Chromium's Out-of-Memory Killer

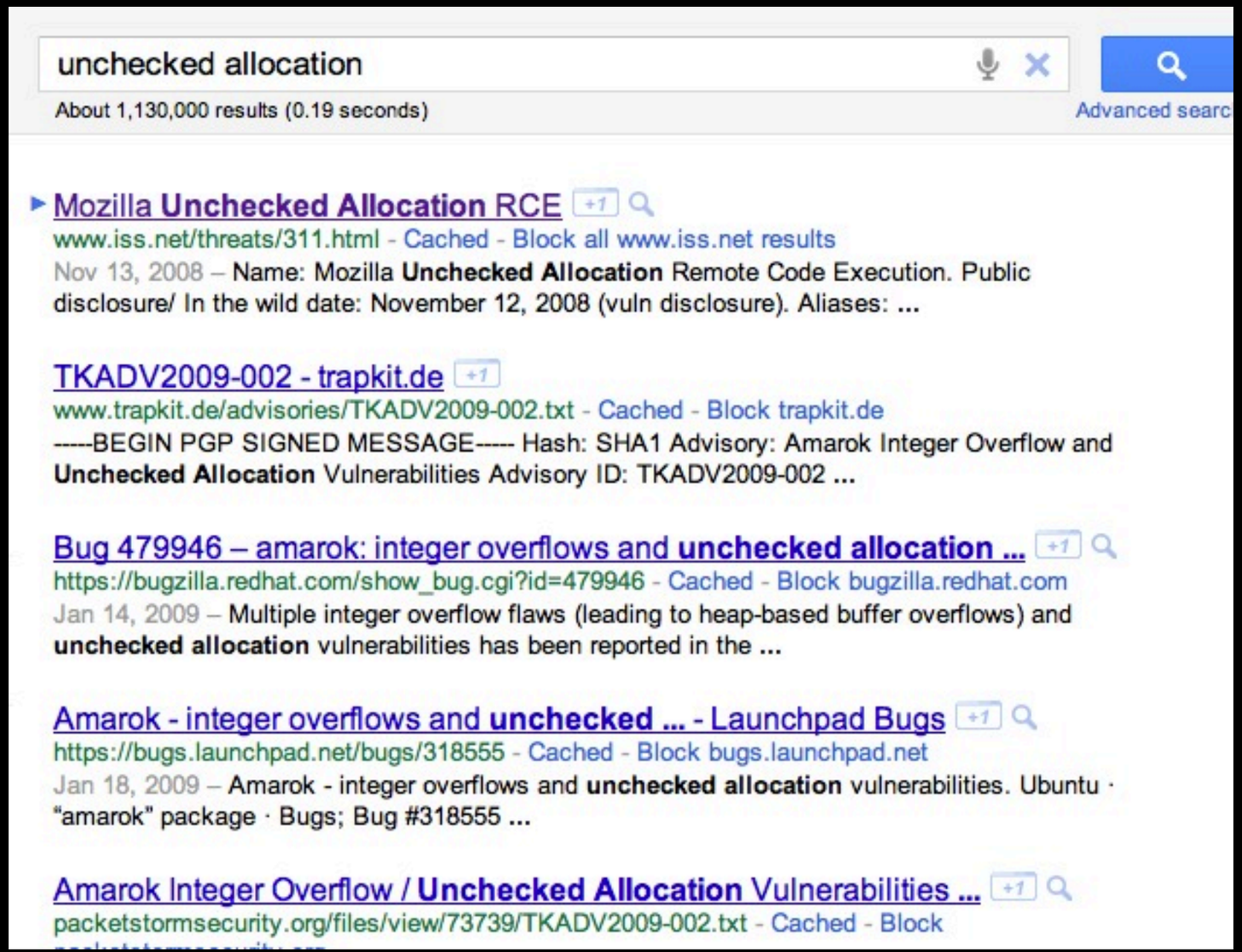
or

Allocators Gonna Allocate

Avi Drissman
CocoaHeads NYC

An unchecked allocation is an unfortunately popular security hole. Untrustworthy data can specify buffer sizes that cannot be allocated, and if the processing code doesn't handle a null return from a memory allocator, that null pointer can be abused for things like remote code execution.

Unchecked Allocations



From Mozilla Firefox,
netwerk\streamconv\converters\nsDirIndexParser.cpp
lines 203-204:

```
mFormat = new int[num+1];
mFormat[num] = -1;
```

From https://bugzilla.mozilla.org/show_bug.cgi?id=443299:

[...] If the fragmentation is successful this will result in a failed allocation of nsDirIndexParser::mFormat and the value 0xFFFFFFFF being written to the nsHTMLTags::sMaxTagNameLength at address 0xB45AD0. [...]

From https://bugzilla.mozilla.org/show_bug.cgi?id=443299:

[...] This stage contains a malformed HTML tag to write beyond the bounds of the static variable buf (address 0xB45CA8) in nsHTMLTags::LookupTag(). The exploit buffer overwrites several function pointers and replaces them with a pointer to the shellcode stub. [...]

Coders should always null-check allocations.

And I want a pony. And some ice cream.
With sprinkles.

malloc() **failure is fatal.**

How many ways can you allocate?

- malloc
  - calloc, valloc, realloc, posix_memalign
- operator new
- +[NSObject alloc], +[NSObject allocWithZone:]
- CFAllocator

How?

- APIs we can use
- APIs we can abuse
- SPIs we can use
- Raw memory mangling

APIs we can use

```
void oom_killer_new() {
  debug::BreakDebugger();
}

void EnableTerminationOnOutOfMemory() {
  // ...
  std::set_new_handler(oom_killer_new);
  // ...
}
```

```
typedef id (*allocWithZone_t)(id, SEL, NSZone*);
allocWithZone_t g_old_allocWithZone;

id oom_killer_allocWithZone(id self, SEL _cmd, NSZone* zone) {
  id result = g_old_allocWithZone(self, _cmd, zone);
  if (!result)
    debug::BreakDebugger();
  return result;
}
```

```
void EnableTerminationOnOutOfMemory() {
  // ...
  Method orig_method =
    class_getClassMethod(
      [NSObject class], @selector(allocWithZone:));
  g_old_allocWithZone =
    reinterpret_cast<allocWithZone_t>(
      method_getImplementation(orig_method));
  method_setImplementation(
    orig_method,
    reinterpret_cast<IMP>(oom_killer_allocWithZone));
  // ...
}
```

APIs we can abuse

#include <malloc/malloc.h>:

```
typedef struct _malloc_zone_t {
  // ...
  void *(*malloc)(struct _malloc_zone_t *zone, size_t size);
  void *(*calloc)(struct _malloc_zone_t *zone, size_t num_items, size_t
size);
  // ...
} malloc_zone_t;

malloc_zone_t *malloc_default_zone(void);
void malloc_zone_register(malloc_zone_t *zone);

void *malloc_zone_malloc(malloc_zone_t *zone, size_t size);
```

```
typedef void* (*malloc_type)
  (struct _malloc_zone_t* zone, size_t size);
malloc_type g_old_malloc;

void* oom_killer_malloc(
    struct _malloc_zone_t* zone, size_t size) {
  void* result = g_old_malloc(zone, size);
  if (!result && size)
    debug::BreakDebugger();
  return result;
}
```

```
malloc_zone_t* default_zone = malloc_default_zone();

g_old_malloc = default_zone->malloc;
g_old_calloc = default_zone->calloc;
g_old_valloc = default_zone->valloc;
g_old_realloc = default_zone->realloc;

default_zone->malloc = oom_killer_malloc;
default_zone->calloc = oom_killer_calloc;
default_zone->valloc = oom_killer_valloc;
default_zone->realloc = oom_killer_realloc;
```

Raw memory mangling

Allocators Gonna Allocate

- void* CFAllocatorAllocate (
  CFAllocatorRef allocator,
  CFIndex size,
  CFOptionFlags hint
  );
- **But what's a** CFAllocator**, really?**

#include <CoreFoundation/CFBase.h>:

```
struct CFAllocatorContext {
    CFIndex version;
    void *info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
    CFAllocatorAllocateCallBack allocate;
    CFAllocatorReallocateCallBack reallocate;
    CFAllocatorDeallocateCallBack deallocate;
    CFAllocatorPreferredSizeCallBack preferredSize;
};
```

```
CFAllocatorAllocateCallBack g_old_cfallocator_system_default;

void* oom_killer_cfallocator_system_default(
    CFIndex alloc_size, CFOptionFlags hint, void* info) {
  void* result = g_old_cfallocator_system_default(alloc_size, hint, info);
  if (!result)
    debug::BreakDebugger();
  return result;
}
```

```
void EnableTerminationOnOutOfMemory() {
  // ...
  CFAllocatorContext context;
  CFAllocatorGetContext(kCFAllocatorSystemDefault, &context);
  g_old_cfallocator_system_default = context.allocate;
  context.allocate = oom_killer_cfallocator_system_default;
  // ...
}
```

```
$ ./base_unittests --gtest_filter=OutOfMemoryDeathTest.CFAllocatorSystemDefault
Note: Google Test filter = OutOfMemoryDeathTest.CFAllocatorSystemDefault
[==========] Running 1 test from 1 test case.
[----------] Global test environment set-up.
[----------] 1 test from OutOfMemoryDeathTest
[ RUN      ] OutOfMemoryDeathTest.CFAllocatorSystemDefault
/Users/avi/Source/chrome-git/src/base/process_util_unittest.cc:920: Failure
Death test: { SetUpInDeathAssert(); while ((value_ = base::AllocateViaCFAllocatorSystemDefault(signed_test_size_)))
{} }
    Result: failed to die.
 Error msg:
[  DEATH   ] base_unittests(2942) malloc: *** mmap(size=2147483648) failed (error code=12)
[  DEATH   ] *** error: can't allocate region
[  DEATH   ] *** set a breakpoint in malloc_error_break to debug
[  DEATH   ]
[  FAILED  ] OutOfMemoryDeathTest.CFAllocatorSystemDefault (3 ms)
[----------] 1 test from OutOfMemoryDeathTest (3 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test case ran. (5 ms total)
[  PASSED  ] 0 tests.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] OutOfMemoryDeathTest.CFAllocatorSystemDefault

 1 FAILED TEST
```

http://www.opensource.apple.com/source/CF/CF-635/CFBase.c:

```
void CFAllocatorGetContext(
    CFAllocatorRef allocator, CFAllocatorContext *context) {
  // ...
  context->allocate = allocator->_context->allocate;
  context->reallocate = allocator->_context->reallocate;
  context->deallocate = allocator->_context->deallocate;
  context->preferredSize = allocator->_context->preferredSize;
  // ...
}
```

#include <CoreFoundation/CFBase.h>:

typedef const struct __CFAllocator * CFAllocatorRef;

http://www.opensource.apple.com/source/CF/CF-635/CFBase.c:

```
struct __CFAllocator {
  // ...
  void *(*malloc)(struct _malloc_zone_t *zone, size_t size);
  void *(*calloc)(struct _malloc_zone_t *zone, size_t num_items, size_t
size);
  // ...
  CFAllocatorRef _allocator;
  CFAllocatorContext _context;
};
```

```
CFAllocatorContext* ContextForCFAllocator(CFAllocatorRef allocator) {
  if (base::mac::IsOSLeopard() || base::mac::IsOSSnowLeopard()) {
    ChromeCFAllocatorLeopards* our_allocator =
        const_cast<ChromeCFAllocatorLeopards*>(
            reinterpret_cast<const ChromeCFAllocatorLeopards*>(allocator));
    return &our_allocator->_context;
  } else if (base::mac::IsOSLion()) {
    ChromeCFAllocatorLion* our_allocator =
        const_cast<ChromeCFAllocatorLion*>(
            reinterpret_cast<const ChromeCFAllocatorLion*>(allocator));
    return &our_allocator->_context;
  } else {
    return NULL;
  }
}
```

```
void EnableTerminationOnOutOfMemory() {
  // ...
  CFAllocatorContext* context =
      ContextForCFAllocator(kCFAllocatorSystemDefault);
  g_old_cfallocator_system_default = context->allocate;
  context->allocate = oom_killer_cfallocator_system_default;
  // ...
}
```

of course, it's not quite that easy...

Subtleties

(break to examine the code)

Q&A