SysCap: Profiling and Crosschecking Syscall and Capability Configurations for Docker Images

Yunlong Xing, Jiahao Cao, Xinda Wang, Sadegh Torabi,

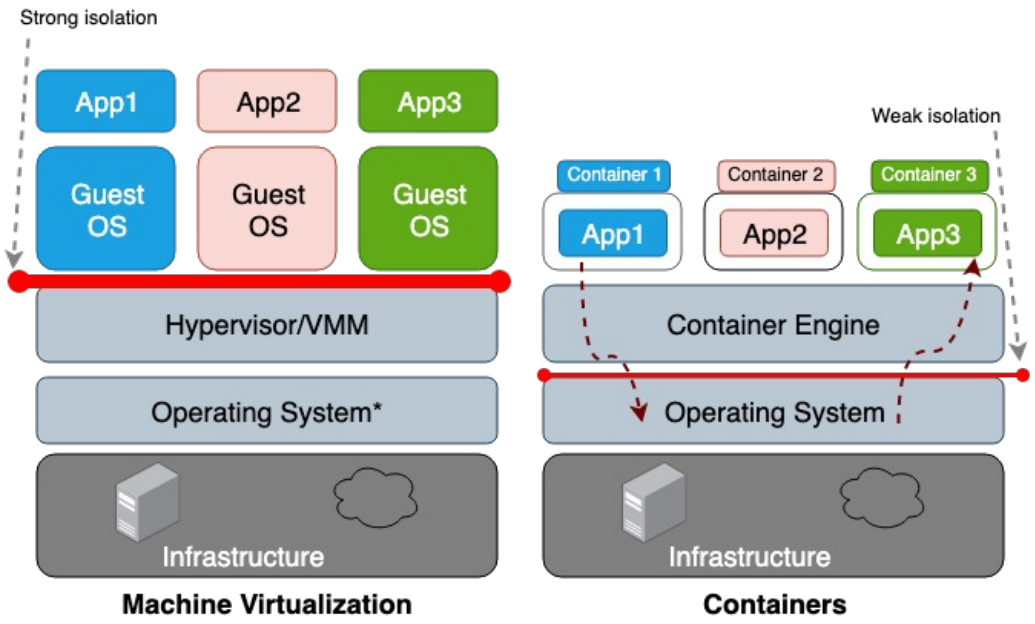
Kun Sun, Fei Yan, Qi Li

Outline

- Background
- Overview of Container Security
- Challenges
- SysCap Design
- Evaluation
- Conclusion

Virtual Machine vs Container

	VM	Container
Virtualization	Hardware-assistant	OS
Sharing	Hardware	Hardware && OS kernel



Container Security

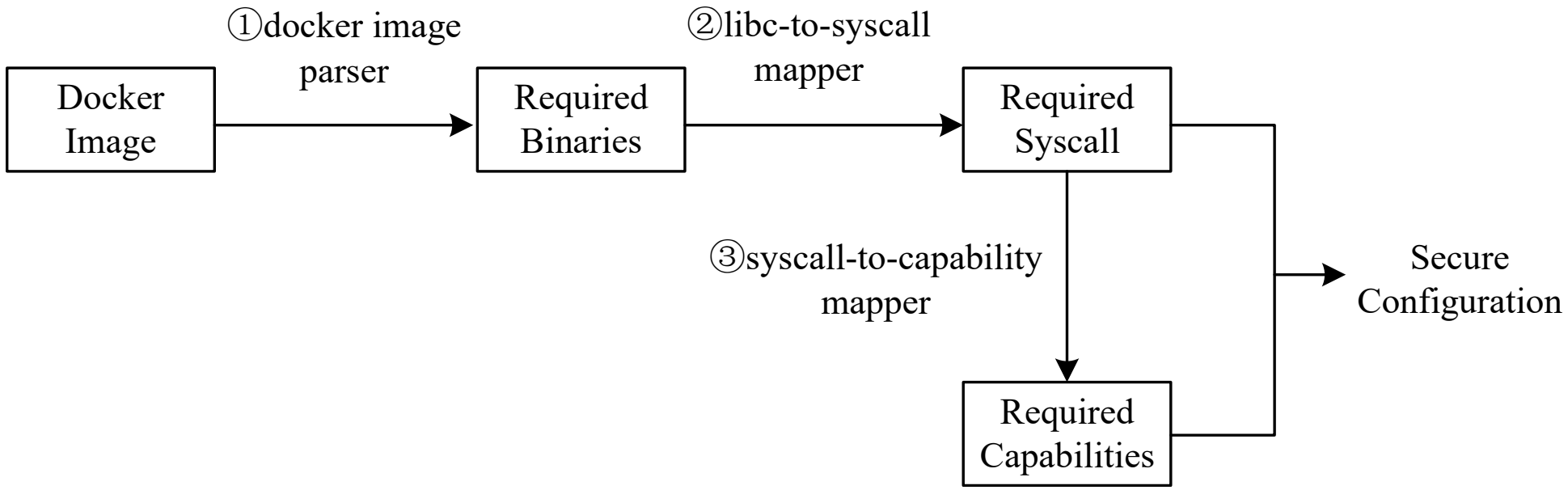
- Both seccomp filters and capabilities play an important part in reducing the risks of container escape
- Seccomp can be used for blocking unnecessary system calls
  - 44 syscalls are disabled by default
- Capability works as a gatekeeper before certain system calls are triggered
  - 14 (out of 41) capabilities are enabled by default

Challenges

- Existing solutions mainly focus on system call reduction and utilize dynamic tracking to obtain the required system calls
  - Simple, but requiring heavy human efforts and an incomplete coverage
  - Ignoring capability may block execution of some system calls
- Static analysis can get a complete result, however
  - Layered and disordered image → target program
  - Target program → related syscall and capability

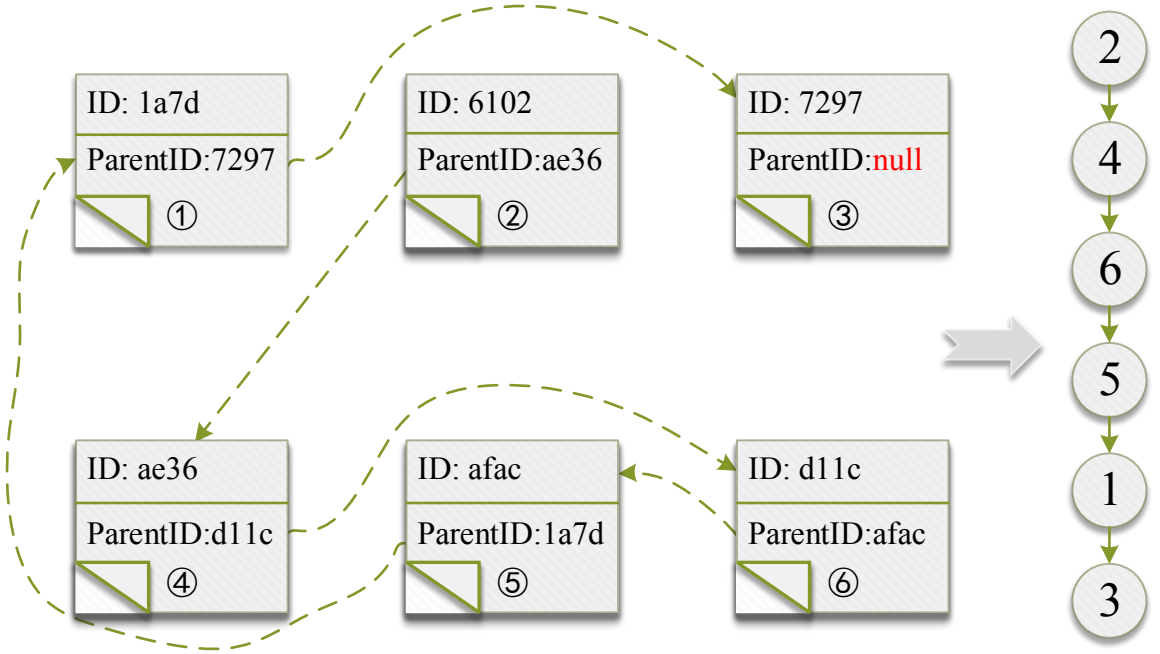
SysCap Architecture

- **Goal:** Obtaining the required syscall and capabilities for a Docker image via static analysis



Docker Image Parser

- Layer Relation Construction
  - ➢ Layers in a Docker image are stored out-of-order
  - ➢ In each layer, a configuration file records the layerID and its parent layerID
  - ➢ Analyzing all configuration files, we can get the layer chain



Docker Image Parser

- Layer Function Extraction
  - ➢ Docker image is generated from Dockerfile
  - ➢ Each layer has a corresponding command in Dockerfile

Dockerfile	Commands	Layer	Parent
FROM	debian:buster-slim	7297	null
RUN	groupadd -r -g 999 redis && useradd -r -g redis -u 999 redis	1a7d	7297
RUN	wget -O /usr/local/bin/gosu ”/download/$GOSU_VERSION”; \	afac	1a7d
RUN	wget -O redis.tar.gz ”$REDIS_DOWNLOAD_URL”; \ ...	d11c	afac
RUN	mkdir /data && chown redis:redis /data	ae36	d11c
COPY	docker-entrypoint.sh /usr/local/bin	6102	ae36

Libc-to-Syscall Mapper

- Syscall Identification
  - In a container, libraries are represented as binaries
  - For glibc, syscall should be identified first before building the mapping
  - Syscall numbers will passed to *eax* or *rax*, and software interrupt is triggered

```
# syscall number:  0xca, syscall: futex
bb ca 00 00 00        mov    $0xca,%ebx
…
89 d8                 mov    %ebx,%eax
0f 05                 syscall
```

(a)

```
# syscall number: 0xca, syscall: futex
b8 ca 00 00 00        mov    $0xca,%eax
48 8d 3d d7 32 0b 00  lea    0xb32d7(%rip),%rdi
0f 05                 syscall
```

(b)

```
# syscall number:  0x0, syscall: read
31 c0                 xor    %eax,%eax
0f 05                 syscall
```

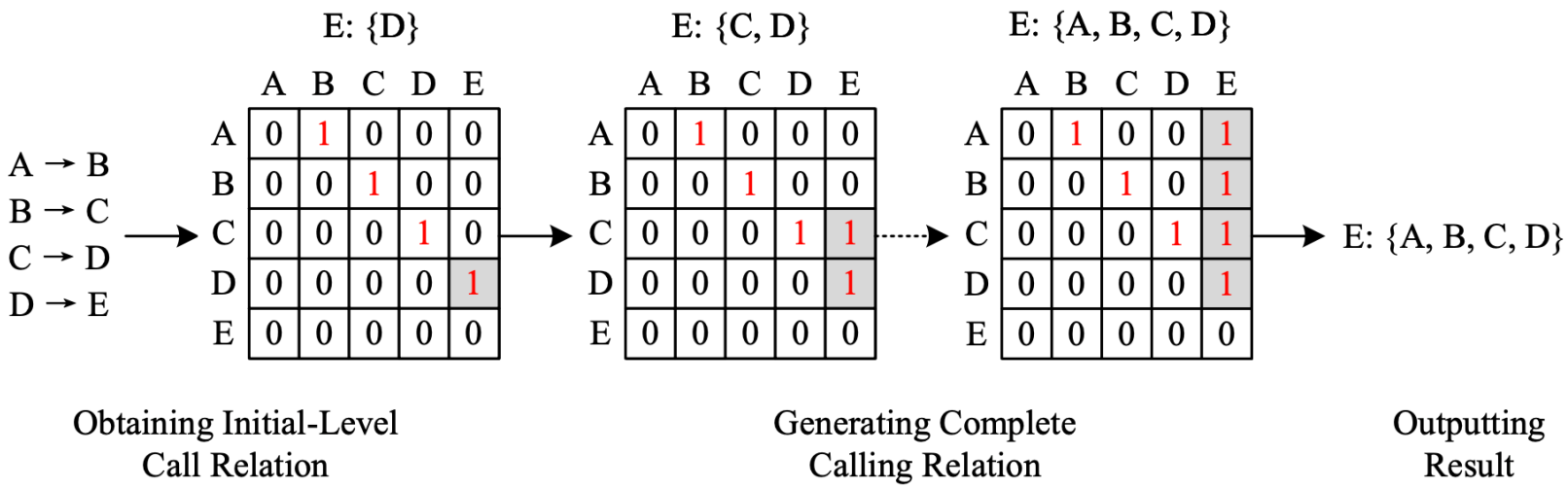
(c)

```
# syscall number:  0xf, syscall: rt_sigreturn
48 c7 c0 0f 00 00 00  mov    $0xf,%rax
0f 05                 syscall
```

(d)

Libc-to-Syscall Mapper

- Syscall Mapping Table Construction



Syscall-to-Capability Mapper

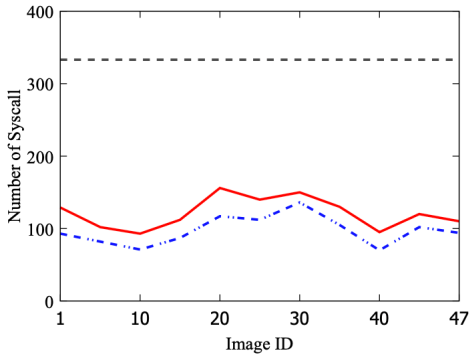
- Syscall-to-Capability mapping can only be obtained from analyzing the Linux kernel
- We compile the Linux kernel into one LLVM bytecode and analyze all functions that take a capability as a parameter

Function	Pos	Function	Pos	Function	Pos
capable	1	security_capable	3	has_capability_noaudit	2
ns_capable	2	amd_iommu_capable	1	pci_find_ext_capability	2
sk_capable	2	ns_capable_noaudit	2	pci_bus_find_capability	3
cap_capable	3	netlink_ns_capable	3	security_capable_noaudit	3
file_ns_capable	3	intel_iommu_capable	1	pci_find_next_capability	3
has_capability	2	netlink_net_capable	2	capable_wrt_inode_uidgid	2
sk_ns_capable	3	sk_filter_trim_cap	3	has_ns_capability_noaudit	3
sk_net_capable	2	cred_has_capability	2	snd_seq_event_port_attach	3
netlink_capable	2	pci_find_capability	2	snd_hda_override_amp_caps	4
selinux_capable	3	__vm_enough_memory	3	ieee80211_ie_build_vht_cap	3
iommu_capable	2	__netlink_ns_capable	3	pci_find_next_ext_capability	3
has_ns_capability	3				

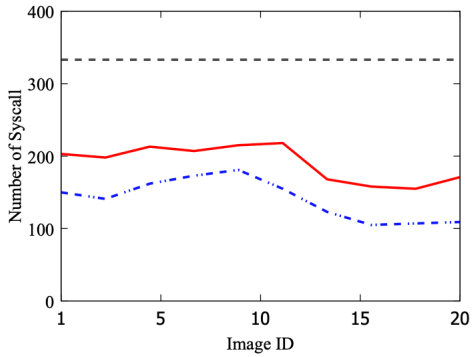
Experiment Setup

- **Hardware:** Intel i5-8265U 4-core processor and 8GB of RAM
- **Software:** Ubuntu 18.04 with Linux kernel 4.10, GNU C Library (glibc) 2.29, and Docker 20.10.2
- **Dataset:** 193 popular Docker images that can be classified into 5 categories in terms of their functionality, i.e., database, OS, DevOps, language, and infrastructure.

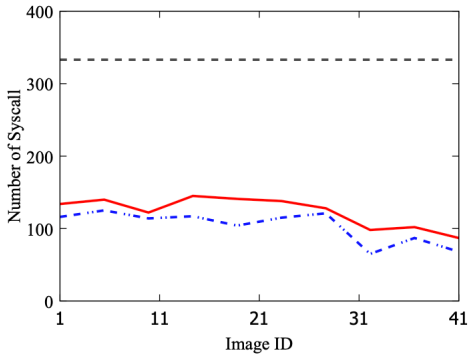
System Call Reduction



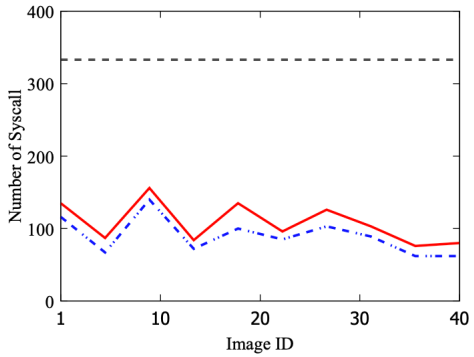
(a) Database



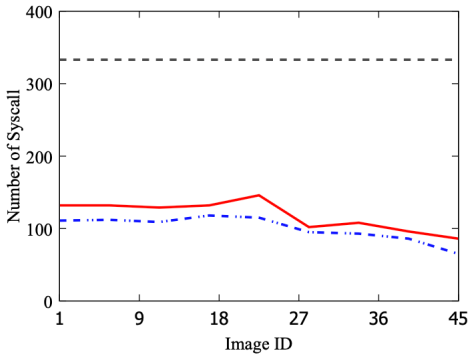
(b) OS



(c) Infrastructure



(d) Language



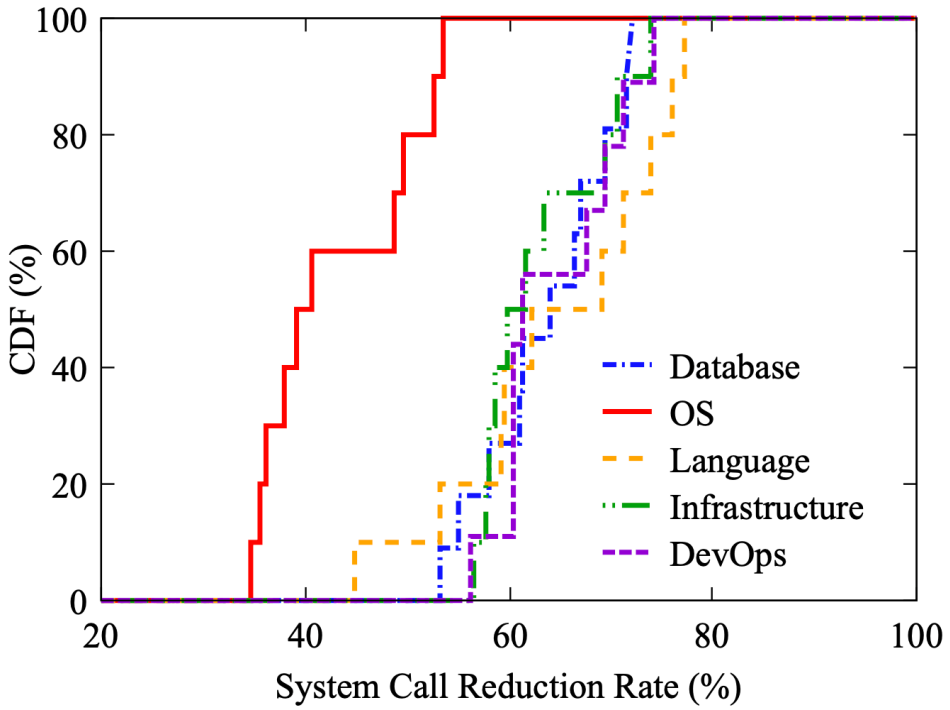
(e) DevOps

The **dashed gray** line is the default setting, the **solid red** line is SysCap, and the **dotted blue** line is dynamic tracking.

Overall, dynamic tracking and SysCap finds 105 and 127 system calls on average.

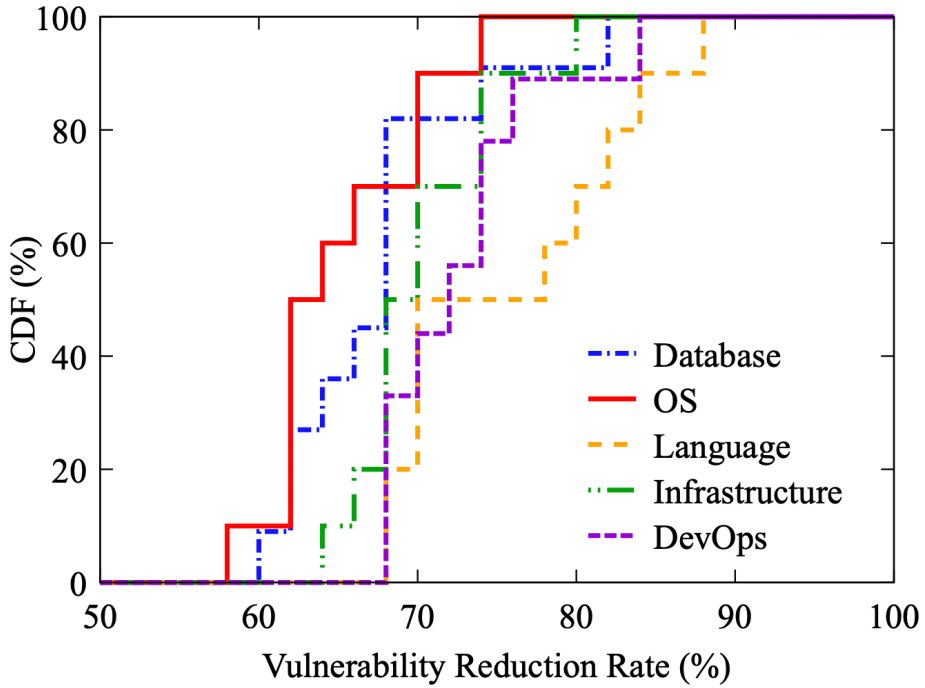
System Call Reduction

- SysCap can reduce up to 78% system calls and 62% system calls on average
- The average system call reduction rate for each category is 63.3% (Database), 42.6% (OS), 67.6% (Infrastructure), 62.8% (Language) and 64.6% (DevOps)



Vulnerability Reduction

- Almost all the vulnerabilities reduction rate are stable between 60% and 80%



Comparison with State-of-the-art work

Image*	Filtered System Calls		Vulnerability Reduction	
	Confine	SysCap	Confine	SysCap
couchdb	157	240 (+83)	50%	68% (+18%)
ubuntu	198	135 (-63)	68%	66% (-2%)
alpine	162	165 (+3)	65%	70% (+5%)
redis	179	223 (+44)	58%	62% (+4%)
node	170	192 (+22)	62%	64% (+2%)
postgres	141	213 (+72)	52%	66% (+14%)
mysql	149	183 (+34)	60%	64% (+4%)
nginx	177	188 (+11)	67%	68% (+1%)
mongo	152	193 (+41)	53%	60% (+7%)
python	154	177 (+23)	66%	70% (+4%)
traefik	211	246 (+35)	59%	66% (+7%)
mariadb	139	177 (+38)	62%	68% (+6%)
httpd	175	199 (+24)	66%	70% (+4%)
golang	197	198 (+1)	78%	78% (+0%)
centos	199	120 (-79)	76%	74% (-2%)

*Images were ranked based on total downloads as of May 27, 2021.

Capability Reduction

- With the syscall-to-capability mapping table, we can make the capability official document clear
  - ➢ e.g., from the mapping, the required syscall for CAP_SETGID are setfsgid, setgid, setgroups, setregid, and setresgid
- Applying capability customization can reduce the attack surface further
  - ➢ 78.5% of containers require less than 7 capabilities, and 5.3 capabilities are required on average

Conclusion

- We design SysCap to automatically generate required system calls and capabilities for Docker images
- SysCap statically obtains a libc-to-syscall mapping and syscall-to-capability mapping table
- After evaluating 193 popular images, SysCap can filter 62% unnecessary system calls and more than a half default capabilities, reducing the attack surface hugely

Q & A