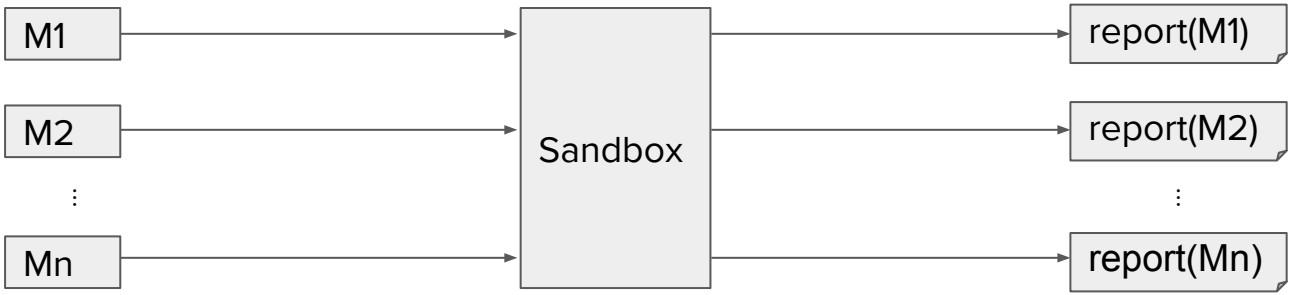
Detecting Hardware-Assisted Virtualization

Michael Brengel, Michael Backes, and Christian Rossow
mbrengel@mmci.uni-saarland.de
CISPA, Saarland University, Germany

Motivation

- Automated Malware Analysis ➜ Sandboxes



- The sandbox is usually a virtual machine (VM)
- The average victim is usually **not** running a VM

  ⇒ Evading the VM = Evading analysis without losing victims

➢ Transparency of the VM/Hypervisor?

Virtualization/Hypervisor Types

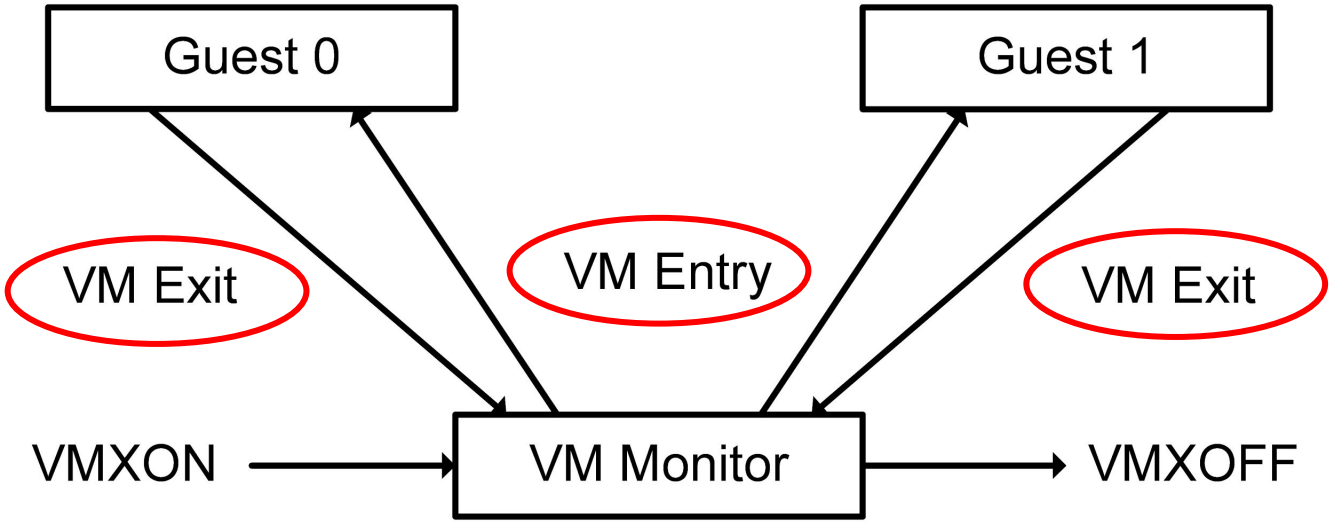
1. Software Emulators
   Emulate the machine completely in software
   ⇒ QEMU, BOCHS, ...

2. Reduced Privilege Guests
   Execute the guest OS at a lower privilege level
   ⇒ VMWare, VirtualBox, ... (if your CPU is really old)

3. Hardware-Assisted Virtualization
   Virtualization implemented in the CPU
   ⇒ Intel VT-x, AMD-V

stealthier
+
more efficient

Intel VT-x


Guest 0
Guest 1
VM Exit
VM Entry
VM Exit
VMXON
VM Monitor
VMXOFF

VM Exits

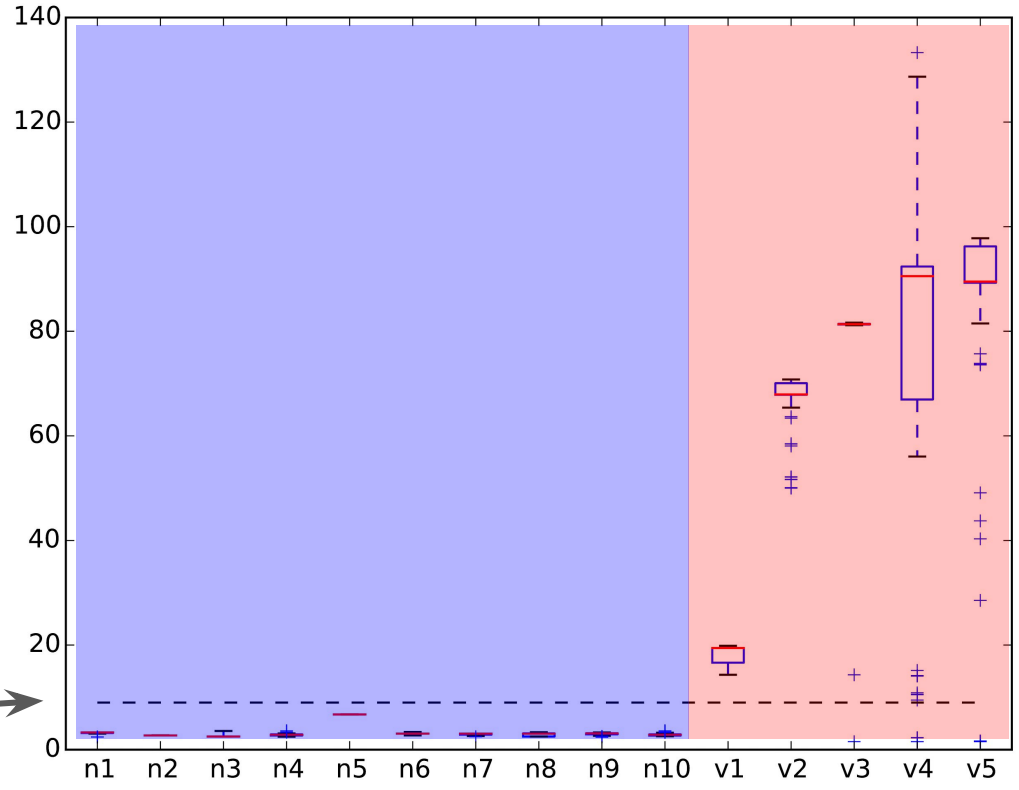
- A VM Exit is triggered whenever a sensitive instruction is executed
  ⇒ Opportunity for emulating information
- Example: Accessing a model-specific register
- We want to force a VM Exit: which instruction to use?
- `cpuid`
  - Not privileged
  - *Always* triggers a VM Exit

Timing Attack

$$\frac{\text{time}(\texttt{cpuid}_{\texttt{native}})}{\text{time}(\text{nop})} \leq \frac{\text{time}(\texttt{cpuid}_{\texttt{vt}})}{\text{time}(\text{nop})}$$

Note: We measure time with `rdtsc`

9



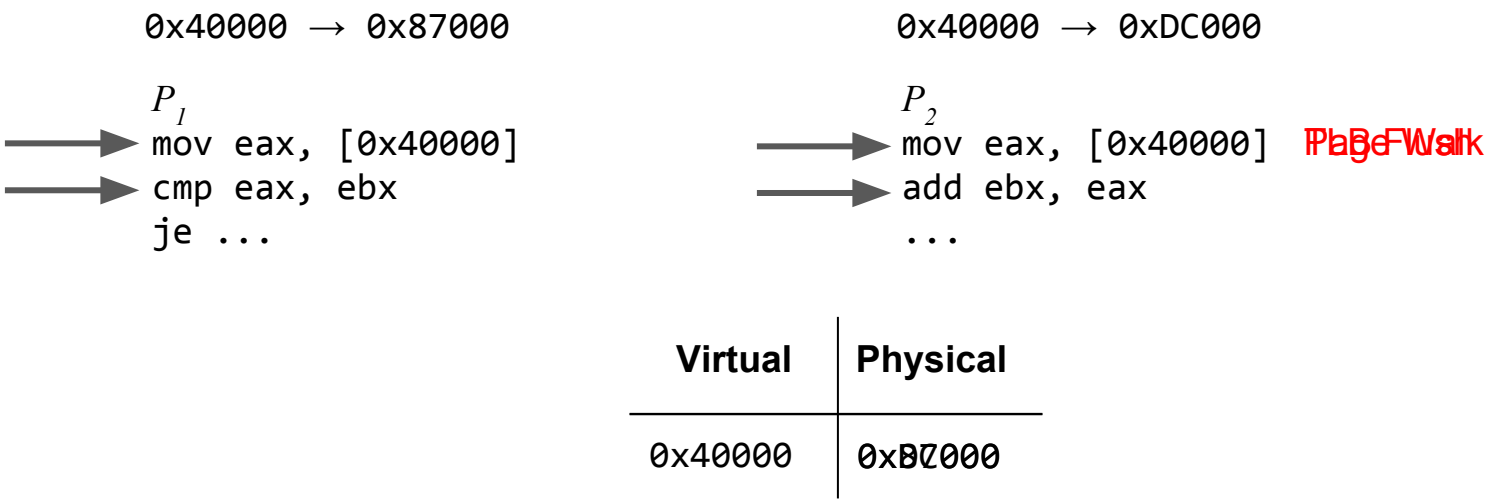
Translation Lookaside Buffer

TLB = A cache for page walks

```
mov eax, [0x400030] ← TLB Miss → Page Walk (0x817030)
mov ebx, [0x410044] ← TLB Miss → Page Walk (0xEF2044)
...
mov eax, [0x400F20] ← TLB Hit (0x817F20)
mov ebx, [0x410044] ← TLB Hit (0xEF2044)
```

Virtual	Physical
0x400000	0x817000
0x410000	0xEF2000

TLB – Context Switches



⇒ Observation: VM Exit = Context Switch

TLB Attack

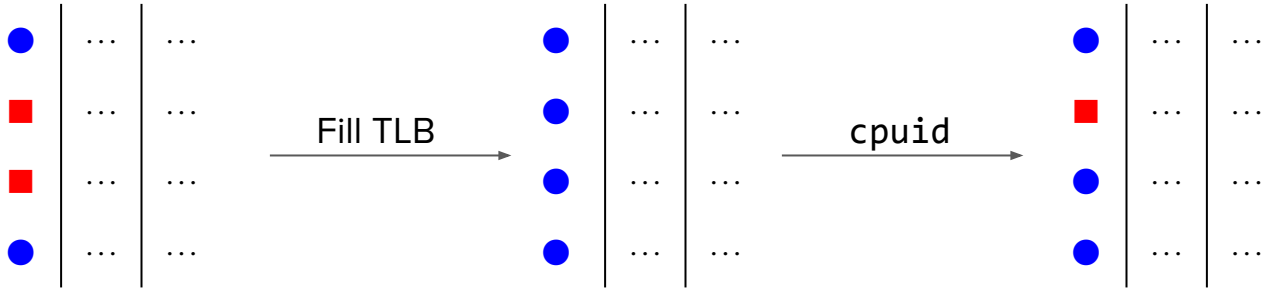
- General Idea:
  - Access a page
  - Execute cpuid
  - Access the page again and check how long it takes

    ⇒ Long access time ⇒ TLB Flush ⇒ Hypervisor present
- Problem: VPID

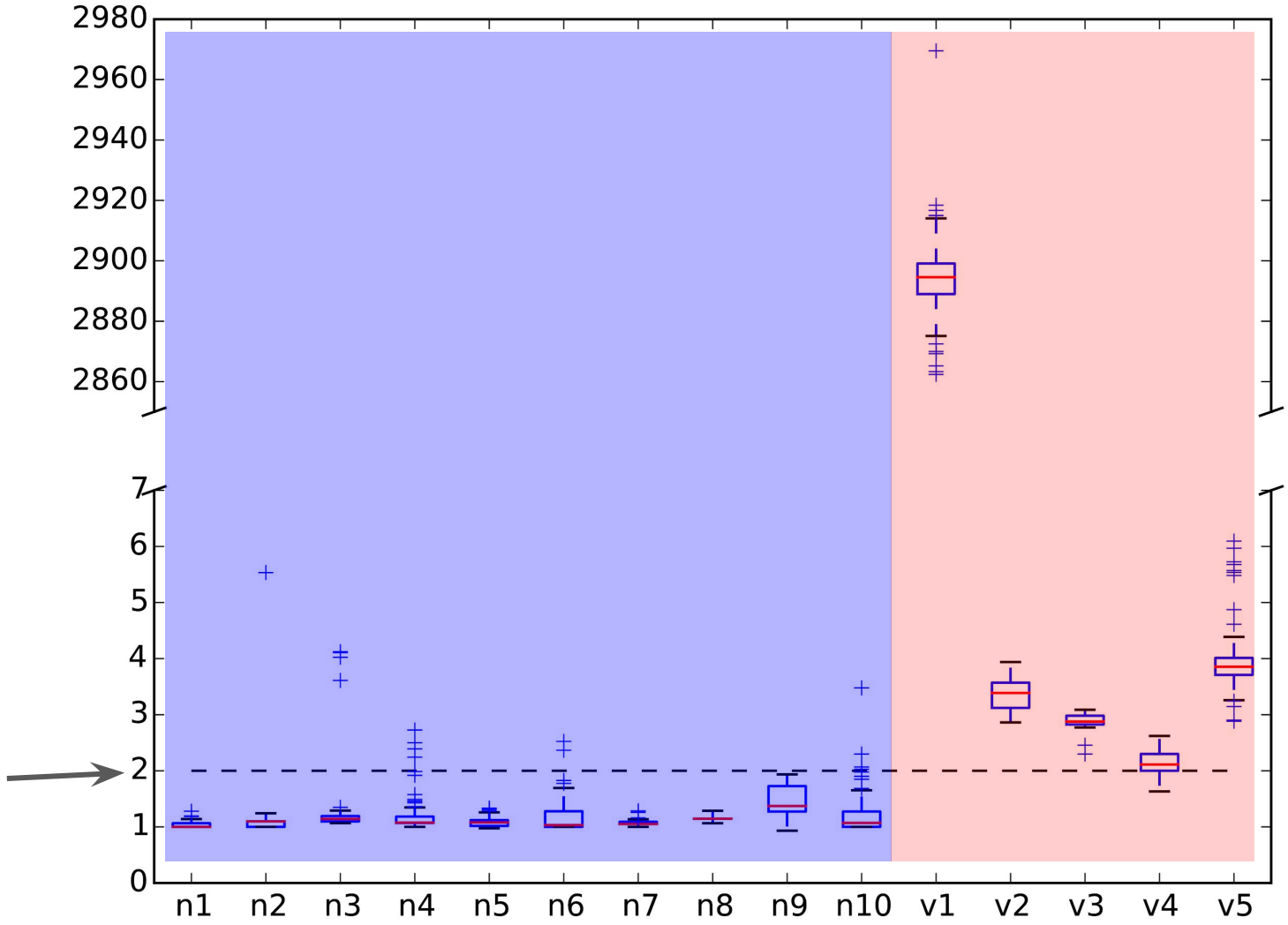
Tag	Virtual	Physical
🔵	0x40000	0x7F000
🟥	0x40000	0x86000
🟥	0x70000	0x6C000
🔵	0x70000	0x94000

⇒ No TLB Flush necessary anymore

TLB Attack (2)



- Rationale: Hypervisor evicts at least 1 TLB-Entry
- Caveat: Highly dependent on the TLB size
  ⇒ Execute multiple times with different TLB sizes
- We compute $r = t_{acc\_after\_cpuid} / t_{acc}$


2980
2960
2940
2920
2900
2880
2860
7
6
5
4
3
2
1
0
n1
n2
n3
n4
n5
n6
n7
n8
n9
n10
v1
v2
v3
v4
v5

Sanity Checks

To account for obvious countermeasures:

1. time(`rdtsc`) $<$ 500 cycles
2. time(`cpuid`) $>$ 20 cycles
3. time(nop) $<$ 500 cycles

To improve the TLB attack:

4. $t_{acc\_after\_cpuid}$ - $t_{acc} \leq$ 150 cycles

Evaluation: Planetlab

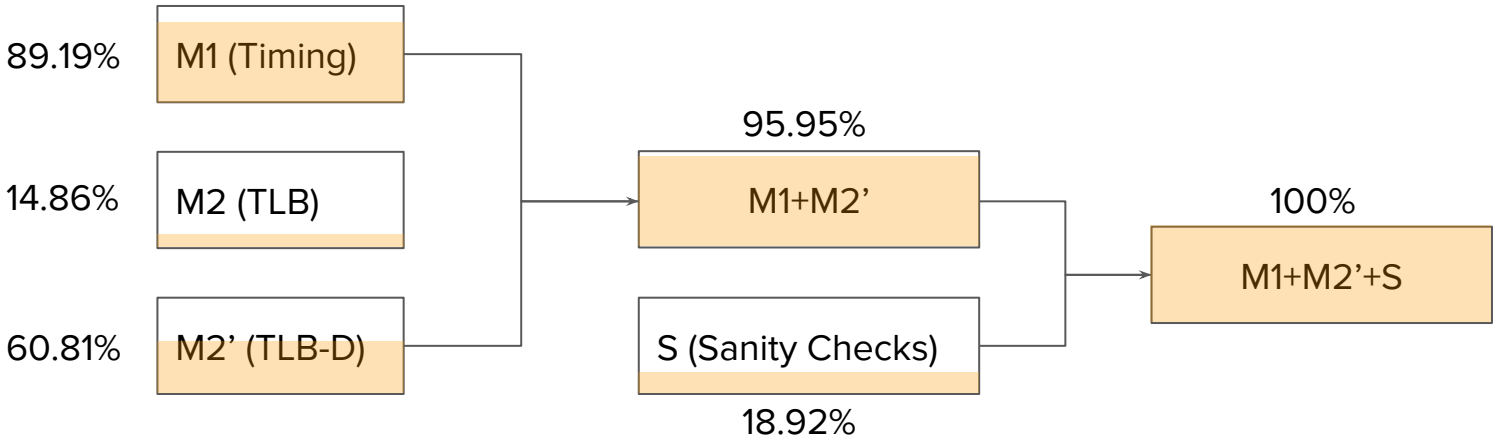
We ran experiments on 239 PlanetLab nodes.
233 native nodes, 6 virtualized nodes

- Timing Attack (M1)
  - True Negative Rate: 99.99%
  - True Positive Rate: 100%
- TLB Attack (M2)
  - True Negative Rate: 99.96%
  - True Positive Rate: 100%

Note: For M2, we had to exclude 3 nodes because of huge pages

Evaluation: Sandboxes

- Sample submitted to malware services (VirusTotal, Threatexpert, ...)
- Results from 30 sandboxes



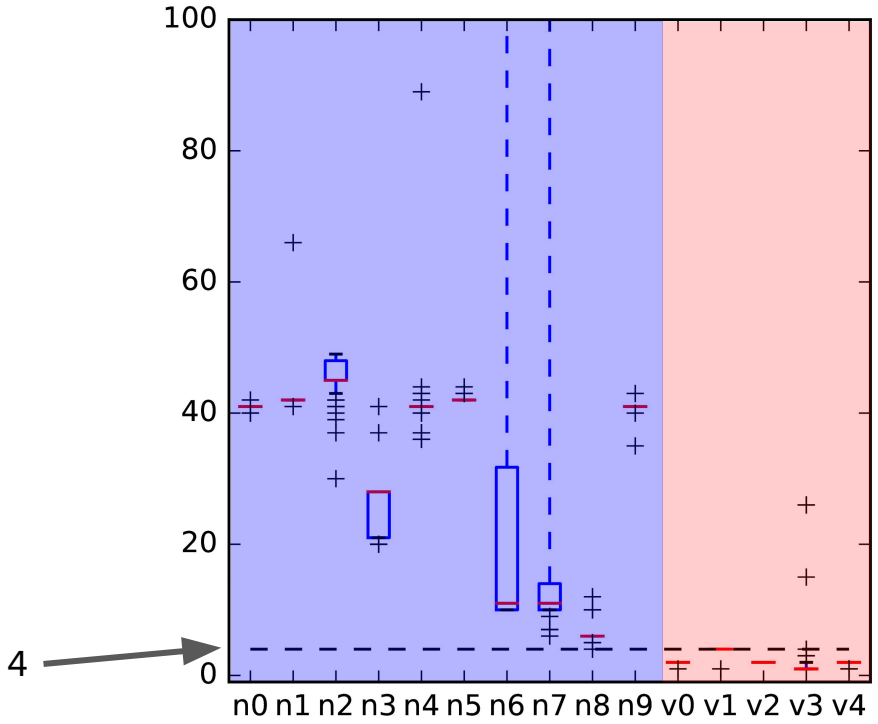
Extra: Stealth Detection

- Extensive use of `cpuid` and `rdtsc`
  ⇒ suspicious
- `rdtsc` → thread-based counting
- `cpuid` → reduce number of calls

```
c = 0

def main():
  spawn(A)
  while c == 0:
      pass
  busy_wait()
  kill(A)
  return c <= thresh
```

```
A:
while True:
    cpuid
    c += 1
```



Conclusion

- In practice, sandboxes can be detected
- Dealt with VPID
- Stealth Detection
- Countermeasures
  - Possible for timing attack (subtract VM Exit overhead)
  - Hard for TLB/Thread Counting
  - AMD-V
  - Huge Pages

Sandbox	M1	M2	M2'	M1+M2	M1+M2'	RDTSC	CPUID	NOP	S	S+M1+M2'
s_0	5/5	2/5	3/5	5/5	5/5	0/5	0/5	0/5	0/5	5/5
s_1	10/10	2/10	5/10	10/10	10/10	0/10	0/10	0/10	0/10	10/10
s_2	4/4	0/4	4/4	4/4	4/4	0/4	0/4	0/4	0/4	4/4
s_3	2/2	0/2	0/2	2/2	2/2	0/2	0/2	0/2	0/2	2/2
s_4	1/1	0/1	1/1	1/1	1/1	0/1	0/1	0/1	0/1	1/1
s_5	1/3	0/3	0/3	1/3	1/3	0/3	0/3	3/3	3/3	3/3
s_6	1/1	0/1	1/1	1/1	1/1	0/1	0/1	0/1	0/1	1/1
s_7	3/3	1/3	1/3	3/3	3/3	0/3	0/3	0/3	0/3	3/3
s_8	1/1	0/1	0/1	1/1	1/1	0/1	0/1	1/1	1/1	1/1
s_9	3/3	0/3	2/3	3/3	3/3	0/3	0/3	0/3	0/3	3/3
s_{10}	6/6	0/6	3/6	6/6	6/6	0/6	0/6	0/6	0/6	6/6
s_{11}	1/1	0/1	1/1	1/1	1/1	0/1	0/1	0/1	0/1	1/1
s_{12}	1/1	1/1	1/1	1/1	1/1	0/1	0/1	0/1	0/1	1/1
$s_{13} \ldots s_{17}$	3/5	0/5	3/5	3/5	4/5	4/5	0/5	4/5	4/5	5/5
s_{18}	6/6	0/6	4/6	6/6	6/6	0/6	0/6	0/6	0/6	6/6
s_{19}	3/3	0/3	1/3	3/3	3/3	0/3	0/3	0/3	0/3	3/3
s_{20}	1/1	1/1	1/1	1/1	1/1	0/1	0/1	1/1	1/1	1/1
s_{21}	3/3	0/3	1/3	3/3	3/3	0/3	0/3	0/3	0/3	3/3
s_{22}	1/1	0/1	1/1	1/1	1/1	0/1	0/1	0/1	0/1	1/1
s_{23}	2/2	2/2	2/2	2/2	2/2	0/2	0/2	0/2	0/2	2/2
s_{24}	0/2	1/2	2/2	1/2	2/2	2/2	0/2	2/2	2/2	2/2
s_{25}, s_{26}	2/2	0/2	1/2	2/2	2/2	0/2	2/2	2/2	2/2	2/2
s_{27}	1/1	1/1	1/1	1/1	1/1	0/1	0/1	0/1	0/1	1/1
s_{28}	1/3	0/3	3/3	1/3	3/3	3/3	0/3	3/3	3/3	3/3
s_{29}	3/3	0/3	3/3	3/3	3/3	0/3	0/3	0/3	0/3	3/3
s_{30}	1/1	0/1	0/1	1/1	1/1	0/1	0/1	0/1	0/1	1/1
	66/74	11/74	45/74	67/74	71/74	9/74	0/74	14/74	14/74	74/74
	89.19%	14.86%	60.81%	90.54%	95.95%	12.16%	0.00%	18.92%	18.92%	100.00%