Single Access Point / Policy Enforcement Point / Reference Monitor:

What counts as an access point? Perhaps the most important part of this constraint.

A spring boot application serves HTTP requests on a single port, which could count as a single access point. In this case, we can put a constraint on the number of @SpringBootApplication annotated classes.

If each endpoint counts as an access point, then that should also be possible to constrain using @Controller and the various @RequestMapping annotations.

There's no one-rule-fits-all, each system must define what counts as an access point. Either by specifying the rule such that it includes everything that counts as an access point in your system, or by annotating all access points explicitly (perhaps more prone to mistakes, easy to forget an annotation).

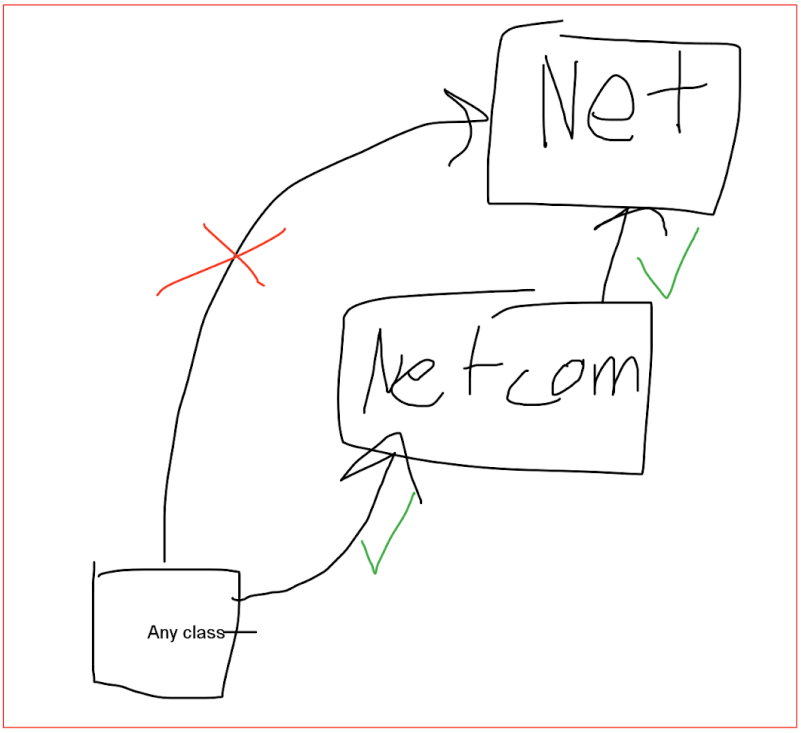
Input Guard / Intercepting Validator

No way of extracting the methods of which a method calls. You can however start with the classes considered to be a part of the model and ensure that these are only accessed by classes marked as "validated input".

Multiple possible approaches

- Using annotations, we can ensure that classes of one annotation may not access classes having another annotation. This can be made as granular as one would like where specific fields or methods may be annotated.
- Similar to the above approach, you may start at the classes that need to have secure input and then check whether it is only used by classes marked as having "validated input".
- In its simplest form, this may also be enforced using a layered architecture where the rules are in terms of entire packages rather than annotated classes/methods.

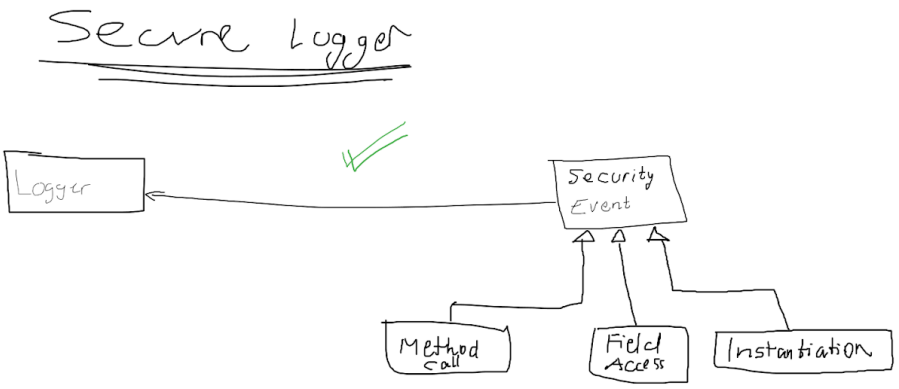
Secure Access Layer / Secure Service Facade



…

Very similar to constraint 2; skip for now

“Secure Logger”

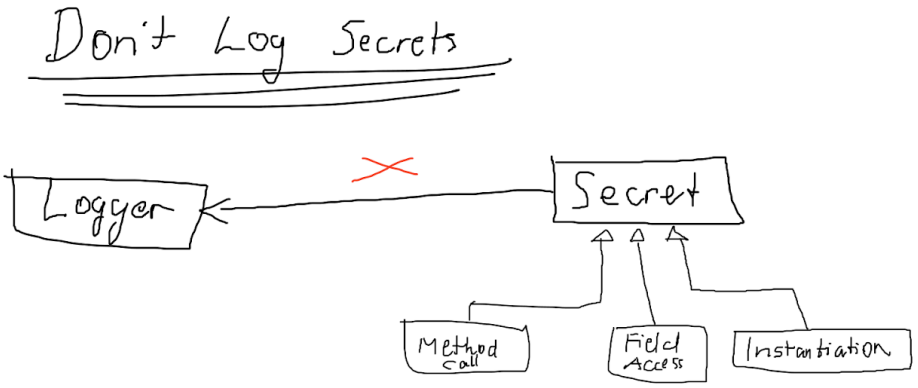


Enforce a call to the logger when a security event occurs.

Need to define/determine on what level you should log events as they need to have enough information. Intra-class getter/setter might not know enough about the caller. The current rule tests the constraint on the **caller** of methods that expose assets.

As for auditability concerns, the methods in the logger class could be defined such that they must be supplied arguments that help identify who did what.

Don’t Log Secrets



How can we know if an asset/secret is exposed in a log entry?

- We can tell if a method that accesses a secret also calls the logger, but we can’t verify that the secret is one of the arguments.
- We can check the static signature of the logger function(s), ensuring that the arguments are not a type that is annotated as a secret. However, a secret could be transformed into arbitrary data making it impossible to discern whether or not it was actually sent.

- A possible extension would be to more accurately determine whether or not a sensitive variable is sent to a logger even after it is transformed into a different type (e.g. string). Examining how a variable is composed?