Reverse Engineering of Data Structures and Types

The 17th Annual Network and Distributed System Security Symposium

Automatic Reverse Engineering of Program Data Structures from Binary Execution

Zhiqiang Lin
Xiangyu Zhang, Dongyan Xu

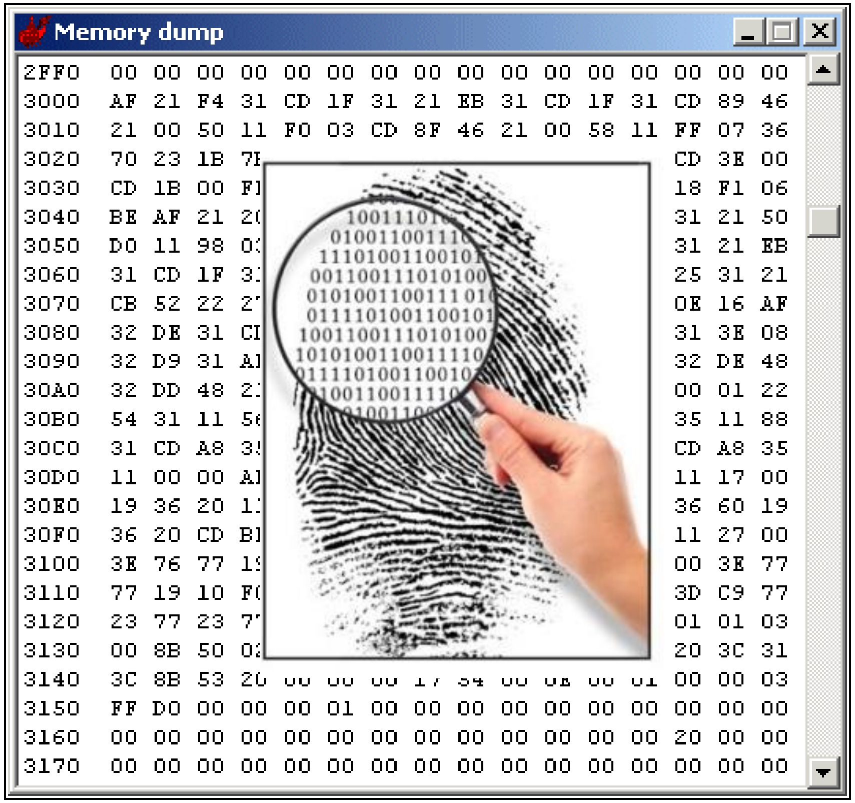
Dept. of Computer Science and CERIAS
Purdue University

March 3rd, 2010

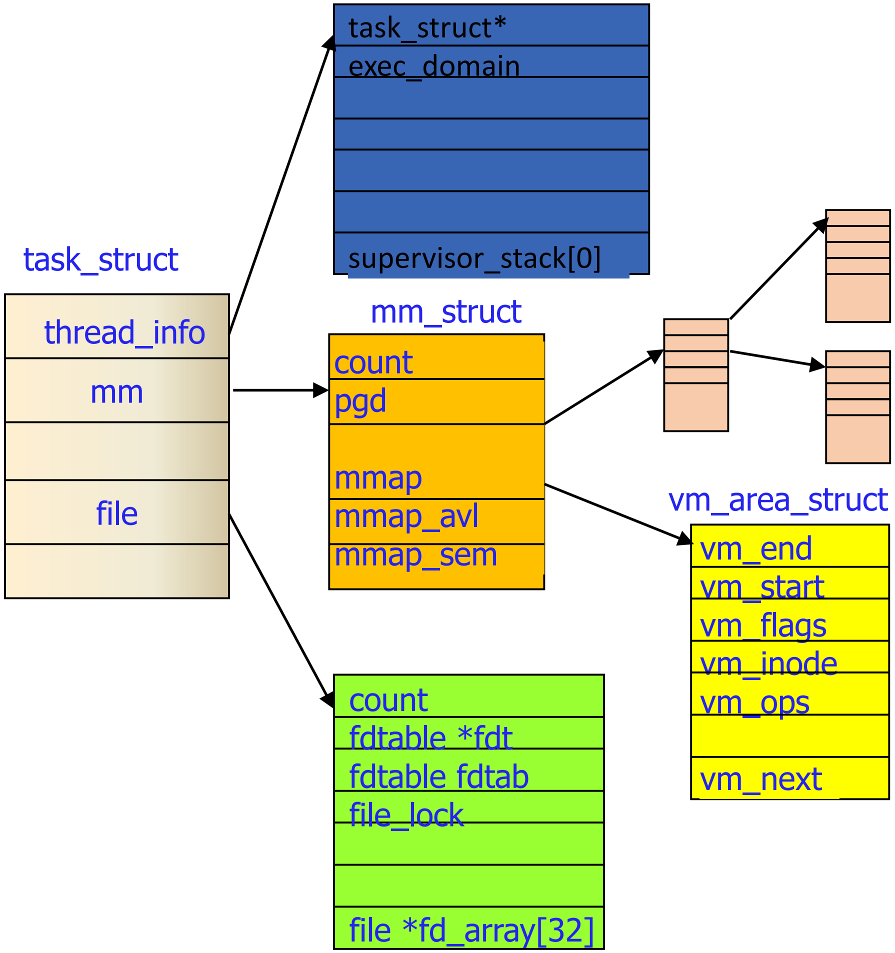
Problem Definition

- Recover data structure specifications from binary
    - Syntactic
        - Layout
        - Offset
        - Size
    - Semantic
        - ip_addr_t, pid_t, …
        - input_t
        - malloc_arg_t, format_string_t…

Security Applications -- Memory Forensics



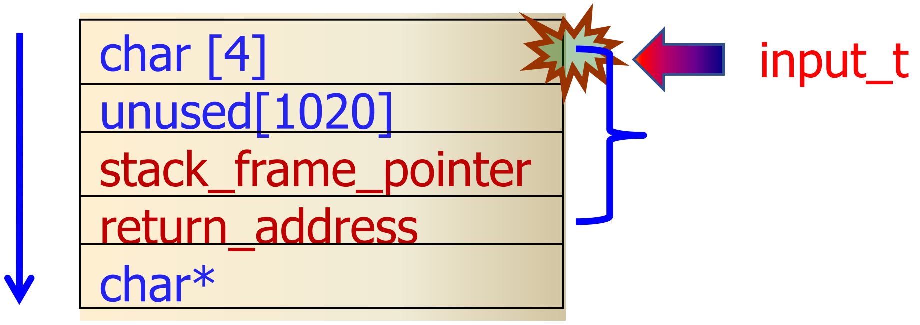
```
2FF0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3000  AF 21 F4 31 CD 1F 31 21 EB 31 CD 1F 31 CD 89 46
3010  21 00 50 11 F0 03 CD 8F 46 21 00 58 11 FF 07 36
3020  70 23 1B                                CD 3E 00
3030  CD 1B 00                                18 F1 06
3040  BE AF 21                                31 21 50
3050  D0 11 98                                31 21 EB
3060  31 CD 1F                                25 31 21
3070  CB 52 22                                0E 16 AF
3080  32 DE 31                                31 3E 08
3090  32 D9 31                                32 DE 48
30A0  32 DD 48                                00 01 22
30B0  54 31 11                                35 11 88
30C0  31 CD A8                                CD A8 35
30D0  11 00 00                                11 17 00
30E0  19 36 20                                36 60 19
30F0  36 20 CD                                11 27 00
3100  3E 76 77                                00 3E 77
3110  77 19 10                                3D C9 77
3120  23 77 23                                01 01 03
3130  00 8B 50                                20 3C 31
3140  3C 8B 53 20 00 00 00 17 34 00 0E 00 01 00 00 03
3150  FF D0 00 00 00 01 00 00 00 00 00 00 00 00 00 00
3160  00 00 00 00 00 00 00 00 00 00 00 00 00 20 00 00
3170  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```



Security Applications -- Vulnerability Discovery

```
void main(int argc, char* argv[])
{
        char    tempname[1024];
        strcpy(tempname, argv[1]);
  }
```

```
$ ./a.out aaa'\n'
```



char [4]

unused[1020]

stack_frame_pointer

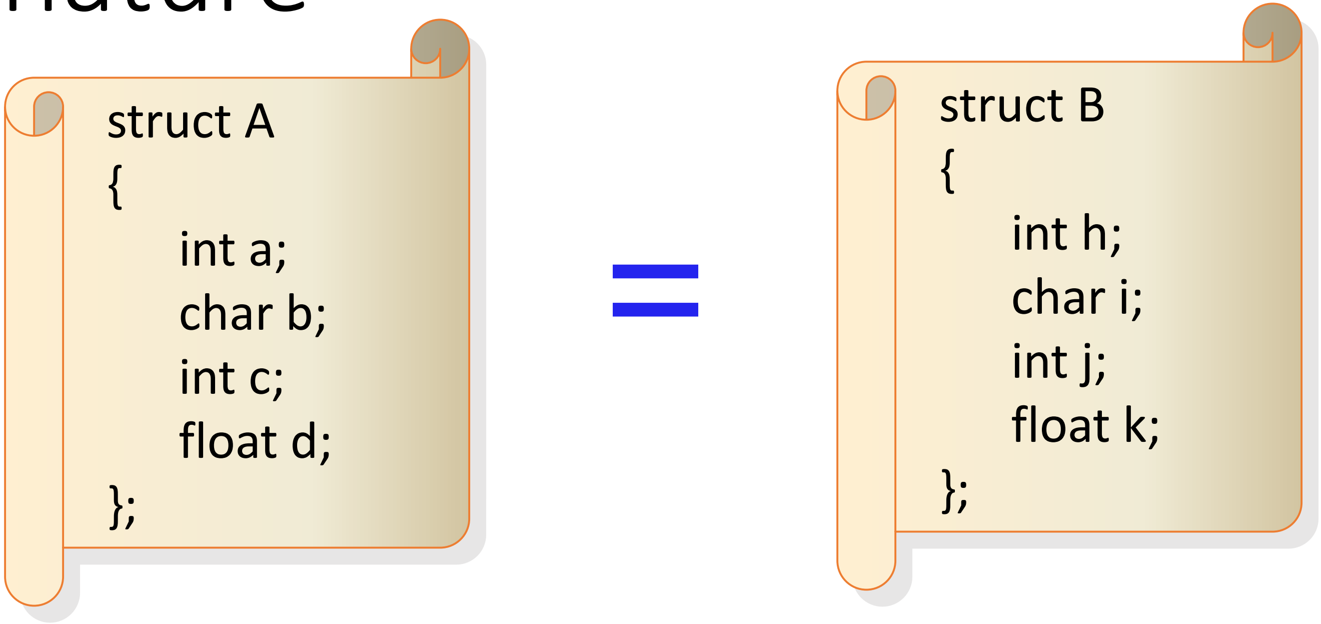
return_address

char*

input_t

```
$ ./a.out aaaaaaaaaaaaaaaa…a'\n'
```

Security Applications -- Program Signature

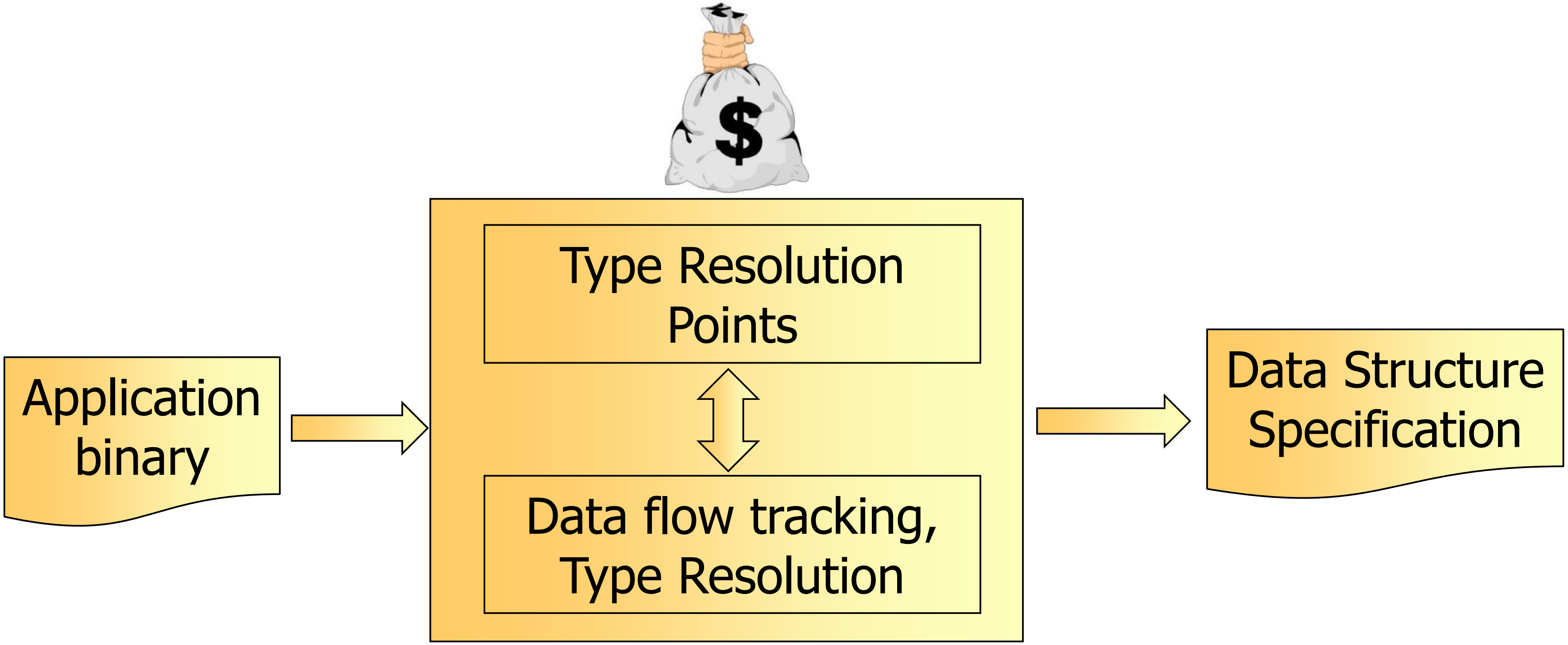


```
struct A
{
      int a;
      char b;
      int c;
      float d;
};
```

```
struct B
{
      int h;
      char i;
      int j;
      float k;
};
```

Laika [Cozzie et al., OSDI'08] Using Data structure as a classifier → Malware Signature

Overview



Application binary

Type Resolution Points

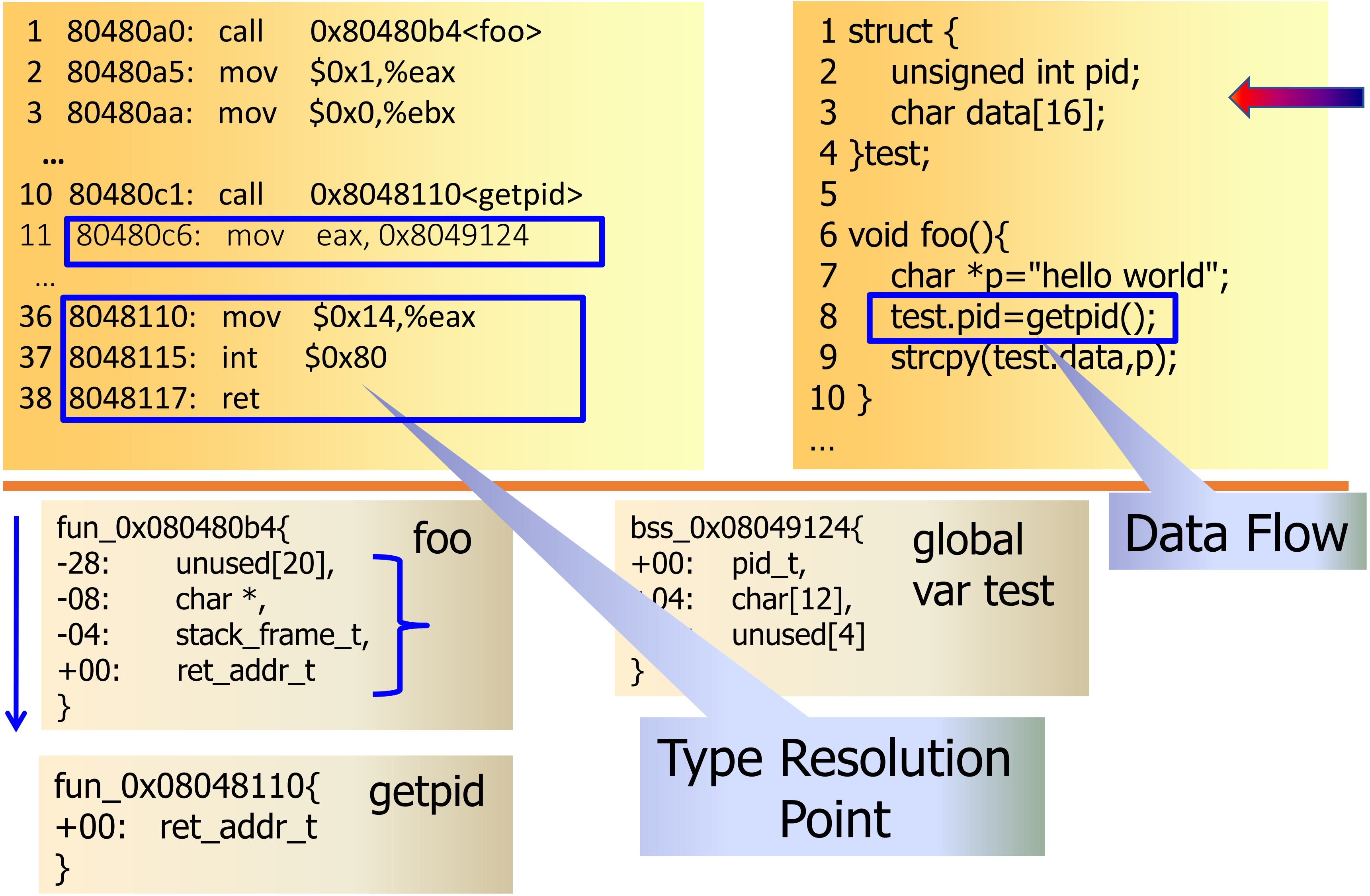
Data flow tracking, Type Resolution

Data Structure Specification

REWARDS

Reverse **E**ngineering **W**ork for **A**utomatic **R**evelation of **D**ata **S**tructures

Key Ideas



```
80480a0:   call     0x80480b4<foo>
80480a5:   mov      $0x1,%eax
80480aa:   mov      $0x0,%ebx
 …
80480c1:   call     0x8048110<getpid>
80480c6:   mov      eax, 0x8049124
 …
8048110:   mov      $0x14,%eax
8048115:   int      $0x80
8048117:   ret
```

```
struct {
    unsigned int pid;
    char data[16];
}test;

void foo(){
    char *p="hello world";
    test.pid=getpid();
    strcpy(test.data,p);
}
…
```

```
fun_0x080480b4{
-28:      unused[20],
-08:      char *,
-04:      stack_frame_t,
+00:      ret_addr_t
}
```

```
fun_0x08048110{
+00:   ret_addr_t
}
```

```
bss_0x08049124{
+00:    pid_t,
+04:    char[12],
+16:    unused[4]
}
```

Type Resolution Point - I

```
<getpid>
8048110:   mov    $0x14,%eax
8048115:   int    $0x80
8048117:   ret
```

- System calls
  - Syscall num →
    - Syscall_enter: Type parameter passing registers (i.e., ebx, ecx, edx, esi, edi, and ebp) if they involved
    - Syscall_exit: type return value (eax)

Type Resolution Point - II

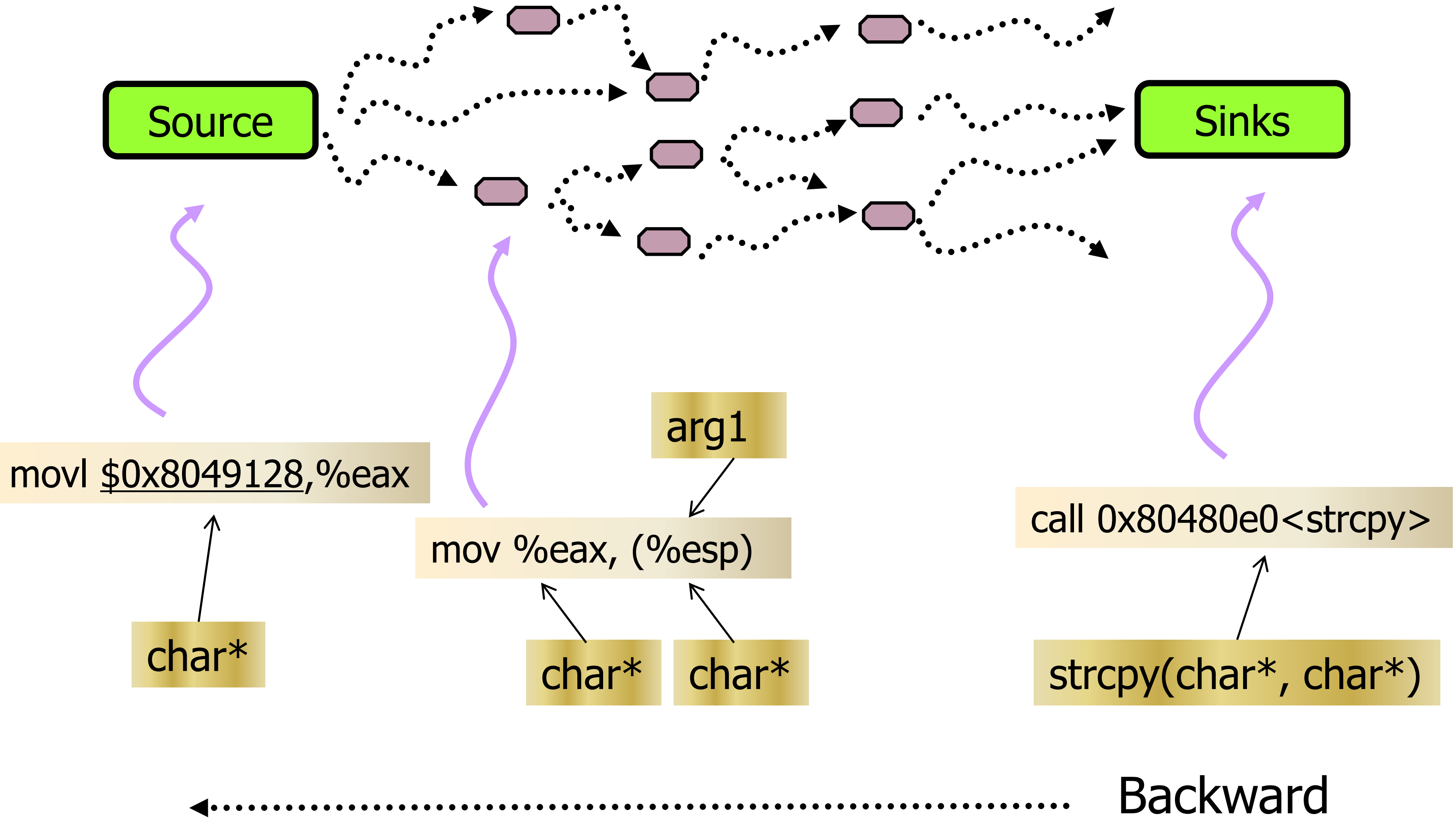
```
80480ce:    mov %eax,0x4(%esp)              <arg2>
80480d2:    movl $0x8049128, (%esp)         <arg1>
80480d9:    call 0x80480e0                  <strcpy>
```

- Standard Library Call (API)
  - Type corresponding argument and return value
  - More wealth than System call
    - 2016 APIs in Libc.so.6 vs. 289 sys call (2.6.15)

Type Resolution Point - III

- Other type revealing instructions in binary code
  - String related (e.g., MOVS/B/D/W, LOADS/STOS/B/D/W)
  - Floating-point related (e.g., FADD, FABS, FST)
  - Pointer-related (e.g., MOV (%edx), %ebx)

Backward Type Resolution



Source

Sinks

movl $0x8049128,%eax

char*

arg1

mov %eax, (%esp)

char*

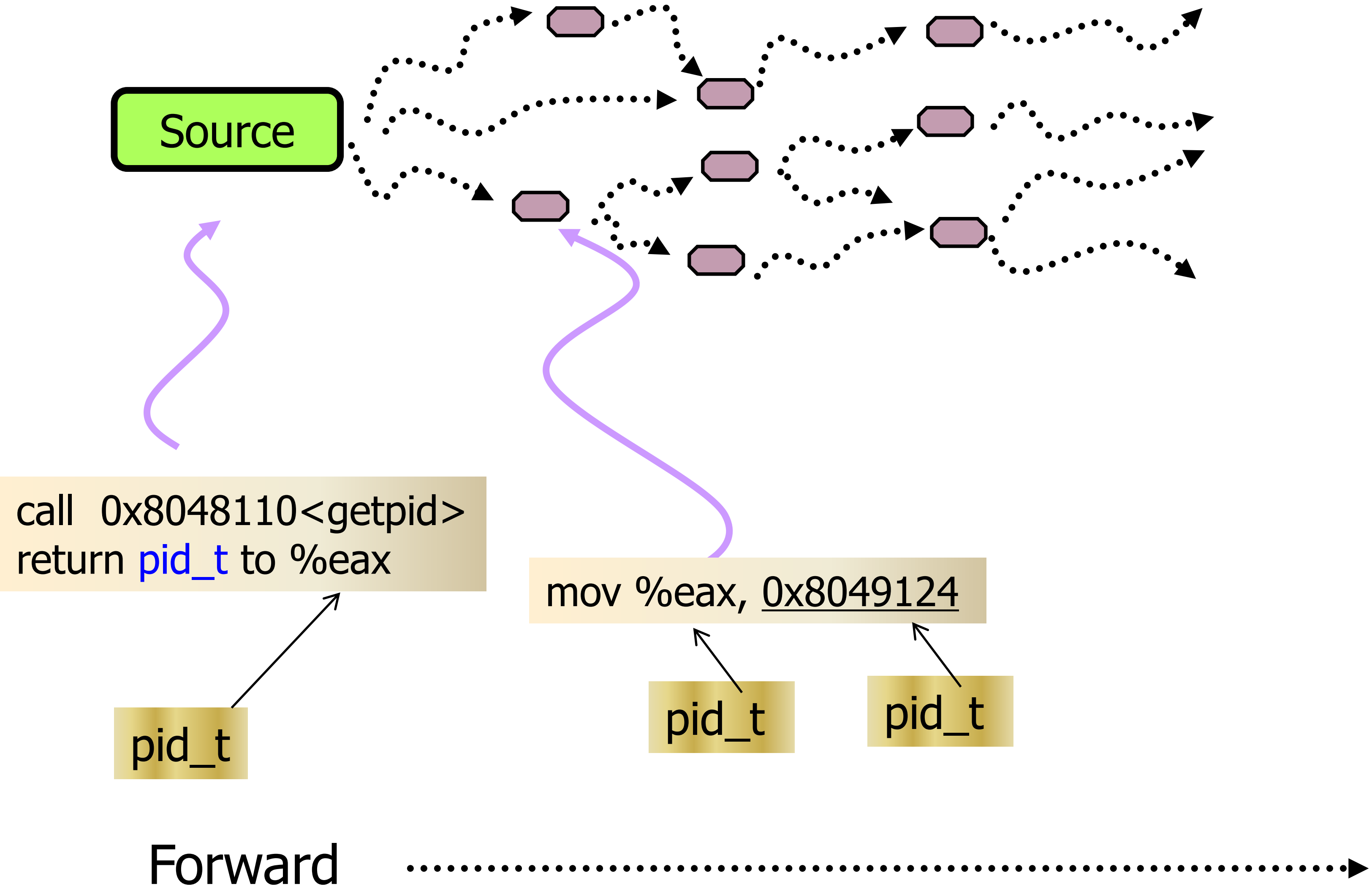
char*

call 0x80480e0<strcpy>

strcpy(char*, char*)

Backward

Forward Type Resolution



Source

call 0x8048110<getpid>
return pid_t to %eax

pid_t

mov %eax, 0x8049124

pid_t

pid_t

Forward

Data Flow Propagation

- Similar to taint analysis, using shadow memory to keep the variable attributes and track the propagation

- New challenges
  - Two-way type resolution
  - Dynamic life time of stack and heap variables
    - Memory locations will be reused
  - Multiple instances of the same type

Implementations

- Dynamic Binary Instrumentation
  - A pin-tool on top of Pin-2.6
    - http://www.pintool.org/

PIN

QEMU open source processor emulator

Valgrind

Dyninst

- Data structures of interest:
  - File information
  - Network communication
  - Process information
  - Input-influenced (for vulnerability discovery)

Evaluations

- Experiment set up
  - 10 utility binaries (e.g., ls, ps, ping, netstat…)
  - Linux 2.6.18, gcc-3.4, 2G mem

- Ground truth → debug information

Experimental Results

- False negative: the data structure we missed (compared with the ground truth)
  - Global:70%
  - Heap: 55%
  - Stack: 60%

- Why false negative
  - Dynamic analysis

Experimental Results

- False positive (we get the wrong data type)
    - Global: 3%
    - Heap: 0%
    - Stack: 15%

Sources of False Positives

- Loss of data structure hierarchy
  - There is no corresponding type resolution point with the same hierarchical type
  - Heuristics exist (e.g., AutoFormat [Lin et al, NDSS'08])
- Path-sensitive memory reuse
  - Compiler might assign different local variables declared in different program paths to the same memory address
  - Path-sensitive analysis
- Type cast
  - int a=(float)b;

Application I: Memory Forensics

```
...
08052170  b0 5b fe b7 b0 5b fe b7  05 00 00 00 02 00 92 7e
08052180  0a 00 00 0b 00 00 00 00  00 00 00 00 c7 b0 af 4a
08052190  c7 b0 af 4a 00 00 00 00  58 2a 05 08 00 00 00 00
080521a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
...
```

```
struct 0x804dd4f {
00:     pthread_t;                 b0 5b fe b7
04:     int;                       b0 5b fe b7
08:     socket;                    05 00 00 00
12:     struct sockaddr_in;        02 00 92 7e 0a 00 00 0b .. 00 00
28:     time_t;                    c7 b0 af 4a
32:     time_t;                    c7 b0 af 4a
36:     unused[4];                 00 00 00 00
40:     struct 0x804ddfb*;         58 2a 05 08
};
```

Application I: Memory Forensics

```
...
08052170  b0 5b fe b7 b0 5b fe b7  05 00 00 00 02 00 92 7e
08052180  0a 00 00 0b 00 00 00 00  00 00 00 00 c7 b0 af 4a
08052190  c7 b0 af 4a 00 00 00 00  58 2a 05 08 00 00 00 00
080521a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
...
```

```
struct 0x804dd4f {
00:     pthread_t;
04:     int;
08:     socket;
0c:     struct sockaddr_in;
28:     time_t;
32:     time_t;
36:     unused[4];
40:     struct 0x804ddfb*;
};
```

```
{
00:     sin_family;            02 00
02:     sin_port;              92 7e
04:     sin_addr;              0a 00 00 0b
08:     sin_zero;              00 00 00 00 00 00 00 00
};
```

02 00 92 7e 0a 00 00 0b .. 00 00

ip_addr_t: 10.0.0.11

Application II: Vulnerability Discovery

Program	buffer overflow		integer overflow		Format string	
	#total	#real	#total	#real	#total	#real
ncompress-4.2.4	1	1	0	0	0	0
bftpd-1.0.11	3	1	0	0	0	0
gzip-1.2.4	3	1	0	0	0	0
nullhttpd-0.5.0	5	1	2	1	0	0
xzgv-5.8	3	0	8	1	0	0
gnuPG-1.4.3	0	1	3	1	0	0
ipgrab-0.9.9	0	1	5	1	0	0
cfingerd-1.4.3	4	0	0	0	1	1
ngircd-0.8.2	12	0	0	0	1	1

Suspects

True

Discussion

- Dynamic analysis
  - Full coverage of data structures
- Only user-level data structure
  - OS kernel
- Obfuscated binary can cheat REWARDS
  - No library call
  - Memory cast
- Lack of other application-specific type resolution points
  - Other APIs
    - E.g., browser-related

Summary

- REWARDS
  - Binary only
  - Dynamic analysis
  - Data flow tracking
- Key insight
  - Using system call/API/Type revealing instruction as type resolution point
  - Two-way type resolution
  - Unique benefits to memory forensics and vulnerability discovery

The 17th Annual Network and Distributed System Security Symposium

Automatic Reverse Engineering of Program Data Structures from Binary Execution

Zhiqiang Lin
Xiangyu Zhang, Dongyan Xu

Dept. of Computer Science and CERIAS
Purdue University

March 3rd, 2010

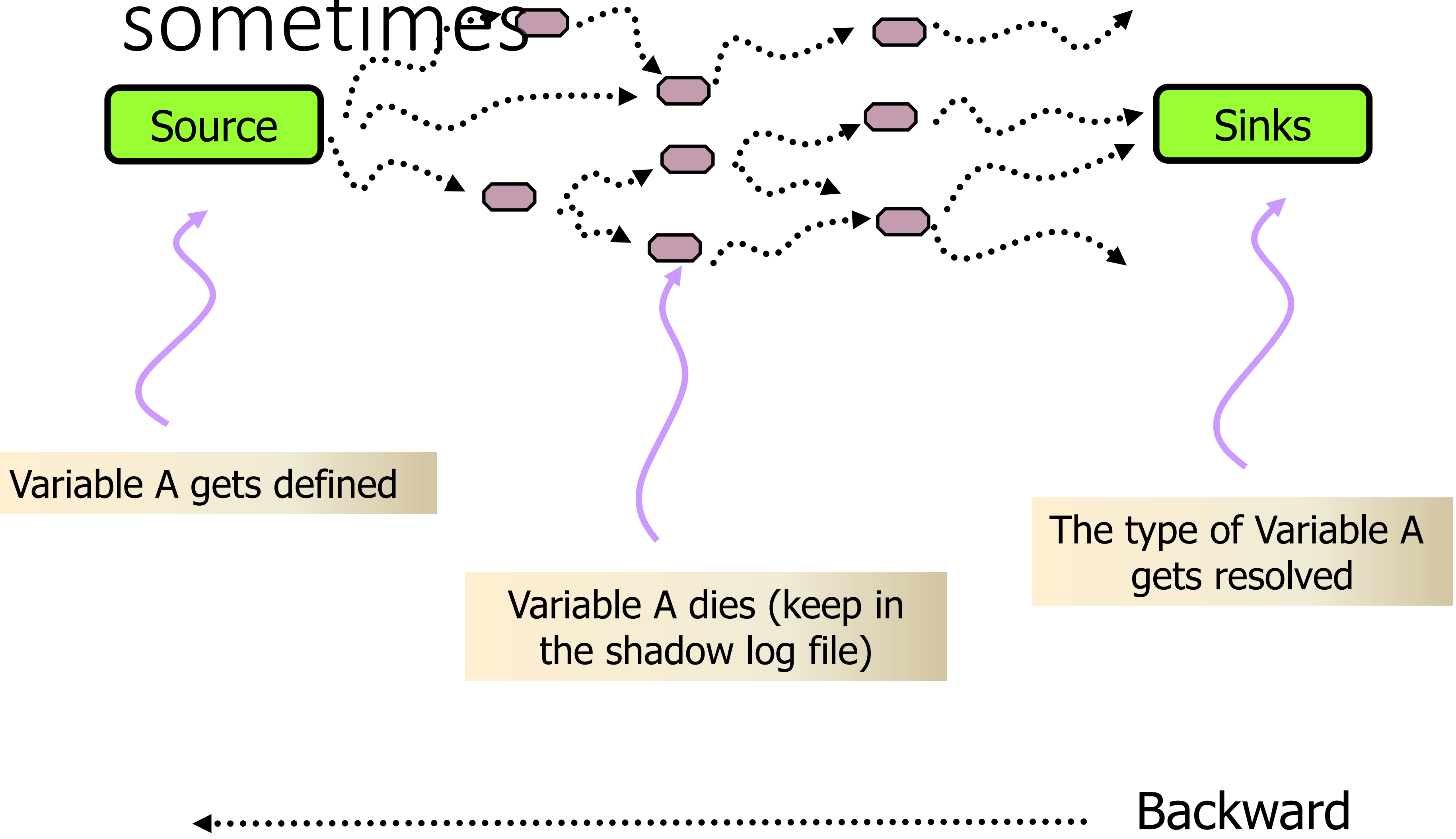
Backup: Path-sensitive memory reuse

```
        case 'e': {
            if (strlen(params[0]) > 1) switch (tolower(params[0][1])) {
                case 's': {
                    struct escan_rec rc;
                    if (num_params <= 2) Log("escan <dest>\n");
                    else {
                        memset((void*)&rc,0,sizeof(struct escan_rec));

if (udp.len != 0) if (!audp_recv(&udp,&tmpclient,buf,3000)) {
    struct header *rc;
    struct llheader *llrc;
    char k=0;
    buf+=sizeof(struct llheader);
    udp.len-=sizeof(struct llheader);
    llrc=(struct llheader *)(buf-sizeof(struct llheader));
    rc=(struct header *)buf;
    if (llrc->type == 0) {
        struct llheader ll;
        memset((void*)&ll,0,sizeof(struct llheader));
```

(pudclient.c)

Backup: offline is needed sometimes



Source

Sinks

Variable A gets defined

Variable A dies (keep in the shadow log file)

The type of Variable A gets resolved

Backward

Backup: Vulnerability Discovery

- Buffer overflow

```
fun_0x08048e76{
- 1052:   char[13],
- 1039:   unused[1023],...
- 0004:   stack_frame_t,
+0000:    ret_addr_t,
+0004:    char*}
```

input_t argv

- Format string

```
fun 0x0805f9a5 { ...,
- 0284:  format_string_t[76],
- 0208:  unused[204],
- 0004:  stack_frame_t,
+0000:   ret_addr_t,...}
```

input_t recv

- Integer overflow

```
bss 0x0809ac80 { ...
+91952:   int (malloc_arg_t),
+91956:   int (malloc_arg_t),
...}
```

input_t fread