The 18th Annual Network and Distributed System Security Symposium

SigGraph: Brute Force Scanning of Kernel Data Structure Instances Using Graph-based Signatures

Zhiqiang Lin[1]

Junghwan Rhee[1], Xiangyu Zhang[1], Dongyan Xu[1], Xuxian Jiang[2]

[1]Purdue University
[2]North Carolina State University

February 7th, 2011

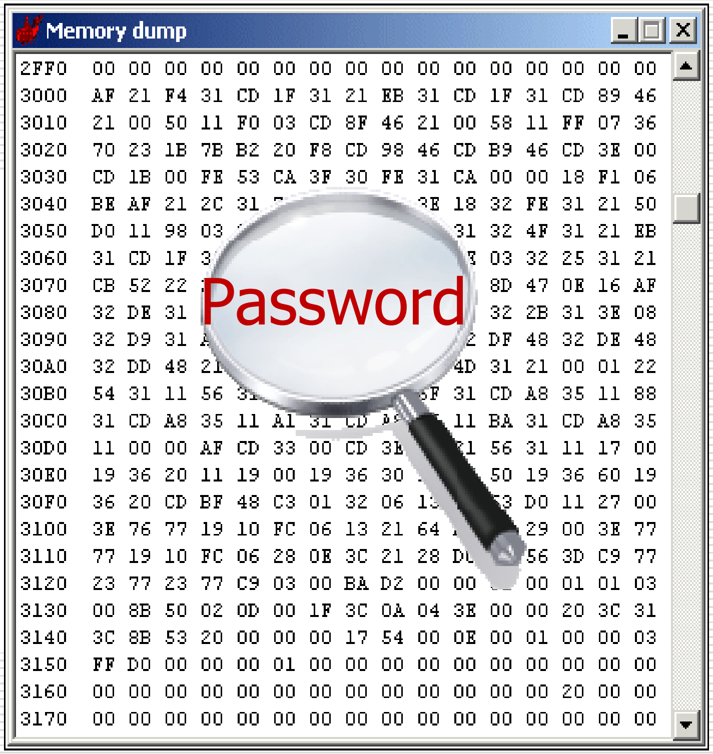
Problem Statement

- Given a kernel data structure definition
- Identifying *instances* of this data structure in a kernel memory image at arbitrary location

```
struct task {
  [0] struct thread *thread;
  [4] struct memory *mm;
  [8] struct signal *signal;
 [12] struct task *parent;
 [16] int magic_number;
}
```

A simplified Linux Kernel **`task_struct`**

Security Applications: Memory Forensics

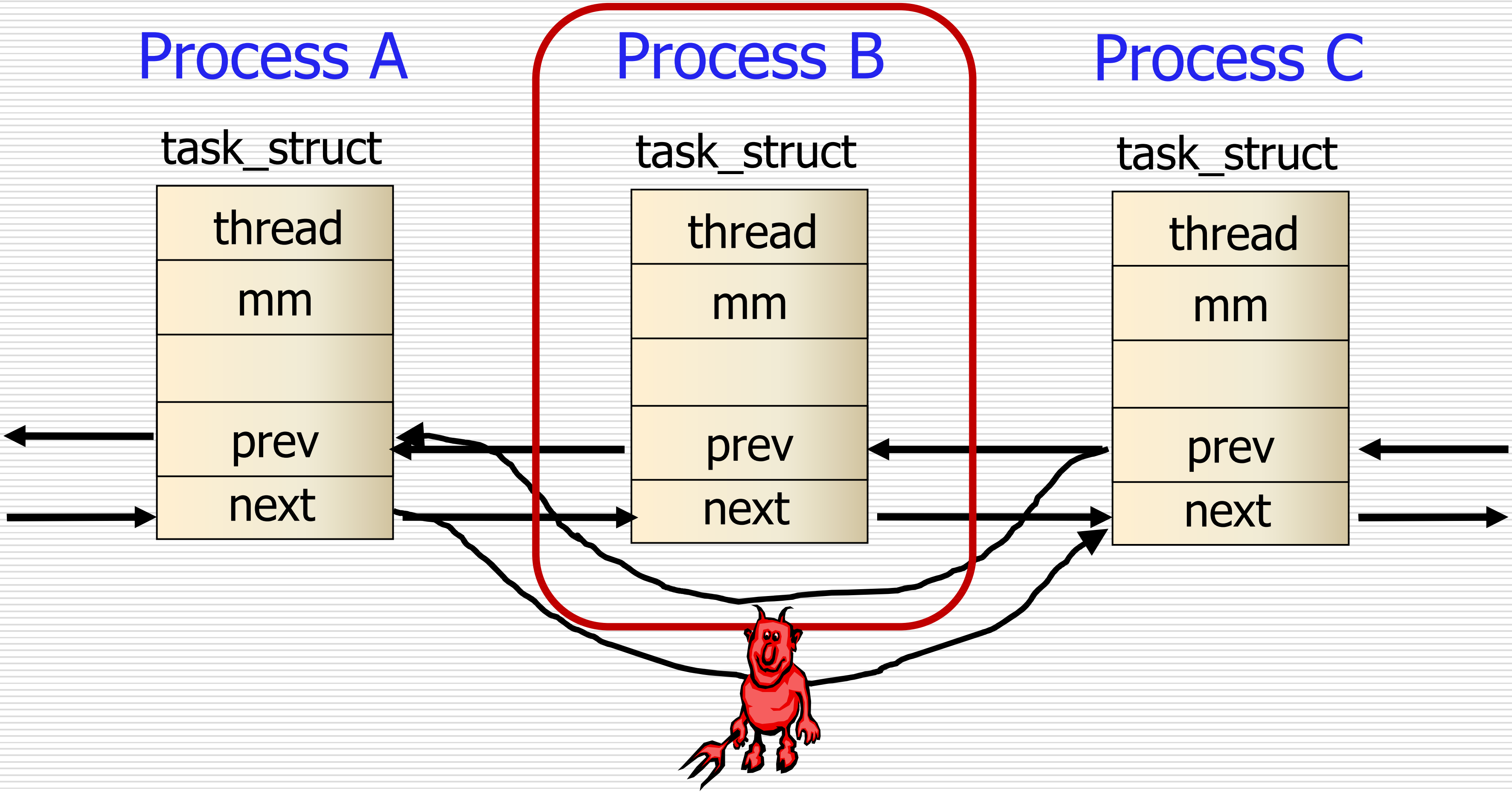


```
2FF0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3000  AF 21 F4 31 CD 1F 31 21 EB 31 CD 1F 31 CD 89 46
3010  21 00 50 11 F0 03 CD 8F 46 21 00 58 11 FF 07 36
3020  70 23 1B 7B B2 20 F8 CD 98 46 CD B9 46 CD 3E 00
3030  CD 1B 00 FE 53 CA 3F 30 FE 31 CA 00 00 18 F1 06
3040  BE AF 21 2C 31             3E 18 32 FE 31 21 50
3050  D0 11 98 03                   31 32 4F 31 21 EB
3060  31 CD 1F                            03 32 25 31 21
3070  CB 52 22                            8D 47 0E 16 AF
3080  32 DE 31                            32 2B 31 3E 08
3090  32 D9 31                            DF 48 32 DE 48
30A0  32 DD 48 21                      4D 31 21 00 01 22
30B0  54 31 11 56                      31 CD A8 35 11 88
30C0  31 CD A8 35 11 A1 31 CD       11 BA 31 CD A8 35
30D0  11 00 00 AF CD 33 00 CD 3E       56 31 11 17 00
30E0  19 36 20 11 19 00 19 36 30       50 19 36 60 19
30F0  36 20 CD BF 48 C3 01 32 06          D0 11 27 00
3100  3E 76 77 19 10 FC 06 13 21 64       29 00 3E 77
3110  77 19 10 FC 06 28 0E 3C 21 28       56 3D C9 77
3120  23 77 23 77 C9 03 00 BA D2 00 00    00 01 01 03
3130  00 8B 50 02 0D 00 1F 3C 0A 04 3E 00 00 20 3C 31
3140  3C 8B 53 20 00 00 00 17 54 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
```

```
struct user_account {
00: short int u_type;
04: pid_t u_pid;
08: char u_line[32];
40: char uid[4];
44: char user[32];
76: char password[128];
204: char u_host[128];
332: short int e_termination;
334: short int e_exit;
336: long int u_session;
340: struct timeval u_tv;
348: int32_t u_addr_v6[4];
}
```

Data structure signatures **play a critical role in memory forensics**

Security Applications: Kernel Rootkit Defense



State-of-the-art

- Value-invariant signature schemes
  - Klist [Rutkowska,2003], GREPEXEC [bugcheck, 2006], Volatility [Walters, 2006], [Schuster, 2006], [Dolan-Gavitt et al., CCS'09]

Invariant value can be changed?

[Dolan-Gavitt et al., CCS'09]

```
struct task {
  [0] struct thread *thread,
  [4] struct memory *mm;
  [8] struct signal *signal;
 [12] struct task *parent;
 [16] int magic_number;
}
```

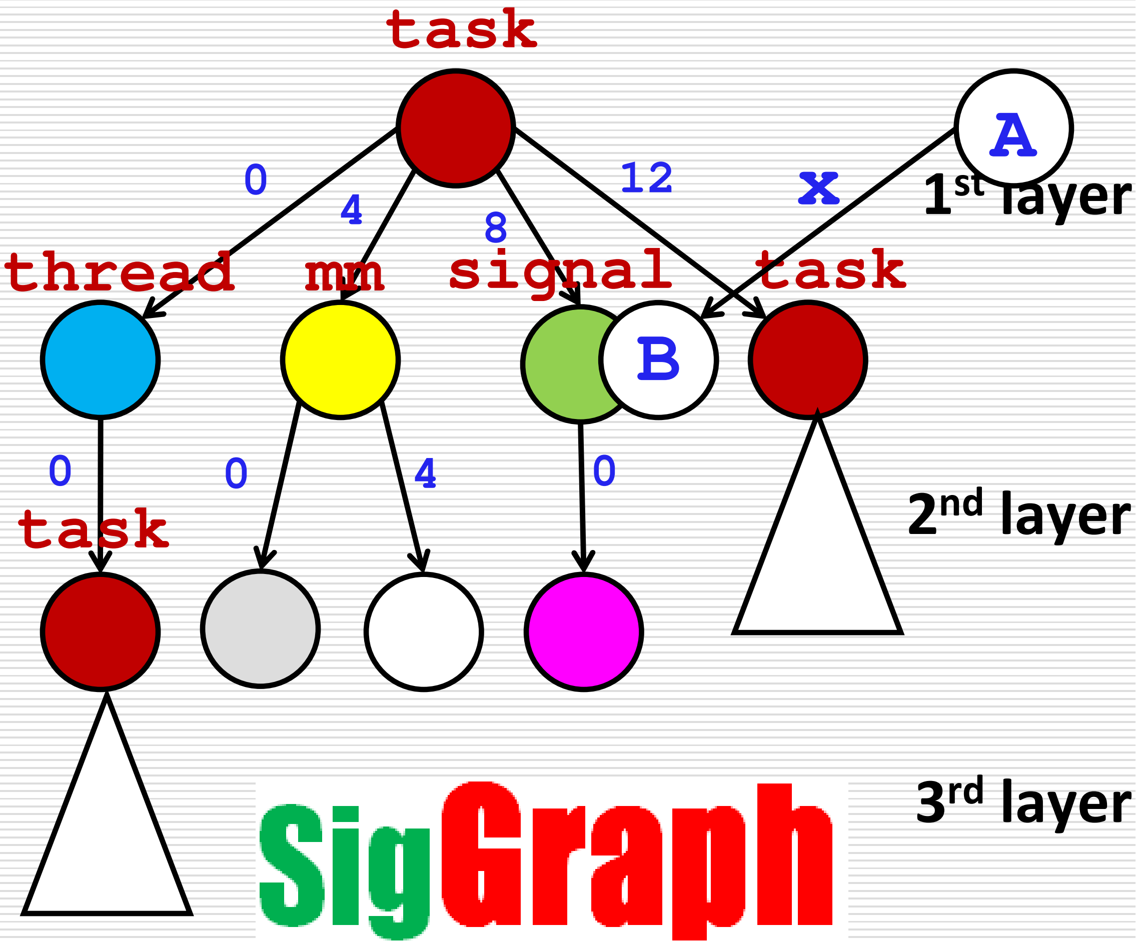
→ magic_number=0xabcdef0f

Field w/o value invariant?

Key Idea

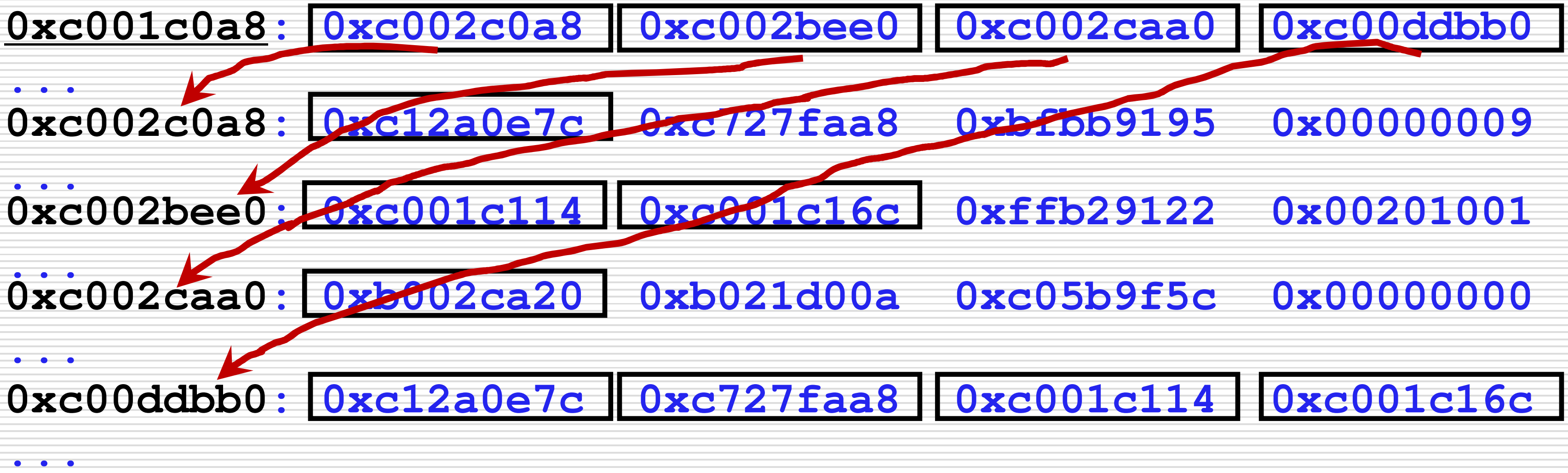
```
struct task {
   [0] struct thread *thread;
   [4] struct memory *mm;
   [8] struct signal *signal;
 [12] struct task *parent;
 }

 struct thread {
  [0] struct task *task;
 }

struct memory {
  [0] struct vma *mmap;
  [4] void (*map_area)
       (struct memory*
mmap);
}

struct signal {
  [0] struct task_status
*status;
}
```

task(x) ⟺ thread(*(x+0)) ∧ mm(*(x+4)) ∧ signal(*(x+8)) ∧ task(*(x+12))



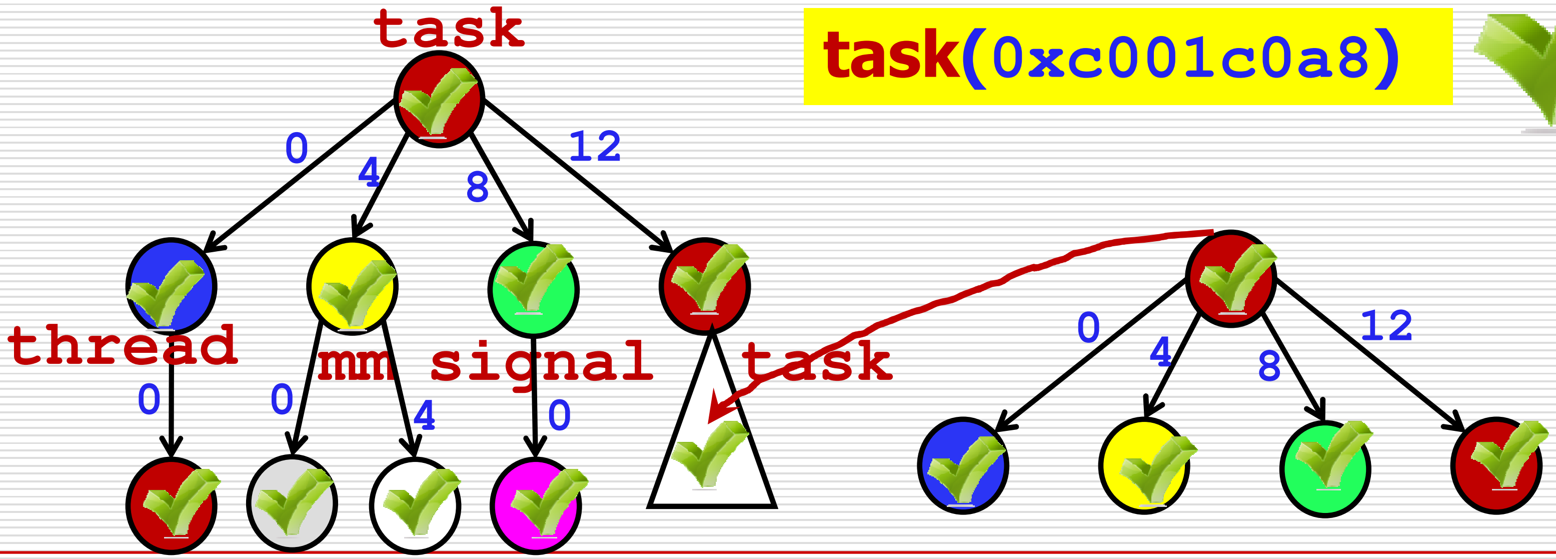
SigGraph

How to Use SigGraph

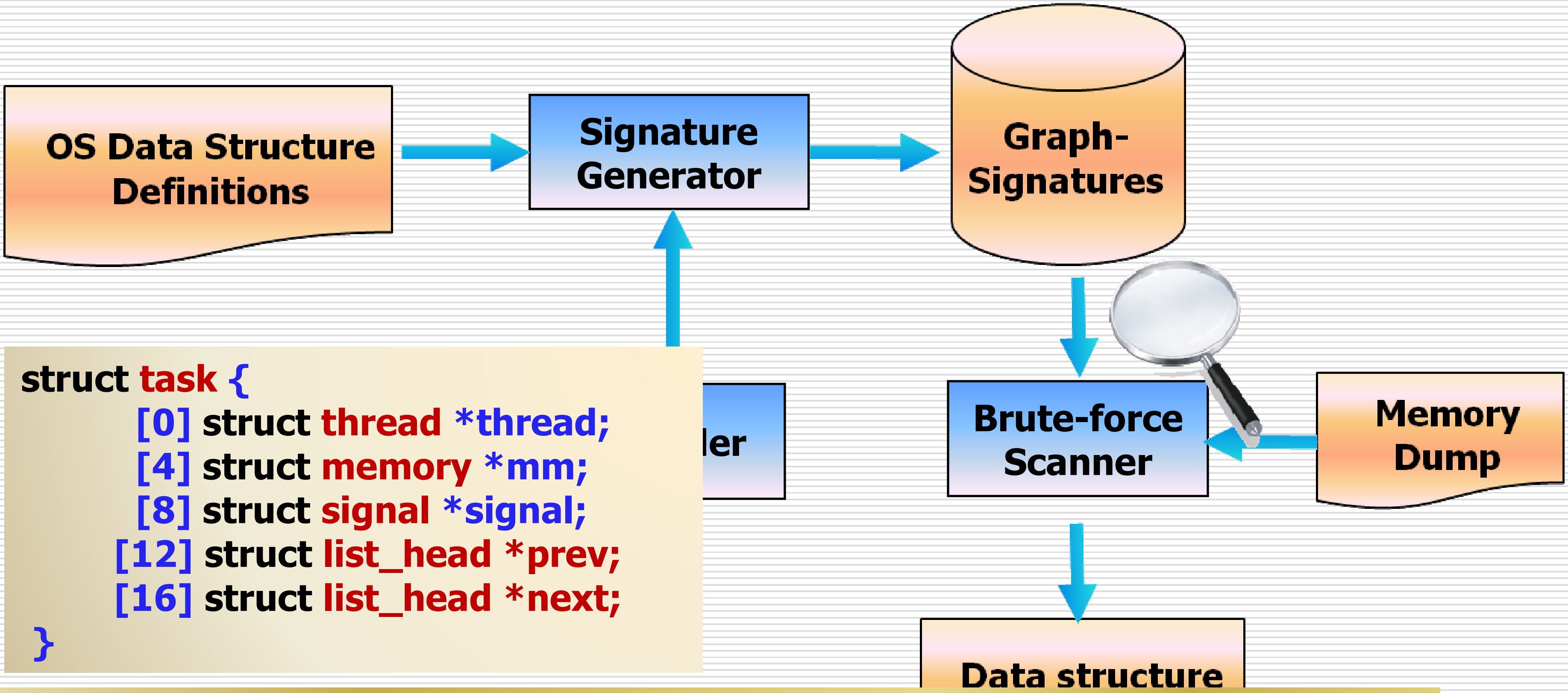


```
0xc001c0a8: 0xc002c0a8 0xc002bee0 0xc002caa0 0xc00ddbb0
...
0xc002c0a8: 0xc12a0e7c 0xc727faa8 0xbfbb9195 0x00000009
...
0xc002bee0: 0xc001c114 0xc001c16c 0xffb29122 0x00201001
...
0xc002caa0: 0xb002ca20 0xb021d00a 0xc05b9f5c 0x00000000
...
0xc00ddbb0: 0xc12a0e7c 0xc727faa8 0xc001c114 0xc001c16c
...
```

task(0xc001c0a8) ✓



SigGraph Overview



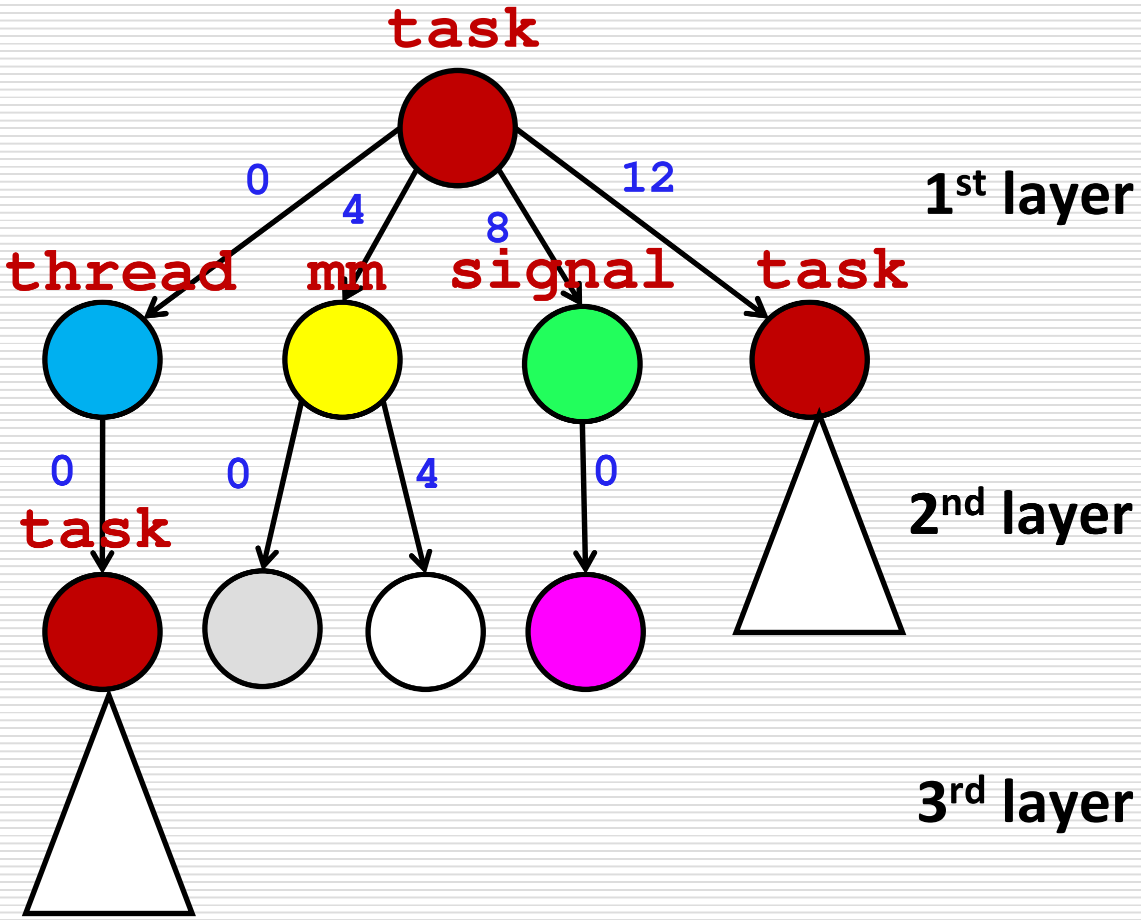
```
struct task {
        [0] struct thread *thread;
        [4] struct memory *mm;
        [8] struct signal *signal;
       [12] struct list_head *prev;
       [16] struct list_head *next;
}
```

(1) Compiler approach ✓

(2) Extracting from debug information

(3) Reverse engineering kernel

Signature Generator



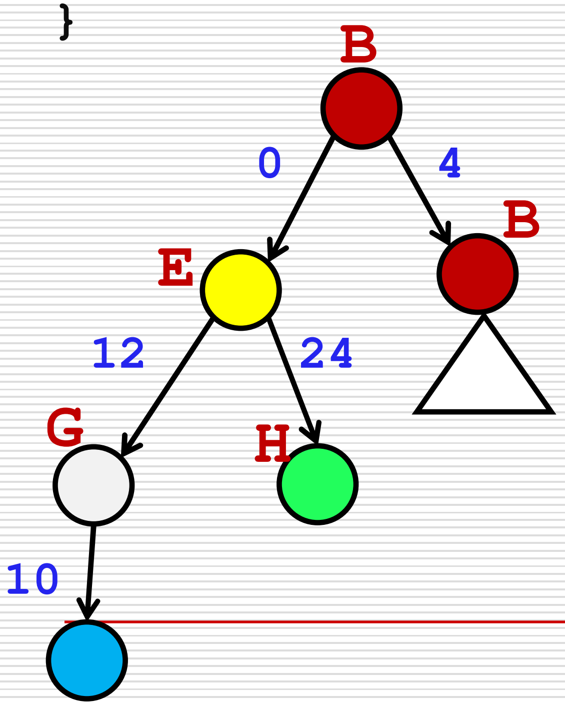
- **Challenge:** Signatures must be *unique*, *non-isomorphic* among each other.

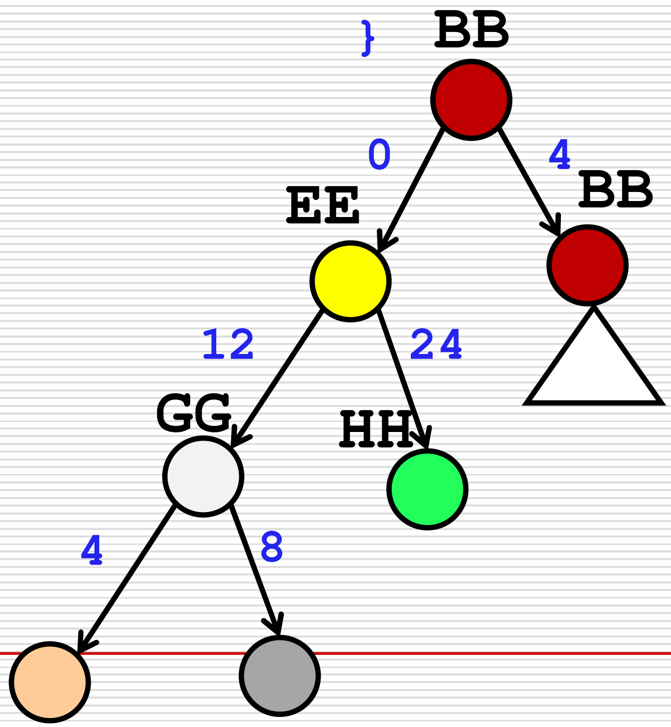
Isomorphism

```
struct B {
   [0]   E * b1;
   [4]   B * b2;
}
struct E {
      ...
   [12] G * e1;
      ...
   [24] H * e3;
}
```

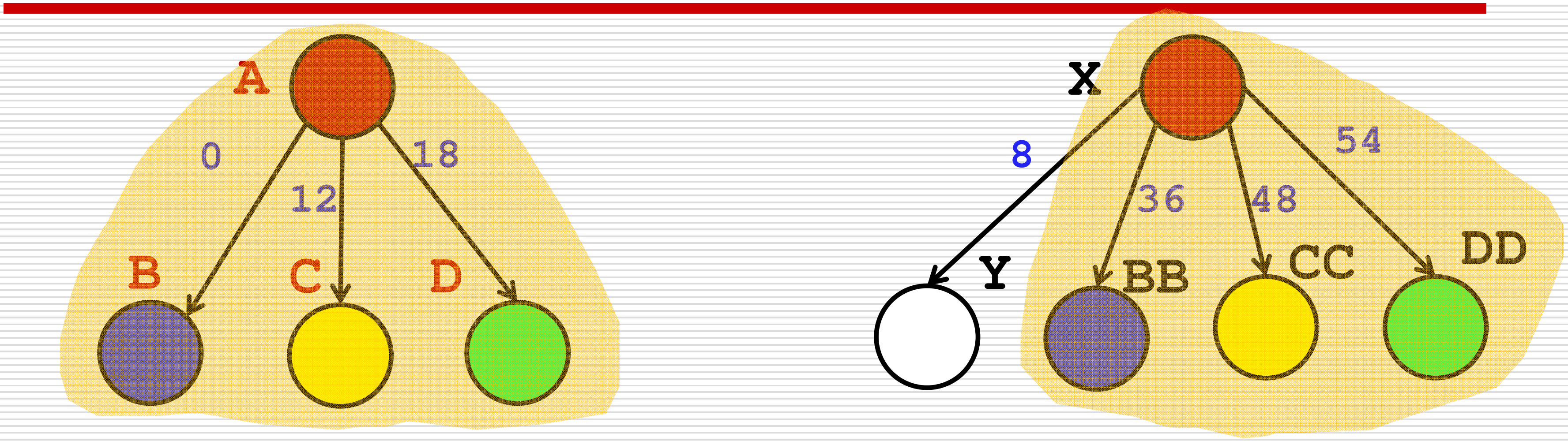
```
struct G {
      ...
   [10] int * g;
}
struct GG {
      ...
   [4] char * gg1;
   [8] char * gg2;
}
```

```
struct BB {
   [0]   EE * bb1;
   [4]   BB * bb2;
}
struct EE {
      ...
   [12] GG * ee1;
      ...
   [24] HH * ee3;
}
```





Isomorphism



```
struct A {
   [0]  struct B * a1;
     ...
   [12] struct C * a2;
     ...
   [18] struct D * a3;
}
```

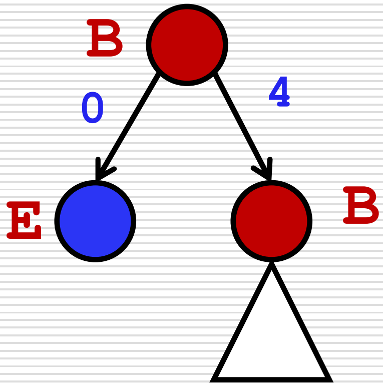
```
struct X {
     ...
   [8]  struct Y  * x1;
     ...
   [36] struct BB * x2;
     ...
   [48] struct CC * x3;
     ...
   [54] struct DD * x4;
}
```

Our Solution

- *Immediate pointer pattern (IPP):* one-layer pointer structure as a *string*

$$\mathit{IPP}(T) = f_1 \cdot t_1 \cdot (f_2 - f_1) \cdot t_2 \cdot \ldots \cdot (f_n - f_{n-1}) \cdot t_n$$

```
struct B {
   [0]  E * b1;
   [4]  B * b2;
}
```



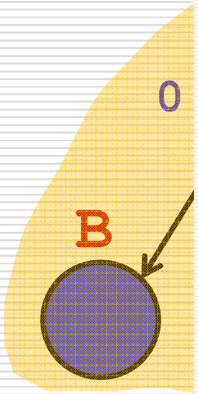
$$\mathit{IPP}(B) = 0 \cdot E \cdot 4 \cdot B$$

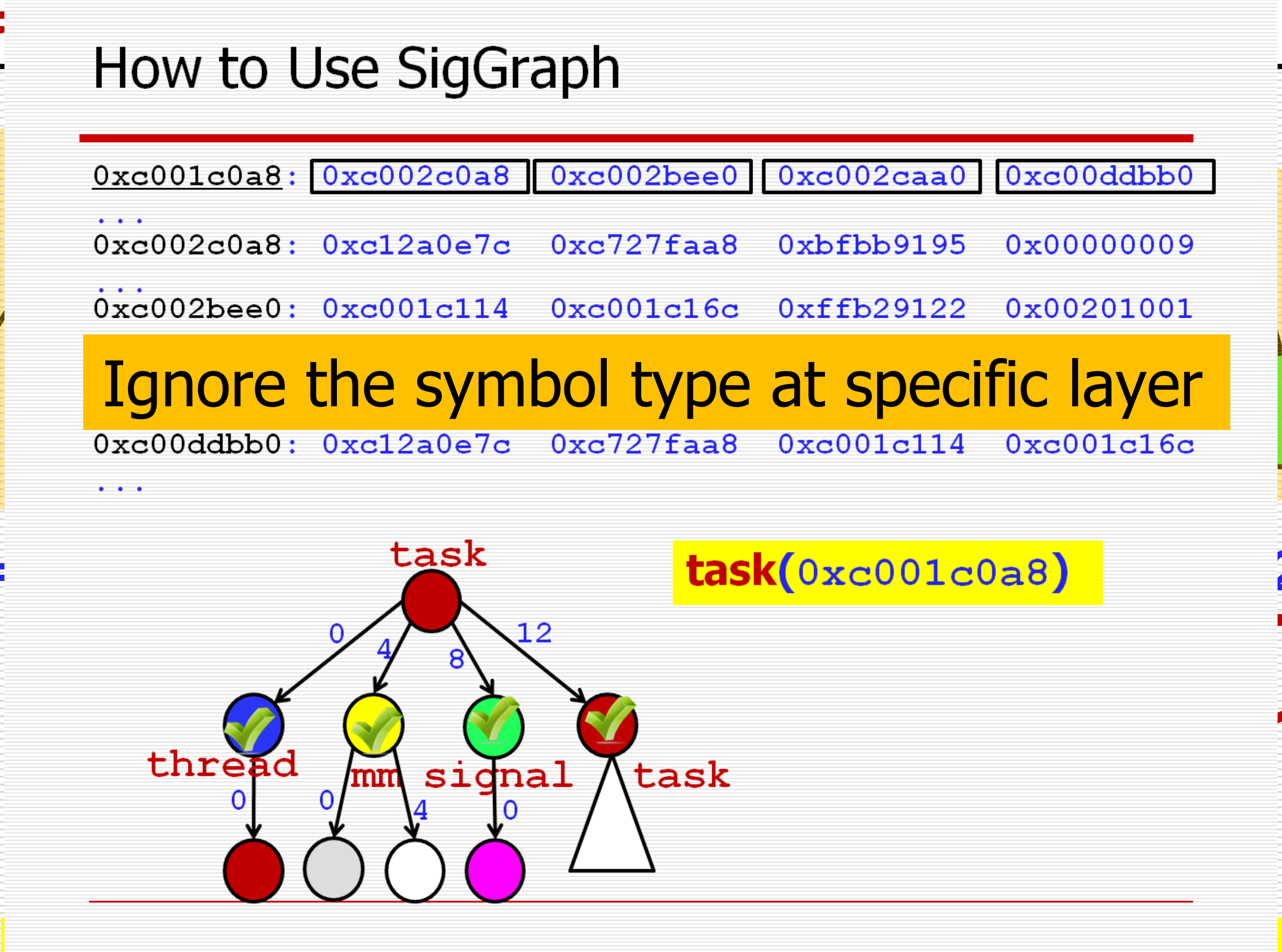
- Pointer expansion '$\xrightarrow{T}$'

$$\mathit{IPP}(B) \xrightarrow{B} 0 \cdot E \cdot 4 \cdot (0 \cdot E \cdot 4 \cdot B)$$

Problem Formulation

IPP $\cdot t_n$



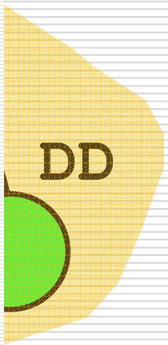


How to Use SigGraph

```
0xc001c0a8: 0xc002c0a8 0xc002bee0 0xc002caa0 0xc00ddbb0
...
0xc002c0a8: 0xc12a0e7c 0xc727faa8 0xbfbb9195 0x00000009
...
0xc002bee0: 0xc001c114 0xc001c16c 0xffb29122 0x00201001
0xc00ddbb0: 0xc12a0e7c 0xc727faa8 0xc001c114 0xc001c16c
...
```

Ignore the symbol type at specific layer

task(0xc001c0a8**)**



IPP(A) : 2·CC·6·DD

“If IPP(A) is a substring of IPP(X)”

Profiler

- Practical pointer issues
  - `null` Pointer
  - `void` Pointer
  - Special Pointer
    - LIST_POISON1 (0x00100100)
    - LIST_POISON2 (0x00200200)
    - SPINLOCK_MAGIC (0xdead4ead)

Pruning a few noisy pointer fields does not degenerate the uniqueness of the graph-based signatures

LiveDM [Rhee et al., RAID'10]

Evaluation

- Memory snapshot collection
  - QEMU
- Ground truth acquisition
  - RedHat crash utility
  - Symbolic information
    - system.map
- Profiling run
  - Long runs with typical workload

crash-utility.redhat.com

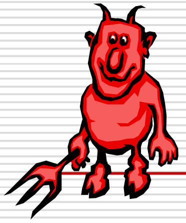
Evaluation on Memory Forensics

Data Structures of Interest	"True" Instance	SigGraph		Value-invariant	
		FP%	FN%	FP%	FN%
`task_struct`	88	0.00	0.00	0.00	0.00
`thread_info`	88	0.00	0.00	6.45	1.08
`mm_struct`	52	0.00	0.00	0.00	0.00
`vm_area_struct`	2174	0.40	0.00	7.52	0.00
`files_struct`	53	0.00	0.00	0.00	0.00
`fs_struct`	52	0.00	0.00	0.00	0.00
`dentry`	31816	0.01	0.00	0.01	0.00
`sysfs_dirent`	2106	0.52	0.00	97.63	0.00
`socket`	55	0.00	0.00	0.00	12.24
`sock`	55	0.00	0.00	0.00	27.90
`user_struct`	10	0.00	0.00	99.91	0.00

crash-utility.redhat.com

Application: Rootkit Detection

Rootkit Name	Target Object	Inside View	Crash Tool		SigGraph	
		#obj.s	#obj.s	detected	#obj.s	detected
adore-ng-2.6	module	23	23	✗	24	✓
adore-ng-2.6’	task_struct	62	63	✓	63	✓
cleaner-2.6	module	22	22	✗	23	✓
enyelkm 1.0	module	23	23	✗	24	✓
hp-2.6	task_struct	56	57	✓	57	✓
linuxfu-2.6	task_struct	59	60	✓	60	✓
modhide-2.6	module	22	22	✗	23	✓
override	task_struct	58	59	✓	59	✓
rmroots	task_struct	56	N/A	✗	55	✓
rmroots’	module	23	N/A	✗	24	✓



```
ps
lsmod
```

Related Work

- Kernel memory mapping and analysis
  - Copilot [Petroni et al., Security'04], [Petroni et al., CCS'07]
  - Gibraltar [Baliga et al., ACSAC'08]
  - KOP [Carbone et al.,CCS'09]
- Memory forensics
  - Memory graph-based: Redhat crash utility, KOP
  - Value-invariant Signature: Klist [Rutkowska,2003], GREPEXEC [bugcheck, 2006], Volatility [Walters, 2006], [Schuster, 2006], [Dolan-Gavitt et al., CCS'09]
- Dynamic heap type inference [Polishchuk et al., 2007]

Conclusion

- Points-to relations can be leveraged to generate graph-based signatures for brute force scanning

- SigGraph, a framework that generates *non-isomorphic structural-invariant signatures*
  - Complements *value-invariant* signatures

- Applications:
  - Kernel memory forensics
  - Kernel rootkit detection

Q&A

Thank you

For more information

{zlin,rhee,xyzhang,dxu}@cs.purdue.edu