iOS 10 - Kernel Heap Revisited

<stefan.esser@sektioneins.de>

Singapore, August 2016

HITB GSEC
vote. attend. network.

SektionEins

Motivation behind this talk

- my talk about the iOS kernel heap was around time of iOS 5
- however many details have changed between iOS 5 and iOS 9
- there are a number of tweets/blog posts about some of these changes but not all
  (e.g. from Azimuth about iOS 7 page lists)
- some iOS kernel talks at BlackHat do not even mention that there were changes
- and no central talk trying to discuss all these changes

And then iOS 10 beta happened ...

- when this talk was submitted **no iOS 10 beta was publicly available**
- it was expected that iOS 10 would again **slightly** change the heap
- expectation was wrong because 1st iOS 10 beta showed bigger changes
- however iOS 10 is still in beta so details might change until final release

- **WARNING:** because any kind of kernel research on iOS is harder than on OS X/ MacOS the preliminary analysis of new features was performed with debugging kernel extensions on MacOS and then manually compared to decompilation of iOS 10 kernel

Agenda

- **Part I:** What is the iOS kernel heap?
- **Part II:** iOS kernel heap around iOS 5
- **Part III:** Changes to the iOS kernel heap between iOS 6 and 9
- **Part IV:** Upcoming changes to the iOS kernel heap in iOS 10

Part I

What is the iOS Kernel Heap?

Kernel Zone Heap Allocator

- most used heap allocator in kernel
- memory is divided into zones
- zones group allocations of same type/size together
- all allocations in one zone are same size

Zone Allocator Usage

- caller decides what zone is allocated from / freed to
    - `ptr = zalloc(zone)`
    - `zfree(zone, ptr)`
- size of allocated block depends on zone
- no variable length allocations

List of Zones

```
$ sudo zprint
Password:
                            elem         cur         max        cur         max         cur  alloc  alloc
zone name                   size        size        size      #elts       #elts       inuse   size  count
-------------------------------------------------------------------------------------------------------------
zones                        288         64K         54K        227         192         215    20K     71
vm.objects                   240      35280K      44286K     150528      188956      134502     4K     17   C
vm.object.hash.entries        40       4928K       5832K     126156      149299      120377     4K    102   C
maps                         248         92K         90K        379         371         329     8K     33
VM.map.entries                80       5832K       7776K      74649       99532       58296    20K    256   C
Reserved.VM.map.entries       80        356K       2560K       4556       32768         146     4K     51
VM.map.copies                 88         16K         24K        186         279           0     8K     93   C
VM.map.holes                  32        236K         16K       7552         512        5750     4K    128   C
pmap                         368        124K        144K        345         400         317     4K     11   C
pagetable.anchors           4096       1328K       1751K        332         437         317     4K      1   C
pv_list                       48      21864K      27217K     466432      580648      466200    12K    256   C
vm.pages                      64     127336K          0K    2037376           0     2034534     4K     64 X C
kalloc.16                     16       1136K       1167K      72704       74733       52626     4K    256   C
kalloc.32                     32       1348K       1751K      43136       56050       40037     4K    128   C
kalloc.48                     48       3308K       3941K      70570       84075       59071     4K     85   C
kalloc.64                     64       6128K       8867K      98048      141877       84972     4K     64   C
kalloc.80                     80       1776K       1751K      22732       22420       21808     4K     51   C
kalloc.96                     96       1748K       2335K      18645       24911       18276     8K     85   C
kalloc.128                   128       5352K       5911K      42816       47292       32835     4K     32   C
kalloc.160                   160       1340K       1556K       8576        9964        6495     8K     51   C
kalloc.256                   256       9328K      13301K      37312       53204       24442     4K     16   C
kalloc.288                   288       1660K       2594K       5902        9226        5563    20K     71   C
kalloc.512                   512      19324K      19951K      38648       39903       35135     4K      8   C
kalloc.1024                 1024       6024K       8867K       6024        8867        4945     4K      4   C
kalloc.1280                 1280        480K        512K        384         410         186    20K     16   C
kalloc.2048                 2048       6164K       8867K       3082        4433        2784     4K      2   C
kalloc.4096                 4096      12968K      13301K       3242        3325        1768     4K      1   C
kalloc.8192                 8192       6440K       7882K        805         985         412     8K      1   C
mem_obj_control               16       1956K       2187K     125184      139968      120377     4K    256   C
...
```

Zone Memory (Region)

- kernel reserves a memory region called **zone_map** for zone allocator
- default size of this region is 1/4 of the physical memory
- reserved very early on during kernel start

Zone Memory (Pages)

- pages from **zone_map** are assigned to zones
- zones are grown by their **allocation size**
- kernel pages are **16kb** on new 64 bit device with lots of memory
- all other devices have **4kb** pages
- to waste less space zones use optimal allocation sizes >= 1 page

Dynamic Length Allocations?

- zone allocator is not suited for dynamic length allocations
- but for dynamic length allocations various wrappers exist, e.g.:
  - ***kalloc(size) / kfree(ptr, size) -*** wrapper around zalloc()
    dynamic length but caller must remember length
  - ***MALLOC(size) / FREE(ptr) -*** wrapper around kalloc()
    dynamic length and uses meta data to remember length

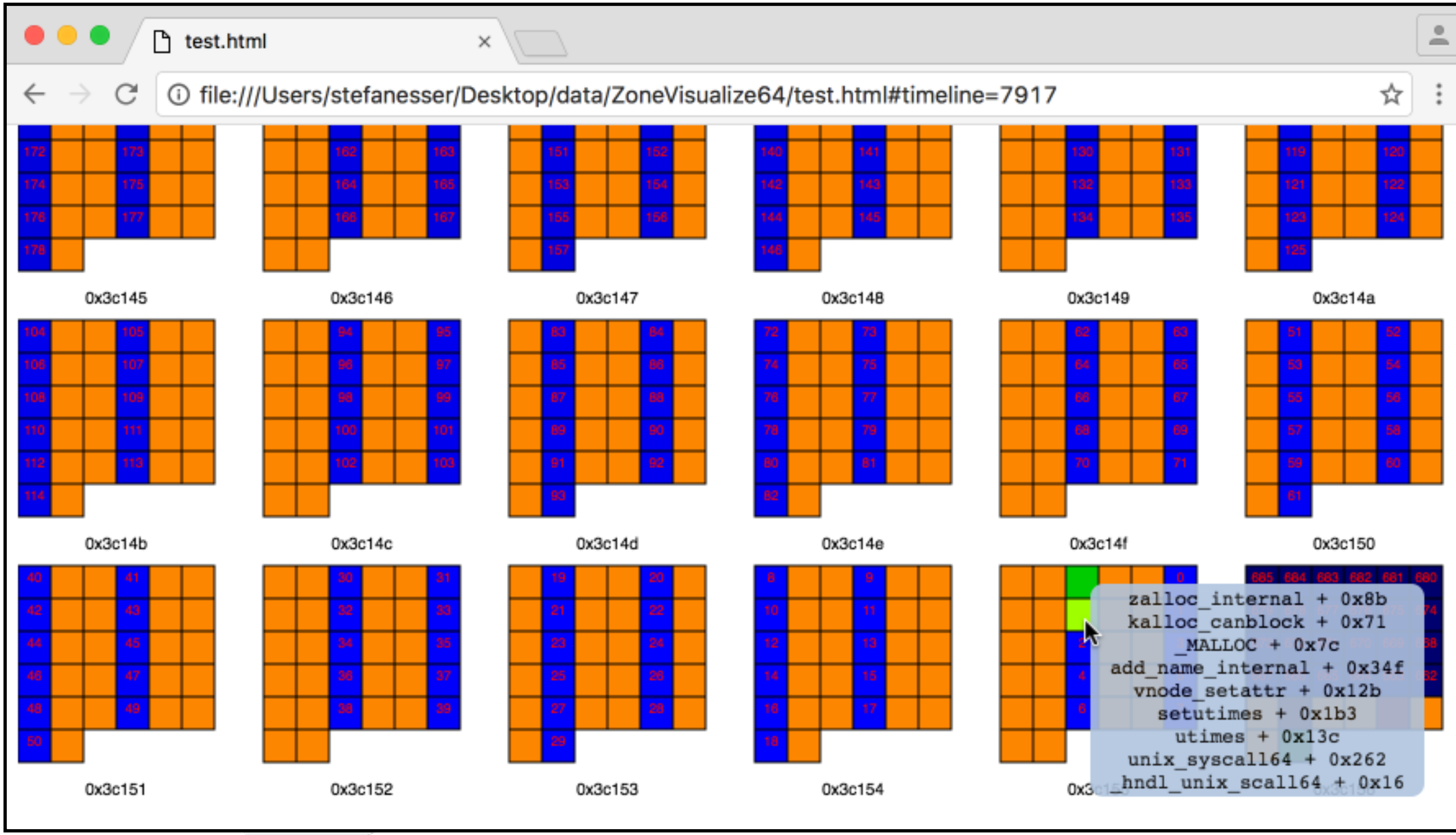
Kernel Heap Allocation Debugging (I)

- iOS has kernel boot arguments to enable heap allocation debugging
- for us the interesting boot arguments are
    - *-zc* selects zone corruption logging mode
    - *zlog=<zonename>* selects ONE zone to log
    - *zrecs=<number>* controls how many log entries should be kept
- because iOS does not allow to control kernel boot arguments these features need to be activated by means of a kernel exploit

Kernel Heap Allocation Debugging (II)

- kernel zone allocator **corruption logging** logs for **all (de)allocations**
  - if **allocation** or **deallocation**
  - **pointer** allocated / deallocated
  - kernel **backtrace** with up to **15** elements
- data is collected via **btlog** routines in kernel
- researcher need to create their own routine to extract data from kernel

Kernel Heap Allocation Debugging (III)



SektionEins

Part II

iOS Kernel Heap around iOS 5

Zone Structure

- information about a zone is stored in **zone** struct
- zone structs are stored in heap zone **"zones"**

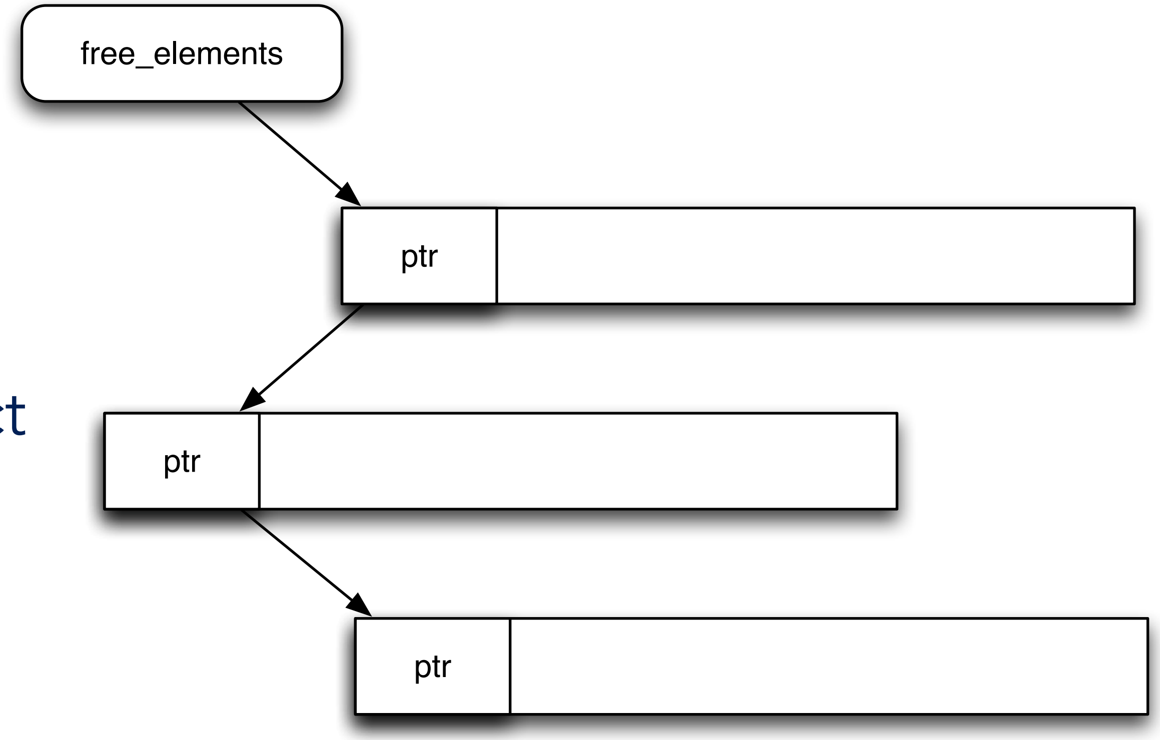
```
struct zone {
    int         count;      /* Number of elements used now */
    vm_offset_t free_elements;
    decl_lck_mtx_data(,lock)     /* zone lock */
    lck_mtx_ext_t    lock_ext;   /* placeholder for indirect mutex */
    lck_attr_t       lock_attr;  /* zone lock attribute */
    lck_grp_t        lock_grp;   /* zone lock group */
    lck_grp_attr_t   lock_grp_attr;  /* zone lock group attribute */
    vm_size_t   cur_size;    /* current memory utilization */
    vm_size_t   max_size;    /* how large can this zone grow */
    vm_size_t   elem_size;   /* size of an element */
    vm_size_t   alloc_size;  /* size used for more memory */
    uint64_t    sum_count;   /* count of allocs (life of zone) */
    unsigned int
    /* boolean_t */ exhaustible :1, /* (F) merely return if empty? */
    /* boolean_t */ collectable :1, /* (F) garbage collect empty pages */
    /* boolean_t */ expandable :1,  /* (T) expand zone (with message)? */
    /* boolean_t */ allows_foreign :1,/* (F) allow non-zalloc space */
    /* boolean_t */ doing_alloc :1, /* is zone expanding now? */
    /* boolean_t */ waiting :1, /* is thread waiting for expansion? */
    /* boolean_t */ async_pending :1,   /* asynchronous allocation pending? */
    /* boolean_t */ caller_acct: 1, /* do we account allocation/free to the caller? *
    /* boolean_t */ doing_gc :1,    /* garbage collect in progress? */
    /* boolean_t */ noencrypt :1,
    /* boolean_t */ no_callout:1,
    /* boolean_t */ async_prio_refill:1;
    int         index;      /* index into zone_info arrays for this zone */
    struct zone *   next_zone;  /* Link for all-zones list */
    call_entry_data_t   call_async_alloc;   /* callout for asynchronous alloc */
    const char  *zone_name; /* a name for the zone */

    vm_size_t   prio_refill_watermark;
    thread_t    zone_replenish_thread;
};
```

DISCLAIMER: this struct layout if taken from OS X 10.7.5 - the iOS 5 layout might have been slightly different

Free Memory Blocks

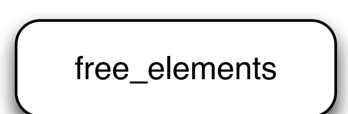
- free elements are kept in a **single linked freelist** per zone
- zone structure has **free_elements** pointer to head of freelist
- free elements have a **pointer to next free** element in beginning
- there is **no pointer(-chain) back** to zone struct
    - **not possible** to know **what zone** an element (a page) **belongs to**
    - **slow garbage collection**



SektionEins

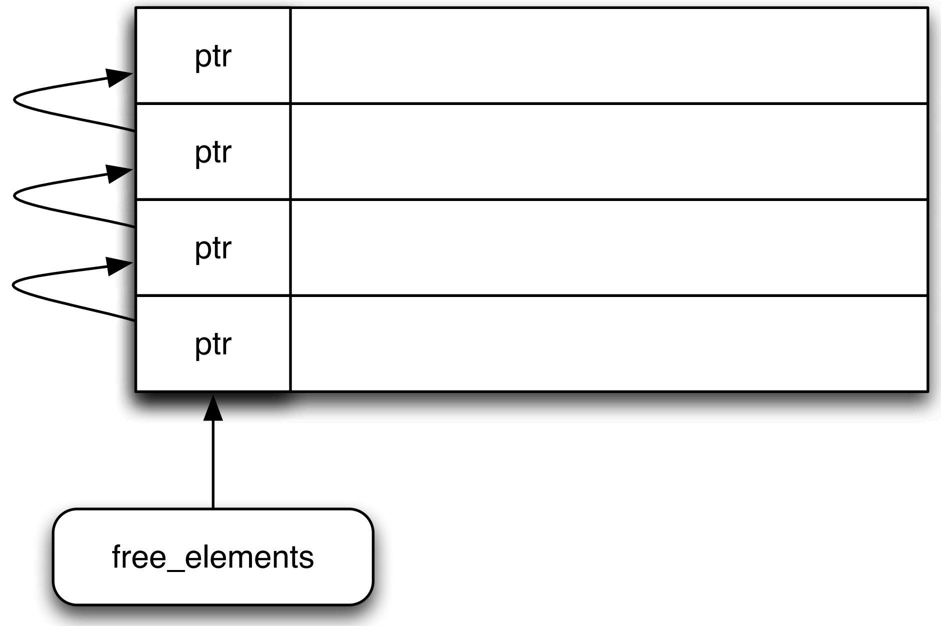
Allocation (I)

- allocation returns next element from freelist
- when free list is empty zone grows by allocation size
- all elements in added pages are added to free list
- last element will be first in freelist (LIFO)



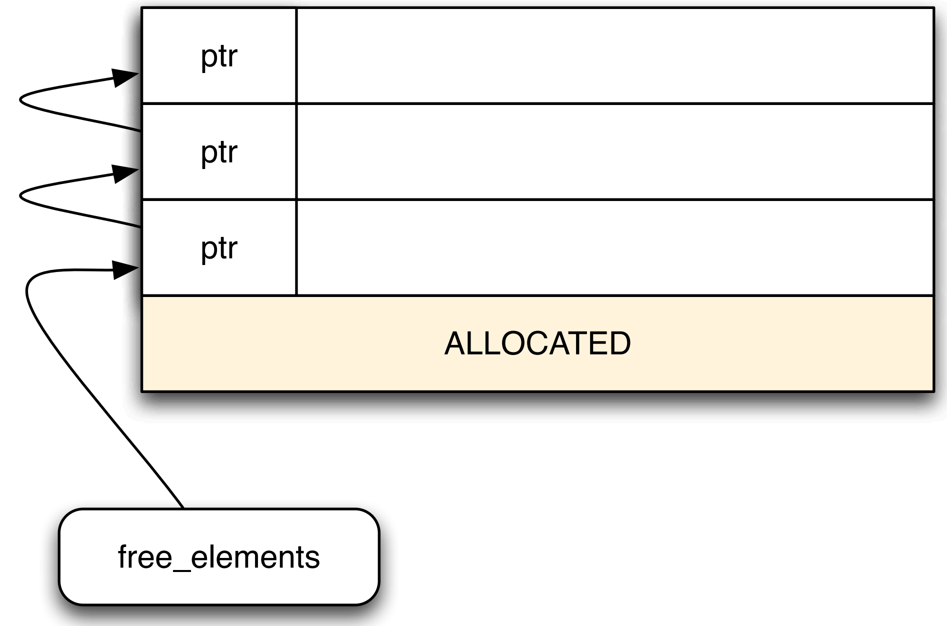
Allocation (II)

- allocation returns next element from freelist
- when free list is empty zone grows by allocation size
- all elements in added pages are added to free list
- last element will be first in freelist (LIFO)



Allocation (III)

- allocation returns next element from freelist
- when free list is empty zone grows by allocation size
- all elements in added pages are added to free list
- last element will be first in freelist (LIFO)

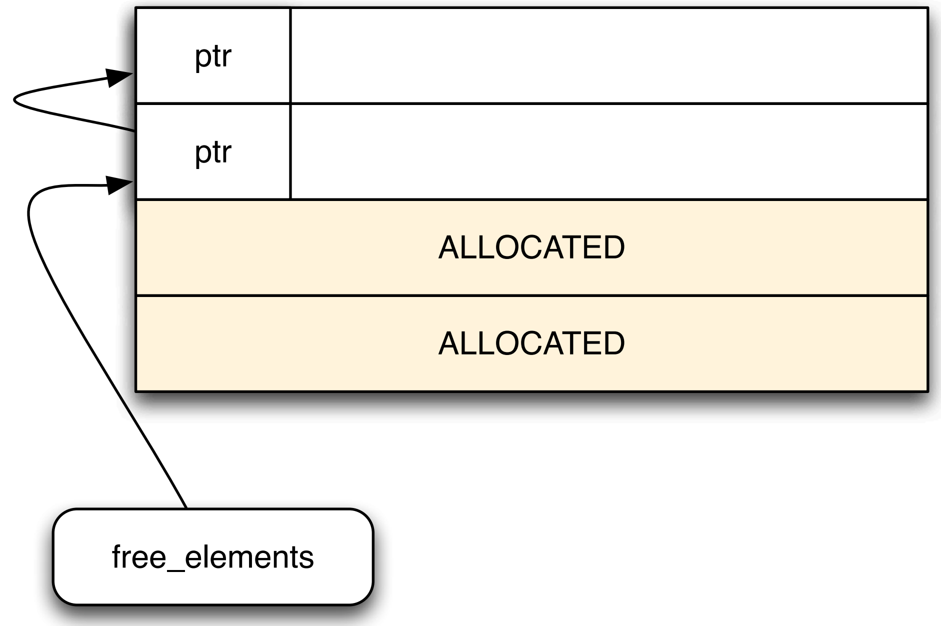


"new memory is returned in backward order by heap allocator"

SektionEins

Allocation (IV)

- allocation returns next element from freelist
- when free list is empty zone grows by allocation size
- all elements in added pages are added to free list
- last element will be first in freelist (LIFO)



"new memory is returned in backward order by heap allocator"

Dynamic Length Allocations - kalloc()

- to handle dynamic lengths **kalloc()** registers multiple zones
- named ***kalloc.<number>***

  e.g. ***kalloc.128*** for **128** byte allocations
- allocations will be put into the next larger zone

  e.g. **97** byte allocations in **kalloc.128** (**31** bytes waste)
- different iOS versions define different zones

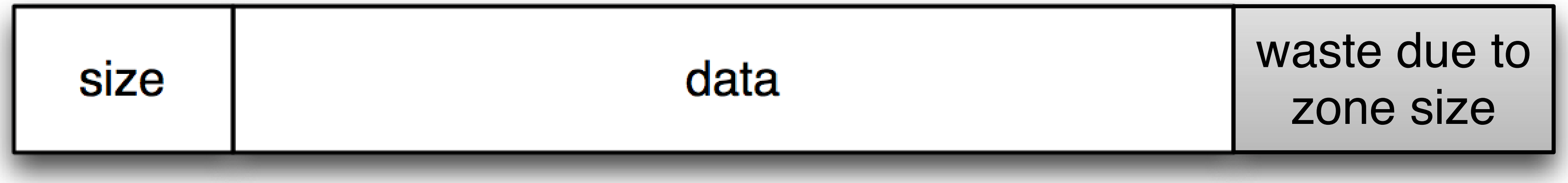
  (also depending on 32 bit vs. 64 bit)
- caller needs to remember allocation size so that

  kfree(ptr, size) can put it back into the right zone

kalloc.16
kalloc.32
kalloc.48
kalloc.64
kalloc.80
kalloc.96
kalloc.128
kalloc.160
kalloc.256
kalloc.288
kalloc.512
kalloc.1024
kalloc.1280
kalloc.2048
kalloc.4096
kalloc.8192

Dynamic Length Allocations - MALLOC()

- to handle dynamic lengths **MALLOC()** stores the size as meta data
- internally uses **kalloc()** / **kfree()**
- stores the size infront of data



SektionEins

How attackers abused the iOS 5 Zone Allocator

- **Heap-Feng-Shui**
  - exploit specific allocation and deallocation primitives
    (e.g. opening/closing NDRV sockets)
  - generic allocation deallocation primitives via **OSUnserializeXML**
    (controlling memory layout via many IOKit API functions using XML - and filling it with objects and object pointers)
- **Corruption Targets**
  - zone allocator freelist **next_ptr** pointers
    (control ptr next allocation will return and overwrite with attacker controlled data)
  - **size fields** of elements allocated via **MALLOC()** or a wrapper
    (tricks kfree() to put element into a zone that is for bigger elements - next allocation will result in a larger bufferoverflow)
  - **application data**
    (control objects/vtable pointers, object pointers, size fields, …)

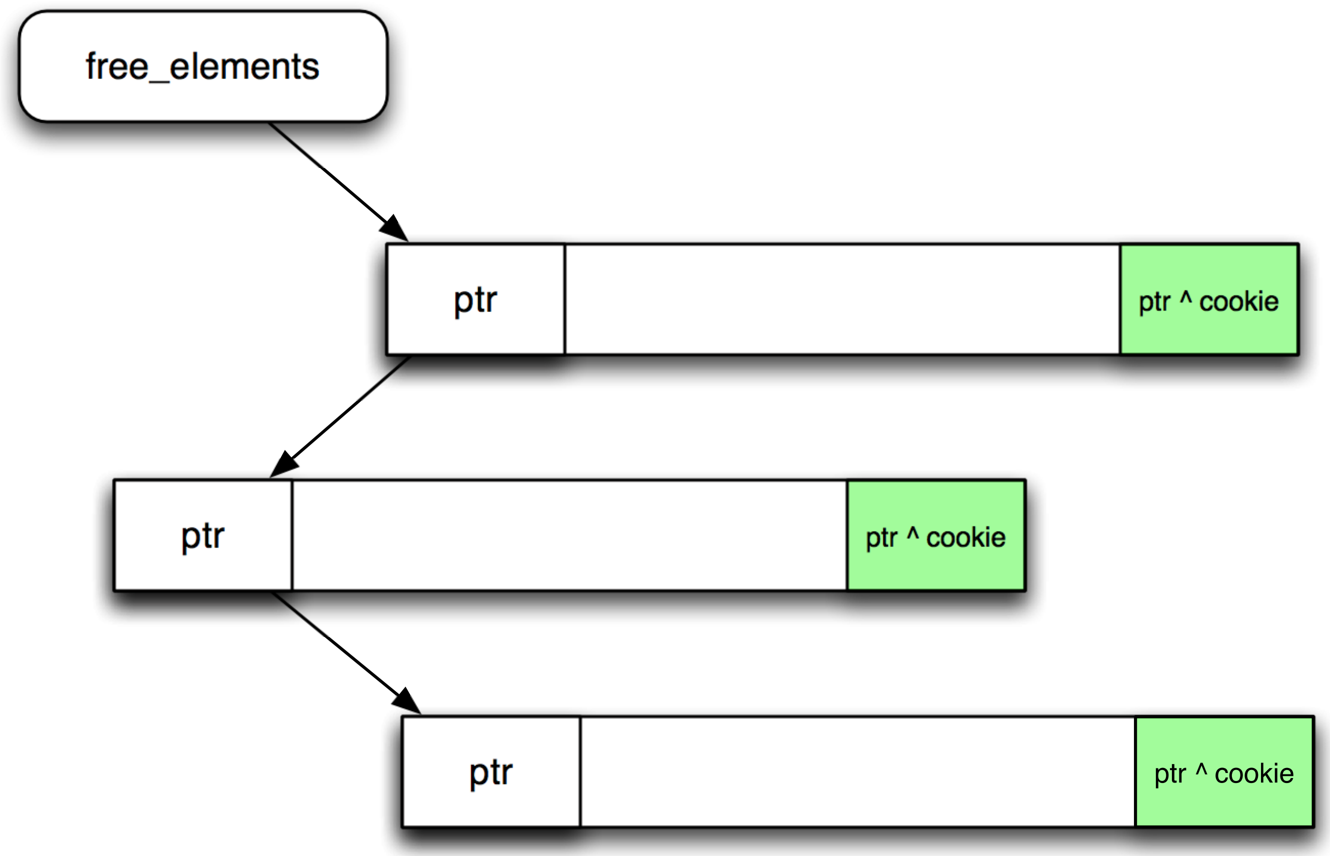
Part II

Changes to the iOS kernel heap between iOS 6 and 9

iOS 6 Heap Changes/Hardening

- Apple made some changes to **OSUnserializeXML** that had little impact on usability for heap feng shui
- location of **zone_map** is **randomized** due to **KASLR**
- single linked freelist now protected by **heap cookies/canaries**
- small free memory blocks are now **poisoned**

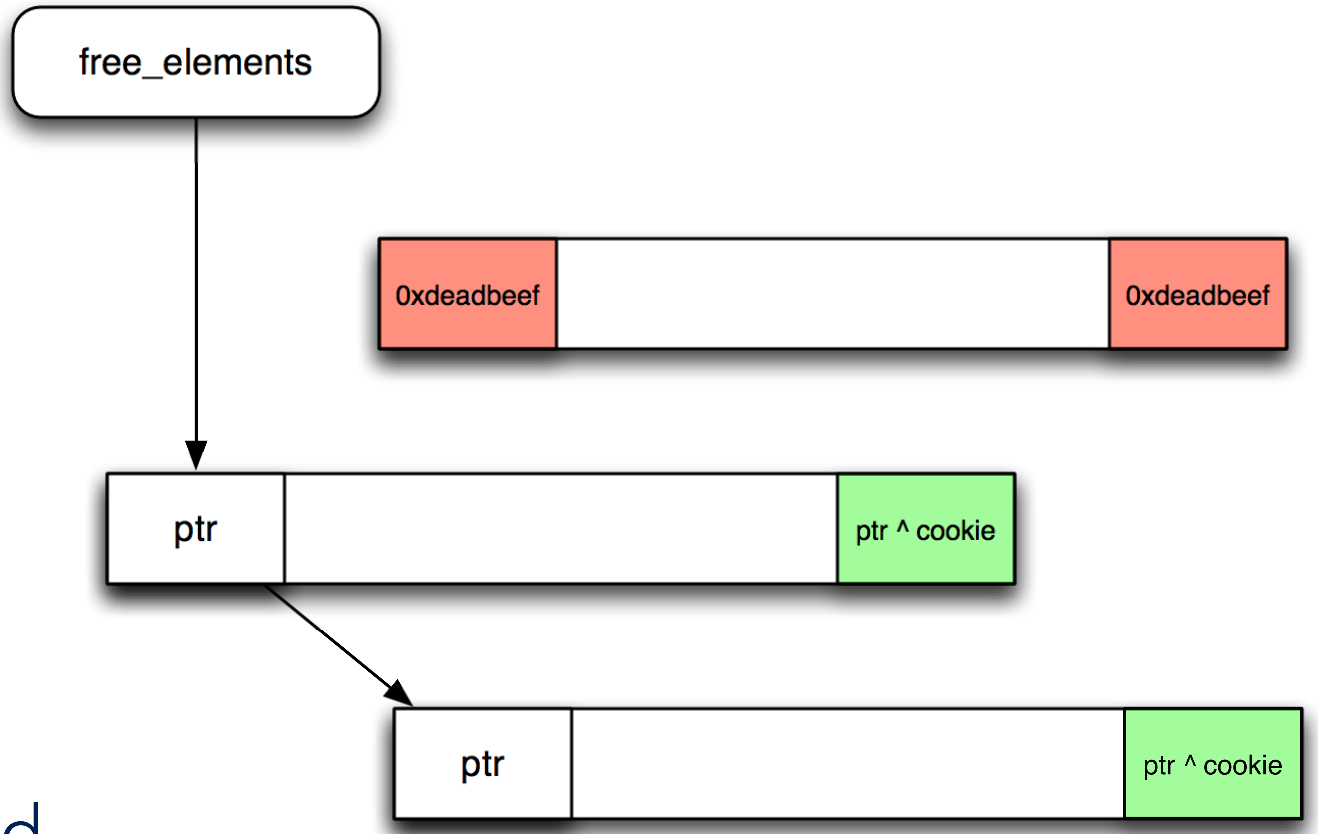
iOS 6 Heap Cookies



- iOS creates two random cookies
  - **zp_nopoison_cookie** (lowest bit cleared)
  - **zp_poisoned_cookie** (lowest bit set)
- last bytes of a free element will be overwritten with: **ptr ^ cookie**
- allocation code will detect illegal cookie values and **panic()**
- protection against overflows and other **ptr** corruption

SektionEins

iOS 6 Heap Cookie Leak Protection

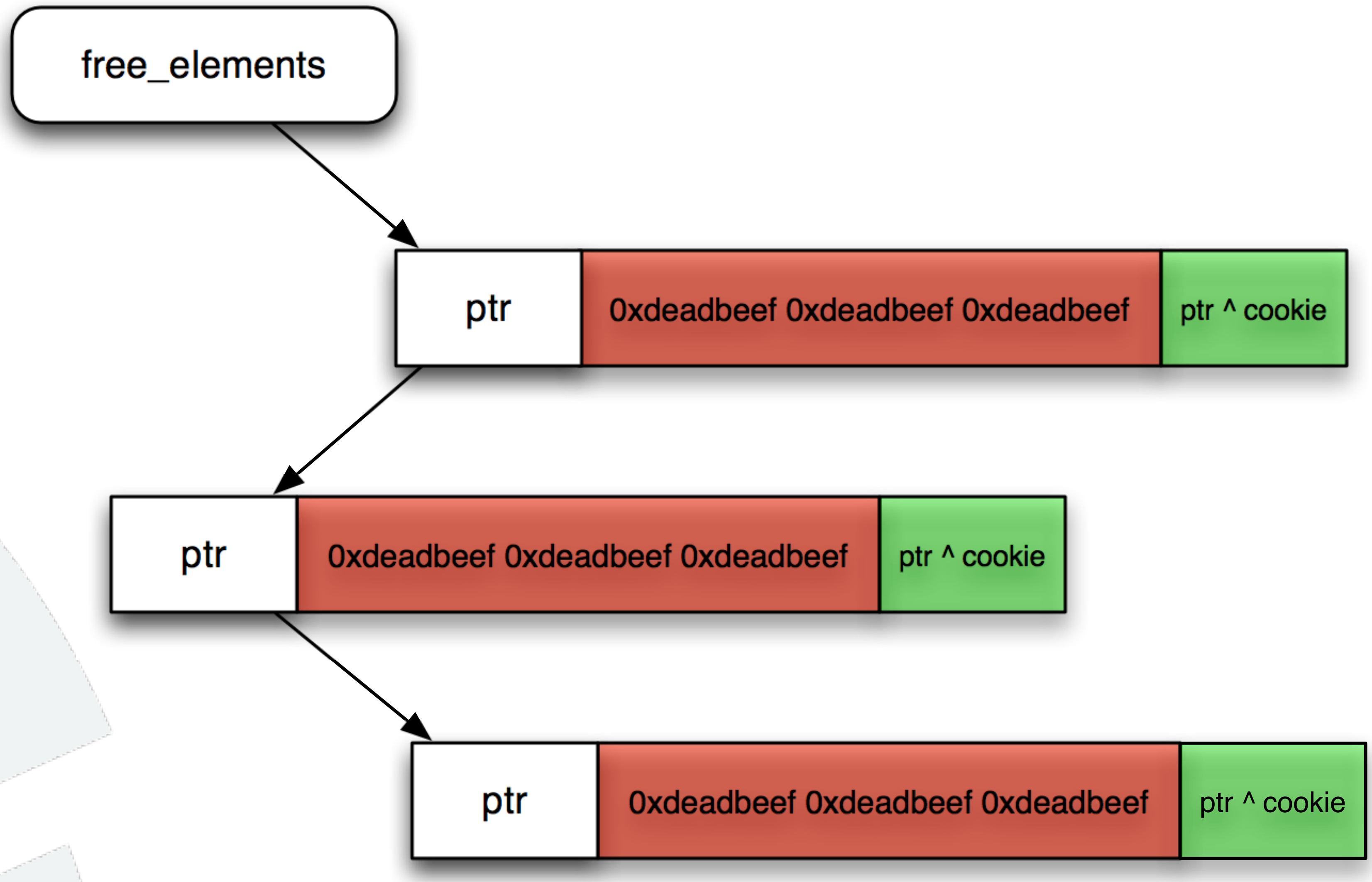


- both **ptr** and **ptr^cookie** get overwritten when block is allocated
- value **0xdeadbeef** is written over sensitive values
- protects against potential information leak after memory block is returned

SektionEins

iOS 6 Heap Poisoning

- small blocks when freed get overwritten with **0xdeadbeef**
- indicated by use of **zp_poisoned_cookie**
- on allocation **0xdeadbeef** is verified - **panic()** if modified



How attackers abused the iOS 6 Zone Allocator

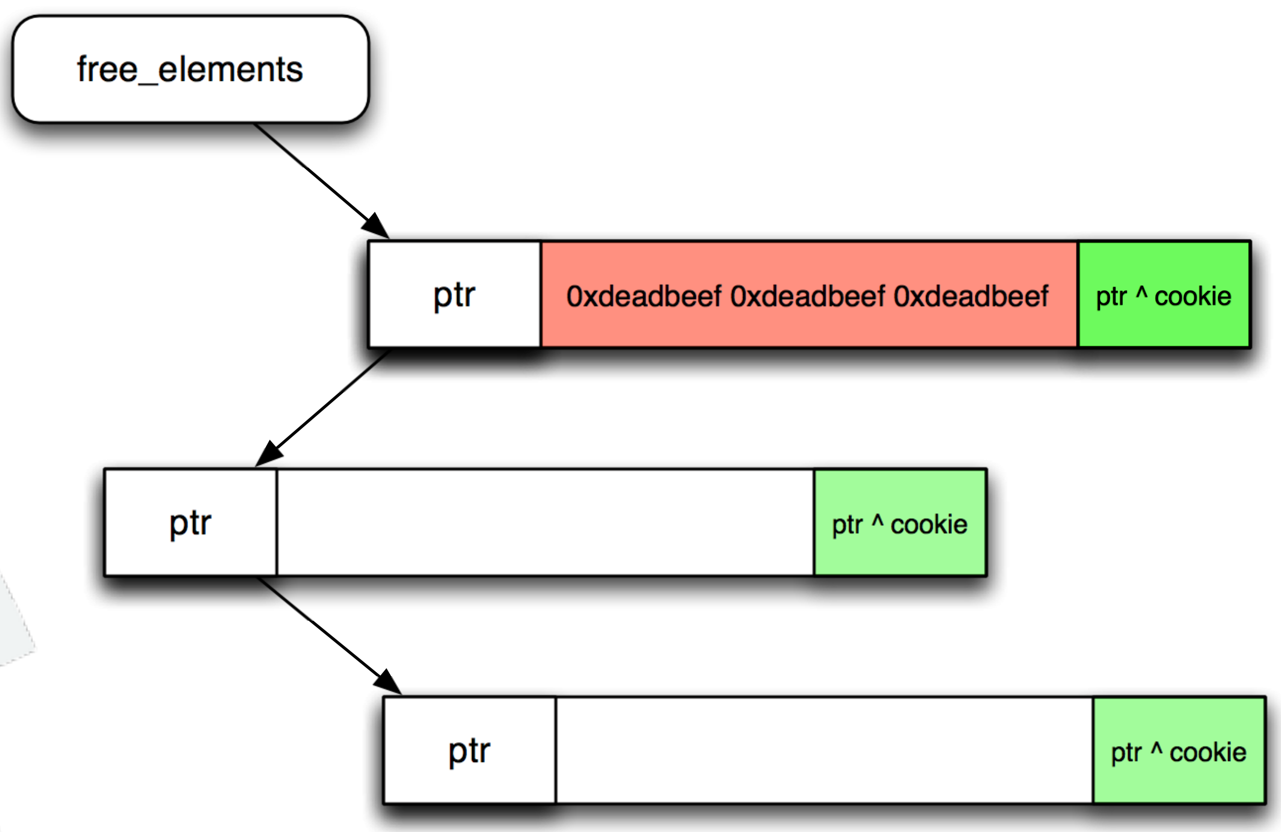
- **Heap-Feng-Shui**
  - with release of iOS 6 creating **vm_map_copy_t** structures by sending **mach messages with OOL data** became heap-feng-shui method of choice
  - previously used methods less often seen but still possible
- **Corruption Targets**
  - because zone allocator freelist got protected all public exploits seem to target **vm_map_copy_t** structure overwrites
    (could be used for arbitrary info leak and kfree() confusion)
  - other size fields of e.g. **MALLOC**() and application data still targeted

iOS 7 Heap Changes/Hardening

- poisoning of larger blocks every **X** frees
- introduction of new **heap pagelist feature** (for easier GC)

iOS 7 Large Block Poisoning

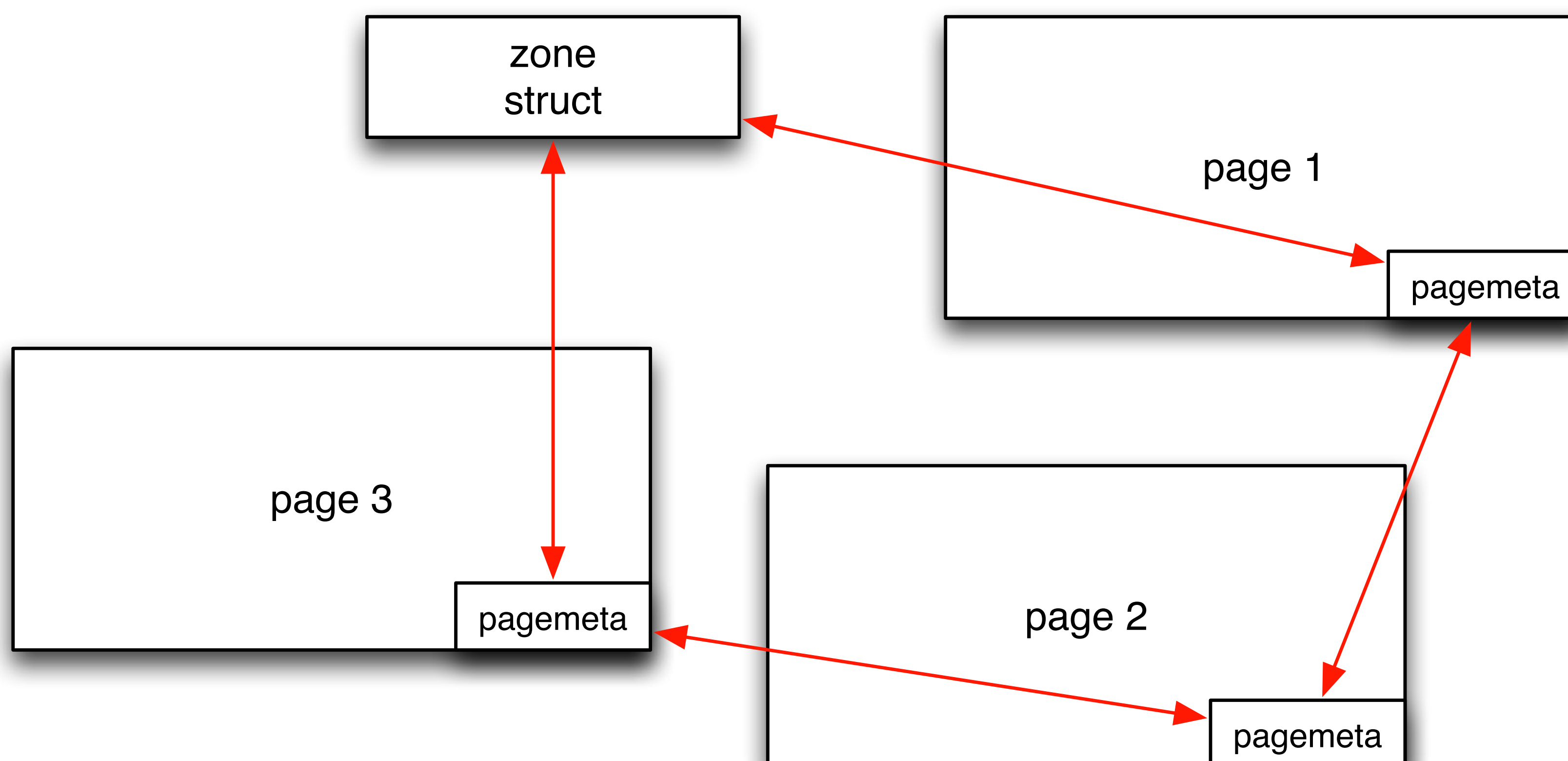
- **zp_factor** debugging feature *(that already existed before)* is now always activated
- randomly set once at boot time to the value **15** *(25%)*, **16** *(50%)* or **17** *(25%)*
- controls after every how many frees in a zone a single block is poisoned (regardless of size)
- counter is zone specific in the **zp_count** field of the zone structure



SektionEins

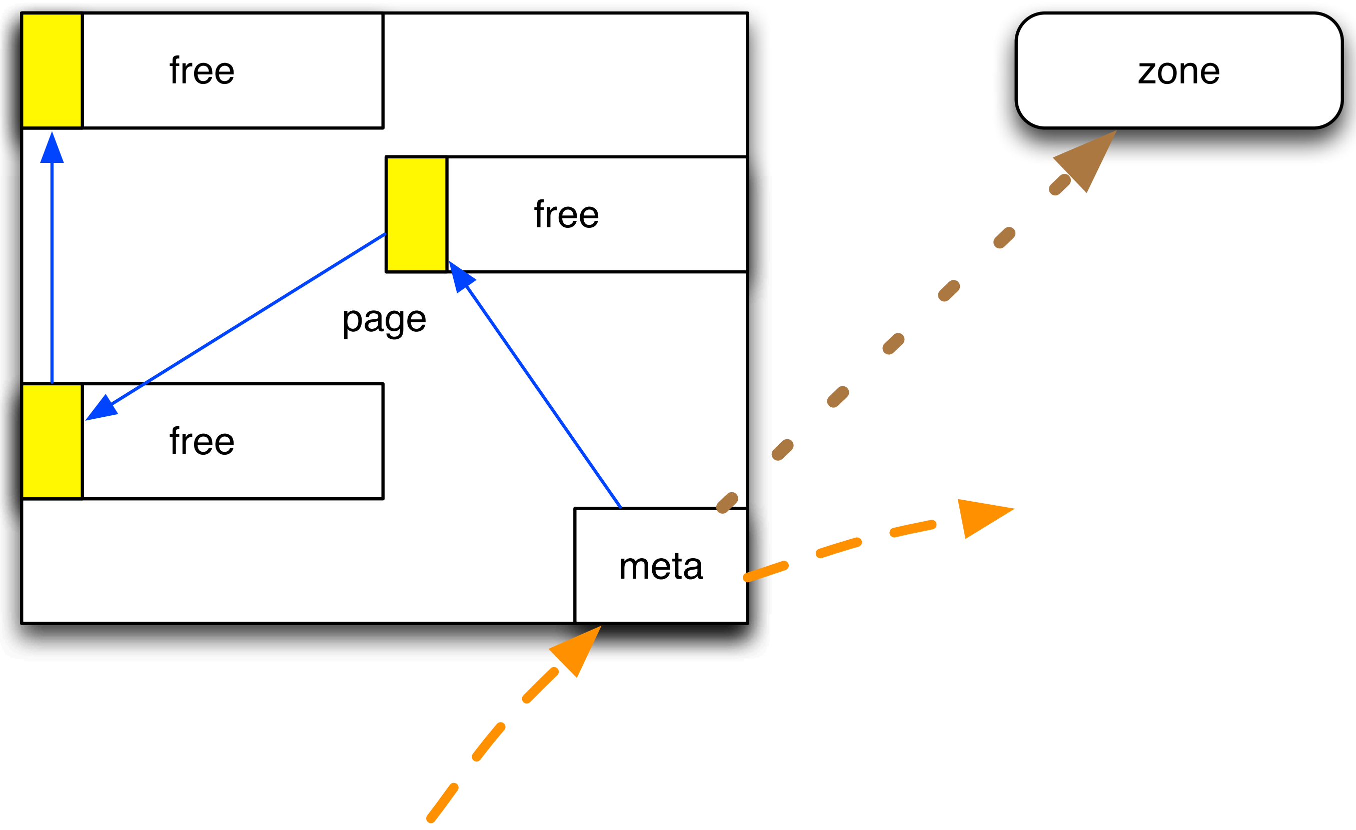
iOS 7 Zone Pagelist Feature

- iOS 7 introduces **new pagelist feature**
- activated via **bit** in zone struct *(only* ***subset of zones*** *use new feature)*
- adds **meta data at end of all pages** inside a zone
- keeps all pages in one of four **double linked lists (queue)**



SektionEins

iOS 7 Zone Page Meta Data




```
struct zone_page_metadata {
    queue_chain_t                pages;
    struct zone_free_element     *elements;
    zone_t                       zone;
    uint16_t                     alloc_count;
    uint16_t                     free_count;
};
```

- backpointer to **zone**
- free **elements** in this page (page local freelist)
- forward and backward pointer to other **pages**
- **alloc_count -** max number of elements in page
- **free_count -** number of free elements in page

SektionEins

iOS 7 Zone Pagelists

- Allocator defines four queues for every zone
    - **any_free_foreign** - for the few zones that allow foreign elements
    - **intermediate** - for pages that are partially allocated at the moment
    - **all_free** - for pages that are completely free at the moment
    - **all_used** - for pages that are completely allocated at the moment
- during allocation or free the allocator ensures that page is always on right queue

iOS 7 Allocation under Page Lists

- allocation now traverses the page queues in the following order
    - **any_free_foreign**
    - **intermediate**
    - **all_free**
- first queue with a usable page is used
- then the first free element from this page's freelist is returned
- if no usable page found - system adds an **all_free** page and retries

iOS 7 Freeing under Page Lists

- freeing an **element** adds it to its page’s **freelist**
- allocator ensures that page is still on **right queue**
- page could move
    - from **intermediate** to **all_free**
    - from **all_used** to **intermediate**

Impact of iOS7 Page Lists

- garbage collection now super easy
  (just give back all pages from the all_free page queue)
- freeing of memory is **local to each page's freelist**
  (less interruption for heap-feng-shui)
- allocations are **local to current front page**
  (not really a change because compatible exploits always had to concentrate on staying in one page)
- meta data contained **double linked list without any exploit mitigation**
  (iOS 7 made heap overflows very easy to exploit)
- ***NEW ATTACK:***
  confusing allocator to free elements from page list zones in freelist zones

Was there a memory corruption? Yes? Continue!

- Apple added code that detects for page list zones if a corruption happened
- **when they detect a corruption they try to repair it <—— NEVER EVER DO THAT!**
- **when they cannot repair they ignore that they detected a memory corruption <—— APPLE REALLY???**

```
if (zone->use_page_list) {
    struct zone_page_metadata *page_meta = get_zone_page_metadata((struct zone_free_element *)addr);
    if (zone != page_meta->zone) {
        /*
         * Something bad has happened. Someone tried to zfree a pointer but the metadata says it is from
         * a different zone (or maybe it's from a zone that doesn't use page free lists at all). We can repair
         * some cases of this, if:
         * 1) The specified zone had use_page_list, and the true zone also has use_page_list set. In that case
         *    we can swap the zone_t
         * 2) The specified zone had use_page_list, but the true zone does not. In this case page_meta is garbage,
         *    and dereferencing page_meta->zone might panic.
         * To distinguish the two, we enumerate the zone list to match it up.
         * We do not handle the case where an incorrect zone is passed that does not have use_page_list set,
         * even if the true zone did have this set.
         */
        zone_t fixed_zone = NULL;
        int fixed_i, max_zones;

        simple_lock(&all_zones_lock);
        max_zones = num_zones;
        fixed_zone = first_zone;
        simple_unlock(&all_zones_lock);

        for (fixed_i=0; fixed_i < max_zones; fixed_i++, fixed_zone = fixed_zone->next_zone) {
            if (fixed_zone == page_meta->zone && fixed_zone->use_page_list) {
                /* we can fix this */
                printf("Fixing incorrect zfree from zone %s to zone %s\n", zone->zone_name, fixed_zone->zone_name);
                zone = fixed_zone;
                break;
            }
        }
    }
}
```

How attackers abused the iOS 7 Zone Allocator

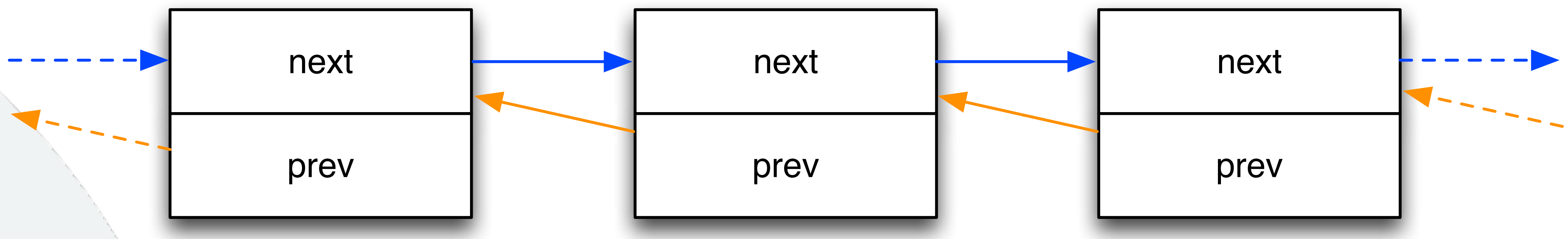
- **Heap-Feng-Shui**
  - same methods as before were used
- **Corruption Targets**
  - all previously targeted areas still work
  - but double linked lists introduced by pagelist feature easiest target
    (target the next/prev pointers for arbitrary writes)

iOS 8 Heap Changes/Hardening

- pagelist queue hardening
- poisoning of larger blocks made less frequent (depending on size)

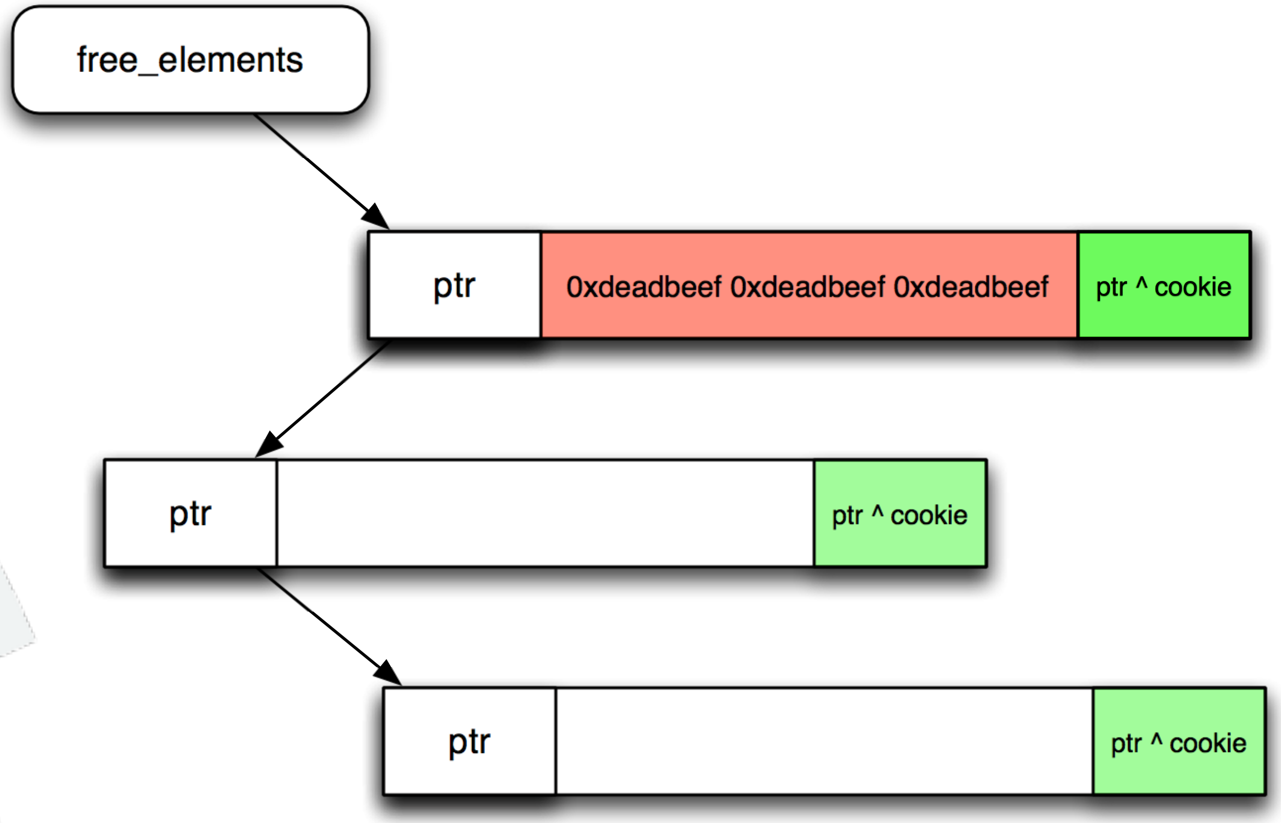
iOS 8 Queue Hardening

- iOS 8 adds safe unlink protection to its queue macros
- double linked lists added to iOS 7 heap are now protected
- heap overflows can no longer go after the queue pointers in metadata



iOS 8 Less Frequent Large Block Poisoning

- introduction of **zp_scale** feature with a default value of **4**
- poisoning of large blocks follows now the following formula
  zp_factor + element_size >> zp_scale
- this means the larger element are in a zone the less often they are poisoned
- example: *(15/16/17) + 256 >> 4 = (31/32/33)*



How attackers abused the iOS 8 Zone Allocator

- **Heap-Feng-Shui**
  - same methods as before were used
- **Corruption Targets**
  - targeting double linked lists (*pagelists*) not possible due to **safe_unlink**
  - all other previously targeted areas still work

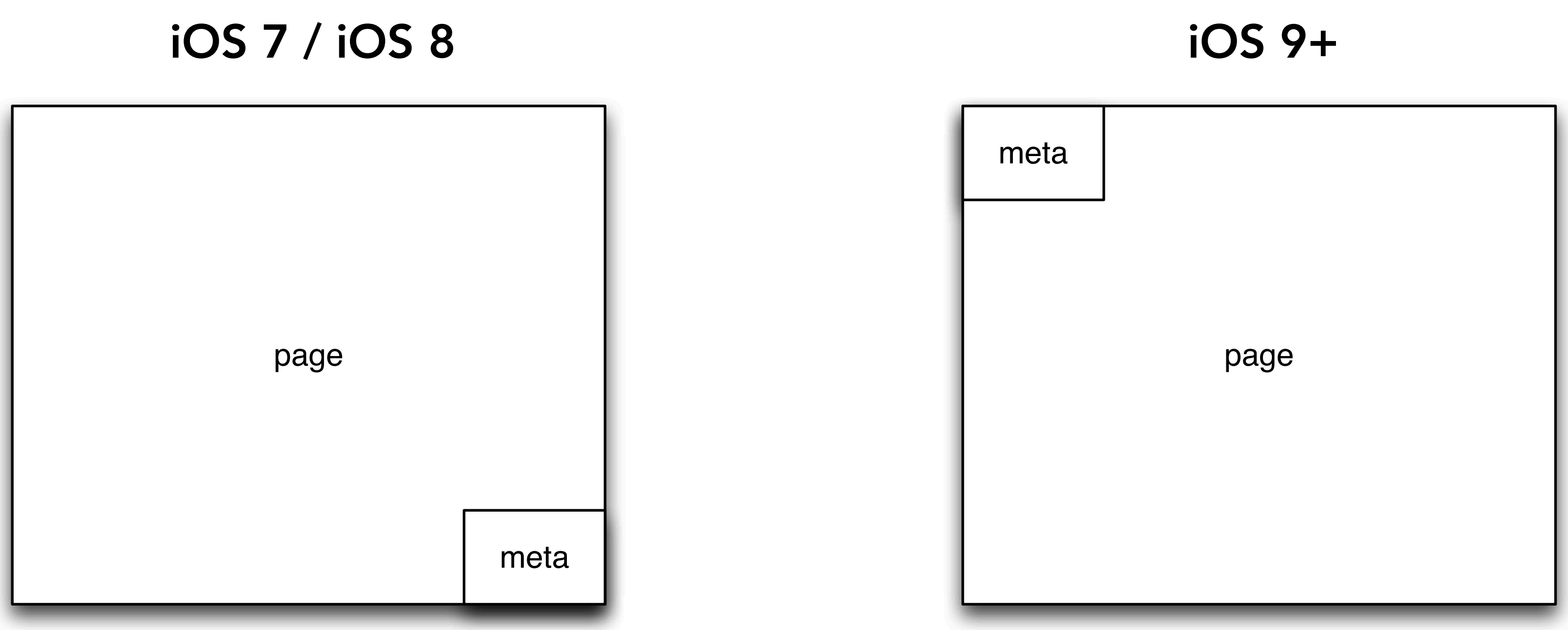
iOS 9 Heap Changes/Hardening

- ***vm_map_copy_t*** “hardening”
- **repositioning** of page metadata
- **randomization** of initial freelist

iOS 9 vm_map_copy_t "hardening"

- **vm_map_copy_t** structure for kernel buffers was stripped down
    - **data pointer removed** because data should be after header
    - secondary **size field** removed because that is **headerlen + size**
- smaller structure **allows controlling** heap in **smaller zones**
- when overwritten can **still lead to zone confusion and info leaks**
- but info leaks are **limited to 4k because hardcoded limit** in code
- and removal of pointer **removes possibility for arbitrary info leaks**
- some places try to verify that size is not overwritten

iOS 9 Zone Page Metadata

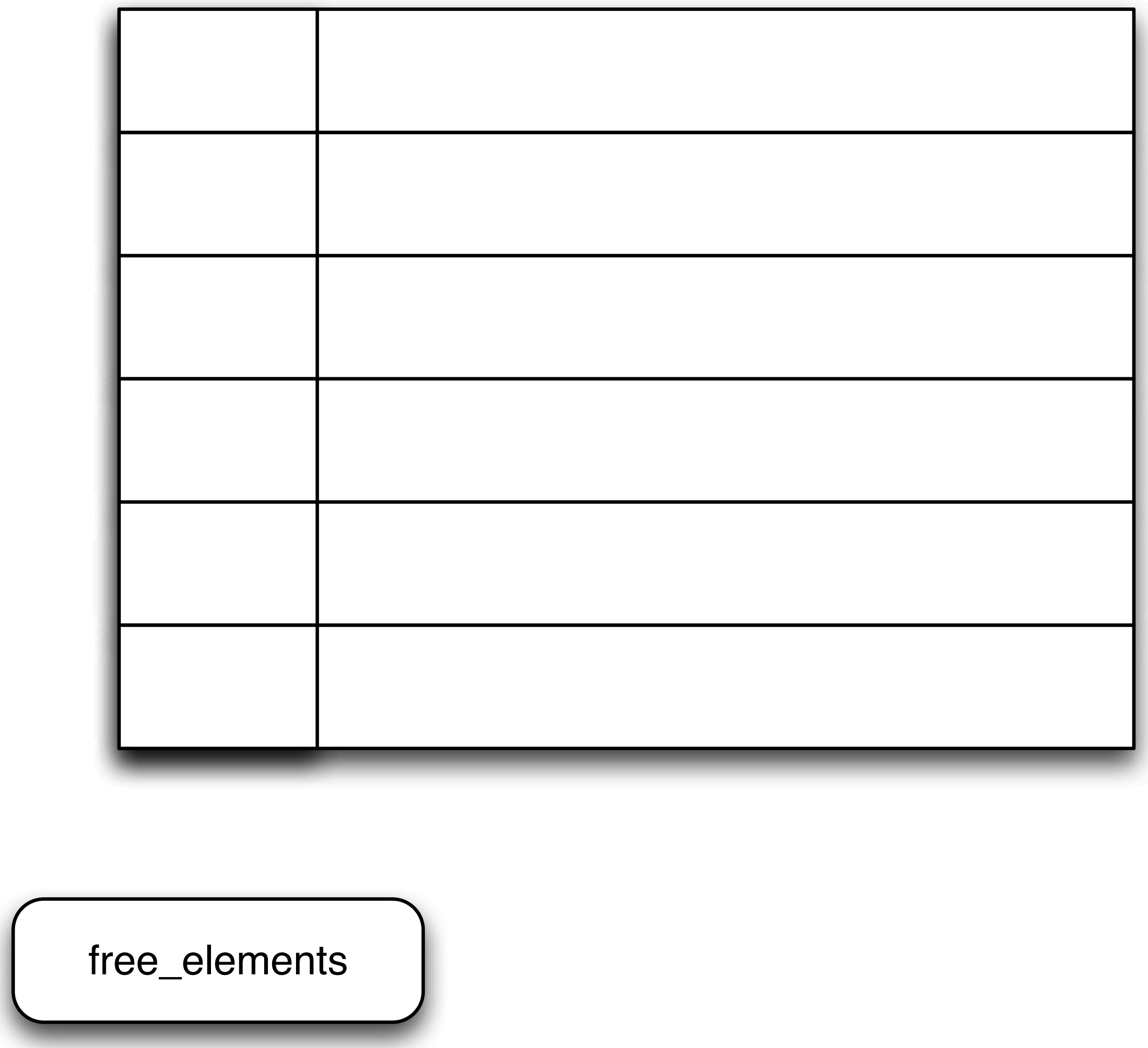


iOS 9 moves the page meta data to the beginning of the page

possible reasoning: protect meta data against overflows

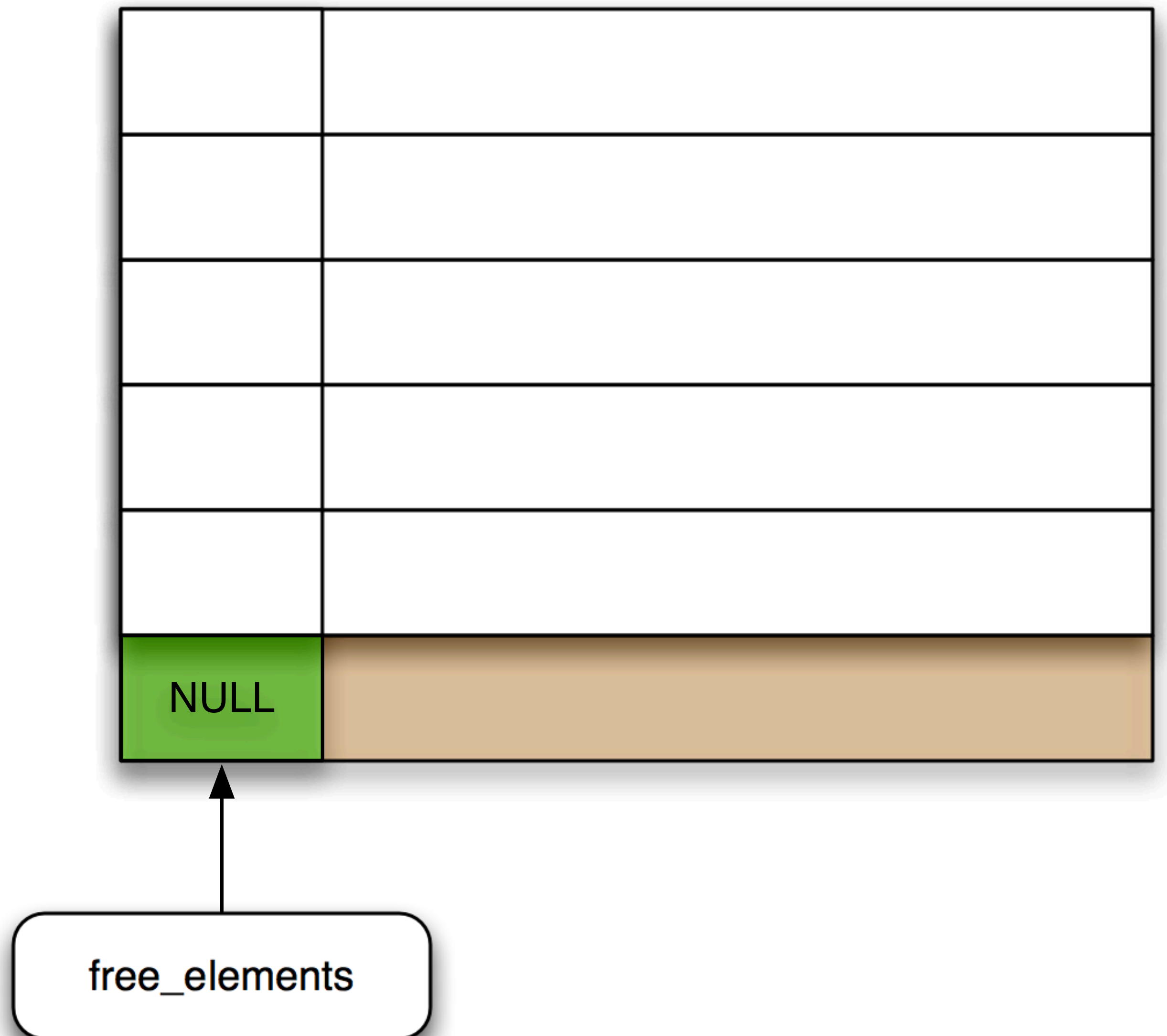
iOS 9.2 Initial Freelist Randomization (I)

- when new memory is added to a zone elements within are freed one by one into the freelist
- starting with iOS 9.2 the elements are no longer added linear from first to last
- instead every time a next element is added a random decision is made if the first or last element should be added



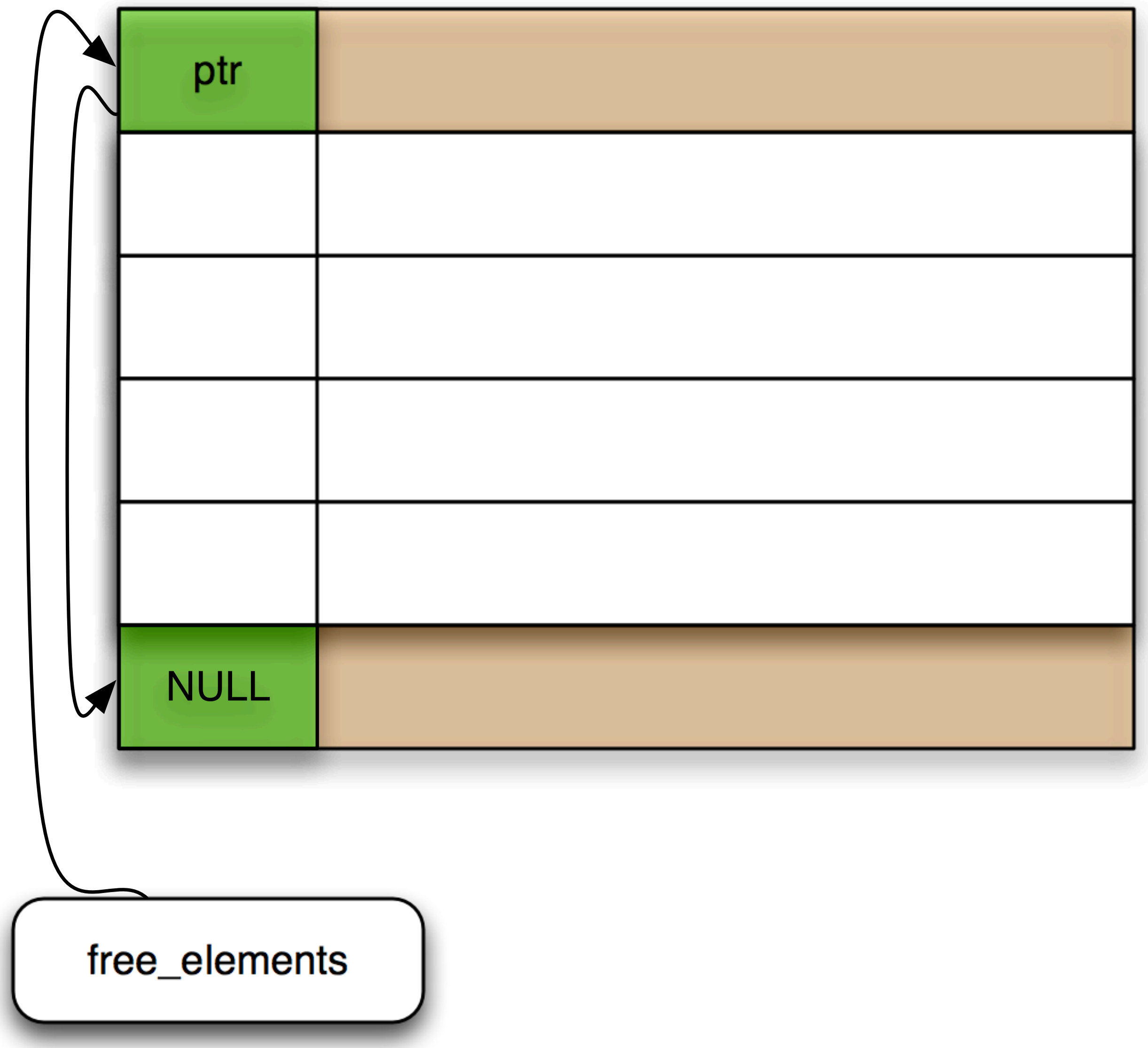
iOS 9.2 Initial Freelist Randomization (II)

- when new memory is added to a zone elements within are freed one by one into the freelist
- starting with iOS 9.2 the elements are no longer added linear from first to last
- instead every time a next element is added a random decision is made if the first or last element should be added



iOS 9.2 Initial Freelist Randomization (III)

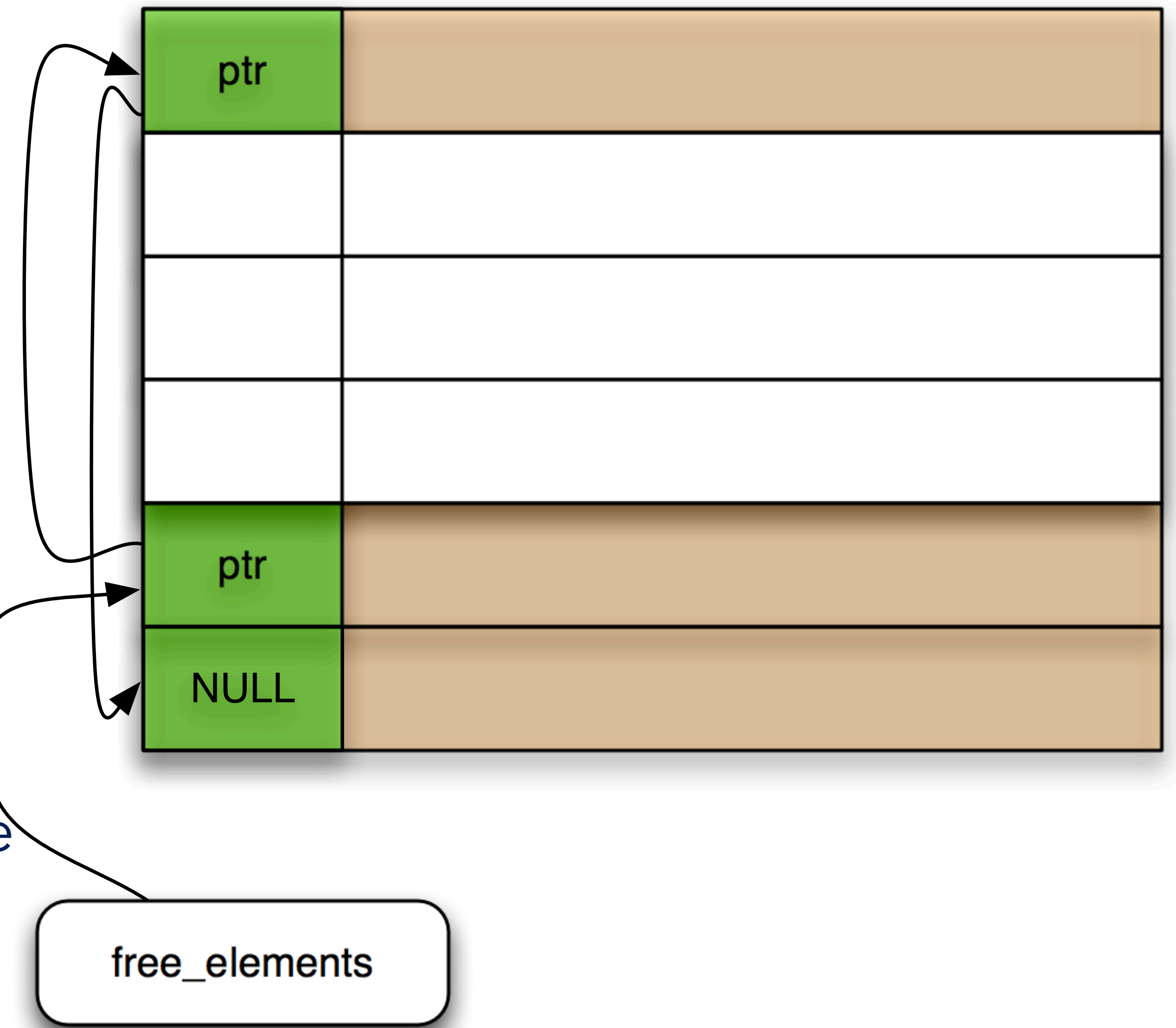
- when new memory is added to a zone elements within are freed one by one into the freelist
- starting with iOS 9.2 the elements are no longer added linear from first to last
- instead every time a next element is added a random decision is made if the first or last element should be added



SektionEins

iOS 9.2 Initial Freelist Randomization (IV)

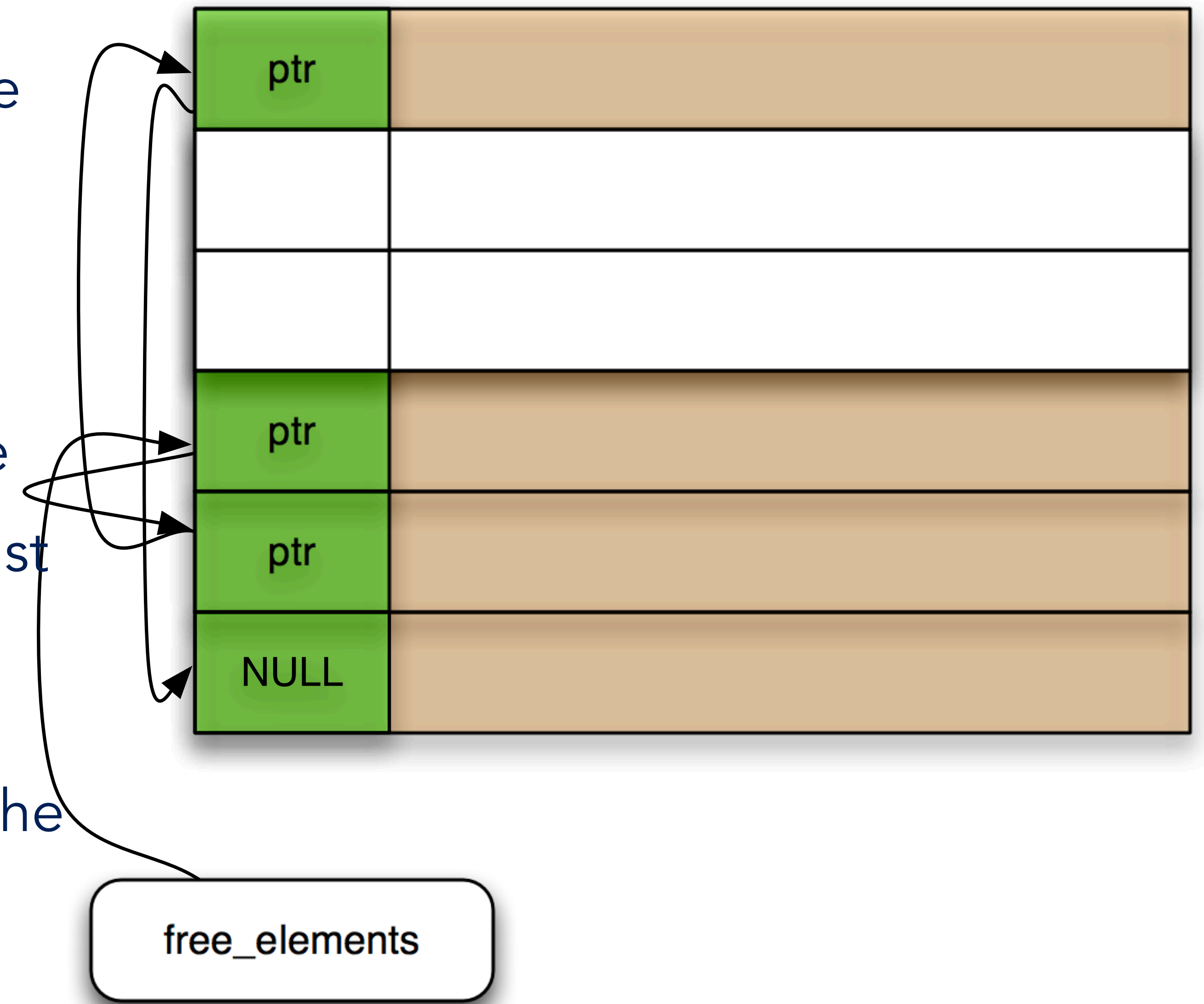
- when new memory is added to a zone elements within are freed one by one into the freelist
- starting with iOS 9.2 the elements are no longer added linear from first to last
- instead every time a next element is added a random decision is made if the first or last element should be added



SektionEins

iOS 9.2 Initial Freelist Randomization (V)

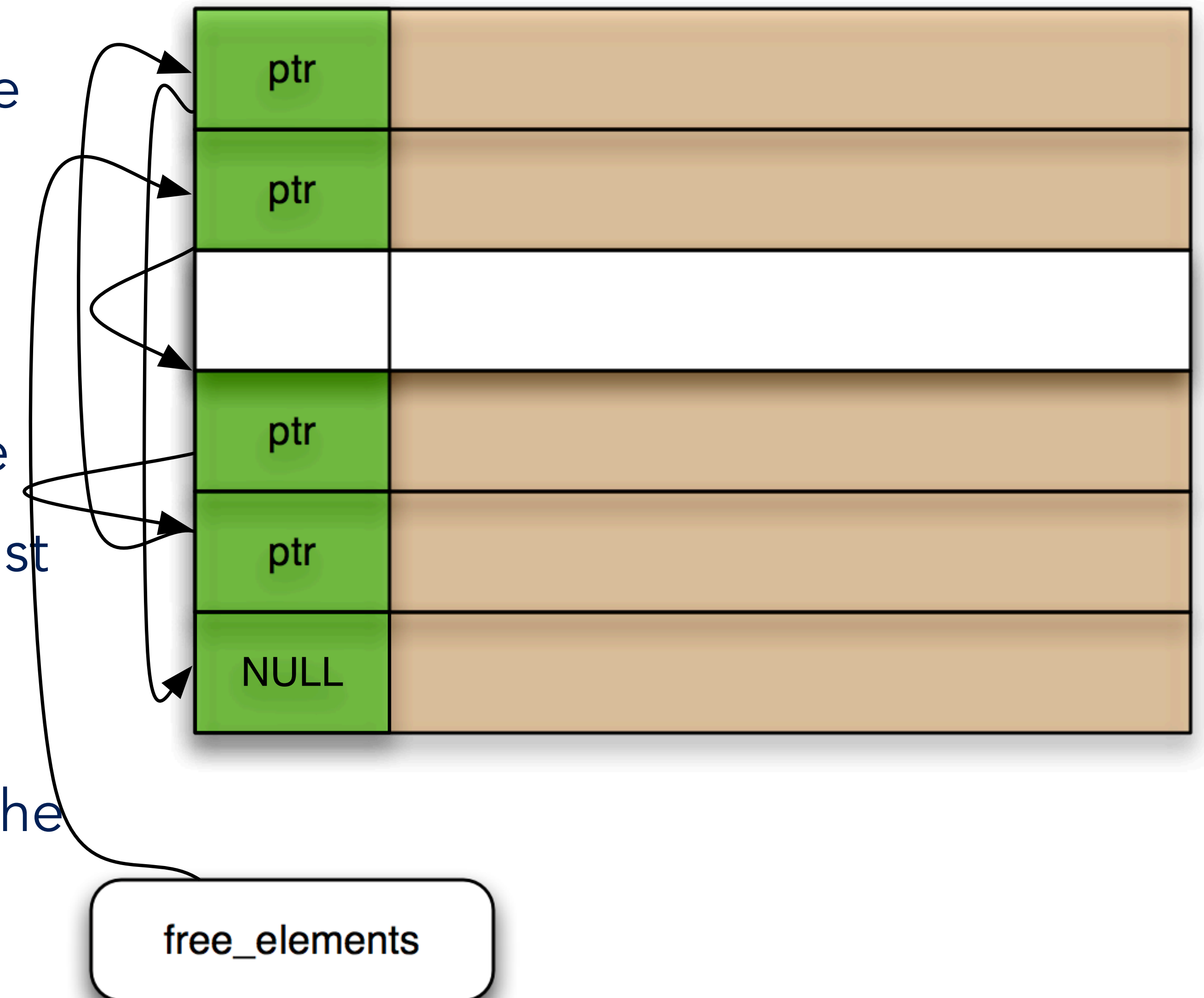
- when new memory is added to a zone elements within are freed one by one into the freelist
- starting with iOS 9.2 the elements are no longer added linear from first to last
- instead every time a next element is added a random decision is made if the first or last element should be added



SektionEins

iOS 9.2 Initial Freelist Randomization (VI)

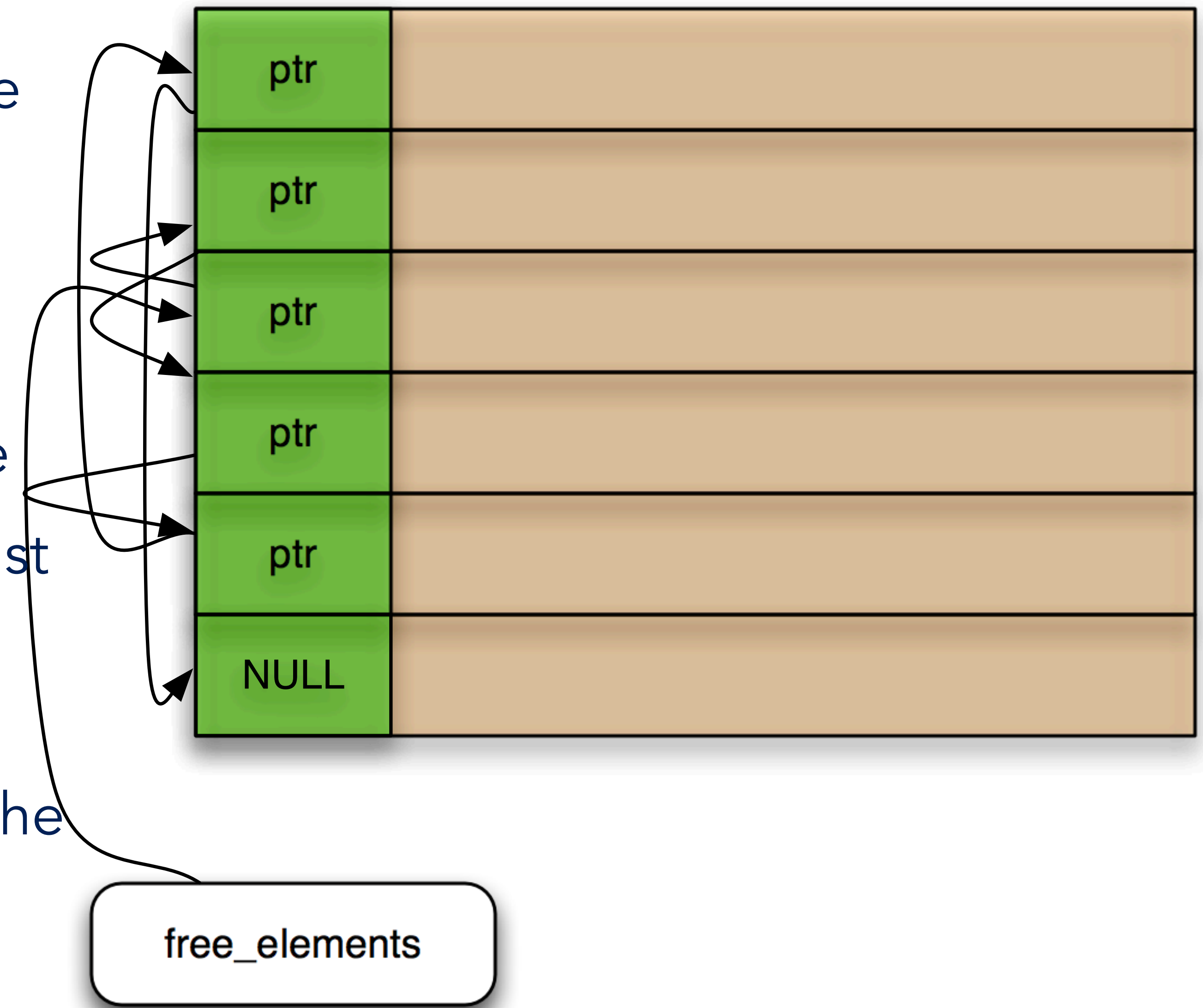
- when new memory is added to a zone elements within are freed one by one into the freelist
- starting with iOS 9.2 the elements are no longer added linear from first to last
- instead every time a next element is added a random decision is made if the first or last element should be added



SektionEins

iOS 9.2 Initial Freelist Randomization (VII)

- when new memory is added to a zone elements within are freed one by one into the freelist
- starting with iOS 9.2 the elements are no longer added linear from first to last
- instead every time a next element is added a random decision is made if the first or last element should be added



How attackers abused the iOS 9 Zone Allocator

- **Heap-Feng-Shui**
    - same methods as before were used
      (vm_map_copy_t after "hardening" still usable for heap-feng-shui)
- **Corruption Targets**
    - targeting **vm_map_copy_t** still allows zone confusion and limited info leaks
      (only larger info leaks and arbitrary pointer info leaks stopped by "hardening")
    - all other previously targeted areas still work

Part III

Upcoming changes to the iOS kernel heap in iOS 10

Changes in upcoming iOS 10

- more **vm_map_copy_t** “hardening”
- fixed zone structure array
- zone page metadata completely revamped
- page freelist pointer leak protection
- **zalloc()** wrappers add no inbound metadata anymore
- zone allocator debugging features revamped

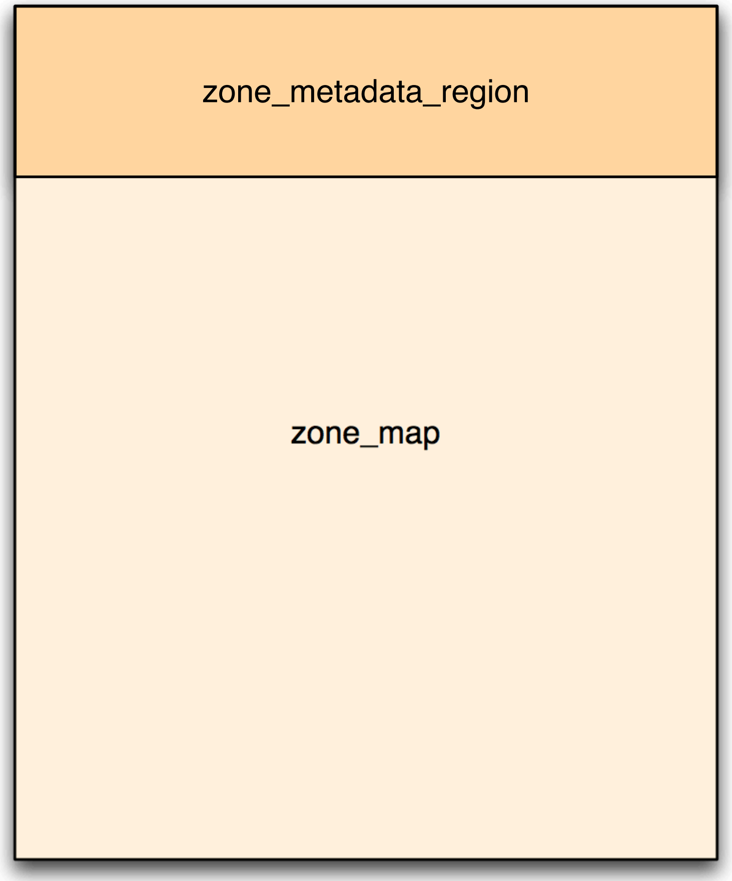
iOS 10 Fixed Zonestructure Array

- zone structures are no longer allocated on zone allocator heap
- instead stored in fixed **zone_array** in *__DATA::__bss*
- seems to be a bad idea because address of zone structure is now fixed (relative to kernelbase)

```
__DATA:__bss:FFFFFFF007552170                 public zone_metadata_region_min
__DATA:__bss:FFFFFFF007552170 ; vm_offset_t zone_metadata_region_min
__DATA:__bss:FFFFFFF007552170 zone_metadata_region_min dq 0         ; DATA XREF: zone_element_size+19r
__DATA:__bss:FFFFFFF007552170                                       ; get_zone_page_metadata+2Fr ...
__DATA:__bss:FFFFFFF007552178                 public zone_metadata_region_max
__DATA:__bss:FFFFFFF007552178 ; vm_offset_t zone_metadata_region_max
__DATA:__bss:FFFFFFF007552178 zone_metadata_region_max dq 0         ; DATA XREF: zone_init+F3w
__DATA:__bss:FFFFFFF007552180                 public zone_array
__DATA:__bss:FFFFFFF007552180 ; zone zone_array[256]
__DATA:__bss:FFFFFFF007552180 zone_array      zone 100h dup(<0>)    ; DATA XREF: panic_display_zprint+4Bo
__DATA:__bss:FFFFFFF007552180                                       ; zone_element_size+5Do ...
```

iOS 10 new zone_metadata_region

- in iOS 10 **ALL** zones make use of page metadata
- a new **zone_metadata_region** is utilised for that
- region is reserved in the **zone_map**
- used to keep meta data for every single page in the **zone_map**



*elements from outside zone_map store meta data at beginning of page

iOS 10 page metadata

- **zindex** index of zone in zone_array (instead of back pointer)
- **page_count** number of pages in allocation size
- **free_count** number of free elements in page
- **freelist_offset** offset of first free element starting from this page's address
- **real_metadata_offset** offset of this secondary meta data from page 0 metadata

real_metadata (page 0)

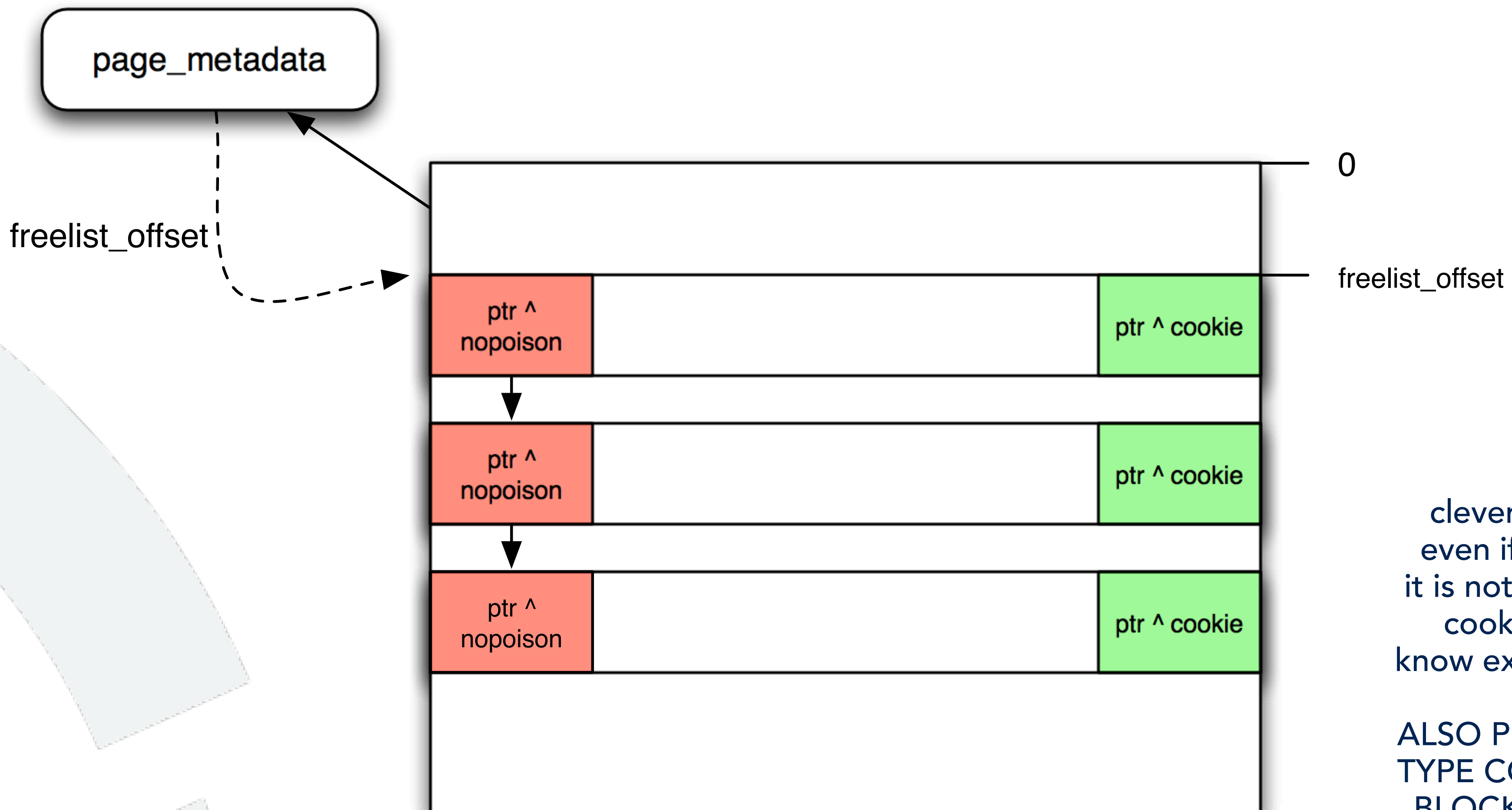
0x00	pages.next
0x08	pages.prev
0x10	freelist_offset
0x14	free_count
0x16	zindex
0x17	page_count
0x18	

metadata for additional pages

0x00	0
0x08	0
0x10	real_metadata_offset
0x14	0
0x16	zindex=0xFF
0x17	0
0x18	

iOS 10 page freelists

- page metadata does not contain pointer to **head of freelist** anymore
- instead **freelist_offset** determines byte offset of first free element in page
- **ptr** to next free block is always XORed against **nopoison_cookie**
- **backup ptr** is ptr value XORed against selected cookie *(noposion/poisened)*



clever change from before
even if whole memory leaks
it is not possible to determine
cookie values unless you
know exact address of next ptr

ALSO PROTECTION AGAINST
TYPE CONFUSION OF EMPTY
BLOCKS AS IOKIT OBJECTS

iOS 10 Metadata vs. Wrong Zone Frees

- every zone uses the new metadata
- this allows determining correct zone by element
- freeing elements into the wrong zone can be reliably detected
- when this happens **zfree()** panics the kernel
- abuse seems not possible anymore

iOS 10 Wrappers and Metadata

- new meta data can be used to determine size of elements on heap
- heap allocation wrappers like **MALLOC**() and **kern_os_malloc**() no longer need to store the size
- both become simpler wrappers around **kalloc**() and are therefore inlined in several places
- allocations are now only application data
  (more codepaths can now be used for heap layout control)

iOS 10 Kernel Heap Allocation Debugging (I)

- kernel heap allocation logging was revamped
- no longer limited to a single zone
- code now limited to logging in max 5 zones
- zones selected by new kernel boot args

```
zlog1, zlog2, zlog3, zlog4, zlog5

usage:  -zc zlog1=kalloc.128 zlog2=kalloc.256
```

iOS 10 Kernel Heap Allocation Debugging (II)

- bitfield in zone struct contains new bit 15 to select if **zone_logging**
- zone struct contains **zlog_btlog** variable instead of global variable
- **btlog_t** structure heavily modified
- log is no longer a simple list but a hash table

```
  unsigned __int32 gzalloc_exempt : 1;
  unsigned __int32 alignment_required : 1;
  unsigned __int32 zone_logging : 1;
  unsigned __int32 zone_replenishing : 1;
  unsigned __int32 _reserved : 15;
  int index;
  const char *zone_name;
  uint32_t_0 zleak_capture;
  uint32_t_0 zp_count;
  vm_size_t prio_refill_watermark;
  thread_t zone_replenish_thread;
  gzalloc_data_t gz;
  btlog_t *zlog_btlog;
};
```

How attackers will abuse the iOS 10 Zone Allocator

- **Heap-Feng-Shui**
  - for simple feng-shui all previous methods should still be usable
    (beside continued hardening of vm_map_copy_t)
  - code paths that use MALLOC() are now also usable for heap-feng-shui
    (e.g. posix_spawn())
- **Corruption Targets**
  - application data like object and mach port pointers very interesting
  - **BUT** size field corruption to confuse free into wrong zone will trigger **panic()**

Time for questions…

SektionEins

Questions…?