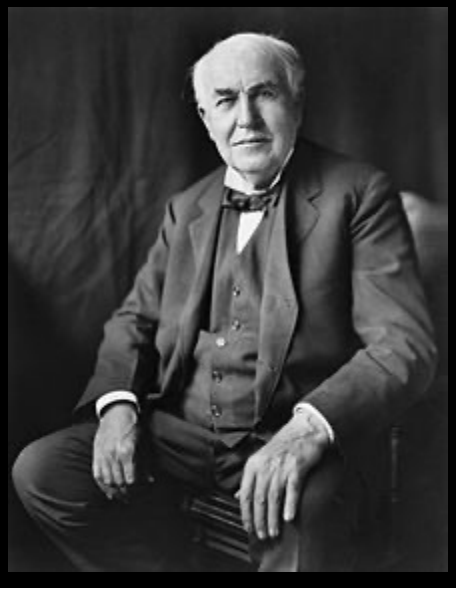
Identifying Arbitrary Memory Access Vulnerabilities in Privilege-Separated Software

Hong Hu, Zheng Leong Chua, Zhenkai Liang,
Prateek Saxena

National University of Singapore

20th European Symposium on Research in Computer Security



Genius is 1% inspiration, 99% perspiration.

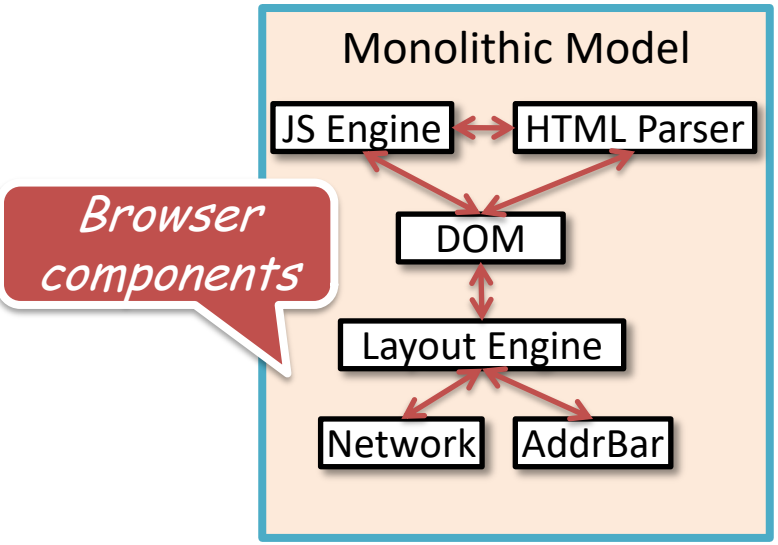
But the 1% inspiration is the most important.

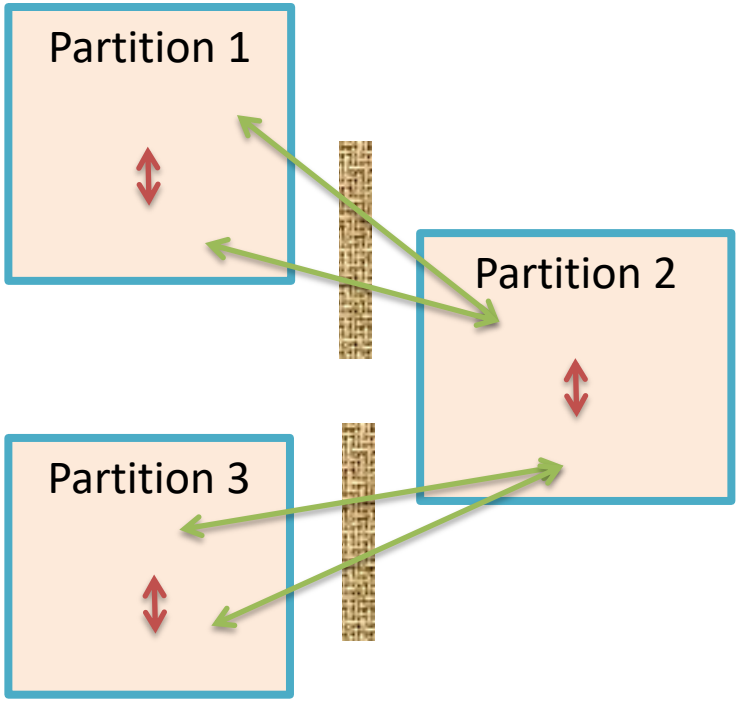
(Thomas Edison)

- *Out-of-context* problem in human language
  - Isolated sentences, phrases
  - Leads to misunderstanding.
- Program code *Out-of-context* ?

Privilege Separation

- A program: set of components



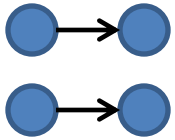


- Privilege Separation
  - new trust relationship
  - memory isolation as protection
  - enough ?

Memory Access Capability

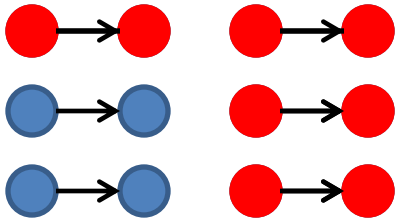
- In-context capability: Cap_{in}
  - necessary operations

```
if( (src == src1 && dst == dst1) ||
    (src == src2 && dst == dst2) )
        memcpy(dst, src);
```



- Out-of-context capability: $Cap_{out} \supseteq Cap_{in}$
  - less limitation

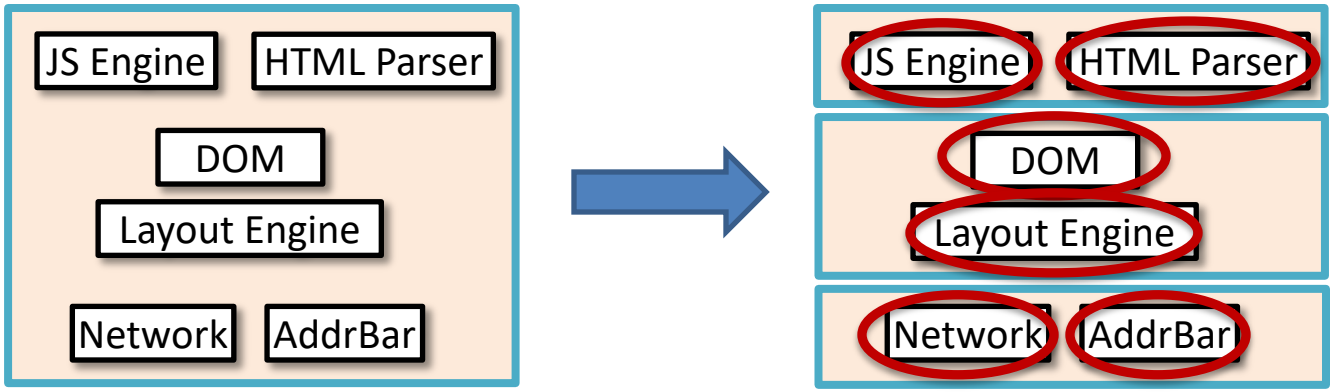
```
memcpy(dst, src);
```



- $Cap_{out} - Cap_{in}$: unnecessary -> errors

Our Approach

- Systematic method
  - evaluate each component separately



- Application on real-world isolations

Motivating Example

```
struct subobj { ... } * p_sub;
struct object { ...
        struct subobj * sub;
} * p_obj;

int main() {
    p_obj = create_object();
    p_sub = create_subobj();
    p_obj->sub = p_sub;
}
```

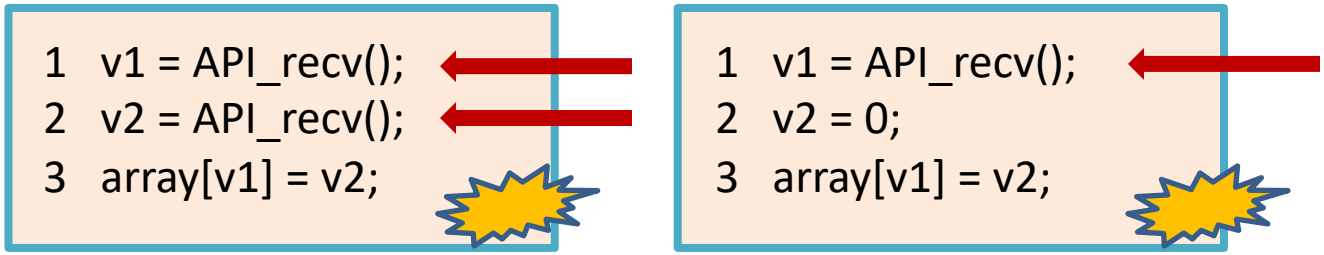
```
// create an object instance
// and return its pointer
struct object * create_object()
{ ... }

// create a subobj instance
// and return its pointer
struct subobj * create_subobj()
{ ... }
```

- *Dereference Under the Influence (DUI)*
  - memory access affected by inputs
  - allow attacks to bypass memory isolation

Write DUI

- Memory write influenced by input
  - memory write address depends on inputs
  - value depends on inputs, or deterministic

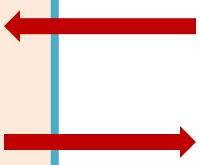


  - corrupt control data / non-control data
    - e.g., return address, function pointer, user id

Read DUI

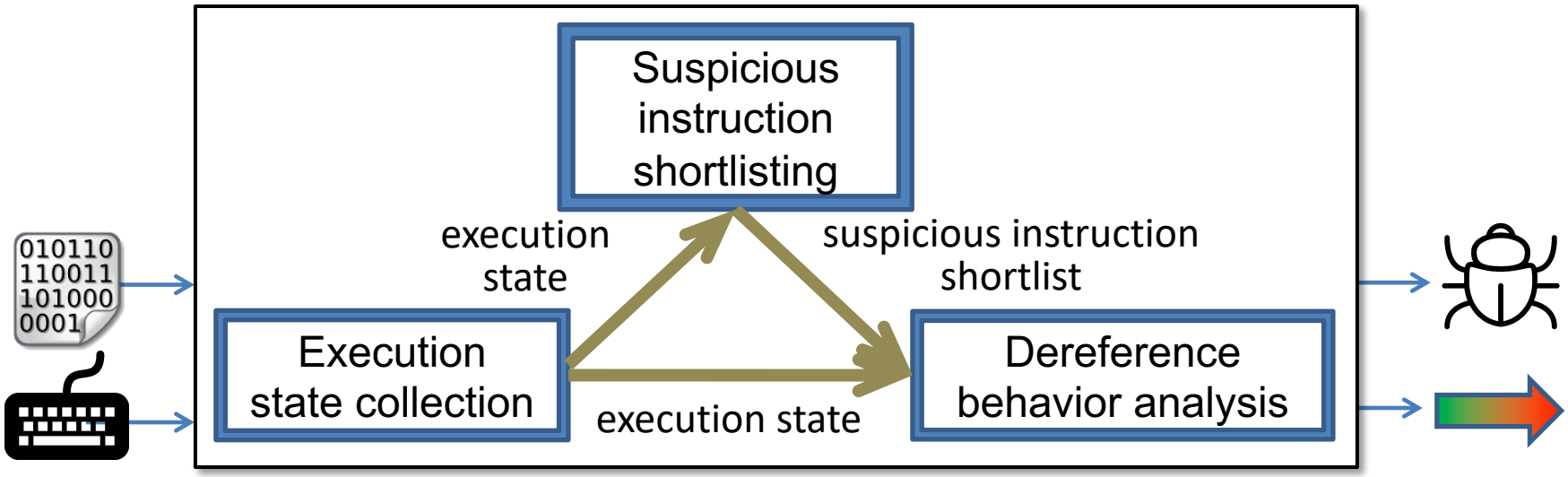
- Memory read influenced by input
  - memory read address depends on inputs
  - retrieved value is sent out

```
1  v1    = API_recv();
2  data = array[v1];
3  API_send(data);
```



  - leak sensitive information
    - e.g., password, private key
    - e.g., stack canary, randomized address, CFI tags

DUI Detector



- Input:
  - Protected components & Sample inputs
- Output
  - DUI vulnerability & severity

Suspicious Inst. Shortlisting

- Data dependency analysis to track input

- Write DUI detection
  - Memory writing instruction
  - Tainted/fixed src operand
  - Tainted base/index address of the dst operand

```
mov  %eax,  (%esi)
```

```
add  %ebx,  (%esi, %ecx, 2)
```

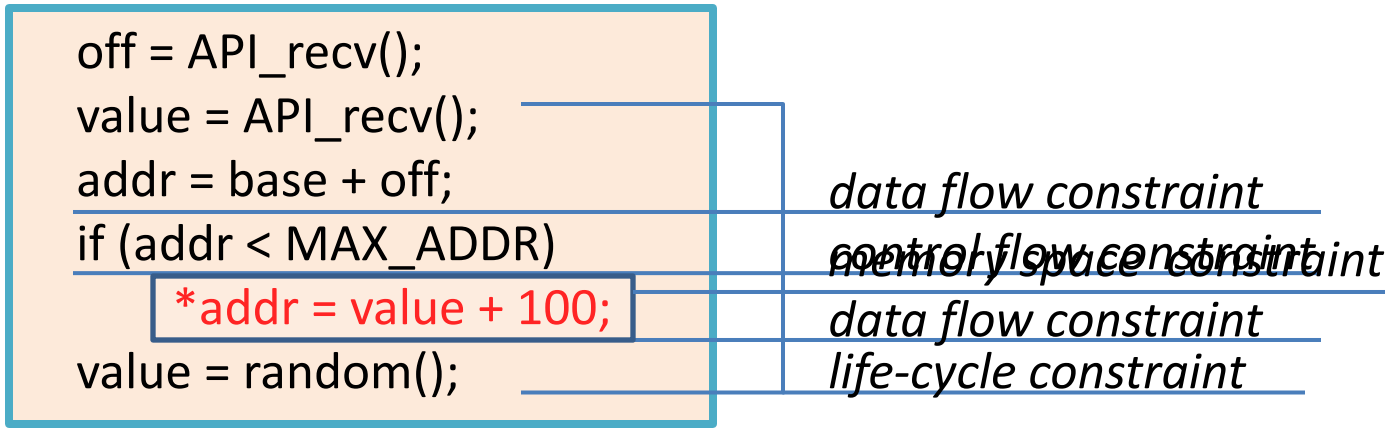
Suspicious Inst. Shortlisting

- Data dependency analysis to track input

- Read DUI detection
  - Memory read operation
  - Tainted base/index address of the src operand
  - Result is used at sinks
    - e.g., send(), cross-partition call/return, etc

```
mov (%esi), %eax
……
sink(%eax)
```

Dereference Behavior Analysis

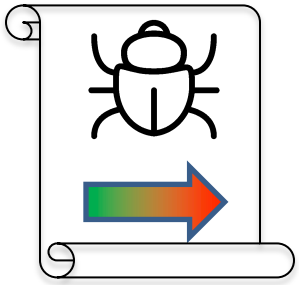


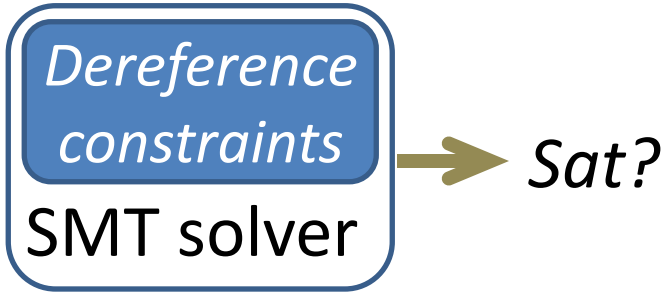
- Capture constraints on inputs
  - data flow constraints
  - control flow constraints
  - memory space constraints
  - data life-cycle constraints

Dereference constraints

Dereference Behavior Analysis

- Attacker's memory access capability *Cap*
  - small |*Cap*| ==> almost non-exploitable
  - larger |*Cap*| ==> more severe
- *Cap* Estimation
  - initial target analysis
    - memory space constraints (R/W permission)
  - bit pattern analysis
    - e.g., address & 0x11 == 0x01
  - range analysis
    - e.g., ∄/∀ address ∈ Range ➔ sat



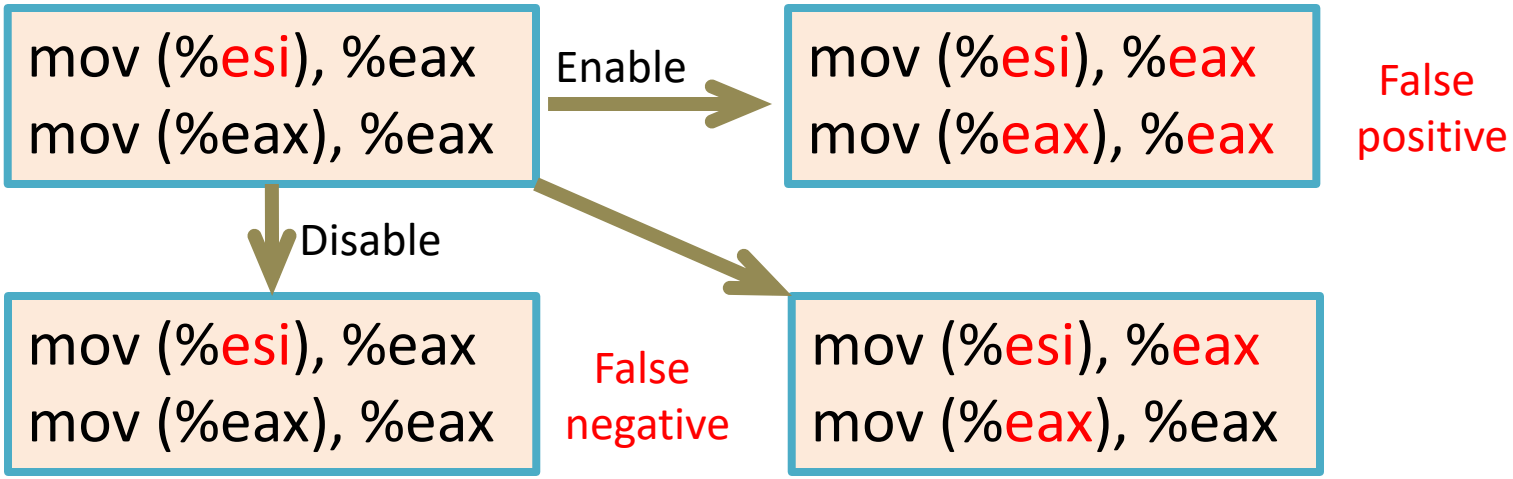


Implementation (1)

- Taint Propagation
  - fine-gained taint record propagation

```
add %eax,  %ebx
```

    - T_source(%ebx) = T_source(%eax) ∪ T_source(%ebx)
  - 1-level table lookup

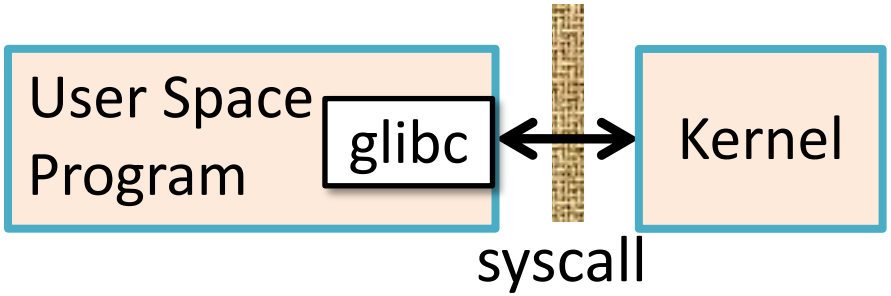


Implementation (2)

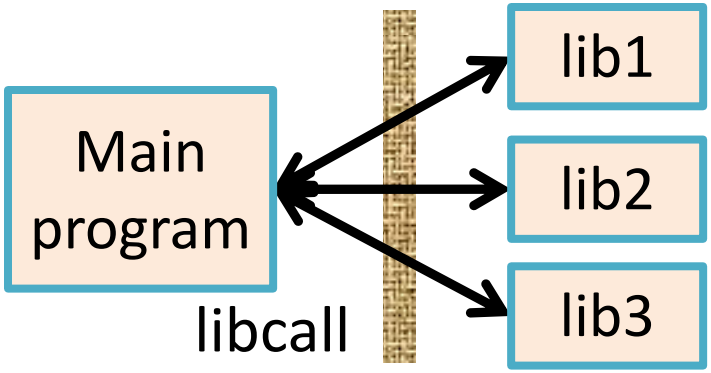
- Constraints capture
  - memory space constraint
    - log module loading/unloading event
      - malloc, free
    - restore memory layout for each instruction
  - data life-cycle constraint
    - next instruction update a location

Evaluation

- Isolation schemes
  - user/kernel isolation (e.g., overshadow)



  - main-code/library isolation (e.g., codejail)



Write DUI in user/kernel Isolation

- *glibc* code handling *brk()*

```
1   addr1 = brk(0);
2   addr2 = brk(argument);
3   *(addr1 + 4) = addr2 - addr1;
```

```
1  mov %eax, 0x4(%edx)
2  ...
3  mov %eax, 0x4(%edi);
```

  - setup heap region
  - line 1: overwritten immediately (|*cap*| = 0)
  - line 3: write DUI

```
condition( brk1%8 == 0 && brk2 > brk1 )

address = brk1 + 0x2718
data = (brk2 - address) | 0x1
```

Write DUI in user/kernel Isolation

- Other paths

```
condition(brk1%8 != 0 && brk1<brk2<brk3)

address : relies on brk1;
data : relies on brk1 and brk2;
```

```
condition(brk1%8 != 0 && brk1<brk2>brk3)

address : relies on brk1;
data : relies on brk1 and brk3;
```

  - Systematic method *vs* manual analysis
- *glibc* code handling *mmap2()*

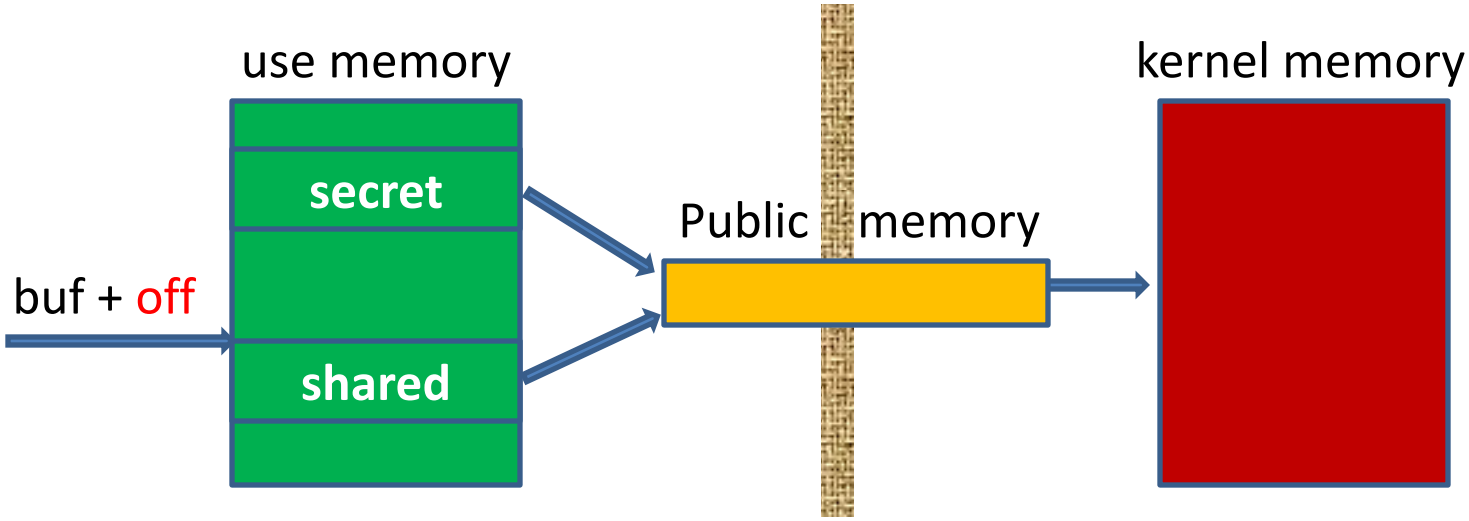
*Stephen Checkoway and Hovav Shacham. **Iago attacks: why the system call API is a bad untrusted RPC interface**. In *ASPLOS* 2013.

Read DUI in user/kernel Isolation

- *cat* code handling *read()/write()*

```
1  nr = read(rfd, buf, size);
2  for (off = 0; nr; nr -= nw, off += nw)  {
3      nw = write(wfd,  buf + off,  nr);
4      if (nw == 0 || nw == -1)
5          goto error;
6  }
```

Pass by copy



Write DUI in main-code/library Isolation

- Programs Using *libsdl*

```
1  screen = SDL_SetVideoMode(...);      // get framebuffer surface
2  color = SDL_MapRGB(...);             // get a pixel value
3  pixmem16 = screen->pixels + x + y * pixelsperline ;
4                                       // get pixel address
5  *pixmem16 = color;                   // set the color
```

  - write DUI one line 5
  - no limitation on attackers

Conclusion

- Privilege separation leaves memory errors
  - *Dereference Under the Influence (DUI)*
- DUI Detector
  - systematic way to detect DUI
- Application
  - user/kernel isolation, library isolation
  - write/read DUIs

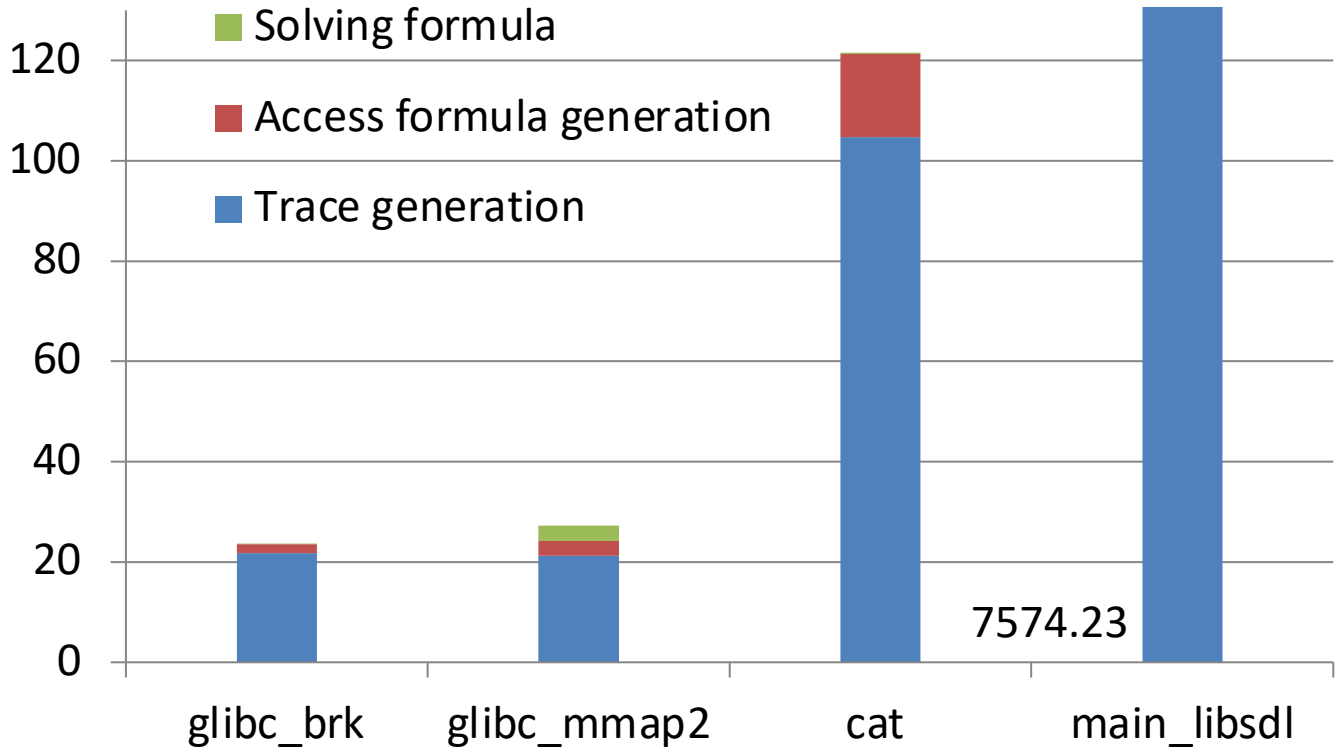
Thanks!

Hong Hu

huhong@comp.nus.edu.sg

http://www.comp.nus.edu.sg/~huhong/

Performance



- DUIs are detected within several minutes
- Trace generation takes most of the time
- Depends on the trace size