OS-SOMMELIER: Memory-Only Operating System Fingerprinting in the Cloud

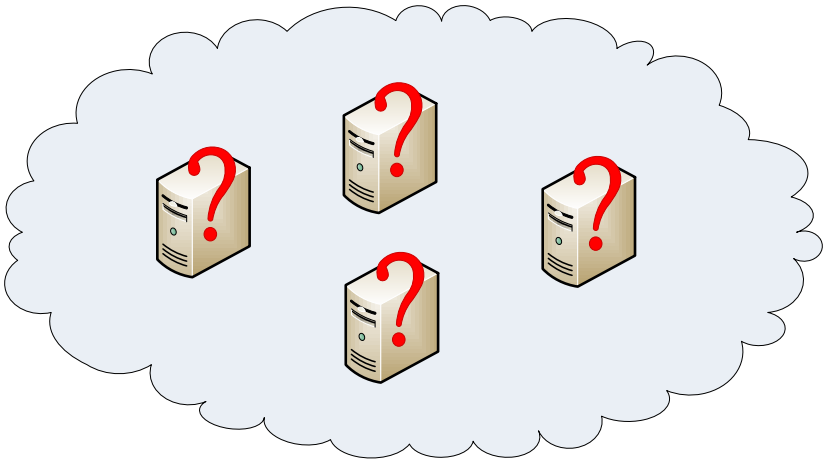
Yufei Gu[†], Yangchun Fu[†], Aravind Prakash[‡]
Dr. Zhiqiang Lin[†], Dr. Heng Yin[‡]

[†]University of Texas at Dallas [‡]Syracuse University

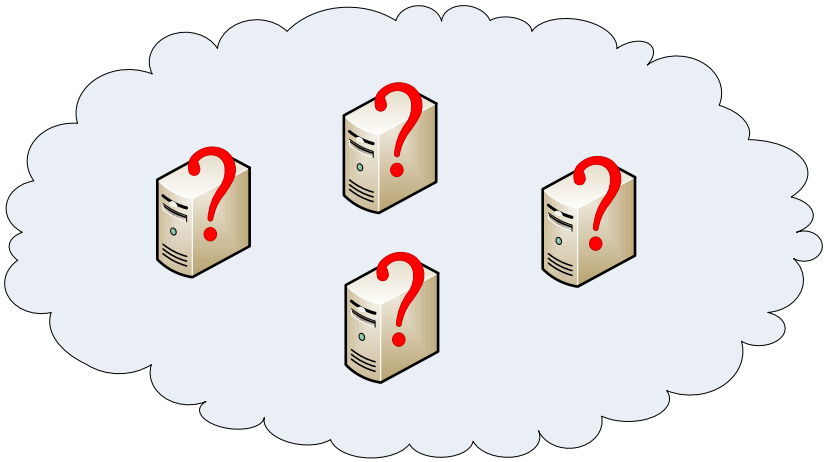
October 16th, 2012

Outline

What is OS Fingerprinting



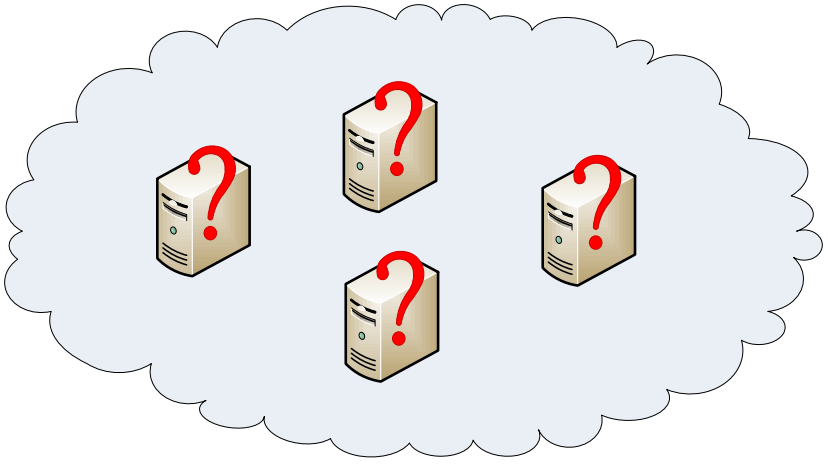
What is OS Fingerprinting



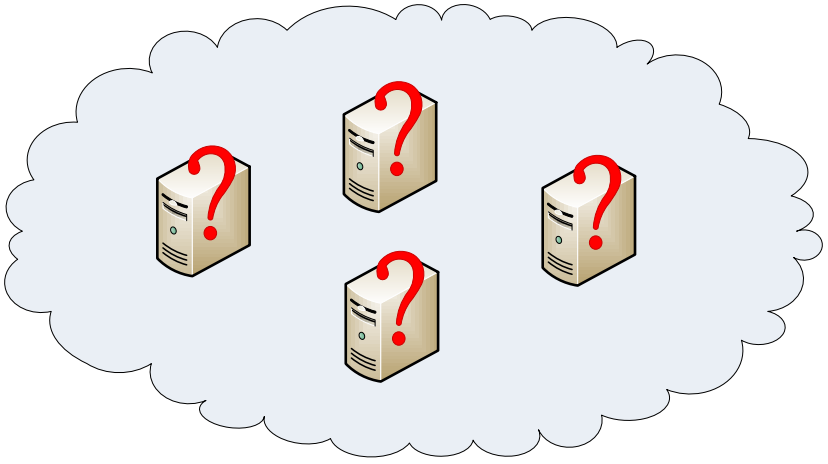
OS Fingerprinting in the Cloud

Given a virtual machine (VM) image (or a running instance), precisely infer its specific OS kernel versions

Why we need OS Fingerprinting in the Cloud

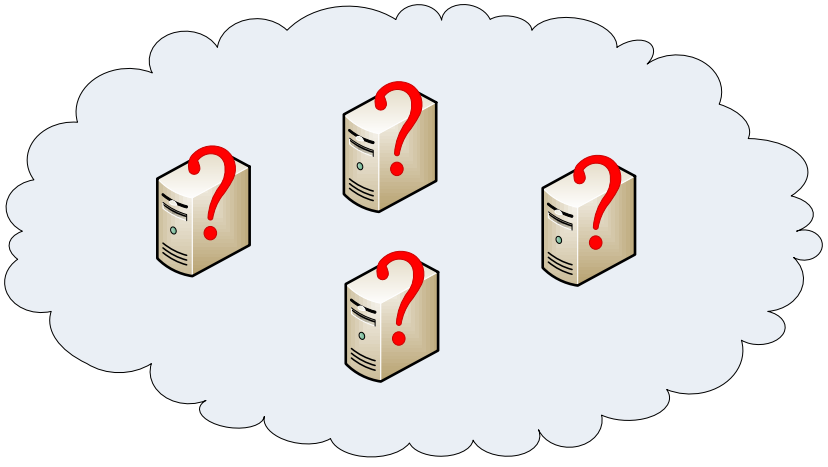


Why we need OS Fingerprinting in the Cloud



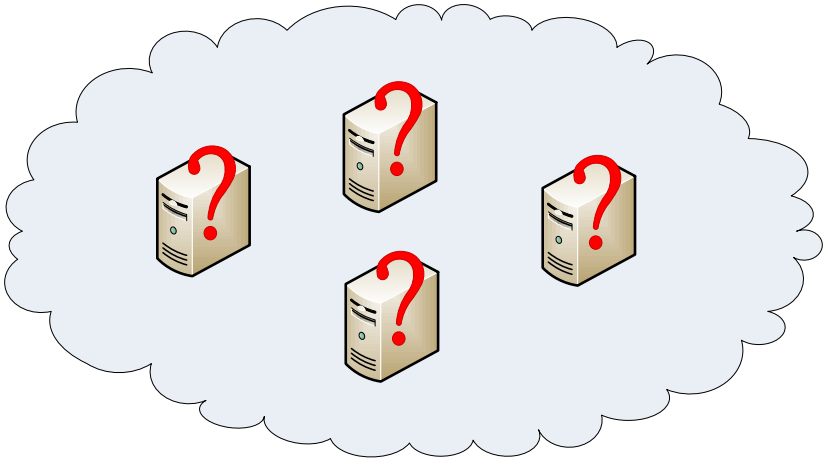
1. Virtual Machine Introspection [Garfinkel and Rosenblum, NDSS'03]

Why we need OS Fingerprinting in the Cloud



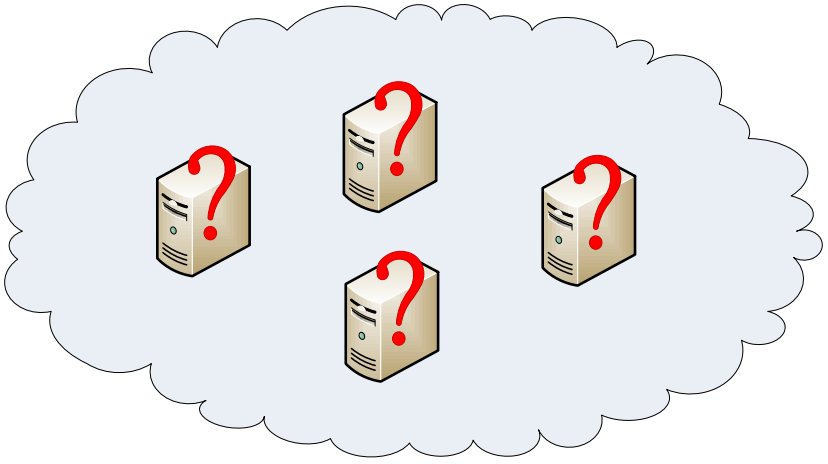
1. Virtual Machine Introspection [Garfinkel and Rosenblum, NDSS'03]
2. Penetration Testing

Why we need OS Fingerprinting in the Cloud



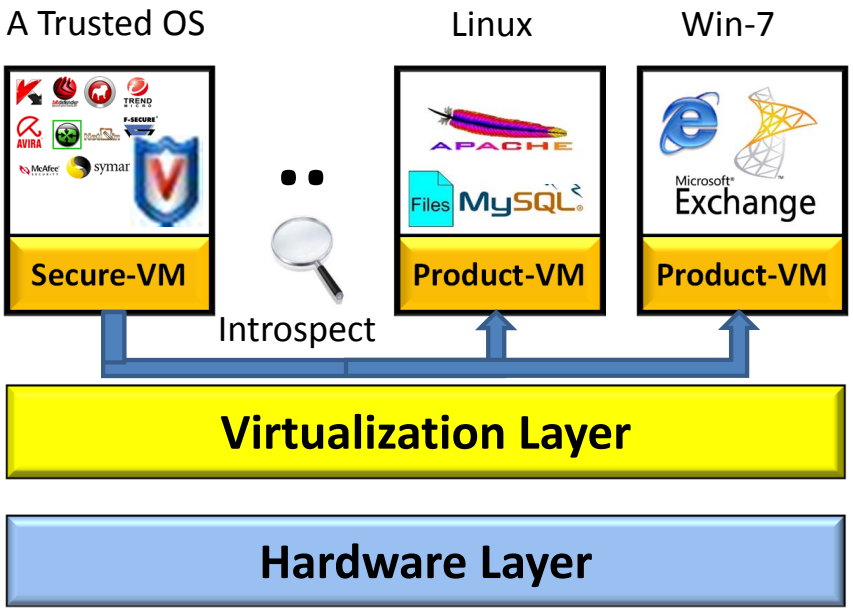
1. Virtual Machine Introspection [Garfinkel and Rosenblum, NDSS'03]
2. Penetration Testing
3. VM Management (Kernel Update)

Why we need OS Fingerprinting in the Cloud

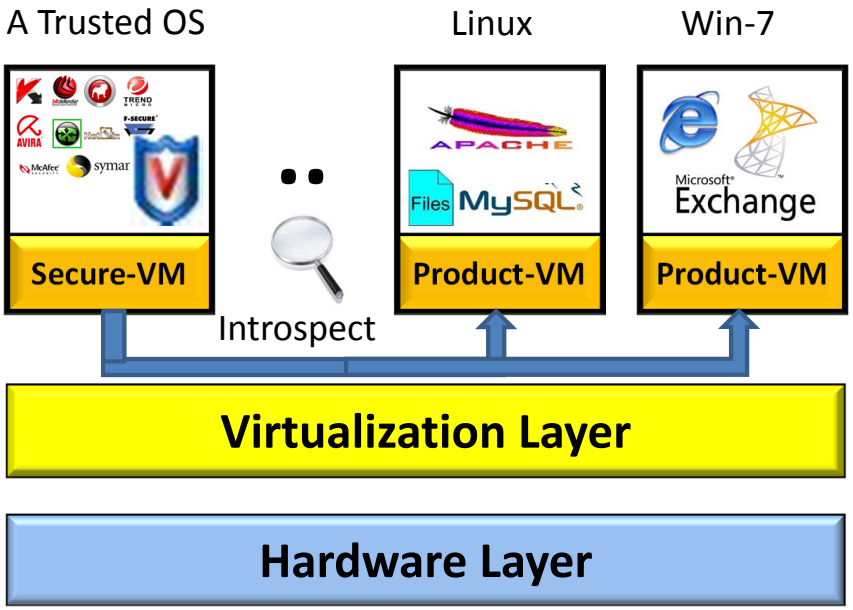


1. Virtual Machine Introspection [Garfinkel and Rosenblum, NDSS'03]
2. Penetration Testing
3. VM Management (Kernel Update)
4. Memory Forensics

Virtual Machine Introspection (VMI) [Garfinkel and Rosenblum, NDSS'03]

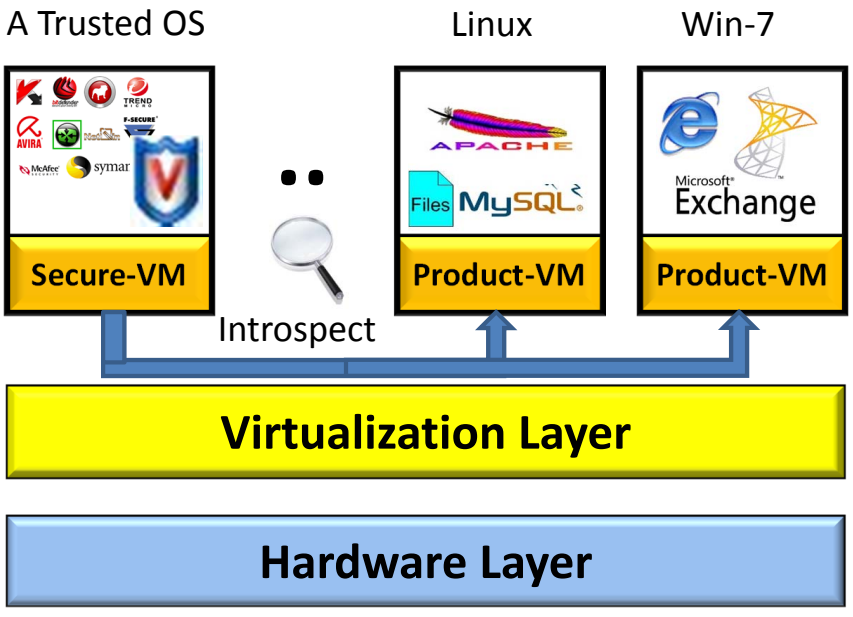


Virtual Machine Introspection (VMI) [Garfinkel and Rosenblum, NDSS'03]



Using a trusted, isolated, dedicated VM to monitor other VMs

Virtual Machine Introspection (VMI) [Garfinkel and Rosenblum, NDSS'03]

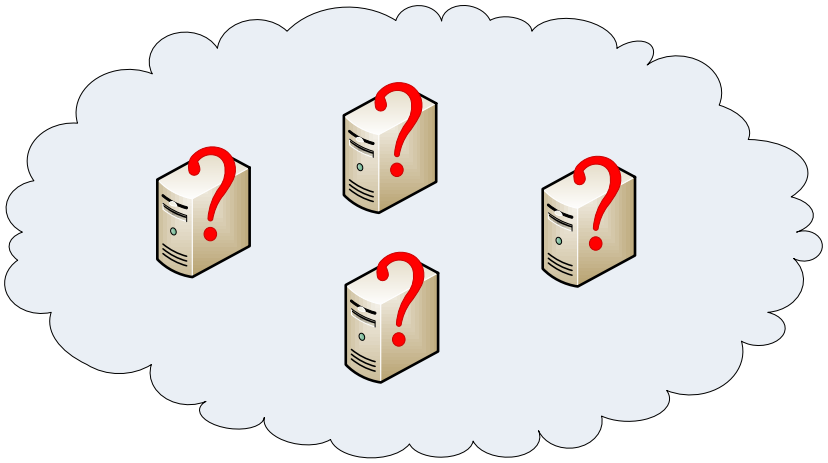


Using a trusted, isolated, dedicated VM to monitor other VMs

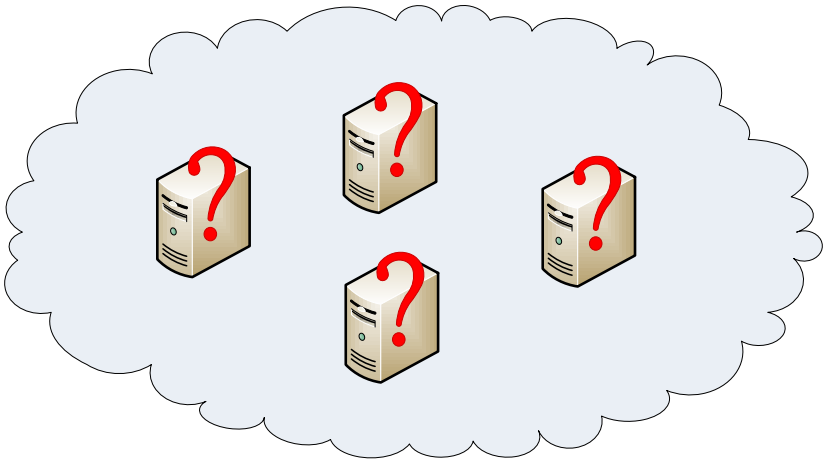
Binary Code Reuse based VMI

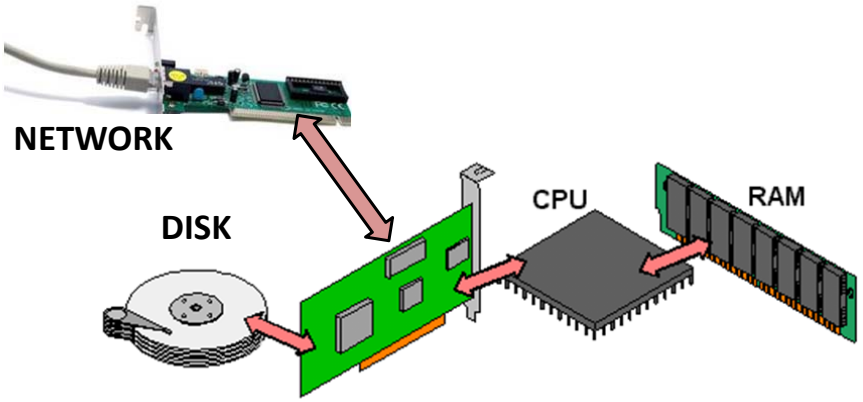
- Virtuoso [Dolan-Gavitt et al, Oakland'11]: using trained existing legacy code to perform VMI
- VM Space Traveler [Fu and Lin, Oakland'12]: dynamically instrumenting legacy binary code to perform VMI

Basic Approaches for OS Fingerprinting



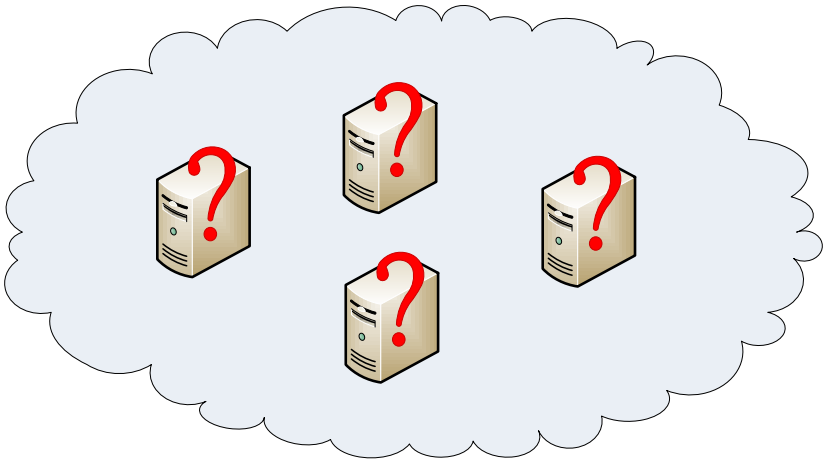
Basic Approaches for OS Fingerprinting

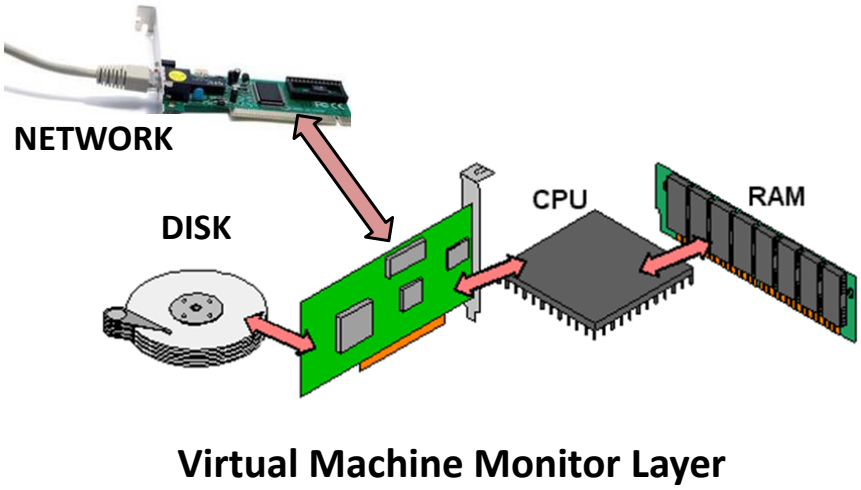




Virtual Machine Monitor Layer

Basic Approaches for OS Fingerprinting

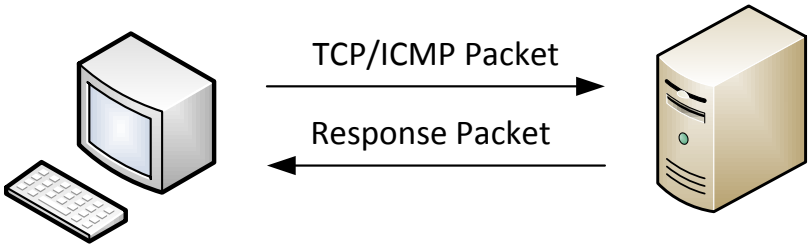




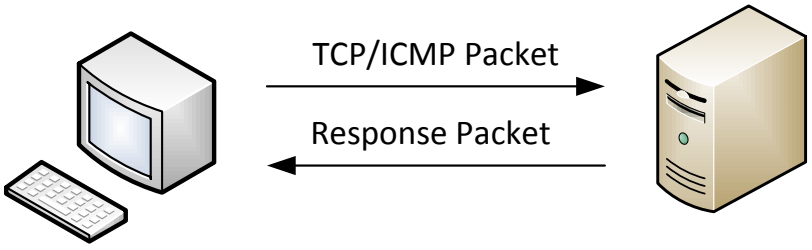
Basic Approaches

1. Network
2. File System
3. CPU State
4. Memory
5. Their Combinations

Network-based OS Fingerprinting



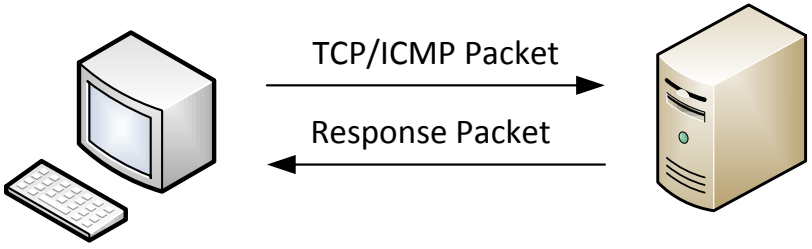
Network-based OS Fingerprinting



Existing Techniques

- Probing TCP implementations [Comer and Lin, USENIX Summer ATC'94]
- Nmap [Fyodor]
- Xprob2 [Yarochkin, DSN'09]
- Synscan [Taleck, CanSecWest'04]
- ...

Network-based OS Fingerprinting



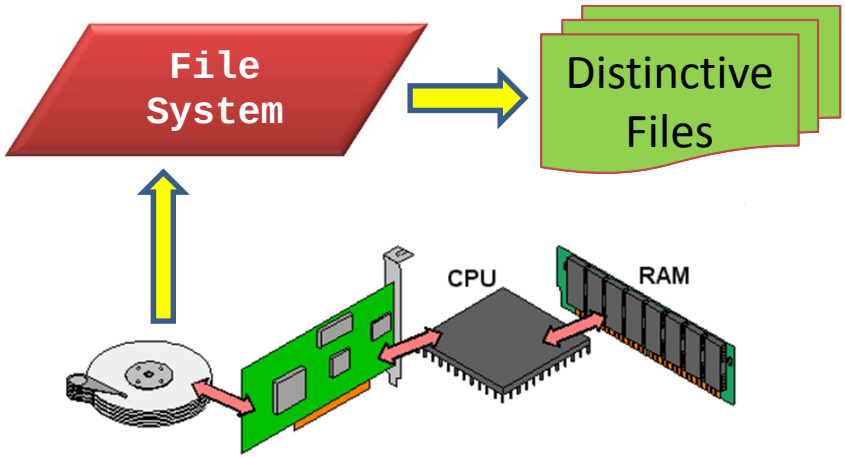
Existing Techniques

- Probing TCP implementations [Comer and Lin, USENIX Summer ATC'94]
- Nmap [Fyodor]
- Xprob2 [Yarochkin, DSN'09]
- Synscan [Taleck, CanSecWest'04]
- ...

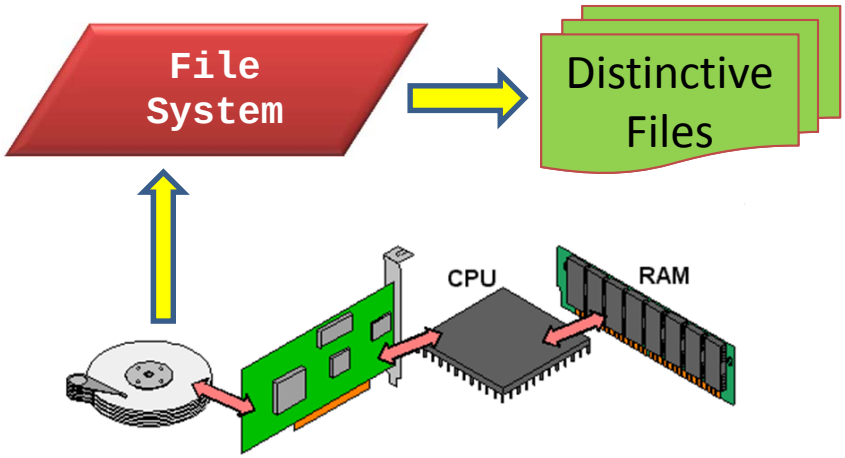
Limitations

- **Imprecise**: not accurate enough, cannot pinpoint minor differences
- **Can be disabled**: many modern OSes disable most of the network services as a default security policy

File-System Based OS Fingerprinting



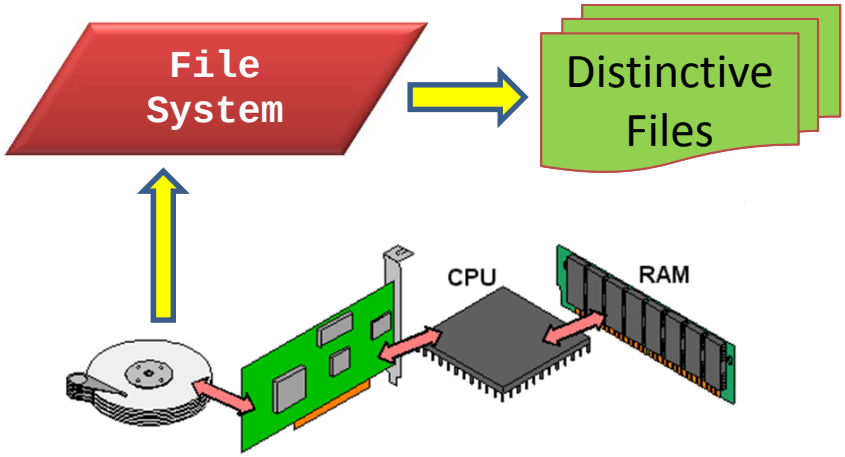
File-System Based OS Fingerprinting



Basic Approach

- Mount the VM file system image
- Walk through the files in the disk
- **Advantages**: Simple, Intuitive, Efficient, and Precise

File-System Based OS Fingerprinting



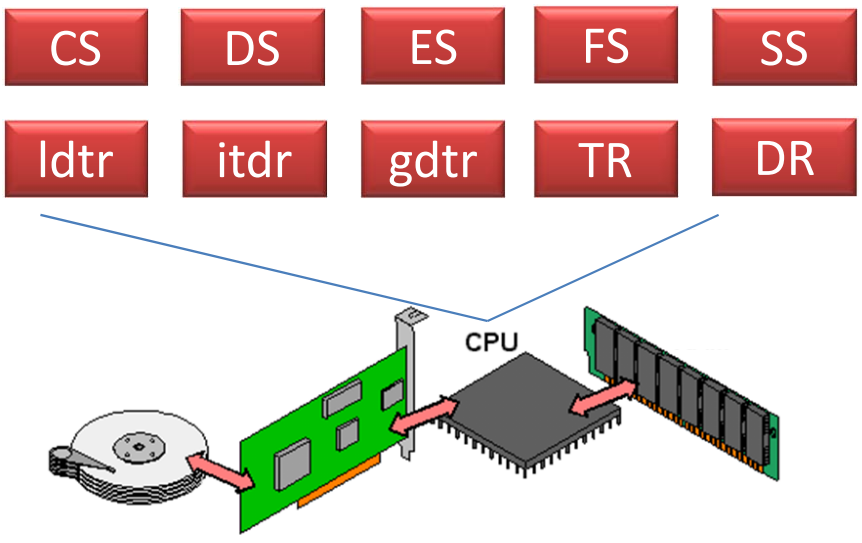
Basic Approach

- Mount the VM file system image
- Walk through the files in the disk
- **Advantages**: Simple, Intuitive, Efficient, and Precise

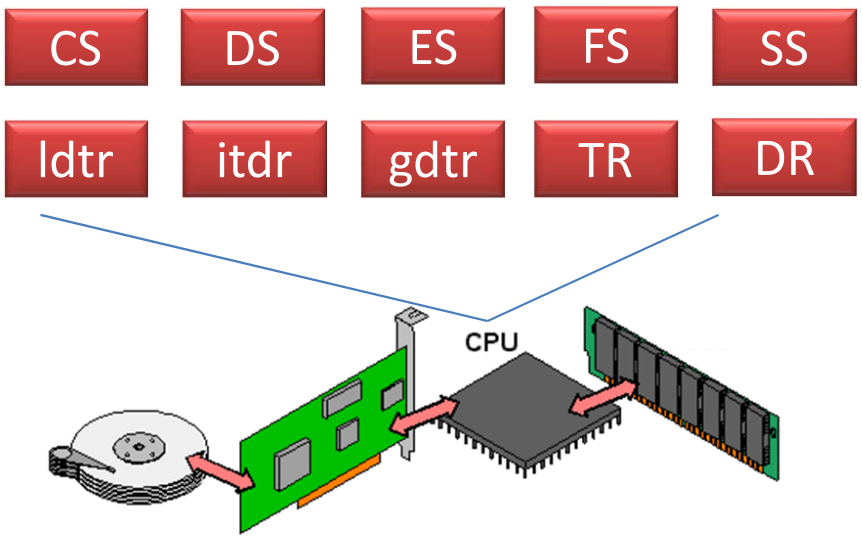
Limitations

- **File System Encryption**
- Cannot suit for memory forensics applications when only having memory dump

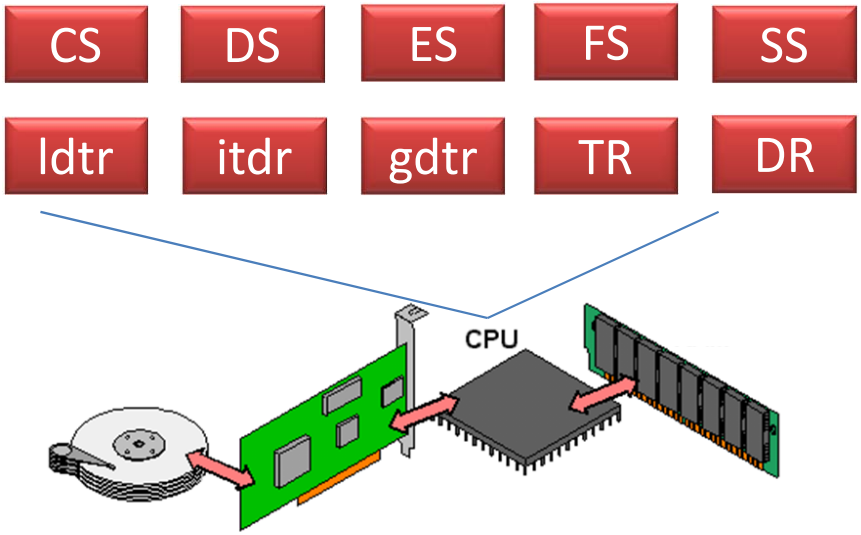
CPU Register based OS Fingerprinting



CPU Register based OS Fingerprinting



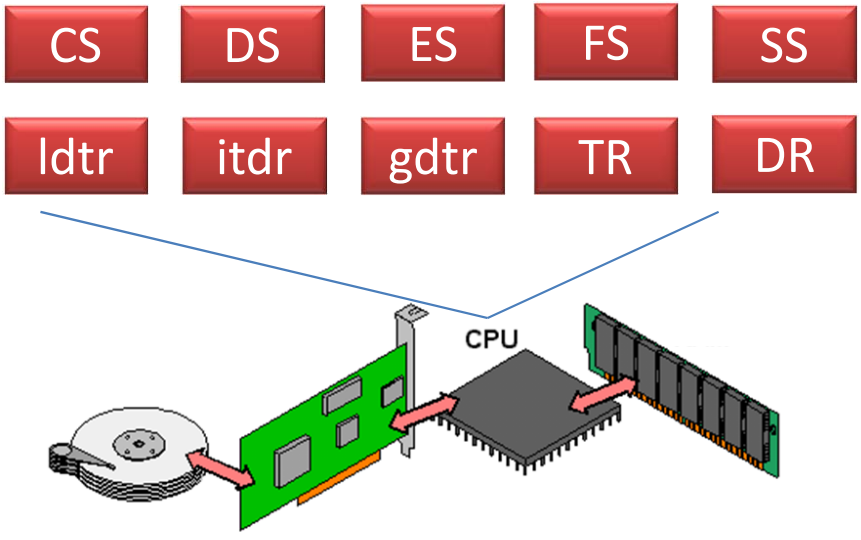
CPU Register based OS Fingerprinting



Existing Technique

- UFO: Operating system fingerprinting for virtual machines [Quynh, DEFCON '10]
- Advantage: efficient (super fast)

CPU Register based OS Fingerprinting



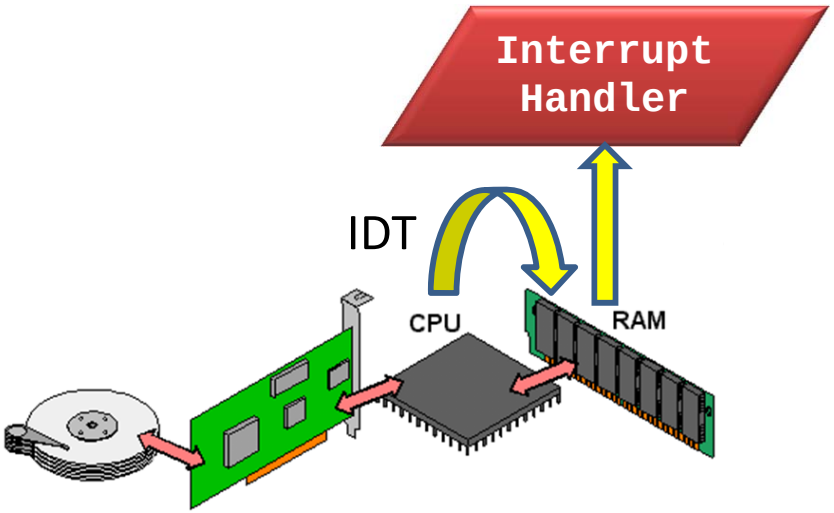
Existing Technique

- UFO: Operating system fingerprinting for virtual machines [Quynh, DEFCON '10]
- Advantage: efficient (super fast)

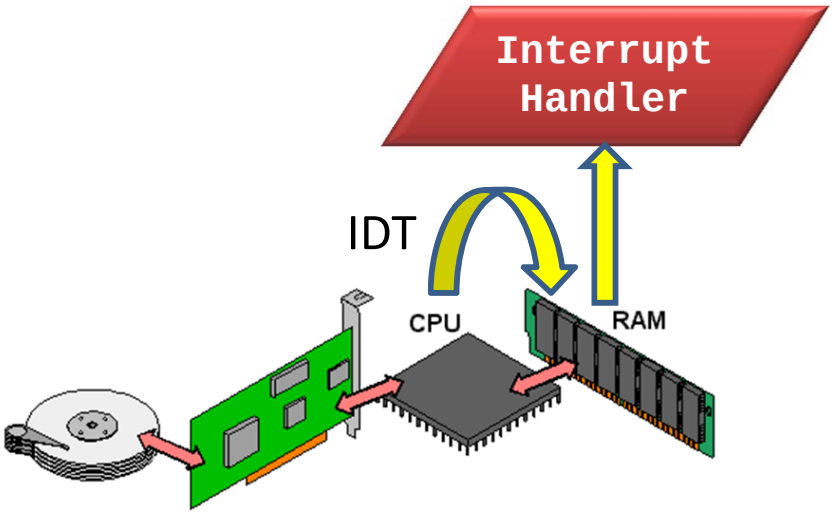
Limitations

- **Imprecise**: not accurate enough. WinXP (SP2) vs WinXP (SP3)
- Cannot suit for memory forensics applications when only having memory dump

CPU and Memory Combination based OS Fingerprinting



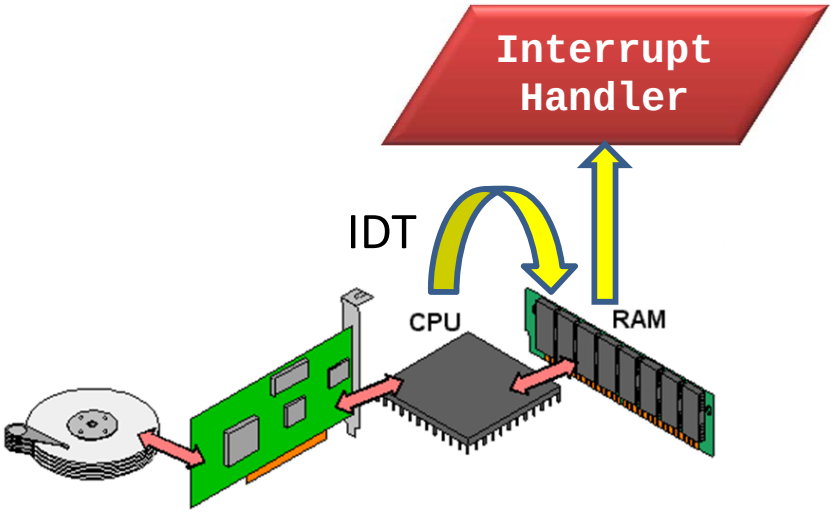
CPU and Memory Combination based OS Fingerprinting



Existing Techniques

- Using IDT pointer to retrieve interrupt handler code, and hash these code to fingerprint guest VM [Christodorescu et al, CCSW'09]

CPU and Memory Combination based OS Fingerprinting



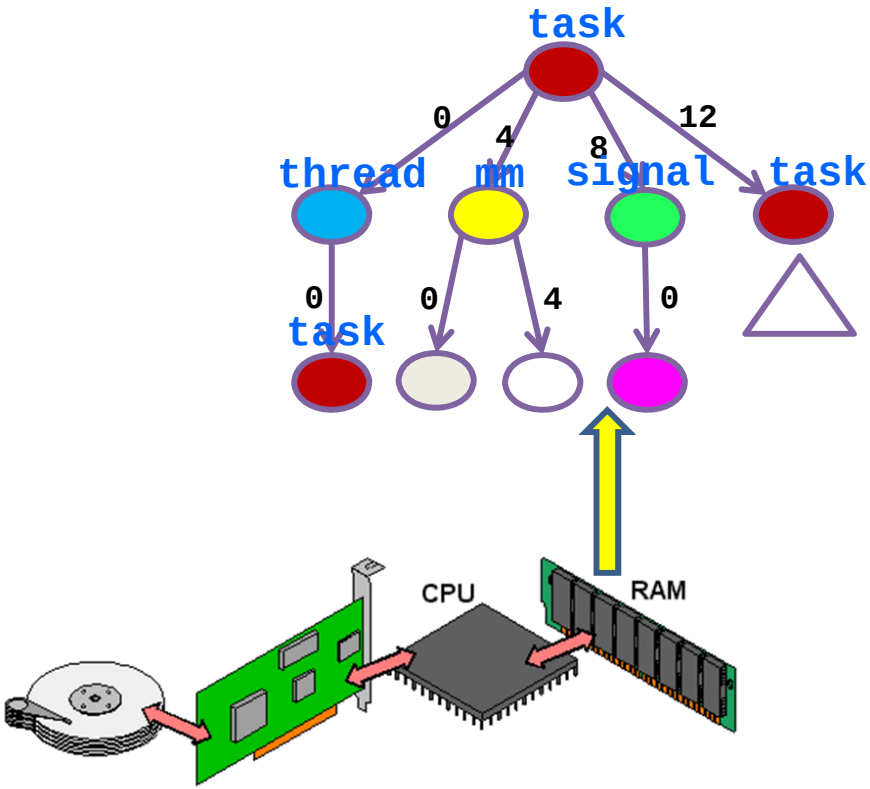
Existing Techniques

- Using IDT pointer to retrieve interrupt handler code, and hash these code to fingerprint guest VM [Christodorescu et al, CCSW'09]

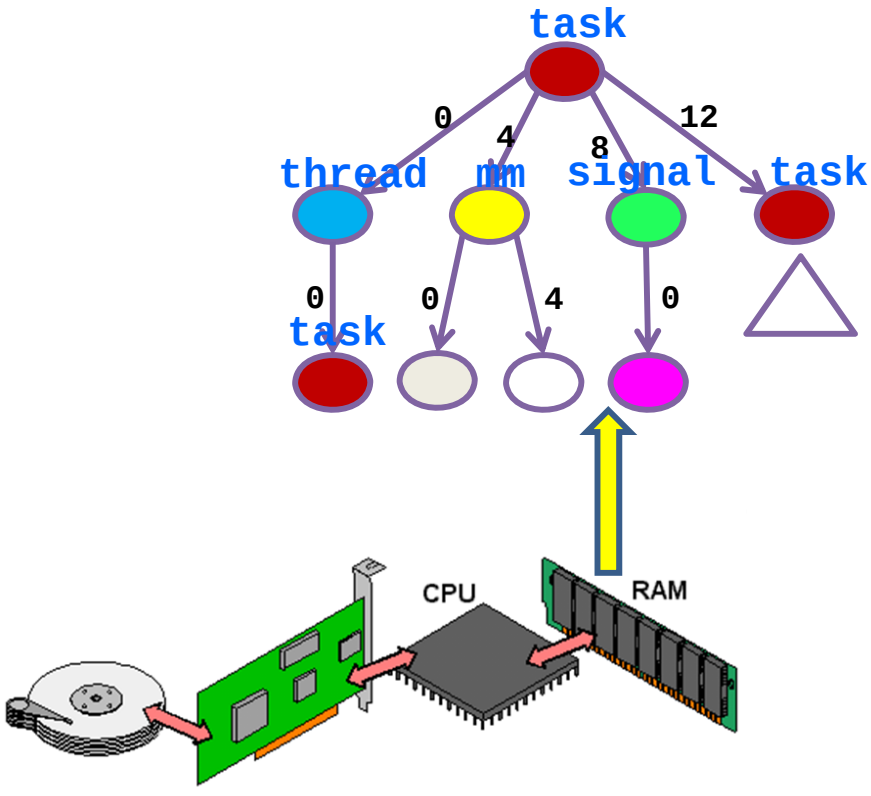
Limitations

- **Imprecise**: not accurate enough, cannot pinpoint minor differences
- Cannot suit for memory forensics applications when only having memory dump

Memory-Only Approach for OS Fingerprinting



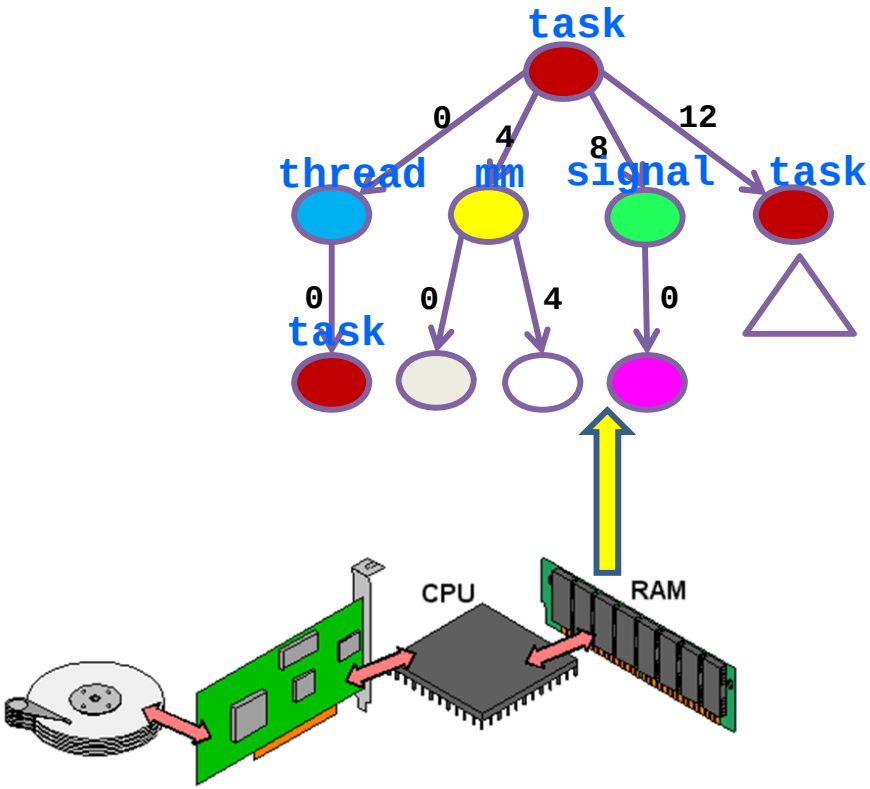
Memory-Only Approach for OS Fingerprinting



Existing Technique

- SigGraph: Brute Force Scanning of Kernel Data Structure Instances Using Graph-based Signatures [Lin et al, NDSS'11]

Memory-Only Approach for OS Fingerprinting



Existing Technique

- SigGraph: Brute Force Scanning of Kernel Data Structure Instances Using Graph-based Signatures [Lin et al, NDSS'11]

Limitations

- **Inefficient**: a few minutes
- Requires kernel data structure definitions

OS-Sommelier: Memory-Only OS Fingerprinting

Goal

- **Precise**: can pinpoint even minor OS differences
- **Efficient**: in a few seconds
- **Robust**: hard to evade, security perspective

OS-Sommelier: Memory-Only OS Fingerprinting

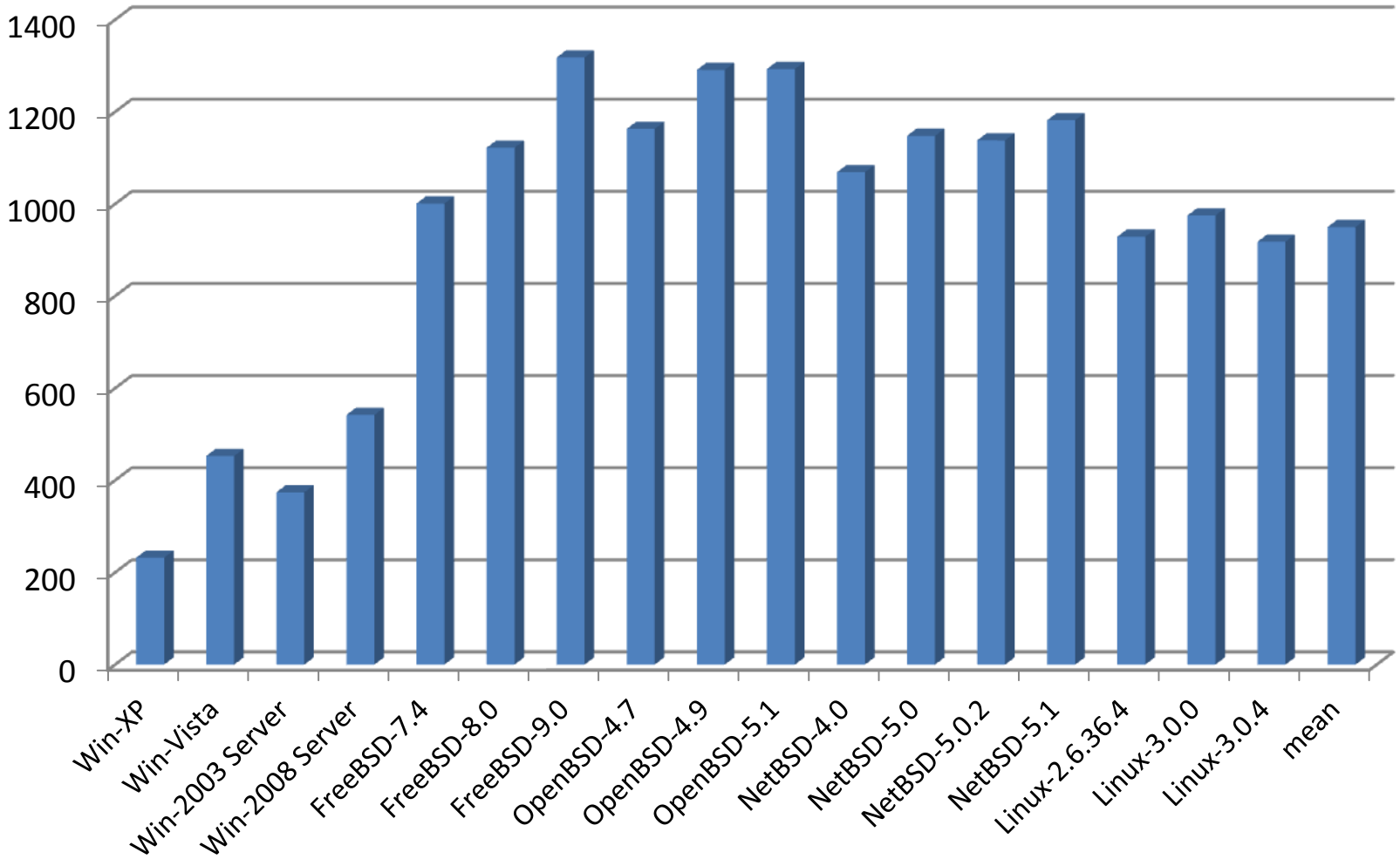
Goal

- **Precise**: can pinpoint even minor OS differences
- **Efficient**: in a few seconds
- **Robust**: hard to evade, security perspective

Key Idea

Compute the hash values of **core** kernel code in the physical memory for the precise fingerprinting.

Some Statistics on Core Kernel Page



OS-Sommelier: Challenges

Challenges

- How to get a robust and generic way to identify the kernel page table (when only having memory dump)?

To traverse memories, We need PGDs to do virtual-to-physical address translation.

OS-Sommelier: Challenges

Challenges

- How to get a robust and generic way to identify the kernel page table (when only having memory dump)?
- How to differentiate the main kernel code from the rest of code and data in the memory?

There are core kernel code, kernel data, module code and module data in memories.

OS-Sommelier: Challenges

Challenges

- How to get a robust and generic way to identify the kernel page table (when only having memory dump)?
- How to differentiate the main kernel code from the rest of code and data in the memory?
- How to correctly disassemble the kernel code?

Code could start from any position. If we start disassembling from wrong positions, we will get totally wrong codes.

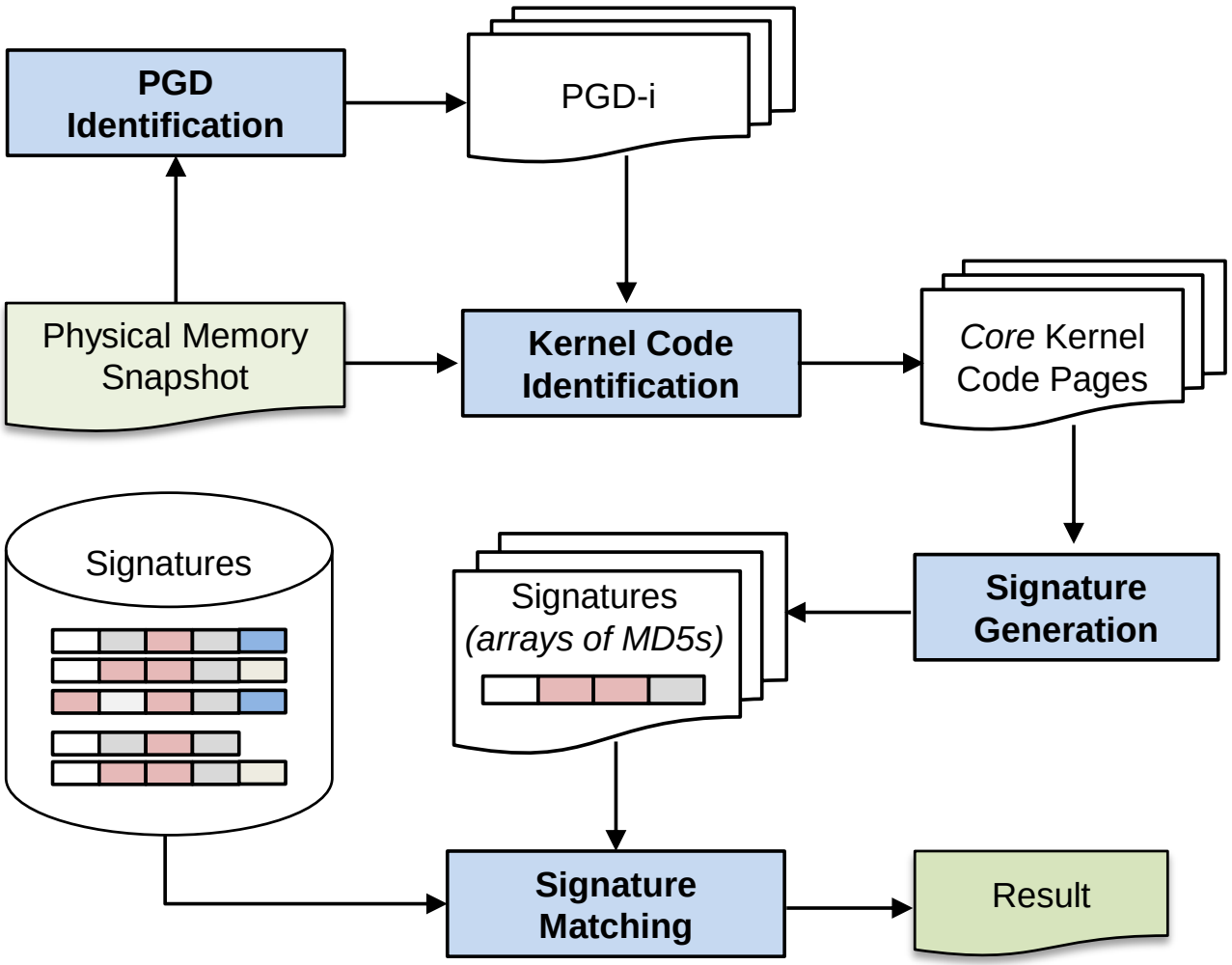
OS-Sommelier: Challenges

Challenges

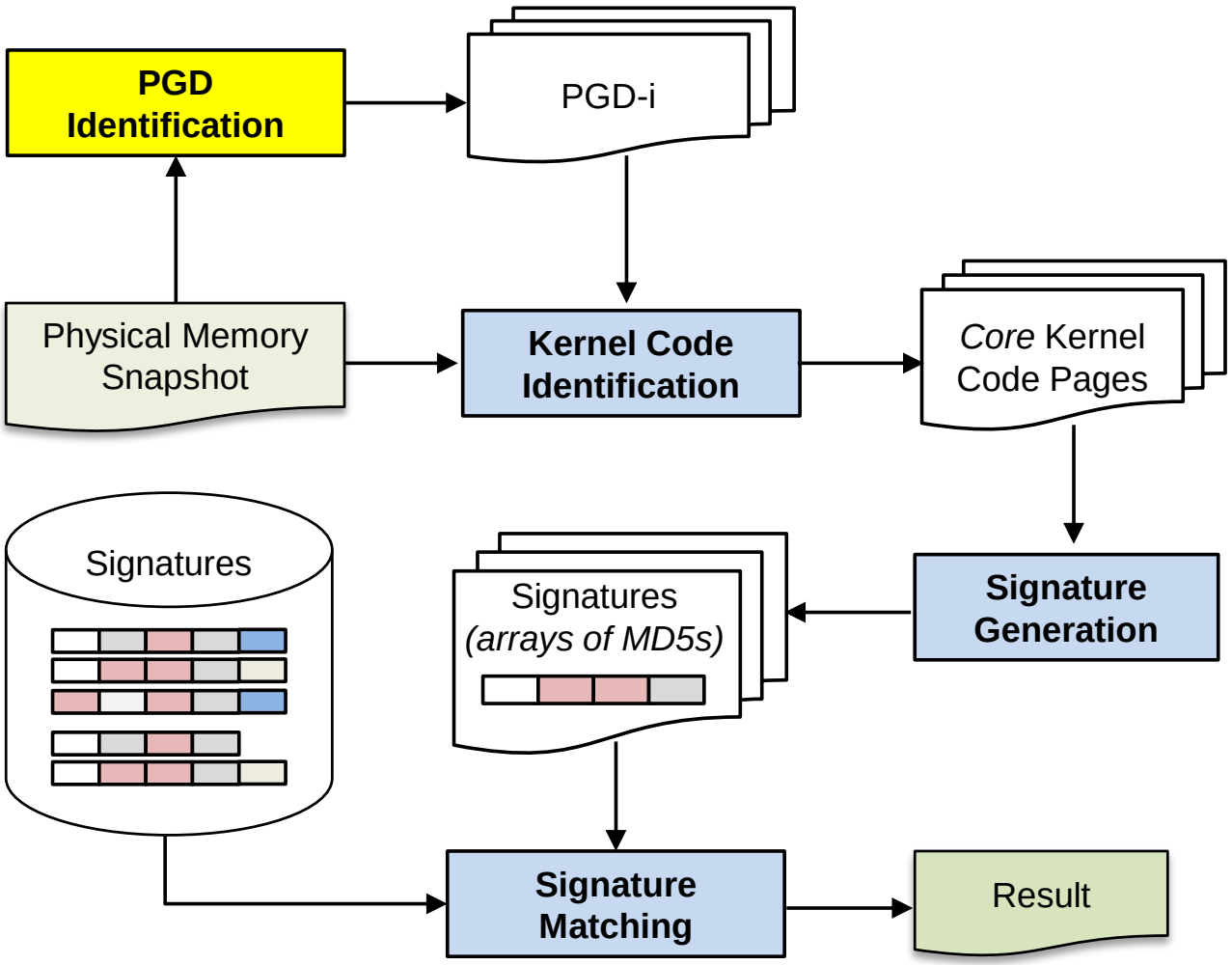
- How to get a robust and generic way to identify the kernel page table (when only having memory dump)?
- How to differentiate the main kernel code from the rest of code and data in the memory?
- How to correctly disassemble kernel code?
- How to normalize the kernel code to deal with practical issues such as ASLR?

Some modern OSs such as Windows Vista and Windows 7 have enabled address space layout randomization(ASLR).

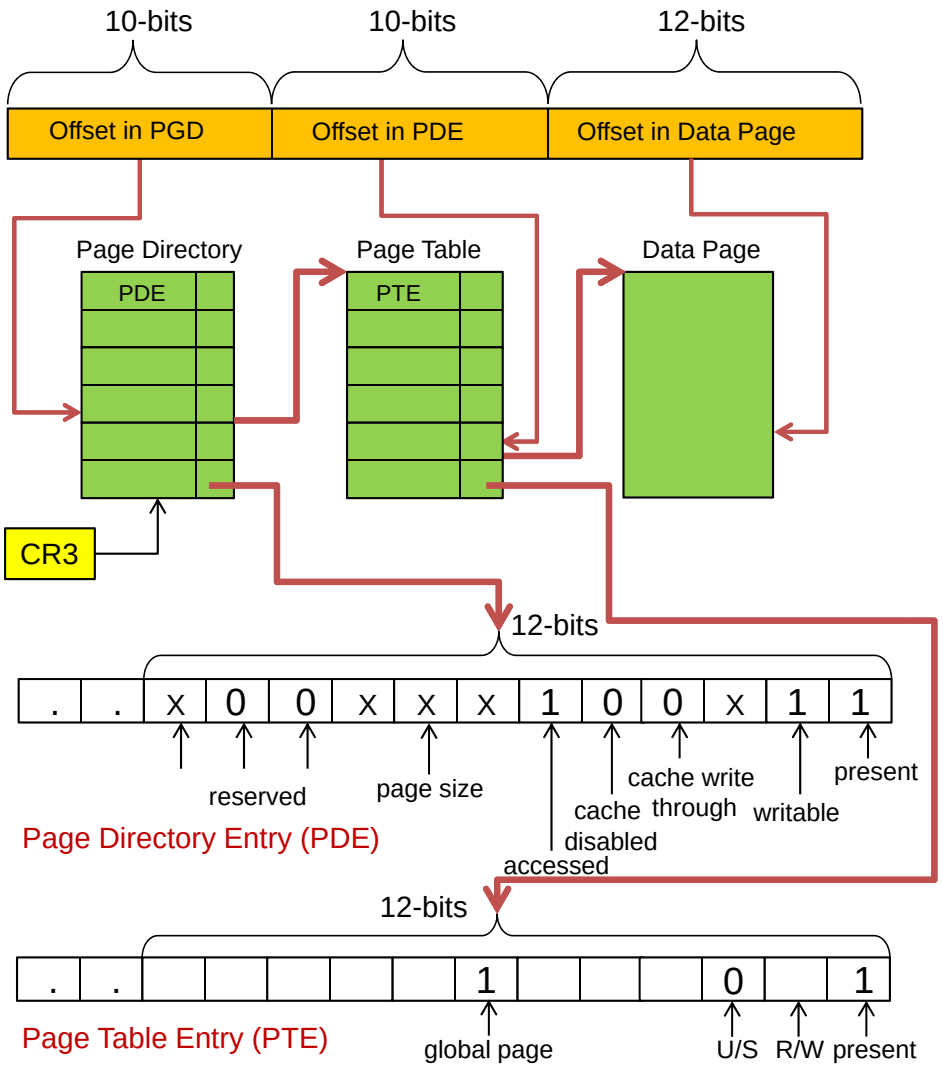
OS-Sommelier: Architecture



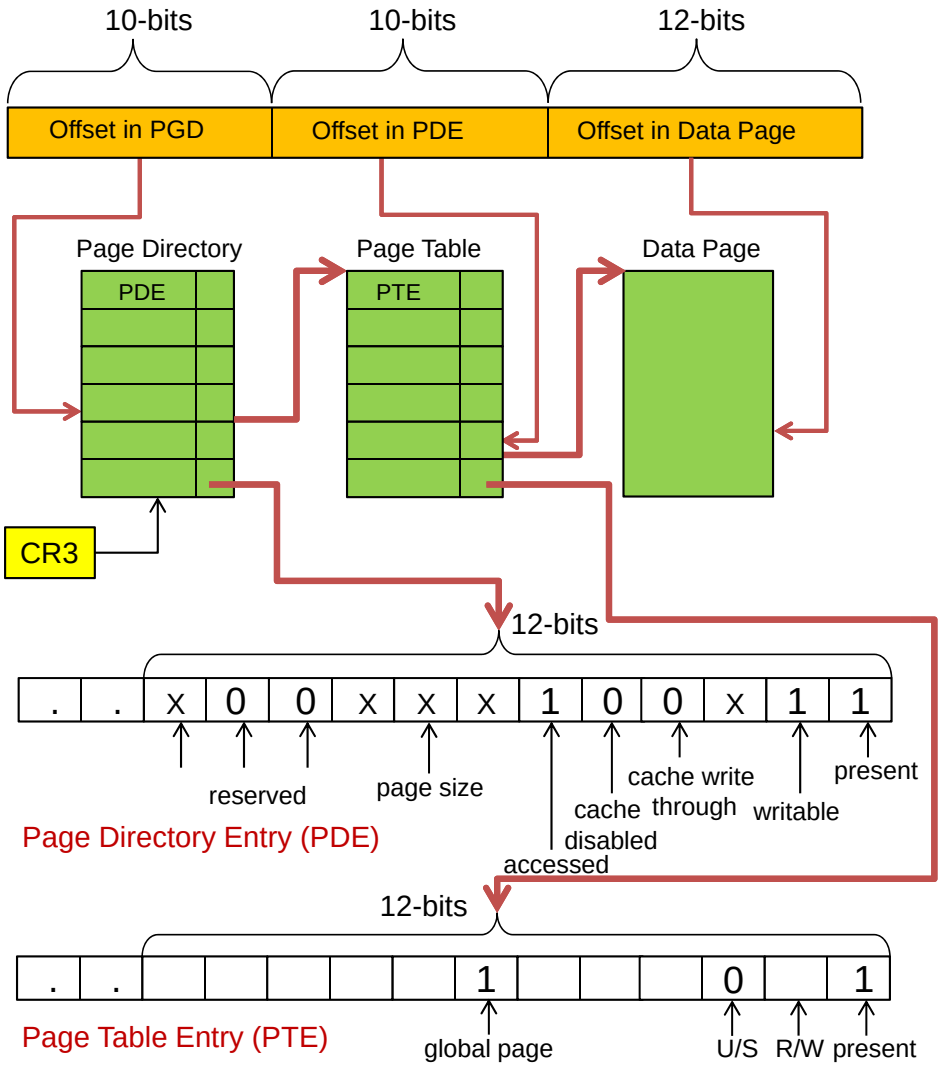
PGD (Page Global Directory) Identification



PGD (Page Global Directory) Identification



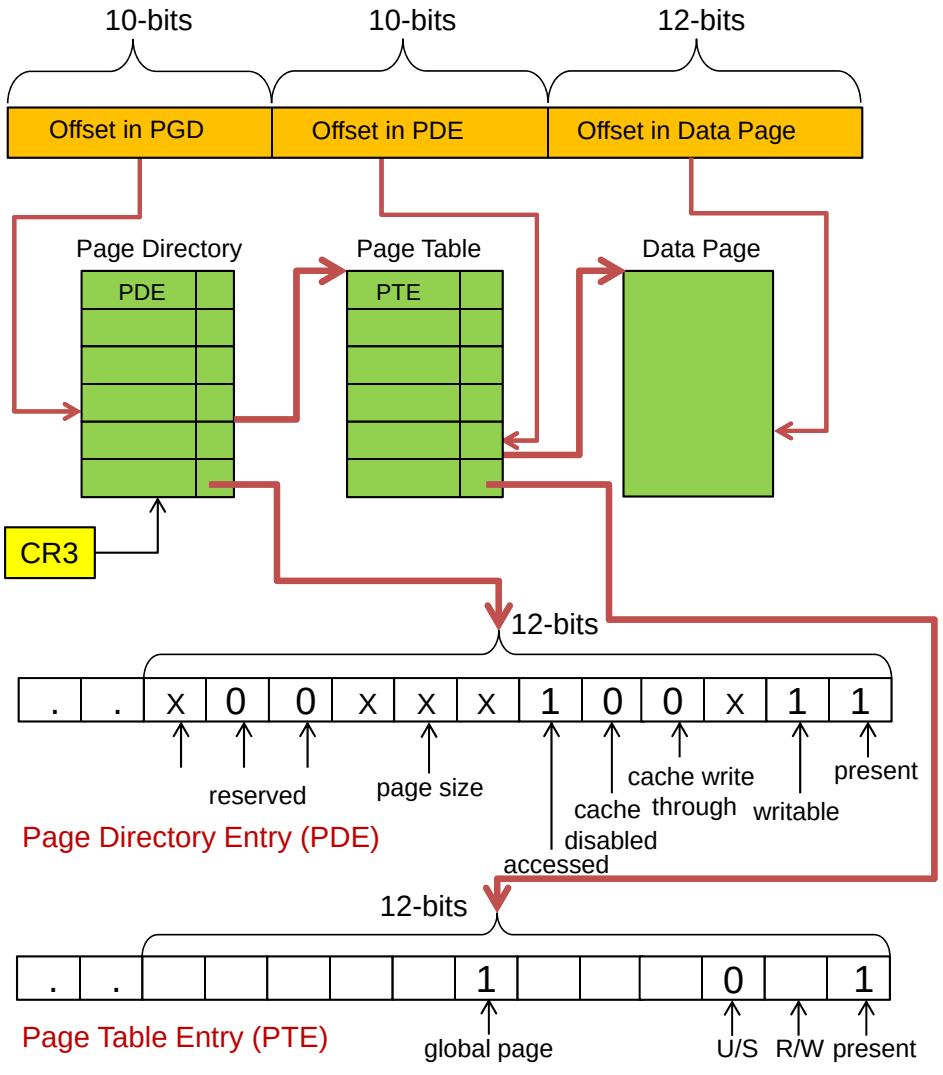
PGD (Page Global Directory) Identification



PGD Signature

- Three-layer points-to relation
- Unique SigGraph [NDSS'11] Signatures.

PGD (Page Global Directory) Identification



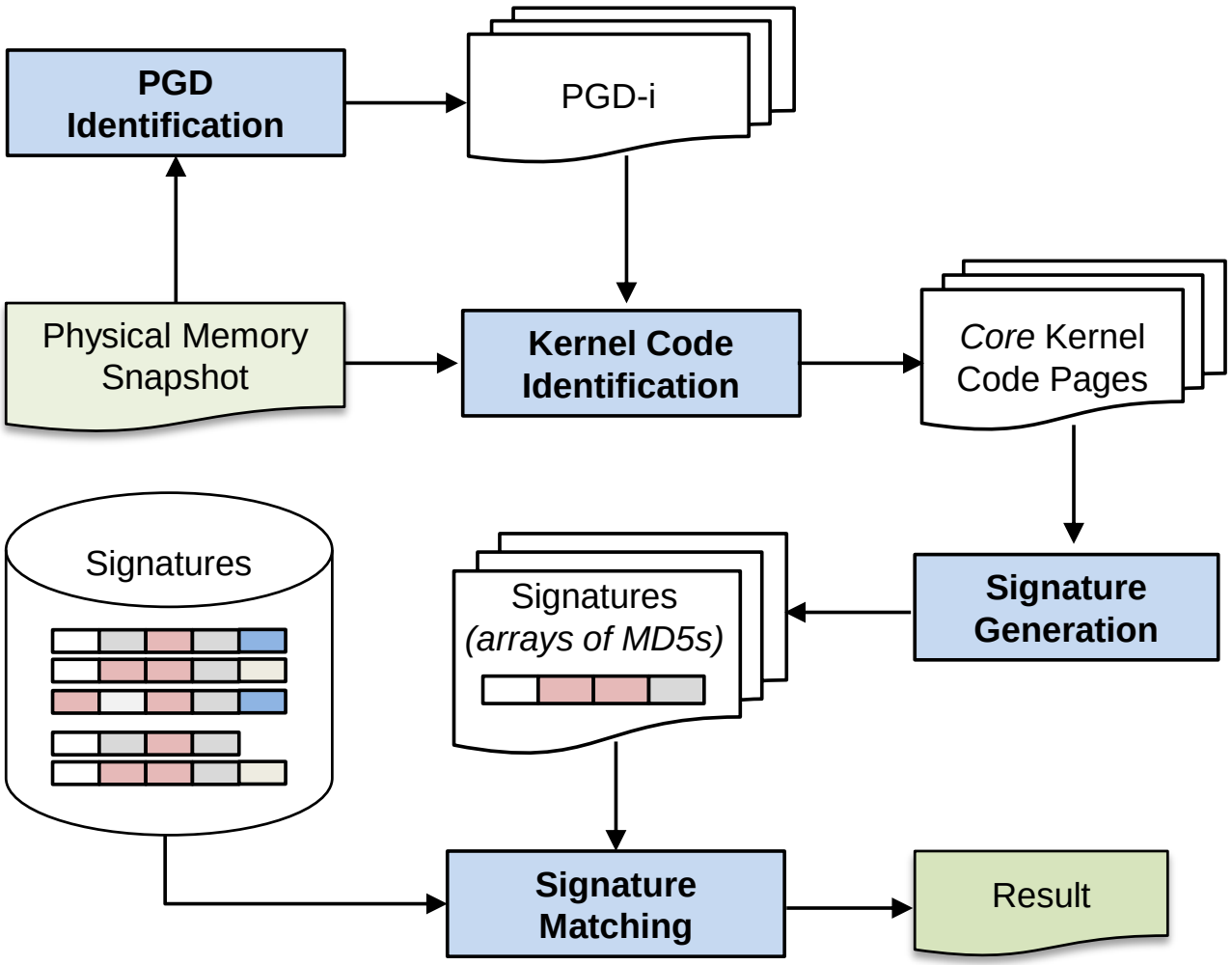
PGD Signature

- Three-layer points-to relation
- Unique SigGraph [NDSS'11] Signatures.

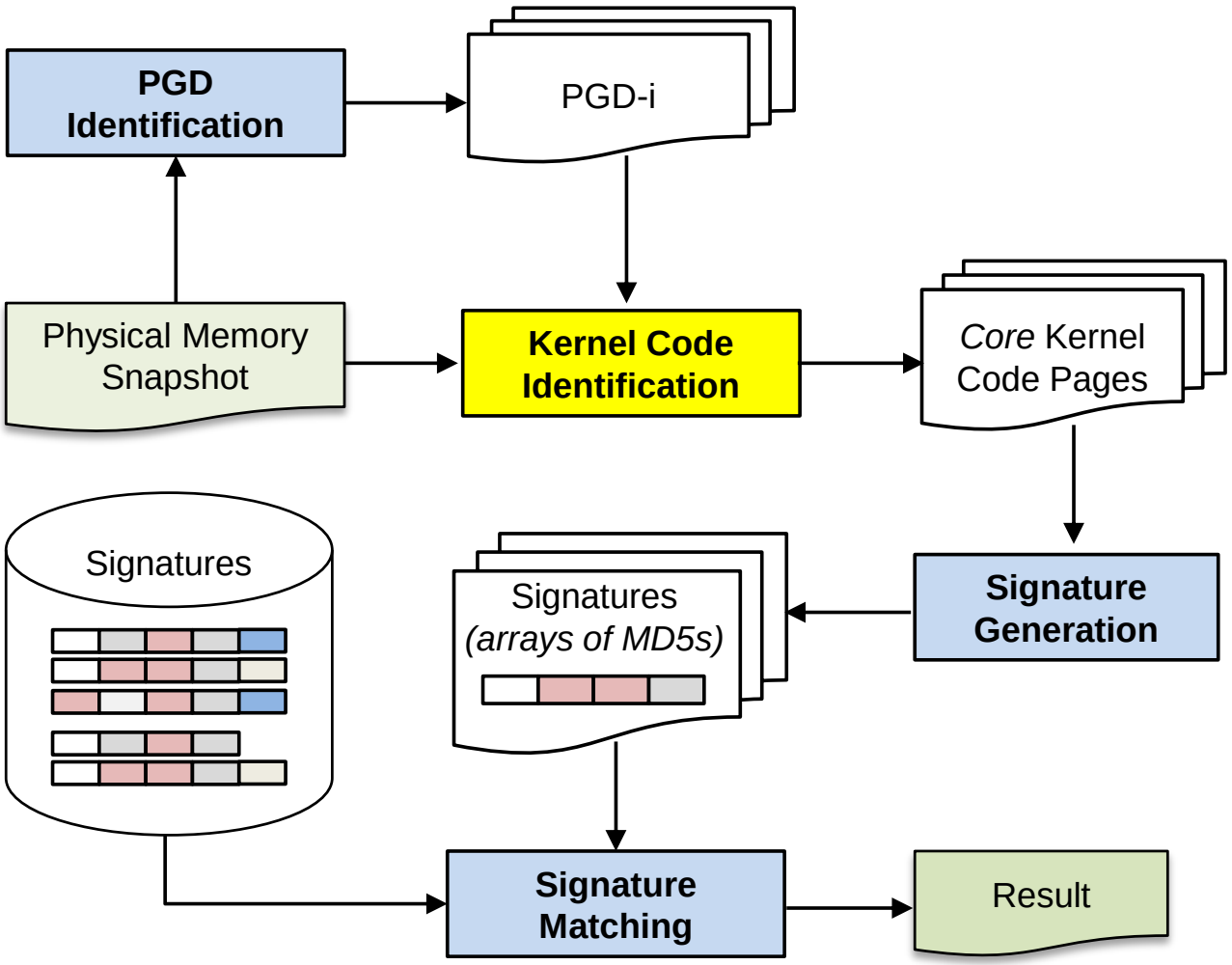
Alternative Approach

- Extract CR3 when taking the memory snapshot

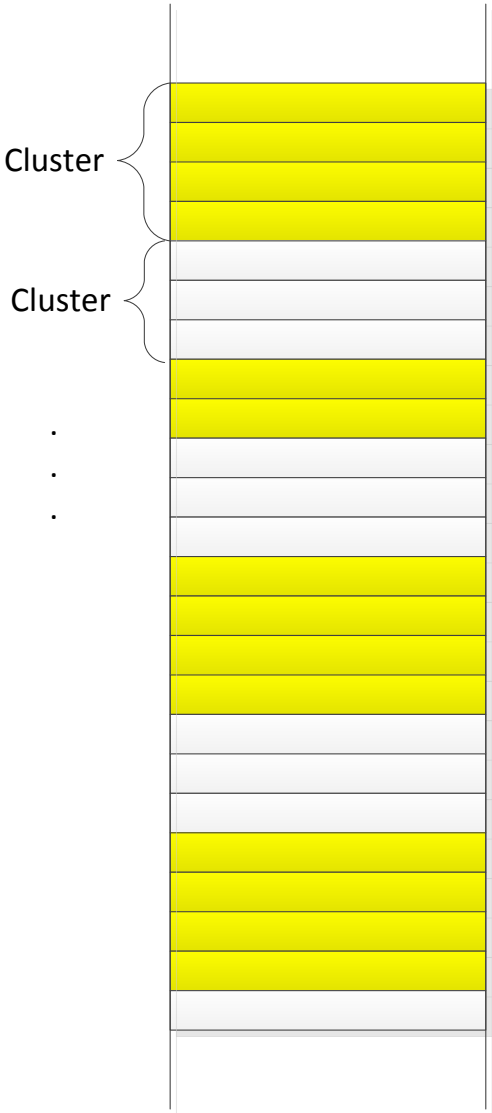
Core Kernel Code Identification

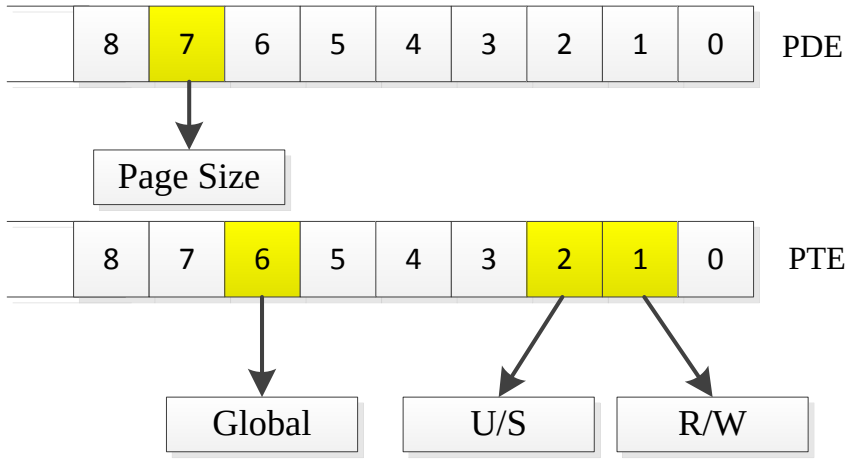


Core Kernel Code Identification



Core Kernel Code Identification: Step I



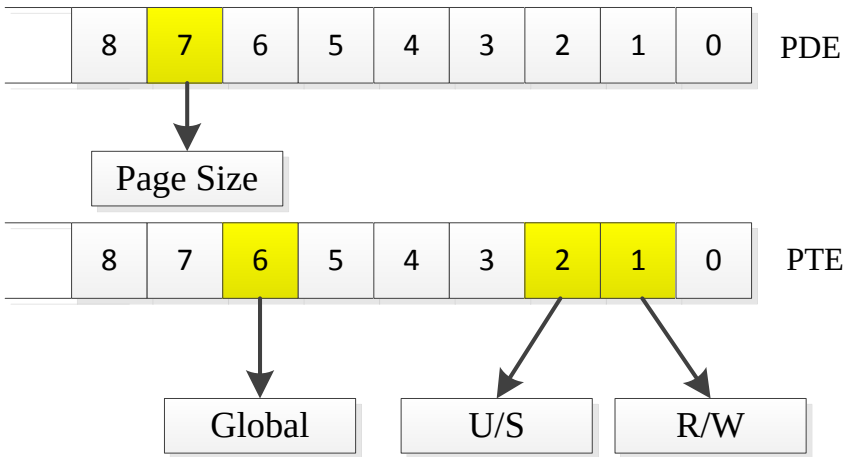


Page Properties

- Read Only $\Longleftrightarrow$ Writable
- User $\Longleftrightarrow$ System
- Global $\Longleftrightarrow$ Non-Global
- Page size: 4M $\Longleftrightarrow$ 4K

Core Kernel Code Identification: Step I

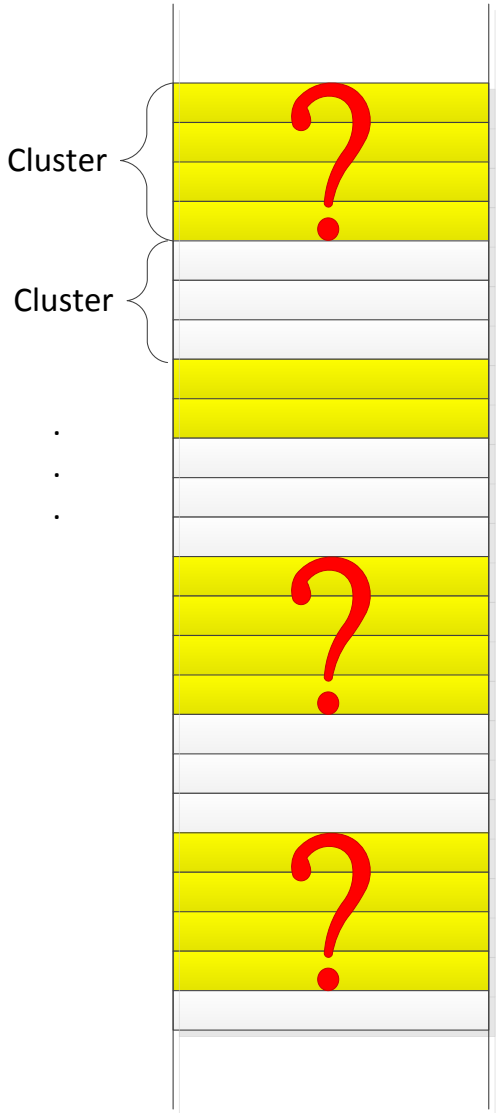
OS-kernels	$\lvert C_K \rvert$
Win-XP	883
Win-XP (SP2)	952
Win-XP (SP3)	851
Win-Vista	2310
Win-7	2011
Win-2003 Server	1028
Win-2003 Server (SP2)	1108
Win-2008 Server	1804
Win-2008 Server (SP2)	1969
FreeBSD-8.0	350
FreeBSD-8.3	412
FreeBSD-9.0	360
OpenBSD-4.7	187
OpenBSD-4.8	833
OpenBSD-5.1	1195
NetBSD-4.0	225
NetBSD-5.1.2	210
Linux-2.6.26	69
Linux-2.6.36.1	36
Linux-2.6.36.2	36
Linux-2.6.36.3	36
Linux-2.6.36.4	36
Linux-3.0.4	183



Page Properties

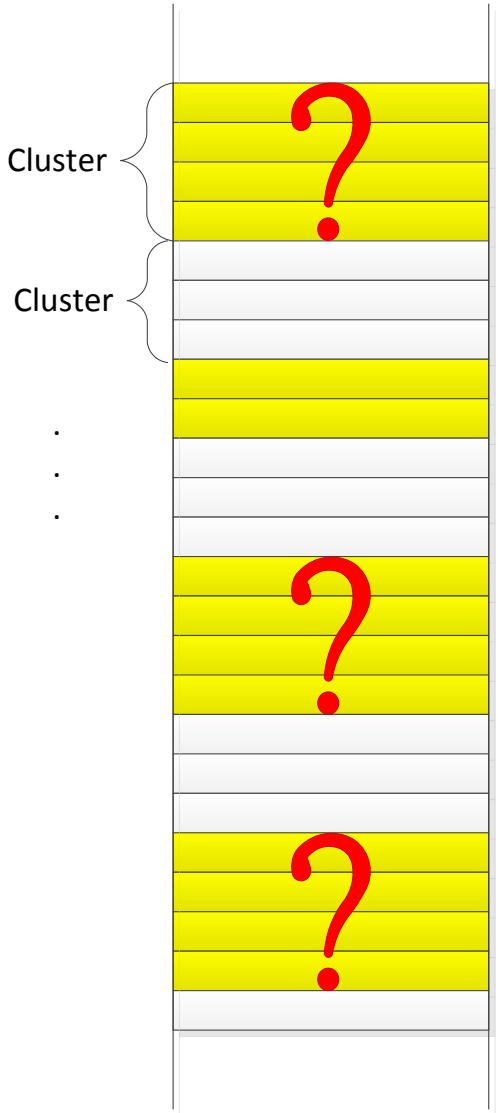
- Read Only $\Longleftrightarrow$ Writable
- User $\Longleftrightarrow$ System
- Global $\Longleftrightarrow$ Non-Global
- Page size: 4M $\Longleftrightarrow$ 4K

Core Kernel Code Identification: Step II



- Which cluster contains the main kernel code?

Core Kernel Code Identification: Step II



- Which cluster contains the main kernel code?

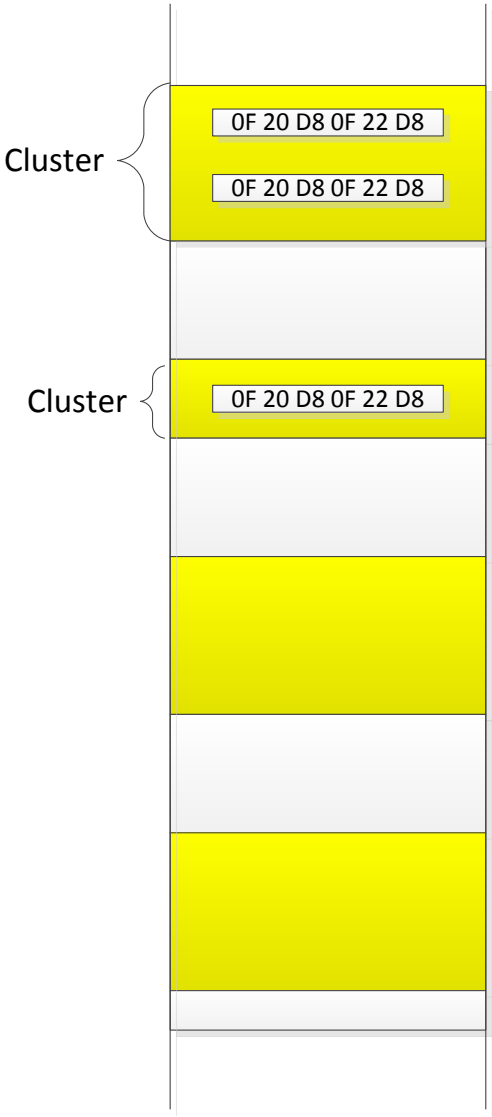
Search system instruction sequences

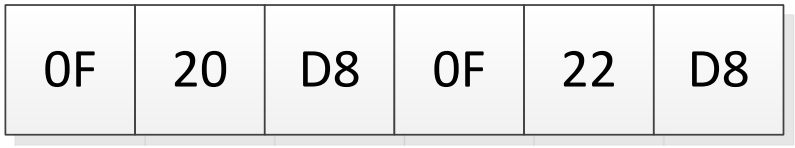
- Appearing in main kernel code
- Having unique pattern
- Not in kernel modules

X86 System Instruction Distributions in Kernel Pages

System Instructions	Inst. Length	Linux-2.6.32		Windows-XP		FreeBSD-9.0		OpenBSD-5.1	
		#Inst.	#pages	#Inst.	#pages	#Inst.	#pages	#Inst.	#pages
LLDT	3	17	10	4	3	5	3	5	4
SLDT	3	1	1	1	1	1	1	2	2
LGDT	3	10	8	1	1	1	1	3	2
SGDT	3	4	4	5	4	1	1	2	2
LTR	3	2	2	2	2	6	5	5	3
STR	3	2	2	2	2	1	1	1	1
LIDT	3	7	6	2	2	5	4	5	3
SIDT	3	2	2	5	4	1	1	2	2
MOV CR0	3	68	16	65	21	33	8	45	12
MOV CR2	3	5	5	2	2	2	2	12	5
MOV CR3	**3**	**70**	**18**	**24**	**10**	**49**	**12**	**17**	**6**
MOV CR4	3	94	23	22	7	25	7	24	8
SMSW	4	0	0	0	0	5	1	0	0
LMSW	3	0	0	0	0	5	1	0	0
CLTS	2	6	5	3	1	6	1	7	2
MOV DRn	3	0	0	262	8	0	0	0	0
INVD	2	0	0	0	0	5	1	2	1
WBINVD	2	28	14	6	3	15	8	14	8
INVLPG	3	7	3	4	3	24	10	14	4
HLT	1	12	6	1	1	5	5	4	1
RSM	2	0	0	0	0	0	0	0	0
RDMSR3	2	113	25	1	1	76	17	79	16
WRMSR3	2	111	28	1	1	51	15	54	17
RDPMC4	2	0	0	0	0	0	0	1	1
RDTSC3	2	26	12	21	7	14	4	5	3
RDTSCP7	3	0	0	0	0	0	0	0	0
XGETBV	3	0	0	0	0	0	0	0	0
XSETBV	3	3	3	0	0	0	0	0	0

Core Kernel Code Identification: Step II





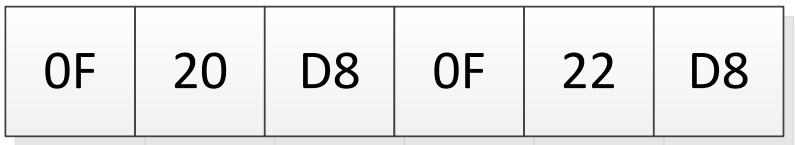
6 bytes system instruction sequence

Search System Instruction

0F 20 D8: mov EAX, CR3;
0F 22 D8: mov CR3, EAX;
This instruction sequence is used for
TLB flush

Core Kernel Code Identification: Step II

OS-kernels	$\|C_K\|$	$\|C_{Kk}\|$
Win-XP	883	2
Win-XP (SP2)	952	2
Win-XP (SP3)	851	2
Win-Vista	2310	1
Win-7	2011	2
Win-2003 Server	1028	2
Win-2003 Server (SP2)	1108	2
Win-2008 Server	1804	1
Win-2008 Server (SP2)	1969	1
FreeBSD-8.0	350	1
FreeBSD-8.3	412	1
FreeBSD-9.0	360	1
OpenBSD-4.7	187	1
OpenBSD-4.8	833	1
OpenBSD-5.1	1195	1
NetBSD-4.0	225	1
NetBSD-5.1.2	210	1
Linux-2.6.26	69	1
Linux-2.6.36.1	36	1
Linux-2.6.36.2	36	1
Linux-2.6.36.3	36	1
Linux-2.6.36.4	36	1
Linux-3.0.4	183	2



6 bytes system instruction sequence

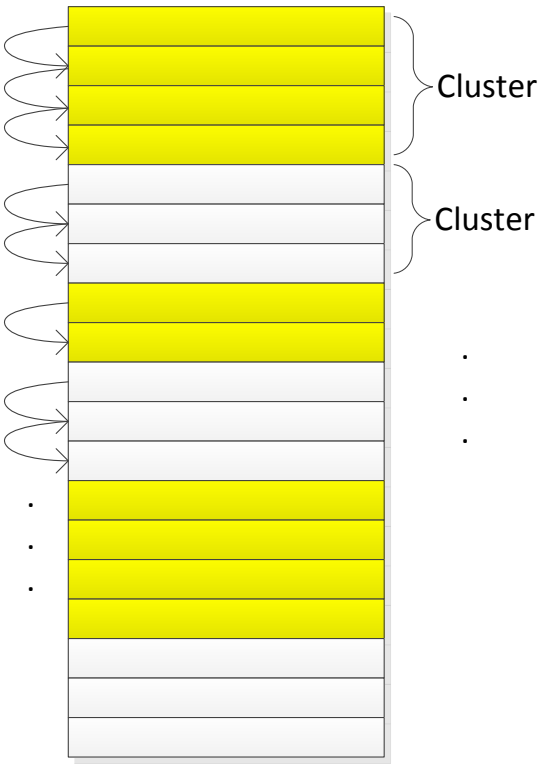
Search System Instruction

0F 20 D8: mov EAX, CR3;

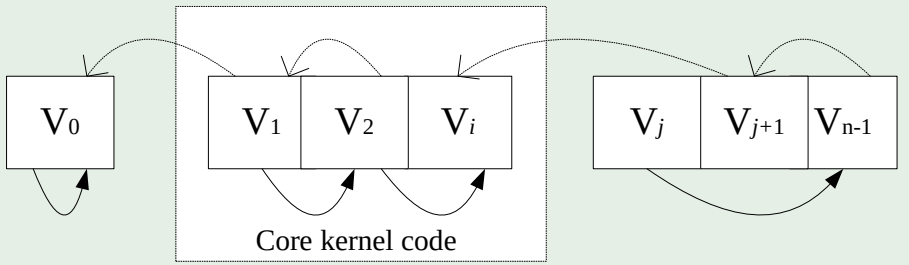
0F 22 D8: mov CR3, EAX;

This instruction sequence is used for TLB flush

Core Kernel Code Identification: Step III



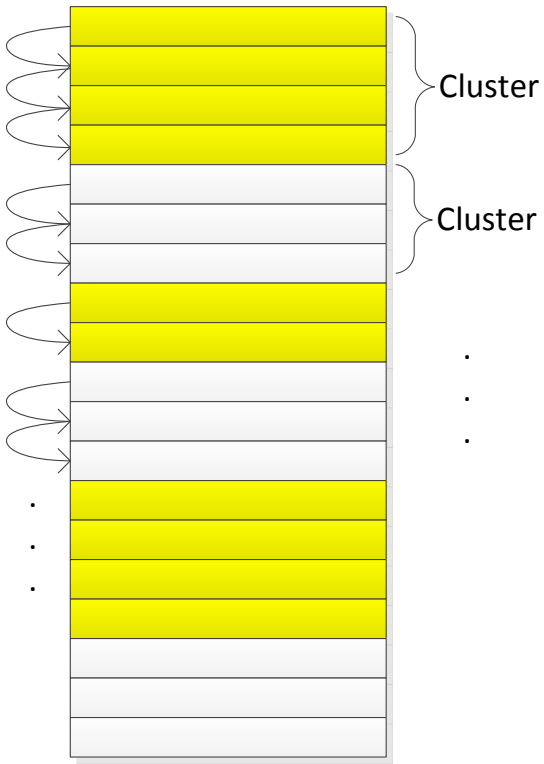
Core Kernel Code Clustering



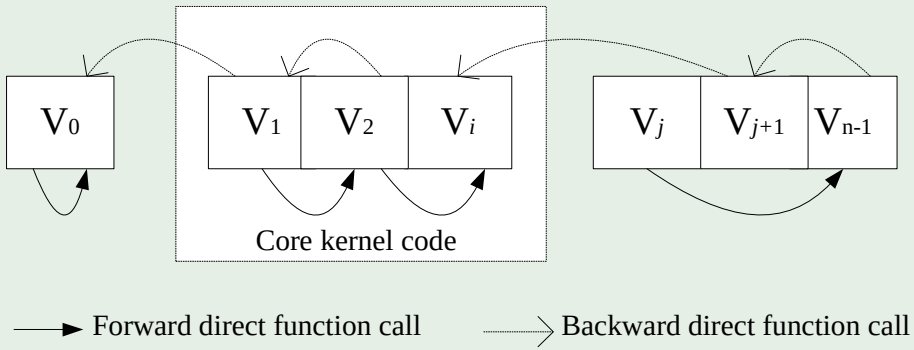
Forward direct function call

Backward direct function call

Core Kernel Code Identification: Step III



Core Kernel Code Clustering



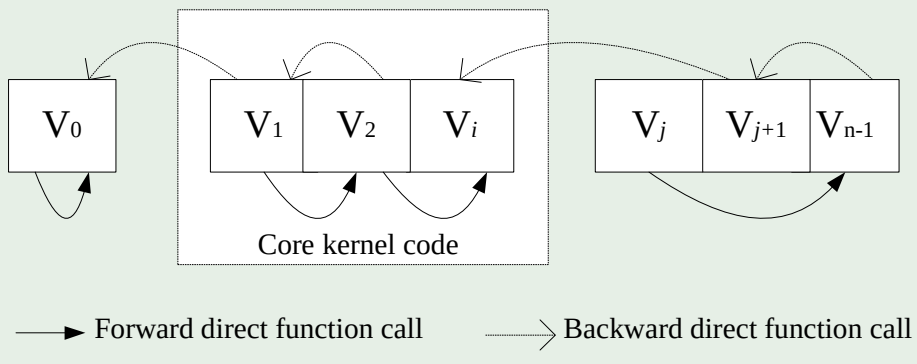
Forward Direct Function Call

A direct forward function call is a call instruction whose operand is a positive value (e.g., the case for `e8 2a 25 38 00`)

Core Kernel Code Identification: Step III

OS-kernels	$\|T\|$	#Pages'
Win-XP	16	384
Win-XP (SP2)	13	421
Win-XP (SP3)	14	423
Win-Vista	5	807
Win-7	1	280
Win-2003 Server	9	659
Win-2003 Server (SP2)	6	563
Win-2008 Server	9	849
Win-2008 Server (SP2)	6	856
FreeBSD-8.0	2	2959
FreeBSD-8.3	2	3966
FreeBSD-9.0	3	2281
OpenBSD-4.7	4	1631
OpenBSD-4.8	3	1934
OpenBSD-5.1	3	1593
NetBSD-4.0	3	1995
NetBSD-5.1.2	9	1792
Linux-2.6.26	2	811
Linux-2.6.36.1	1	1023
Linux-2.6.36.2	1	1023
Linux-2.6.36.3	1	1023
Linux-2.6.36.4	1	1023
Linux-3.0.4	1	1023

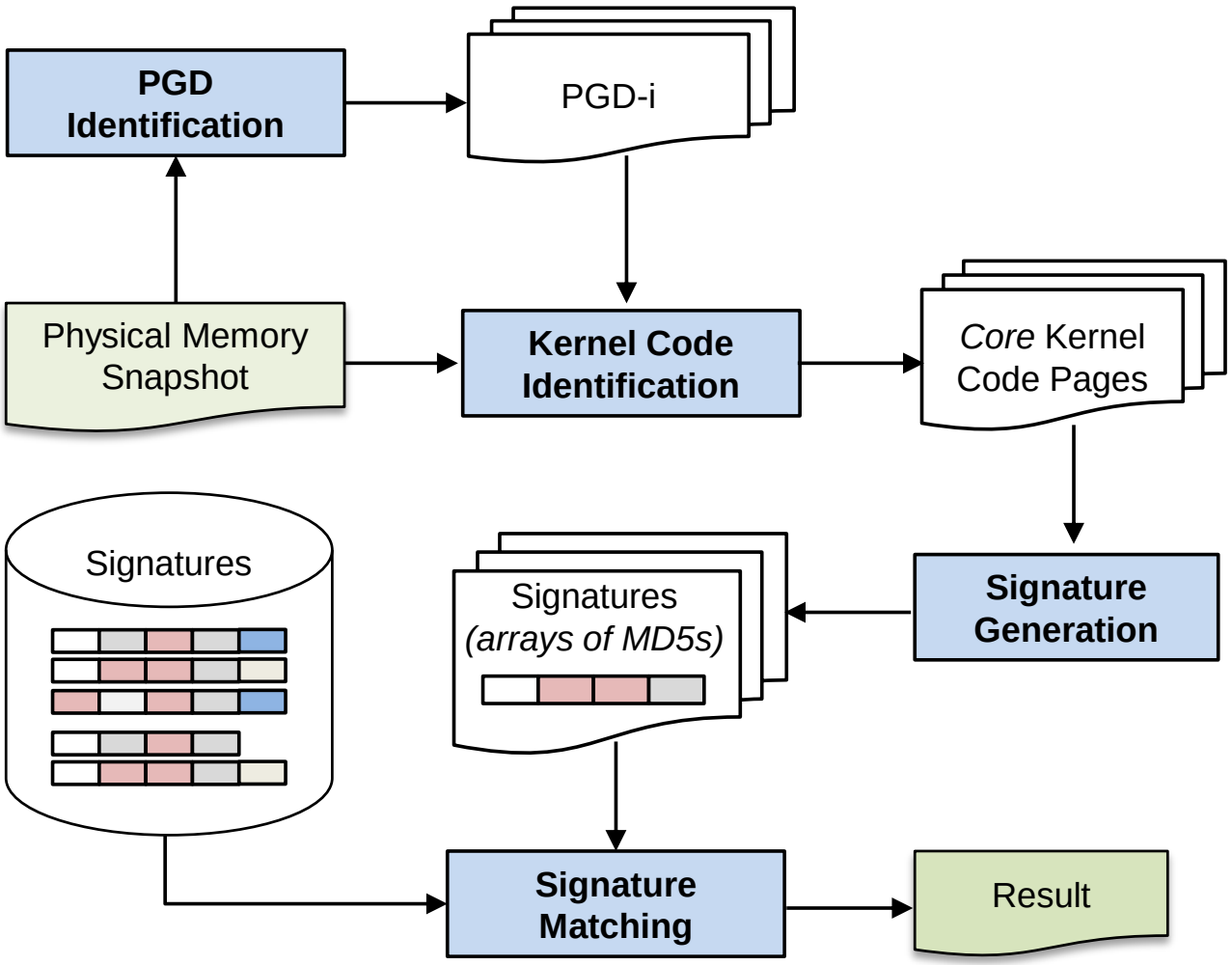
Core Kernel Code Clustering



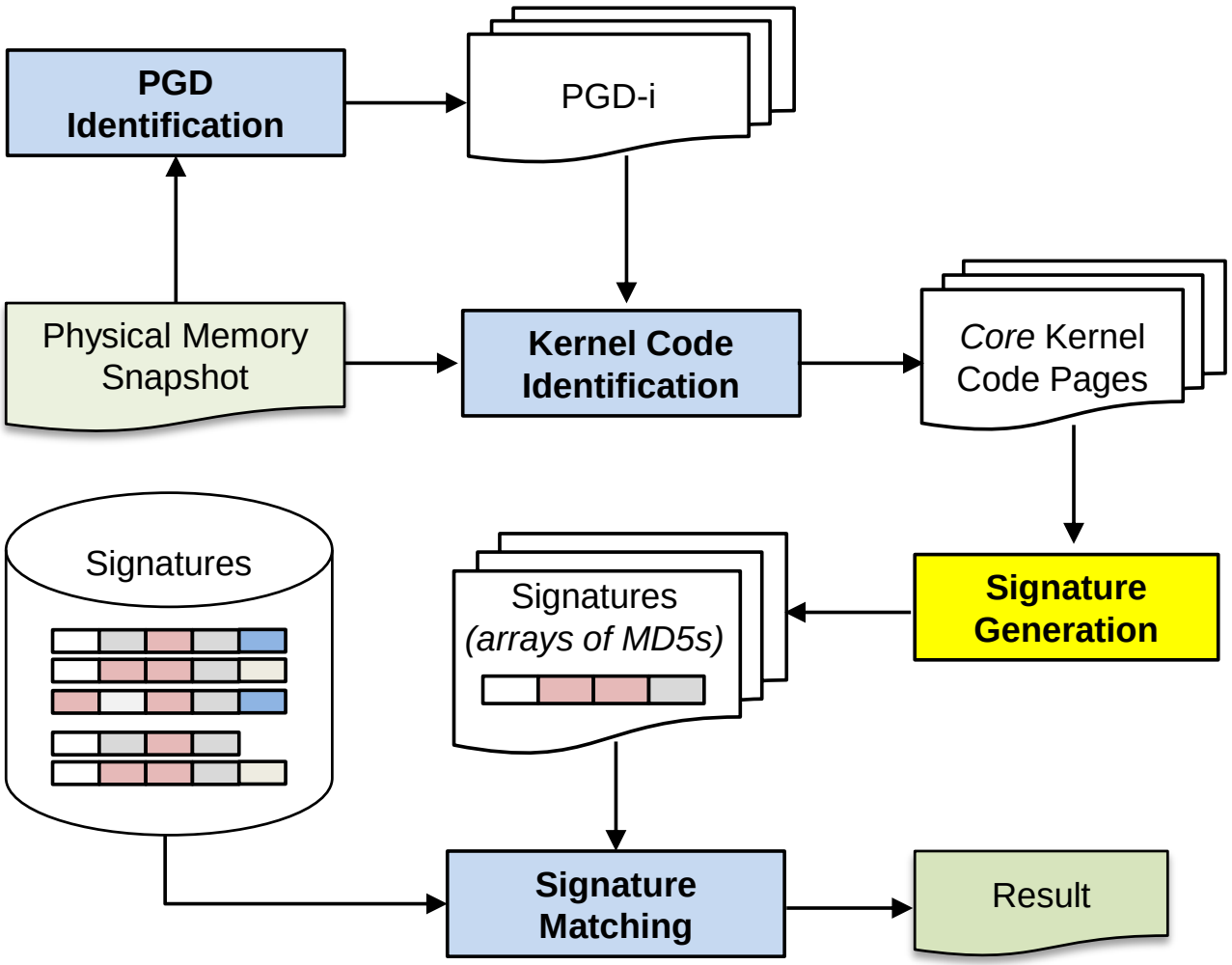
Forward Direct Function Call

A direct forward function call is a call instruction whose operand is a positive value (e.g., the case for `e8 2a 25 38 00`)

Signature Generation



Signature Generation



Signature Generation

- Can we directly hash the code page?

Signature Generation

- Can we directly hash the code page?

Distill Operand to Neutralize the Effect of ASR (Code Rebase)

```
0x828432b6: 33 f6                   xor esi, esi
0x828432b8: 83 3d 38 fe 99 82 02    cmp dword ptr ds[0x8299fe38], 0x2
0x828432bf: 0f 87 95 00 00 00       jnbe 0x8284335a
0x828432c5: 8b 0d 3c fe 99 82       mov ecx, dword ptr ds[0x8299fe3c]
0x828432cb: 33 c0                   xor eax, eax
...
0x82843432: e8 e4 9e 09 00          call 0x828dd31b
```

```
0x828182b6: 33 f6                   xor esi, esi
0x828182b8: 83 3d 38 4e 97 82 02    cmp dword ptr ds[0x82974e38], 0x2
0x828182bf: 0f 87 95 00 00 00       jnbe 0x8281835a
0x828182c5: 8b 0d 3c 4e 97 82       mov ecx, dword ptr ds[0x82974e3c]
0x828182cb: 33 c0                   xor eax, eax
...
0x82818432: e8 e4 9e 09 00          call 0x828b231b
```

Correlative Disassembling

Correlative Disassembling

```
0xc1087c86:  e8 25 2c 00 00        call   0xc108a8b0
...
0xc108a8b0:  55                    push   ebp
0xc108a8b1:  89 e5                 mov    ebp, esp
```

(a) Linux Kernel

```
0x806eee0a:  e8 3d 69 00 00        call   0x806f574c
...
0x806f574c:  8b ff                 mov    edi, edi
0x806f574e:  55                    push   ebp
0x806f574f:  8b ec                 mov    ebp, esp
```

(b) Windows Kernel

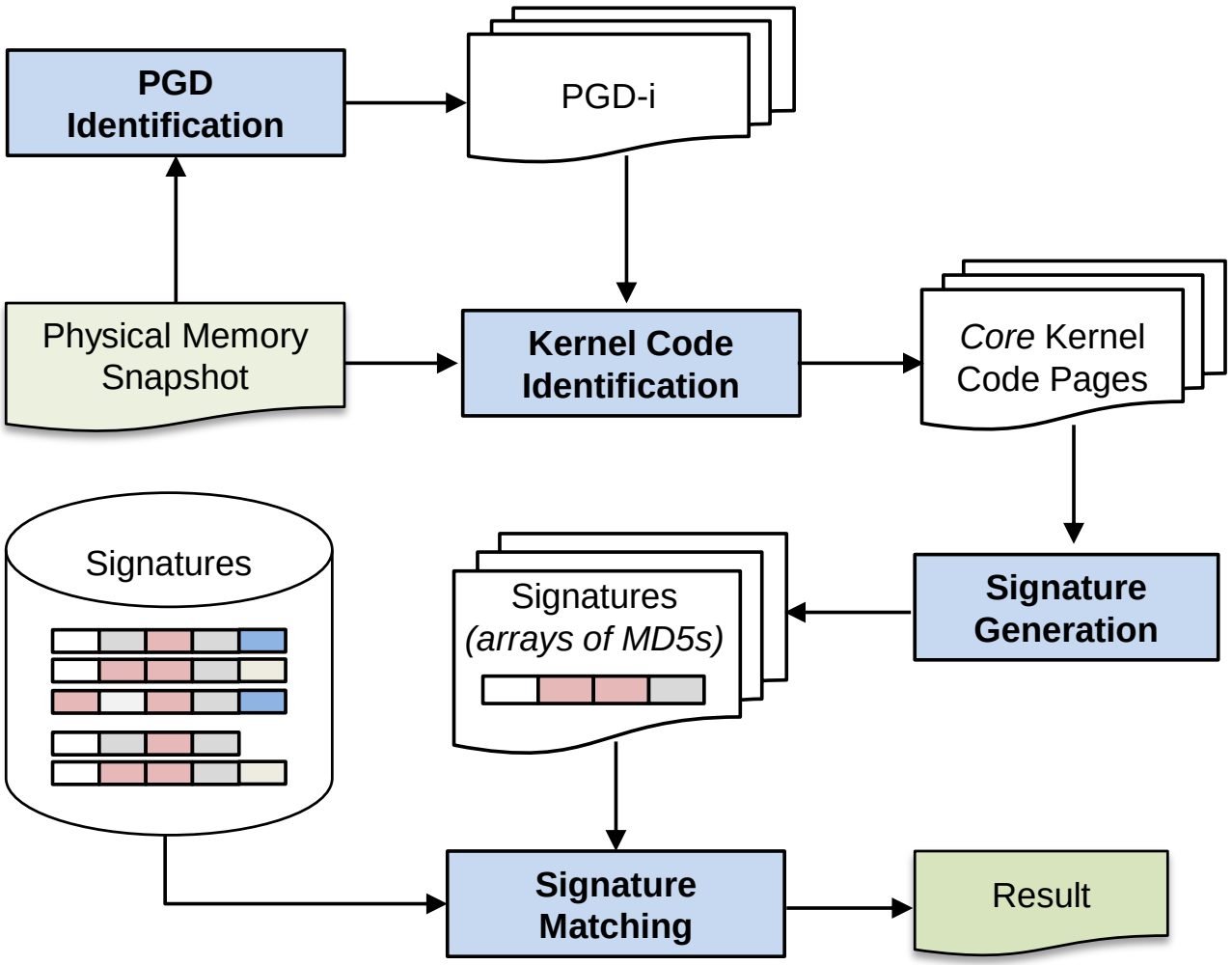
```
0xc04d675f:  e8 0c cf 00 00        call   0xc04e3670
...
0xc04e3670:  55                    push   ebp
0xc04e3671:  89 e5                 mov    ebp,esp
```

(c) FreeBSD/OpenBSD/NetBSD Kernel

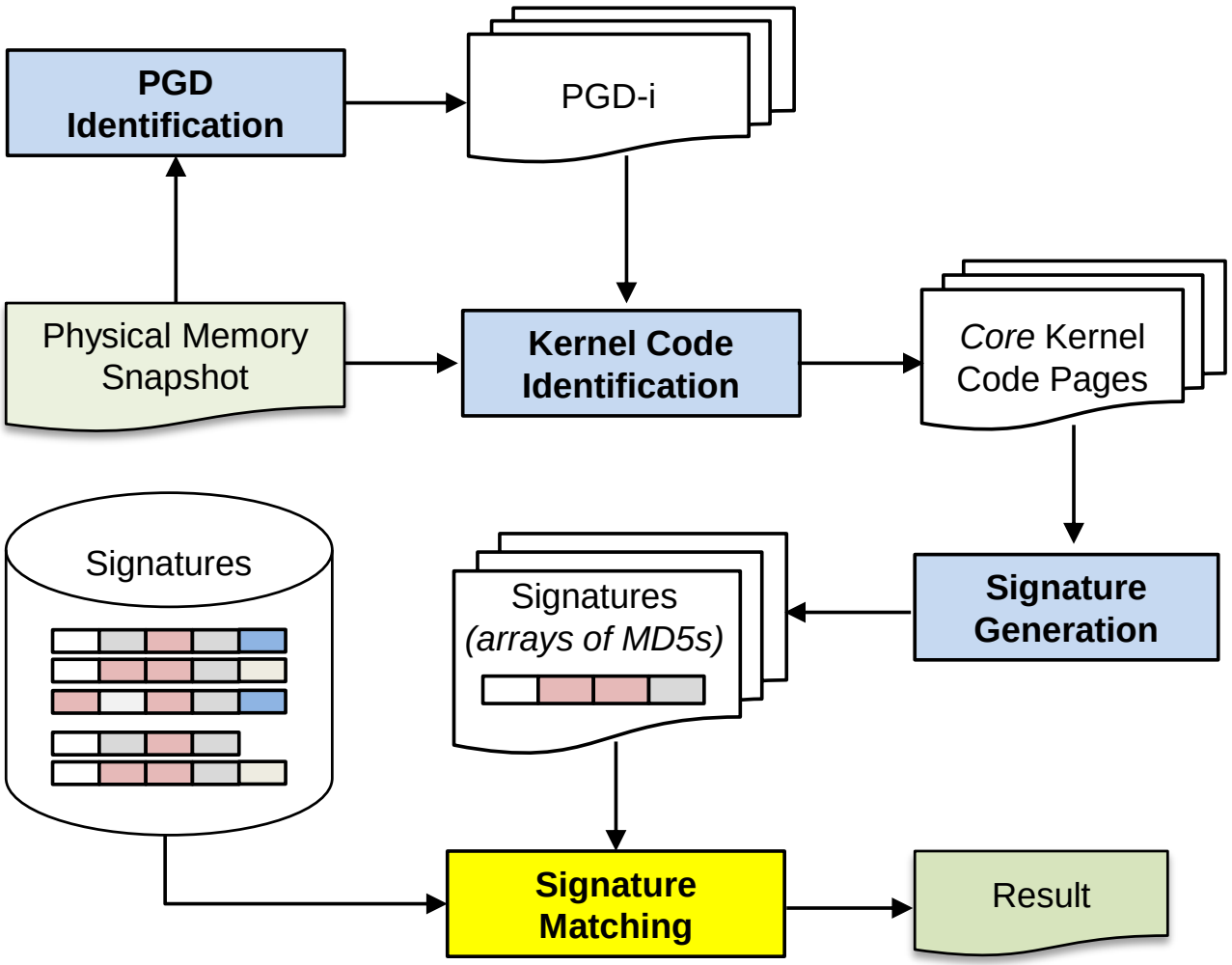
Algorithm

1. Search machine code `e8 x x x x`.
2. Compute callee address.
3. If the callee address has the pattern of a function prologue, start to disassemble the target page from the callee address.
4. Stop when encountering a `ret` or a direct or indirect `jmp` instruction.

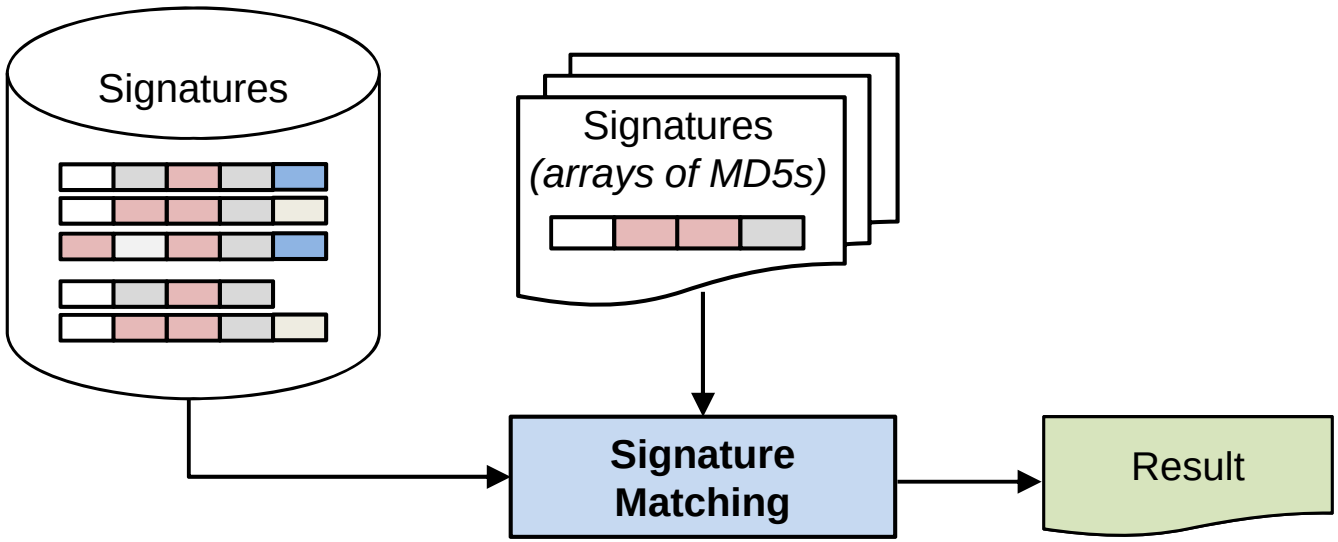
Signature Matching



Signature Matching



Signature Matching



Signature Matching

- Works similar to KMP [Knuth, 1977] string matching algorithm except the element of the string is a 32-bytes MD5 Value

Evaluation

Implementation

- Implemented with 4.5K lines of C code
- Correlative disassembler is based on XED library.

Evaluation

Implementation

- Implemented with 4.5K lines of C code
- Correlative disassembler is based on XED library.

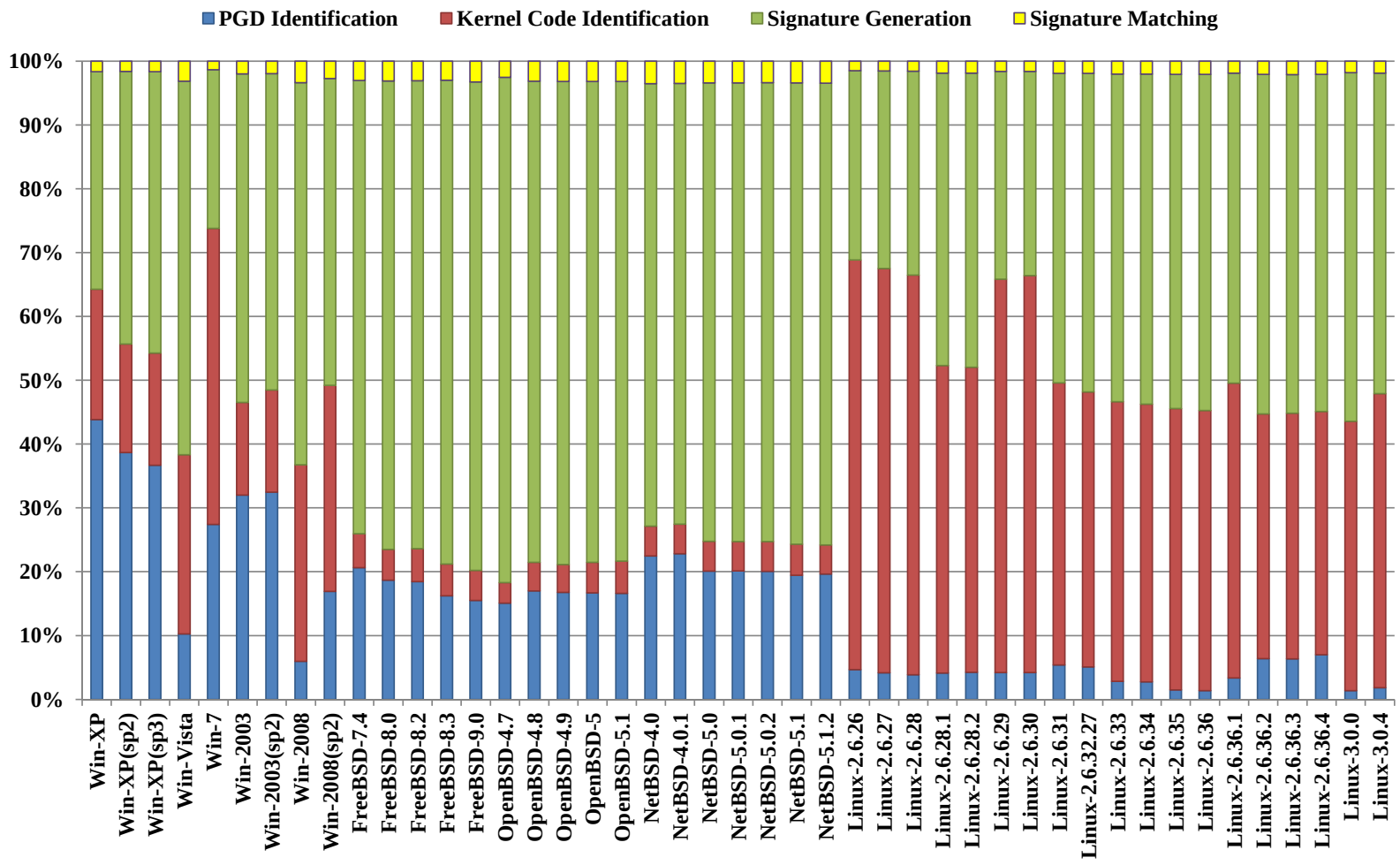
Experimental Setup

- Using over 45 OS kernels from five widely used OS families (Microsoft Windows, Linux, *BSD).
- Comparing with other state-of-the-art OS fingerprinting techniques: UFO and IDT.

Effectiveness

OS-kernels	#PGD	$\|C_K\|$	$\|C_{Kk}\|$	#Pages	$\|T\|$	#Pages'	#Sig-Gen	#Sig-Match
Win-XP	12	883	2	1024	16	384	232	1
Win-XP (SP2)	15	952	2	1024	13	421	277	1
Win-XP (SP3)	15	851	2	1024	14	423	282	1
Win-Vista	24	2310	1	1024	5	807	453	1
Win-7	18	2011	2	280	1	280	178	1
Win-2003 Server	20	1028	2	1024	9	659	374	1
Win-2003 Server (SP2)	19	1108	2	1024	6	563	342	1
Win-2008 Server	20	1804	1	1024	9	849	542	2
Win-2008 Server (SP2)	21	1969	1	1024	6	856	536	2
FreeBSD-8.0	20	350	1	3072	2	2959	1122	1
FreeBSD-8.3	18	412	1	4096	2	3966	1187	1
FreeBSD-9.0	21	360	1	4096	3	2281	1318	1
OpenBSD-4.7	20	187	1	1634	4	1631	1163	1
OpenBSD-4.8	12	833	1	1936	3	1934	1258	1
OpenBSD-5.1	7	1195	1	1596	3	1593	1293	1
NetBSD-4.0	16	225	1	2006	3	1995	1069	60
NetBSD-5.1.2	13	210	1	2048	9	1792	1183	24
Linux-2.6.26	82	69	1	812	2	811	526	1
Linux-2.6.36.1	78	36	1	1024	1	1023	926	5
Linux-2.6.36.2	78	36	1	1024	1	1023	925	31
Linux-2.6.36.3	76	36	1	1024	1	1023	930	31
Linux-2.6.36.4	81	36	1	1024	1	1023	929	22
Linux-3.0.4	73	183	2	1024	1	1023	918	1
mean	43.24	481.57	1.17	1588.53	4.97	1351.57	879.91	8.5

Performance Overhead of Each Component



The signature generation process takes 1.50 seconds on average.

Experiment Result

OS-Kernels	UFO	IDT-based	OS-Sommelier
Win-XP	✗	✗	✓
Win-XP (SP2)	✗	✗	✓
Win-XP (SP3)	✗	✗	✓
Win-Vista	✓	✓	✓
Win-7	✓	✓	✓
Win-2003 Server	✗	✓	✓
Win-2003 Server (SP2)	✗	✓	✓
Win-2008 Server	✓	✓	✓
Win-2008 Server (SP2)	✓	✓	✓

Table: Experiment with Windows kernel.

Experiment Result

OS-Kernels	UFO	IDT-based	OS-Sommelier
FreeBSD 7.4	✓	✗	✓
FreeBSD 8.0	✓	✓	✓
FreeBSD 8.2	✓	✗	✓
FreeBSD 8.3	✓	✓	✓
FreeBSD 9.0	✓	✓	✓
OpenBSD 4.7	✓	✗	✓
OpenBSD 4.8	✓	✗	✓
OpenBSD 4.9	✓	✗	✓
OpenBSD 5.0	✓	✓	✓
OpenBSD 5.1	✓	✗	✓
NetBSD 4.0	✗	✗	✓
NetBSD 4.0.1	✗	✗	✓
NetBSD 5.0	✓	✓	✓
NetBSD 5.0.1	✗	✗	✓
NetBSD 5.0.2	✗	✗	✓
NetBSD 5.1	✗	✗	✓
NetBSD 5.1.2	✗	✗	✓

Table: Experiment with BSD Family kernel.

Experiment Result

OS-Kernels	UFO	IDT-based	OS-Sommelier
Linux-2.6.26	✓	✓	✓
Linux-2.6.27	✓	✓	✓
Linux-2.6.28	✓	✓	✓
Linux-2.6.28.1	✓	✓	✓
Linux-2.6.28.2	✓	✓	✓
Linux-2.6.29	✓	✓	✓
Linux-2.6.30	✓	✓	✓
Linux-2.6.31	✓	✓	✓
Linux-2.6.32	✓	✓	✓
Linux-2.6.33	✓	✓	✓
Linux-2.6.34	✓	✓	✓
Linux-2.6.35	✓	✓	✓
Linux-2.6.36.1	✗	✓	✓
Linux-2.6.36.2	✗	✓	✓
Linux-2.6.36.3	✗	✓	✓
Linux-2.6.36.4	✓	✓	✓
Linux-3.0.0	✓	✓	✓
Linux-3.0.4	✓	✓	✓

Table: Experiment with Linux kernel.

Limitation and Future work

Limitations

- Too sensitive
- Kernel recompilation
- Obfuscating the kernel code

Limitation and Future work

Limitations

- Too sensitive
- Kernel recompilation
- Obfuscating the kernel code

Future Work

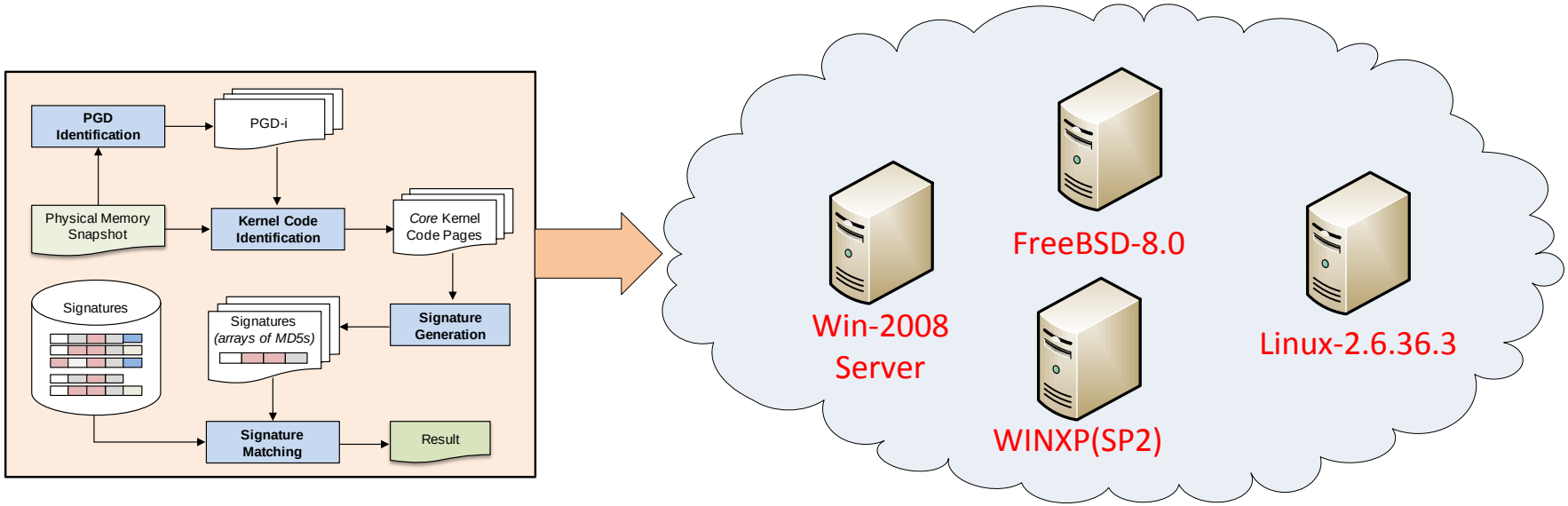
- Micro-kernel (MINIX)?

Conclusion

OS-SOMMELIER

- A physical memory-only based system for OS fingerprinting in Cloud.
  - Precise
  - Efficient
  - Robust

Thank you



To contact us

{yufei.gu, yangchun.fu, zhiqiang.lin}@utdallas.edu
{arprakas, heyin}@syr.edu