CS 6V81-05: System Security and Malicious Code Analysis
Design and Implementation of Data Flow Analysis

Zhiqiang Lin

Department of Computer Science
University of Texas at Dallas

February 13\textsuperscript{th}, 2012
Outline

1. Shadow Memory
2. Example: Type Inference
3. Implementation
   - Valgrind
   - QEMU
   - PIN
4. Summary
Outline

1. Shadow Memory

2. Example: Type Inference

3. Implementation
   - Valgrind
   - QEMU
   - PIN

4. Summary
Shadow memory describes a computer science technique in which potentially every byte used by a program during its execution has a shadow byte or bytes.
How to Store Abstract State

Shadow memory describes a computer science technique in which potentially every byte used by a program during its execution has a shadow byte or bytes.

These shadow bytes are typically invisible to the original program and are used to record information about the original piece of data.
Shadow memory describes a computer science technique in which potentially every byte used by a program during its execution has a shadow byte or bytes.

These shadow bytes are typically invisible to the original program and are used to record information about the original piece of data.

The program is typically kept unaware of the existence of shadow memory by using a dynamic binary translator/instrumentor, which, among other things, may translate the original programs memory read and write operations into operations that do the original read and write and also update the shadow memory as necessary.
How to Store Abstract State

- Shadow memory
  - We need a mapping
    - Addr → Abstract State
    - Register → Abstract
Shadow Memory
– We need a mapping
  Addr → Abstract State
  Register → Abstract

Virtual Space
[addr] → val
How to Store Abstract State

- Shadow memory
  - We need a mapping
    - Addr → Abstract State
    - Register → Abstract
How to Store Abstract State

- Shadow memory
  - We need a mapping
    - Addr → Abstract State
    - Register → Abstract
Shadow Memory

- We need a mapping
  - Addr → Abstract State
  - Register → Abstract State

Virtual Space

[addr] → val
abs

Shadow Space
typedef
  struct {
    UChar abits[65536];
  } SecMap;

static SecMap* primary_map[65536];
static SecMap  default_map;

static void init_shadow_memory (void)
{
  for (i = 0; i < 65536; i++)
    default_map.abits[i] = 0;
  for (i = 0; i < 65536; i++)
    primary_map[i] = &default_map;
}

static SecMap* alloc_secondary_map()
{
  map=malloc(sizeof(SecMap));
  for (i = 0; i < 65536; i++)
    map->abits[i] = 0;
  return map;
}

void Accessible (addr)
{
  if (primary_map[(addr) >> 16] == default_map)
    primary_map[(addr) >> 16] = alloc_secondary_map(caller);
}
Outline

1. Shadow Memory
2. Example: Type Inference
3. Implementation
   - Valgrind
   - QEMU
   - PIN
4. Summary
Data Flow Based Type Resolution

- movl $0x8048118,%eax
- mov %eax, 0x4(%esp)
- movl $0x8049128,(%esp)
- call 0x80480e0 <strcpy>
- mov $0x14, %eax
- int $0x80
- ret

<table>
<thead>
<tr>
<th>Mem,Reg</th>
<th>Tag</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Data Flow Based Type Resolution

<table>
<thead>
<tr>
<th>Mem,Reg</th>
<th>Tag</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8048118</td>
<td>✗</td>
<td>N/A</td>
</tr>
</tbody>
</table>

- `movl $0x8048118,%eax`
- `mov %eax, 0x4(%esp)`
- `movl $0x8049128,(%esp)`
- `call 0x80480e0 <strcpy>`
- `mov $0x14, %eax`
- `int $0x80`
- `ret`
- `mov %eax, 0x8049124`
### Data Flow Based Type Resolution

```assembly
movl $0x8048118,%eax
mov %eax, 0x4(%esp)
movl $0x8049128,(%esp)
call 0x80480e0 <strcpy>
mov $0x14, %eax
int $0x80
ret
mov %eax, 0x8049124
```

<table>
<thead>
<tr>
<th>Mem,Reg</th>
<th>Tag</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8048118</td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>eax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>int $0x80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ret</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mov %eax, 0x8049124</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Data Flow Based Type Resolution

movl $0x8048118,%eax
mov %eax, 0x4(%esp)
movl $0x8049128,(%esp)
call 0x80480e0 <strcpy>
mov $0x14, %eax
int $0x80
ret
mov %eax, 0x8049124

<table>
<thead>
<tr>
<th>Mem, Reg</th>
<th>Tag</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8048118</td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>eax</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Data Flow Based Type Resolution

```
movl $0x8048118,%eax
mov %eax, 0x4(%esp)
movl $0x8049128,(%esp)
call 0x80480e0 <strcpy>
mov $0x14, %eax
int $0x80
ret
mov %eax, 0x8049124
```

<table>
<thead>
<tr>
<th>Mem,Reg</th>
<th>Tag</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8048118</td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>eax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x4(%esp)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Data Flow Based Type Resolution

```
movl $0x8048118,%eax
movl %eax, 0x4(%esp)
movl $0x8049128,(%esp)
call 0x80480e0 <strcpy>
mov $0x14, %eax
int $0x80
ret
mov %eax, 0x8049124
```

<table>
<thead>
<tr>
<th>Mem,Reg</th>
<th>Tag</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8048118</td>
<td>●</td>
<td>N/A</td>
</tr>
<tr>
<td>eax</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>0x4(%esp)</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>0x8049128</td>
<td>●</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### Data Flow Based Type Resolution

#### Code Snippet
```
movl $0x8048118,%eax
mov %eax, 0x4(%esp)
movl $0x8049128,(%esp)
call 0x80480e0 <strcpy>
mov $0x14, %eax
int $0x80
ret
```
Data Flow Based Type Resolution

```
movl $0x8048118,%eax
mov %eax, 0x4(%esp)
movl $0x8049128,(%esp)
call 0x80480e0 <strcpy>
mov $0x14, %eax
int $0x80
ret
mov %eax, 0x8049124
```

<table>
<thead>
<tr>
<th>Mem,Reg</th>
<th>Tag</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8048118</td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>eax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x4(%esp)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x8049128</td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>(%esp)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(esp+4) → char*
(esp) → char*
strcpy(char*, char*)
# Data Flow Based Type Resolution

```assembly
movl $0x8048118, %eax
mov %eax, 0x4(%esp)
movl $0x8049128, (%esp)
call 0x80480e0 <strcpy>
mov $0x14, %eax
int $0x80
ret
mov %eax, 0x8049124
```

<table>
<thead>
<tr>
<th>Mem,Reg</th>
<th>Tag</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8048118</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>eax</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x4(%esp)</td>
<td>char*</td>
<td></td>
</tr>
<tr>
<td>0x8049128</td>
<td>N/A</td>
<td>char*</td>
</tr>
<tr>
<td>(%esp)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(esp+4) → char*
(esp) → char*
strcpy(char*, char*)
**Data Flow Based Type Resolution**

```
movl $0x8048118,%eax
mov %eax, 0x4(%esp)
movl $0x8049128,(%esp)
call 0x80480e0 <strcpy>
mov $0x14, %eax
int $0x80
ret
mov %eax, 0x8049124
```

### Mem, Reg

<table>
<thead>
<tr>
<th>Tag</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>char*</td>
<td></td>
</tr>
</tbody>
</table>

### strcpy(char*, char*)

`(esp+4) → char*`

`(esp) → char*`
Data Flow Based Type Resolution

movl $0x8048118,%eax
mov %eax, 0x4(%esp)
movl $0x8049128,(%esp)
call 0x80480e0 <strcpy>
mov $0x14, %eax
int $0x80
ret
mov %eax, 0x8049124

<table>
<thead>
<tr>
<th>Mem,Reg</th>
<th>Tag</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8048118</td>
<td></td>
<td>char*</td>
</tr>
<tr>
<td>eax</td>
<td></td>
<td>char*</td>
</tr>
<tr>
<td>0x4(%esp)</td>
<td></td>
<td>char*</td>
</tr>
<tr>
<td>0x8049128</td>
<td></td>
<td>char*</td>
</tr>
<tr>
<td>(%esp)</td>
<td></td>
<td>char*</td>
</tr>
</tbody>
</table>
## Data Flow Based Type Resolution

```
movl $0x8048118,%eax
mov %eax, 0x4(%esp)
movl $0x8049128,(%esp)
call 0x80480e0 <strcpy>
mov $0x14, %eax
int $0x80
ret
mov %eax, 0x8049124
```

<table>
<thead>
<tr>
<th>Mem,Reg</th>
<th>Tag</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8048118</td>
<td>red</td>
<td>char*</td>
</tr>
<tr>
<td>eax</td>
<td>yellow</td>
<td>imm_t</td>
</tr>
<tr>
<td>0x4(%esp)</td>
<td>red</td>
<td>char*</td>
</tr>
<tr>
<td>0x8049128</td>
<td>blue</td>
<td>char*</td>
</tr>
<tr>
<td>(%esp)</td>
<td>blue</td>
<td>char*</td>
</tr>
</tbody>
</table>
Data Flow Based Type Resolution

movl $0x8048118,%eax
mov %eax, 0x4(%esp)
movl $0x8049128,(%esp)
call 0x80480e0 <strcpy>
mov $0x14, %eax
int $0x80
ret
mov %eax, 0x8049124

Mem,Reg | Tag | Type
---|---|---
0x8048118 | char* |
eax | imm_t |
0x4(%esp) | char* |
0x8049128 | char* |
(%esp) | char* |

getpid eax → pid_t
Data Flow Based Type Resolution

```c
movl $0x8048118,%eax
mov %eax, 0x4(%esp)
movl $0x8049128,(%esp)
call 0x80480e0 <strcpy>
mov $0x14, %eax
int $0x80
ret
mov %eax, 0x8049124
```

<table>
<thead>
<tr>
<th>Mem,Reg</th>
<th>Tag</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8048118</td>
<td></td>
<td>char*</td>
</tr>
<tr>
<td>eax</td>
<td>green</td>
<td>pid_t</td>
</tr>
<tr>
<td>0x4(%esp)</td>
<td></td>
<td>char*</td>
</tr>
<tr>
<td>0x8049128</td>
<td>blue</td>
<td>char*</td>
</tr>
<tr>
<td>(%esp)</td>
<td></td>
<td>char*</td>
</tr>
</tbody>
</table>

getpid  eax → pid_t
Data Flow Based Type Resolution

```
movl $0x8048118,%eax
mov %eax, 0x4(%esp)
movl $0x8049128,(%esp)
call 0x80480e0 <strcpy>
mov $0x14, %eax
int $0x80
ret
mov %eax, 0x8049124
```
Data Flow Based Type Resolution

```
movl $0x8048118,%eax
mov %eax, 0x4(%esp)
movl $0x8049128,(%esp)
call 0x80480e0 <strcpy>
mov $0x14, %eax
int $0x80
ret
mov %eax, 0x8049124
```

<table>
<thead>
<tr>
<th>Mem,Reg</th>
<th>Tag</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8048118</td>
<td>●</td>
<td>char*</td>
</tr>
<tr>
<td>eax</td>
<td>●</td>
<td>pid_t</td>
</tr>
<tr>
<td>0x4(%esp)</td>
<td>●</td>
<td>char*</td>
</tr>
<tr>
<td>0x8049128</td>
<td>●</td>
<td>char*</td>
</tr>
<tr>
<td>(%esp)</td>
<td>●</td>
<td>char*</td>
</tr>
<tr>
<td>0x8049124</td>
<td>●</td>
<td>pid_t</td>
</tr>
</tbody>
</table>
Outline

1. Shadow Memory
2. Example: Type Inference
3. Implementation
   - Valgrind
   - QEMU
   - PIN
4. Summary
Valgrind

UCodeBlock* SK_(instrument)(UCodeBlock* cb_in, Addr orig_addr)
{
    UCodeBlock* cb;
    ...
    switch (u_in->opcode) {
        case LOAD:
            VG_(ccall_RR_R) (cb, (Addr) HELPER_bdd_load ,
                u_in->val1, SHADOW(u_in->val1), SHADOW(u_in->val2), 2);
            break;
    ...
    }
}

bdd HELPER_bdd_load(Addr a, bdd addr_bdd)
{
    bdd mem_bdd = get_ii_vbytes4_ALIGNED(a);
    bdd_allsat (mem_bdd, allsatPrintHandler);
    return mem_bdd;
}

...
Outline

1. Shadow Memory
2. Example: Type Inference
3. Implementation
   - Valgrind
   - QEMU
   - PIN
4. Summary
static target_ulong disas_insn(DisasContext *s, target_ulong pc_start)
{
    /* inc, dec, and other misc arith */
    case 0x40 ... 0x47: /* inc Gv */
        ot = dflag ? OT_LONG : OT_WORD;
        gen_inc(s, ot, OR_EAX + (b & 7), 1);
        break;
    case 0x48 ... 0x4f: /* dec Gv */
        ot = dflag ? OT_LONG : OT_WORD;
        gen_inc(s, ot, OR_EAX + (b & 7), -1);
        break;
    case 0x134: /* sysenter */
        gen_helper_sysenter();
        break;
}

static void gen_inc(DisasContext *s1, int ot, int d, int c)
{
    if (d != OR_TMP0)
        gen_op_mov_TN_reg(ot, 0, d);
    else
        gen_op_ld_T0_A0(ot + s1->mem_index);
}

void helper_sysenter(void)
{
    ESP = env->sysenter_esp;
    EIP = env->sysenter_eip;
}
...

QEMU
Outline

1. Shadow Memory
2. Example: Type Inference
3. Implementation
   - Valgrind
   - QEMU
   - PIN
4. Summary
main()
{
    ...
    INS_AddInstrumentFunction(SetupDataflow, 0);
    setup_inst_hook();
    ...
}
void SetupDataflow(INS ins, void *v)
{
    xed_iclass_t opcode = (xed_iclass_t) INS_Opcode(ins);
    (*instrument_functions[opcode])(ins, v);
}
void setup_hook()
{
    for(int i = 0; i < XED_ICLASS_LAST; i++) {
        instrument_functions[i] = &UnimplementedInstruction;
    }
    instrument_functions[XED_ICLASS_ADD] = &Instrument_ADD;
}
static void Instrument_MOV(INS ins, void *v)
{
    //1. R -> R | M
    if(INS_OperandIsReg(ins, 1)) {
        INS_InsertCall(ins, IPOINT_BEFORE, AFUNPTR(GetRegTag),
                       IARG_ADDRINT, INS_OperandReg(ins, 1),
                       IARG_PTR, &reg_tag_src,
                       IARG_END);
    }
}
Outline

1. Shadow Memory
2. Example: Type Inference
3. Implementation
   - Valgrind
   - QEMU
   - PIN
4. Summary
Summary

Key steps

- Designing shadow memory
- Instrument each instruction
- Generate or propagate data flow facts
- Query data flow facts