Lexfo

ICONV, set the charset to RCE

EXPLOITING THE GLIBC TO HACK THE PHP ENGINE

Charles Fol
@cfreal_



DEFCON 32 ENGAGE

Summary

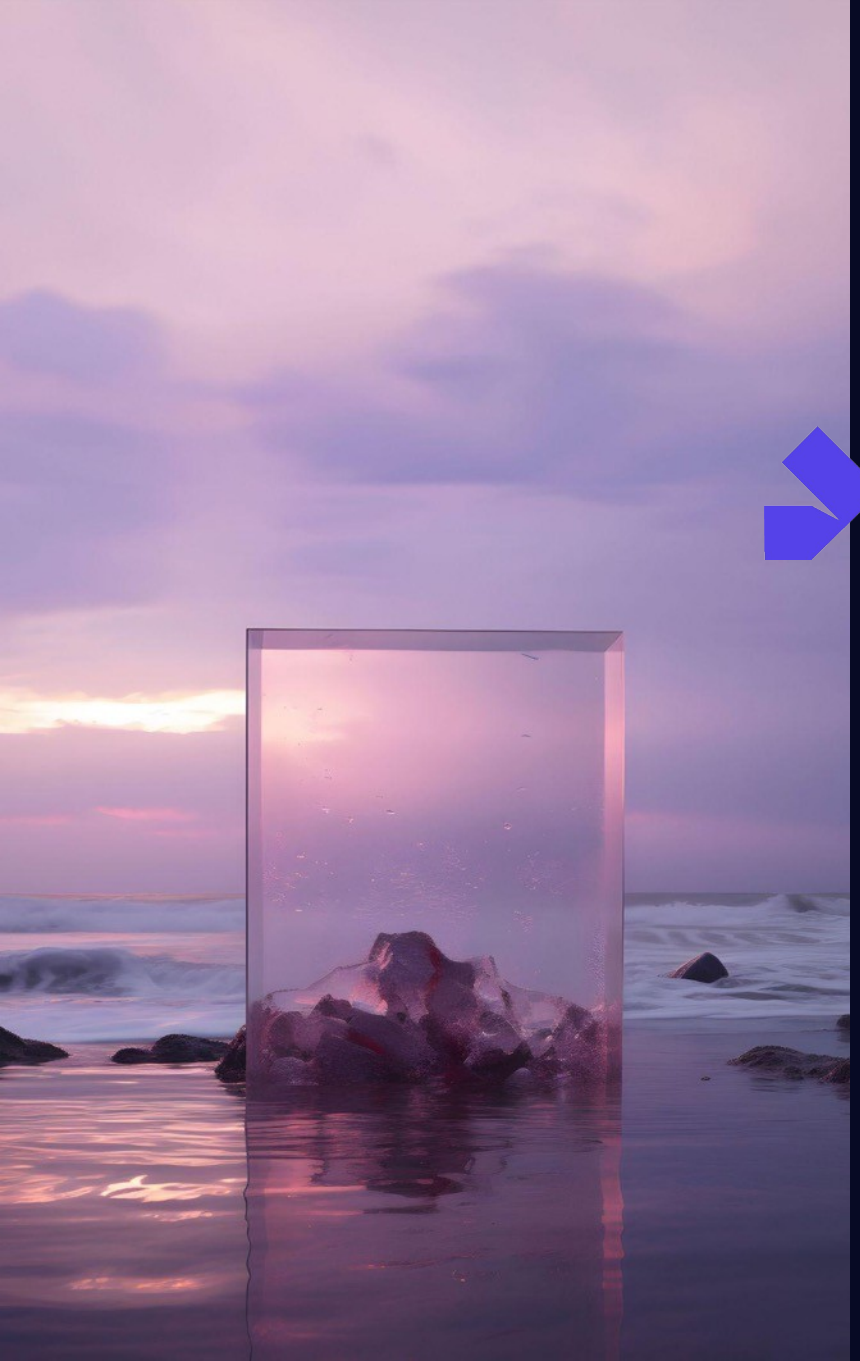
Discovery of CVE-2024-2961

Attacking PHP filters

Attacking direct iconv() calls

Conclusion

Lexfo



Discovery

Lexfo

PHP file read primitive

What can you do with a file read primitive?

```php
echo file_get_contents($_GET['file']);
```




PHP file read primitive

What can you do with a file read primitive?

```php
echo file_get_contents($_GET['file']);
```

```php
echo file_get_contents('/etc/passwd');
```




PHP file read primitive

What can you do with a file read primitive?

```php
echo file_get_contents($_GET['file']);
```

```php
echo file_get_contents('/etc/passwd');
```

```php
echo file_get_contents('http://ambionics.io/blog');
```




PHP file read primitive

What can you do with a file read primitive?

```php
echo file_get_contents($_GET['file']);
```

```php
echo file_get_contents('/etc/passwd');
```

```php
echo file_get_contents('http://ambionics.io/blog');
```

```php
echo file_get_contents('ftp://ftp.ambionics.io/file.bin');
```




PHAR : a dangerous protocol

```
phar:///var/www/archive.phar/file.txt
```

PHAR: PHP archive

- Source files
- Resources
- Serialized metadata
  - **Deserialized** when reading an archive



PHAR : File read to RCE

Harder to exploit

- PHAR:
  - PHP 8+: no more deserialisation of the metadata
  - phar:// might be disabled (Drupal, Magento)

- Deserialisation:
  - Gadget chains are being patched more and more
  - Types are making a comeback



PHP filters: applying transforms to the stream

```
php://filter/[filters]/resource=[resource]
```

`[resource]`	original stream: file, etc.
`[filters]`	list of filters to apply to the stream



PHP filters: applying transforms to the stream

```
php://filter/convert.base64-encode/resource=/etc/passwd
```





PHP filters: applying transforms to the stream

```
php://filter/convert.base64-encode|convert.base64-encode/resource= ...
```





PHP filters: lots of options

Many available filters

- string.upper *Convert stream to uppercase*
- string.lower *Convert stream to lowercase*
- string.rot13 *Do some BC crypto*
- dechunk *Decode chunked encoding*
- convert.iconv.X.Y *Convert from charset X to charset Y*



PHP filters: applying transforms to the stream

```
php://filter/convert.iconv.UTF-8.UTF-16/resource=/etc/passwd
```

```
00000000: fffe 7200 6f00 6f00 7400 3a00 7800 3a00  ..r.o.o.t.:.x.:.
00000010: 3000 3a00 3000 3a00 7200 6f00 6f00 7400  0.:.0.:.r.o.o.t.
00000020: 3a00 2f00 7200 6f00 6f00 7400 3a00 2f00  :./.r.o.o.t.:./.
00000030: 6200 6900 6e00 2f00 6100 7300 6800 0a00  b.i.n./.a.s.h...
00000040: 6200 6900 6e00 3a00 7800 3a00 3100 3a00  b.i.n.:.x.:.1.:.
00000050: 3100 3a00 6200 6900 6e00 3a00 2f00 6200  1.:.b.i.n.:./.b.
00000060: 6900 6e00 3a00 2f00 7300 6200 6900 6e00  i.n.:./.s.b.i.n.
00000070: 2f00 6e00 6f00 6c00 6f00 6700 6900 6e00  /.n.o.l.o.g.i.n.
00000080: 0a00 6400 6100 6500 6d00 6f00 6e00 3a00  ..d.a.e.m.o.n.:.
00000090: 7800 3a00 3200 3a00 3200 3a00 6400 6100  x.:.2.:.2.:.d.a.
000000a0: 6500 6d00 6f00 6e00 3a00 2f00 7300 6200  e.m.o.n.:./.s.b.
000000b0: 6900 6e00 3a00 2f00 7300 6200 6900 6e00  i.n.:./.s.b.i.n.
000000c0: 2f00 6e00 6f00 6c00 6f00 6700 6900 6e00  /.n.o.l.o.g.i.n.
000000d0: 0a00 6100 6400 6d00 3a00 7800 3a00 3300  ..a.d.m.:.x.:.3.
000000e0: 3a00 3400 3a00 6100 6400 6d00 3a00 2f00  :.4.:.a.d.m.:./.
000000f0: 7600 6100 7200 2f00 6100 6400 6d00 3a00  v.a.r./.a.d.m.:.
00000100: 2f00 7300 6200 6900 6e00 2f00 6e00 6f00  /.s.b.i.n./.n.o.
00000110: 6c00 6f00 6700 6900 6e00 0a00 6c00 7000  l.o.g.i.n...l.p.
00000120: 3a00 7800 3a00 3400 3a00 3700 3a00 6c00  :.x.:.4.:.7.:.l.
00000130: 7000 3a00 2f00 7600 6100 7200 2f00 7300  p.:./.v.a.r./.s.
00000140: 7000 6f00 6f00 6c00 2f00 6c00 7000 6400  p.o.o.l./.l.p.d.
```




PHP filters: arbitrary suffix

Arbitrary prefix and suffix? [1]

```php
$json = file_get_contents($_GET['file']);
$json = json_decode($json);

echo $json['message']['title'];
```

[1] https://www.ambionics.io/blog/wrapwrap-php-filters-suffix



PHP filters: finding the bug

Bruteforcing conversions

```
php://filter/convert.iconv.X.Y/resource=data:,test123
```

```
segmentation fault (core dumped)
```




A crash

Woops, a crash

iconv: glibc's character set conversion library

```
iconv_t iconv_open(const char *tocode, const char *fromcode);
size_t iconv(iconv_t cd,
             char **inbuf, size_t *inbytesleft,
             char **outbuf, size_t *outbytesleft);
```

Processes at most **inbytesleft** bytes from **inbuf** and writes at most **outbytesleft** bytes to **outbuf***



* in theory

The vulnerability: OOB write

glibc-2.39/iconvdata/iso-2022-cn-ext.c

```c
if ((used & SO_mask) != 0 && (ann & SO_ann) != (used << 8))
{
    if (outptr + 4 > outend)
    {
        result = __GCONV_FULL_OUTPUT;
        break;
    }

    assert(used >= 1 && used <= 4);
    escseq = ")A\0\0)G)E" + (used - 1) * 2;
    *outptr++ = ESC;
    *outptr++ = '$';
    *outptr++ = *escseq++;
    *outptr++ = *escseq++;

    ann = (ann & ~SO_ann) | (used << 8);
}
else if ((used & SS2_mask) != 0 && (ann & SS2_ann) != (used << 8))
{
    assert(used == CNS11643_2_set); /* XXX */
    escseq = "*H";
    *outptr++ = ESC;
    *outptr++ = '$';
    *outptr++ = *escseq++;
    *outptr++ = *escseq++;

    ann = (ann & ~SS2_ann) | (used << 8);
}
else if ((used & SS3_mask) != 0 && (ann & SS3_ann) != (used << 8))
{
    assert((used >> 5) >= 3 && (used >> 5) <= 7);
    escseq = "+I+J+K+L+M" + ((used >> 5) - 3) * 2;
    *outptr++ = ESC;
    *outptr++ = '$';
    *outptr++ = *escseq++;
    *outptr++ = *escseq++;

    ann = (ann & ~SS3_ann) | (used << 8);
}
```

When converting to ISO-2022-CN-EXT, the library fails to **check the bounds** before writing **escape sequences** to the output buffer

Overflow of 1 to 3 bytes:

```
$*H 24 2A 48
$+I 24 2B 49
$+J 24 2B 4A
$+K 24 2B 4B
$+L 24 2B 4C
$+M 24 2B 4D
```




The vulnerability: OOB write

glibc-2.39/iconvdata/iso-2022-cn-ext.c

```c
if ((used & SO_mask) != 0 && (ann & SO_ann) != (used << 8))
{
    if (outptr + 4 > outend)
    {
        result = __GCONV_FULL_OUTPUT;
        break;
    }

    assert(used >= 1 && used <= 4);
    escseq = ")A\0\0)G)E" + (used - 1) * 2;
    *outptr++ = ESC;
    *outptr++ = '$';
    *outptr++ = *escseq++;
    *outptr++ = *escseq++;

    ann = (ann & ~SO_ann) | (used << 8);
}
else if ((used & SS2_mask) != 0 && (ann & SS2_ann) != (used << 8))
{
    assert(used == CNS11643_2_set); /* XXX */
    escseq = "*H";
    *outptr++ = ESC;
    *outptr++ = '$';
    *outptr++ = *escseq++;
    *outptr++ = *escseq++;

    ann = (ann & ~SS2_ann) | (used << 8);
}
else if ((used & SS3_mask) != 0 && (ann & SS3_ann) != (used << 8))
{
    assert((used >> 5) >= 3 && (used >> 5) <= 7);
    escseq = "+I+J+K+L+M" + ((used >> 5) - 3) * 2;
    *outptr++ = ESC;
    *outptr++ = '$';
    *outptr++ = *escseq++;
    *outptr++ = *escseq++;

    ann = (ann & ~SS3_ann) | (used << 8);
}
```

CVE-2024-2961

AAAAAA 劄
BBBBBB 砐
CCCCCC 岍
DDDDDD 湿



Tedious conditions

Vulnerability

- 1, 2 or 3 bytes overflow
- **fixed** byte values

Conditions

- Control the **output charset** (ISO-2022-CN-EXT)
- Control **part of the input buffer**
- Have a **suitable output buffer**





Attacking PHP filters

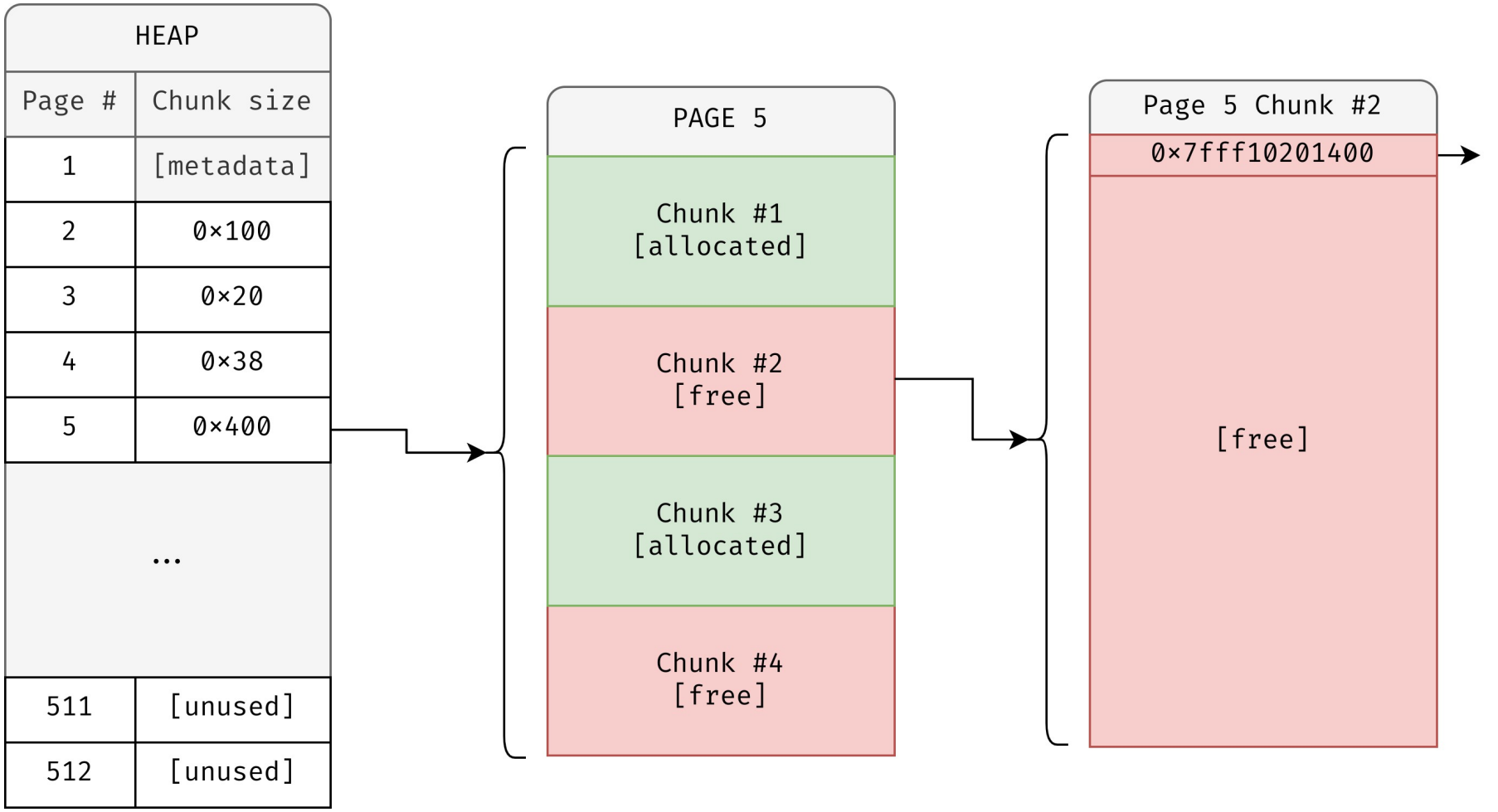
Lexfo

PHP's heap

- PHP's heap is page-based
  - 512 pages of 0x1000 bytes
- Each page contains chunks of the same size
  - Page 10: chunks of size 0x100
  - Page 11: chunks of size 0x30
  - Page 12: chunks of size 0xa00
- PHP holds a singly-linked free list for each chunk size
  - Last In, First Out (LIFO)
  - Similar to libc's tcache, but unlimited



PHP's heap

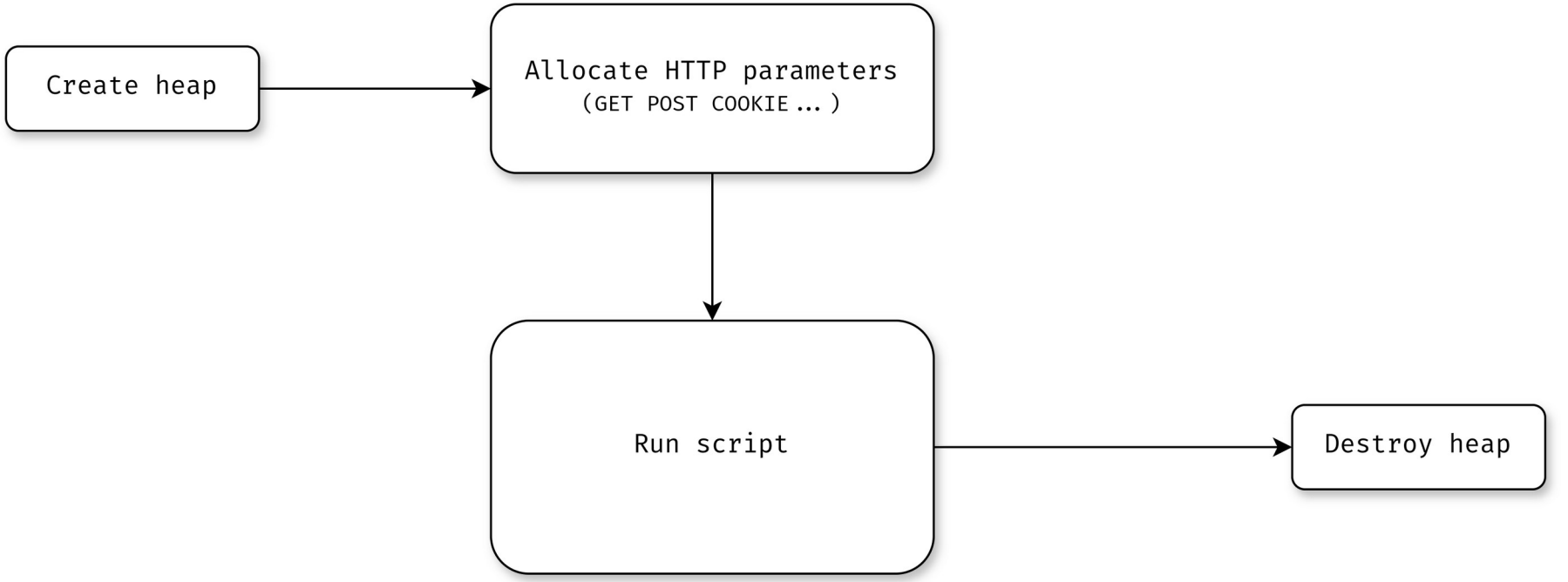

HEAP
Page #
Chunk size
1
[metadata]
2
0×100
3
0×20
4
0×38
5
0×400
...
511
[unused]
512
[unused]
PAGE 5
Chunk #1
[allocated]
Chunk #2
[free]
Chunk #3
[allocated]
Chunk #4
[free]
Page 5 Chunk #2
0×7fff10201400
[free]



Request lifecycle

1 heap for 1 request

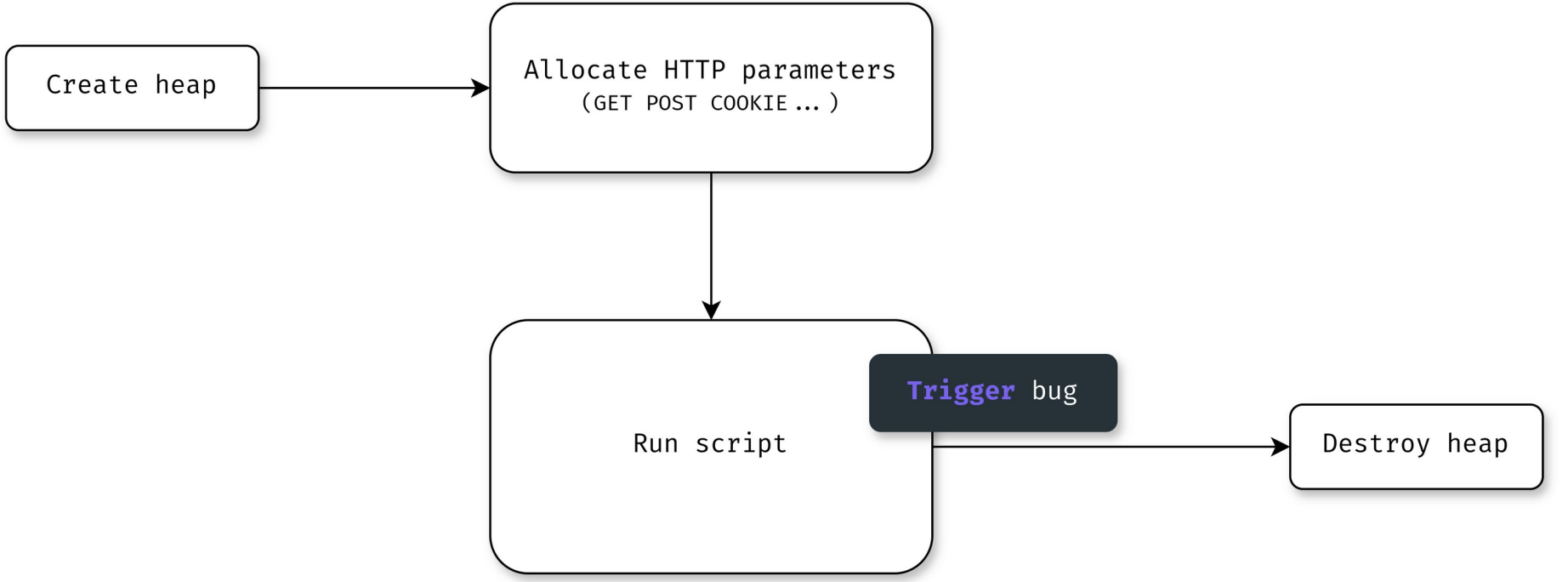
(bugs do not linger)



Request lifecycle

Triggering the **bug**

(finding a vulnerable code path)

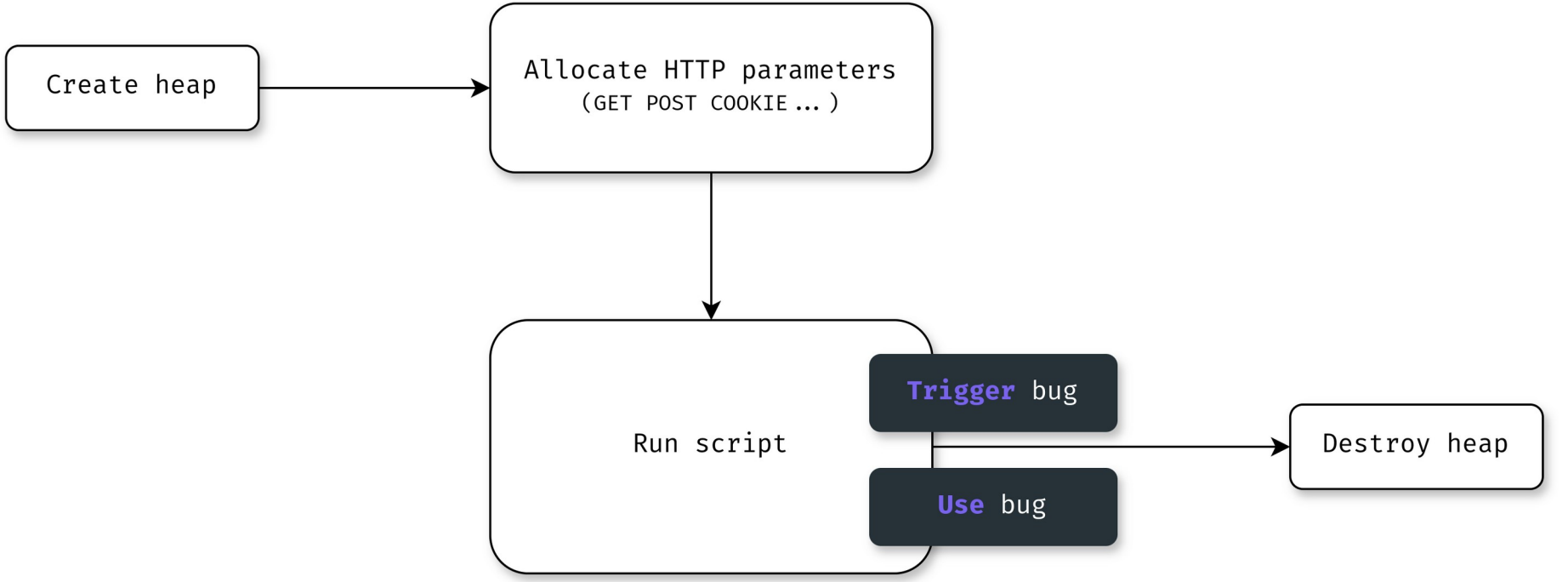




Request lifecycle

Using the **bug**

(in the same request)





Handling of PHP filters

```
php://filter/[filters]/resource=[resource]
```

PHP processes a php://filter
in 2 steps

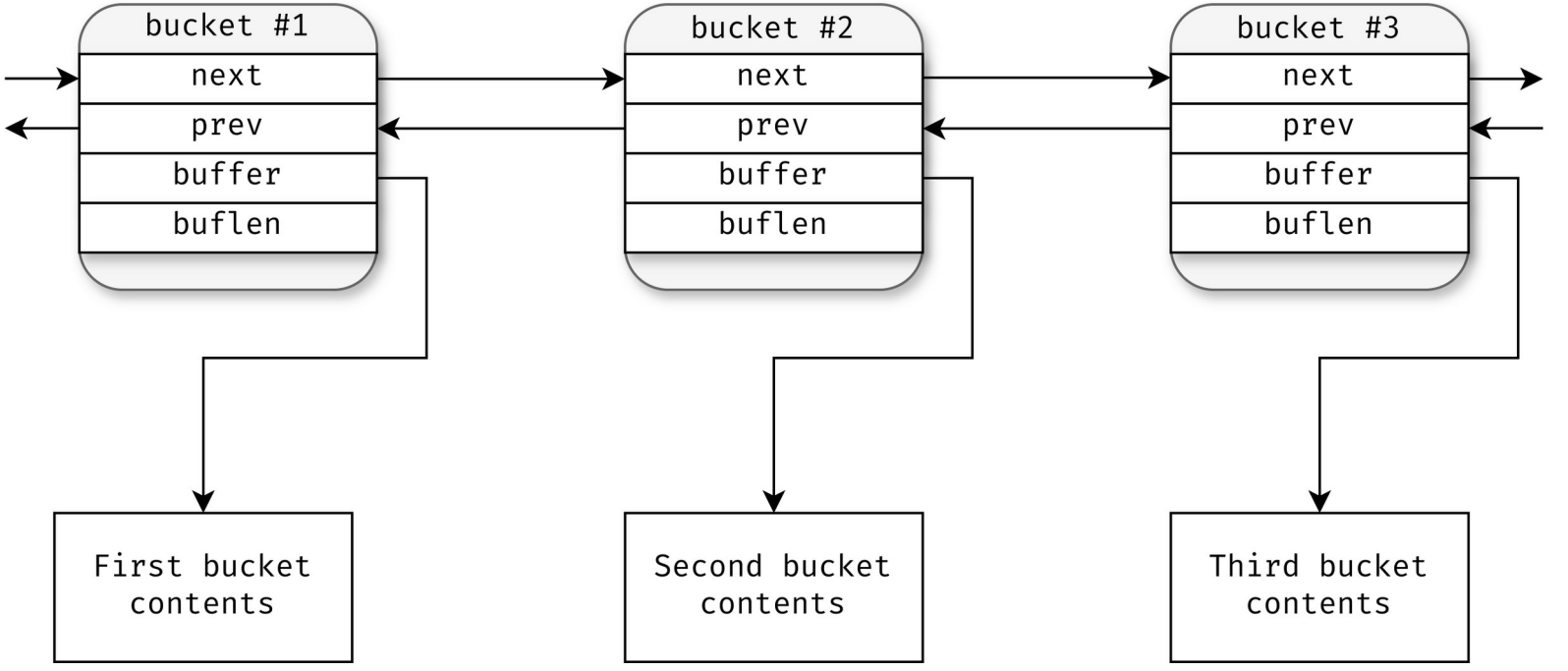
1. Read the resource as a stream
2. Apply filters



Handling of PHP filters

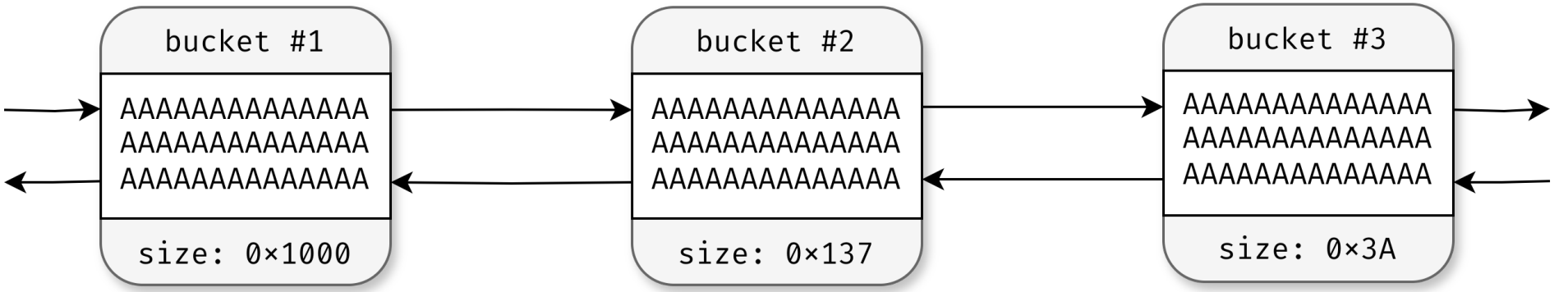
```
php://filter/[filters]/resource=[resource]
```

1. Fetching the stream



Handling of PHP filters

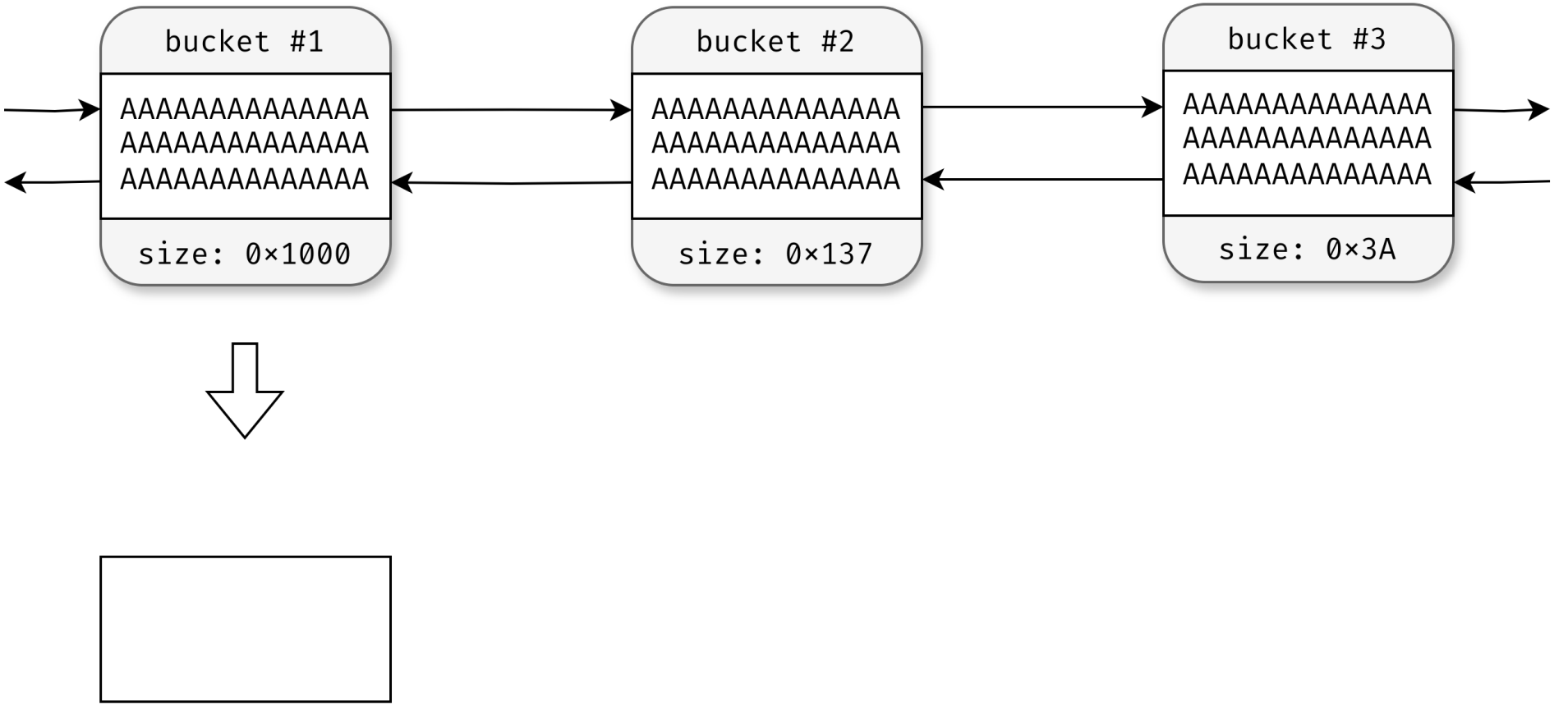
2. Applying filters





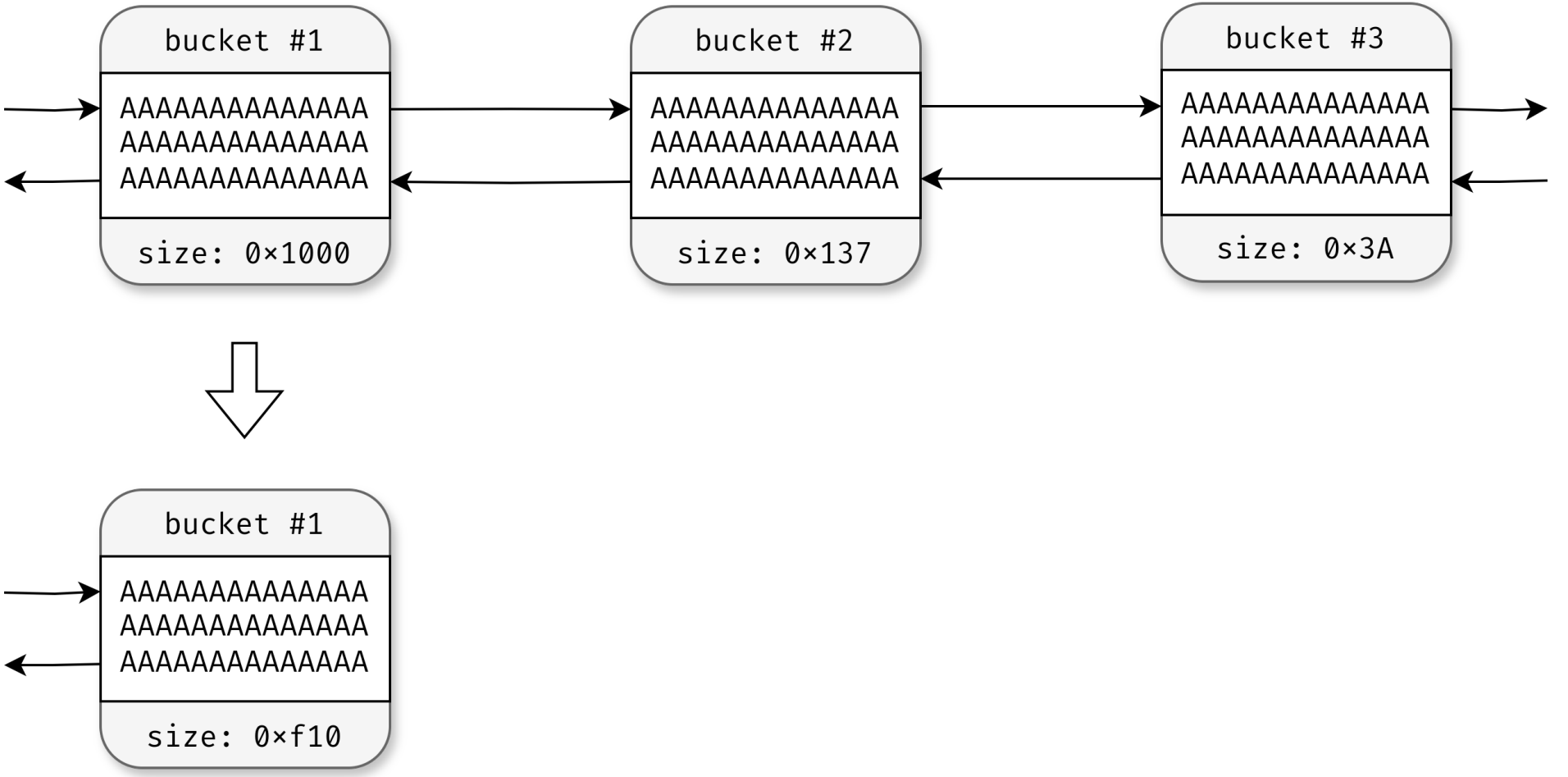
Handling of PHP filters

3. Applying filters



Handling of PHP filters

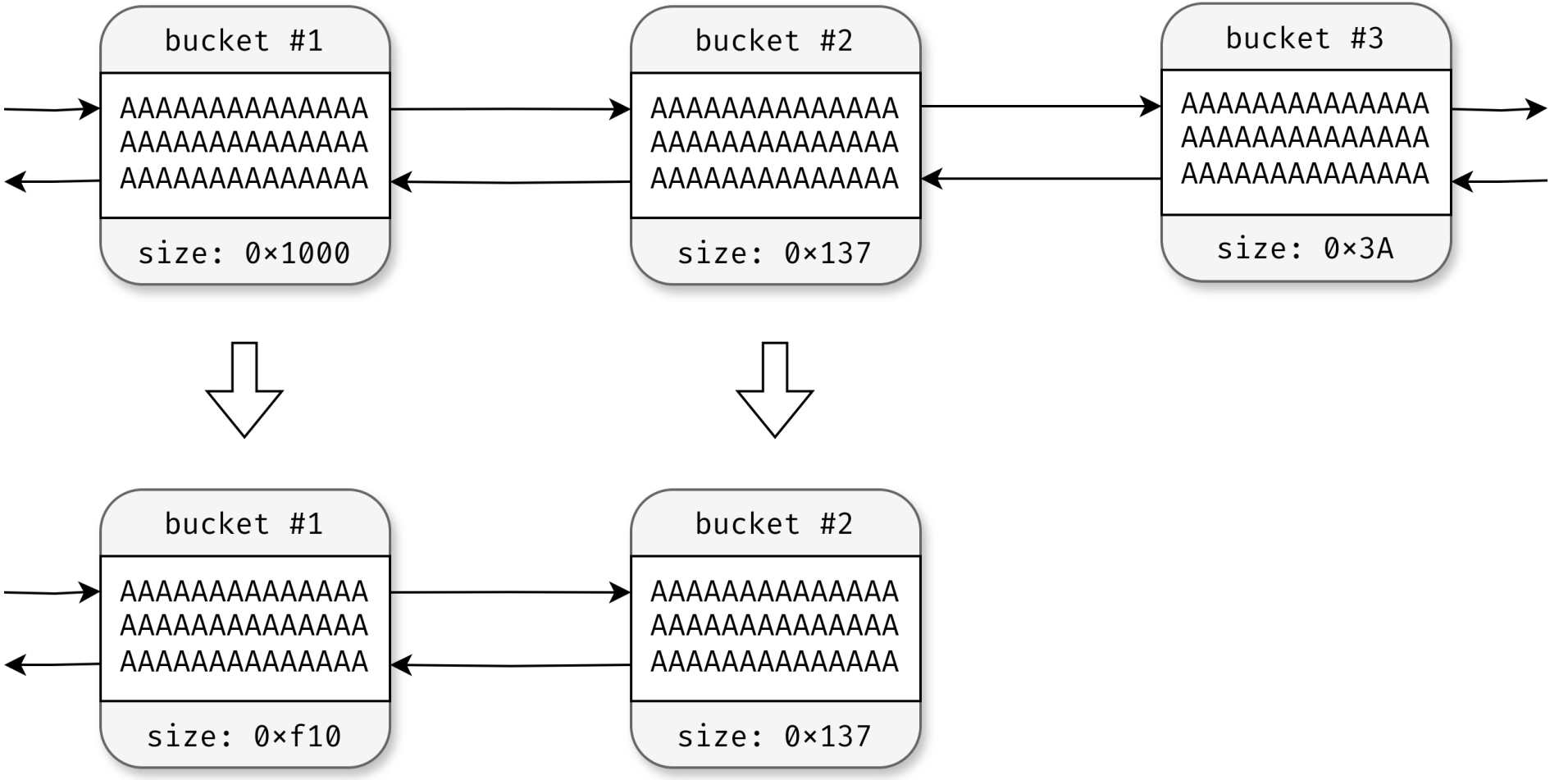
3. Applying filters





Handling of PHP filters

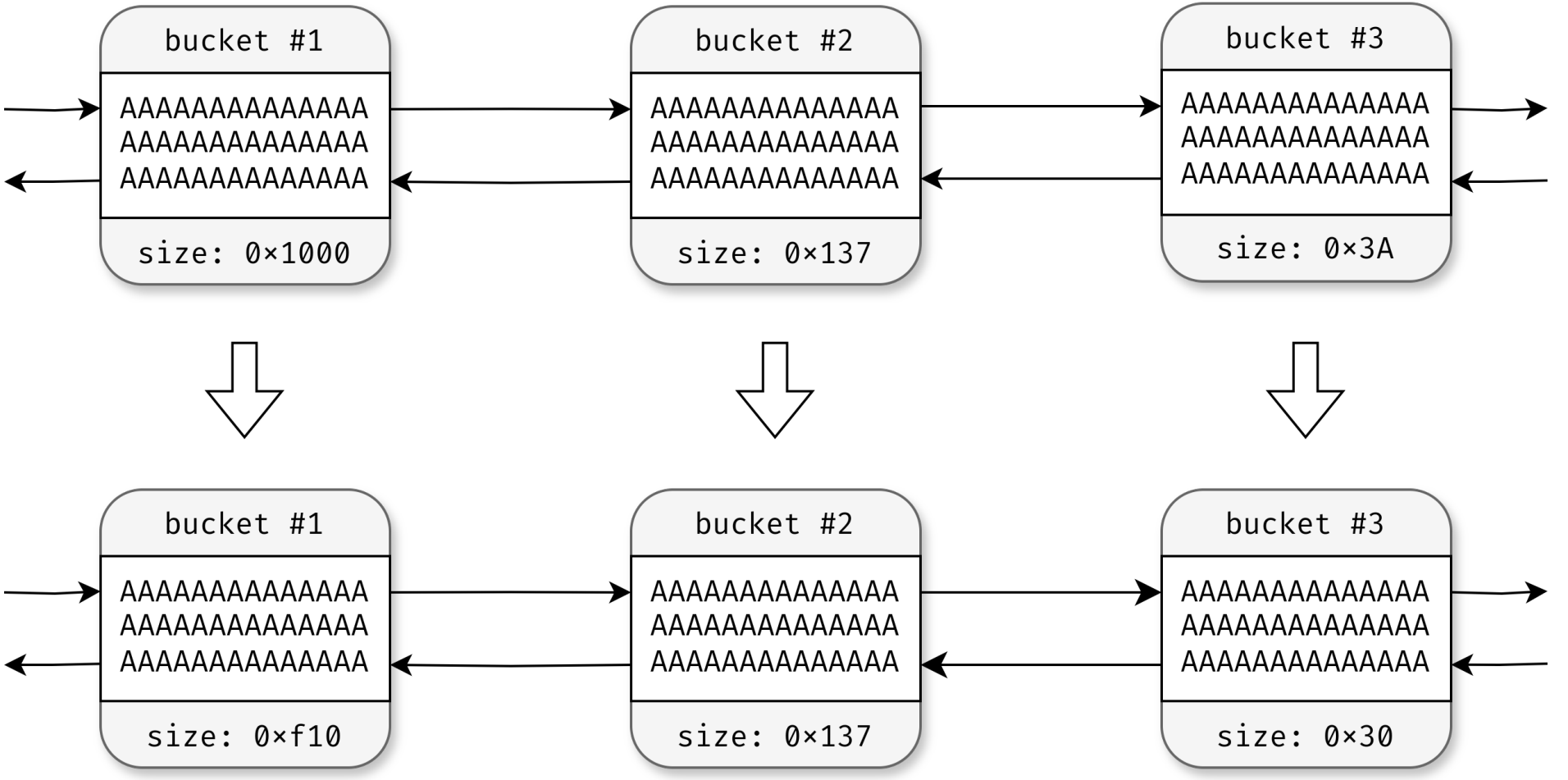
3. Applying filters





Handling of PHP filters

3. Applying filters





Building an exploit: file read

```php
echo file_get_contents($_GET['file']);
```

Not too hard ?

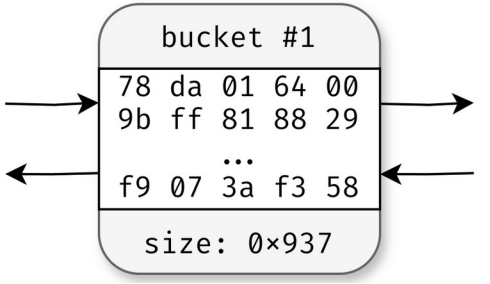
- Read binaries (php, libc)
- ASLR/PIE is NOT a problem
- Address of PHP's heap is known
- Arbitrary allocations/frees

But...

- Unknown environment:
  - Application:
    - Wordpress, Drupal, Symfony...
  - PHP version
  - Heap state
- Need a reliable exploit



Building an exploit: a single bucket



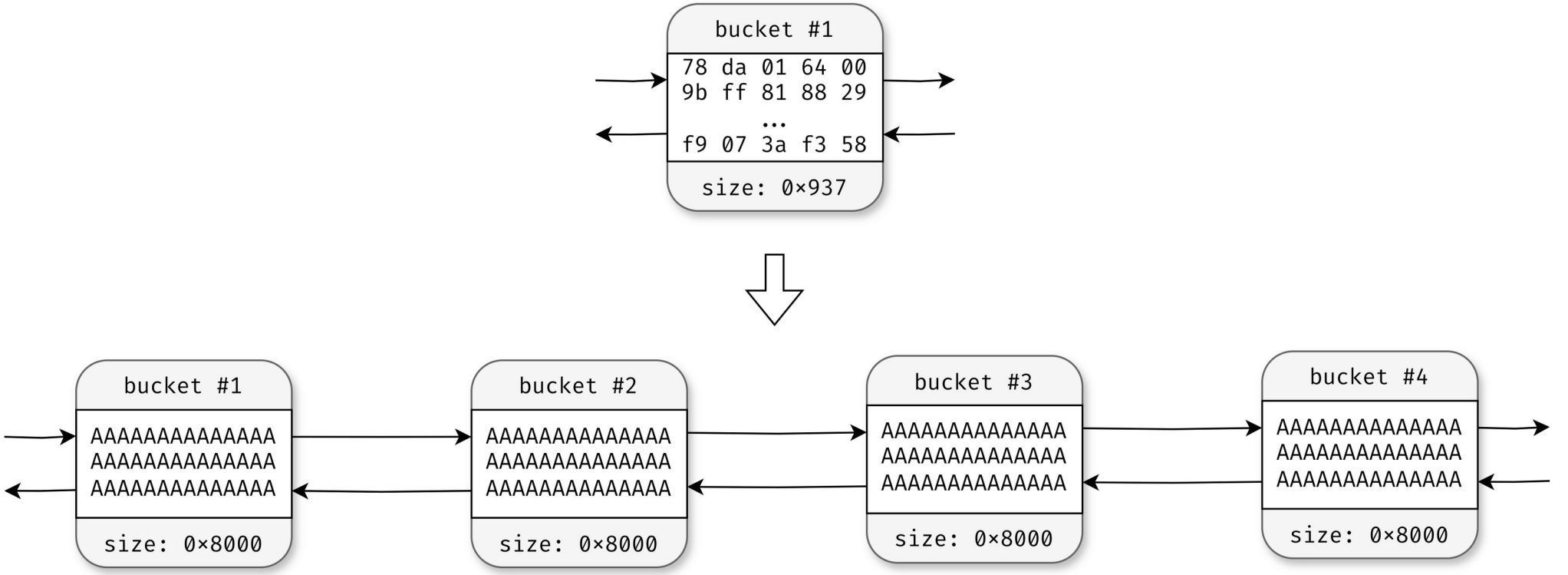
Streams are read in one bucket

Impractical for exploitation

- Cannot :
  - Spray
  - Pad
  - Use altered free list

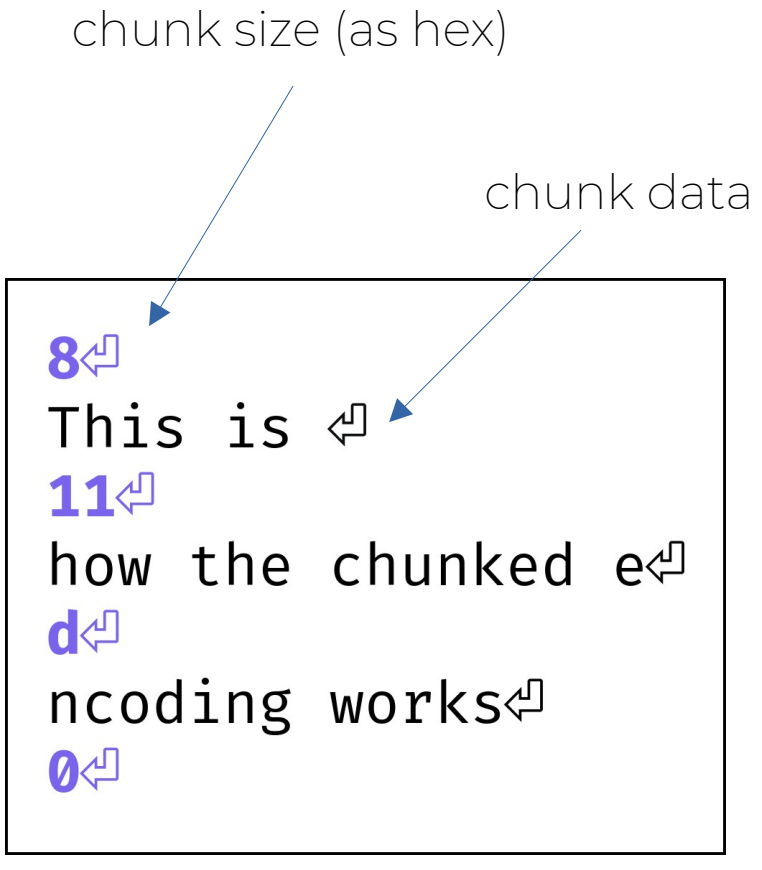


Building an exploit: using zlib.inflate to create buckets





Building an exploit: unfitting chunk size



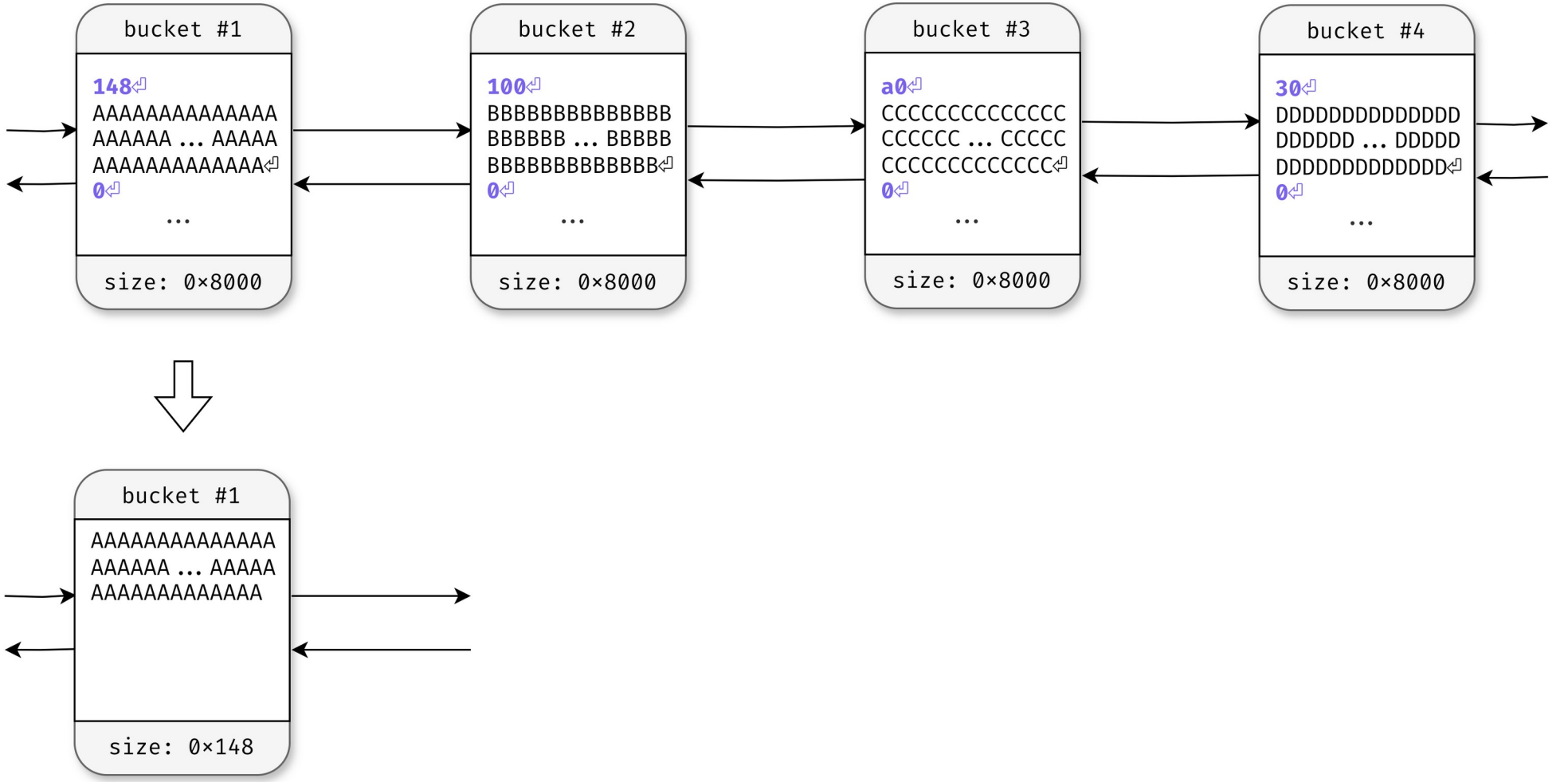
Resizing chunks to arbitrary sizes

*Using the undocumented **dechunk** filter*

« Decodes a string which is HTTP-chunked encoded. »

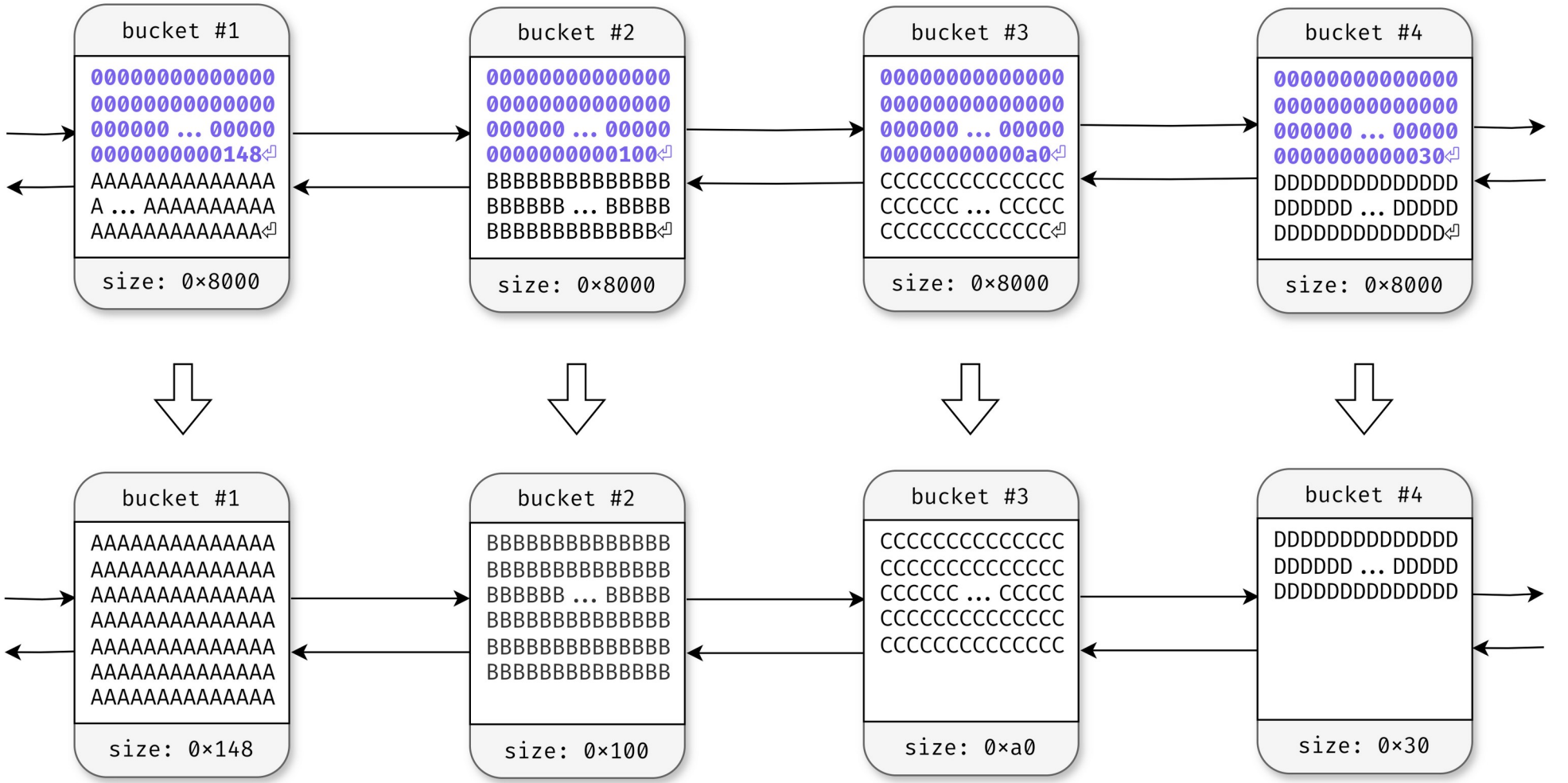


Building an exploit: unfitting chunk size (0x8000)





Building an exploit: using dechunk to change chunk sizes





Building an exploit: arbitrary allocations



Chunk attack

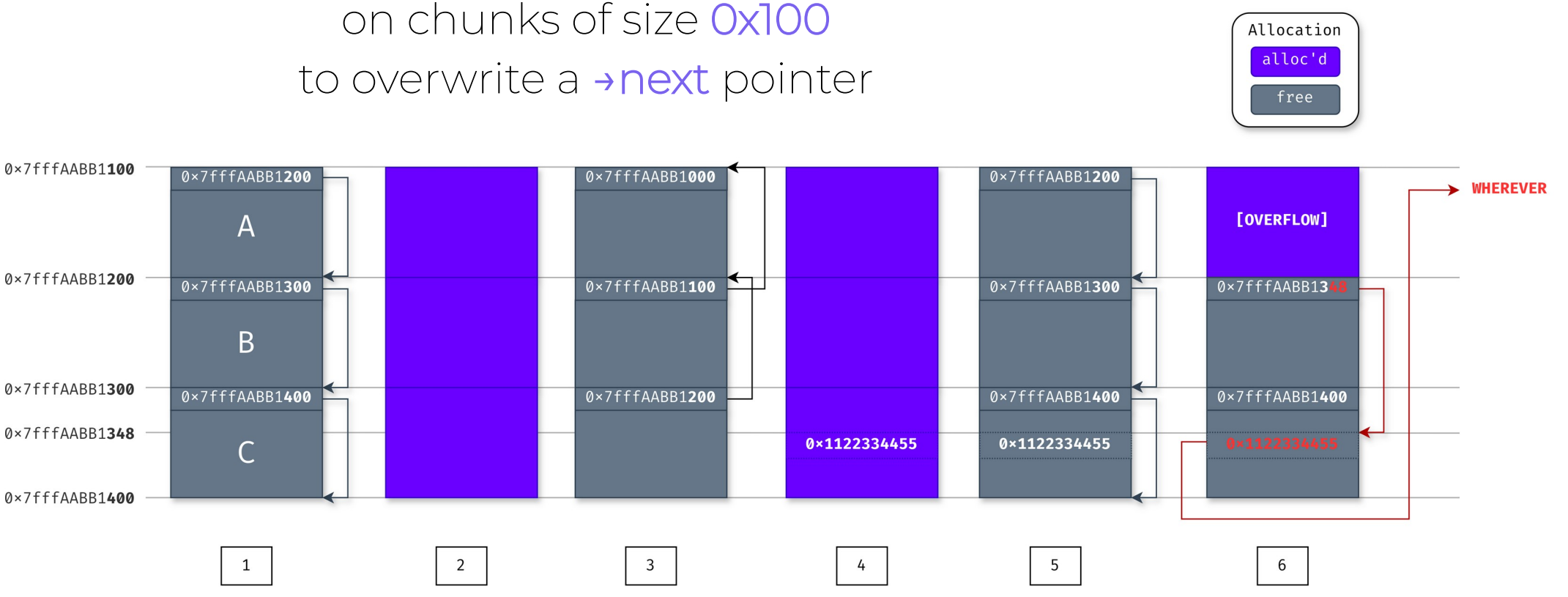
- We attack chunks of size 0x100
- Chunks of other sizes do **not** matter
- Because they are not in the same page

Dechunking buckets

- Use **dechunk** to make buckets appear/disappear
- Buckets appear when they are needed

Building an exploit: controlling a free list

1-byte overflow (48h)
on chunks of size 0x100
to overwrite a →next pointer





Building an exploit: endgame

php-8.3.2/Zend/zend_alloc.c

```
1   struct _zend_mm_heap {
2       // Whether to use custom allocation functions
3       int                use_custom_heap;
4
5       ...
6
7       // Free lists for each chunk size
8       zend_mm_free_slot *free_slot[ZEND_MM_BINS];
9
10      ...
11
12      // Custom allocation functions
13      struct {
14              void      *(*_malloc)(size_t);
15              void       (*_free)(void*);
16              void      *(*_realloc)(void*, size_t);
17      } custom_heap;
18
19      ...
20  };
```

Code execution

- 1-byte overflow (0x48) to alter a free list
- Allocate at arbitrary address
- Overwrite zend_mm_heap
  - At the top of the heap region
- Overwrite function pointers
  - free == system
- Call free(command)



Building an exploit

CNEXT performance

- ✔ Works against any target
  - ➔ From PHP 7+ to current (2015 to 2024)
  - ➔ Any PHP application
- ✔ 100% reliable
  - ➔ No crash
- ✔ Payload < 1000 bytes
  - ➔ Suitable for GET parameters
- ✔ Self-contained exploit
  - ➔ No additional HTTP parameters

File read → RCE



Demo: Magento XXE to RCE

CosmicSting

Magento XXE

(CVE-2024-34102)

June, 2024

IMPACT

Without cnext

Read files

Create API keys

With cnext

Remote

Code

Execution

https://sansec.io/research/cosmicsting





Exploiting direct iconv() calls

Lexfo

Another sink: iconv() calls

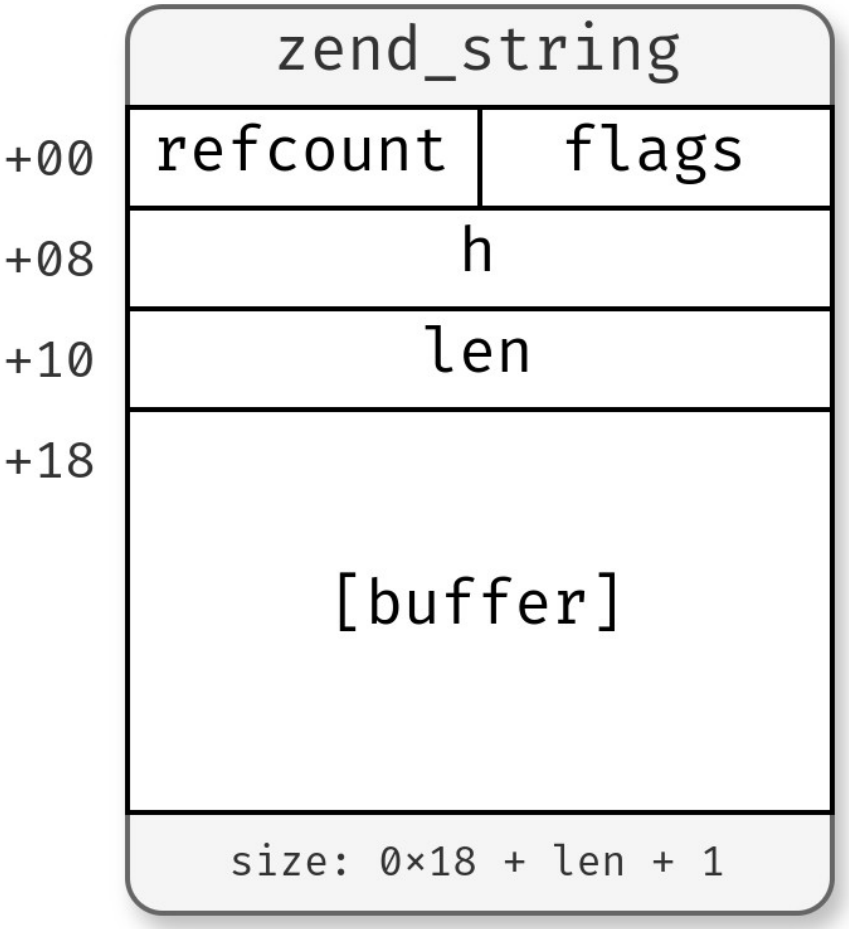
```
iconv(string $from_encoding, string $to_encoding, string $string): string

    iconv_substr( ... ) iconv_mime_decode( ... )
iconv_mime_decode_headers( ... ) iconv_strlen( ... )
    iconv_strpos( ... ) iconv_strrpos( ... )
```

- Direct calls to iconv*() are vulnerable
- Exploitation is completely different



Exploit strategy: getting a leak



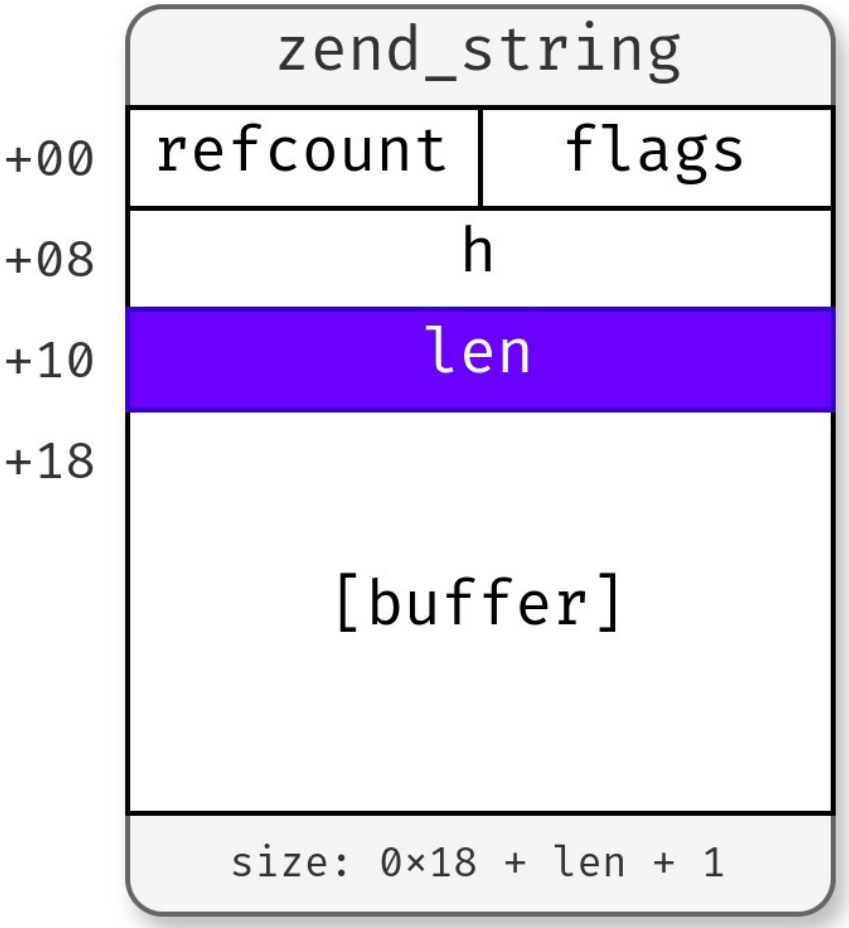
Info leak

- Overwrite → len of a zend_string
- *Before* it is displayed

Bad bug

- 1 to 3 bytes overflow…
- Can only overwrite **refcount**
- Need to **alter free list**

Exploit strategy: getting a leak



Steps

- Corrupt free list
  - Two chunks overlap
- Allocate on top of string
  - Overwrite →len
- String gets displayed



Exploit strategy: reliance on the code

We rely on the code

After we send HTTP parameters, we **cannot** interact with the script anymore

Need to use **« code gadgets »** to perform actions



Exploit strategy: reliance on the code

We rely on the code

```php
<?php

$text = iconv('UTF-8', $_POST['charset'], $_POST['text']);

# end of script

```




Exploit strategy: reliance on the code

We rely on the code

```php
<?php

$text = iconv('UTF-8', $_POST['charset'], $_POST['text']);

$lines = explode("\n", $text); # ← a way to use the bug

```






Real-world exploit

Lexfo

Attacking Roundcube



« Free and open source webmail software for the masses, written in PHP. »

Attacking Roundcube: codewise

When creating an email...

program/include/rcmail_sendmail.php

```php
$charset = rcube_utils::get_input_string('_charset', INPUT_POST);

$mailto  = $this->email_input_format(
    rcube_utils::get_input_string('_to', INPUT_POST, $charset)
);
$mailcc  = $this->email_input_format(
    rcube_utils::get_input_string('_cc', INPUT_POST, $charset)
);
$mailbcc = $this->email_input_format(
    rcube_utils::get_input_string('_bcc', INPUT_POST, $charset)
);

if (!empty($this->invalid_email)) {
    display_error('emailformaterror', 'error', [
        'email' => $this->invalid_email
    ]);
}
```




Attacking Roundcube: memory leak strategy

Simple strategy

1. Set _to to an invalid email ⇒ invalid_email gets set
2. Use _cc to trigger bug ⇒ free list **altered**
3. Use _bcc to allocate on top of invalid_email, altering its size
4. Error is displayed ⇒ **memory leak**



Attacking Roundcube: memory leak failure

A slight oversight

error message is JSON-encoded

```
<script>
  if (window.parent && parent.rcmail) {
    parent.rcmail.iframe_loaded("1");
    parent.rcmail.display_message(
    "Adresse courriel invalide : ajnaajoaajpaajqaa\u0000
                                                
```

Attacking Roundcube: working memory leak strategy

include/rcmail_output_html.php

```php
function get_js_commands(&$framed = null)
{
    $unlock = isset($_REQUEST['_unlock']) ? preg_replace('/[^a-z0-9]/i', '', $_REQUEST['_unlock']) : 0;

    if ($this->framed) {
        $top_commands[] = ['iframe_loaded', $unlock];
    }
    else if ($unlock) {
        $top_commands[] = ['hide_message', $unlock];
    }

    $commands = array_merge($top_commands, $this->js_commands);

    foreach ($commands as $i => $args) {
        $method = array_shift($args);

        foreach ($args as $i => $arg) {
            $args[$i] = self::json_serialize($arg, $this->devel_mode);
        }

        $method = self::JS_OBJECT_NAME . '.' . $method;

        $out .= sprintf("%s(%s);\n", $method, implode(',', $args));
    }

    $framed = $parent_prefix && $parent_commands == count($commands);

    // make the output more compact if all commands go to parent window
    if ($framed) {
        $out = "if (window.parent && parent." . self::JS_OBJECT_NAME . ") {\n"
            . str_replace($parent_prefix, "\tparent.", $out)
            . "}\n";
    }

    return $out;
}
```

Everything is useful

(de)allocations are everywhere

(de)allocations

Made **implode()** of the **second** iteration of the loop overlap with **$out**



Attacking Roundcube: Leak to RCE

RCE is just around the corner

- Data only attack:
  - overwrite **roles**, **serialized** data, ...
- Binary attack:
  - overwrite **zend_array→pDestructor**

Visible part of the iceberg

- Need proper heap setup
- Variable scope is hell
- Very **specific** to server + application
- Generally a huge **pain** to pull off



Attacking Roundcube: DEMO

DEMO





Conclusion

Lexfo

Conclusion: a bad bug... but with some uses

Targeting PHP

- New attack vector: file read to RCE
- Application specific attacks: Roundcube RCE

Other targets

- Other applications may be vulnerable
- Discovered as time go by?



Conclusion: timeline

Timeline (2024)

- **Last year** Crash found
- **February** Started working on bug
- **March 26th** Bug reported to glibc security team
  - They did an amazing job!
- **April 04th** Bug reported to linux distros
- **April 17th** Bug released as **CVE-2024-2961**



Conclusion: resources

Blogposts and exploits

- ambionics.io/blog
- github.com/ambionics/cnext-exploits
- twitter.com/cfreal_





PARIS LYON TOULOUSE

+33 (0)1 40 17 91 28
contact@lexfo.fr

ambionics.io/blog
blog.lexfo.fr

@LexfoSecurite
@ambionics

Charles FOL
@cfreal_

Lexfo



What ifs

Lexfo

What if: blind file read

```
~~echo~~ file_get_contents($_GET['file']);
```

What if there is no output?

- ➔ No /proc/self/maps
- ➔ No access to binaries



What if: no zlib.inflate

What if there is no zlib.inflate?*

- Attack with a single bucket
- Cannot pad heap
- Cannot setup the free list
- Cannot even USE the altered free list

* Very unlikely



Blind, single bucket exploit: DEMO

DEMO

Blind, single-bucket exploit





PARIS LYON TOULOUSE

+33 (0)1 40 17 91 28
contact@lexfo.fr

ambionics.io/blog
blog.lexfo.fr

@LexfoSecurite
@ambionics

Charles FOL
@cfreal_

Lexfo