BACK TO BLOG

Detecting MITRE ATT&CK: Defense evasion techniques with Falco

BY KAIZHE HUANG - FEBRUARY 2, 2021

TOPICS: COMPLIANCE, KUBERNETES & CONTAINER SECURITY, OPEN SOURCE

The **defense evasion** category inside **MITRE ATT&CK** covers several techniques an attacker can use to avoid getting caught. Familiarizing yourself with these techniques will help secure your infrastructure.

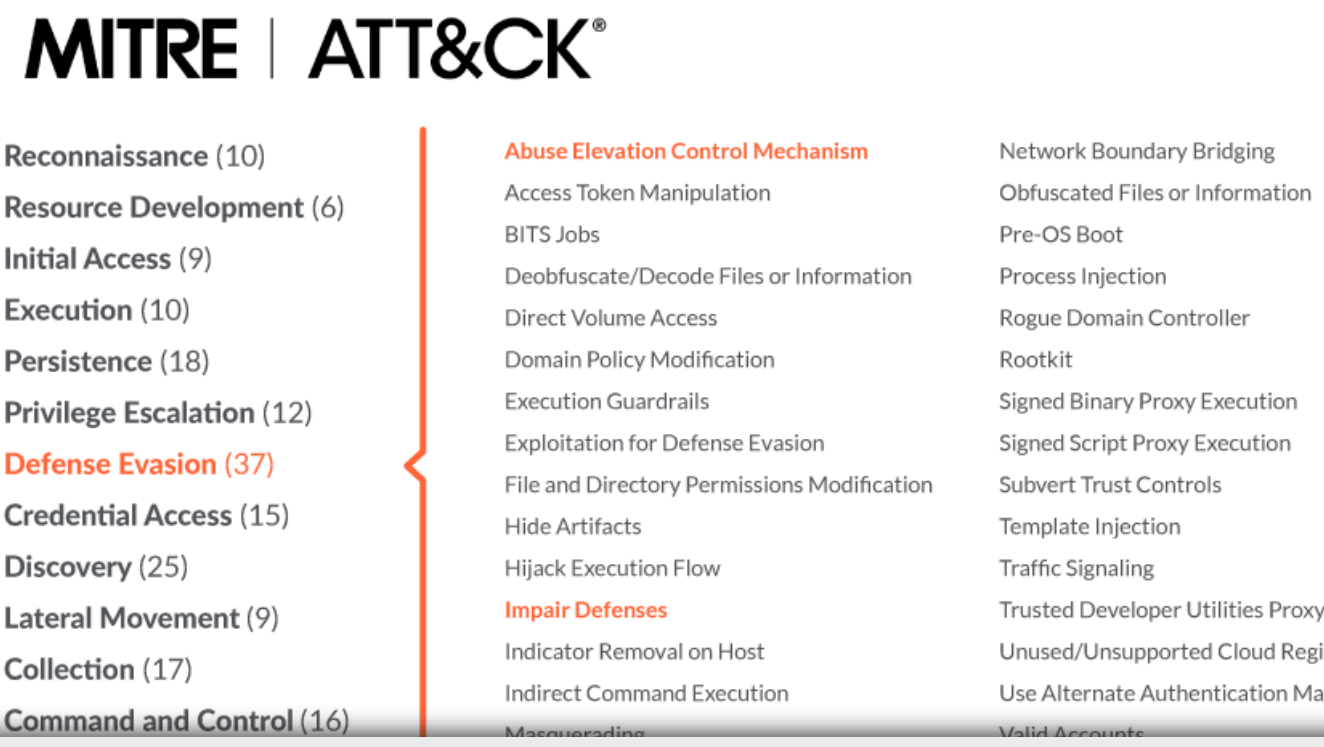
MITRE ATT&CK is a comprehensive knowledge base that analyzes all of the tactics, techniques, and procedures (TTPs) that advanced threat actors could possibly use in their attacks. Rather than a compliance standard, it is a framework that serves as a foundation for threat models and methodologies. These techniques are grouped into 14 categories.

In this article, we will walk through a few techniques that can be classified as **MITRE defense evasion**. We'll also provide examples of how an open-source tool like **Falco** can help you detect these container security attacks.

Looking for a commercial offering?
If so, checkout Sysdig Secure that provides image scanning, compliance runtime security and incident response for containers and Kubernetes. Start a free 30 day trial here!

MITRE category: Defense evasion

mask their code as known processes, or disabling the system defenses.



to have the added benefit of subverting a particular defense or mitigation.

> **This is how attackers avoid getting caught. Learn about defense evasion in MITRE ATT&CK, and how to protect yourself with Falco.**
>
> Click to tweet

Hands on with defense evasion techniques

Let's try two of these techniques. We'll see how they can benefit an attacker, and we'll look for some insights that will help us detect them.

Abuse elevation control mechanism: setuid and setgid

In modern operating systems, processes run under a user space with limited permissions. However, they include mechanisms to elevate privileges when it's time to install an application, open a port, or run other administrative tasks.

Attackers can abuse these elevation control mechanisms to run code as administrators and access to private information.

Let's dig into this concept with a simple example.

The following C program performs two tasks. First, it prints what user the program is running as and then reads from a sensitive file `/etc/shadow`, which stores the actual password in encrypted format (more like the hash of the password) for all of the user accounts in the system.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int main()
{
   FILE *fp;
   char line[256];
   // print user ID and the effective user ID
   printf("uid: %d, euid: %d\n", getuid(), geteuid());
   fp = fopen("/etc/shadow", "r");
   return 0;
}
```


Now, let's compile this code, set `root` as the owner for the resulting binary, and allow other users to execute it.

What will happen if we run this program with and without the `setuid` bit set?

When the setuid bit is not set (note the `-rwx` permissions on the file):

Running the program as the current user (`ubuntu`) raises some errors:

Reading the `/etc/shadow` file fails.

Note that both the user ID and effective user ID are 1000, corresponding to the `ubuntu` user.

Now, what will happen if the `setuid` is set? Note the `-rws` permissions on the file:

Running the program as the current user (`ubuntu`) is now a success:

Although it was `ubuntu` who executed the program (`uid: 1000`), the effective user is `root` (`euid: 0`). Ultimately, the `ubuntu` user borrowed the `root` user's privilege to read the `/etc/shadow` file successfully due to the `setuid` bit.

We can draw a quick conclusion here: **You should remove the unnecessary setuid bit from files owned by high privilege users**.

Impair defenses: Disable or modify tools

Once an attacker gets access into the victim's environment, before they start probing the environment too much, it will try to disable security tools so that they may not be blocked or caught.

We can take our recent research of the kinsing attack as an example. There, we noticed such a pattern inside the malicious payload:

```
# Disable firewall
ufw disable
# Remove iptable rules
iptables -F
# Stop apparmor
service apparmor stop
systemctl disable apparmor
# Stop SELinux
setenforce 0
```

```
rm -rf /etc/init.d/agentwatch /usr/sbin/aliyun-service
rm -rf /usr/local/aegis*
systemctl stop aliyun.service
systemctl disable aliyun.service
service bcm-agent stop
yum remove bcm-agent -y
apt-get remove bcm-agent -y
```

This code snippet is part of the Kinsing payload, and we added comments to make it easier to follow.

We can see how the attacker tried to disable the firewall, flushed the iptables rules, and disabled SELinux, AppArmor, and the security agents from Ali Cloud. The script will succeed if the attacker managed to acquire root privileges when they hacked into the environment, as the operations above usually require admin privileges.

This may be easier than you think. As we saw in our latest annual container security and usage report, **56% of container images are running as root**. If the attacker

Detecting MITRE defense evasion techniques with Falco

Now that we know what threats we are facing, we can start defending against them.

You can cover some attack entry points in advance. With image scanning, you can check your images for things like unnecessary open ports, images running as root, and known vulnerabilities. However, attacks occur in runtime, while your containers are already in production. You need runtime security tools that continuously monitor your workloads searching for abnormal behaviour.

Falco is the CNCF open-source project for runtime threat detection for containers and Kubernetes. It is a container security tool designed to **detect anomalous activity** in your containers. Falco taps into system calls to generate an event stream of all system activity. Falco's rules engine then allows you to create rules based on this event stream, allowing you to alert on system events that seem abnormal. Falco's rich language allows you to write rules at the host level and identify suspicious activity.

Let's follow up on our previous examples, and see how you can detect defense evasion with Falco.

Detect setuid bit or setgid bit changes of a file

This is as simple as:

```
find . -perm /6000
```

But what if the file permissions change during runtime?

You can use the following Falco rule to detect such a scenario:

```
- rule: Set Setuid or Setgid bit
  desc: >
    When the setuid or setgid bits are set for an application,
    this means that the application will run with the privileges of the owning user or group
respectively.
    Detect setuid or setgid bits set via chmod
```



```
    command=%proc.cmdline container_id=%container.id container_name=%container.name
image=%container.image.repository:%container.image.tag)
  priority:
    NOTICE
  tags: [process, mitre_persistence]
```

And once there is an attempt to add the `setuid` bit or `setgid` bit to a file, an alert will be generated:

Detect disabling security tools

Going back to the Kinsing attack, we saw how attackers will try to disable security tools for the purpose of not being blocked or caught.

Here is the Falco rule that detects such behavior:

```
- macro: disable_apparmor
  condition: (proc.name in (systemctl, service) and (proc.cmdline contains "disable" or
proc.cmdline contains \"stop\") and (proc.cmdline contains "apparmor"))
- macro: disable_selinux
  condition: (proc.cmdline = "setenforce 0")
- macro: disable_ufw
  condition: (proc.name=ufw and proc.cmdline contains "disable")
- rule: Disable Security Tools
  desc: Detect an attempt to disable security tools like ufw, AppArmor, SELinux
  condition: spawned_process and (disable_apparmor or disable_selinux or disable_ufw)
```

If someone tries to disable ufw, the following alert is generated:

Conclusion

Falco has become the defacto standard for runtime container security. It relies on syscalls and K8s audit logs to detect intrusions and malicious activity. By mapping Falco rules to the MITRE ATT&CK Defense Evasion category, security teams can streamline their threat detection and response workflows.

To learn more about Falco, please check out the Falco repo and Falco website, or join our CNCF Falco slack channel.

Looking for a commercial offering?
If so, checkout Sysdig Secure that provides image scanning, compliance runtime security and incident response for containers and Kubernetes. Start a free 30 day trial here!

sysdig

SUBMIT →

Also keep me informed of Sysdig news + updates