

Trusted Type Systems

This document gives a brief overview of the type systems built on top of the Trusted type system. The Trusted type system is a simple type system that describes the relationship between objects that may or may not be trusted and objects that are definitely trusted. By default, objects are considered untrusted and must be explicitly annotated as trusted. A trusted object can be used anywhere an untrusted one can, but an variable with an `@Untrusted` type cannot be used if variable with an `@Trusted` type one is required. A program can require trusted values in certain situations to create a guarantee about some security property. Each type system that is built using the trusted type system describes a security property, and consists of two qualifiers, where `@Untrusted` (the default) represents a variable which may or may not be secure, and `@Trusted` represents a variable that is definitely secure. Each system also has a `jdk.astub` file which describes how the JDK would be annotated with the type system qualifiers.

What are the type systems?

Trusted

Qualifiers: `@Untrusted` \rightarrow `@Trusted`

Security Property: [10] CWE-807 Reliance on Untrusted Inputs in a Security Decision

This type system is the original trusted type system, and for such a broad application it is the most appropriate. By default, variables are `@Untrusted` and must be explicitly annotated as `@Trusted`.

OsTrusted

Qualifiers: `@OsUntrusted` \rightarrow `@OsTrusted`

Security Property: [2] CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

This type system describes `@OsUntrusted` variables, which may or may not be sanitized for use in OS commands, and `@OsTrusted` variables which have definitely been sanitized for use in OS commands. Code that creates OS commands should require `@OsTrusted` inputs.

NotHardCoded

Qualifiers: `@MaybeHardCoded` \rightarrow `@NotHardCoded`

Security Property: [7] CWE-798 Use of Hard-coded Credentials

This type system describes `@MaybeHardCoded` variables that may have been hard-coded into the source code, and `@NotHardCoded` variables that have been generated instead of hard-coded. An authentication routine can require that credentials are `@NotHardCoded`.

Random

Qualifiers: @MaybeRandom :> @Random

Security Property: [31] CWE-330: Use of Insufficiently Random Values

Some random number generators are not cryptographically secure. This type system describes @MaybeRandom variables that might be random and @Random variables that are definitely securely random. Routines can require secure @Random variables when necessary.

Encrypted

Qualifiers: @Plaintext :> @Encrypted

Security Property: [8] CWE-311 Missing Encryption of Sensitive Data,
[19] CWE-327 Use of a Broken or Risky Cryptographic Algorithm

@Plaintext describes any variable that may or may not be encrypted. @Encrypted describes a variable that is known to be encrypted.

OneWayHashWithSalt

Qualifiers: @MaybeHash :> @OneWayHashWithSalt

Security Property: [25] CWE-759 Use of a One-Way Hash without a Salt

This type system describes @MaybeHash variables that may or may not be a hash, and @OneWayHashWithSalt variables that are definitely hashes with a salt. Routines can require values that are @OneWayHashWithSalt to prevent against dictionary attacks.

SafeFileType

Qualifiers: @UnknownFileType :> @SafeFileType

Security Property: [9] CWE-434 Unrestricted Upload of File with Dangerous Type

This type system describes @UnknownFileType variables that may or may not be a safe file type for upload and @SafeFileType variables that definitely are a safe file type for upload. Upload routines can require @SafeFileType variables.

Internal

Qualifiers: @Internal :> @Public

Security Property: [39] CWE-209: Information Exposure Through an Error Message

This type system describes @Internal variables that may or may not be appropriate to display to the end-user, and @Public variables that are definitely alright to display. Error handling code can require that the error messages it shows be annotated as @Public.

Encoding

Qualifiers: @UnknownEncoding :> @AppropriateEncoding

Security Property: [30] CWE-838: Inappropriate Encoding for Output Context

This type system describes @UnknownEncoding variables that have an unknown encoding, and @AppropriateEncoding variables that have the correct encoding needed to be safely

used by another system component. Note that `@AppropriateEncoding` could be renamed to reflect the specific required encoding.

Download

Qualifiers: `@ExternalResource` :> `@VerifiedResource`

Security Property: [14] CSE-494 Download of Code Without Integrity Check

This type system describes `@ExternalResource` variables that have been downloaded, such as external libraries, etc., and `@VerifiedResource` variables that have been downloaded and gone through a verification routine to guarantee they have not been tampered with. Then, when using external resources, the program can require `@VerifiedResource` variables.

Examples of Stub Files

Each type system has a corresponding JDK stub file, which contains “stub classes” with annotated method signatures but no method bodies. A checker uses the annotated signatures at compile time.

This stub file is for the `ostrusted` type system, and describes which methods in the JDK require `@OSTrusted` values:

```
import ostrustedquals.*;

package java.lang;

class Runtime {
    public Process exec(@OSTrusted String command);
    public Process exec(@OSTrusted String[] cmdarray);
    public Process exec(@OSTrusted String[] cmdarray, String[] envp);
    public Process exec(@OSTrusted String[] cmdarray, String[] envp, File
dir);
    public Process exec(@OSTrusted String command, String[] envp);
    public Process exec(@OSTrusted String command, String[] envp, File
dir);

    public void load(@OSTrusted String filename);
    public void loadLibrary(@OSTrusted String libname);
}

class ProcessBuilder {
    public ProcessBuilder command(List<@OSTrusted String> command);
    public ProcessBuilder command(@OSTrusted String... command);
    public Map<String, @OSTrusted String> environment();
}

class System {
    public static String setProperty(String key, @OSTrusted String value);
```

```
        public static void load(@OsTrusted String filename);
        public static void loadLibrary(@OsTrusted String libname);
    }
```

This stub file is for the Random type system, and describes which methods in the JDK return sufficiently random values:

```
import random.qual.*;

package java.security;

class SecureRandom {
    public @Random boolean nextBoolean();
    public @Random float nextFloat();
    public @Random double nextDouble();
    public @Random double nextGaussian();
    public @Random long nextLong();
    public @Random int nextInt();
}
```