TALOS

PROTECTING YOUR NETWORK

Tyler Bohan - @1blankwall1

SummerCon 2016

In the Zone:
OS X Heap Exploitation

Introduction

- Tyler Bohan
  - Senior Research Engineer
  - Cisco Talos
- Team
  - Richard Johnson
  - Aleksandar Nikolich
  - Ali Rizvi-Santiago
  - Marcin Noga
  - Piotr Bania
  - Yves Younan
  - Cory Duplantis
- Talos Vulndev
  - Third party vulnerability research
    - 170 bug finds in last 12 months
      - Microsoft
      - Apple
      - Oracle
      - Adobe
      - Google
      - IBM, HP, Intel
      - 7zip, libarchive, NTP
  - Security tool development
    - Fuzzers, Crash Triage
  - Mitigation development
    - FreeSentry

Why?

- Discovered a heap overflow and moved forward with exploitation
- Not much recent documentation about the default malloc implementation on OS X
- Needed to ensure that overflow does not cause any unintentional side effects

Why?

- Heap spraying not reliable or elegant
- In depth knowledge of the heap algorithm is important to consistently position chunks relative to one another
- Constantly running into already freed blocks corrupting headers and failing

Roadmap

- Heap Structures - Boring (important)
- Heap Strategies - Super Fun (unique techniques)
- Exploit Strategies - Fun (putting it all together)

Prior Work

- Insomnia
  - https://www.insomniasec.com/downloads/publications/Heaps_About_Heaps.ppt
- Immunity
  - http://www.slideshare.net/seguridadapple/attacking-the-webkit-heap-or-how-to-write-safari-exploits
- Phantasmal Phantasmagoria
- Chris Valasek
  - https://media.blackhat.com/bh-us-12/Briefings/ValasekBH_US_12_Valasek_Windows_8_Heap_Internals_Slides.pdf
- Etc.

Prior Work (OS X)

- OS X Heap Exploitation Techniques - 2005
  - Nemo
- Mac OS Xploitation (and others) - 2009
  - Dino A. Dai Zovi
- Cocoa with Love - How Malloc Works on Mac - 2010
  - Matt Gallagher

TALOS

OS X

- Research done on El Capitan version 10.11.5
  - minor changes easy to implement into tooling

- Scalable Malloc – Documented in OS X Hackers Handbook

- Magazine Malloc - Default allocator in El Capitan

Heap Primer(?)

Heap Implementations

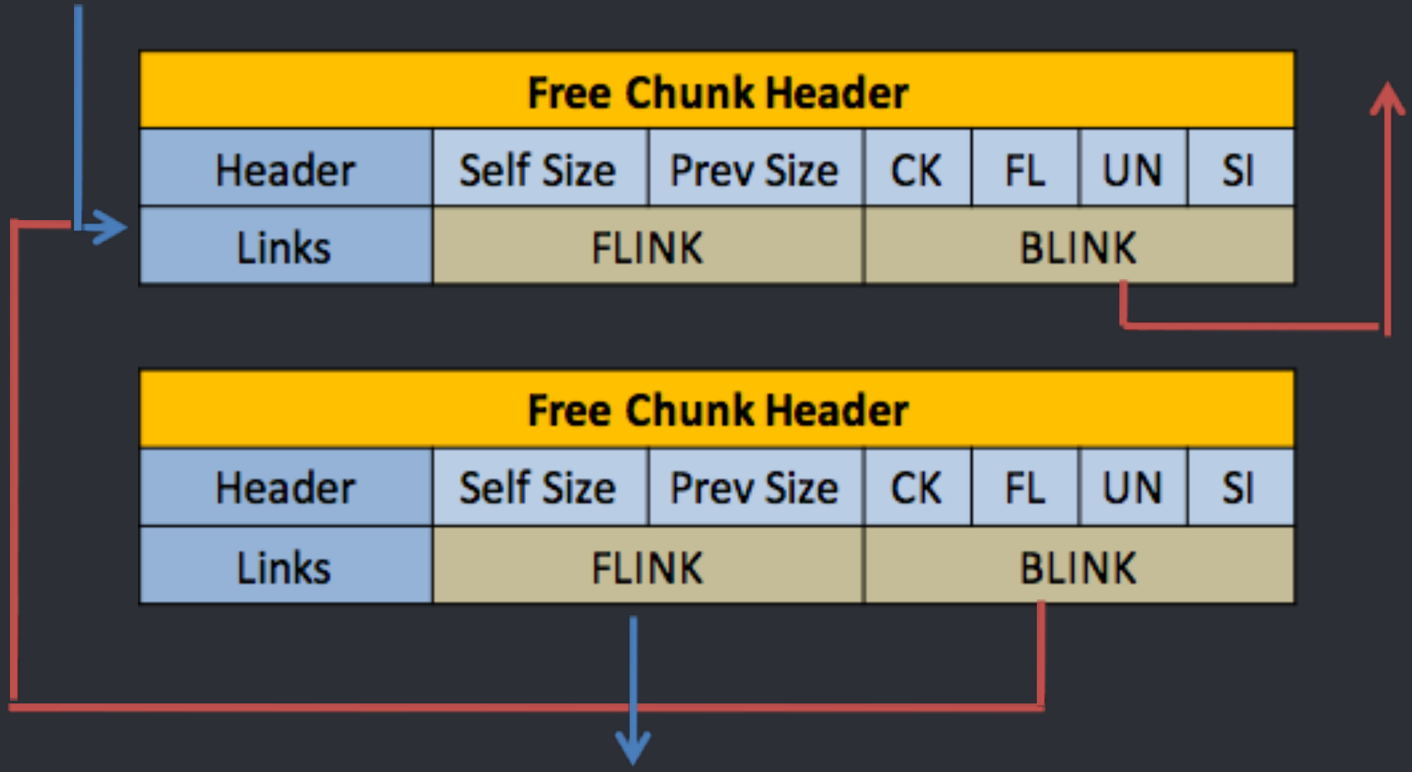
- Magazine allocator – This one
- Scalable allocator – Older OSX allocator
- JEmalloc – Jason Evans (bsd)
- DLmalloc – Doug Lea (glibc)
- PTmalloc – Wolfram Gloger (newer-glibc)
- TCmalloc – Webkit’s General allocator (google)
- Lookaside List -- Windows (earlier)
- LF Heap -- Windows (recent)
- Etc.

Heap Implementations

- DL Malloc, TC Malloc, JE Malloc, etc…
  - Used to cache or sort chunks that were recently freed
  - Generally a hybrid between a linked-list/metadata and an arena
  - Chunk-based allocators prefix a chunk with metadata headers that allow an allocator to divide or navigate through the different chunks that are available.

Heap Implementation Schemes

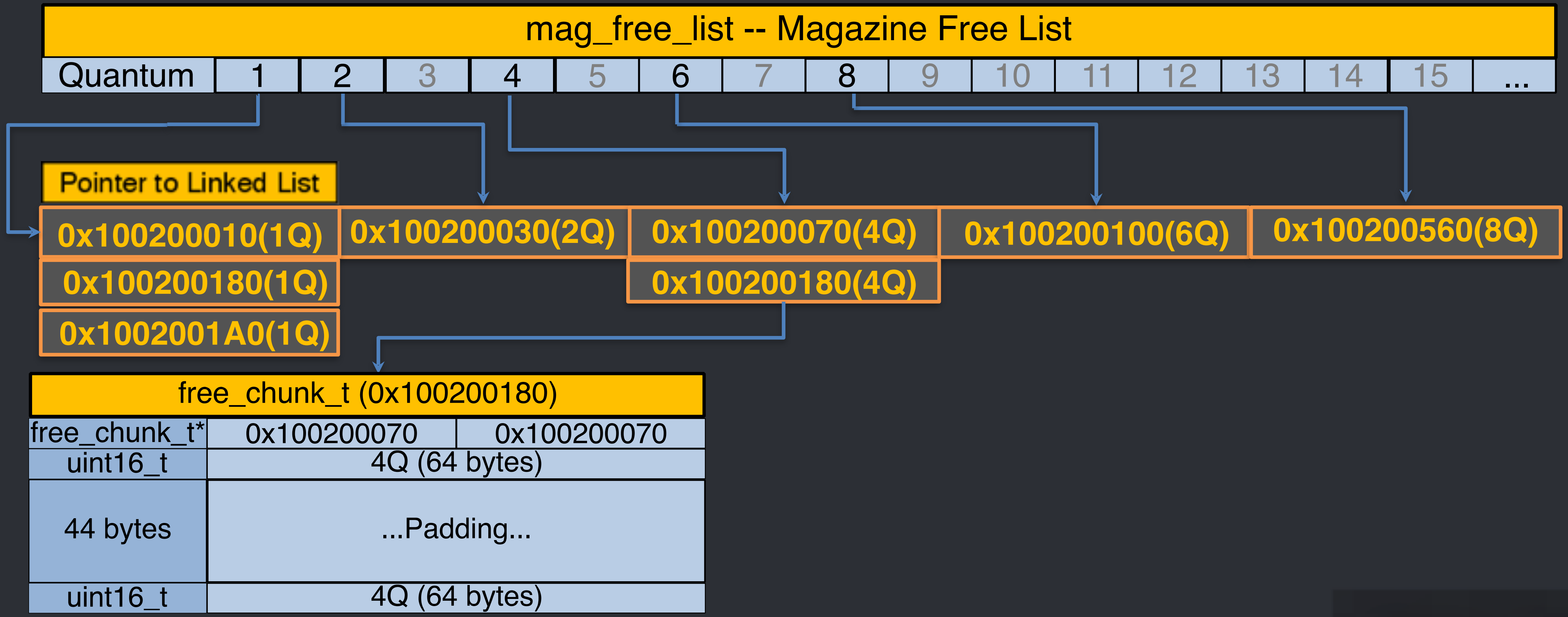
Linked List Allocator:



Heap Implementations

- Magazine Allocator, LFH, BSD malloc etc…
    - Arena-based allocators tend to group allocations of the same size on each page
    - Allocated objects efficiently deallocated

Heap Implementation Schemes



OS X Divergence

OS X – CoreFoundation

- Contains various higher-level types for passing arguments to lower-level api.

- These include things such as CFDictionary, CFSet, CFString, etc.

- Referenced and garbage collected pointers: CFRetain/CFRelease

OS X – CoreFoundation

- Allocates things such as hash table using the system malloc.
- Initialization code allocates onto default heap for setup changing heap state — more on this later
- Heap objects available for overwrite coming from CoreFoundation
- Used by WebKit and Safari for various allocations

OS X – Magazine Allocator

- Provided heap utilities

- Guard Malloc
  - Page heap like

libgmalloc is used in place of the standard system malloc, and uses the virtual memory system to identify memory access bugs. Each malloc allocation is placed on its own virtual memory page (or pages). By default, the returned address for the allocation is positioned such that the end of the allocated buffer is at the end of the last page, and the next page after that is kept unallocated. Thus, accesses beyond the end of the buffer cause a bad access error immediately. When memory is freed, libgmalloc deallocates its virtual memory, so reads or writes to the freed buffer cause a bad access error. Bugs which had been difficult to isolate become immediately obvious, and you'll know exactly which code is causing the problem.

TALOS

OS X – Magazine Allocator

- Malloc Stack Logging:
  - Requires debug flags - MallocStackLogging
  - Alters heap layout

  - malloc_info in LLDB
    - command script import lldb.macosx.heap
    - malloc_info -s 0x1080715e0

```
(lldb) malloc_info -s 0x00000001080715e0
0x00000001080715e0: malloc(   160) -> 0x1080715e0
stack[0]: addr = 0x1080715e0, type=malloc, frames:
```

OS X – Magazine Allocator

- Malloc Stack Logging:
  - malloc_history from the terminal
    - more detailed output
    - all stack frames for malloc and free throughout program history
    - not just current allocation stack frame

OS X – Magazine Allocator

- Malloc Stack Logging:

```
➜ ~ malloc_history 48863 0x00000001080715e0
malloc_history Report Version:  2.0
ALLOC 0x108071550-0x10807160f [size=192]: thread_7fff7a2a5000 |start | NSApplicationMain | -[NSApplication run] | -[NSApplication _nextEventMatchingEventMask:untilDate:inMode:dequeue:] | _DPSNextEvent | _BlockUntilNextEventMatchingListInModeWithFilter | ReceiveNextEventCommon | RunCurrentEventLoopInMode | CFRunLoopRunSpecific | __CFRunLoopRun | __CFRUNLOOP_IS_SERVICING_THE_MAIN_DISPATCH_QUEUE__ | _dispatch_main_queue_callback_4CF | _dispatch_client_callout | _dispatch_call_block_and_release | 0x1000f4dfe | 0x100009f13 | 0x10000b0df | -[NSObject(NSThreadPerformAdditions) performSelectorOnMainThread:withObject:waitUntilDone:] | -[NSObject(NSThreadPerformAdditions) performSelector:onThread:withObject:waitUntilDone:modes:] | 0x10000b19e | -[NSWindowController showWindow:] | 0x100009d9a | -[NSWindowController window] | -[NSWindowController _windowDidLoad] | 0x10000bb47 | 0x100014e5d | 0x10001b162 | -[IKImageContentView setImageURL:imageAtIndex:withDisplayProperties:] | __69-[IKImageContentView setImageURL:imageAtIndex:withDisplayProperties:]_block_invoke | -[IKImageContentView _newCGImageFromImgSrc:index:displayProperties:imageScale:createBitmapImmediately:] | CGImageSourceCreateThumbnailAtIndex | CGImageCreateCopyWithParametersNew | CGContextDrawImage | CGContextDrawImageWithOptions | ripc_DrawImage | ripc_AcquireImage | CGSImageDataLock | img_data_lock | CGColorTransformConvertNeedsCMS | CGColorTransformCacheGetConversionType | CGCMSConverterCreate | ColorSyncTransformCreate | ColorSyncCMMInitializeTransform | AppleCMMInitializeTransform | DoInitializeTransform | ConversionManager::MakeConversionSequence(CMMProfileInfoContainer*, CMMColorConversionInfo*) | ConversionManager::AddLabToXYZ(icXYZNumber const&) | CMMMatrix::MakeMatrixConv(CMMMemMgr&, CMMConvNode*) | CMMBase::NewInternal(unsigned long, CMMMemMgr&, char const*) | CMMMemMgr::New(unsigned long) | calloc | malloc_zone_calloc
----
FREE  0x108071550-0x10807160f [size=192]: thread_7fff7a2a5000 |start | NSApplicationMain | -[NSApplication run] | -[NSApplication _nextEventMatchingEventMask:untilDate:inMode:dequeue:] | _DPSNextEvent | _BlockUntilNextEventMatchingListInModeWithFilter | ReceiveNextEventCommon | RunCurrentEventLoopInMode | CFRunLoopRunSpecific | __CFRunLoopRun | __CFRUNLOOP_IS_SERVICING_THE_MAIN_DISPATCH_QUEUE__ | _dispatch_main_queue_callback_4CF | _dispatch_client_callout | _dispatch_call_block_and_release | 0x1000f4dfe | 0x100009f13 | 0x10000b0df | -[NSObject(NSThreadPerformAdditions) performSelectorOnMainThread:withObject:waitUntilDone:] | -[NSObject(NSThreadPerformAdditions) performSelector:onThread:withObject:waitUntilDone:modes:] | 0x10000b19e | -[NSWindowController showWindow:] | 0x100009d9a | -[NSWindowController window] | -[NSWindowController _windowDidLoad] | 0x10000bb47 | 0x100014e5d | 0x10001b162 | -[IKImageContentView setImageURL:imageAtIndex:withDisplayProperti
```

OS X Heap

OS X – Allocators

- Malloc Zones
- Manages multiple heap implementations used by an application
- Each zone contains a version and name for identification
- Scalable Allocator (version=5)
- Magazine Allocator (version=8)
- Mapped/Released via mmap/munmap.

OS X – Magazine Allocator

- Used by:
  - Anything calling the default CRT malloc
  - libsystem_malloc.dylib, CoreFoundation, etc.
  - Reachable through components such as Preview, Color Sync, Safari, Keynote, etc…

- Not Used by:
  - Webkit (minus lower-level things like Objective-c, libxpc, renderers)
  - ImageIO copies implementation internally

OS X – Magazine Allocator

- Free list compared to a bucket list on other allocators
- Contains a cookie/checksum for pointers

```
// Because the free list entries are previously freed objects, a misbehaved
// program may write to a pointer after it has called free() on that pointer,
// either by dereference or buffer overflow from an adjacent pointer. This write
// would then corrupt the free list's previous and next pointers, leading to a
// crash.  In order to detect this case, we take advantage of the fact that
// malloc'd pointers are known to be at least 16 byte aligned, and thus have
// at least 4 trailing zero bits.
//
// When an entry is added to the free list, a checksum of the previous and next
// pointers is calculated and written to the high four bits of the respective
// pointers.  Upon detection of an invalid checksum, an error is logged and NULL
```

OS X – Magazine Allocator

- Each region/arena is bound to a core or “magazine”
- One Magazine per core. One special magazine for free’d regions known as the “depot”.
- Allocation sizes are divided into 3 types (tiny, small, large)

OS X – Magazine Allocator

- Tiny and Small allocations are managed by a unit of measurement known as a “Quantum” (Q)
- Each magazine has a single-allocation cache for short-lived memory allocations

    i.e.

```
NSNumber* myNumber = [NSNumber alloc]
```

OS X – Magazine Allocator

- Tiny allocations (Q of 16) – maximum size of 1008
- Small allocations (Q of 512) - minimum size of 1009
- Large allocations (size defined in zone and is defined based on “hw.memsize” sysctl)

OS X – Magazine Allocator

- All allocations come from an arena known as a region
- Free-lists are linked-lists indexed by number of Q
- When a chunk is moved into free-list it is coalesced
- When a region in the depot is empty, region is relinquished back to the OS

Magazine Allocator – Tiny Allocations

- Chunks are handed out by a tiny region.
  - Allocations are from 1Q (16 bytes) to 63Q (1008 bytes)
  - Region is always 0x100000 bytes (0x100 pages) in size
  - Contains 64520 Q-sized blocks for allocations
  - Tiny metadata is described with two bitmaps:
    One describing which block starts a chunk
    Another describing whether the chunk is busy/free

Magazine Allocator – Small Allocations

- Chunks handed out from a small region (Q = 512b)
  - Allocations are from 1Q (1008 / 512b) to szone.large_threshold
  - Region is always 0x800000 bytes (0x800 pages) in size
  - Contains 16320 Q-sized blocks
  - Metadata is a simple list of uint16_t each representing a block and the number of Q to get to the next one
  - The high-bit of each uint16_t describes free/busy

Magazine Allocator – Large Allocations

- Anything larger than szone.large_threshold.
    * Determined by sysctl hw.memsize being > 2^{30}
- Managed by a circular linked list + cache
- Cached until a certain fragmentation threshold is reached.
- When fragmentation threshold is reached, returned back to OS.
- Not elaborated on in this presentation - not practical for exploitation purposes

OS X – Magazine Allocator

- Thread Migration
  - Threads migrate between cores when user space yields to kernel
  - Kernel can wake up thread on a different core!

OS X – Magazine Allocator

- Thread Migration
  - Core determines magazine
    - magazine determines heap layout
  - Precise heap manipulation difficult or impossible if constantly migrating cores

OS X – Magazine Allocator

- Thread Migration
  - Patent pending information here

TALOS

OS X – Magazine Allocator

- Thread Migration – How to deal?
  - Try to prevent core switching via occupying threads
    - i.e. tight loops etc…

OS X – Magazine Allocator

- Thread Migration – How to deal?
- 5 iterations available vs blocking
    - available = ~460,000 iterations
    - blocking = ~860,000 iterations

Magazine Allocator Structures

Magazine Allocator

- Tied together by various structures to manage the three different allocation types
- Attempts to keep the regions associated with a magazine tied to it’s respective core for any allocations

- Keeps track of an arena and region emptiness for release back to OS
- Tracks large allocations and fragmentation due to reuse of cached large allocations

Talos

Magazine Allocator

- Each magazine maintains a single-allocation cache that is used for very short-lived allocations
- Each magazine contains a free-list indexed by Q for quickly identifying free chunks in a region
- Free chunks contain metadata to aid with forwards or backwards coalescing
- Regions contain metadata used to describe how much of the region is in use, and how it’s in use

Malloc Zones

Magazine Allocator – malloc_zones

- Zones:
  - Version types
    - built in support for multiple different mallocs in one container
  - Major feature is extensibility
    - can handle many allocation schemes
    - makes platforms like Safari possible

Magazine Allocator – malloc_zone_t

```
struct malloc_zone_t {
…
    funcptr_t* malloc;
    funcptr_t* calloc;
…
    const char* zone_name;
…
    struct malloc_introspection_t* introspect;
    unsigned version;
…
}
```

```
struct malloc_introspection_t {
…
 funcptr_t* check;
 funcptr_t* print;
 funcptr_t* log;
…
 funcptr_t* statistics;
…
}
```

Magazine Allocator – malloc_zone_t

- General structure used to identify each malloc
- Zone_Name – pointer to a string with an identifier
- Version – uint32_t identifying allocator type
- Contains function pointers for the different
  - entry points: malloc, free, calloc, etc
  - introspection: size, memalign, pressure_relief, etc
- Szone_t attacked by Nemo in Phrack article

Magazine Allocator – malloc_zone_t

- Phrack digression
  - Szone_t overwrite - not practical would need new szone creation
  - Wrap around bug - squashed with move to 64 bit systems
  - Double-Free - now checked against

Magazine Allocator – szone_t

```
struct szone_t {
    malloc_zone_t basic_zone;
…
    unsigned debug_flags;
    …tiny allocations…
    …small allocations…
    unsigned num_small_slots;
    …large allocations…
    unsigned is_largememem;
    unsigned large_threshold;
    uintptr_t cookie;
…
}
```

- pointer-sized and is xor'd against pointers that are intended to be obfuscated

TALOS

Magazine Allocator – szone_t

- Encoded pointers contain a 4-bit checksum within the top 4-bits of the pointer.

- Obfuscated ptr is converted into a checksum ptr via:
  csum := pointer ^ cookie

- Checksum extracted from pointer via: rol(csum,4) & 0xF

Magazine Allocator – szone_t

```
static INLINE uintptr_t
free_list_checksum_ptr(szone_t *szone, void *ptr)
{
    uintptr_t p = (uintptr_t)ptr;
    return (p >> NYBBLE) |
        (free_list_gen_checksum(p ^ szone->cookie))
        << ANTI_NYBBLE); // compiles to rotate instruction
}
```

0x100103310 ^ 0x233688 = 0x100330598

Checksum(0x100330598) = 1

ror(0x100103310 ,4) | Checksum(…) = 10000000010010331

TALOS

Magazine Allocator – tiny szone_t

```
struct szone_t {
...
    size_t num_tiny_regions;
...
    region_hash_generation_t* tiny_region_generation;
    region_hash_generation_t[2] trg;

    int num_tiny_magazines;
...
    magazine_t* tiny_magazines;
...
    unsigned num_small_slots;
...
    region_t[64] initial_tiny_regions;
...
}
```

```
struct region_hash_generation_t {
   size_t num_regions;
   size_t num_regions_shift;
   region_t*[num_regions] hash_regions;
   region_hash_generation_t* nextgen;
}
```

Talos

Magazine Allocator – small szone_t

```
struct szone_t {
...
    size_t num_small_regions;
...
    region_hash_generation_t* small_region_generation;
    region_hash_generation_t[2] srg;

    int num_small_magazines;
...
    magazine_t* small_magazines;
…
    unsigned large_threshold;
…
    region_t[64] initial_small_regions;
…
}
```

```
struct region_hash_generation_t {
 size_t num_regions;
 size_t num_regions_shift;
 region_t*[num_regions] hash_regions;
 region_hash_generation_t* nextgen;
}
```

Magazine Allocator – large szone_t

```
struct szone_t {
...
    unsigned num_large_objects_in_use;
    unsigned num_large_entries;
    large_entry_t* large_entries;
    size_t num_bytes_in_large_objects;
...
    int large_entry_cache_oldest, large_entry_cache_newest;
    large_entry_t[16] large_entry_cache;
...
    unsigned large_threshold;
...
}
```

Magazine_T

Magazine Allocator – magazine_t

- One magazine allocated per core. +1 for the “depot”.
  - depot used as place holder magazine
  - indexed at slot -1
  - never gets used, only for caching freed regions
- Used to bind regions locally to a core
  - Regions may migrate between magazines only after being relinquished to the depot

Magazine Allocator – magazine_t

- Contains a chunk cache, a free list with bitmap, last region used for an allocation, linked list of all regions
- Monitors some usage stats which are used to manage allocations from the current region

Magazine Allocator – magazine_t

```
struct magazine_t {
...
    void* mag_last_free;
    region_t mag_last_free_rgn;
...
    free_list_t*[256] mag_free_list;
    unsigned[8] mag_bitmap;
...
    size_t mag_bytes_free_at_end, mag_bytes_free_at_start;
    region_t mag_last_region;
...
    unsigned mag_num_objects;
    size_t mag_num_bytes_in_objects, num_bytes_in_magazine;
...
    unsigned recirculation_entries;
    region_trailer_t* firstNode, *lastNode;
...
}
```

Local magazine cache

111000000001100111000110000111XXX

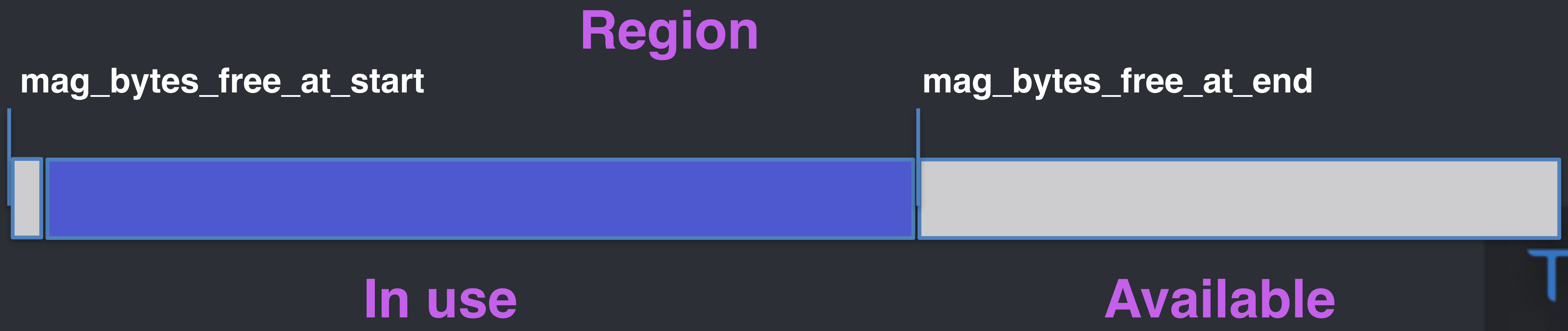
Linked list of all regions

TALOS

Magazine Allocator – magazine_t

- Statistics
  - Mag_bytes_free_at_Start

  - Mag_num_objects
    - number of contiguous slots available - free and busy



Magazine Allocator – magazine_t

- mag_last_free cache – Short-lived allocations
    - Quantum size encoded along with pointer.
    - Tiny Q=16, lower **4**-bits represent quantum size
    - Small Q=512, lower **9**-bits represent quantum size
    - Rest of bits contain pointer with chunk.

tiny 0x100101436

Pointer: 0x100101430

Quantum: 6Q = 96 bytes

small 0x100f09ee1f

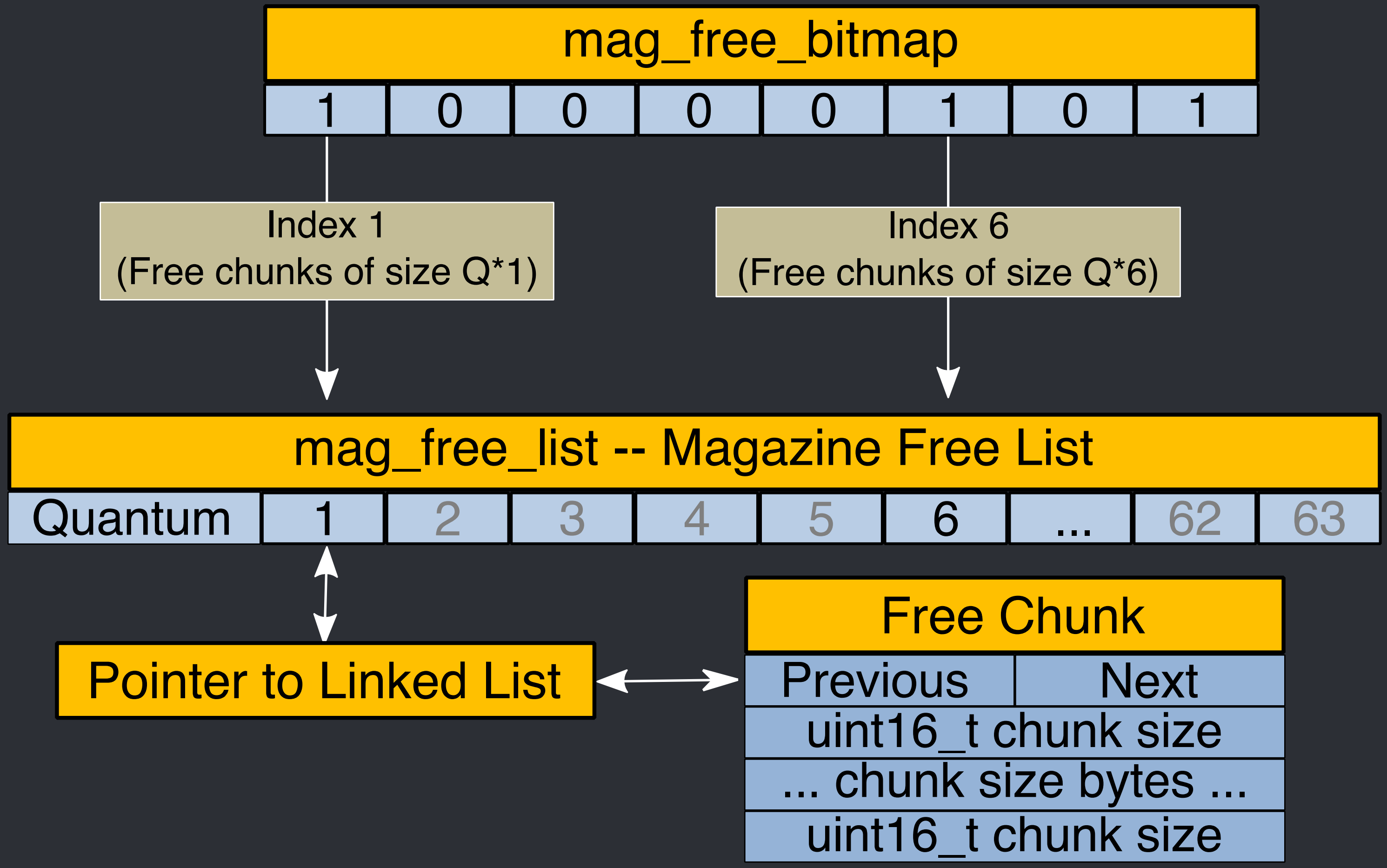
Pointer: 0x100f09ee00

Quantum: 31Q = 15872 bytes

Magazine Allocator – magazine_t

Mag_free_bitmap
Represents which free-list entries are populated

Mag_free_list
Pointers to a circular linked list of a free chunk.



TALOS

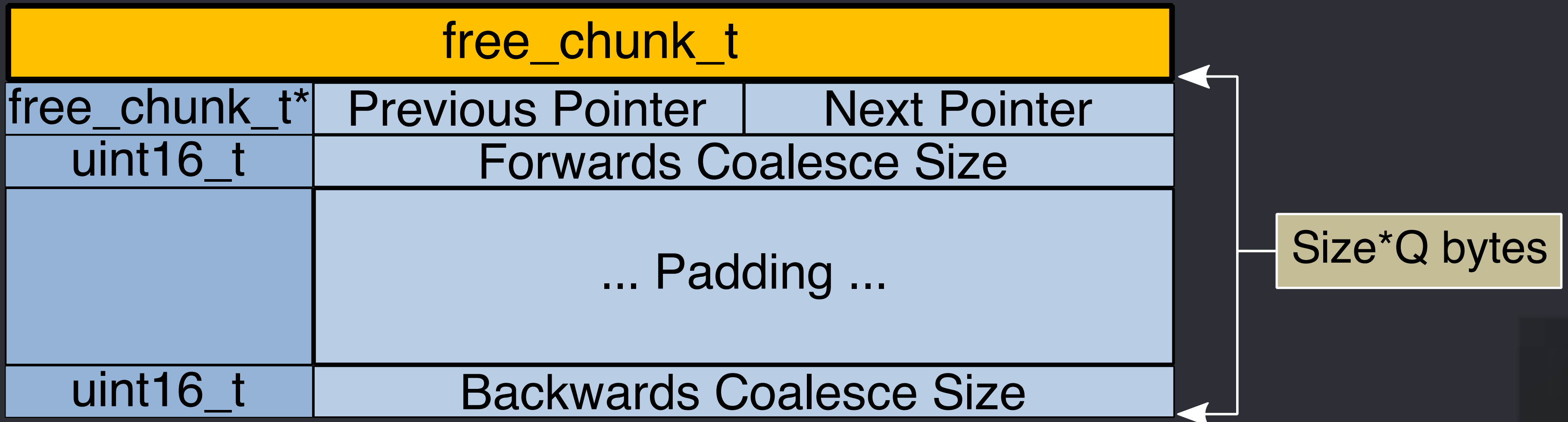
Magazine Allocator – magazine_t

- Magazine Free-list
  - Points to a circular-linked list of all free-chunks.
  - Next and Free pointers are checksummed.
  - Checksum is encoded in the top 4-bits of pointers.
  - Pointer is bit-rotated to the right by 4-bits, and then Or'd with the 4-bit checksum.
  - Checksum not checked if chunk is being coalesced with.

0x30000000001002032b ⟷ 0x1002032b0

Magazine Allocator – magazine_t

- When a chunk moves from cache to free-list
  - Circular linked list written to chunk.
  - The size (in Q) is written at the end of chunk if chunk is > 1Q.
  - Used for forwards and backwards coalescing



Magazine Allocator – tiny magazine_t

- Free list contains 256 slots only 64 in use.
  - Defined by szone.num_small_slots
- Leads to some dead-space due to 1008 maximum.
  - NUM_SMALL_SLOTS = 64

```
[10010e228] <instance macheap._mag_bitmap 'mag_bitmap'> XX.X.X.X.X..X..X.X.X.X.X.X.X.X.X.X.X.X.X.X.X.X.X.X.X.X.X.X.X.X
.X....................................................................................................................
............................................................................
```

Magazine Allocator – small magazine_t

- Free-list uses all 256 slots available for it
- based on large threshold
- typically 0x1fc00 / 512 == 254

```
[100125228] <instance macheap._mag_bitmap 'mag_bitmap'> X..X...X...X...X...X...X...X...X...X...X...X...X...X...X...X..
.X...X..X...X...X...X...X...X...X...X...X...X...X...X...X...X...X...X..X...X...X...X...X...X...X...X...X...X...X..
.X...X...X...X...X...X..X...X...X...X...X...X...X...X...X...X...X...X.......
```

Region_T

Magazine Allocator – region_t

- Two different types for “tiny” and “small” allocations.

- Chunks are composed of a number of chunks split on Q: Tiny = 16 bytes, Small = 512 bytes.

- Contains a number of Q-sized blocks that is calculated based on the size of the region + metadata + region trailer

Magazine Allocator – region_t

- Metadata contains information describing chunk sizes and whether chunk is busy or free.
- Tiny metadata maps each bit in header/in-use to the region's blocks
- Small metadata maps each uint16_t directly to a block

Magazine Allocator – region_t

```
typedef uint8_t[Q] Quantum_t;

struct region_t {
  Quantum_t[NUM_BLOCKS] blocks;
  region_trailer_t trailer;
  metadata_t metadata;
}
```

```
struct small_metadata_t {
    uint16_t[NUM_BLOCKS] msize;
}

struct tiny_header_inuse_pair_t {
    uint32_t header, inuse;
}
```

```
struct tiny_metadata_t {
 tiny_header_inuse_pair_t[NUM_BLOCKS / sizeof(uint32_t)] blocks;
}
```

* Metadata correlates directly with blocks in region_t

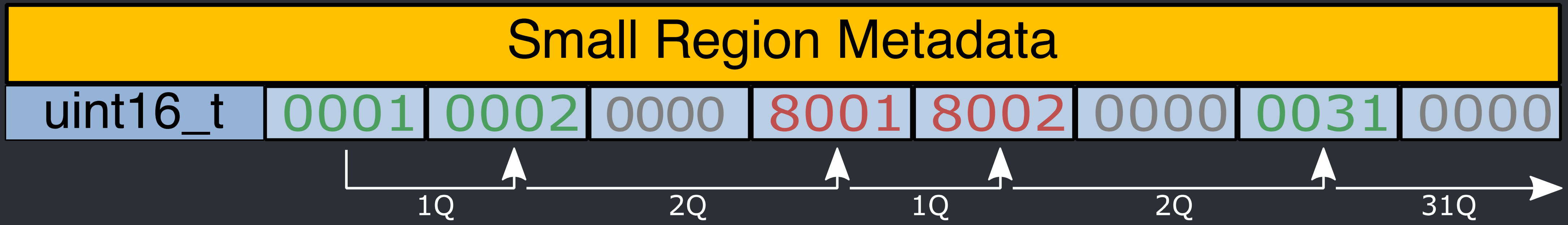
Magazine Allocator – tiny region_t

- Tiny Metadata – Array of structures with two uint32_t
  1. Bit represents whether block is beginning of a chunk. (Used to calculate chunk sizes in Q).
  2. Bit represents whether that entire chunk is busy(1) or free(0)

Tiny Region Metadata	
Header	10010000 10111010 10001011 00001100
In-use	10010000 10110010 00001010 00001100

Magazine Allocator – small region_t

- Small Metadata – Array of encoded uint16_t
  - High-bit represents whether busy(0) or free(1)
  - Rest of bits describe the number of Q for chunk
  - Q represents how many elements to skip to get to next chunk.



Magazine Allocator – large regions

- Large Region
- Simple. Just an address and it’s size.
- doesn't get unmapped until heavily fragmented
- stored in cache

```
typedef struct large_entry_t {
        vm_address_t address;
        vm_size_t size;
        boolean_t did_madvise_reusable;
}
```

Talos

Magazine Allocator – region_trailer_t

- Each region knows which magazine it's associated with
  - To navigate, a magazine_t points to the trailer, allows a magazine to iterate through all regions

```
struct region_trailer_t {
    struct region_trailer* prev, *next;
    boolean_t recirc_suitable;
    int pinned_to_depot;
    unsigned bytes_used;
    mag_index_t mag_index;
}
```

Linked list of all magazines in a region

depot magazine?

Magazine whom retains ownership

Questions?

Strategies

Strategies – for cache

- Tool we call Mac Heap
- LLDB python script allowing access to all this information
- Open Source 🤑

Strategies

- Positioning
  - Strategic block placement, cache management
- Overwriting
  - Block placement before or after free and busy chunks
- Metadata
  - Unlinking attacks

Strategies – for cache

- Cache
  - Emptying cache
    - Malloc exact size as cached object

  - Cache to free list
    - free another object less than 256b(tiny) in size
    - moves new object to cache and cache object to free list

Strategies – for free list

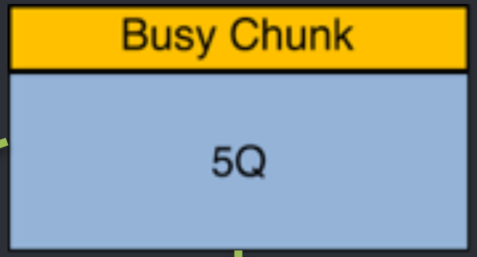
- Free-List
  - Linked List
  - Indexed by Quantum Size
  - Tiny - 64 slots — Small - 256 slots
  - Small minimum size 1009 cant use 1 Q in size due to inability to allocate
  - Q = 512 - 512 goes into tiny region 1 Q allocations are dead

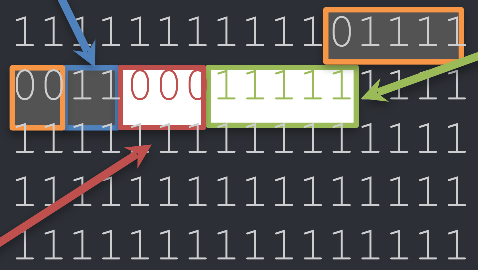
Strategies – mag_free_list

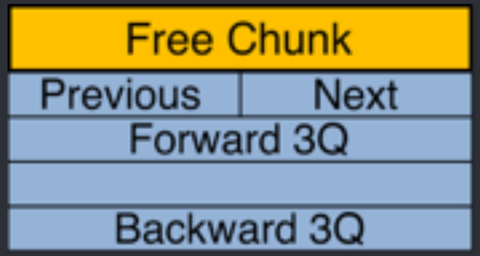
- Coalescing – When chunk is moved to free-list
  - ForwardQ/BackwardQ – Only written to chunks >1Q
  - If overwriting ForwardQ or BackwardQ, need to ensure that specified count is pointing to a free chunk.
  - This will add entire chunk (including any used chunks in between) to free list.

Strategies – mag_free_list – Coalesce

Overwrite chunk with 2Q + 3Q bytes data.
Set BackwardQ to **12** (3+2+7).







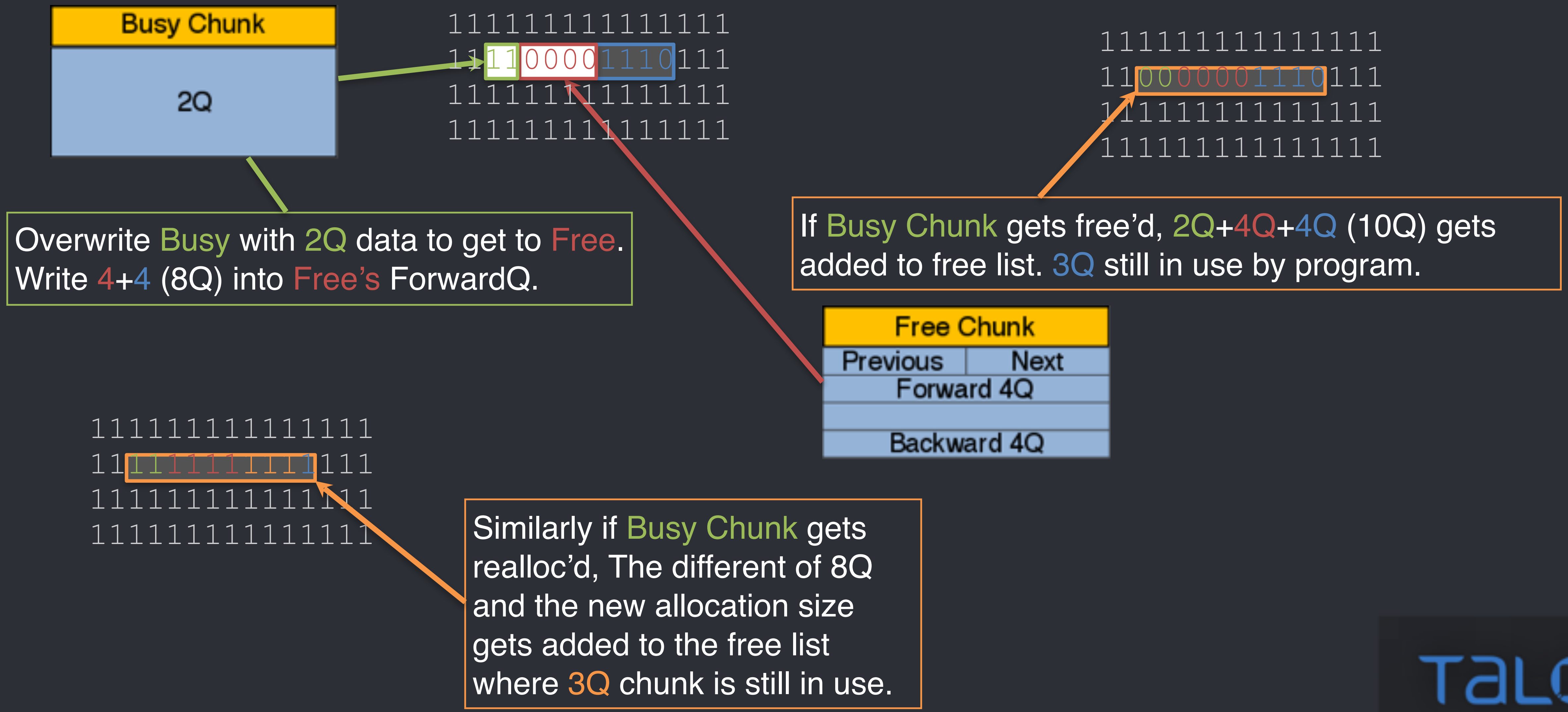
Freeing Busy chunk will coalesce with 3Q Chunk and use overwritten BackwardQ of **12** joining 2Q chunk with 7Q chunk.

Due to BackwardQ being **12**. Freeing 5Q chunk will read BackwardQ and add 7Q+2Q+3Q (**12Q**) to free list whilst chunk still being by program.

Strategies – mag_free_list

- Forwards Coalesce – When realloc() or moving from cache to free list.
  - Looks in ForwardQ of free-chunk to determine how far to coalesce or how far to realloc in-place...
  - When done, gets added to free list.

Strategies – mag_free_list – Coalesce

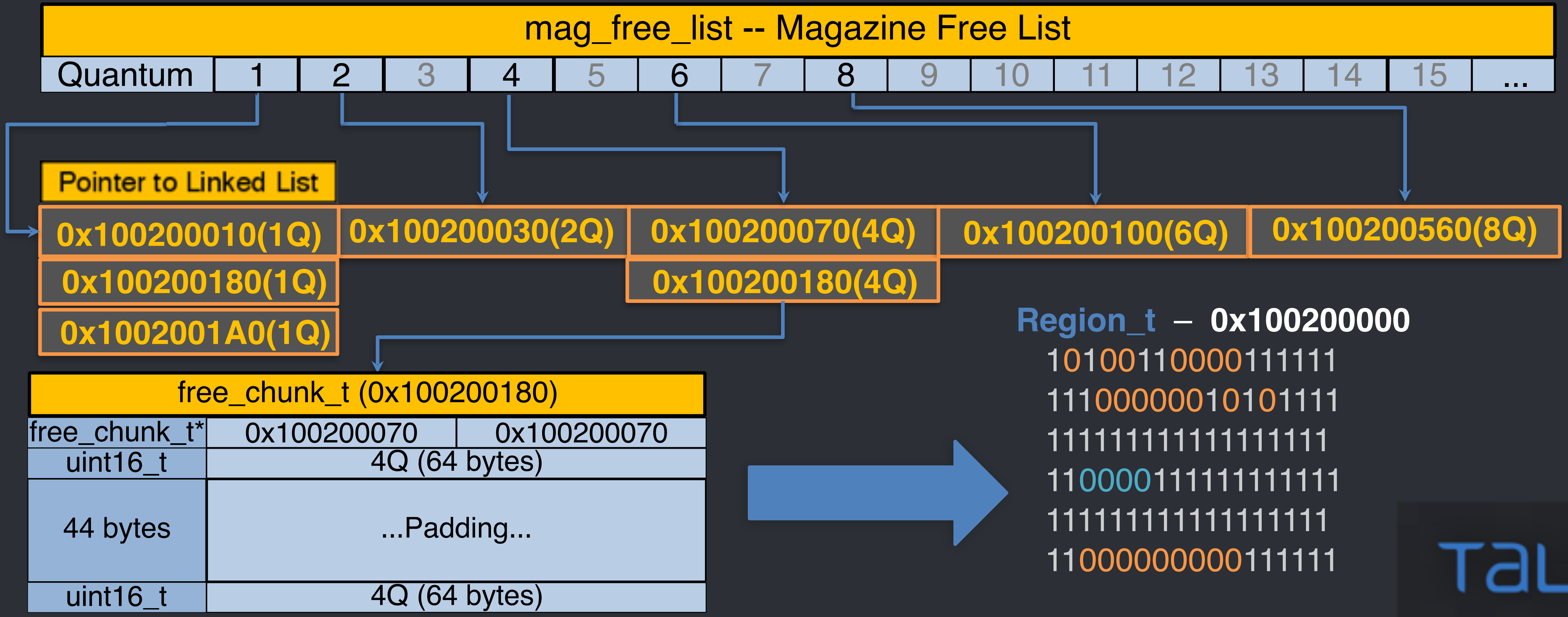


Strategies – mag_free_list

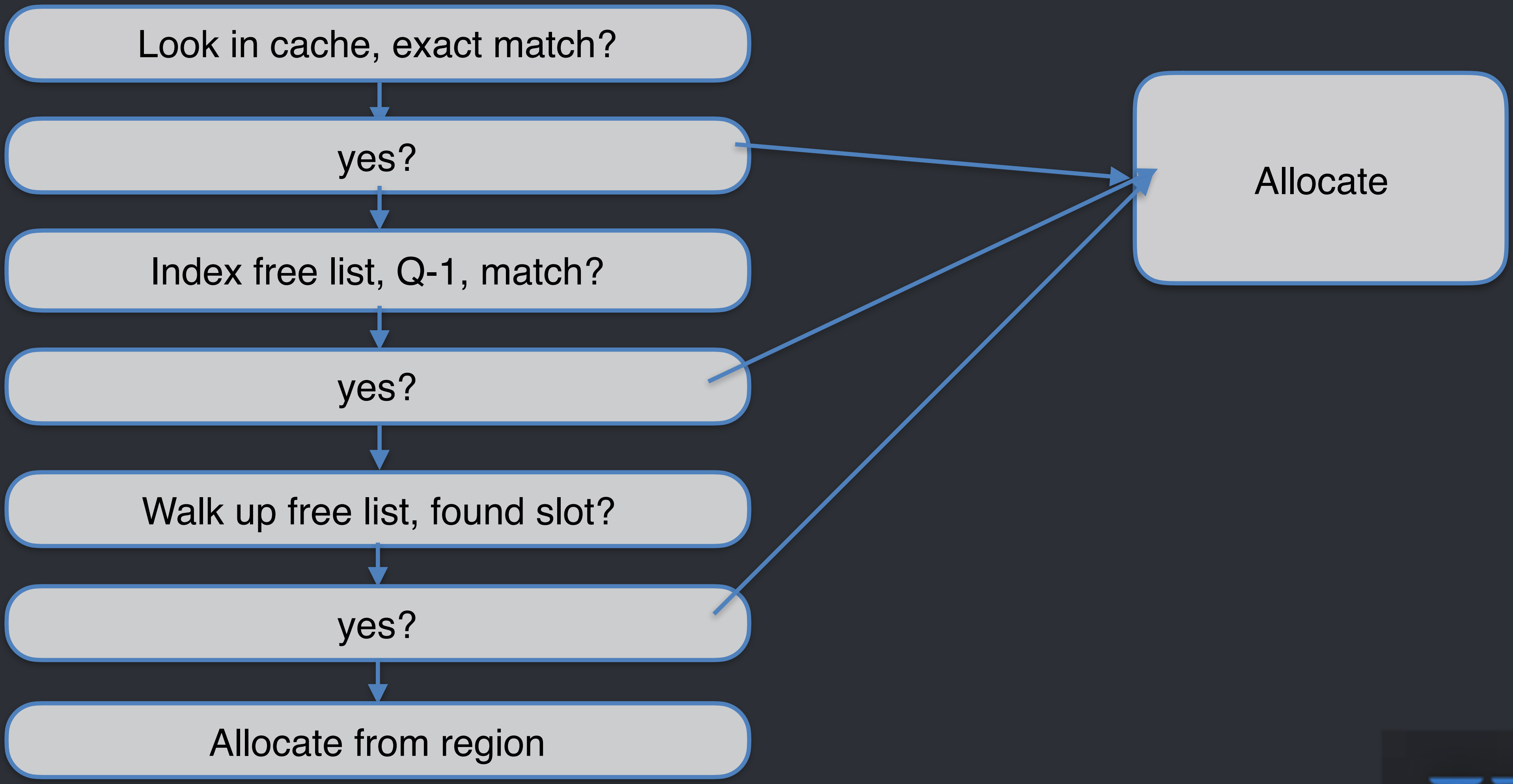
- Overwrite linked list at beginning of freed chunk
  - Write 4 with unlink - not super useful ASLR - 7% odds of getting correct check sum
  - If linked list pointers are NULL, then checksum evaluates to 0. Unfortunately, this clears the bit in mag_bitmap.
  - Need to move another chunk to free-list to set bit in bitmap, to re-gain access to coalesced entry.

Strategies – mag_free_list

- Allocating from the free list - 4Q

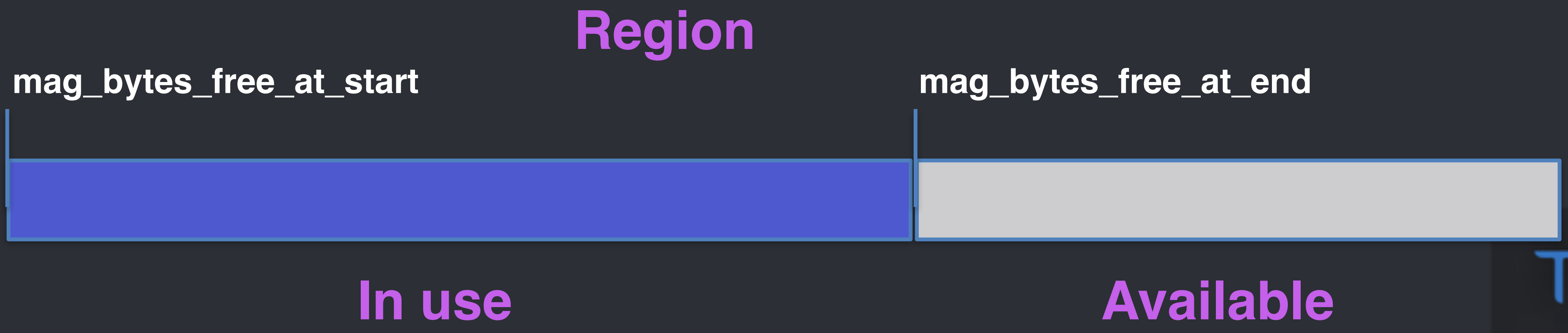


Strategies – for free list



Strategies – for region

- Allocate from region
    - calculate bytes currently in use
    - add to region base address
    - allocate space needed
    - free'd goes back on the free list



Demo

Strategies – for region

- Region Bitmap
  - Bitmap fitting
    - fill up single quanta slots inside of region bitmap to grow region bitmap
    - slot next allocation at end of in use region
    - allocations slotted together

Strategies – for region

```
>>>region.getoffset()
0x100200000
>>>region.bitmap()
```

Fragmented Bitmap
In Use = 1   Free = 0

```
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111110100110000 1111
1100000011111100000000111111111111111111111111111100000000000000000000000111111111111111111111000000000
000011111111111111111111111111111111111
```

```
>>> len(region.bitmap())
901
```

Strategies – for region

```
>>>region.getoffset()
0x100200000
>>>region.bitmap()
```

```
>>> print magazine.dump()
[1] 0x100202eb0
[2] 0x100202ed0
[4] 0x100202f10
[6] 0x100202fb0
[8] 0x100203070
[13] 0x100203550
[22] 0x1002032b0
```

```
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111010011000011111
1100000011111100000000111111111111111111111111111110000000000000000000000111111111111111111111000000000
0000111111111111111111111111111111111111
```

Strategies – for region

>>>t['mag_free_list'].quantum()
56

1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111101001100001111
1100000011111100000000111111111111111111111111111100000000000000000000001111111111111111111000000000
000011111111111111111111111111111111111

Strategies – for region

make 56 single quantum allocations to flatten bitmap

Strategies – for region

Fragmented Bitmap
In Use = 1   Free = 0

region.bitmap()

```
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111
```

```
>>> len(region.bitmap())
901
```

Strategies – for region

next allocation will grow the bitmap (subsequently the region) the necessary quantum

Strategies – for region

Fragmented Bitmap
In Use = 1   Free = 0

region.bitmap()

```
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111
```

```
>>> len(region.bitmap())
925
```

Strategies – Scoping bugs(?)

- Double Free
  - Not possible when freeing due to explicit checks against metadata and cache

Strategies – Scoping bugs(?)

- Heap Spraying
  - Blocks at beginning of boundary (0x100000 - tiny) (0x800000 - small)
  - 0xXXXFC080 through 0xXXXFFFFF - tiny structures
  - 0xXX800000 through 0xXXFF8000 - small structures

Demo

Debugging Strategies

LLDB Init

- LLDB lacks functionality
  - viewing page protections
  - dumping memory
  - expression handling
  - etc

LLDB Init

- Overly verbose commands (limited functionality)
    - breakpoint set -a `(void()) main`
    - breakpoint command add -o “x/x $rax” 1

LLDB Init

- LLDB lacks functionality
- Difficult or cumbersome for certain commands
- Python functionality inside of LLDB init is quite cumbersome

LLDB Init

- Created generic way of adding functions
- Alias and breakpoint managers
- Full expression parser built in to allow more WinDBG like commands, breakpoint management etc…
- ie. poi(main+$rax+local_1)

LLDB Init

- Single Core Script

```
script import com
breakpoint set -N singlemag1 -s libsystem_malloc.dylib -a 0x262a
breakpoint set -N singlemag2 -s libsystem_malloc.dylib -a 0x2a6a
breakpoint set -N singlemag3 -s libsystem_malloc.dylib -a 0xb95d
breakpoint set -N singlemag4 -s libsystem_malloc.dylib -a 0xbd43
breakpoint command add -F com.cont singlemag1
breakpoint command add -F com.cont singlemag2
breakpoint command add -F com.cont2 singlemag3
breakpoint command add -F com.cont2 singlemag4
```

Exploit Strategies

Safari

- 5 malloc zones
  - Default Malloc - version 8 (magazine alloc)
  - GFX Malloc - version 8 (magazine alloc)
  - WebKit Malloc - version 4 (bmalloc)
  - Quartz Core Malloc - version 5(scalable alloc)
  - Default Purgeable Malloc - version 8(magazine alloc)

Safari

- 5 malloc zones

MALLOC ZONE	VIRTUAL SIZE	ALLOCATION COUNT	BYTES ALLOCATED	% FULL	REGION COUNT
DefaultMallocZone_0x102226000	72.0M	16490	3466K	4%	5
WebKit Malloc_0x7fff7bf4ce68	12.0M	3	12.0M	100%	7
QuartzCore_0x7fa735009600	100K	767	46K	46%	10
GFXMallocZone_0x10245c000	0K	0	0K		0
DefaultPurgeableMallocZone_0x108036000	0K	0	0K		0
TOTAL	84.1M	17260	15.4M	18%	22

Vulnerability Details

- Image based heap overflow
- Core Foundation -> ImageIO
- Default malloc zone
- ImageIO has its own built in malloc (ImageIOMalloc) - it’s barely used :)

Exploitation Strategy

- Heap based overflow

- Primitives needed:
  - Information leak
  - Object to overwrite

- And here comes the zone problem…

Exploitation Strategy

- Majority of controllable allocations fall into the WebKit malloc
- However, calls to New go into the default zone
  - opens up quite a few objects to overwrite
- Information Leak still needed

Talos

Exploitation Strategy

- Heap massaging primitives found via Javascript's interaction with CoreFoundation

- AudioContext object makes reliable allocations in default malloc zone

- Multiple images needed to properly position heap, due to CoreFoundations and initialization and setup allocating to the default heap - CSS used for proper loading

Exploitation Strategy

- More reversing reveals one string allocated into our default zone
- Date.prototype.ToLocale
- Overwrite able via our overflow and still accessible via JavaScript

Exploitation Strategy

- Date.prototype.ToLocale … — 3-quantum allocation
- Empty a quantum after our locale
- Metadata overwrite -> backwards coalesce
- Get chunk freed -> clear the region bitmap
- Next allocation positions us in the middle of the locale string
- Remove the null terminator
- Read the date

Exploitation Strategy

- Information leak obtained
- Overwrite object
- Code execution 👻

TALOS

Conclusion

- https://github.com/blankwall/MacHeap
  - MacHeap tool
  - libsystem_malloc.idb
  - LLDB Init
- @1blankwall1