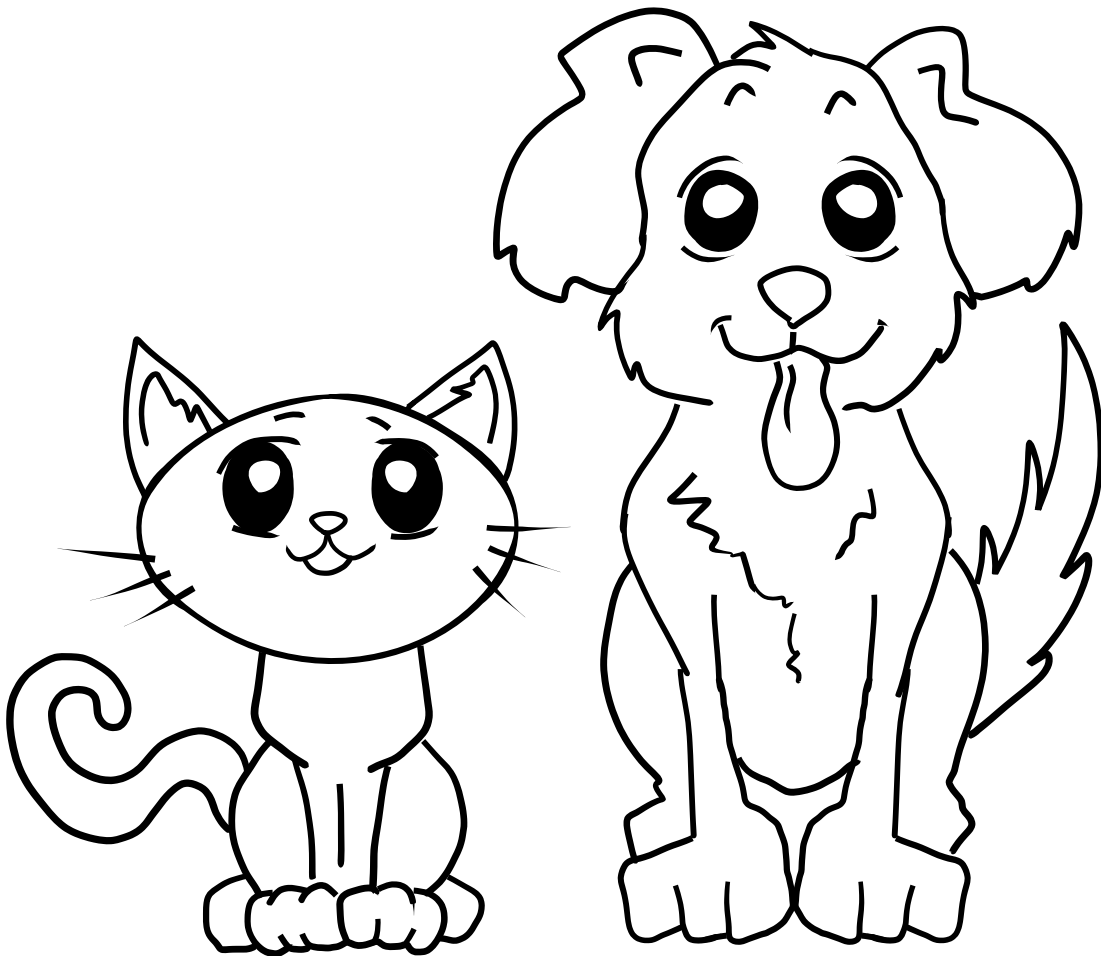# TYPE ENFORCEMENT

PROCESS TYPES

The SELinux primary model of enforcement is called type enforcement. Basically this means we define the label on a process based on its type, and the label on a file system object based on its type.

Imagine a system where we define types on objects like cats and dogs. A cat and dog are process types.
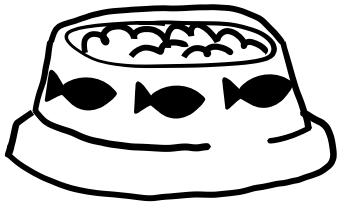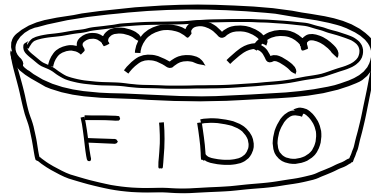
CAT   DOG

OBJECT TYPES

We have a class of objects that they want to interact with which we call food. And I want to add types to the food, cat_chow and dog_chow.
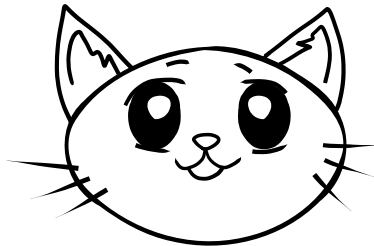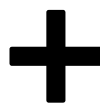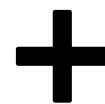

CAT_CHOW

DOG_CHOW

POLICY RULES

As a policy writer, I would say that a dog has permission to eat dog_chow food and a cat has permission to eat cat_chow food. In SELinux we would write this rule in policy, as shown below.
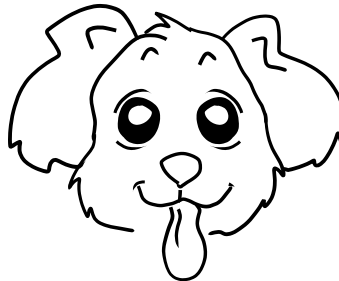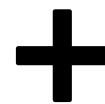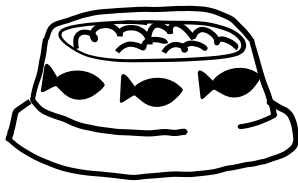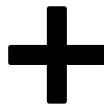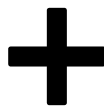

ALLOW + CAT +

ALLOW + DOG +

CAT_CHOW:FOOD

+

EAT

DOG_CHOW:FOOD

+

EAT

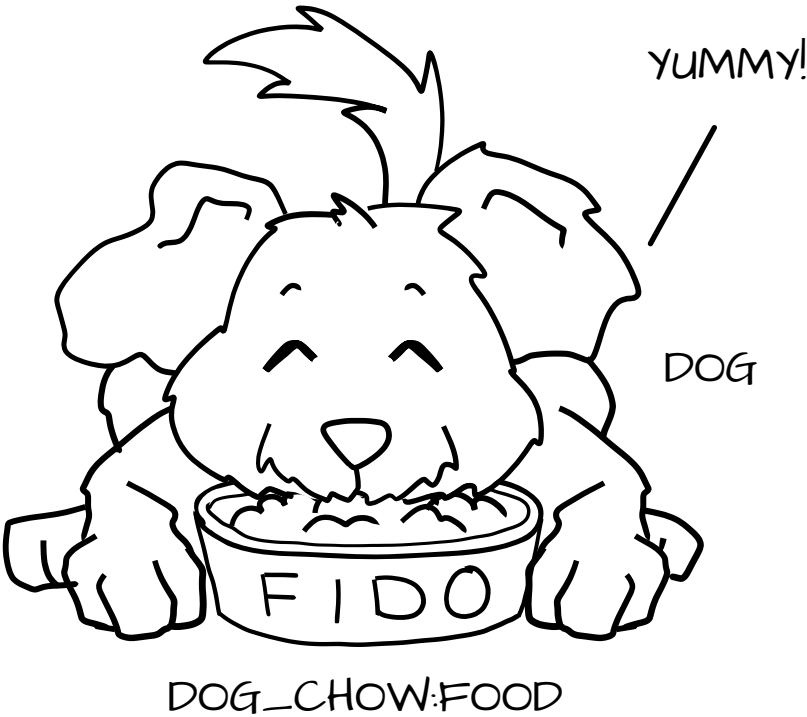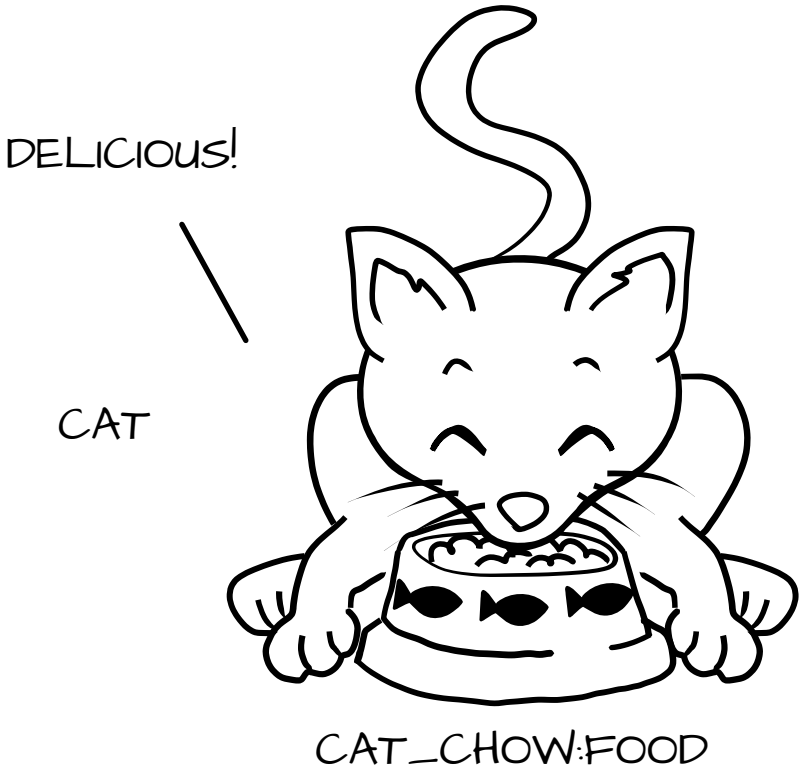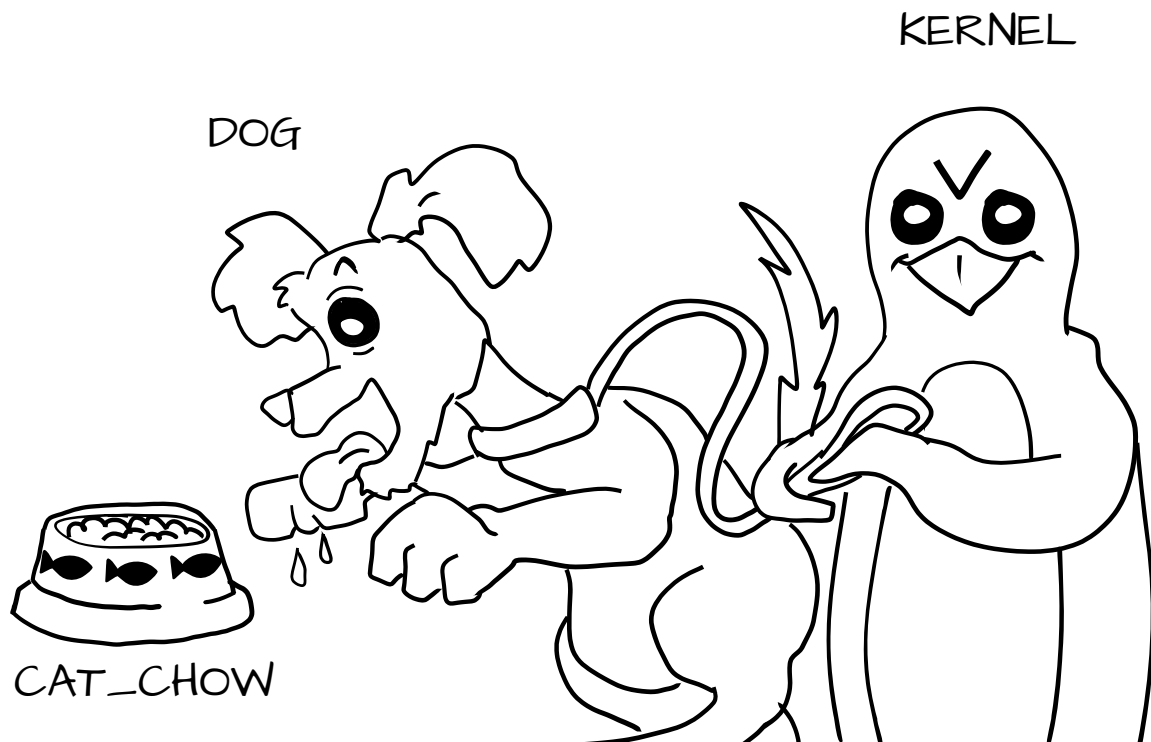With these rules the kernel would allow the cat process to eat food labeled cat_chow and the dog to eat food labeled dog_chow.

DELICIOUS!

CAT

CAT_CHOW:FOOD

YUMMY!

DOG

DOG_CHOW:FOOD

But in an SELinux system, everything is denied by default. This means that if the dog process tried to eat the cat_chow, the kernel would prevent it.
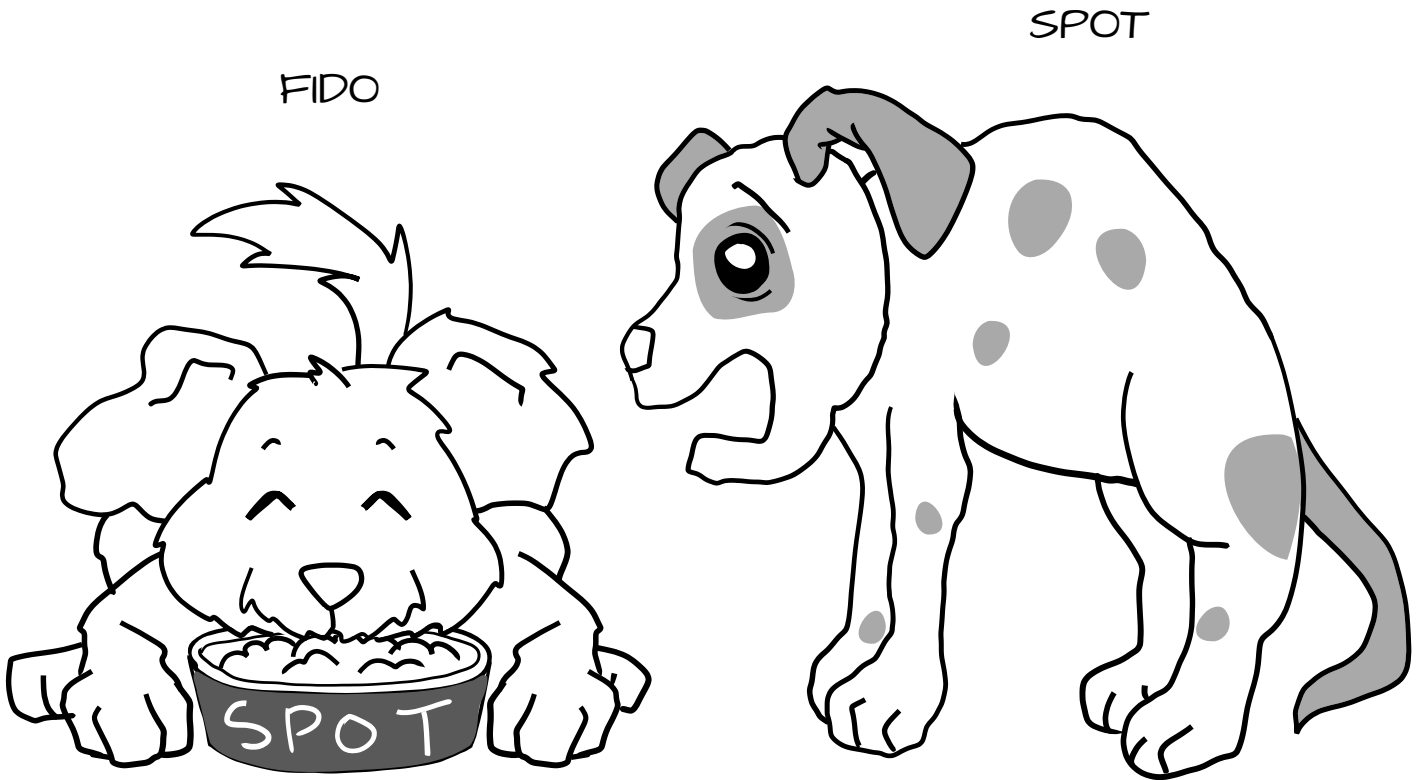

Likewise, cats would not be allowed to touch dog food.
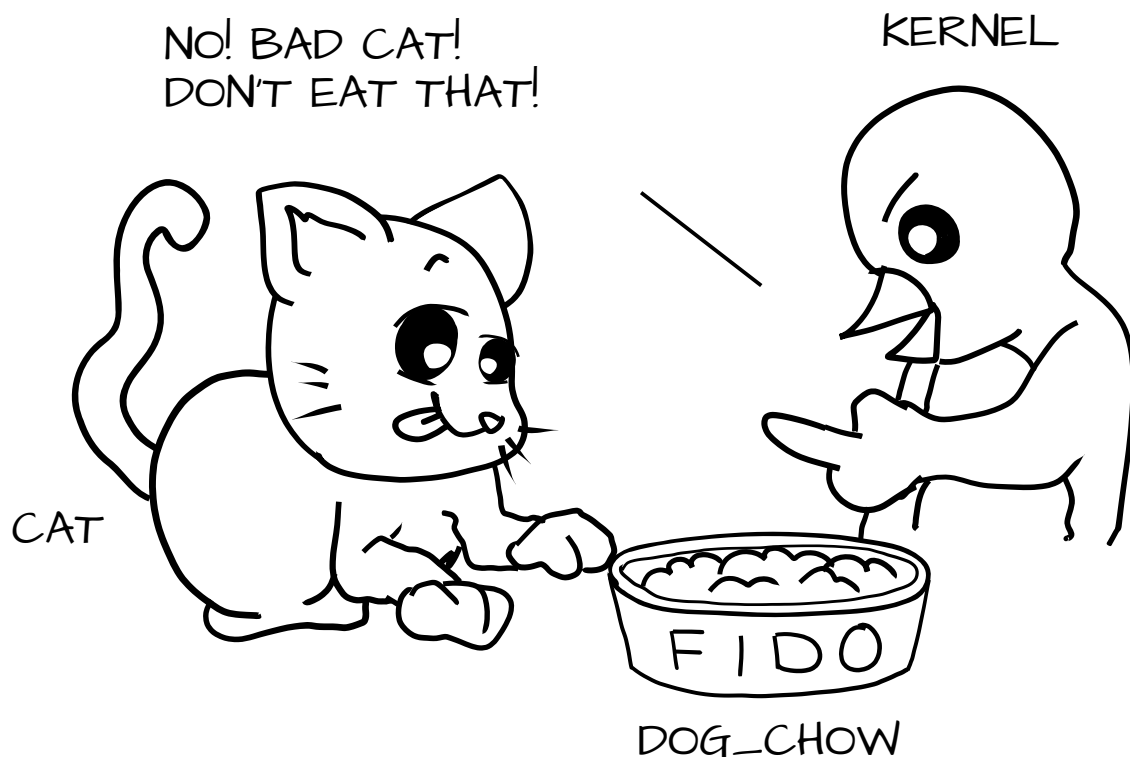
MCS ENFORCEMENT

We've typed the dog process and cat process, but what happens if you have multiple dog processes: Fido and Spot? You want to stop Fido from eating Spot's dog_chow.

One solution would be to create lots of new types, like Fido_dog and Fido_dog_chow. But, this will quickly become unruly because all dogs have pretty much the same permissions.

To handle this we developed a new form of enforcement, which we call multi Category Security (MCS). In MCS, we add another section of the label which we can apply to the dog process and to the dog_chow food. Now we label the dog process as dog:random1 (Fido) and dog:random2 (Spot).
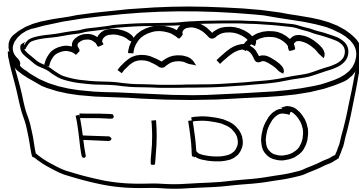
We label the dog chow as dog_chow:random1 (Fido) and
dog_chow:random2 (Spot).
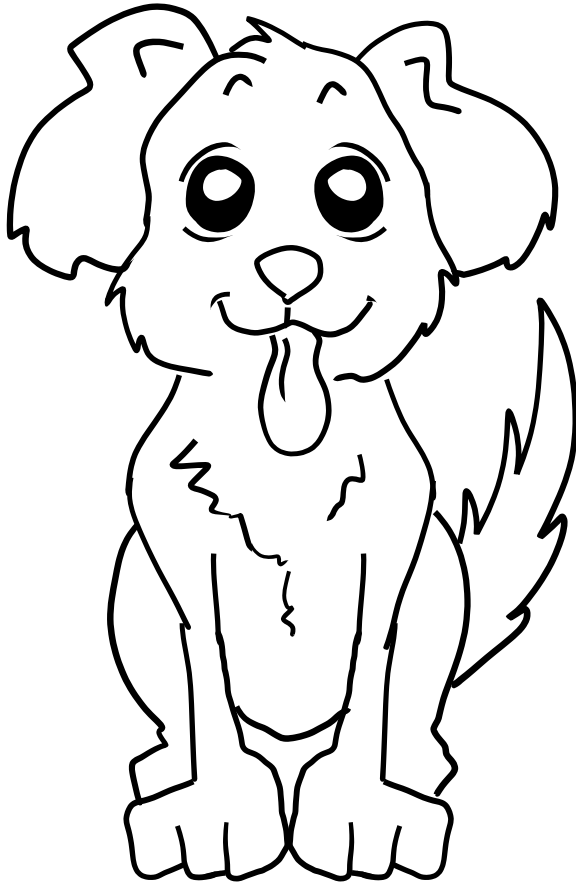
DOG:RANDOM1
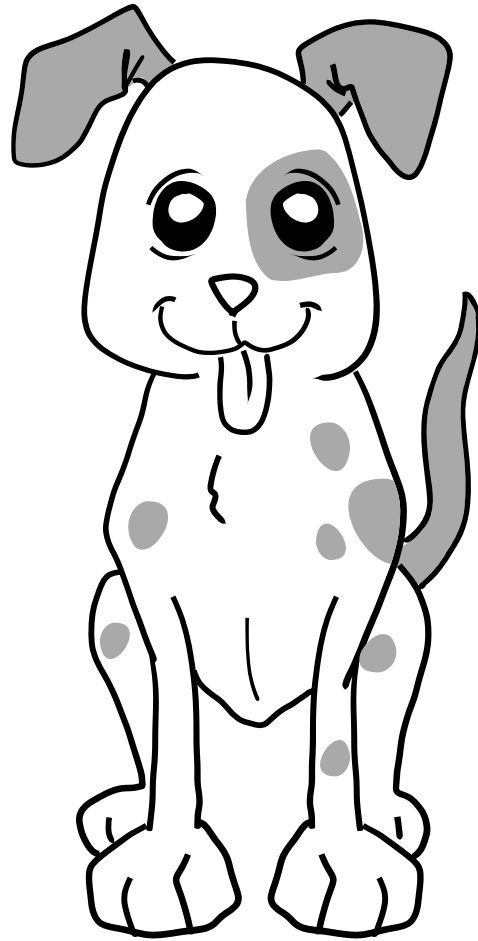
DOG:RANDOM2

DOG_CHOW:
RANDOM1

DOG_CHOW:
RANDOM2

MCS rules say that if the type enforcement rules are OK and the random MCS
labels match exactly, then the access is allowed, if not it is denied.

TYPE ENFORCEMENT

Fido (dog:random1) trying to eat cat_chow:food is denied by type enforcement.

KERNEL

DOG

CAT_CHOW
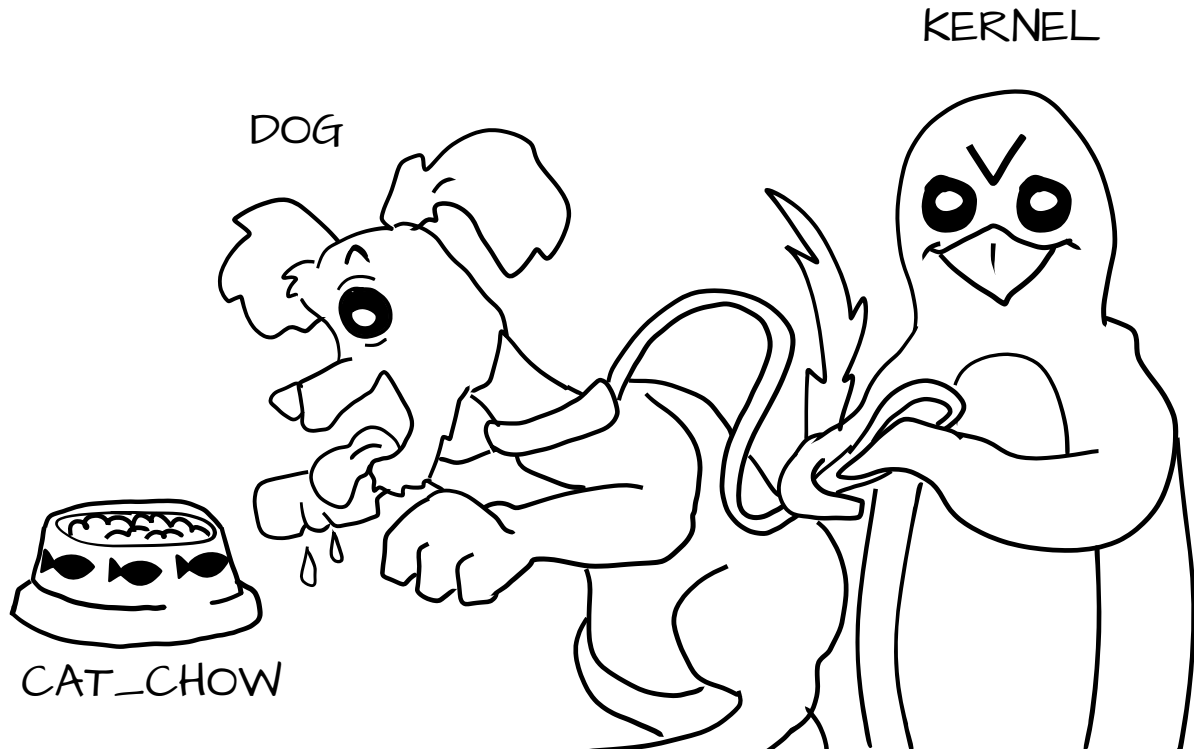
Fido (dog:random1) is allowed to eat dog_chow:random1.

DOG:RANDOMI

FIDO

DOG_CHOW:
RANDOMI

MCS ENFORCEMENT

Fido (dog:random1) denied to eat spot's (dog_chow:random2) food.

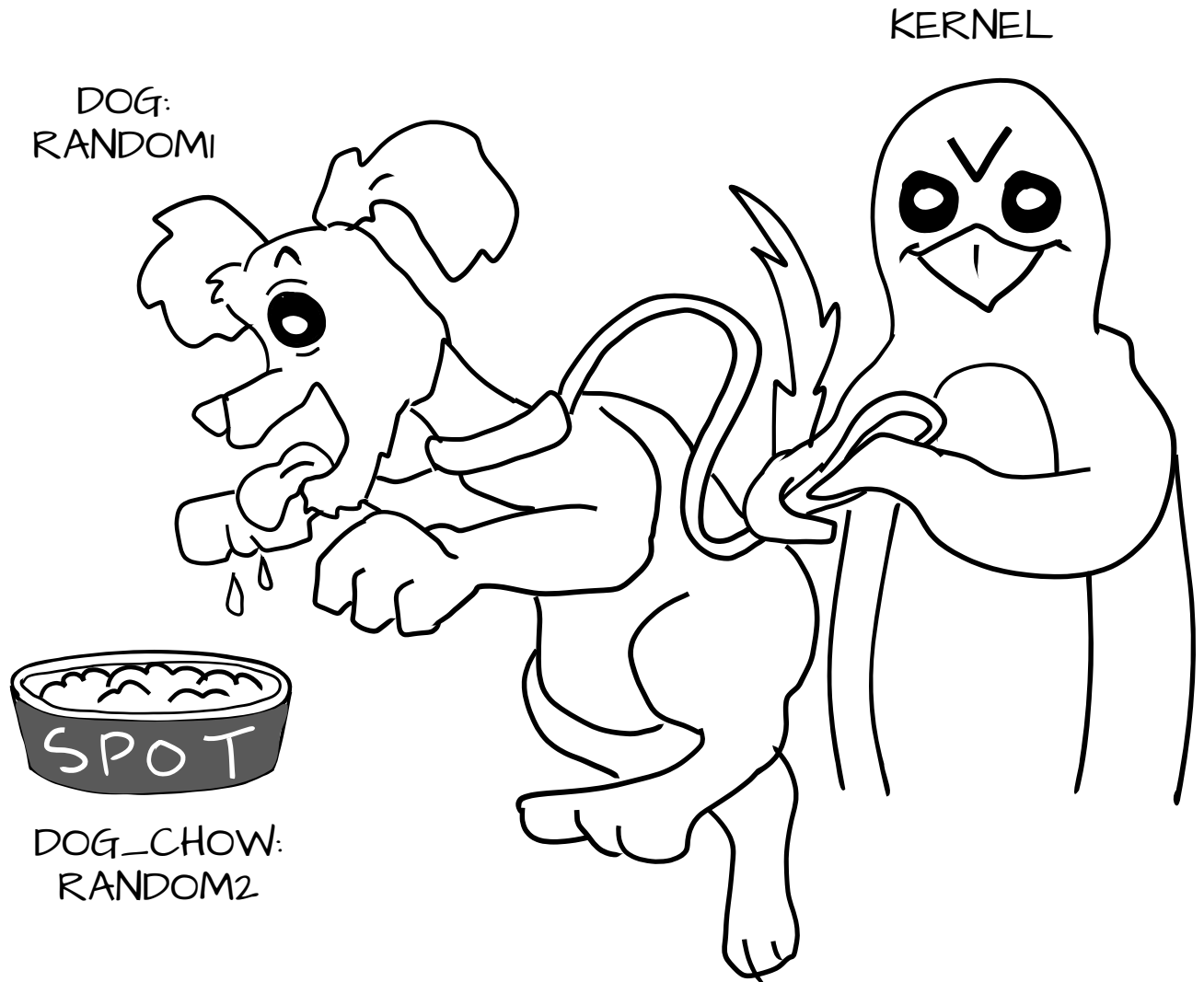# MLS ENFORCEMENT

Another form of SELinux enforcement, used much less frequently, is called Multi Level Security (MLS); it was developed back in the 60s and is used mainly in trusted operating systems like Trusted Solaris.

The main idea is to control processes based on the level of the data they will be using. A secret process can not read top secret data.

Instead of talking about different dogs, we now look at different breeds. We might have a Greyhound and a Chihuahua.
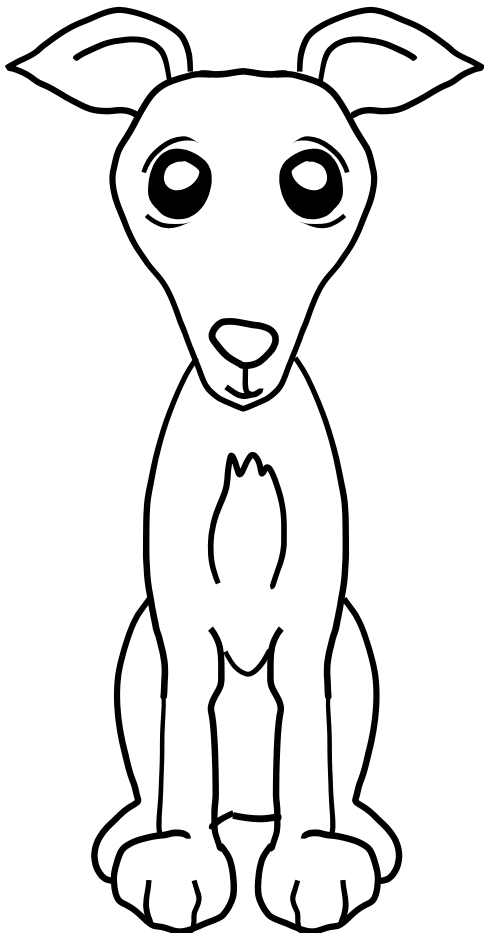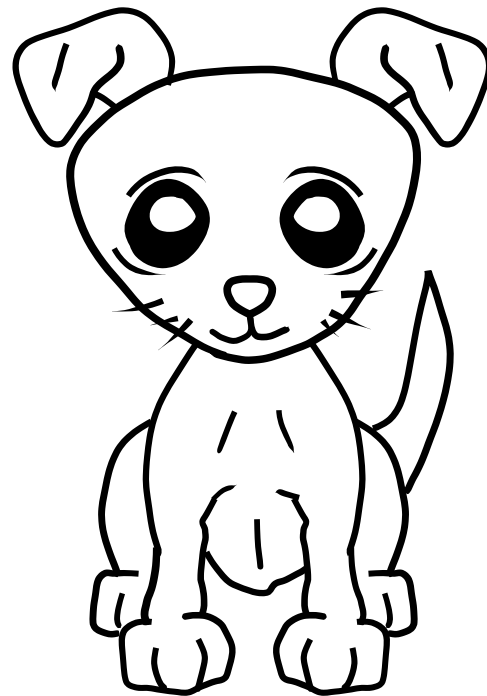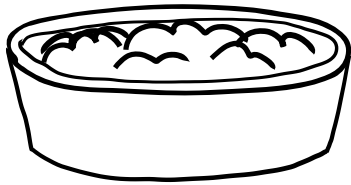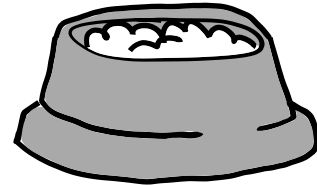
GREYHOUND   CHIHUAHUA

We might want to allow the Greyhound to eat any dog food, but a Chihuahua could choke if it tried to eat Greyhound dog food.

We want to label the Greyhound as dog:Greyhound and his dog food as dog_chow:Greyhound, and label the Chihuahua as dog:Chihuahua and his food as dog_chow:Chihuahua.
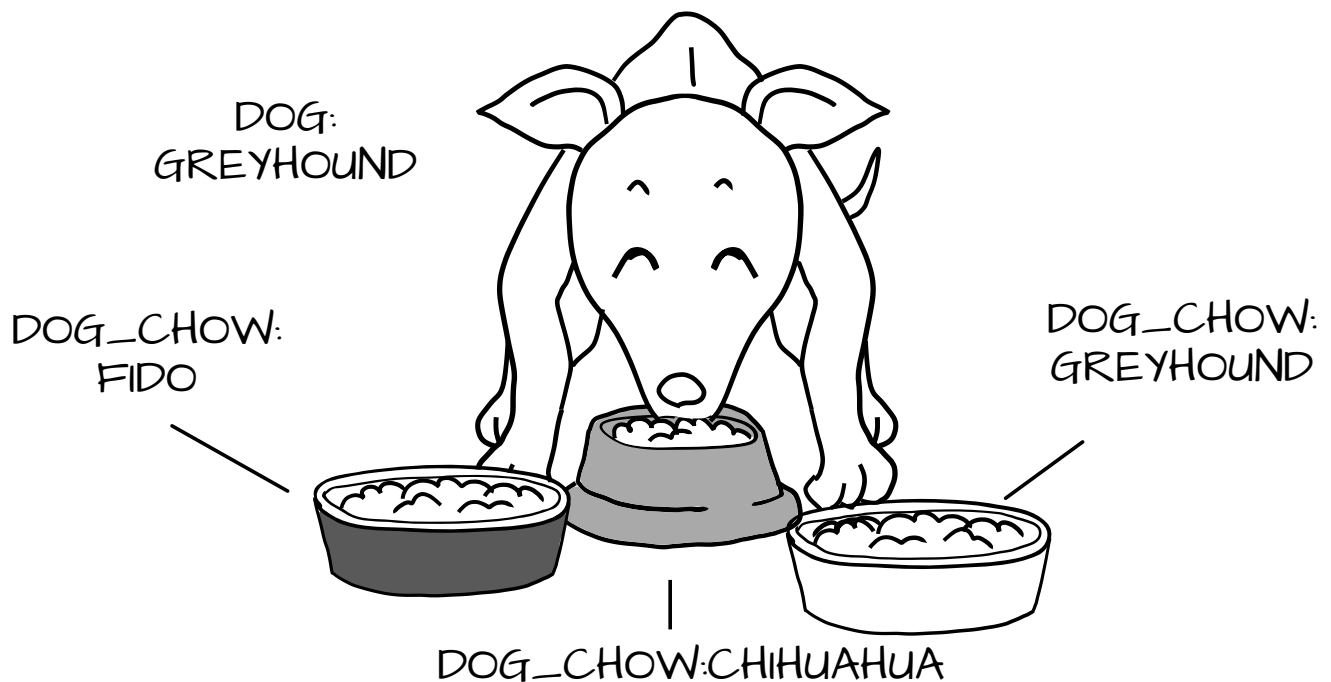
DOG_CHOW:
GREYHOUND

DOG_CHOW:
CHIHUAHUA

With the MLS policy, we would have the MLS Greyhound label dominate the Chihuahua label. This means dog:Greyhound is allowed to eat dog_chow:Greyhound and dog_chow:Chihuahua.
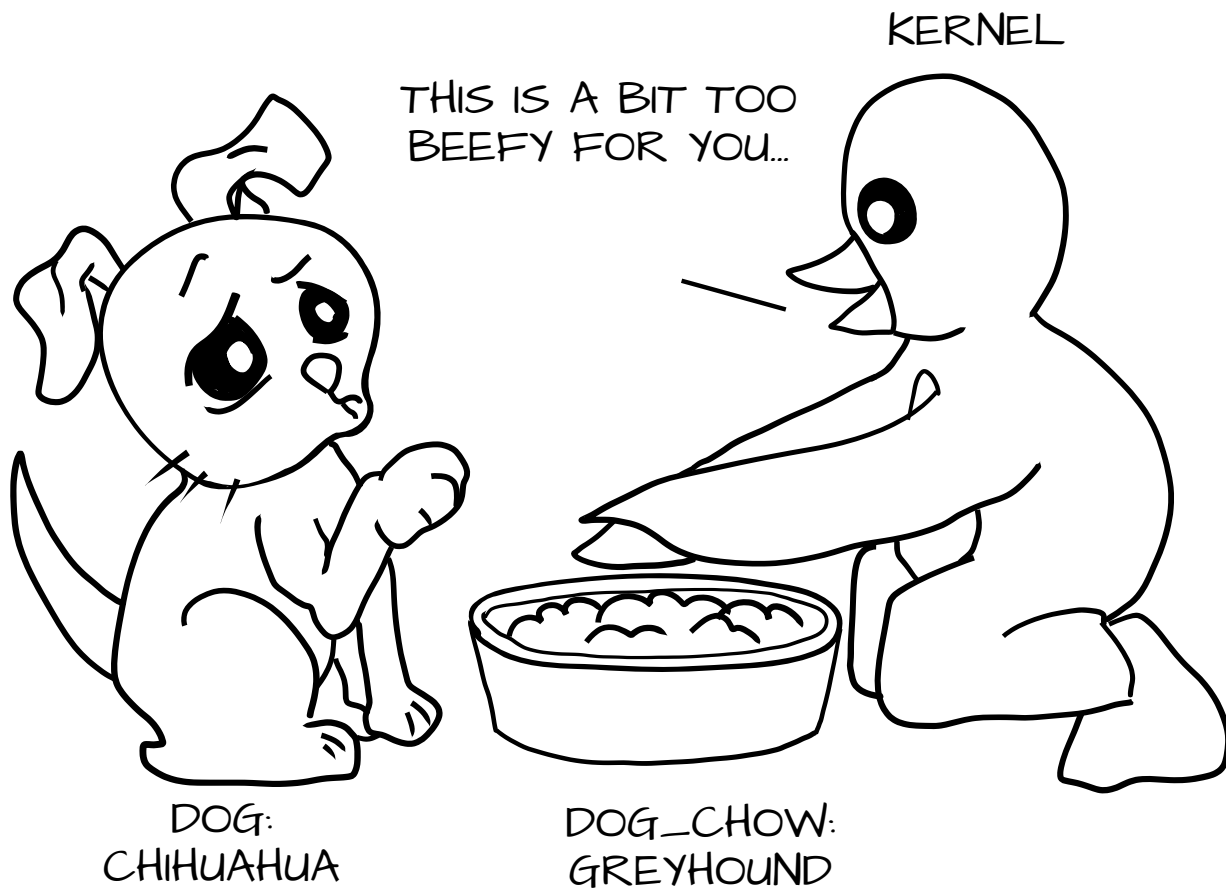
DOG:
GREYHOUND

DOG_CHOW:
FIDO

DOG_CHOW:
GREYHOUND

DOG_CHOW:CHIHUAHUA

DOG:
CHIHUAHUA

DOG_CHOW:
CHIHUAHUA

But dog:Chihuahua is not allowed to eat dog_chow:Greyhound.

THIS IS A BIT TOO BEEFY FOR YOU...

KERNEL

DOG: CHIHUAHUA

DOG_CHOW: GREYHOUND

Of course, dog:Greyhound and dog:Chihuahua are still prevented from eating cat_chow:Siamese by type enforcement, even if the MLS type Greyhound dominates Siamese.

KERNEL

DOG: GREYHOUND

DOG: CHIHUAHUA

CAT_CHOW: SIAMESE