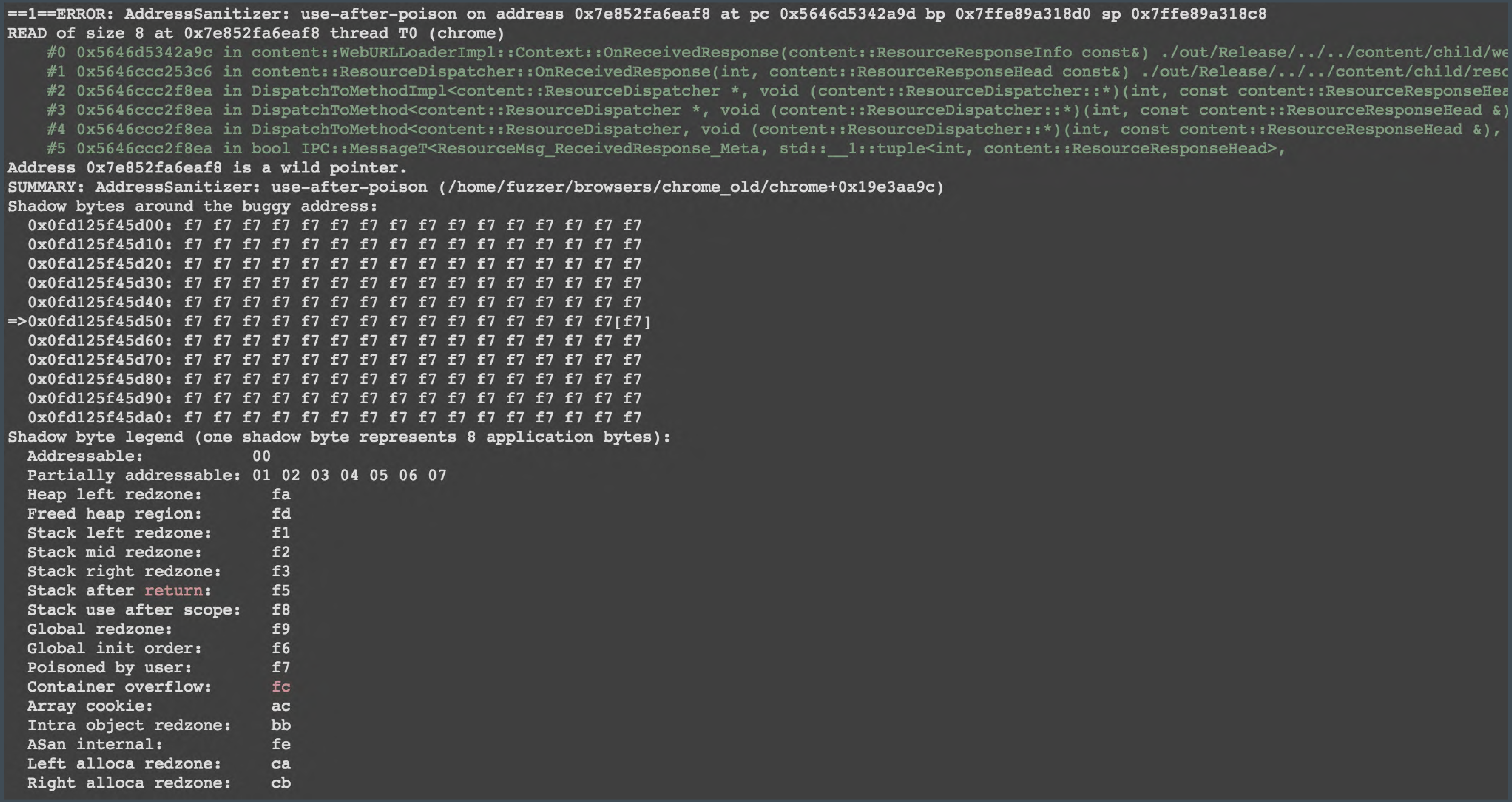


```
==3982==ERROR: AddressSanitizer: FPE on unknown address 0x03e800006ebb (pc 0x5568dd4102d6 bp 0x7fffee4a3c30 sp 0x7fffee4a3bf0 T0)
    #0 0x5568dd4102d5 in AddAndSetEntry v8/src/source-position-table.cc:37:9
    #1 0x5568dd4102d5 in v8::internal::SourcePositionTableIterator::Advance() v8/src/source-position-table.cc:178
    #2 0x5568dcacf1c5 in v8::internal::AbstractCode::SourcePosition(int) v8/src/objects.cc:14269:17
    #3 0x5568dc893717 in v8::internal::Isolate::ComputeLocation(v8::internal::MessageLocation*) v8/src/isolate.cc:1501:38
    #4 0x5568dc891066 in v8::internal::Isolate::Throw(v8::internal::Object*, v8::internal::MessageLocation*) v8/src/isolate.cc:1130:29
    #5 0x5568dc752132 in Throw<v8::internal::Object> v8/src/isolate.h:727:5
    #6 0x5568dc752132 in v8::internal::IC::TypeError(v8::internal::MessageTemplate::Template, v8::internal::Handle<v8::internal::Object>,
    #7 0x5568dc755e97 in v8::internal::LoadIC::Load(v8::internal::Handle<v8::internal::Object>, v8::internal::Handle<v8::internal::Name>)
    #8 0x5568dc77e8a7 in __RT_impl_Runtime_LoadIC_Miss v8/src/ic/ic.cc:2537:5
    #9 0x5568dc77e8a7 in v8::internal::Runtime_LoadIC_Miss(int, v8::internal::Object**, v8::internal::Isolate*) v8/src/ic/ic.cc:2519
    #10 0x7fa712b043a6  (<unknown module>)
    #11 0x7fa712c04d43  (<unknown module>)
    #12 0x7fa712b5e4c2  (<unknown module>)
    #13 0x7fa712b27dc0  (<unknown module>)
    #14 0x5568dc468d84 in v8::internal::(anonymous namespace)::Invoke(v8::internal::Isolate*, bool, v8::internal::Handle<v8::internal::Obj
    #15 0x5568dc468563 in v8::internal::Execution::Call(v8::internal::Isolate*, v8::internal::Handle<v8::internal::Object>, v8::internal::
    #16 0x5568db60decf in v8::Script::Run(v8::Local<v8::Context>) v8/src/api.cc:1928:7
    #17 0x5568db5dbd3d in ExecuteString(v8::Isolate*, v8::Local<v8::String>, v8::Local<v8::Value>, bool, bool) v8/samples/shell.cc:353:18
    #18 0x5568db5d9f97 in RunMain(v8::Isolate*, v8::Platform*, int, char**) v8/samples/shell.cc:301:22
    #19 0x5568db5d9686 in main v8/samples/shell.cc:88:14
    #20 0x7fa8824a482f in __libc_start_main (/lib/x86_64-linux-gnu/libc.so.6+0x2082f)

AddressSanitizer can not provide additional info.
SUMMARY: AddressSanitizer: FPE v8/src/source-position-table.cc:37:9 in AddAndSetEntry
```

Fuzzing strategies

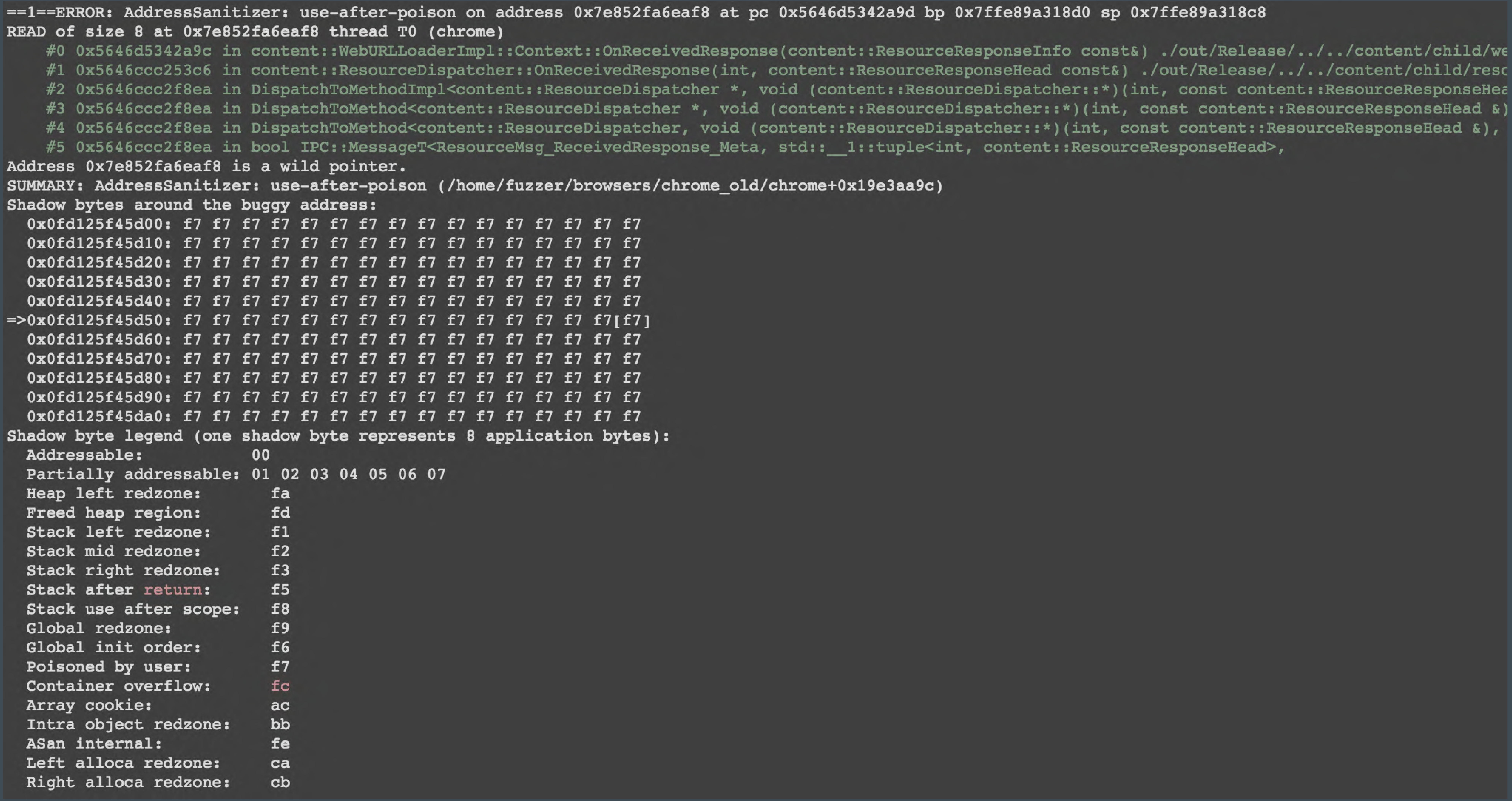
Components: Blink>Loader



```
==1==ERROR: AddressSanitizer: use-after-poison on address 0x7e852fa6eaf8 at pc 0x5646d5342a9d bp 0x7ffe89a318d0 sp 0x7ffe89a318c8
READ of size 8 at 0x7e852fa6eaf8 thread T0 (chrome)
    #0 0x5646d5342a9c in content::WebURLLoaderImpl::Context::OnReceivedResponse(content::ResourceResponseInfo const&) ./out/Release/../../content/child/we
    #1 0x5646ccc253c6 in content::ResourceDispatcher::OnReceivedResponse(int, content::ResourceResponseHead const&) ./out/Release/../../content/child/reso
    #2 0x5646ccc2f8ea in DispatchToMethodImpl<content::ResourceDispatcher *, void (content::ResourceDispatcher::*)(int, const content::ResourceResponseHea
    #3 0x5646ccc2f8ea in DispatchToMethod<content::ResourceDispatcher *, void (content::ResourceDispatcher::*)(int, const content::ResourceResponseHead &)
    #4 0x5646ccc2f8ea in DispatchToMethod<content::ResourceDispatcher, void (content::ResourceDispatcher::*)(int, const content::ResourceResponseHead &),
    #5 0x5646ccc2f8ea in bool IPC::MessageT<ResourceMsg_ReceivedResponse_Meta, std::__1::tuple<int, content::ResourceResponseHead>,
Address 0x7e852fa6eaf8 is a wild pointer.
SUMMARY: AddressSanitizer: use-after-poison (/home/fuzzer/browsers/chrome_old/chrome+0x19e3aa9c)
Shadow bytes around the buggy address:
  0x0fd125f45d00: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7
  0x0fd125f45d10: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7
  0x0fd125f45d20: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7
  0x0fd125f45d30: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7
  0x0fd125f45d40: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7
=>0x0fd125f45d50: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7[f7]
  0x0fd125f45d60: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7
  0x0fd125f45d70: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7
  0x0fd125f45d80: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7
  0x0fd125f45d90: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7
  0x0fd125f45da0: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
```

Fuzzing strategies

IPC componente, sec-high fixed in Mozilla Firefox 47

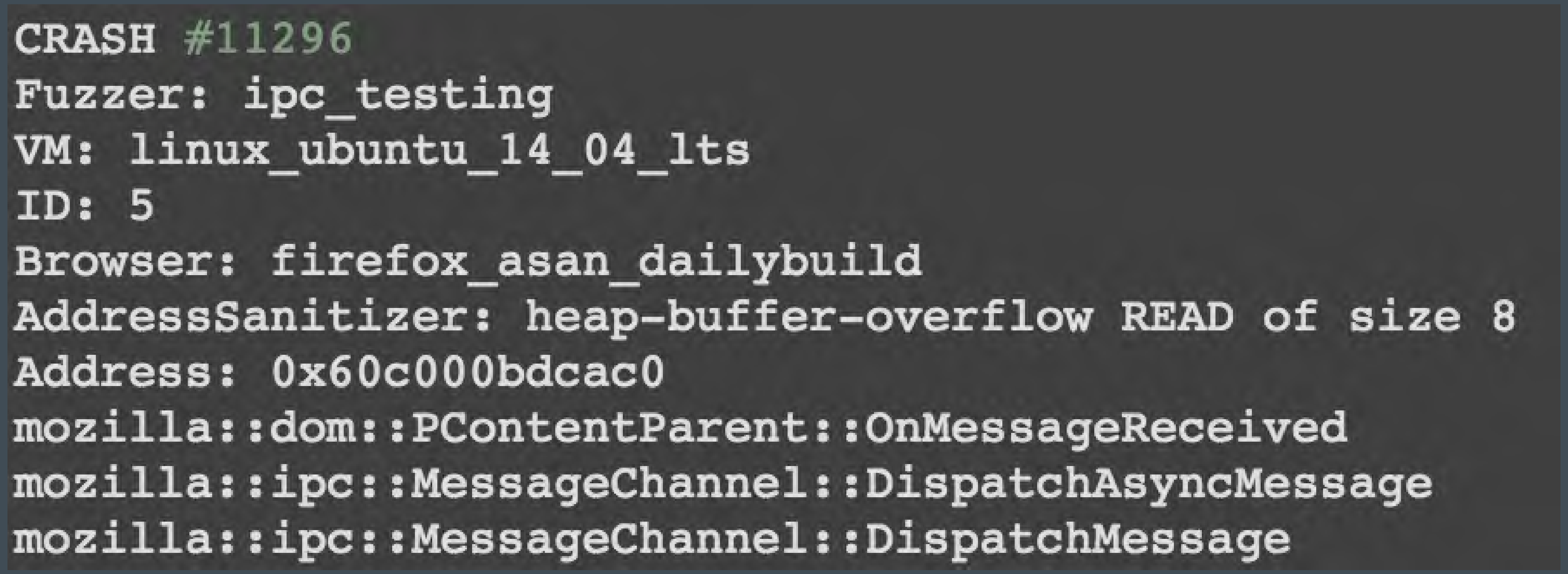


```
==1==ERROR: AddressSanitizer: use-after-poison on address 0x7e852fa6eaf8 at pc 0x5646d5342a9d bp 0x7ffe89a318d0 sp 0x7ffe89a318c8
READ of size 8 at 0x7e852fa6eaf8 thread T0 (chrome)
    #0 0x5646d5342a9c in content::WebURLLoaderImpl::Context::OnReceivedResponse(content::ResourceResponseInfo const&) ./out/Release/../../content/child/we
    #1 0x5646ccc253c6 in content::ResourceDispatcher::OnReceivedResponse(int, content::ResourceResponseHead const&) ./out/Release/../../content/child/reso
    #2 0x5646ccc2f8ea in DispatchToMethodImpl<content::ResourceDispatcher *, void (content::ResourceDispatcher::*)(int, const content::ResourceResponseHea
    #3 0x5646ccc2f8ea in DispatchToMethod<content::ResourceDispatcher *, void (content::ResourceDispatcher::*)(int, const content::ResourceResponseHead &)
    #4 0x5646ccc2f8ea in DispatchToMethod<content::ResourceDispatcher, void (content::ResourceDispatcher::*)(int, const content::ResourceResponseHead &),
    #5 0x5646ccc2f8ea in bool IPC::MessageT<ResourceMsg_ReceivedResponse_Meta, std::__1::tuple<int, content::ResourceResponseHead>,
Address 0x7e852fa6eaf8 is a wild pointer.
SUMMARY: AddressSanitizer: use-after-poison (/home/fuzzer/browsers/chrome_old/chrome+0x19e3aa9c)
Shadow bytes around the buggy address:
  0x0fd125f45d00: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7
  0x0fd125f45d10: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7
  0x0fd125f45d20: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7
  0x0fd125f45d30: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7
  0x0fd125f45d40: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7
=>0x0fd125f45d50: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7[f7]
  0x0fd125f45d60: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7
  0x0fd125f45d70: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7
  0x0fd125f45d80: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7
  0x0fd125f45d90: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7
  0x0fd125f45da0: f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7 f7
Shadow byte legend (one shadow byte represents 8 application bytes):
  Addressable:           00
  Partially addressable: 01 02 03 04 05 06 07
  Heap left redzone:       fa
  Freed heap region:       fd
  Stack left redzone:      f1
  Stack mid redzone:       f2
  Stack right redzone:     f3
  Stack after return:      f5
  Stack use after scope:   f8
  Global redzone:          f9
  Global init order:       f6
  Poisoned by user:        f7
  Container overflow:      fc
  Array cookie:            ac
  Intra object redzone:    bb
  ASan internal:           fe
  Left alloca redzone:     ca
  Right alloca redzone:    cb
```

Triage

Crash Metadata

- No line numbers: refactoring, versions
- Useful info: registers, dissassembly
- Symbolize: llvm-symbolizer
- Signatures
- Blacklist known bugs, group by.
- Impact



```
CRASH #11296
Fuzzer: ipc_testing
VM: linux_ubuntu_14_04_lts
ID: 5
Browser: firefox_asan_dailybuild
AddressSanitizer: heap-buffer-overflow READ of size 8
Address: 0x60c000bdcac0
mozilla::dom::PContentParent::OnMessageReceived
mozilla::ipc::MessageChannel::DispatchAsyncMessage
mozilla::ipc::MessageChannel::DispatchMessage
```

Triage

Minimize & Bisect

- **Delta debugging: trim useless functions, LOC not needed to reproduce the bug.**
- **lang-based: delete statements, functions and sub-expressions. *JSDelta***
- **Line-based: *lithium* (Mozilla)**
- **Algorithm-based: Genetic**
- **Reducers are Fuzzers: large testcases after being minimized some times trigger new bugs.**
- **Bisection: finding the patch or commit that introduced or fix a bug**
- **Specific versions of a library used, last revision**

Conclusions

- **Mobile Lab for testing is required, Mobile provisioning and automate testing (Frida helps a lot)**
  iPhone devices are expensive, but not logic boards, Happy HW Hacking!
- **Focus on Small areas, custom buzzers, custom mutators, custom dict**
- **Be patient**
- **Stay informed (mailing list, commits monitor, Future Q plans)**
- **Bugs are expensive because the work is complex and requires be constant**
- **Race Conditions in Render Process TODO.txt**
- **Concolic Fuzzers TODO.txt**

Q&A