# ENV03-J. Do not grant dangerous combinations of permissions

Certain combinations of permissions can produce significant capability increases and should not be granted. Other permissions should be granted only to special code.

### AllPermission

The permission `java.security.AllPermission` grants all possible permissions to code. This facility was included to reduce the burden of managing a multitude of permissions during routine testing as well as when a body of code is completely trusted. Code is typically granted `AllPermission` via the security policy file; it is also possible to programmatically associate `AllPermission` with a `ProtectionDomain`. This permission is dangerous in production environments. Never grant `AllPermission` to untrusted code.

### ReflectPermission, suppressAccessChecks

Granting `ReflectPermission` on the target `suppressAccessChecks` suppresses all standard Java language access checks when the permitted class attempts to operate on package-private, protected, or private members of another class. Consequently, the permitted class can obtain permissions to examine any field or invoke any method belonging to an arbitrary class [Reflect 2006]. As a result, `ReflectPermission` must never be granted with target `suppressAccessChecks`.

According to the technical note *Permissions in the Java SE 6 Development Kit* [Permissions 2008], Section ReflectPermission, target `suppressAccessChecks`:

> ***Warning***: Extreme caution should be taken before granting this permission to code, *for it provides the ability to access fields and invoke methods in a class. This includes not only public, but protected and private fields and methods as well.*

### RuntimePermission, createClassLoader

The permission `java.lang.RuntimePermission` applied to target `createClassLoader` grants code the permission to create a `ClassLoader` object. This permission is extremely dangerous because malicious code can create its own custom class loader and load classes by assigning them arbitrary permissions. A custom class loader can define a class (or `ProtectionDomain`) with permissions that override any restrictions specified in the systemwide security policy file.

*Permissions in the Java SE 6 Development Kit* [Permissions 2008] states:

> *This is an extremely dangerous permission to grant. Malicious applications that can instantiate their own class loaders could then load their own rogue classes into the system. These newly loaded classes could be placed into any protection domain by the class loader, thereby automatically granting the classes the permissions for that domain.*

## Noncompliant Code Example (Security Policy File)

This noncompliant example grants `AllPermission` to the `klib` library:

```
// Grant the klib library AllPermission
grant codebase "file:${klib.home}/j2se/home/klib.jar" {
  permission java.security.AllPermission;
};
```

The permission itself is specified in the security policy file used by the security manager. Program code can obtain a permission object by subclassing the `java.security.Permission` class or any of its subclasses (`BasicPermission`, for example). The code can use the resulting object to grant `AllPermission` to a `ProtectionDomain`.

## Compliant Solution

This compliant solution shows a policy file that can be used to enforce fine-grained permissions:

```
grant codeBase
    "file:${klib.home}/j2se/home/klib.jar", signedBy "Admin" {
  permission java.io.FilePermission "/tmp/*", "read";
  permission java.io.SocketPermission "*", "connect";
};
```

To check whether the caller has the requisite permissions, standard Java APIs use code such as the following:

```
// Security manager check
FilePermission perm =
    new java.io.FilePermission("/tmp/JavaFile", "read");
AccessController.checkPermission(perm);
// ...
```

Always assign appropriate permissions to code. Define custom permissions when the granularity of the standard permissions is insufficient.

## Noncompliant Code Example (`PermissionCollection`)

This noncompliant code example shows an overridden `getPermissions()` method, defined in a custom class loader. It grants `java.lang.ReflectPermission` with target `suppressAccessChecks` to any class that it loads.

```
protected PermissionCollection getPermissions(CodeSource cs) {
  PermissionCollection pc = super.getPermissions(cs);
  pc.add(new ReflectPermission("suppressAccessChecks"));   // Permission to create a class loader
  // Other permissions
  return pc;
}
```

## Compliant Solution

This compliant solution does not grant `java.lang.ReflectPermission` with target `suppressAccessChecks` to any class that it loads:

```
protected PermissionCollection getPermissions(CodeSource cs) {
  PermissionCollection pc = super.getPermissions(cs);
  // Other permissions
  return pc;
}
```

## Exceptions

**ENV03-J-EX0:** It may be necessary to grant `AllPermission` to trusted library code so that callbacks work as expected. For example, it is common practice, and acceptable, to grant `AllPermission` to the optional Java packages (extension libraries):

```
// Standard extensions extend the core platform and are granted all permissions by default
grant codeBase "file:${{java.ext.dirs}}/*" {
  permission java.security.AllPermission;
};
```

## Risk Assessment

Granting `AllPermission` to untrusted code allows it to perform privileged operations.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
|---|---|---|---|---|---|
| ENV03-J | High | Likely | Low | **P27** | **L1** |

## Automated Detection

Static detection of potential uses of dangerous permissions is a trivial search. Automated determination of the *correctness* of such uses is not feasible.

## Related Vulnerabilities

CVE-2007-5342 describes a vulnerability in Apache Tomcat 5.5.9 through 5.5.25 and 6.0.0 through 6.0.15. The security policy used in the JULI logging component failed to restrict certain permissions for web applications. An attacker could modify the log level, directory, or prefix attributes in the `org.apache.juli.FileHandler` handler, permitting them to modify logging configuration options and overwrite arbitrary files.

## Related Guidelines

| | |
|---|---|
| MITRE CWE | CWE-732, Incorrect Permission Assignment for Critical Resource |

## Android Implementation Details

The `java.security` package exists on Android for compatibility purposes only, and it should not be used. Android uses another permission mechanism for security purposes.

## Bibliography


| | |
|---|---|
| [API 2014] | Class `AllPermission`<br>Class `ReflectPermission`<br>Class `RuntimePermission` |
| [Gong 2003] | |
| [Long 2005] | Section 2.5, "Reflection" |
| [Permissions 2008] | Section "`ReflectPermission`" |
| [Reflect 2006] | |
| [Security 2006] | Security Architecture<br>Section "`RuntimePermission`" |