CrowdStrike

Exploiting a Coalmine

Abusing Complex Bugs in Webkit's RenderArena

Georg Wicherski

Senior Security Researcher

Wednesday, April 11, 2012

WebKit

- Based on KHTML (KDE)
  - Apple forked in 2001
- Chrome, (Mobile) Safari, **Android Browser**, Qt, PS 3 Vita, …
  - Rule of Thumb: If it's not Internet Explorer or Firefox, it uses WebKit
- SLOC: 753,572 (Android 2.3.5)
  - If that's not enough: libpng, libtiff, …

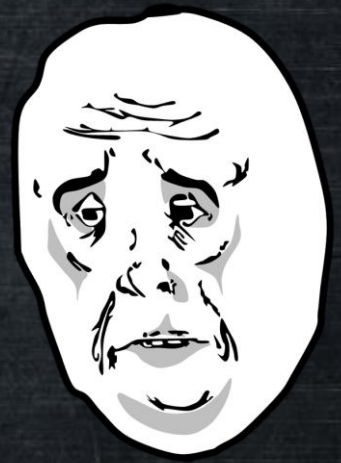
TCMALLOC


RPG RAPTOR SHARK
YOUR ARGUMENT IS INVALID
memegenerator.net

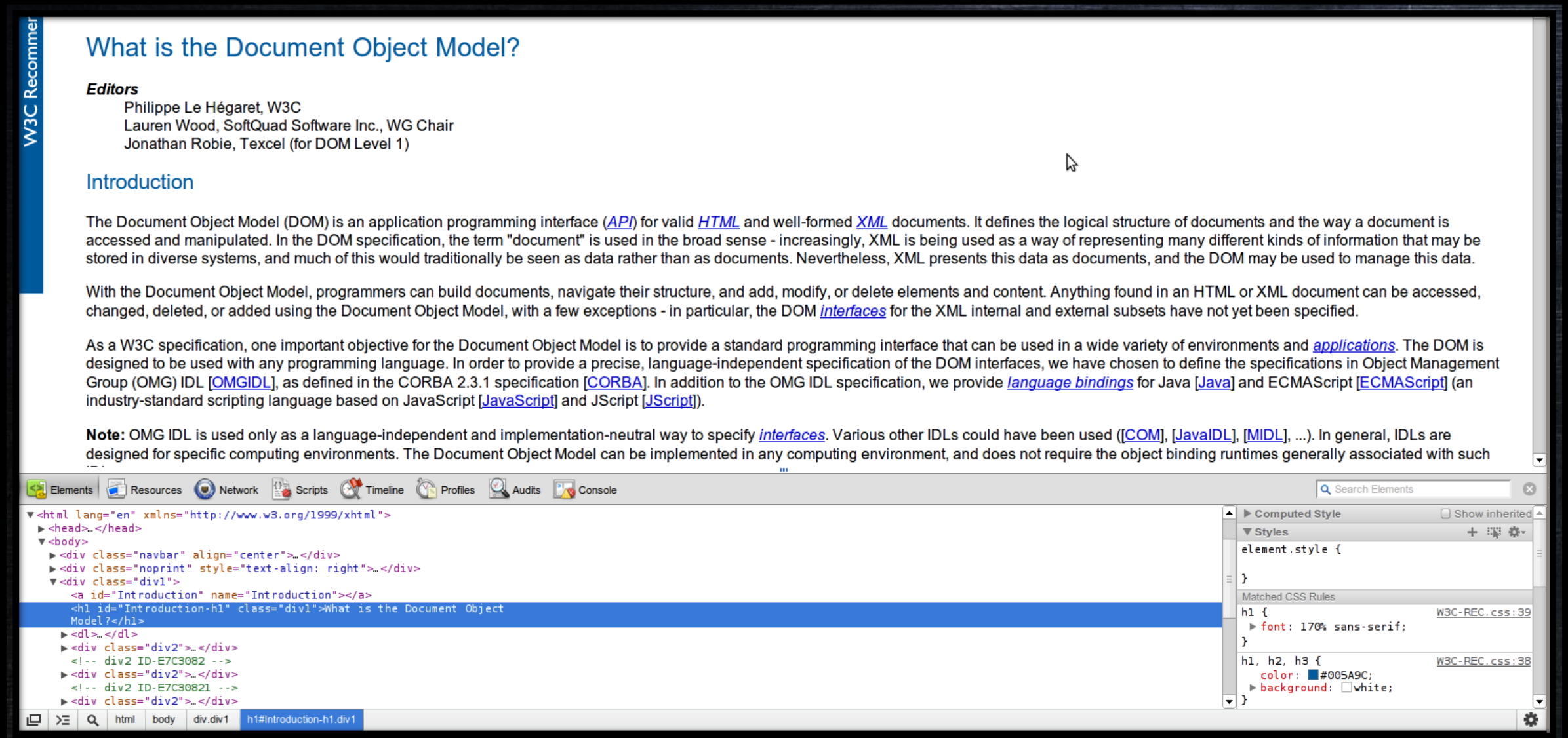
DLmalloc

```
#if PLATFORM(ANDROID)
  #define WEBCORE_NAVIGATOR_VENDOR "Google Inc."
  #define USE_SYSTEM_MALLOC 1
  …
```

- bionic (Android's libc) uses Doug Lea's malloc
  - This is the same allocator glibc uses
  - Without safe unlinking checks
- DLmalloc coalesces adjacent free chunks
  - No per thread caches or free-lists



Document Object Model Tree



Render Tree

“At the heart of rendering is the render tree. The render tree is very similar to the DOM in that it is a tree of objects, where each object can correspond to the document, elements or text nodes. The render tree can also contain additional objects that have no corresponding DOM node.

The base class of all render tree nodes is `RenderObject`.”

http://www.webkit.org/blog/114/webcore-rendering-i-the-basics/

Render Object Creation

- The Render Tree is updated every time rendering changes
  - This includes when objects are repositioned and text flow changes
  - Resizing Window, Scrolling (on Android anyway), …
- A simple DOM Text Node can get additonal associated Render Tree children just be resizing your window

➢ Allocations and deallocations happen very frequently

The RenderArena

"Yo Dawg, I heard you like allocators, so I put an allocator into your allocator, so you can allocate while you're allocating!"

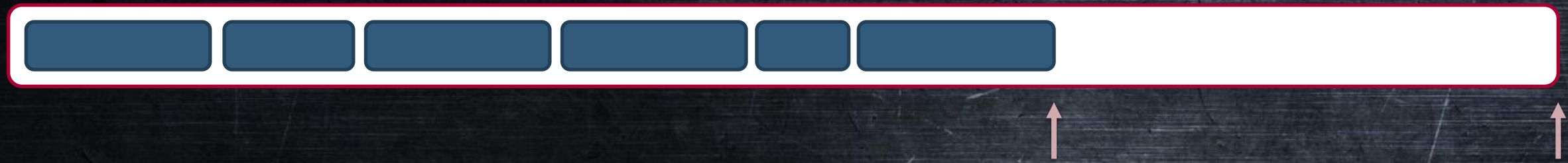
- `RenderArena` is the allocator for `RenderObjects`
- A `RenderArena` consists of multiple Arenas that are allocated with... `fastMalloc`!
  - Recall that `fastMalloc` is an alias for DLmalloc on Android

RenderArena Integration

```
void* RenderObject::operator new(size_t sz, RenderArena* renderArena) throw()
{
    return renderArena->allocate(sz);
}
```

- Every Render Tree element is derived from `RenderObject`
  - `operator new` is inherited by every `Render*`
- All allocations for RenderObjects happen on the Arena sub-heap!
  - Unfortunately, this means also nothing else can be allocated there.

RenderArena Allocation



- Allocation sizes rounded up to 8 bytes
  - Only for alignment, low bits are meaningless
- Attempts to recycle a free chunk of requested size
  - Simple single-linked list, much like FreeList
- Simple forward allocation (current & limit pointers)
  - No chunk headers or other inline information

RenderArena Deallocation

- Free chunks are put into a single-linked list
  - Pointer to next free chunk is first 32bit word in chunk
- There is no coalescing of free chunks!
  - This allows for easier (sub-)heap massaging

Enter the Coalmine

- There is a lot of bugs in the Render Tree
  - And they are mostly considered “just crashes”
  - Fixes are not backported for Android, takes some time until they end up in Chrome mainline (after being public on Webkit Trac)
- Invalid Casts / Type Confusion
  - Pass around RenderObject *, cast to Render* with wrong expectations
- Use-after-free
  - Happens when stuff gets removed due to re-CSS-ing

Use-After-Free Exploitation

"The Wicherski"

- All allocations in the `RenderArena` are by definition C++ objects
  - `RenderObject` has virtual functions, so all allocations have a `vtable`
- `vtable` overlaps with the free chunks single-linked list pointers

a. Free element that resembles fake `vtable`
b. Trigger free of buggy element, so it points to fake `vtable`
c. Trigger Use-After-Free virtual call

Use-After-Free Exploitation

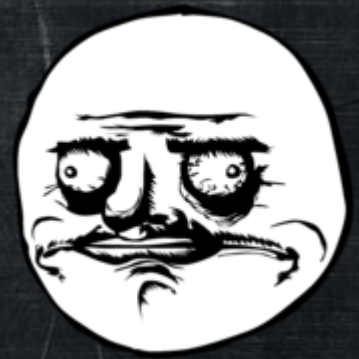
"The Wicherski"

- It is extremely difficult to build a fake `vtable` with RenderObjects
  - RenderObjects are small and contain only pointers and some CSS value copies
- CSS Values by definition have the high nibble cleared
  - The CSS code internally uses high 4 bits for flags
  - Flags are cleared before values arrive in rendering code

Use-After-Free Exploitation

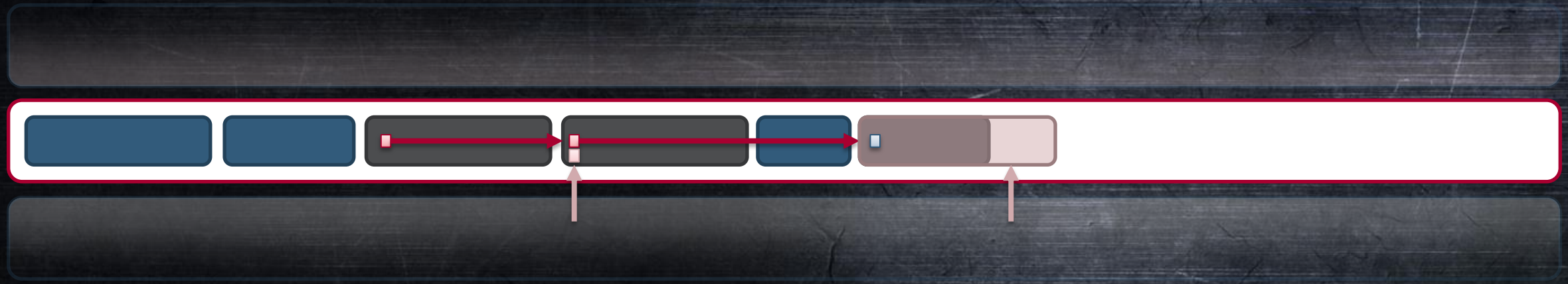
"The Wicherski"

- The heap is rwx on Android ≤ 2.2.1 and is reachable
- There is **one** plain integer that is copied
  - List Item *value* (for setting numbered list item values)
  - For our convenience, assigned to two consecutive members
- Read the code, we can even get a pointer to an arbitrary long list of integers we control…



Use-After-Free Exploitation

"The Refined Aubizziere"



- RenderArena allocations come from the system allocator
  - We can control memory contents if not overwritten

a. Spray Arena sized objects (e.g. strings) and free holes
b. Free small dummy element at end of Arena
c. Trigger Use-After-Free virtual call

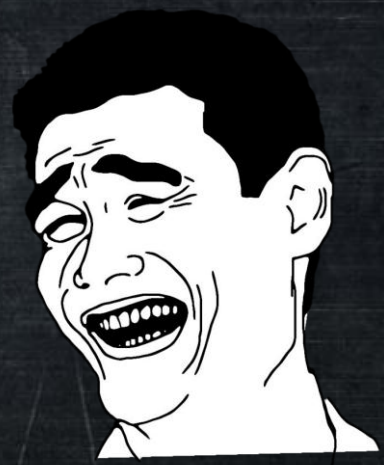
Use-After-Free Exploitation

"The Refined Aubizziere"

\+ Enables arbitrary `vtable` contents

– Requires reliable heap allocation and free primitives
- We cannot "just create strings in JavaScript" because of GC
- `unescape` (praised by Immunity) is only one free
- DLmalloc will happily coalesce chunks, split large chunks, …
- Have fun debugging this with GDB!

➢ We need something better (and faster) than GDB to debug the heap allocations!



CrowdStrike AndroProbe

```
enabled hooking points:
        AFD14520 ( absolute address ), tag BABEFACE, r0
        A8403422 ( absolute address ), tag BABECAFE, r0
        A84033EC ( absolute address ), tag ABADCAFE, r1
        A84033D4 ( absolute address ), tag 500DCAFE, r2
  libc +00014510 (symbol "   malloc"), tag ABAD1DEA, r0
  libc +0001452C (symbol "     free"), tag 500D1DEA, r0
found `svc 0' instruction at AFD0AF78h
code has been allocated at AFCFF000h
assembling trampoline for AFD14520 at AFCFF03C
  encountered pop {.., pc} instruction at copy offset 00
assembling trampoline for A8403422 at AFCFF054
  encountered pop {.., pc} instruction at copy offset 00
assembling trampoline for A84033EC at AFCFF06C
  building LDR link to original code at AFCFF084h
assembling trampoline for A84033D4 at AFCFF090
  building LDR link to original code at AFCFF0AAh
assembling trampoline for AFD14510 at AFCFF0B8
  cloning constant #1 from AFD14524h for load: LDR r3, =0002D00Ch
  cloning constant #1 from AFD14528h for load: LDR r2, =FFFFFF28h
  building B link to original code at AFCFF0CCh
assembling trampoline for AFD1452C at AFCFF0DC
  cloning constant #1 from AFD14540h for load: LDR r3, =0002CFF0h
  cloning constant #1 from AFD14544h for load: LDR r2, =FFFFFF28h
  building B link to original code at AFCFF0F0h
passed buffer location 48B07000h to debugee
everything in place, resuming execution
```


CrowdStrike