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
What is OS Fingerprinting

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

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]

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
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
Basic Approaches for OS Fingerprinting

### Basic Approaches

1. Network
2. File System
3. CPU State
4. Memory
5. Their Combinations
Network-based OS Fingerprinting

TCP/ICMP Packet

TCP/ICMP Packet
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
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

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
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

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 Register based OS Fingerprinting

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 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

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
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]
Motivation
State-of-the-Art
Detailed Design
Evaluation
Conclusion

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

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
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
**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
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 Identification
- Physical Memory Snapshot
- Kernel Code Identification
- Core Kernel Code Pages
- Signature Generation
- Signatures (arrays of MD5s)
- Signature Matching
- Result
PGD (Page Global Directory) Identification

- PGD Identification
  - Physical Memory Snapshot
  - Signatures (arrays of MD5s)
  - Signature Generation
  - Signature Matching
  - Result

- PGD-i
  - Core Kernel Code Pages
  - Kernel Code Identification
  - Core Kernel Code Pages
  - Signature Matching
  - Result
PGD (Page Global Directory) Identification

Motivation

State-of-the-Art

Detailed Design

Evaluation

Conclusion

10-bits 10-bits 12-bits
CR3
Page Directory
pte
Offset in PGD Offset in PDE Offset in Data Page
Page Table
PDE
PTE
Data Page
X     X
1    0
X 0
0    X
0 1
1    1
.     .

12-bits
present
writable
cache write
cache through
disabled
accessed
reserved
page size

Page Directory Entry (PDE)

Page Table Entry (PTE)

Global Page
U/S
R/W present

PGD Signature
Three-layer points-to
relation
Unique SigGraph
[NDSS'11]

Alternative Approach
Extract CR3 when taking the memory snapshot
PGD (Page Global Directory) Identification

**PGD Signature**
- Three-layer points-to relation
- Unique SigGraph
Motivation

State-of-the-Art

Detailed Design

Evaluation

Conclusion

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

PGD Identification → Physical Memory Snapshot → Kernel Code Identification → Core Kernel Code Pages

PGD-i → Signatures (arrays of MD5s) → Signature Generation → Signature Matching → Result
Core Kernel Code Identification

- **PGD Identification**
- **Physical Memory Snapshot**
- **Kernel Code Identification**
- **Signatures (arrays of MD5s)**
- **Signature Matching**
- **Result**
Core Kernel Code Identification: Step I

Page Properties
- Read Only ⇔ Writable
- User ⇔ System
- Global ⇔ Non-Global
- Page size: 4M ⇔ 4K
Core Kernel Code Identification: Step I

| OS-kernels                  | $|C_K|$ |
|----------------------------|------|
| 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 $\Leftrightarrow$ Writable
- User $\Leftrightarrow$ System
- Global $\Leftrightarrow$ Non-Global
- Page size: 4M $\Leftrightarrow$ 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

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

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

- Cluster
- Cluster
- Cluster
- Cluster

Core kernel code

```
V_0 V_1 V_2 V_i V_j V_j+1 V_{n-1}
```

- Forward direct function call
- Backward direct function call

Backward direct function call

Forward direct function call
Core Kernel Code Identification: Step III

Core Kernel Code Clustering

- Core kernel code
- Backward direct function call
- Forward direct function call

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

![Core Kernel Code Clustering Diagram]

- **V₀**
- **V₁**
- **V₂**
- **V_i**
- **V_j**
- **V_j+1**
- **V_n-1**

**Core kernel code**: Core kernel code is inherently clustered with the following structure:

- **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`).

- **Backward direct function call**: Direct backward function call is illustrated in the diagram.
Signature Generation

- **PGD Identification**
- **Physical Memory Snapshot**
- **Kernel Code Identification**
- **Core Kernel Code Pages**
- **Signatures** (arrays of MD5s)
- **Signature Generation**
- **Signature Matching**
- **Result**
Signature Generation

- **PGD Identification**
  - Physical Memory Snapshot

- **Kernel Code Identification**
  - Core Kernel Code Pages

- **Signature Generation**
  - Signatures (arrays of MD5s)

- **Signature Matching**
  - Result
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)

```assembly
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

Algorithm

1. Search machine code $\textbf{e8 x x x x}$.

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 $\textbf{ret}$ or a direct or indirect $\textbf{jmp}$ instruction.

---

(a) Linux Kernel

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

(b) Windows Kernel

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

(c) FreeBSD/OpenBSD/NetBSD Kernel

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

Signature Matching

PGD Identification

Physical Memory Snapshot

Kernel Code Identification

Core Kernel Code Pages

Signature Generation

Signatures

(arrays of MD5s)

Signature Matching

Result
Signature Matching

- **PGD Identification**
  - Physical Memory Snapshot
  - Core Kernel Code Pages

- **Kernel Code Identification**
  - Signatures (arrays of MD5s)

- **PGD-i**

- **Signature Generation**
- **Signature Matching**
- **Result**
Signature Matching

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

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

<table>
<thead>
<tr>
<th>OS-Kernels</th>
<th>UFO</th>
<th>IDT-based</th>
<th>OS-Sommelier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win-XP</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Win-XP (SP2)</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Win-XP (SP3)</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Win-Vista</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Win-7</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Win-2003 Server</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Win-2003 Server (SP2)</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Win-2008 Server</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Win-2008 Server (SP2)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Table:** Experiment with Windows kernel.
<table>
<thead>
<tr>
<th>OS-Kernels</th>
<th>UFO</th>
<th>IDT-based</th>
<th>OS-Sommelier</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeBSD 7.4</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>FreeBSD 8.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FreeBSD 8.2</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>FreeBSD 8.3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>FreeBSD 9.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OpenBSD 4.7</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>OpenBSD 4.8</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>OpenBSD 4.9</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>OpenBSD 5.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OpenBSD 5.1</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>NetBSD 4.0</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>NetBSD 4.0.1</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>NetBSD 5.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>NetBSD 5.0.1</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>NetBSD 5.0.2</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>NetBSD 5.1</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>NetBSD 5.1.2</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table: Experiment with BSD Family kernel.
## Experiment Result

<table>
<thead>
<tr>
<th>OS-Kernels</th>
<th>UFO</th>
<th>IDT-based</th>
<th>OS-Sommelier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linux-2.6.26</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Linux-2.6.27</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Linux-2.6.28</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Linux-2.6.28.1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Linux-2.6.28.2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Linux-2.6.29</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Linux-2.6.30</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Linux-2.6.31</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Linux-2.6.32</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Linux-2.6.33</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Linux-2.6.34</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Linux-2.6.35</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Linux-2.6.36.1</td>
<td>✔</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Linux-2.6.36.2</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Linux-2.6.36.3</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Linux-2.6.36.4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Linux-3.0.0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Linux-3.0.4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

*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