Obfuscar Documentation

Release 2.2

Ryan Williams, Calvin Rien, Lex Li, RemObjects Software, and oth

Oct 05, 2017

Contents

1 General **3**
- 1.1 Getting Started . . . 4
- 1.2 Tutorials . . . 16
- 1.3 Support . . . 17
- 1.4 Contribute . . . 18

2 Contribute to Documentation **25**

Attention: Obfuscar 2.2.9 is now available on NuGet! Please read the *Getting Started* instructions for installing the latest version from NuGet.

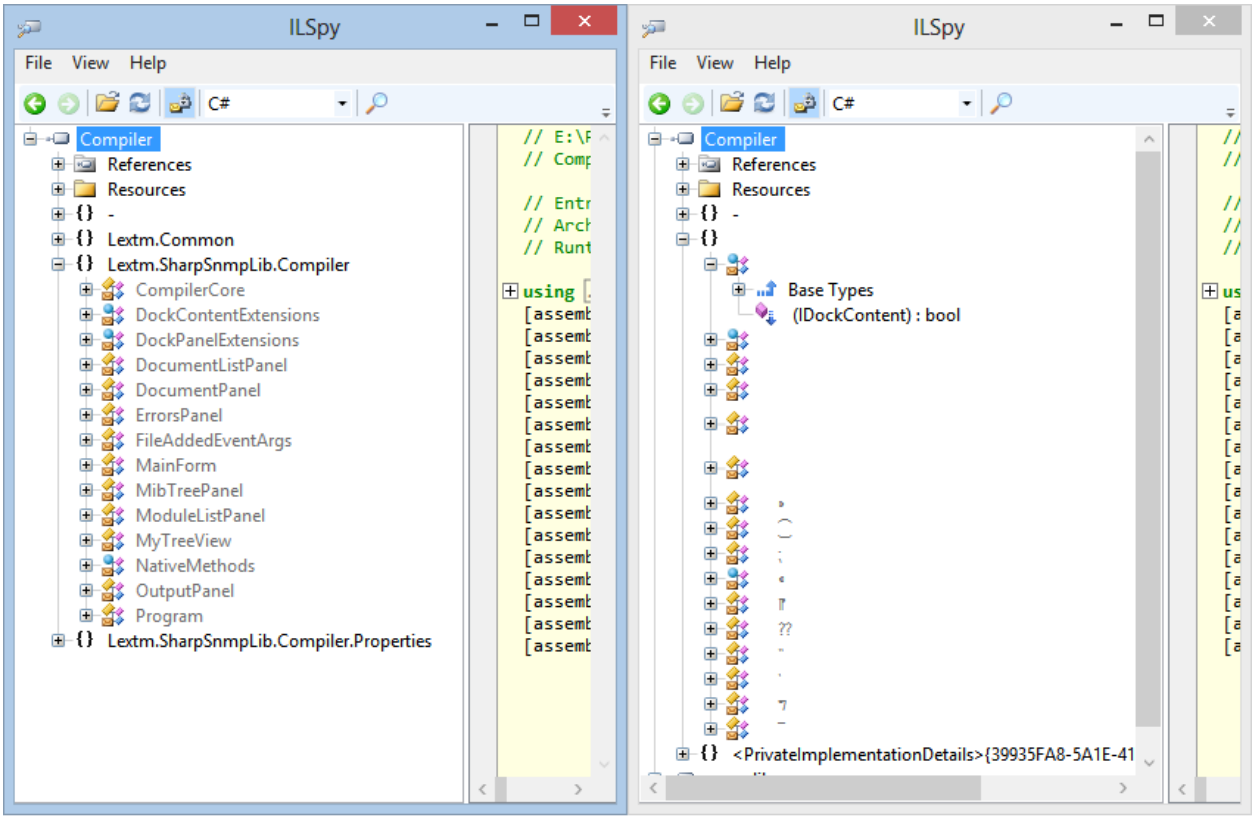
CHAPTER 1

General

Obfuscar is a basic obfuscator for .NET assemblies. It uses massive overloading to rename metadata in .NET assemblies (including the names of methods, properties, events, fields, types and namespaces) to a minimal set, distinguishable in most cases only by signature.

For example, if a class contains only methods that accept different parameters, they can all be renamed 'A'. If another method is added to the class that accepts the same parameters as an existing method, it could be named 'a'.

It makes decompiled code very difficult to follow. The wiki has more details about What It Does.



The current release is Obfuscar 2.x. This is a port of Obfuscar 1.5.4 to the latest Mono.Cecil library. By using this new library Obfuscar now supports .NET 4.0 assemblies. Because there are a lot of subtle changes in Cecil's new API and patched merged from different sources, this release of Obfuscar must be considered unstable.

Note: Since version 1.5 the attrib attribute is evaluated correctly. Be sure to check if there are any unintended attrib values from the example in your configuration file.

Its source code can be found at GitHub,

https://github.com/lextm/obfuscar

Obfuscar works its magic with the help of Jb Evain's fantastic Mono Cecil library.

The documentation section includes the following topics,

Getting Started

What does Obfuscar do?

By Lex Li

Basically, Obfuscar scrambles the metadata in a set of assemblies. It renames everything to the minimal set of names that can be used to identify them, given signatures and type information. Since these new names are shorter than the old ones, it also dramatically shrinks executable size.

In this article:

- *An Example*
- *Caveat*
- *Related Resources*

An Example

The following method is from the example included in the release:

```
public ExampleUI( )
{
        InitializeComponent( );

        ClassX cx = new ClassX( "Some Text" );

        displayText.Text = cx.DisplayText;
}
```

The code can be decompiled (via ILSpy) to:

```
public ExampleUI()
{
        this.InitializeComponent();
        this.displayText.Text = new ClassX("Some Text").get_DisplayText();
}
```

After obfuscation, the code can be decompiled (via ILSpy) to:

```
public A()
{
        this.A();
        this.a.Text = new A.A("Some Text").A();
}
```

It's a simple example, but it scales...For example, given a reasonably sized code base, one could easily run into a class named A (in the namespace A) with 7 methods, 4 properties, and 5 fields named A, with several more methods, properties, and fields named a.

To try it out, see Basic Example.

Caveat

It makes debugging / reverse engineering very difficult, but wouldn't stop someone who really wants to reverse engineer it. It would at least slow them down, and would deter casual observers.

Deobfuscators such as de4dot make it easy to reverse most protection a commercial obfuscator might set. But note that the names obfuscated by Obfuscar still remains useful, as what's lost cannot be restored.

Related Resources

- *Project History*

- *Basic Example*

Configuration

By Lex Li

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

In this article

- *Table of Settings*
- *Variables, InPath and OutPath*
- *Assembly Search Path (2.2.5+)*
- *KeepPublicApi and HidePrivateApi*
- *Modules*
- *Exclusion Rules by Configuration*
- *Name Matching*
- *Accessibility Check*
- *Exclusion by Attributes in Code*
- *Inclusion/Exclusion Rule Priorities*
- *Control Generation of Obfuscated Names*
- *Control Hiding of Strings*
- *Signing of Strongly Named Assemblies*
- *Configuration Fragments (2.2.5+)*
- *Related Resources*

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.

Table of Settings

Name	Description
InPath	Input file location
OutPath	Output file location
LogFile	Obfuscation log file path (mapping.txt)
XmlMapping	Whether the log file should be of XML format
KeyFile	Key file path
RegenerateDebugInfo	Whether to generate debug symbols for obfuscated assemblies
MarkedOnly	Whether only to obfuscate marked items
RenameProperties	Whether to rename properties
RenameEvents	Whether to rename events
RenameFields	Whether to rename fields (2.2.0+)
KeepPublicApi	Whether to keep public API out of obfuscation
HidePrivateApi	Whether to hide private API via 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 inform ILdism that assemblies are obfuscated

Variables, InPath and OutPath

The following is a is an example of a minimal configuration. It is provided in the release 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 the example configuration, two variables are defined, InPath and OutPath, using the Var element, and two assemblies are listed for obfuscation, an executable and a dll.

Variables defined using the Var element will be expanded in strings following the definition...After defining InPath as follows:

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

It can be used in another location:

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

In addition to being usable like macros, there are a few special variables that have additional effects. The variable InPath is used when trying to find dependencies (the specified path is searched), and the variable OutPath is used as the output path for the obfuscated assemblies and the map. If either InPath or OutPath is unspecified, they default to the current path (".").

Assembly Search Path (2.2.5+)

This setting specifies an additional place to search for references. There can be multiple instances of this tag.

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

KeepPublicApi and HidePrivateApi

A common case of assembly obfuscation is to strip out private information and keep public items. This can be achieved by setting the following combination,

```
<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 setting since 2.2.0.

Another common case is to strip out everything, which can be achieved by setting

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

Of course to keep everything we can use

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

The last combination is which strips out public information only,

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

It should be rarely used, but was the default for releases such as 2.1.*.

Modules

For each assembly to be obfuscated, there must be a Module element. Assemblies referenced by an assembly specified by a Module element must be resolvable, either via Cecil's regular resolution process, or they must be present in the path specified by InPath.

Though additional assemblies are loaded for examination, only the specified assemblies will be obfuscated.

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:

```
<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" />
```

Control Hiding of Strings

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.

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 no KeyFile 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.

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*

Frequent Asked Questions

By Lex Li

This page shows you the common questions.

In this article

- *How to troubleshoot TypeLoadException during reflection?*
- *How to troubleshoot TypeLoadException if a method does not have an implementation?*
- *How to analyze exceptioni call stack if obfuscated?*
- *Related Resources*

How to troubleshoot TypeLoadException during reflection?

After obfuscation, the application might not work and `TypeLoadException` is quite common a case. One such case is

http://stackoverflow.com/questions/24058302/obfuscar-2-0-could-not-load-type-from-assembly

There can be other similar cases, where either explicitly or implicitly the application code itself requires an instance to be initialized at runtime by reflection. Since reflection requires metadata, which obfuscation manipulates heavily, such initialization could fail.

The workaround is to skip such types or methods in obfuscation, so that reflection can still find them out.

Note: It is rarely a bug of Obfuscar.

How to troubleshoot TypeLoadException if a method does not have an implementation?

One such case is

https://github.com/lextm/obfuscar/issues/47

Note: It is very likely a bug of Obfuscar.

Obfuscar can mistakenly rename a virtual function, so that at runtime CLR cannot find the expected method from the type.

The workaround is to skip such methods in obfuscation explicitly. A bug report can also be fired at GitHub.

How to analyze exceptioni call stack if obfuscated?

Obfuscation replaces class and method names so that exception call stacks would be difficult to read. But there is a separate open source project called ObfuscarMappingParser to address the challenge.

Related Resources

- *What does Obfuscar do?*
- *Basic Example*

Project History

By Lex Li

The popularity of managed programming languages and their ease of decompilation introduce a common issue, "How can I protect my binaries?"[1] . Obfuscation has been invented to add some protection. But most of obfuscator products are commercial and really expensive.

In this article:

- *Started by Ryan*
- *Resumed by Werner*
- *Various Forks After 2010*
- *Inherited by Lex Li*
- *Related Resources*

Started by Ryan

Ryan Williams (@drcforbin) announced the first release of an open source obfuscator named obfuscar on May 3, 2007[2] , while the initial checkin on Google Code occurred in late April[3] .

It started to gain popularity and soon the following releases were published,

- 1.0.1 on May 10, 2007[4]
- 1.1.0 on May 18, 2007[5]
- 1.3.0 on Jul 12, 2007[6]
- 1.3.1 on Sep 12, 2007[7]

[1] http://en.wikipedia.org/wiki/Obfuscation_(software)
[2] https://groups.google.com/forum/#!topic/obfuscar/2QL0ObML40o
[3] https://code.google.com/p/obfuscar/source/list?num=25&start=10
[4] https://groups.google.com/forum/#!topic/obfuscar/ctAN-hUaIB8
[5] https://groups.google.com/forum/#!topic/obfuscar/SrPG1BDZgrM
[6] https://groups.google.com/forum/#!topic/obfuscar/aH6I8v9Q_uI
[7] https://groups.google.com/forum/#!topic/obfuscar/554b9QOYsx4

Resumed by Werner

Werner (@webbie) first contributed an important patch to the project on June 19, 2007[8] . He seemed to be taking care of the project since April 2009, and made the following releases,

- 1.4.0 on Apr 24, 2009[9]
- 1.5.0 on Nov 06, 2009[10]
- 1.5.2 on Nov 20, 2009[11]
- 1.5.4 on ??

He was trying to upgrade the project to work with latest Cecil at that moment under 2.0 branch, but that work was not finished.

Przemysław Rajpold(@rejpon) contributed two patches on July 29, 2010[12][13] . @mikael.hansen.idevio also contributed one patch on May 26, 2009.

Various Forks After 2010

Carlo Kok from RemObjects has participated in the project since around Oct 8, 2009[14] . RemObjects releases a commercial obfuscator called Oxfuscator in early 2010, which is based on Obfuscar[15] . However, RemObjects still keeps an open source version available[16] .

Calvin Rien (@darktable) created a fork on BitBucket on Oct 24, 2012 and included his changes[17] .

Emil Johansen (@AngryAnt) forked Calvin's fork, and hosted it on GitHub[18] .

Inherited by Lex Li

Lex Li was working on his commercial project #SNMP Pro in Apr 2013, and came across Emil's fork. He soon created a new fork and started to merge various patches he could find from the Internet to this[19] .

Gradually Lex gained knowledge about Cecil and Obfuscar and started to heavily change the obfuscation process so as to better serve his own needs.

Carlo from RemObjects also contributes to this new fork, since Jan 2014. The collaboration aims to keep the two forks in sync in all possible ways, though due to the varied needs and design goals of two parties the forks still present significant differences.

The project homepage is now switched, and the latest release is 2.*.

Related Resources

- *What does Obfuscar do?*

[8] https://groups.google.com/forum/#!topic/obfuscar/__vjzvP9m1o
[9] https://groups.google.com/forum/#!topic/obfuscar/TLyU9ybfsKg
[10] https://groups.google.com/forum/#!topic/obfuscar/o61if_nFeog
[11] https://groups.google.com/forum/#!topic/obfuscar/vsVCfSMYk3U
[12] https://groups.google.com/forum/#!topic/obfuscar/PYs0KNps8mk
[13] https://groups.google.com/forum/#!topic/obfuscar/2LY3WoU72IQ
[14] https://groups.google.com/forum/#!topic/obfuscar/Fy0wFMVpA9Y
[15] http://blogs.remobjects.com/blogs/mh/2010/01/12/p303
[16] http://code.remobjects.com/p/obfuscar
[17] https://bitbucket.org/darktable/obfuscar/wiki/Home
[18] https://github.com/AngryAnt/obfuscar
[19] https://github.com/lextm/obfuscar

- *Basic Example*

Tutorials

Basic Example

By Lex Li

This page shows you the basics about Obfuscar.

In this article:

- *Folder Structure*
- *Technical Details*
- *Related Resources*

Folder Structure

The example has been moved to a separate repository at GitHub .

Technical Details

Note: The example assumes you have Visual Studio 2017 installed.

The example consists of a solution that includes and executable and a dll. The test.bat script builds the solution, copies the output to a temporary input path. The files are then obfuscated into a temporary output path. After obfuscation, the files are copied along with the map file to their final output path.

The reason for this indirection is safety. The release process should be as follows: 1. Files that will not be obfuscated should be copied directly to the final output path. 1. Files to be obfuscated should be copied to a temporary location (Obfuscator_Input, in the example), along with their dependencies. 1. Files to be obfuscated should be obfuscated from the first location (Obfuscator_Input) to a different location (Obfuscator_Output, in the example). 1. Obfuscated files should be copied from the obfuscation output location (Obfuscator_Output) to the final output path. 1. Package files in final output path (beyond the scope of this document).

In the event that the obfuscator fails, an in-place obfuscation could allow files to enter the release without being obfuscated. Having the obfuscator explicitly save to a different location and using files from that location means that failure results in missing files.

Related Resources

- *What does Obfuscar do?*

Support

Frequent Asked Questions

By Lex Li

This page shows you where to find the answers.

In this article:

- *How to troubleshoot TypeLoadException during reflection?*
- *How to troubleshoot TypeLoadException if a method does not have an implementation?*
- *GitHub FAQ List*
- *StackOverflow*
- *Related Resources*

How to troubleshoot TypeLoadException during reflection?

After obfuscation, the application might not work and TypeLoadException is quite common a case. One such case is

http://stackoverflow.com/questions/24058302/obfuscar-2-0-could-not-load-type-from-assembly

There can be other similar cases, where either explicitly or implicitly the application code itself requires an instance to be initialized at runtime by reflection. Since reflection requires metadata, which obfuscation manipulates heavily, such initialization could fail.

The workaround is to skip such types or methods in obfuscation, so that reflection can still find them out.

Note: It is rarely a bug of Obfuscar.

How to troubleshoot TypeLoadException if a method does not have an implementation?

One such case is

https://github.com/lextm/obfuscar/issues/47

It is very likely a bug of Obfuscar, where it mistakenly renames a virtual function, so that at runtime CLR cannot find the expected method from the type.

The workaround is to skip such methods in obfuscation explicitly. A bug report can also be fired at GitHub.

GitHub FAQ List

We use GitHub to manage the list.

StackOverflow

You might also want to check out existing discussions at StackOverflow .

Related Resources

- *What does Obfuscar do?*
- *Project History*
- *Basic Example*
- *Support Services*

Support Services

By Lex Li

This page shows you where to find services.

In this article:

- *Community Support*
- *Commercial Support*
- *Related Resources*

Community Support

You can reach the Obfuscar user community by posting to StackOverflow .

Commercial Support

Commercial support is provided by LeXtudio, and its model is similar to #SNMP Pro .

Related Resources

- *What does Obfuscar do?*
- *Project History*
- *Basic Example*
- *Frequent Asked Questions*

Contribute

Note: We reuse the ASP.NET docs style guide.

Contributors

By Lex Li

This page shows you the contributors.

In this article:

- *Previous Contributors*
- *Related Resources*

Previous Contributors

Contributors to this project include,

- Ryan Williams (drcforbin@gmail.com)
- Werner (webbie@cablemail.de)
- Przemysław Rajpold (@rejpon in Google Group)
- Carlo Kok (@carlokok on Twitter/GitHub)
- Calvin Rien (@darktable on Twitter/GitHub)
- Emil Johansen (@AngryAnt on Twitter/GitHub)
- Lex Li (@lextm on Twitter/GitHub)

Related Resources

- *What does Obfuscar do?*
- *Basic Example*

GitHub Guide

By Lex Li

This page shows you how to use Obfuscar repo on GitHub to collaborate.

In this article:

- *Submitting a Patch*
- *Reviewing a Patch*
- *Reporting a Bug*
- *Reviewing a Bug*
- *Asking a Question*
- *Answering a Question*
- *Overview of Issue Tags*

- *Suggestions on Creating Pull Requests*
- *Suggestions on Reviewing Pull Requests*
- *Related Resources*

Submitting a Patch

An issue might be opened to discuss with the maintainers before creating the patch. This helps the maintainers track your progress, and provide guidance if needed.

1. Learn about GitHub via http://help.github.com/.
2. Create your own fork from the main repo at https://github.com/lextm/obfuscar.
3. Create a new branch with a meaningful name.
4. Make the changes on this new branch in your fork, and test it fully.
5. Create a pull request back (link the new branch in your fork to the master branch of main repo).
6. Document all your changes in details as part of the pull request, which is critical to better communicate with maintainers.

Patches will be reviewed and merged as early as possible.

Note: If the patch is large, it might take several weeks to review and merge.

Note: To see what makes a good pull request, please follow *Suggestions on Creating Pull Requests* section.

Reviewing a Patch

Maintainers should respond to new pull requests as early as possible by commenting like this,

- "Acknowledged. Will start review."

which gives the contributors a hint that the process has begun.

Further responses can be like,

- "Comments have been left at **. Please revise your patch like this,"
- "Reviewed. ** will be merged, while ** will not. The reasons are,"

which will keep all discussions public to reveal all necessary technical information for future reference.

Reporting a Bug

Note: If you already have a patch for the bug, please follow *Submitting a Patch* section.

Note: If you are not sure whether it is a bug, please follow *Asking a Question* section.

1. Learn about GitHub via http://help.github.com/.
2. Review existing issues at https://github.com/lextm/obfuscar/issues that are marked as bug or enhancement to avoid duplicate.
3. Create a new issue on https://github.com/lextm/obfuscar/issues and provide all information in details.

You are welcome to provide step by step instructions, as that can help other reproduce and investigate the issue. If you are willing to share a sample project, please use a service such as DropBox or OneDrive.

Reviewing a Bug

Maintainers should respond to bug reports as early as possible by commenting like this,

- "Acknowledged. Will start review."

which gives the contributors a hint that the process has begun.

Further responses can be like,

- "Could not reproduce it. Please provide more information to assist investigation such as **,"
- "Reviewed. ** is a bug that can be reproduced. Will perform further investigation on how to resolve it. This may take a long time."

which will keep all discussions public to reveal all necessary technical information for future reference.

Asking a Question

Asking a Question on StackOverflow (Recommended)

1. Log into StackOverflow.
2. Before starting asking a new question, please review all questions under tag `obfuscar` in case yours has already been answered.
3. Click Ask Question to create a new question.
4. Add tag `obfuscar` to this question, and also include all information in details.

Then you can wait till users reply to your question.

Asking a Question on GitHub

1. Learn about GitHub via http://help.github.com/.
2. Before creating the issue, please review all existing issues especially our FAQ in case the issue has already been reported and resolved.
3. Create a new issue on https://github.com/lextm/obfuscar/issues and provide all information in details.

Answering a Question

Maintainers might join StackOverflow and monitor discussions under `obfuscar` tag.

Maintainers should respond to questions on GitHub as early as possible by commenting like this,

- "Acknowledged. Will start review."

which gives the contributors a hint that the process has begun.

Further responses can be like,

- "Could not reproduce it. Please provide more information to assist investigation such as **,"
- "Reviewed. ** is a bug that can be reproduced. Will perform further investigation on how to resolve it. This may take a long time."

which will keep all discussions public to reveal all necessary technical information for future reference.

Tag such an issue with question tag.

Close such issues once a meaningful answer is given.

Mark an issue as `faq candidate` if it should be considered as an FAQ.

Overview of Issue Tags

Maintainers should use the tags as early as possible so as to help each other to easily track the progress. The decoration tags are most useful for items which are not yet assigned to milestones.

Tags for Item Categories

The following are used to assign an item to a specific category,

- bug This item was reported as a bug of this product. The reporter expects a fix.
- enhancement This item was reported as an enhancement request. The reporter expects a certain feature to be enhanced or a new feature to be implemented.
- task This item was reported as a task. The reporter expects a maintainer to perform a piece of work (usually not development).
- idea This item was reported as a new idea. The reporter expects some discussion on a feature request. Once discussed, this item might be upgraded to an enhancement.
- question This item was reported as a question. The reporter expects some discussion on a problem met about this product. Once discussed, this item might be upgraded to a bug, an enhancement, or an idea.
- tech debt This item was reported as bad smells detected in the code base. The reporter expects changes in the code base to remove the bad smells.
- pull request This item was used to handle a pull request.

Tags for Decoration

The following are used to decorate an item so as to make it easy to see its status and required actions,

- dependency bug This only applies to bug items. It means the bug was caused by a bug of one of the dependencies (such as bugs of .NET Framework/Mono bugs, or bugs of the operating systems).
- not an issue This means after discussion, there is nothing to be done further (usually for false positives).
- wontfix This means the item (usually bugs) won't be fixed due to a strong justification. An agreement must be achieved among the maintainers.
- duplicate This means the item is exactly the same as another existing item. The maintainers should explicitly point out which item will be the focus and mark all the rest as duplicate.

- tentative This means based on the provided information it is not likely to move on. The reporter should provide more information and drive the discussion.
- soon to close This means there is little left to do on the item. The maintainers are going to close the item after a few more days (usually applied to tentative and cannot reproduce items).
- cannot reproduce This means the maintainers failed to reproduce the symptoms described in a bug report. The reporter should provide more information (process dumps, sample projects, screen shots, video clips and so on) and drive the investigation.
- in progress This means the item has been actively investigated by the maintainers.
- up for grabs This means community contribution is welcome.

Suggestions on Creating Pull Requests

All pull requests are appreciated (even if some we cannot merge). The following can make the pull requests simpler for reviewers, so hope you can follow them.

- If possible, send multiple pull requests for individual tasks and avoid a pull request for multiple tasks. Properly isolating changes to meaningful batches makes it quicker to analyze and assert the changes.
- Fork and create a new branch with a meaningful name first before making the changes.
- Squash all commits on this new branch to only one or two before sending the pull request.
- Wait for comments from the reviewers. It usually takes weeks as the reviewers might not be able to finish quickly. Don't make further changes at this stage to avoid changes of this pull request.
- Revise the code based on feedbacks, and then make a second commit with necessary changes and push to the branch in your fork, where GitHub automatically appends it to the pull request for further review.

Then the reviewers will decide whether to accept or reject the pull request based on code quality.

One important notice is that some pull requests might not be accepted, but they are still valuable to the community,

- It contains a nice-to-have feature (such as options to enable/disable part of a theme, or a visual element) for some users but not all.
- It introduces a feature (such as new visual elements) that goes beyond Visual Studio look and feel.

Such pull requests are of great value of course. But since the primary goal of DPS is to simulate Visual Studio look and feel, and the code base is already huge to maintain, we try to avoid bringing in non-core features.

Suggestions on Reviewing Pull Requests

Please leave a message that you are going to review a pull request. That should let the submitter know it's been reviewed.

Leave all comments at a time, so that the submitter can revise them altogether to form a new commit.

Decide carefully whether to accept or reject a pull request. Leave explanation for future reference.

Related Resources

- /getting-started/installing-on-windows
- *Basic Example*
- /themes/existing-themes

CHAPTER 2

Contribute to Documentation

The documentation on this site is the handiwork of our many contributors.

We accept pull requests! But you're more likely to have yours accepted if you follow these guidelines:

Read the Contributing Guide.