Kubernetes Container Escape Using Linux Kernel Exploit | CrowdStrike

By Manoj Ahuje

Archived: 2026-04-05 16:15:46 UTC

On Jan. 18, 2022, researchers found a heap base buffer overflow flaw (CVE-2022-0185) in the Linux kernel (5.1-rc1+) function "legacy_parse_param" of filesystem context functionality, which allows an out-of-bounds write in kernel memory. Using this primitive, an unprivileged attacker can escalate its privilege to root, bypassing any Linux namespace restrictions.

CVE-2022-0185 Needs CAP_SYS_ADMIN

This flaw is triggered by sending 4095 bytes or greater input to legacy_parse_param function, which provides write primitive to exploit this vulnerability. The input data is added via fsconfig system call, which is used for configuring filesystem creation context (e.g., ext4 filesystem superblock).

```
fsconfig(fd, FSCONFIG_SET_STRING, "\x00", val, 0);
```

Use of fsconfig system call to add NULL terminated string pointed by "val"

To use the fsconfig system call, an unprivileged user must have CAP_SYS_ADMIN privileges at least in its current namespace. That means if a user can enter another namespace that has these privileges, it will be enough to exploit this vulnerability. If CAP_SYS_ADMIN privileges are unavailable to an unprivileged user, attackers have another way to get these privileges using unshare(CLONE_NEWNS|CLONE_NEWUSER) system call. Unshare system call lets a user create or clone a namespace or user that can have necessary privileges required to conduct further attack. This type of technique becomes very relevant in Kubernetes and the container world where Linux namespaces are used to isolate pods.Let's take a closer look at how an unshare system call can add CAP_SYS_ADMIN privileges in Kubernetes and the mitigation. The exploit code and proof of concepts were released Jan. 25 on github by the research team that discovered this vulnerability.

unshare to Gain CAP_SYS_ADMIN Privileges on Kubernetes

Seccomp profile protects Linux namespace boundaries by blocking dangerous system calls being used by pods that are isolated using such namespaces. Docker and other runtimes have used such profiles to protect namespace boundaries for a long time. But Kubernetes by default doesn't apply any Seccomp or AppArmor/SELinux profile restrictions when the pod is scheduled to run. Hence, such a pod by default gets free access to dangerous system calls that allow it to escalate privileges and gain necessary capabilities such as CAP_SYS_ADMIN for further attack. In the following example, you can see that a non-privileged Kubernetes pod is scheduled to run and what its capabilities are. As you can see, this pod doesn't have CAP_SYS_ADMIN privileges assigned to its root user.

```
 $ # create non-privileged pod
$ kubectl run attacker --image=manojahuje/attacker
pod/attacker created

$ kubectl exec -it attacker bash
root@attacker:/# lsns | grep user
        NS TYPE   NPROCS   PID USER COMMAND
4026531837 user        4   1 root /bin/sh -c while true;do sleep 5; done
root@attacker:/# capsh --print
Current: cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net_bi
Bounding set =cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_ne
```

Attacker pod doesn't have CAP_SYS_ADMIN privileges

Now let's escalate the privileges of our command line session to achieve CAP_SYS_ADMIN capabilities using an unshare system call. As you can see below, the attacker pod has gained CAP_SYS_ADMIN privileges after the use of unshare.

```
 kubectl exec -it attacker bash
root@attacker:/# unshare -Urm
#
#
# capsh --print
Current: =ep
Bounding set =cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setu
#
# lsns | grep user
        NS TYPE   NPROCS   PID USER COMMAND
4026532810 user        2  3149 root -sh
```

Attacker pod gained CAP_SYS_ADMIN privileges, and user namespace is changed

Mitigation

Since v1.22, Kubernetes has provided a way to add default Seccomp or AppArmor profiles using SecurityContext to protect any pod, deployment, statefulset, replicaset or daemonset. Though this feature is Alpha at the moment, a user can add their own Seccomp or AppArmor profile and define it in SecurityContext. Let's take a look at the following pod yaml where a default Seccomp profile is applied to a pod.

```
 apiVersion: v1
kind: Pod
metadata:
  name: protected
spec:
  containers:
    - name: protected
      image: manojahuje/attacker
      securityContext:
        seccompProfile:
          type: RuntimeDefault
```

Pod with RuntimeDefault seccomp profile

Now, let's schedule this unprivileged pod and try to gain CAP_SYS_ADMIN privileges using unshare. As you can see below the unshare system call is blocked and potential attackers are unable to escalate privileges.

```
 $ kubectl apply -f pod-test.yaml
pod/protected created

$ kubectl exec -it protected bash
root@protected:/#
root@protected:/# unshare -Urm
unshare: unshare failed: Operation not permitted
```

Pod fails to escalate privileges using unshare

Kubernetes Container Escape Using CVE-2022-0185

As we saw, container orchestrators like Kubernetes heavily rely on namespace isolation to separate pods from each other on the node operating system. In addition, some of the pods or namespaces even on hosts outside the context of Kubernetes will always have CAP_SYS_ADMIN privileges. Hence if these pods are not protected by Seccomp or AppArmor profile, there is a good chance even a novice attacker will be able to get CAP_SYS_ADMIN privileges using unshare system call as discussed earlier or by any other means (e.g., by compromising privileged pods). Once such privileges are gained, this vulnerability can be exploited with relative ease. The POC was posted by researchers on twitter to show that reliable exploitation is possible on any vulnerable Kernel i.e. 5.1-rc1 and above with high probability of success.

Mitigation

You can secure your Kubernetes and Linux deployments using the following mitigation steps:

- CrowdStrike recommends upgrading the Kernel version to the latest as soon as possible. Patches are already out thanks to responsible disclosure by researchers on all major distros including Ubuntu and Red Hat.
- If you are using Kubernetes, use SecurityContext to apply Seccomp and AppArmor/SELinux policies. It has been available as an alpha feature since v1.22.
- You can also use Pod Security to enforce policy throughout the cluster using the admission controller. This feature has been available since v1.23, replacing the older Pod Security Policy (PSP).
- If you have a non-containerized environment, you can use the following command to disable the functionality.

    ```
    $ sudo sysctl -w kernel.unprivileged_userns_clone=0
    ```

- On Red Hat, you can disable user namespaces using the following commands (it is recommended that you do so for non-containerized environments):

    ```
    # echo "user.max_user_namespaces=0" > /etc/sysctl.d/userns.conf
    # sysctl -p /etc/sysctl.d/userns.conf
    ```

CrowdStrike Protection

The CrowdStrike Falcon® platform protects Kubernetes workloads by auditing the runtime configuration of each deployment. CrowdStrike follows CIS benchmarks for Kubernetes to identify any indicators of misconfiguration (IOMs). As we saw, Kubernetes workloads running without Seccomp or AppArmor/SELinux profiles are a huge risk and can result in container escape and cluster compromise. Figures 1 and 2 show CrowdStrike alerts when a Kubernetes workload is detected running without a seccomp or AppArmor/SELinux profile or with an incorrect seccomp profile. This helps with early detection of IOM and enables users to take immediate preventative measures.

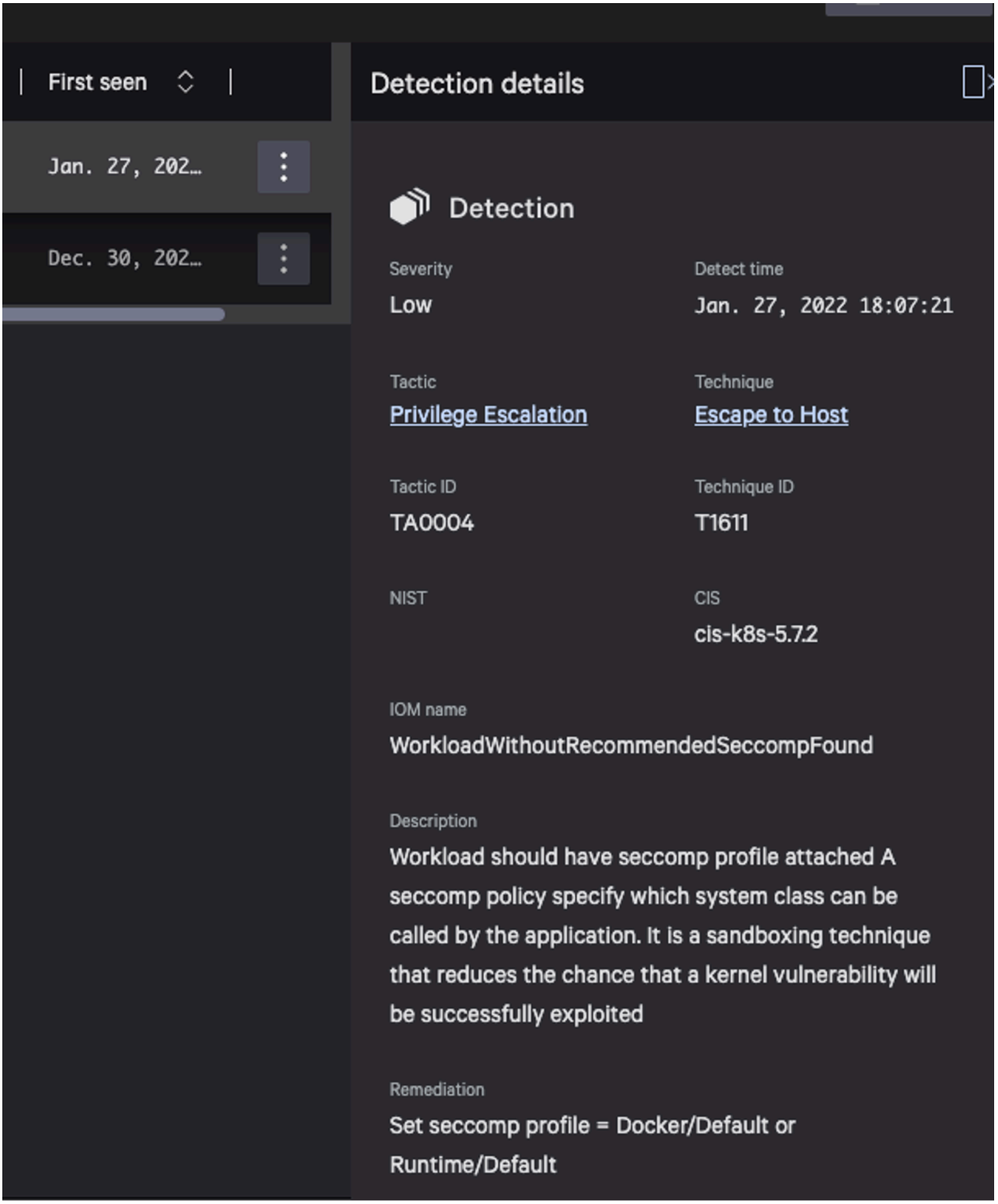

Figure 1. IOMs that CrowdStrike identified as missing seccomp profile on Kubernetes workload

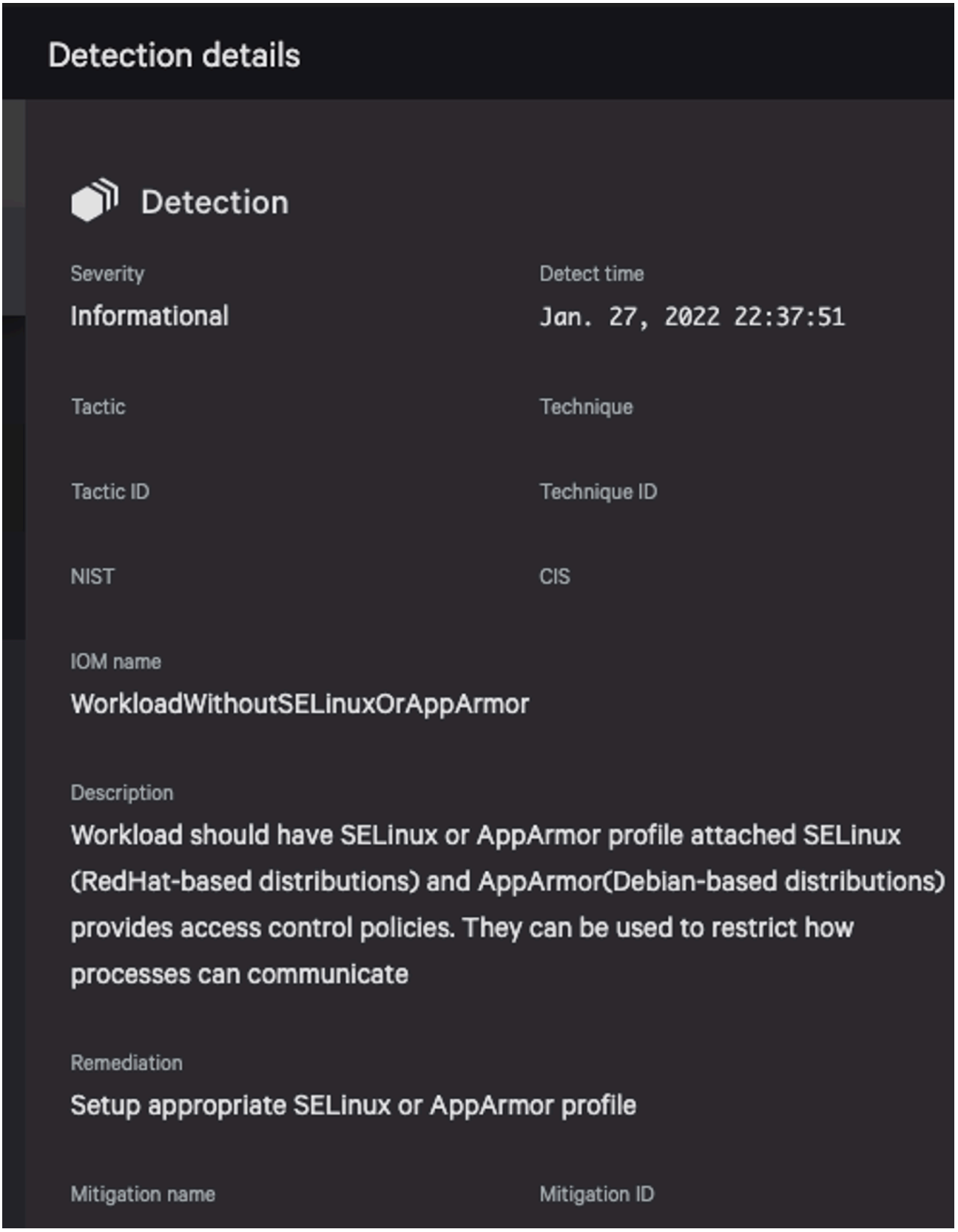

Figure 2. IOMs that CrowdStrike identified as missing AppArmor or SELinux profile on Kubernetes workload

Hunting with Falcon to Find Unshare Uses

As discussed, attackers can use Linux utility unshare to make unshare() system calls to gain CAP_SYS_ADMIN privileges. To hunt for the uses of unshare utility throughout the container environment you can use the following Falcon query, which casts the widest net possible to locate unshare utility uses within your container environment.

```
 index=main sourcetype=ProcessRollup2* event_simpleName=ProcessRollup2 event_platform=Lin OciContainerId=*
| search FileName=unshare
| stats dc(aid) as ContainerAid count(aid) as detectionCount, values(ComputerName) as endpointNames by ParentBas
| sort - detectionCount
```

You can further drill down using CrowdStrike Threat Graph® to learn more about the issue. Figure 3 shows unshare utility being used within a container located with the above Falcon hunt query.

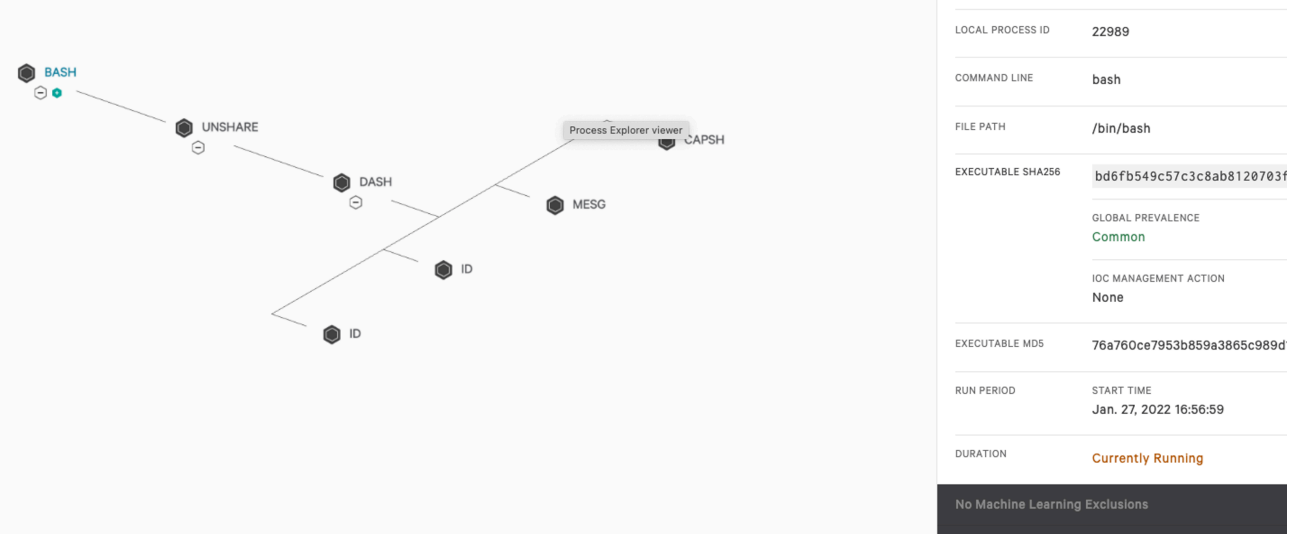

Figure 3. CrowdStrike Threat Graph for unshare utility (Click to enlarge)

Conclusion

Orchestration platforms like Kubernetes are distributed systems software that depend on a number of small components with their own lifecycle and attack surfaces. From time to time we see that a relatively simple bug can become a huge security risk when combined with design mechanics, security issues and default insecure configurations. It is therefore paramount that users stay on top of any security concerns and make appropriate changes as soon as possible. Securing containers need not be an overly complex task. Using the Falcon platform, you can easily identify security issues in your environment in real time. You can use built-in features of Kubernetes and best practices to keep your container environment safe. For enhanced security you can use integrated container security products such as CrowdStrike Falcon® Cloud Security that can protect your kubernetes environment seamlessly. CrowdStrike strives to support organizations that allow their users to stay ahead of the curve and remain fully protected from adversaries and breaches.

Additional Resources

- *Learn more about CrowdStrike Falcon® Cloud Security.*
- *Learn more about CrowdStrike Falcon® Insight™ endpoint detection and response .*

- *Test CrowdStrike next-gen AV for yourself. Start your free trial of Falcon Prevent™ today.*
