HyperShell:

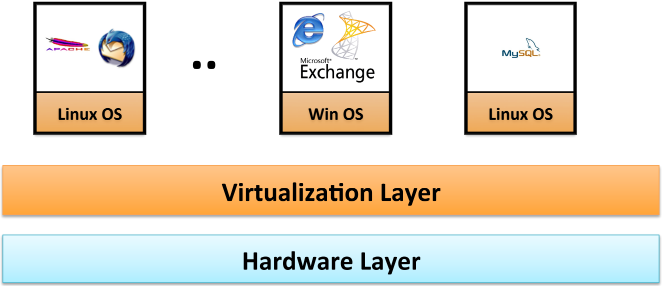
A Practical Hypervisor Layer Guest OS Shell for Automated In-VM Management

Yangchun Fu, Junyuan Zeng, Zhiqiang Lin

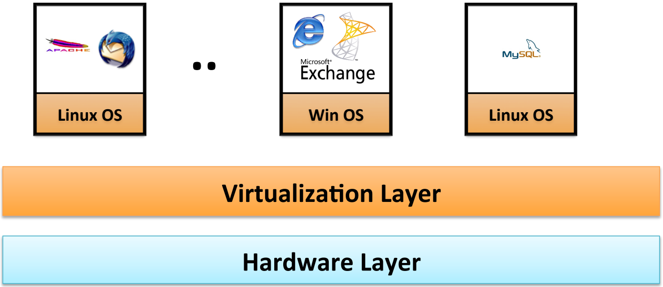
Department of Computer Science
The University of Texas at Dallas

June 19th, 2014

How to manage the guest OS?



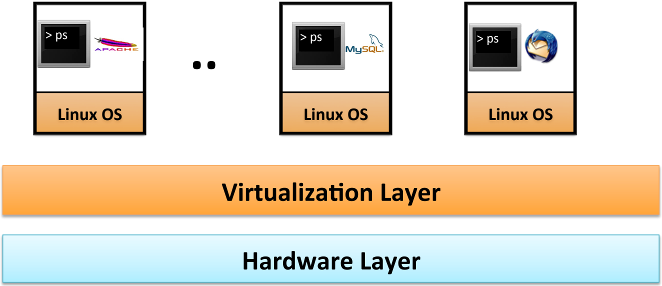
How to manage the guest OS?



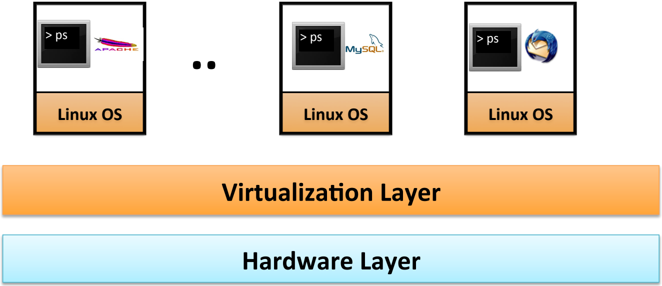
Requiring Large Scale, Automated Management

- Private, Public Cloud, Data Centers
- Usually hosts tens of thousands of virtual machines

Approach-I


> ps
APACHE
Linux OS
..
> ps
MySQL
Linux OS
> ps
Linux OS
Virtualization Layer
Hardware Layer

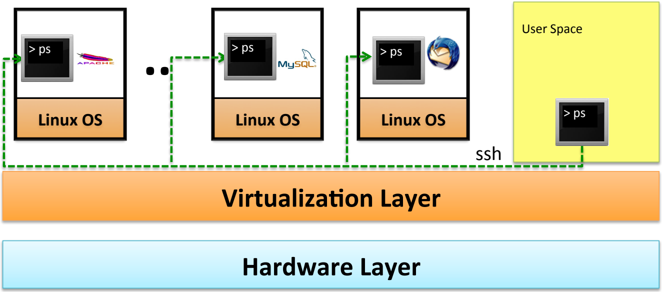
Approach-I



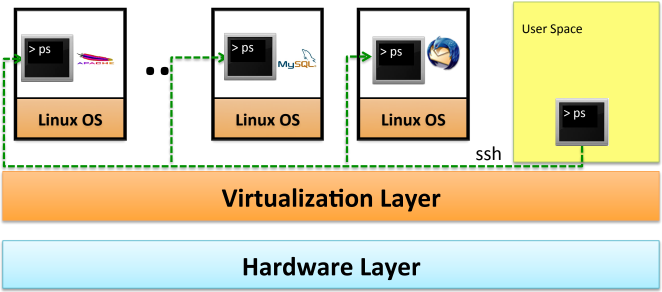
Disadvantages

- Scattered, distributed
- Install, update, and execute in each VM

Approach-II


> ps
> ps
> ps
> ps
Linux OS
Linux OS
Linux OS
User Space
ssh
Virtualization Layer
Hardware Layer

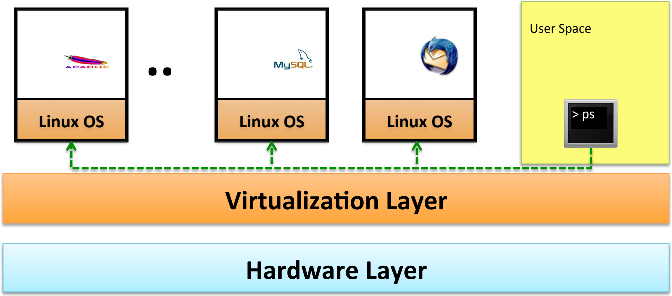
Approach-II



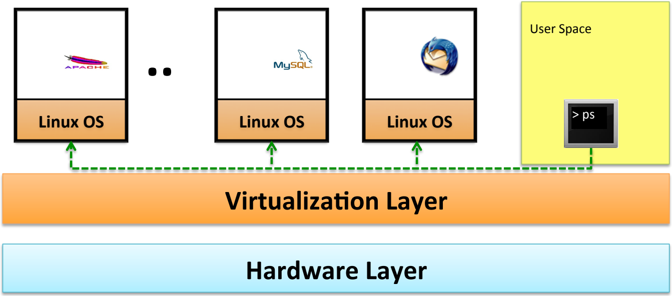
Disadvantages

- Requiring the (admin) login password.
- Requiring install the management utilities in each VM.

Our Approach



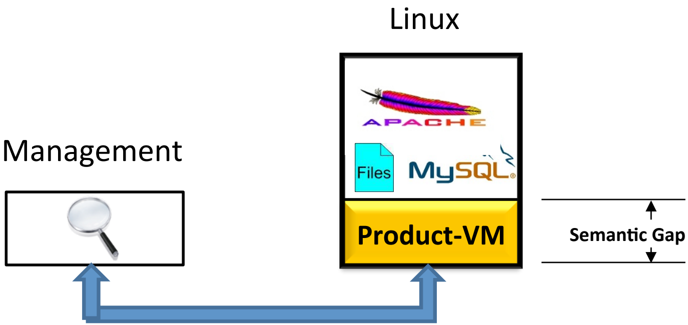
Our Approach



Advantages

- Only install the management utilities at hypervisor layer.
- Automated, uniformed, and centralized management.

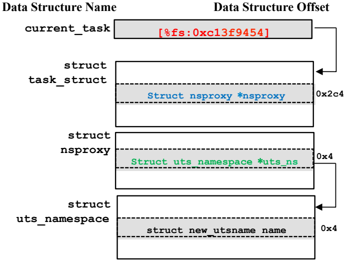
The Semantic Gap [HotOS'01]



Observation: Reuse Existing Code?

```
 1. execve("/bin/hostname", ["hostname"], ...) = 0
 2. brk(0)                                  = 0x8113000
 3. access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT
 4. mmap2(NULL, 8192, ..., -1, 0) = 0xb7795000
...
36. uname({sys="Linux", node="debian", ...}) = 0
...
40. write(1, "debian\n", 7)                 = 7
41. exit_group(0)
```

System call trace of "hostname"



Data Structure Name

Data Structure Offset

current_task [%fs:0xc13f9454]

struct task_struct

Struct nsproxy *nsproxy 0x2c4

struct nsproxy

Struct uts_namespace *uts_ns 0x4

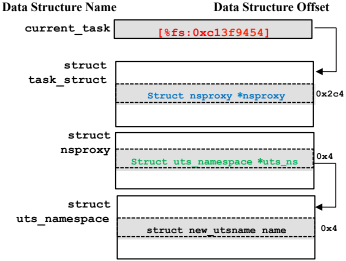
struct uts_namespace

struct new_utsname name 0x4

Observation: Reuse Existing Code?

```
 1. execve("/bin/hostname", ["hostname"], ...) = 0
 2. brk(0)                                   = 0x8113000
 3. access("/etc/ld.so.nohwcap", F_OK)       = -1 ENOENT
 4. mmap2(NULL, 8192, ..., -1, 0) = 0xb7795000
...
36. uname({sys="Linux", node="debian", ...}) = 0
...
40. write(1, "debian\n", 7)                  = 7
41. exit_group(0)
```

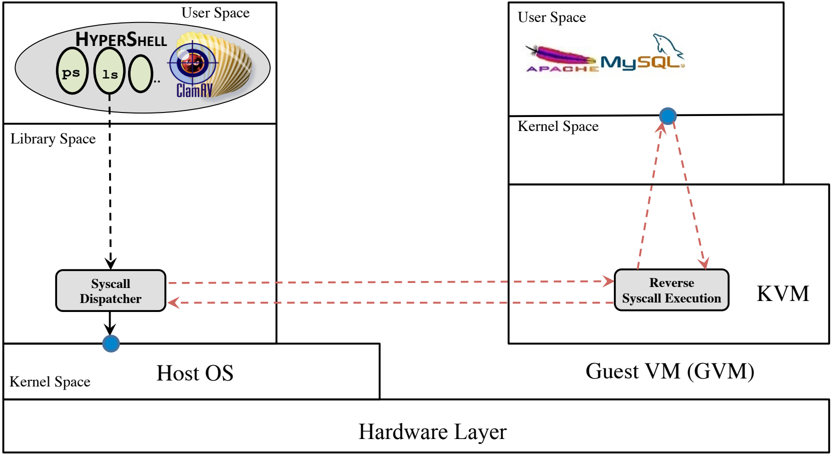
System call trace of "hostname"



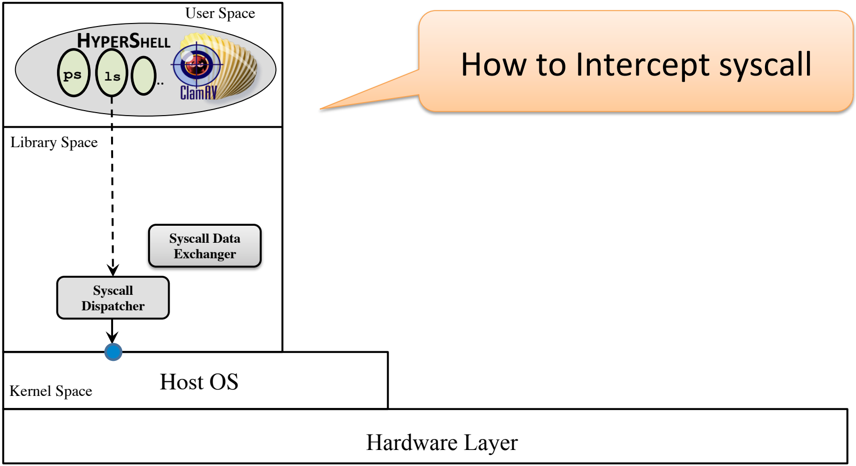
Key Insight

- System call is the only interface to request OS service.
- Redirecting the system call execution from one VM to the other.

Introducing Our HyperShell

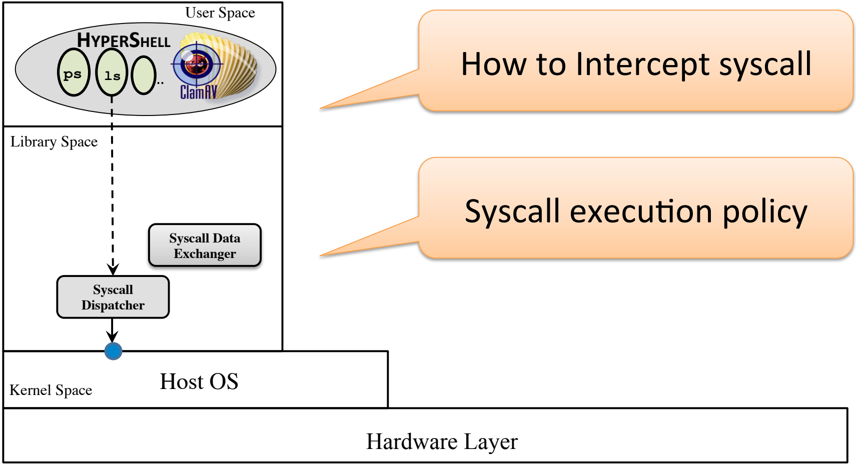


Host OS side design



How to Intercept syscall

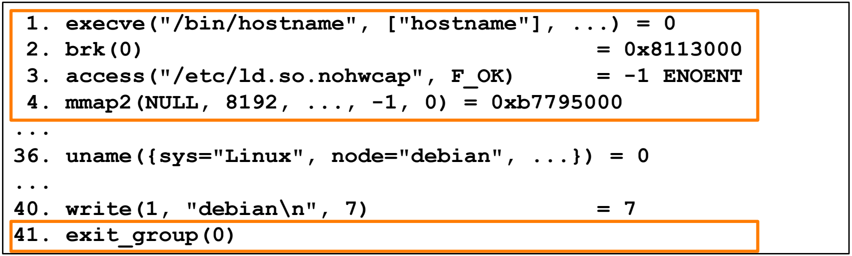
Host OS side design



How to Intercept syscall

Syscall execution policy

Syscall Execution Policy



```
 1. execve("/bin/hostname", ["hostname"], ...) = 0
 2. brk(0)                                      = 0x8113000
 3. access("/etc/ld.so.nohwcap", F_OK)          = -1 ENOENT
 4. mmap2(NULL, 8192, ..., -1, 0) = 0xb7795000
...
36. uname({sys="Linux", node="debian", ...}) = 0
...
40. write(1, "debian\n", 7)                     = 7
41. exit_group(0)
```

System call trace of command “`hostname`”

🟧 In Host

🟩 In Guest

Syscall Execution Policy

```
 1. execve("/bin/hostname", ["hostname"], ...) = 0
 2. brk(0)                                   = 0x8113000
 3. access("/etc/ld.so.nohwcap", F_OK)       = -1 ENOENT
 4. mmap2(NULL, 8192, ..., -1, 0) = 0xb7795000
...
36. uname({sys="Linux", node="debian", ...}) = 0
...
40. write(1, "debian\n", 7)                  = 7
41. exit_group(0)
```

System call trace of command “hostname”

🟧 In Host

🟩 In Guest

Syscall Execution Policy

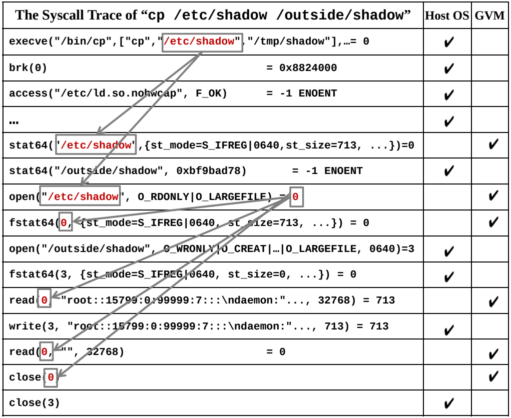
```
 1. execve("/bin/hostname", ["hostname"], ...) = 0
 2. brk(0)                                      = 0x8113000
 3. access("/etc/ld.so.nohwcap", F_OK)         = -1 ENOENT
 4. mmap2(NULL, 8192, ..., -1, 0) = 0xb7795000
...
36. uname({sys="Linux", node="debian", ...}) = 0
...
40. write(1, "debian\n", 7)                    = 7
41. exit_group(0)
```

System call trace of command "`hostname`"

🟧 In Host

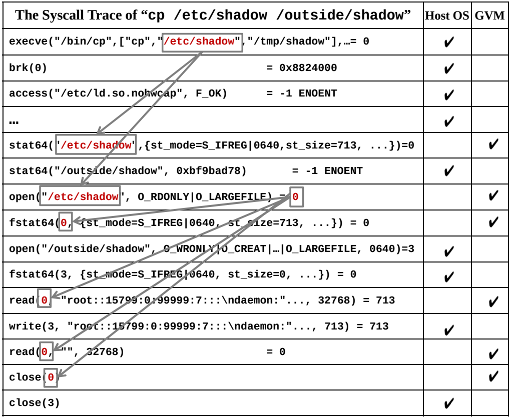
🟩 In Guest

Syscall Executuion Policy



The Syscall Trace of "cp /etc/shadow /outside/shadow"	Host OS	GVM
execve("/bin/cp",["cp","/etc/shadow","/tmp/shadow"],…= 0	✓	
brk(0) = 0x8824000	✓	
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT	✓	
…	✓	
stat64('/etc/shadow',{st_mode=S_IFREG|0640,st_size=713, ...})=0		✓
stat64("/outside/shadow", 0xbf9bad78) = -1 ENOENT	✓	
open("/etc/shadow", O_RDONLY|O_LARGEFILE) = 0		✓
fstat64(0, {st_mode=S_IFREG|0640, st_size=713, ...}) = 0		✓
open("/outside/shadow", O_WRONLY|O_CREAT|…|O_LARGEFILE, 0640)=3	✓	
fstat64(3, {st_mode=S_IFREG|0640, st_size=0, ...}) = 0	✓	
read(0, "root::15799:0:99999:7:::\ndaemon:"..., 32768) = 713		✓
write(3, "root::15799:0:99999:7:::\ndaemon:"..., 713) = 713	✓	
read(0, "", 32768) = 0		✓
close(0)		✓
close(3)	✓	

Syscall Executuion Policy



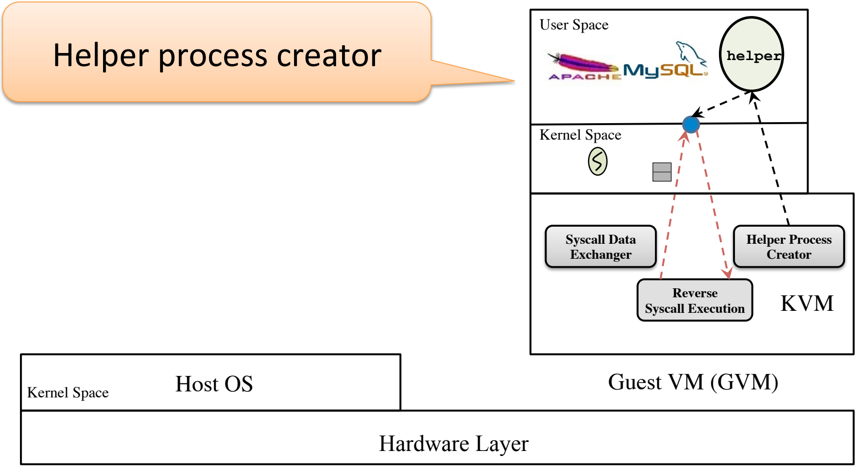
The Syscall Trace of “`cp /etc/shadow /outside/shadow`”	Host OS	GVM
`execve("/bin/cp",["cp","/etc/shadow","/tmp/shadow"],…= 0`	✓	
`brk(0)                                    = 0x8824000`	✓	
`access("/etc/ld.so.nohwcap", F_OK)       = -1 ENOENT`	✓	
…	✓	
`stat64("/etc/shadow",{st_mode=S_IFREG|0640,st_size=713, ...})=0`		✓
`stat64("/outside/shadow", 0xbf9bad78)        = -1 ENOENT`	✓	
`open("/etc/shadow", O_RDONLY|O_LARGEFILE) = 0`		✓
`fstat64(0, {st_mode=S_IFREG|0640, st_size=713, ...}) = 0`		✓
`open("/outside/shadow", O_WRONLY|O_CREAT|…|O_LARGEFILE, 0640)=3`	✓	
`fstat64(3, {st_mode=S_IFREG|0640, st_size=0, ...}) = 0`	✓	
`read(0, "root::15799:0:99999:7:::\ndaemon:"..., 32768) = 713`		✓
`write(3, "root::15799:0:99999:7:::\ndaemon:"..., 713) = 713`	✓	
`read(0, "", 32768)                        = 0`		✓
`close(0)`		✓
`close(3)`	✓	

Solution

- File descriptor is just an index and has a limited maximum value. We can add an extra value to differentiate it.

Guest VM Side Design

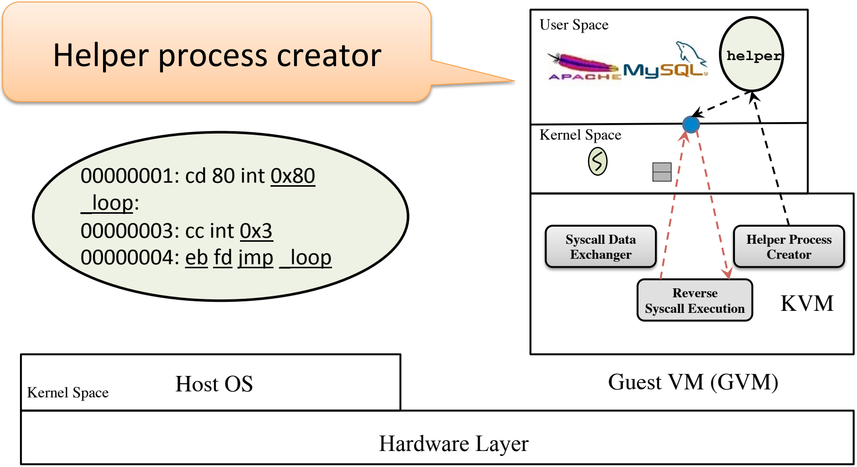
Helper process creator



Guest VM Side Design

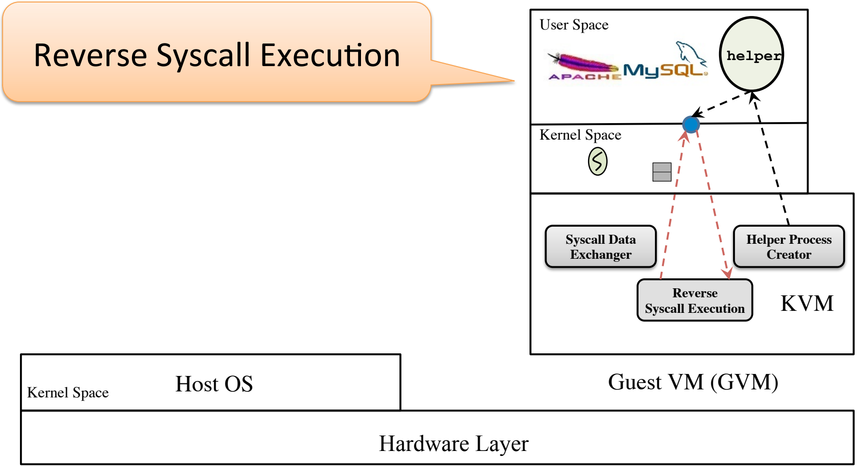
Helper process creator

```
00000001: cd 80 int 0x80
_loop:
00000003: cc int 0x3
00000004: eb fd jmp _loop
```

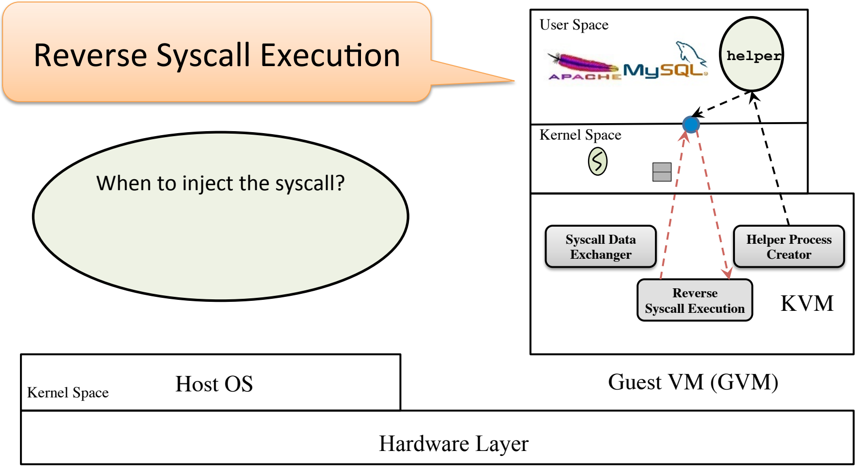


Guest VM Side Design

Reverse Syscall Execution



Guest VM Side Design



Reverse Syscall Execution

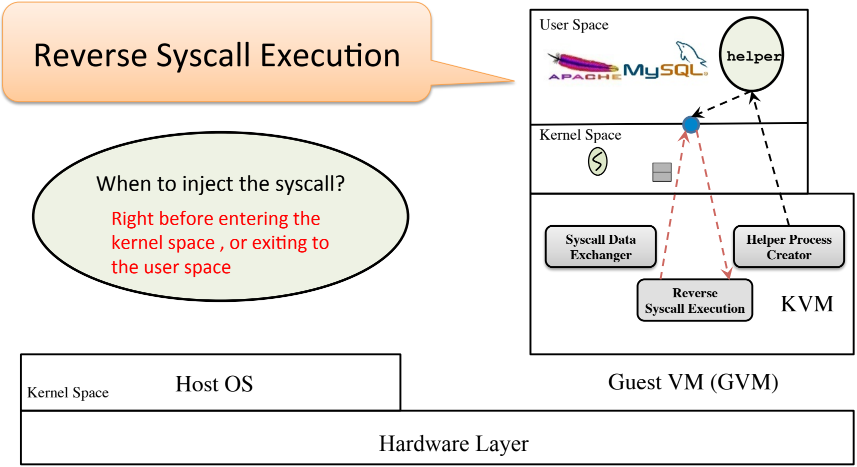
When to inject the syscall?

Guest VM Side Design

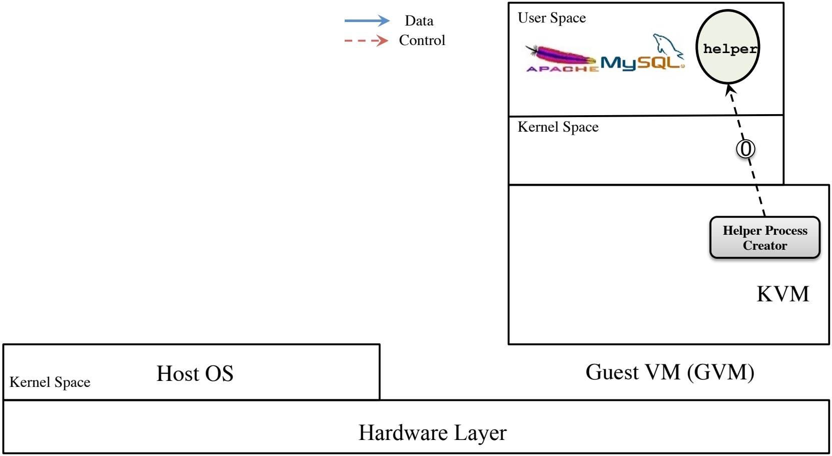
Reverse Syscall Execution

When to inject the syscall?

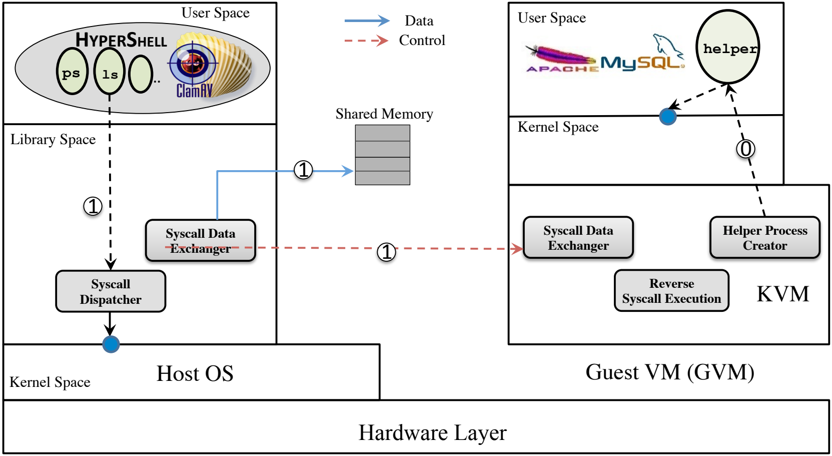
Right before entering the kernel space , or exiting to the user space



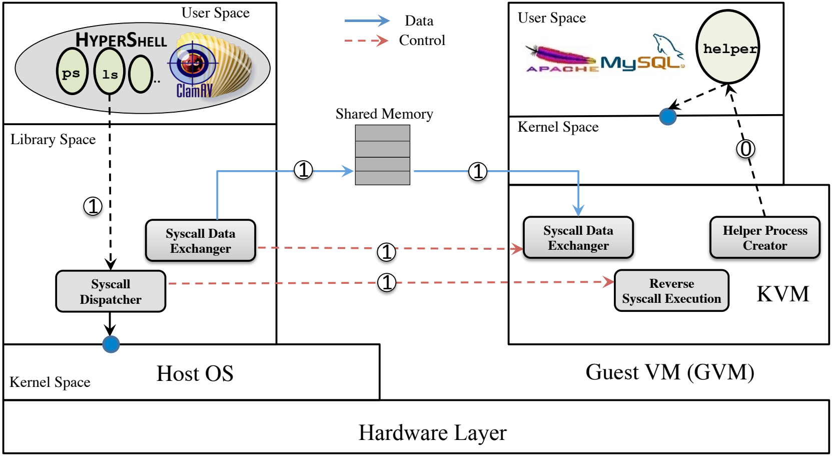
HyperShell Overview



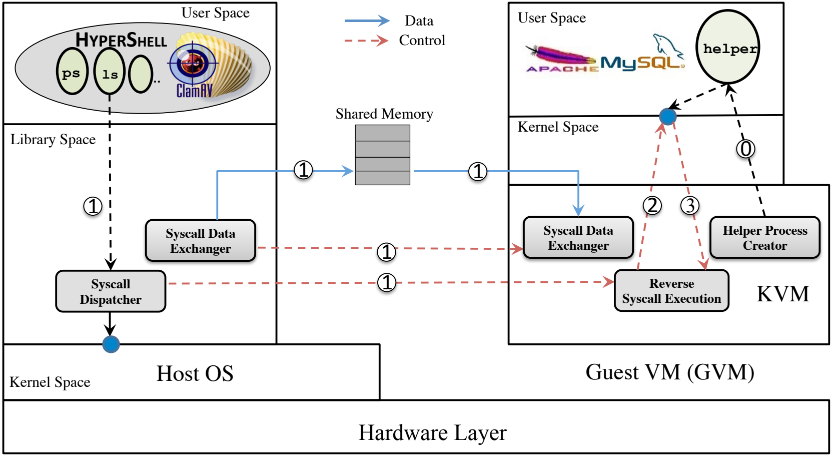
HYPERSHELL Overview



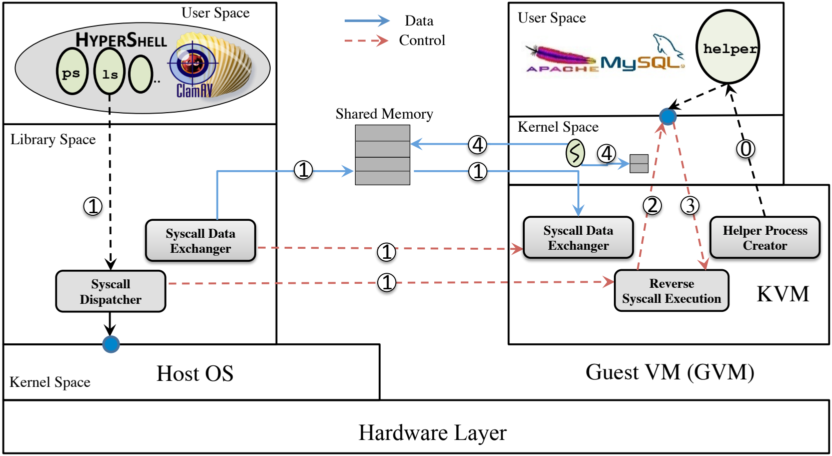
HYPERSHELL Overview



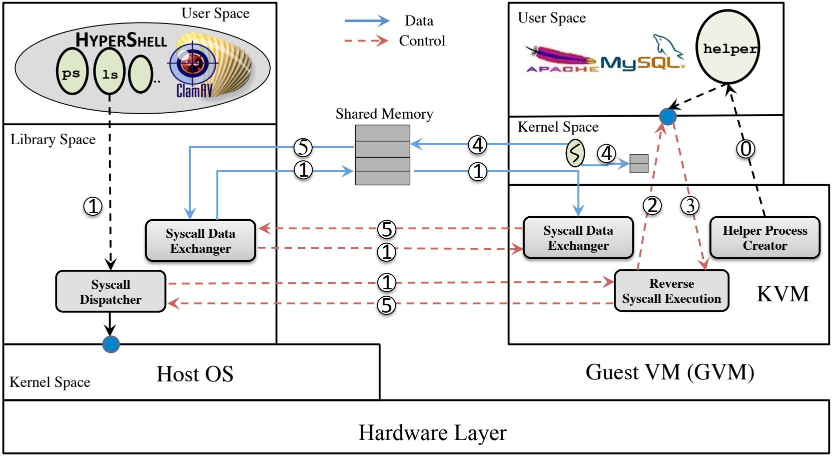
HyperShell Overview



HyperShell Overview



HYPERSHELL Overview



Performance Impact to the Native Utilities

Process	S	$B(ms)$	$D(ms)$	$T(X)$	date	✗	0.11	0.12	1.09	mkdir	✓	0.10	0.19	1.90
ps	✗	1.33	5.42	4.08	w	✗	0.95	6.62	6.97	mkfifo	✓	0.10	0.19	1.90
pidstat	✗	1.95	7.56	3.88	hostname	✓	0.04	0.06	1.50	mknod	✓	0.10	0.19	1.90
nice	✓	0.07	0.11	1.57	groups	✓	0.21	0.62	2.95	mv	✓	0.15	0.31	2.07
getpid	✓	0.01	0.02	2.00	hostid	✓	0.16	0.56	3.50	rm	✓	0.08	0.15	1.88
mpstat	✗	0.29	0.66	2.28	locale	✓	0.09	0.17	1.89	od	✓	0.12	0.35	2.92
pstree	✗	0.69	6.03	8.74	getconf	✓	0.09	0.34	3.78	cat	✓	0.07	0.18	2.57
chrt	✓	0.11	0.16	1.45	**System Utils**	S	$B(ms)$	$D(ms)$	$T(X)$	link	✓	0.07	0.13	1.86
renice	✓	0.11	0.18	1.64	uptime	✗	0.07	0.47	6.71	comm	✓	0.08	0.22	2.75
top	✗	504.92	510.85	1.01	sysctl	✓	8.5	42.72	5.03	shred	✗	0.72	0.92	1.28
nproc	✓	0.07	0.26	3.71	arch	✓	0.07	0.11	1.57	truncate	✓	0.07	0.26	3.71
sleep	✓	1.27	1.28	1.01	dmesg	✓	0.38	0.51	1.34	head	✓	0.07	0.15	2.14
pgrep	✓	0.89	4.72	5.30	lscpu	✓	0.26	1.21	4.65	vdir	✓	0.63	3.95	6.27
pkill	✓	0.87	4.33	4.98	mcookie	✗	0.29	0.49	1.69	nl	✓	0.08	0.17	2.13
snice	✓	0.17	0.65	3.82	**Disk/Devices**	S	$B(ms)$	$D(ms)$	$T(X)$	tail	✓	0.08	0.20	2.50
echo	✓	0.07	0.09	1.29	blkid	✓	0.14	0.61	4.36	namei	✓	0.07	0.13	1.86
pwdx	✓	0.05	0.07	1.40	badblocks	✓	0.35	0.44	1.26	whereis	✓	2.05	4.86	2.37
pmap	✓	0.16	0.36	2.25	lspci	✓	31.40	36.52	1.16	stat	✓	0.27	0.78	2.89
kill	✓	0.01	0.04	4.00	iostat	✓	0.45	1.04	2.31	readlink	✓	0.07	0.12	1.71
killall	✓	0.62	3.03	4.89	du	✓	0.11	0.53	4.82	unlink	✓	0.07	0.13	1.86
Memory	S	$B(ms)$	$D(ms)$	$T(X)$	df	✓	0.16	0.35	2.19	cut	✓	0.08	0.17	2.13
free	✗	0.04	0.08	2.00	**Filesystem**	S	$B(ms)$	$D(ms)$	$T(X)$	dir	✓	0.07	0.20	2.86
vmstat	✗	0.19	0.33	1.74	sync	✓	8.07	6.53	0.81	mktemp	✓	0.09	0.18	2.00
slabtop	✗	0.22	0.36	1.64	getcap	✓	0.04	0.08	2.00	rmdir	✓	0.07	0.13	1.86
Modules	S	$B(ms)$	$D(ms)$	$T(X)$	lsof	✓	3.31	6.12	1.85	ptx	✓	0.12	0.45	3.75
rmmod	✓	0.51	3.14	6.16	pwd	✓	0.07	0.11	1.57	chcon	✓	0.06	0.12	2.00
modinfo	✓	0.48	1.54	3.21	**Files**	S	$B(ms)$	$D(ms)$	$T(X)$	**Network**	S	$B(ms)$	$D(ms)$	$T(X)$
lsmod	✓	0.10	0.17	1.70	chgrp	✓	0.19	0.47	2.47	ifconfig	✗	0.32	1.15	3.59
Environment	S	$B(ms)$	$D(ms)$	$T(X)$	chmod	✓	0.07	0.14	2.00	ip	✓	0.10	0.20	2.00
who	✓	0.14	0.72	5.14	chown	✓	0.19	0.47	2.47	route	✓	138.65	150.32	1.08
env	✓	0.07	0.11	1.57	cp	✓	0.11	0.27	2.45	ipmaddr	✓	0.13	0.34	2.62
printenv	✓	0.07	0.1	1.43	uniq	✓	0.09	0.35	3.89	iptunnel	✓	0.09	0.29	3.22
whoami	✓	0.19	0.45	2.37	file	✓	0.87	1.72	1.98	nameif	✓	0.10	0.21	2.10
stty	✓	0.11	0.46	4.18	find	✓	0.20	0.58	2.90	netstat	✗	0.25	0.37	1.48
users	✓	0.09	0.53	5.89	grep	✓	0.35	2.14	6.11	arp	✓	0.14	0.24	1.71
uname	✓	0.09	0.11	1.22	ln	✓	0.08	0.14	1.75	ping	✗	15.02	18.2	1.21
id	✓	0.26	0.85	3.27	ls	✓	0.14	0.27	1.93	Avg.	-	7.27	8.45	2.73

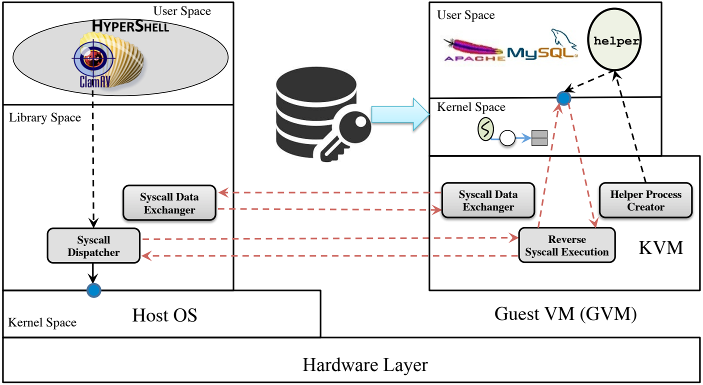
Micro-benchmark Test Result of GVM.

Tested Item	Native-KVM	GVM-RI-Phase	Slowdown (%)	GVM-RE-Phase	Slowdown (%)
stat (μs)	0.39	2.28	82.89	0.41	**4.88**
fork proc (μs)	47.20	147.26	67.95	47.54	**0.72**
exec proc (μs)	158.20	480.00	67.04	161.30	**1.92**
sh proc (μs)	384.90	1088.10	64.63	386.30	**0.36**
ctxsw (μs)	0.59	1.23	52.03	0.73	**19.18**
10K File Create (μs)	17.80	40.67	56.23	17.96	**0.89**
10K File Delete (μs)	4.64	7.16	35.20	4.65	**0.22**
Bcopy (MB/s)	5689.17	5647.71	0.73	5605.40	**1.47**
Rand mem (ns)	72.20	72.65	0.62	73.24	**1.42**
Mem read (MB/s)	10150.00	10000.00	1.48	10000.00	**1.48**
Mem write (MB/s)	8567.70	8543.00	0.29	8540.40	**0.32**

Macro-benchmark Test Result of GVM.

Benchmark Program	Native-KVM	GVM-RI-Phase	Slowdown (%)	GVM-RE-Phase	Slowdown (%)
bzip (s)	16.83	18.35	8.28	17.04	**1.23**
kbuild (s)	1799.00	2270.25	20.76	1889.97	**4.81**
memcached (s)	1.57	3.11	49.52	1.64	**4.27**
Apache (#request/s)	1104.60	904.12	18.15	1065.28	**3.56**

Full disk encryption (FDE) protected virus scanning



Full disk encryption (FDE) protected virus scanning

1. Encypted by dm-crypt
2. 101,415 files
3. 1336.09 megabytes in size

Full disk encryption (FDE) protected virus scanning

1. Encypted by dm-crypt
2. 101,415 files
3. 1336.09 megabytes in size

Clamav successfully detect two viruses!!

Comparison with the most related work

Systems	Execution Context Reuse	wo/ Dual-VM Architecture	wo/ Identical Kernel	wo/ Trust to Guest Kernel	High Code Coverage	Fully Automated	Memory Introspection	Disk Introspection	Guest Management	Process Monitoring
Virtuoso	✗	✓	✗	✓	✗	✗	✓	✗	✗	✗
VMST	✓	✗	✗	✓	✓	✓	✓	✗	✗	✗
Exterior	✓	✗	✗	✓	✓	✓	✓	✗	✓	✗
Pi	✓	✓	✓	✗	✓	✓	✗	✗	✓	✗
PoG	✓	✗	✓	✓	✓	✓	✗	✗	✗	✓
Gears	✓	✓	✓	✗	✓	✗	✓	✓	✓	✓
HyperShell	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓

Limitation and future work

- HYPERSHELL will circumvent all of the existing user login and system audit for each managed VM.

Limitation and future work

- HYPERSHELL will circumvent all of the existing user login and system audit for each managed VM.
- Add a new log record at the hypervisor layer.

Limitation and future work

- HYPERSHELL will circumvent all of the existing user login and system audit for each managed VM.
- Add a new log record at the hypervisor layer.

- HYPERSHELL requires the trust of the guest OS kernel as well as the init process.

Limitation and future work

- HYPERSHELL will circumvent all of the existing user login and system audit for each managed VM.
- Add a new log record at the hypervisor layer.

- HYPERSHELL requires the trust of the guest OS kernel as well as the init process.
- It cannot be used for security critical applications unless special care is taken for these code.

Limitation and future work

- HYPERSHELL will circumvent all of the existing user login and system audit for each managed VM.
- Add a new log record at the hypervisor layer.

- HYPERSHELL requires the trust of the guest OS kernel as well as the init process.
- It cannot be used for security critical applications unless special care is taken for these code.

- HYPERSHELL requires both OSes running in the host OS and VM to have compatible syscall interface.

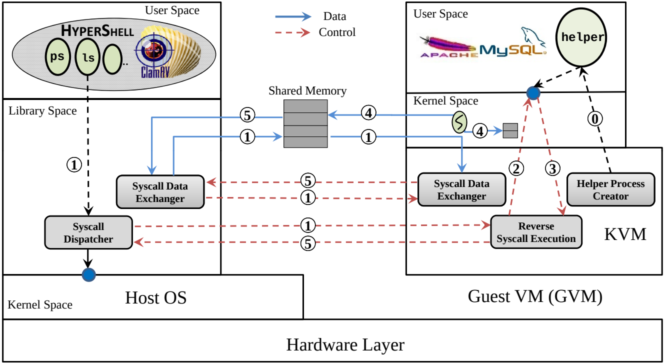
Limitation and future work

- HYPERSHELL will circumvent all of the existing user login and system audit for each managed VM.
- Add a new log record at the hypervisor layer.

- HYPERSHELL requires the trust of the guest OS kernel as well as the init process.
- It cannot be used for security critical applications unless special care is taken for these code.

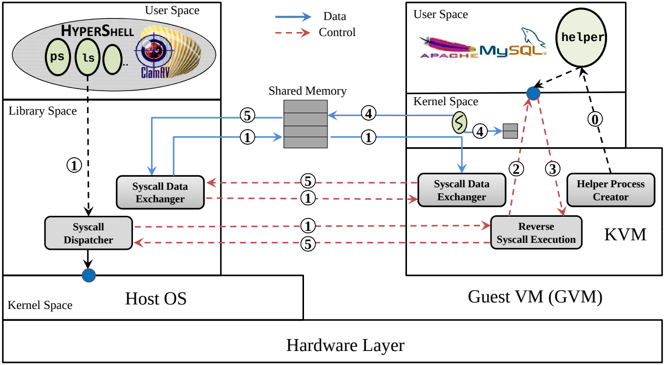
- HYPERSHELL requires both OSes running in the host OS and VM to have compatible syscall interface.
- Perform additional syscall translations can make it work for even larger set of OSes.

Summary



- HyperShell is practical, and can be used for automated, uniformed, and centralized guest OS management
- It automatically bridges the semantic-gap through system call execution redirection.

Thank you!



To contact us

firstname.lastname@utdallas.edu for questions and source code