Configuration

By Lex Li

Obfuscar accepts a single command line argument: the path to its configuration file.

In this article

The configuration file is used to specify what assemblies should be obfuscated, where to find the dependencies for the assemblies, and where the obfuscated assemblies should be saved.

The configuration file is an XML file with top-level element `Obfuscator`.

Settings

Variables are typically used to store settings. Recommended variable names to use are:

Name	Description
InPath	Directory containing the input assemblies, such as `c:\\in`.
OutPath	Directory to contain the obfuscated assemblies, such as `c:\\out`.

Name	Description
LogFile	Obfuscation log file path (mapping.txt).
XmlMapping	Whether the log file should be of XML format.
KeyFile	Key file path, such as `c:\folder\key.pfx` .
KeyContainer	Key container name.
RegenerateDebugInfo	Whether to generate debug symbols for obfuscated assemblies.
MarkedOnly	Whether to only obfuscate marked items. All items are obfuscated when s
RenameProperties	Whether to rename properties.
RenameEvents	Whether to rename events.
RenameFields	Whether to rename fields.
KeepPublicApi	Whether to exclude public types and type members from obfuscation.
HidePrivateApi	Whether to include private types and type members from obfuscation.
ReuseNames	Whether to reuse obfuscated names.
UseUnicodeNames	Whether to use Unicode characters as obfuscated names.
UseKoreanNames	Whether to use Korean characters as obfuscated names.
HideStrings	Whether to hide strings.
OptimizeMethods	Whether to optimize methods.
SuppressIldasm	Whether to include an attribute for ILDASM to indicate that assemblies ar
AnalyzeXaml	Whether to analyze XAML related metadata for obfuscation.

The default values can be found in source code .

Variables, InPath and OutPath

The following is an example of a minimal configuration. It is provided in the source code as part of the Basic Example:

```
<?xml version='1.0'?>
<Obfuscator>
  <Var name="InPath" value=".\Obfuscator_Input" />
  <Var name="OutPath" value=".\Obfuscator_Output" />

  <Module file="$(InPath)\BasicExampleExe.exe" />
  <Module file="$(InPath)\BasicExampleLibrary.dll" />
</Obfuscator>
```

In this sample, the two variables InPath and Output are defined using the `Var` element. The sample also specifies that two assemblies should be obfuscated: an executable and a dll.

Variables defined using the `Var` element will be expanded in strings following the definition. For example, when InPath is defined as:

```
<Var name="InPath" value=".\Obfuscator_Input" />
```

it can be used in a module:

```
<Module file="$(InPath)\BasicExampleExe.exe" />
```

A few special variables have additional effects:

- The variable `InPath` is used when resolving dependencies by searching the specified path. The default of `InPath` is the current working directory (".").
- The variable `OutPath` is used as the output path for the obfuscated assemblies and the log file specified in the variable `LogFile`. The default of `OutPath` is the current working directory (".").

Assembly Search Path (2.2.5+)

This setting specifies one additional directory to search for referenced assemblies. There can be multiple instances of this setting, which are searched sequentially.

```
<AssemblySearchPath path=".\Library\UnityAssemblies" />
<AssemblySearchPath path=".\Assets\SpriteSharp\Editor\3rdParty" />
```

KeepPublicApi and HidePrivateApi

A common case of assembly obfuscation is to obfuscate the names of private types and type members and keep public items. You can achieve this by:

```
<Var name="KeepPublicApi" value="true" />
<Var name="HidePrivateApi" value="true" />
```

Note

By using above you don't need to set any obfuscation attribute or rule.

This is the default behavior since version 2.2.0.

Another common case is to obfuscate all types and type members, which you can achieve using

```
<Var name="KeepPublicApi" value="false" />
<Var name="HidePrivateApi" value="true" />
```

Of course to obfuscate nothing you can use

```
<Var name="KeepPublicApi" value="true" />
<Var name="HidePrivateApi" value="false" />
```

The last combination obfuscates solely public types and type members:

```
<Var name="KeepPublicApi" value="false" />
<Var name="HidePrivateApi" value="false" />
```

It has little practical use, but was the default setting for version 2.1.*.

Modules

The assemblies to be obfuscated are listed one-by-one as a separate `Module` element. Assemblies referenced by an assembly specified by a `Module` element must be resolvable, either via Cecil's regular resolution process, via the path specified by InPath or via a directory listed as `AssemblySearchPath`.

Only assemblies specified in a `Module` element will be obfuscated. Resolved assemblies are not altered.

It is highly recommended that you list assemblies one by one, and you can write simple PowerShell scripts to iterate files in the folder and generate a list of `Module` tags.

Note

A more complex way of specifying assemblies with `Modules` element were added by Thomas Caudal. You can refer to this pull request.

Exclusion Rules by Configuration

It is possible to include additional elements within the Module elements to skip types (the `SkipTypes` element), methods (the `SkipMethod` element), fields (`SkipField`), properties (`SkipProperty`), and events (`SkipEvent`, of course). Methods can be excluded from string obfuscation by `SkipStringHiding`. Special types such as enumerations can be excluded by `SkipEnums`.

The `SkipNamespace` element specifies a namespace that should be skipped. All types, methods, fields, etc., within the namespace will be skipped.

The `SkipType` element specifies the name of the type to skip, including the full namespace. It can also specify whether to skip the method, fields, properties, and/or events within the type.

The `SkipMethod` element specifies the name of the type containing the method, a protection specifier, and a name or regex to match the method. The protection specifier is currently ignored, but will eventually be used for additional filtering.

The `SkipField` element specifies the name of the type containing the field, a protection specifier, and a name or regex to match the field. The protection specifier is currently ignored, but will eventually be used for additional filtering.

The `SkipProperty` element specifies the name of the type containing the property, a protection specifier, and a name or regex to match the property. The protection specifier is currently ignored, but will eventually be used for additional filtering.

The `SkipEvent` element specifies the name of the type containing the event, a protection specifier, and a name or regex to match the event. The protection specifier is currently ignored, but will eventually be used for additional filtering.

The `SkipStringHiding` element works like the `SkipMethod` element, but specifies within which methods not to obfuscate the string constants. To make it harder to analyze the code, Obfuscar normally replaces string loads by method calls to lookup functions, which incurs a small performance penalty.

A more complete example:

```xml
<Module file="$(InPath)\AssemblyX.exe">
  <!-- skip a namespace -->
  <SkipNamespace name="Company.PublicBits" />

  <!-- to skip a namespace recursively, just put * on the end -->
  <SkipNamespace name="Company.PublicBits*" />

  <!-- skip field by name -->
  <SkipField type="Full.Namespace.And.TypeName"
    attrib="public" name="Fieldname" />

  <!-- skip field by regex -->
  <SkipField type="Full.Namespace.And.TypeName"
    attrib="public" rx="Pub.*" />

  <!-- skip type...will still obfuscate its methods -->
  <SkipType name="Full.Namespace.And.TypeName2" />

  <!-- skip type...will skip its methods next -->
  <SkipType name="Full.Namespace.And.TypeName3" />
  <!-- skip TypeName3's public methods -->
  <SkipMethod type="Full.Namespace.And.TypeName3"
    attrib="public" rx=".*" />
  <!-- skip TypeName3's protected methods -->
  <SkipMethod type="Full.Namespace.And.TypeName3"
    attrib="family" rx=".*" />

  <!-- skip type and its methods -->
  <SkipType name="Full.Namespace.And.TypeName4" skipMethods="true" />
  <!-- skip type and its fields -->
  <SkipType name="Full.Namespace.And.TypeName4" skipFields="true" />
  <!-- skip type and its properties -->
  <SkipType name="Full.Namespace.And.TypeName4" skipProperties="true" />
  <!-- skip type and its events -->
  <SkipType name="Full.Namespace.And.TypeName4" skipEvents="true" />
  <!-- skip attributes can be combined (this will skip the methods and fields) -->
  <SkipType name="Full.Namespace.And.TypeName4" skipMethods="true" skipFields="true" />
  <!-- skip the hiding of strings in this type's methods -->
  <SkipType name="Full.Namespace.And.TypeName4" skipStringHiding="true" />

  <!-- skip a property in TypeName5 by name -->
  <SkipProperty type="Full.Namespace.And.TypeName5"
    name="Property2" />
  <!-- skip a property in TypeName5 by regex -->
  <SkipProperty type="Full.Namespace.And.TypeName5"
    attrib="public" rx="Something\d" />

  <!-- skip an event in TypeName5 by name -->
  <SkipProperty type="Full.Namespace.And.TypeName5"
    name="Event2" />
  <!-- skip an event in TypeName5 by regex -->
  <SkipProperty type="Full.Namespace.And.TypeName5"
    rx="Any.*" />

  <!-- avoid the hiding of strings in TypeName6 on all methods -->
  <SkipStringHiding type="Full.Namespace.And.TypeName6" name="*" />
</Module>
```

To prevent all properties from being obfuscated, set the RenameProperties variable to “false” (it’s an xsd boolean). To prevent specific properties from being renamed, use the `SkipProperty` element. It will also skip the property’s accessors, get_XXX and set_XXX.

To prevent all events from being obfuscated, set the RenameEvents variable to “false” (it’s also xsd boolean). To prevent specific events from being renamed, use the `SkipEvent` element. It will also skip the event’s accessors, add_XXX and remove_XXX.

Inclusion Rules by Configuration (new)

To supplement `Skip*` elements, `Force*` has been added.

Name Matching

The `SkipMethod`, `SkipProperty`, `SkipEvent`, `SkipField`, and `SkipStringHiding` elements accept an rx attribute that specifies a regular expression used to match the name of the thing to be skipped. The `SkipType`, `SkipMethod`, `SkipProperty`, `SkipEvent`, `SkipField`, and `SkipStringHiding` elements all accept a name attribute that specifies a string with optional wildcards or a regular expression used to match the name of the thing to be skipped. For elements where both the name and rx attributes are specified, the rx attribute is ignored.

The name attribute can specify either a string or a regular expression to match the name of the thing to be skipped. If the value of the name attribute begins with a ‘^’ character, the value (including the ‘^’) will be treated as a regular expression (e.g., the name ‘^so.*g’ will match the string something). Otherwise, the value will be used as a wildcard string, where ‘*’ matches zero or more characters, and ‘?’ matches a single character (e.g., the wildcard string som?t*g will match the string something).

This behavior also applies to the value of the type attribute of the `SkipMethod`, `SkipProperty`, `SkipEvent`, `SkipField`, and `SkipStringHiding` elements.

Accessibility Check

The `SkipMethod`, `SkipProperty`, `SkipEvent`, `SkipField`, and `SkipStringHiding` elements also accept an attrib attribute.

- Not specified or `attrib=''`: All members are skipped from obfuscation.
- `attrib='public'`: Only public members are skipped.
- `attrib='protected'`: Only public and protected members are skipped.
- All other values for attrib generate an error by now.

Members which are internal or protected internal are not skipped when attrib is public or protected.

Properties and events do not directly have an accessibility attribute, but their underlying methods (getter, setter, add, remove) have. For properties the attribute of the getter and for events the attribute of the add method is used.

Exclusion by Attributes in Code

There's also some functionality where you can mark types with an attribute to prevent them from being obfuscated.

System.Reflection.ObfuscationAttribute

Note

The Obfuscar attribute defined in Obfuscar itself is obsolete.

And if you only want specific classes obfuscated, you can set the MarkedOnly variable to "true" (also an xsd boolean), and apply the Obfuscation attribute to the things you want obfuscated. This is done in the ObfuscarTests project (included w/ the source…it's intended to be a place for unit tests, but for now does little) to obfuscate a subset of the classes. For example, if MarkedOnly is set to true, to include obfuscation of X, its methods, fields, resources, etc.

Inclusion/Exclusion Rule Priorities

Above several inclusion/exclusion methods have been documented. What if multiple rules apply to a single item? Which rule is executed while others ignored?

The rule of thumb is as below,

1. Attributes set on the item is always of top priority. If an attribute is detected, then all other rules are ignored. For members of a type, if the member itself does not contain such attributes, the type's attributes take effect.
2. If no attribute is set, inclusion rules (`Force*`) are of top priority.
3. If no inclusion rule is set, exclusion rules (`Skip*`) are of top priority.
4. If no exclusion rule is set, `KeepPublicApi` and `HidePrivateApi` take effect.

Control Generation of Obfuscated Names

By default all new type and member names generated by Obfuscar are only unique within their scopes. A type with name A may be part of namespace A.A and A.B. The same holds true for type members. Multiple types may have fields and properties with the same name.

When using `System.Xml.Serialization.XmlSerializer` on obfuscated types, the names of generated Xml elements and attributes have to be specified with one of the `XmlXXXXXAttribute` attributes. This is because the original type and member names do not exist any more after obfuscation. For some reasons the `XmlSerializer` uses the obfuscated names internally even though they are overridden by attributes. Because of that it fails on duplicate names. The same is true for the XML Serializer Generator Tool (Sgen.exe).

You can work around this problem by setting the ReuseNames variable to false. In this case the obfuscator does not reuse names for types, fields and properties. The generated names are unique over all assemblies. This setting does not apply to methods.

Add the following line to the configuration file to enable unique names:

```
<Var name="ReuseNames" value="false" />
```

You can use `UseUnicodeNames` and `UseKoreanNames` to further control the characters used in obfuscated names. Unicode characters are often not readable, while Korean characters look almost the same to most audience. They provide good alternatives if you think the default characters do not provide the strongest obfuscation.

Control String Hiding

By default Obfuscar hides all string constants by replacing the string load (`LDSTR` opcode) by calls to methods which return the string from a buffer. This buffer is allocated on startup (in a static constructor) by reading from a XOR-encoded UTF8 byte array containing all strings. This comes with a small performance cost. You can disable this feature completely by adding the following line to the configuration file:

```
<Var name="HideStrings" value="false" />
```

If you only want to disable it on specific methods, use the SkipStringHiding elements.

Important

This feature hides the strings in a reversible way so that your code can remain valid, which means a de-obfuscation tool can reverse the string contents easily. Therefore, never store confidential information as strings in your assemblies, because this feature won’t protect them from being read.

SuppressIldasm Attribute

Microsoft designed an attribute `SuppressIldasmAttribute`, which if set on an .NET assembly can indicate that ILDASM utility from Microsoft should not display IL of the assembly.

Important

Obfuscar inserts this attribute if you enable this feature. However, decompilers (ILSpy, .NET Reflector, JustDecompile, or dotPeek) do not honor this attribute at all. Thus, practically speaking, it is a useless feature designed by Microsoft.

Signing of Strongly Named Assemblies

Signed assemblies will not work after obfuscation and must be re-signed.

Add the following line to the configuration file to specify the path to your key file. When given a KeyFile in the configuration, Obfuscar will sign a previously signed assembly with the given key. Relative paths are searched from the current directory and, if not found, from the directory containing the particular assembly.

```
<Var name="KeyFile" value="key.snk" />
```

If the project uses a .pfx file to sign the assembly, by default Visual Studio would create a key container in Windows, whose name can be found from MSBuild diagnostic logging.

Note

Once MSBuild diagnostic logging is enabled via `/v:diag` switch, the key container name can be found by searching for `KeyContainerName=VS_KEY_XXXXXX` in the output.

The key container name can then be used in Obfuscar configuration,

```
<Var name="KeyContainer" value="VS_KEY_XXXXXX" />
```

Important

KeyContainer setting is supported in release 2.2.13 and above.

If neither KeyFile nor KeyContainer is specified, Obfuscar normally throws an exception on signed assemblies. If an assembly is marked delay signed, the signing step will be skipped in case no key file is given.

Note

With the special key file name auto, Obfuscar uses the value of the AssemblyKeyFileAttribute instead (if existing).

Configuration Fragments (2.2.5+)

Configuration can now be split into multiple files.

Usage example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Obfuscator>
  <Var name="InPath" value="..\..\Input" />
  <Var name="OutPath" value="..\..\Output" />
  <Var name="KeepPublicApi" value="false" />
  <Var name="HidePrivateApi" value="true" />
  <Include path="$(InPath)\TestInclude.xml" />
  <Module file="$(InPath)\AssemblyWithCustomAttr.dll">
      <Include path="$(InPath)\TestIncludeModule.xml" />
  </Module>
</Obfuscator>
```

TestInclude.xml:

```
<?xml version='1.0'?>
<Include>
  <Var name='TestIncludeVar' value='Foo' />
</Include>
```

TestIncludeModule.xml:

```
<?xml version='1.0'?>
<Include>
  <SkipMethod type='SkipVirtualMethodTest.Interface1' name='Method1' />
</Include>
```

Related Resources

- What does Obfuscar do?
- Basic Example