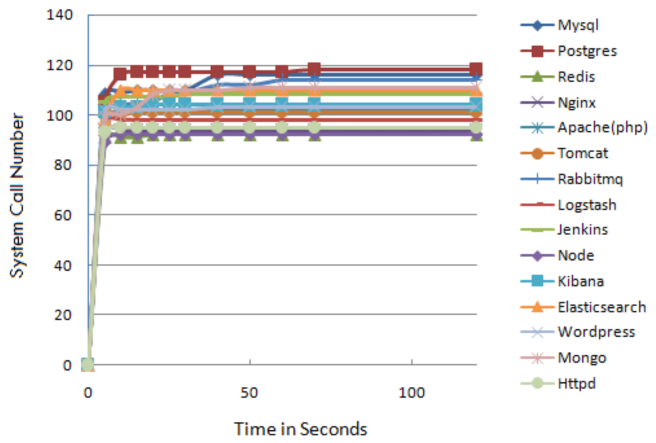
eBook:

Hardening Kubernetes Containers Security with Seccomp

An often overlooked way to harden Kubernetes containers' security is by applying seccomp profiles. A relatively ancient security mechanism in the Linux kernel, seccomp (short for secure computing mode) tells the Linux kernel which system calls a process can make.

Restricting a process from accessing the kernel via system calls restricts the attack surface, and can prevent privilege escalation. The original seccomp was very restrictive and unwieldy to use. The first version of seccomp was merged in 2005 into Linux 2.6.12. It was enabled by writing a "1" to /proc/PID/seccomp. Then, the process could only make 4 syscalls: read(), write(), exit(), and sigreturn()"). Today, the seccomp-bpf extension, which uses the Berkeley Packet Filter rules, is more commonly used as it allows filtering system calls using a configurable policy.

Given the number of system calls invoked to execute a container, each of which is a potential entry vector for attackers, appropriately applying seccomp profiles goes a long way to securing a container.



Number of System Calls Invoked over Container Execution Time

Customizing seccomp profiles, in effect, provides a deeply embedded line of defense that adds a layer of protection to your application in case of breach. As the probability of any application being breached is constantly rising, limiting the possible extent of a successful breach should be applied at as many levels as possible.

Ever-increasing interconnections between applications, and increased reliance on external service providers as well as open-source images makes restricting seccomp profiles crucial to improving cloud-native security.

Filtering system calls is not the same as sandboxing. Even though it provides a mechanism to reduce the kernel's exposed surface, filtering should be used in combination with other system hardening techniques.

Why Seccomp is Crucial to Kubernetes Security

On Kubernetes, in order to use the default seccomp profile DevOps need to add a field in the pod security context:
seccompProfile:
type: RuntimeDefault. You can either create your own custom seccomp profile, or use the default one, from the container runtime. Syscalls are typically not filtered, so securing your workload and containers with seccomp profiles considerably enhances clusters' security.

For example, imagine that you are writing an application that requires opening files. To open a file, you need to create a system call to access a file, open() syscall, and the kernel is responsible for performing it. If, for example, you want to bar processes from accessing files, you can create a seccomp profile restricting their access. If any unauthorized program or user tries to access a file, it can deny access, trigger an alert, or even kill the application if that's the desired outcome.

Basically, seccomp profiles are enforcing defined syscalls that your application requires from the kernel, for example, read files, communicate, or any of the about 200 syscalls that can be calibrated with seccomp profiles.

How to Increase Kubernetes' Security with Seccomp

Seccomp profiles are not native to Kubernetes, which makes integrating seccomp profiles in your Kubernetes security strategy slightly more complex.
The procedure recommended by Kubernetes is to define a .json file to specify precisely which syscalls you want to allow or block or audit.

An additional operation, installing that JSON file on the node, is then required.

What Are the Components of a Seccomp?

You can also choose to audit the syscalls

A seccomp is composed of three basic elements:

1. The defaultaction
2. The architectures AKA archmaps
3. The syscalls

1. The defaultaction is self-explanatory and will be applied by default to any system call not otherwise defined. The two main defaultaction values are:

   - SCMP_ACT_ERRNO to block system calls execution.
   - SCMP_ACT_ALLOW to allow system calls execution.

2. The architectures element defines the targeted architectures based on system calls IDs. At the kernel level, the filter is based on the system calls IDs, and these might vary based on the architecture they are running on, so selecting the right system call ID for each architecture is crucial.

3. The element lets you define each system call defaultaction and action, where the value in action supersedes the value in defaultaction. Selecting the most appropriate action enables granular fine tuning of the kernel system calls filter.

Here is a list of the available actions with a short description of what they perform:

SCMP_ACT_KILL_THREAD (or SCMP_ACT_KILL)	The syscall is not executed, and the kernel terminates the thread that made the system call. Other threads in the same thread group will continue to execute. **Warning:** Depending on the application being enforced (i.e., multi-threading) and its error handling, using the action to block syscalls may affect the functioning of the overall application as it does not trigger any alert.

SCMP_ACT_KILL_PROCESS (from Linux 4.14)	Causes the kernel to immediately terminate entire processes, with a core dump, when calling a syscall that does not match the configured seccomp rules.
SCMP_ACT_TRAP	Causes the kernel to decline executing the syscall and to send a thread-directed SIGSYS signal to the thread that tried to make the call. This requires setting various fields in the siginfo_t structure: * si_signo will contain SIGSYS. * si_call_addr will show the address of the system call instruction. * si_syscall and si_arch will indicate which system call was attempted. * si_code will contain SYS_SECCOMP. * si_errno will contain the SECCOMP_RET_DATA portion of the filter return value.

SCMP_ACT_ERRNO	Causes the kernel to decline executing the syscall and opt instead to return an error. **Warning:** the type of error handling enforced by the application determines if the failure to execute is silent or not. As for SCMP_ACT_KILL above, silently blocking the syscall may affect the application's overall functioning.
SCMP_ACT_TRACE	The decision on whether or not to execute the syscall will come from a tracer. If no tracer is present, it behaves like SECCOMP_RET_ERRNO. This risks being used to automate profile generation or to modify the syscall being made. This action is not recommended when trying to enforce seccomp to a line of business applications.

SCMP_ACT_ALLOW	Causes the kernel to execute the syscall.
SCMP_ACT_LOG (from Linux 4.14)	Causes the kernel to execute the syscall and log the filter return action. This can come in handy for running seccomp in "complain-mode", as it enables logging the syscalls are mapped (or catch-all) without blocking their execution. It can be used in conjunction with other action types to document all allow and deny action.

Should You Apply Seccomp at the Pod Level or at Container Level?

It might sound tempting to apply seccomp at Pod level to limit the workload, but that creates a number of issues. When seccomp profiles are applied at pod level, they are applied to each individual container in that pod which creates a number of problems:

- Kubernetes has some startup containers (specifically pause containers) which require high permissions and perform many syscalls that are not related to main container business logic. Applying a seccomp profile in the pod level will need to include all the syscalls made by the pause container, which require you to give a much more extensive and less restrictive seccomp profile to the pod.
  Note: This can be mitigated by setting the "pause" container's AllowPrivilege Escalation to false, but that requires effecting changes in the source code.

- It requires excessive permissions, including :
  - capset
  - set_tid_address
  - setgid
  - setgroups
  - setuid

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: audit-pod
  labels:
    app: audit-pod
spec:
  securityContext:
    seccompProfile:
      type: Localhost
      localhostProfile: profiles/audit.json
  containers:
  - name: test-container
    image: hashicorp/http-echo:0.2.3
    args:
    - "-text=just made some syscalls!"
    securityContext:
      allowPrivilegeEscalation: false
```

Setting a Secopmp profile for a pod example

The Issues with Manually Adding Seccomp Policies to Kubernetes

While many think that host based solutions are the optimal solution we offer an alternative. Manually configuring seccomp policies is time-consuming and, additionally, might trip up a Kubernetes developer unfamiliar with seccomp.

The procedure requires understanding exactly which syscall does what, and how to configure it with the proper IDs to achieve the desired result. Given the number of possible syscalls, that knowledge is often lacking, and as seccomp are not native to Kubernetes, they have to be manually installed. This means that, to configure and integrate seccomp profiles properly, the developer has to:

- Find out which syscall each process is generating.
- Build the correct seccomp profile.
- Manually install it on each node.
- Manually install it on each new node in the future.

The result is a disastrous combination of:

- **Lack of clarity:** as each container's seccomp policies have to be defined individually, it generates a lack of clarity and visibility in the overall seccomp status.
- **Manual process:** both seccomp profile configuration and their installation on nodes have to be performed manually, which is time-consuming and increases the risk for error.
- **Difficult process:** the complex and intricate process of configuring and installing seccomp profile is both time-consuming and prone to errors.
- **High OPEX:** the direct hour-cost needed to configure and install seccomp profiles coupled with the risk of application crash in case of error generates high operating cost

The Simple Way to Integrate Seccomp into Your Kubernetes Security strategy

At Cisco, we have developed a solution, Secure Cloud-Native, to simplify the process of integrating seccomp profiles to your Kubernetes security strategy by simply selecting a recommended profile or defining a custom profile directly from the UI.
Secure Cloud-Native then seamlessly installs those seccomp profiles as needed.

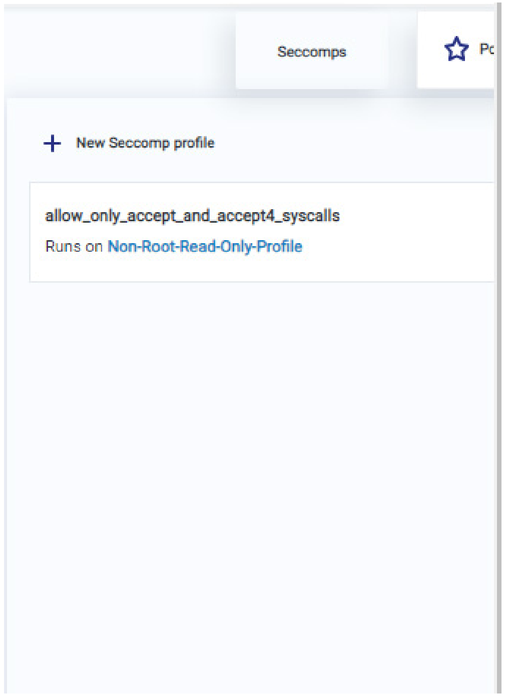
In addition, the smart recommendation module will monitor your low-level processes and identify each syscall they perform. Based on this information, Secure Cloud-Native will recommend the best off-the-shelf seccomp profile for each process.

This means that, instead of having to evaluate and define a seccomp profile for each syscall individually, developers will be able to simply follow a template procedure based on the actual processes at play. If needed, advanced customization for complex specific processes can can always be added.

Regardless of reliance on a template or custom seccomp profile, Secure Cloud-Native will perform the nodes' installation and update it when new nodes are added.

This is an example of how Secure Cloud-Native works:

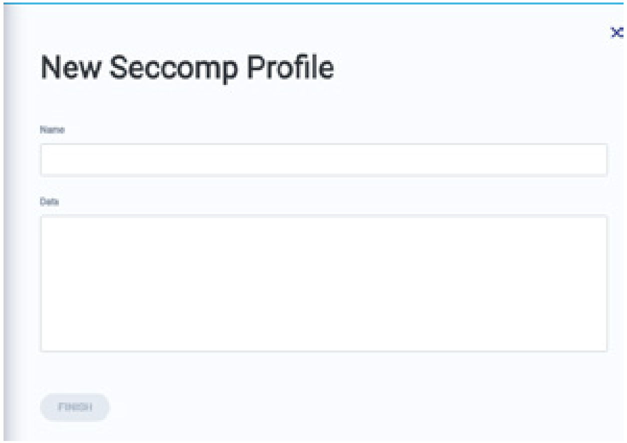
1. Click *Seccomps* (on the right).



allow_only_accept_and_accept4_syscalls
Runs on Non-Root-Read-Only-Profile

2. Click *New Seccomp profile*.

3. Enter a name for the profile.



New Seccomp Profile

Name

Data

4. Enter details for the seccomp policy as a JSON block, and then click *FINISH*. For example:

JSON

```
{
   "defaultAction": "SCMP_ACT_LOG", <--- Default action is to log/detect the system call
   "architectures": [
      "SCMP_ARCH_X86_64" ,
      "SCMP_ARCH_X86" ,
      "SCMP_ARCH_X32"
   ],
   "syscalls": [
      {
         "names": [
            "arch_prctl" ,
            "sched_yield" ,
            "futex" ,
            "write" ,
            "mmap" ,
            "exit_group" ,
            "madvise" ,
            "rt_sigprocmask" ,
            "getpid" ,
            "gettid" ,
            "tgkill" ,
            "rt_sigaction" ,
            "read" ,
            "getpgrp"
         ],
         "action": "SCMP_ACT_ALLOW" <------------- Allowed system calls
      },
      {
         "names": [
            "add_key" ,
            "keyctl" ,
            "ptrace"
         ],
         "action": "SCMP_ACT_ERRNO" <------------- Blocked system calls
      }
   ]
}
```

In short Secure Cloud-Native will:

- Seamlessly install seccomp profiles on nodes.
- Shrink the time needed to configure and install seccomp profile.
- Reduce error by automating a large part of the process.
- Provide a far better bird eyes view of the seccomp profiles.
- Provide smart recommendations/custom profiles based on monitoring the low-level processes.

Instead of spending precious time researching, tinkering, configuring, and installing seccomp profiles, developers can focus on supervising the solution activity, while creating custom seccomp profiles for advanced complex applications.

This is bound to save considerable time and drastically reduce the potential for introducing human errors.

If you are interested in finding out more or in being amongst the first users, contact us.