Verifying Practical Garbage Collectors

Chris Hawblitzel

Joint work with Erez Petrank (Technion)

Virtual machines for languages

Java bytecode / .NET bytecode / JavaScript / ActionScript

Virtual Machine

compiler

libraries

x86

evaluation handling

garbage collector

concurrency

run-time system
Virtual machine vulnerabilities

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Entry Date: Aug 15 2007</td>
<td>Published: July 10, 2007</td>
</tr>
<tr>
<td>A remote user can create specially crafted JavaScript that, when loaded by the target user, will make a virtual function call on an invalid pointer to trigger a memory dereference error and execute arbitrary code on the target system.</td>
<td>What causes the vulnerability? An unchecked buffer in the .NET Framework 2.0 JIT Compiler service within the .NET Framework.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mozilla Firefox Bug in JavaScript Garbage Collector Lets Remote Users Deny Service</th>
<th>From: Apple Product Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advisory: Mozilla Foundation Security Advisory</td>
<td>Date: Fri, 11 Jul 2008</td>
</tr>
<tr>
<td>Date: Apr 16 2008</td>
<td></td>
</tr>
<tr>
<td>A remote user can create specially crafted HTML that, when loaded by the target user, will trigger a flaw in the JavaScript garbage collector code and cause the target user's browser to crash. ... The vendor indicates that similar crashes have been exploitable in the past, so arbitrary code execution may be possible...</td>
<td>Available for: iPhone v1.0 through v1.1.4, iPod touch v1.1 through v1.1.4</td>
</tr>
<tr>
<td>Description: A memory corruption issue exists in JavaScriptCore's handling of runtime garbage collection. Visiting a maliciously crafted website may lead to an unexpected application termination or arbitrary code execution. This update addresses the issue through improved garbage collection.</td>
<td></td>
</tr>
</tbody>
</table>

Outline

- Virtual machines, vulnerabilities
- Proof-carrying code
  - Goals:
    - verify properties of compiler output (e.g. type safety)
    - verify properties of run-time system (e.g. correctness)
- Simple verified GC
- Practical verified GCs
Proof-carrying code (Necula, Lee 1996)

Java bytecode / .NET bytecode / JavaScript / ActionScript

Virtual Machine

compiler

libraries

x86

exception

garbage collector

concurrence

run-time system

Verifier

Type-preserving compilation in Bartok

Bartok Compiler (~200,000 LOC)

Optimizations: copy propagation, constant propagation, constant folding, CSE, dead-code elimination, loop-invariant removal, loop rotation, inlining, tree shaking, array bounds check elimination (ABCD, induction var), reverse CSE for LEA instruction, peephole optimizations, elimination of redundant CC ops, optimize boolean test and branch, floating point stack optimizations, convert “ADD” to “LEA”, graph-coloring register allocation, code layout, ...

“Typed assembly language” (Morrisett et al 1998)
Typed output from Bartok

```
B1: {eax:int, ebx:int}
cmp eax, 5
jne B3

B2:
    mov ecx, eax
    jmp B1

B3:
    mov ecx, ebx
    add ecx, 1

B4: {ecx:int}
    add ecx, 1
    ...
```

Array bounds safety in Bartok

Bartok Compiler (~200,000 LOC)

**Optimizations:** copy propagation, constant propagation, constant folding, CSE, dead-code elimination, loop-invariant removal, loop rotation, inlining, tree shaking, array bounds check elimination, unsafe induction var, reverse CSE for LEA instruction, peephole optimizations, elimination of redundant CC ops, optimize boolean test and branch, floating point stack optimizations, convert “ADD” to “LEA”, graph-coloring register allocation, code layout, ...
A bug in Bartok: ABCD

counterexample:
- \( i = -2^{31} \)
- \( j = 2^{31} - 1 \)
- \( j > i \)

vulnerability:
- Address = \&\( \text{arr}[0] \) + 4*\( j \)
  = \&\( \text{arr}[0] \) + 4*(2^{31} - 1)
  = \&\( \text{arr}[0] \) - 4

Outline

- Virtual machines, vulnerabilities
- Proof-carrying code
- Simple verified GC
  - Program verification with Boogie/Coq, Boogie/Z3
  - Miniature mark-sweep collector
  - Integers, bit vectors, arrays, quantifiers
- Practical verified GCs
Program verification

(demo)
Garbage collector properties

- **safety:** gc does no harm
  - graph isomorphism
    - concrete graph represents abstract graph
      (McCreight, Shao et al. 2007)

- **effectiveness**
  - after gc, unreachable objects reclaimed

- **termination**

- **efficiency**
  

---

### Proving safety

**Abstract graph**

- A (root)
- B
- C

**Concrete graph**

- A
- B
- $\text{StoAbs}$
- Mem

---

**procedure GarbageCollect**

```plaintext
requires MutatorInv(root, Color, $\text{StoAbs}$, $\text{AbsMem}$, Mem);
```

**function MutatorInv(...)** returns (bool) {
  WellFormed($\text{StoAbs}$) && memAddr(root) && $\text{StoAbs}$[root] != NO_ABS
  && (forall i:int::{T(i)} T(i) && memAddr(i) => $\text{ObjInv}(i, \text{StoAbs}, \text{AbsMem}, \text{Mem}))
  && (forall i:int::{T(i)} T(i) && memAddr(i) => White(Color[i]))
  && (forall i:int::{T(i)} T(i) && memAddr(i) => ($\text{StoAbs}$[i] == NO_ABS <=> Unalloc(Color[i])))
}

**function ObjInv(...)** returns (bool) {
  $\text{StoAbs}$[i] != NO_ABS =>
    ...
    $\text{StoAbs}$[Mem[i, field]] != NO_ABS ...
    ...
    $\text{StoAbs}$[Mem[i, field]] == $\text{AbsMem}$[$\text{StoAbs}$[i], field] ...
}
Verifying Mark

procedure Mark(ptr:int)
requires GcInv(Color, $toAbs, $AbsMem, Mem);
requires memAddr(ptr) && T(ptr);
requires $toAbs[ptr] != NO_ABS;
modifies Color;
ensures GcInv(Color, $toAbs, $AbsMem, Mem);
ensures (forall i:int::{T(i)} T(i) && !Black(Color[i]) ==> Color[i] == old(Color[i]));
ensures !White(Color[ptr]);
{
  if (White(Color[ptr])) {
    Color[ptr] := 2; // make gray
    call Mark(Mem[ptr,0]);
    call Mark(Mem[ptr,1]);
    Color[ptr] := 3; // make black
  }
}
forall i:int::{T(i)} T(i) && memAddr(i) ==> ObjInv(i, $toAbs, $AbsMem, Mem)
Integers, bit vectors, arrays (decidable)

procedure Mark(ptr:int)
requires GcInv(Color, $toAbs, $AbsMem, Mem);
requires memAddr(ptr) & $toAbs != NO_ABS;
modifies Color;
ensures GcInv(Color, $toAbs, $AbsMem, Mem);
ensures (forall i:int::{T(i) & !Black(Color[i])} => Color[i] == old(Color[i]));
ensures !White(Color[ptr]);
{
  if (White(Color[ptr])) {
    Color[ptr] := 2; // make gray (equivalent to “Color := Color[ptr := 2];”
    call Mark(Mem[ptr,0]);
    call Mark(Mem[ptr,1]);
    Color[ptr] := 3; // make black
  }
}

memAddr(i) <=> memLo <= i && i < memHi

boolean expressions

e ::= true | false | x | !e | e & & e | e => e | ...
linear integer arithmetic

e ::= ... | -2 | -1 | 0 | 1 | 2 | e + e | e - e | e <= e | ...
bit vector arithmetic

e ::= ... | e & e | e << e | ...
arrays

e ::= ... | e[e] | e[e := e]

Quantifiers (undecidable)

procedure Mark(ptr:int)
requires GcInv(Color, $toAbs, $AbsMem, Mem);
requires memAddr(ptr) & & T(ptr);
requires $toAbs[ptr] != NO_ABS;
modifies Color;
ensures GcInv(Color, $toAbs, $AbsMem, Mem);
ensures (forall i:int::{T(i)} T(i) & & !Black(Color[i]) => Color[i] == old(Color[i]));
ensures !White(Color[ptr]);
{
  if (White(Color[ptr])) {
    Color[ptr] := 2; // make gray (equivalent to “Color := Color[ptr := 2];”
    call Mark(Mem[ptr,0]);
    call Mark(Mem[ptr,1]);
    Color[ptr] := 3; // make black
  }
}
Outline

• Virtual machines, vulnerabilities
• Proof-carrying code
• Simple verified GC
• Practical verified GCs
  – Mark-sweep collector
    • Mark stack
    • Table of color bits (2 bits per color)
    • Allocate small objects from cache
    • Allocate large objects, caches from free list
  – Copying collector
    • Simple 2-space Cheney queue algorithm
    • Not generational

Mutator-GC interface

The mutator (the program) calls these procedures:

- Initialize()
- read($ptr\text{:int}, $field\text{:int})
- write($ptr\text{:int}, $field\text{:int}, $val\text{:int})
- AllocateObject(VTable vtable)
- AllocateString(int stringLength)
- AllocateVector(VTable vtable, int numElems)
- AllocateArray(VTable vtable, int numDims, int numElems)

(Note: we define a field to be an aligned, 4-byte word)
Object layouts

Class object
- preheader
- vtable
- data field
  - char
- data field

String
- preheader
- vtable
- numChars
- (private)
- char
- char

Vector
- preheader
- vtable
- numElems
- elements
- elements

Array
- preheader
- vtable
- numElems
- numElems
- base/length
- elements

VTables and GC information

Class object
- preheader
- vtable
- data field
  - char
- data field

VTable
- gc info
  - bit
  - tag

VTable
- gc info
  - bit
  - tag

... tag(vtable) == ?STRING_TAG =>
  numFields(abs) >= 4
  & & 4 * numFields(abs) == pad(16 + 2 * numChars)
...

... tag(vtable) == ?DENSE_TAG =>
  (forall j:int::{T(j)} T(j)) =>
  2 <= j & & j < numFields(abs) =>
  VFieldPtr(abs, j) == (j < 30 & & getBit(mask(vtable), 2 + j))
...

...
forall j1:int, j2:int ...
0 <= j1 && j1 < j2 && j2 <= tableSize ==> 
readOnlyMem(table + 4 * j1) < readOnlyMem(table + 4 * j2)
...

Other features

• Supported:
  – structs (multi-word values)
  – interior pointers

• Not supported
  – threads
  – pinning
procedure CopyAndForward($ptr:int, $_tj:int)
requires ecx == $ptr;
requires CopyGcInv(...);
requires Pointer($r1, $ptr, $r1[$ptr]);
...
{ 
call edx := GcRead(ecx + 4);
esp := esp - 4; callGetSize($ptr, edx, $r1, $r1);
ebp := eax; 
... 
edi := 0;
edx := 0;
loop:
assert 4 * edi == edx;
assert CopyGcInv(...);
...
if (edx >= ebp) { goto loopEnd; }
call copyWord($ptr, $_tj, esi, edi, ebp);
call edi := Add(edi, 1);
call edx := Add(edx, 4);
goto loop;
loopEnd:
call eax := Lea(esi + 4);
call GcWrite(ecx + 4, eax);
... 
}
Real verified code: copying collector

automatic translation

mov edi, 0
mov edx, 0
CopyAndForward$loop:

cmp edx, ebp
jae CopyAndForward$loopEnd

size of collector:
boogie source (excl comments, blanks): 3177 lines
x86 instructions (before inlining): 979
Boogie/Z3 verification time: 2 minutes

Performance

Benchmark: Bartok (largest of 10 benchmarks)
Conclusions

• Ultimate goals:
  – typed compiled code
  – verified run-time system

• Verified run-time status: two practical GCs
  – Original mutator-GC interface specification had two bugs, so:
    • Copying collector:
      – crashed 1st time (spec bug: Initialize not asked to save ebp)
      – crashed 2nd time (spec bug: preheader vs. header confusion)
      – worked 3rd time
    • Mark-sweep (with corrected spec): worked 1st time