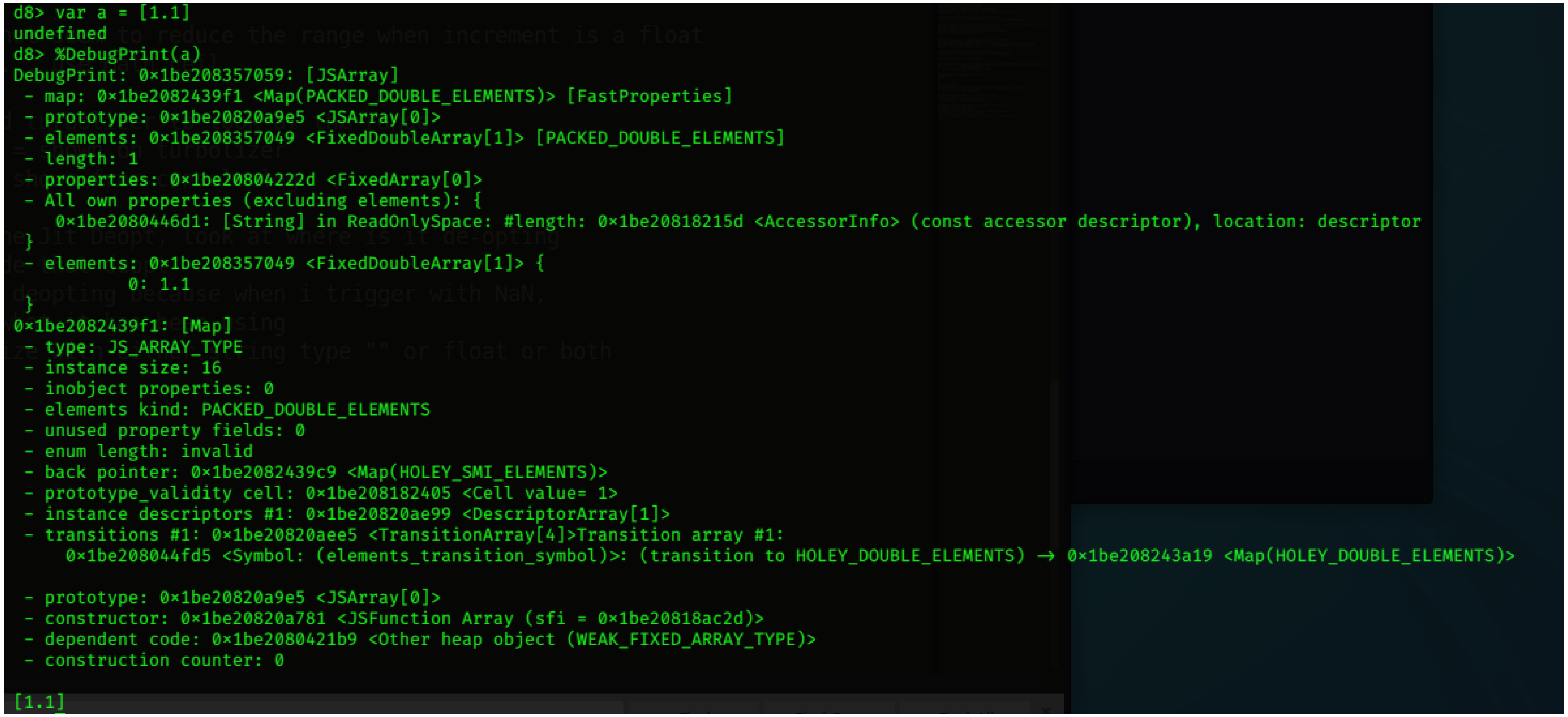
D8 Internals

Why this update?

- Pointer compression
  - Now pointers are only 4 bytes long
  - Harder to see some values
- Also for my own clarity
- Most importantly, breaking down **addrof** and **fakeobj** primitives!!!

Basic

- Run d8 with with native syntax flag ***(./d8 –allow-natives-syntax)***



```
d8> var a = [1.1]
undefined
d8> %DebugPrint(a)
DebugPrint: 0x1be208357059: [JSArray]
 - map: 0x1be2082439f1 <Map(PACKED_DOUBLE_ELEMENTS)> [FastProperties]
 - prototype: 0x1be20820a9e5 <JSArray[0]>
 - elements: 0x1be208357049 <FixedDoubleArray[1]> [PACKED_DOUBLE_ELEMENTS]
 - length: 1
 - properties: 0x1be20804222d <FixedArray[0]>
 - All own properties (excluding elements): {
    0x1be2080446d1: [String] in ReadOnlySpace: #length: 0x1be20818215d <AccessorInfo> (const accessor descriptor), location: descriptor
 }
 - elements: 0x1be208357049 <FixedDoubleArray[1]> {
           0: 1.1
 }
0x1be2082439f1: [Map]
 - type: JS_ARRAY_TYPE
 - instance size: 16
 - inobject properties: 0
 - elements kind: PACKED_DOUBLE_ELEMENTS
 - unused property fields: 0
 - enum length: invalid
 - back pointer: 0x1be2082439c9 <Map(HOLEY_SMI_ELEMENTS)>
 - prototype_validity cell: 0x1be208182405 <Cell value= 1>
 - instance descriptors #1: 0x1be20820ae99 <DescriptorArray[1]>
 - transitions #1: 0x1be20820aee5 <TransitionArray[4]>Transition array #1:
     0x1be208044fd5 <Symbol: (elements_transition_symbol)>: (transition to HOLEY_DOUBLE_ELEMENTS) -> 0x1be208243a19 <Map(HOLEY_DOUBLE_ELEMENTS)>

 - prototype: 0x1be20820a9e5 <JSArray[0]>
 - constructor: 0x1be20820a781 <JSFunction Array (sfi = 0x1be20818ac2d)>
 - dependent code: 0x1be2080421b9 <Other heap object (WEAK_FIXED_ARRAY_TYPE)>
 - construction counter: 0

[1.1]
```

Float Array Internals

Addr	First 4 Bytes	Last 4 bytes
0x1be208357048	…	…
0x1be208357050		
0x1be208357058 (Start from here)	0x0804222d	0x082439f1 (Map)
0x1be208357060	0x00000002 (Length * 2)	0x08357049 (Element Pointer)

Addr	First 4 Bytes	Last 4 bytes
0x1be208357048		
0x1be208357050	0x3ff199999999999a (converted to 1.1)	
0x1be208357058 (Start from here)	0x0804222d	0x082439f1 (Map)
0x1be208357068	0x00000002 (Length * 2)	0x08357049 (Element Pointer)

Important things to take note:

1. The map is **located after** the elements
2. The **map** of the object determine **how** the element is read
   - Whether it should be **converted into a float** or used as a **pointer** to an object
3. The **element pointer** determine **where** we want to read

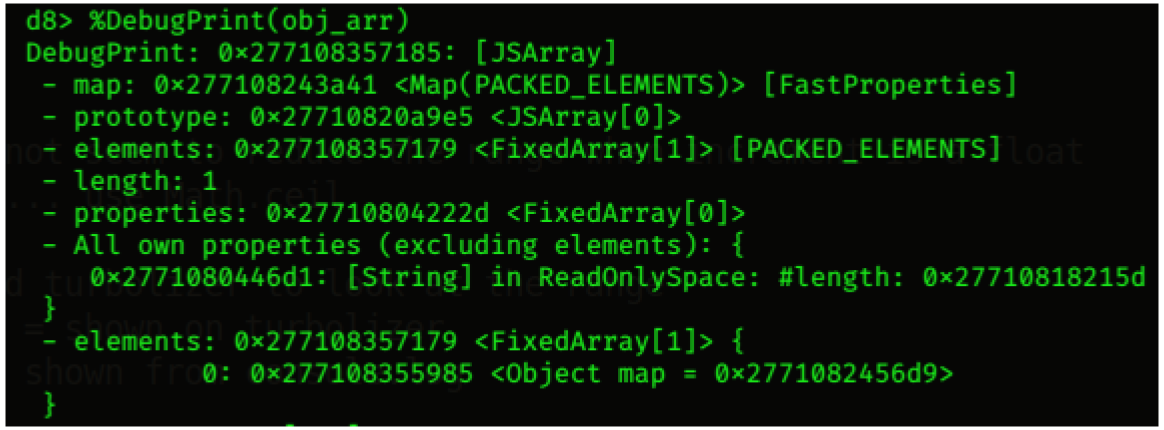
Float to hex convertor:
https://babbage.cs.qc.cuny.edu/IEEE-754.old/Decimal.html

Object Array

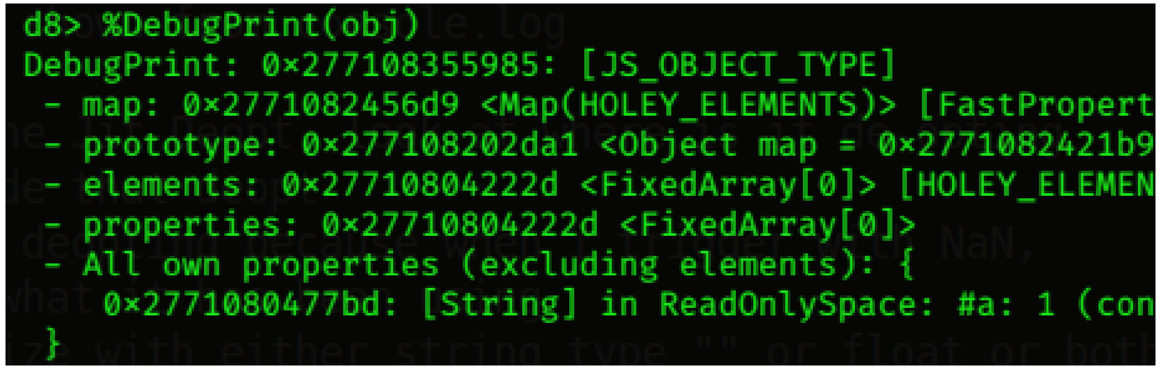
obj = { "a" : 1};

obj_arr = [obj];

%DebugPrint(obj_arr);



```
d8> %DebugPrint(obj_arr)
DebugPrint: 0x277108357185: [JSArray]
 - map: 0x277108243a41 <Map(PACKED_ELEMENTS)> [FastProperties]
 - prototype: 0x27710820a9e5 <JSArray[0]>
 - elements: 0x277108357179 <FixedArray[1]> [PACKED_ELEMENTS]
 - length: 1
 - properties: 0x27710804222d <FixedArray[0]>
 - All own properties (excluding elements): {
    0x2771080446d1: [String] in ReadOnlySpace: #length: 0x27710818215d
 }
 - elements: 0x277108357179 <FixedArray[1]> {
           0: 0x277108355985 <Object map = 0x2771082456d9>
 }
```



```
d8> %DebugPrint(obj)
DebugPrint: 0x277108355985: [JS_OBJECT_TYPE]
 - map: 0x2771082456d9 <Map(HOLEY_ELEMENTS)> [FastPropert
 - prototype: 0x277108202da1 <Object map = 0x2771082421b9
 - elements: 0x27710804222d <FixedArray[0]> [HOLEY_ELEMEN
 - properties: 0x27710804222d <FixedArray[0]>
 - All own properties (excluding elements): {
    0x2771080477bd: [String] in ReadOnlySpace: #a: 1 (con
 }
```

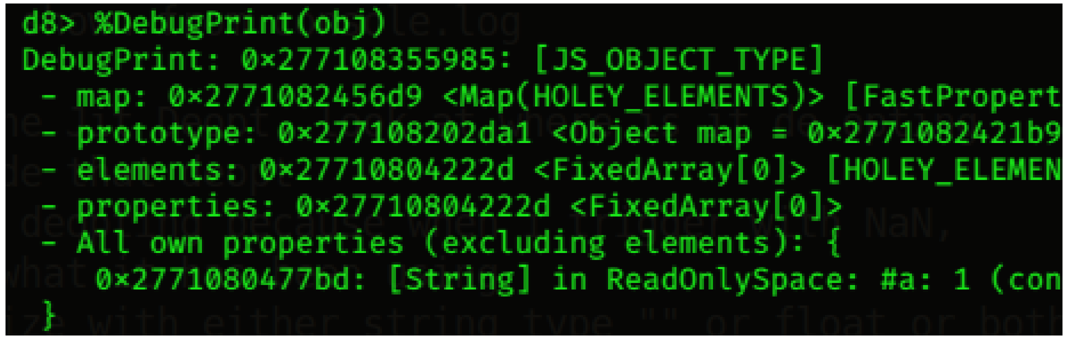
Object Array Internals

Addr	First 4 Bytes	Last 4 bytes
0x277108357178	…	…
0x277108357180		
0x277108357184 (Start from here)	0x0804222d	0x08243a41 (Map)
0x27710835718C	0x00000002 (Length * 2)	0x08357179 (Element Pointer)

Addr	First 4 Bytes	Last 4 bytes
0x277108357178		
0x277108357180		**0x08355985 (Pointer to obj)**
0x277108357184 (Start from here)	0x0804222d	0x08243a41 (Map)
0x27710835718C	0x00000002 (Length * 2)	0x08357179 (Element Pointer)

Important things to take note:

1. The map is **located after** the elements
2. The **map** of the object determine **how** the element is read
    - Whether it should be **converted into a float** or used as a **pointer** to an object
3. The **element pointer** determine **where** we want to read



```
d8> %DebugPrint(obj)
DebugPrint: 0x277108355985: [JS_OBJECT_TYPE]
 - map: 0x2771082456d9 <Map(HOLEY_ELEMENTS)> [FastPropert
 - prototype: 0x277108202da1 <Object map = 0x2771082421b9
 - elements: 0x27710804222d <FixedArray[0]> [HOLEY_ELEMEN
 - properties: 0x27710804222d <FixedArray[0]>
 - All own properties (excluding elements): {
    0x2771080477bd: [String] in ReadOnlySpace: #a: 1 (con
 }
```

AddrOf Primitive

- If we overwrite an obj_arr with a float_array map, we will read the element pointer as a float value instead of a pointer to the obj.
  - This element pointer is set to the object we want to leak

```
Function addrof(obj) {
//Set obj you want to leak to the first element of obj_arr
obj_arr[0] = obj;

//Overwrite the obj_arr map with a float_arr map, assuming you have OOB R/W
//Since the map is located after elements, it is at obj_arr[length+X]
obj_arr[length + X] = float_array_map;

addr = obj_arr[0];
return addr;

//Usually not done like this since you will corrupt the obj_arr. I usually do it by placing a corrupted array above the
obj_arr and float_arr and using it to control both the element and map pointer
}
```

AddrOf Primitivces (Memory layout of Obj_Arr)

Addr	First 4 Bytes	Last 4 bytes
0x277108357178	…	…
0x277108357180		
0x277108357184 (Start from here)	0x0804222d	**Overwritten with a float map**
0x27710835718C	0x00000002 (Length * 2)	0x08357179 (Element Pointer)

Addr	First 4 Bytes	Last 4 bytes
0x277108357178		
0x277108357180		**0x08355985 (Set pointer to obj we want to leak)**
0x277108357184 (Start from here)	0x0804222d	Overwritten with a float map
0x27710835718C	0x00000002 (Length * 2)	0x08357179 (Element Pointer)

FakeObj Primitive

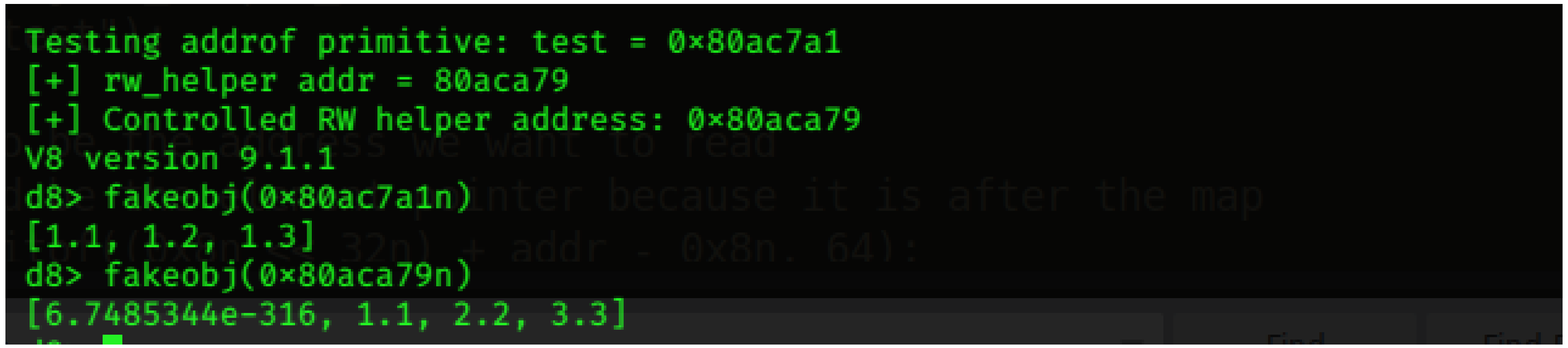
- Harder to visualize
- Reverse of AddrOf Primitive
- Concept:
  1. Set **float_arr[0]** with the **addr** you want to place the **fake object**
  2. Change map of **float_arr** to **obj_arr**
  3. Compiler now thinks float_arr[0] is a pointer to another ‘object’

```
Function fakeobj(addr){
let fake;
// [1]
float_arr[0] = addr;

// [2]
float_arr[map] = obj_arr_map;
fake = float_arr[0];
return fake;
}
```

FakeObj Primitive

- I don't have a good way of testing this primitive, but this is what I do…
- Create a random test array object
- Test your addrof primitive to leak the test array addr
- Use it on the fakeobj primitive
- Screenshot of how fakeobj should return:



```
Testing addrof primitive: test = 0x80ac7a1
[+] rw_helper addr = 80aca79
[+] Controlled RW helper address: 0x80aca79
V8 version 9.1.1
d8> fakeobj(0x80ac7a1n)
[1.1, 1.2, 1.3]
d8> fakeobj(0x80aca79n)
[6.7485344e-316, 1.1, 2.2, 3.3]
```

Some Samples

- You can find some of my code here:
  - https://github.com/cddc12346/RandomCTFs/blob/master/PicoCTF%202021/HorsePower/pwn.js
  - https://github.com/cddc12346/RandomCTFs/blob/master/PicoCTF%202021/turboflan/pwn4.js
- I apologize they were not the neatest…
- Or just ping me on twitter (@n00bsh1t)
- You may also check out my blog (https://ditt0.medium.com/)