GC woah
What?
Figuring out the heap address of a script object.
What?
Figuring out the heap address of a script object. (without a traditional memory read bug)

Dion Blazakis Center for Bugs Who Can’t Read Good
Why?
Knowing is half the battle.
‘Cause we can.
Disclaimer:

Probably least practical thing you’ll see all Summerc0n.

Contact redpantz about refund. (just be sure to speak his language)
How?

Use imprecision in the conservative garbage collection (GC) as a heap address oracle.
How?

Garbage (collection)
How?

Turn a conservative garbage collector into a heap address oracle
Garbage collection

Memory management is handled by the GC:

```javascript
var obj = new Object();
obj.drink = "at Summerc0n.";
obj = null;
```
Garbage collection

Memory management is handled by the GC:

```javascript
var obj = new Object();
obj.drink = "at Summerrc0n.";
obj = null;

o1 = new ScriptObject(OBJECT);
```
Garbage collection

Memory management is handled by the GC:

```javascript
var obj = new Object();
obj.drink = "at Summerc0n.";
obj = null;

o1 = new ScriptObject(OBJECT);
o2 = new ScriptObject(STRING);
```
Garbage collection

Memory management is handled by the GC:

```javascript
var obj = new Object();
obj.drink = “at Summerc0n.”;
obj = null;

o1 = new ScriptObject(OBJECT);
o2 = new ScriptObject(STRING);
delete o1;
delete o2;
```
Runtimes started out with reference counting.

Keep track of the number of references and free if this goes to 0.

Reference Counting
Runtimes started out with reference counting
Keep track of number of references and free when 0
Reference Counting

Keep track of number of references and free when 0
Reference Counting

boost::shared_ptr

Keep track of number of references and free when 0
Example: Reference Cycles

```javascript
var x = new Object(); // obj 1: 1 refs
var y = new Object(); // obj 2: 1 refs
x.wizard_hat = y; // obj 2: 2 refs
y.wizard_robe = x; // obj 1: 2 refs
x = null; // obj 1: 1 refs
y = null; // obj 2: 1 refs
forceGC(); // uh oh 😞
```
GC: Mark and Sweep

1. Find all live objects.
2. Mark them as used.
3. Ask the allocator to walk all allocations and free those that aren't marked. (SWEEEEEEEP)
GC: Mark and Sweep

1. Find all live objects.
2. Mark them as used.
3. Ask the allocator to walk all allocations and free those that aren’t marked. (SWEEEEEEEEEP)
GC: Marking

worklist = gc_roots
while (!worklist.isEmpty()) {
    gc_obj = worklist.dequeue();
    if (!gc_obj.isMarked()) {
        gc_obj.mark();
        worklist.queue(gc_obj.getRefs());
    }
}
worklist = gc_roots;
while (!worklist.isEmpty()) {
    gc_obj = worklist.dequeue();
    if (!gc_obj.isMarked()) {
        gc_obj.mark();
        worklist.queue(gc_obj.getRefs());
    }
}

Ensuring all roots are accounted for isn’t trivial.

Especially with JIT and native structures holding refs to script objects.

Some engines make the effort to do this precisely (see V8).

Most punt.
Conservative GC

If you can't prove otherwise, assume a value in memory (only some regions are – like the stack) is a GC-able object.
Conservative GC

If you can't prove otherwise, assume a value in memory (only some regions are – like the stack) is a GC-able object.

OVERMARKING!
var f = function(x) {
    do_something_that-causes_gc();
};

var y = 0x24242424;

f(y);
var f = function(x) {
    do_something_that-causes_gc();
};

var y = 0x24242424;

f(y);

“Hey, allocator, does 0x24242424 look like it could be a pointer to a heap object? Should I mark it?”
var f = function(x) {
  do_something_that_causes_gc();
};

var y = 0x24242424;

f(y);

Hey, allocator, does 0x24242424 look like it could be a pointer to a heap object? Should I mark it?”
var f = function(x) {
    do_something_that_causes_gc();
};

var y = 0x24242424;

f(y);
The Plan

1. Make a bunch of objects
2. Put some address guesses on the stack
3. Remove all refs to objects and force GC
4. Are they all gone?
   • If not we found the address!
   • If so, guess another address!
The Plan

1. Make a bunch of objects
2. Put some address guesses on the stack
3. Remove all refs to objects and force GC
4. Are they all gone?
   • If not we found the address!
   • If so, guess another address!
Steps: 1. Bunch o’ Objects

count = a_whole_bunch;
strongs = new Array();

while (count > 0) {
    var obj: Object = createSprayObject(count);
    strongs.push(obj);
    count -= 1;
}

Steps 2: Put guesses on stack

```
sprayStackArguments(
    pageToDouble(baseAddress + scanDelta * 0),
    pageToDouble(baseAddress + scanDelta * 1),
    pageToDouble(baseAddress + scanDelta * 2),
    pageToDouble(baseAddress + scanDelta * 3),
    pageToDouble(baseAddress + scanDelta * 4),
    pageToDouble(baseAddress + scanDelta * 5),
    pageToDouble(baseAddress + scanDelta * 6),
    pageToDouble(baseAddress + scanDelta * 7),
    recurseDepth);
```
Step 3: Remove refs and GC

```javascript
strongs = new Array();
forceGC();
```
Steps 4: Any pinned objs?

Shit.
Interlude: Chicken and egg

How do we ask if something has been collected if we have no references left to it?
Interlude: Chicken and egg

How do we ask if something has been collected if we have no references left to it?

For ActionScript:
Use a weak Dictionary
Dictionary() Constructor
public function Dictionary(weakKeys:Boolean = false)

Language Version: ActionScript 3.0
Runtime Versions: AIR 1.0, Flash Player 9, Flash Lite 4

Creates a new Dictionary object. To remove a key from a Dictionary object, use the delete operator.

Parameters

weakKeys:Boolean (default = false) — Instructs the Dictionary object to use "weak" references on object keys. If the only reference to an object is in the specified Dictionary object, the key is eligible for garbage collection and is removed from the table when the object is collected. Note that the Dictionary never removes weak string keys from the table. Specifically, in the case of string keys, the weak
Steps: 1. Bunch o’ Objects (v2)

count = a_whole_bunch;
weaks = new Dictionary(true);
strongs = new Array();

while (count > 0) {
    var obj: Object = createSprayObject(count);
    weaks[obj] = count;
    strongs.push(obj);
    count -= 1;
}
Step 4. Any pinned objs? (v2)

```java
public function countKeys(d:Dictionary) {
    var count = 0;
    for (var key : Object in d) {
        count += 1;
    }
    return count;
}
```
Step 4. Any pinned objs? (v2)

```java
public function anyLeft() {
    for (var key : Object in weaks) {
        // Pin this again so it doesn't
        // go away when stack unwinds
        strongs.push(key);
    }
    return countKeys(weaks);
}
```
Demonstration.
Firefox? (SpiderMonkey?)
This is an experimental technology, part of the Harmony (EcmaScript 6) proposal. Because this technology's specification has not stabilized, check the compatibility table for usage in various browsers. Note that the syntax and behavior of an experimental technology is subject to change in future versions of browsers.

**Introduction**

WeakMaps are key/value maps in which keys are objects.

**API**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>myWeakMap.get(key [, defaultValue])</code></td>
<td>Returns the value associated to the key object, defaultValue if not found.</td>
</tr>
<tr>
<td><code>myWeakMap.set(key, value)</code></td>
<td>Set the value for the key object in myWeakMap. Returns undefined.</td>
</tr>
<tr>
<td><code>myWeakMap.has(key)</code></td>
<td>Returns a boolean asserting whether a value has been associated.</td>
</tr>
<tr>
<td><code>myWeakMap.delete(key)</code></td>
<td>Removes any value associated to the key object. After such a call, the key is not associated to any value.</td>
</tr>
<tr>
<td><code>myWeakMap.clear()</code></td>
<td>Empty the myWeakMap from all its elements. Returns undefined.</td>
</tr>
</tbody>
</table>

**Example**

```javascript
var wm1 = new WeakMap(),
    wm2 = new WeakMap(),
    wm3 = new WeakMap();
var o1 = {};
```
“Because of references being weak, WeakMap keys are not enumerable (i.e. there is no method giving you a list of the keys). If they were, the list would depend on the state of garbage collection, introducing non-determinism.”
Solution: Hash Growth Timing

897  static const uint8_t sMaxAlphaFrac = 192; // (0x100 * .75)
...
1023  bool overloaded()
1024  {
1025      return entryCount + removedCount >= ((sMaxAlphaFrac * capacity()) >> 8);
1026  }
...
1178  RebuildStatus checkOverloaded()
1179  {
1180      if (!overloaded())
1181          return NotOverloaded;
...
1193      return changeTableSize(deltaLog2);
1194  }
Solution: Hash Growth Timing

1142 RebuildStatus changeTableSize(int deltaLog2)
1143 {
...
1147     uint32_t newLog2 = sHashBits - hashShift + deltaLog2;
1148     uint32_t newCapacity = JS_BIT(newLog2);
...
1154     Entry *newTable = createTable(*this, newCapacity);
...
1164     // Copy only live entries, leaving removed ones behind.
1165     for (Entry *src = oldTable, *end = src + oldCap; src < end; ++src) {
1166         if (src->isLive()) {
1167             HashNumber hn = src->getKeyHash();
1168             findFreeEntry(hn).setLive(hn, Move(src->get()));
1169             src->destroy();
1170         }
1171     }