Obfuscat ing your code with SmartAssembly

SmartAssembly - 6.2
Why should I obfuscate my code?

The benefits of obfuscation

Obfuscating your code with Red Gate's SmartAssembly can help to protect you against intellectual property theft by making your code hard to read.

This paper discusses the different forms of obfuscation offered by SmartAssembly and when to use them.
Obfuscating type and method names

SmartAssembly offers three levels of obfuscation for type names and method names. These are:

<table>
<thead>
<tr>
<th>Obfuscation Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ASCII characters</strong></td>
<td>Renames types and methods using ASCII characters. For example, the type <code>FullScreenSplash</code> might be obfuscated to <code>#dc</code>. Note: You must choose this option if you want to be able to decode stack tracks after obfuscation.</td>
</tr>
<tr>
<td><strong>Unicode unprintable characters</strong></td>
<td>Renames types and methods using Unicode unprintable characters. For example, the type <code>FullScreenSplash</code> might be obfuscated to <code>U+1D11E</code> (which is unprintable).</td>
</tr>
<tr>
<td><strong>Unicode unprintable characters and advanced renaming algorithm</strong></td>
<td>Renames types and methods using Unicode unprintable characters and renames multiple items with the same name. For example, the types <code>FullScreenSplash</code> and <code>FontStyle</code> are both obfuscated to <code>U+017D</code>.</td>
</tr>
</tbody>
</table>

Obfuscating field names

SmartAssembly also offers three different schemes for renaming field names. These are:

<table>
<thead>
<tr>
<th>Renaming Scheme</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>One-to-one renaming scheme</strong></td>
<td>Changes all fields in all classes so that they have different names. A field's name is unique within an assembly. Note: You must choose this option if you want to be able to retrieve the original field names from stack traces.</td>
</tr>
<tr>
<td><strong>Standard renaming</strong></td>
<td>Alters the field name so that it is unique within a class, however the same name is often used within other classes in the assembly. For example, <code>#a: string, #b: boolean, #c: string, #d: boolean</code>.</td>
</tr>
<tr>
<td><strong>Advanced renaming</strong></td>
<td>Alters the field name, however it not unique within a class and is reused. Fields of a different type will have the same name. For example, <code>#a: string, #a: boolean, #b: string, #b: boolean</code>.</td>
</tr>
</tbody>
</table>
Regardless of the renaming scheme chosen, the field name is also obfuscated in the same style as the selected types/methods name obfuscation (ASCII or Unicode unprintable characters).

Understanding name obfuscation: example

To illustrate the effect of using obfuscation options, this example uses the following code sample, part of the C# source code for a program called QueryBee (a simple WinForms application which can query SQL servers).

Code before obfuscation

```csharp
namespace QueryBee
{
    public class ConnectForm : Form
    {
        // Fields
        private IContainer components;
        private EventHandler Foregrounded;
        private ComboBox m_ComboDatabase;
        private ComboBox m_comboServer;
        private Button m_connectButton;
        private ProgressBar m_connectProgress;
        private SystemHotkey m_hotkey;
        private Label m_labelDatabase;
        private Label m_labelError;
        private Label m_labelServer;
        private NotifyIcon m_trayIcon;
        private ContextMenuStrip m_traymenu;
        private ToolStripMenuItem m_trayMenuClose;
        private ToolStripMenuItem m_trayMenuOpen;

        // Events
        public event EventHandler Foregrounded;

        // Methods
        public ConnectForm();
        private void connectForm_Resize(object sender, EventArgs e);
    }
}
```
protected override void Dispose(bool disposing);
private void InitializeComponent();
private void m_comboDatabase_clickIn(object sender, EventArgs e);
private void m_ComboDatabase_TextChanged(object sender, EventArgs e);
private void m_connectButton_Click(object sender, EventArgs e);
private void m_trayIcon_DoubleClick(object sender, EventArgs e);
private void m_trayMenuClose_Click(object sender, EventArgs e);
private void m_trayMenuOpen_Click(object sender, EventArgs e);
protected override bool ProcessCmdKey(ref Message msg, Keys keyData);

The code above is obfuscated using SmartAssembly's most basic level of obfuscation: ASCII characters and one-to-one field renaming. To show the effects of this obfuscation on the code, Red Gate’s .NET Reflector (a tool which can disassemble .NET executables) can be used to view the result.

Code after ASCII character obfuscation
	namespace #W8z
    {
        internal class #V8z : Form
        {
            // Fields
            private EventHandler #58z;
            private ComboBox #68z;
            private Label #78z;
            private ComboBox #88z;
            private Label #98z;
            private NotifyIcon #a9z;
            private Button #b9z;
            private ProgressBar #c9z;
            private #Z8z #d9z;
            private ContextMenuStrip #e9z;
            private ToolStripMenuItem #f9z;
            private ToolStripMenuItem #g9z;
            private Label #h9z;
            private IContainer #vyc;
By changing the names of these methods (from 'ComboDatabase' to '#68z', for example), it is already much harder to understand the function of this code.

Note: in the last two lines 'disposing' and 'keyData' have not been obfuscated. This is because SmartAssembly detects when it would be unsafe to obfuscate a name, and leaves names that would break your code if changed.

In the example above, the field names ('#58z' and so on) are unique to the class '#V8z', however, and so with some effort it may be possible to work out their functionality. To avoid this, SmartAssembly's standard field renaming function can be applied to reuse the same field names within multiple classes in the assembly.

Code after standard field renaming

namespace #W8z
{
    internal class #V8z : Form
    {
        // Fields
        private EventHandler #a;
        private IContainer #b;
        private ComboBox #c;
    }
}
The level of obfuscation can be improved still further by replacing the ASCII characters in the names above with unprintable Unicode characters (and symbols). Most names are not displayed at all.

**Code after obfuscation with Unicode unprintable characters**

```csharp
namespace ¨°
{
    internal class ¨° : Form
    {
```

© Red Gate Software Ltd
19 October 2011
// Fields
private EventHandler;
private IContainer;
private ComboBox;
private Label;
private ComboBox;
private Label;
private NotifyIcon;
private Button;
private ProgressBar;
private »;
private ContextMenuStrip;
private ToolStripMenuItem;
private ToolStripMenuItem;
private Label;

// Events
public event EventHandler Foregrounded;

// Methods
public — ();
private void () ;
private void Z,(object sender, EventArgs e);
private void Ž,(object sender, EventArgs e);
private void Ž,(object sender, EventArgs e);
private void Ž,(object sender, EventArgs e);
private void Ž,(object sender, EventArgs e);
private void Ž,(object sender, EventArgs e);
private void Ž,(object sender, EventArgs e);
private void Ž,(object sender, EventArgs e);
protected override void Dispose(bool disposing);

Finally, advanced field renaming uses identical names for different fields of different types within the same class. In the example below (which uses ASCII names), the three fields of type 'label' have different names but many fields of different types share the name '#a'.

By combining Unicode unprintable character obfuscation with advanced field renaming, you can greatly reduce the risk of anyone understanding your source code, even after disassembly. The result of this level of obfuscation is not shown here because it would be impossible to show that the same unprintable characters are used for multiple field names.
Software often stores passwords, login information, licensing information and/or SQL requests in strings. SmartAssembly can encode strings in an assembly to protect them. Choosing this option may affect the performance of the application.

To enable strings encoding, select the appropriate options in the strings encoding section of the main project settings window in SmartAssembly.

Notes:

- You cannot use 'strings encoding with improved protection' with .NET 1.1, or with any version of Microsoft Compact Framework, Silverlight or XNA.
- You cannot use 'compress and encrypt the encoded strings' with .NET 1.1, Microsoft Compact Framework version 1, Silverlight or XNA.
In addition to renaming crucial parts of your code, you can also use SmartAssembly to make your code more difficult to read by making it unnecessarily complex (converting it into ‘spaghetti code’).

Control flow obfuscation prevents most programs from being able to disassemble the code. For example, attempting to disassemble control-flow-obfuscated code using Red Gate’s .NET Reflector will give:

```csharp
private static void Main()
{
    // This item is obfuscated and can not be translated.
}
```

Note: Using control flow obfuscation can impact the performance of your application; there are three levels of control flow obfuscation in SmartAssembly, which which allow you to balance performance overhead against increased complexity.

To enable control flow obfuscation, select the option in the control flow obfuscation section of the main project settings window in SmartAssembly.

Note: Control flow obfuscation does not work with any version of Microsoft Compact Framework or XNA.
Using a dynamic proxy for external references

If you make calls outside of your assembly, these calls will still be visible in your code after string encoding and control flow obfuscation. For example, if you make a call to a message box, it would still be possible to find that call in the disassembled application, even with string encoding and control flow obfuscation enabled. By following external calls like this, it may still be possible for people to understand parts of your code.

To solve this, you can use SmartAssembly to create a proxy (the 'references dynamic proxy') at runtime. This proxy makes it harder to understand your code because all external calls appear to be sent to the proxy and not to the external method, property or field which eventually receives it.

Enabling the references dynamic proxy also protects your code against any changes. This means that you cannot use the references dynamic proxy if you need to modify your application after SmartAssembly has built it.

To enable the references dynamic proxy, select the option in the references dynamic proxy section of the main project settings window in SmartAssembly.

Note: You cannot use the references dynamic proxy with .NET 1.1, or with any version of Microsoft Compact Framework, Silverlight or XNA.
**Pruning code**

SmartAssembly can remove any unused code in the assembly (for example, meta-data or code which will never be executed). Pruning meta-data from your code helps to make it harder to understand; pruning unused code can improve performance.

**Note:** SmartAssembly will try to identify and remove only unused code. However, if you call a member by reflection (including serialization, remoting or data-binding), SmartAssembly may also remove useful code. To prevent this, click **Exclusions...** to specify those members that should not be pruned by SmartAssembly.

To prune code, select the option in the pruning section of the main project settings window in SmartAssembly.

---

**Pruning**

Pruning automatically scans your software and removes any code that is not essential. This includes removing non-useful metadata such as the names of events, properties, and method parameters. Removing this metadata protects your code by making it more difficult to understand.

Pruning your software reduces the program size, making it easier to deploy through the Internet or a corporate network, and improves loading time and global performance.

Select the merged assemblies you want to prune:

You can use custom attributes (DoNotPruneAttribute and DoNotPruneTypeAttribute) to specifically exclude members from the pruning.
Questions about obfuscation

Will obfuscation break my code?

SmartAssembly attempts to determine when it would be unsafe to obfuscate a name. If required, you can also manually exclude specific members from being obfuscated by clicking [Exclusions...](#) in the Obfuscation section of the project settings.

By default, SmartAssembly will obfuscate all public members in your code. You can choose not to obfuscate public members if this stops your program from running.

Can I obfuscate DLLs?

You cannot normally obfuscate external members of DLLs with SmartAssembly because this would make them useless to other applications which share them.

You can use SmartAssembly to merge most DLLs into your code, however. This then allows you to protect the code in the DLL by obfuscating it. It will also improve the performance of your application as all calls to the DLL will now be internal rather than public.

Note: Some DLLs will break if you merge them, including shared DLLs used for remoting in a client-server architecture and third-party DLLs which employ code-integrity protection.
If your DLLs cannot be merged, you can use SmartAssembly to embed DLLs in your code instead. This means that a compressed version of the DLL is stored in the assembly. At runtime it will be decompressed and loaded into memory. Embedded DLLs can be obfuscated but this technique may not improve the performance of your application because the resulting DLL calls will still be public.

Note: You cannot use embedding with .NET 1.1, or with any version of Microsoft Compact Framework, Silverlight or XNA.

If I have a bug in my obfuscated code and a user sends me a stack trace, how can I read it?

Because obfuscation makes parts of your code unreadable, a stack trace generated from obfuscated code is meaningless.

SmartAssembly's error reporting feature will automatically translate the obfuscated parts of your code back to their original values when you view an error report. If users send stack traces to you manually (by email, for example), you can use SmartAssembly to translate these, too.

Note:

- To be able to decode stack traces after obfuscation, you must only use ASCII characters to obfuscate your code.
- To be able to see the original field names in the stack traces, you must also choose the one-to-one field renaming scheme.
Conclusion

In this paper, I have shown how you can use SmartAssembly to protect your intellectual property by obfuscating your code (including DLLs), hiding most calls to external methods and removing unessential code. I have also shown that, thanks to SmartAssembly’s built-in error reporting feature, obfuscating your code does not mean that you have to lose important debugging information from your customers.
Acknowledgements

Trademarks and registered trademarks

Red Gate is a registered trademark of Red Gate Software Ltd registered in the U.S. Patent and Trademark Office.

.NET Reflector and SQL Compare are registered trademarks of Red Gate Software Ltd registered in the U.S. Patent and Trademark Office.

ANTS Performance Profiler, ANTS Memory Profiler, .NET Reflector Pro, Exception Hunter, Schema Compare for Oracle, SQL Backup, SQL Data Compare, SQL Comparison SDK, SQL Dependency Tracker, SQL Doc, SQL HyperBac, SQL Log Rescue, SQL Monitor, SQL Multi Script, SQL Packager, SQL Prompt, SQL Refactor, SQL Scripts Manager, SQL Storage Compress, SQL Toolbelt, SQL Virtual Restore, and Exchange Server Archiver are trademarks of Red Gate Software Ltd.


Oracle is a registered trademark of Oracle Corporation and/or its affiliates.

InstallShield is a registered trademark and service mark of InstallShield Software Corporation.

PagerDuty is a registered trademark of PagerDuty, Inc.

Copyright information

All Red Gate applications are © Red Gate Software Ltd 1999 - 2011

SQL Backup, SQL Compare, SQL Data Compare, SQL Packager, and SQL Prompt contain software that is Copyright © 1995 - 2005 Jean-loup Gailly and Mark Adler.

SQL Doc includes software developed by Aspose (http://www.Aspose.com).

SQL Backup contains software that is Copyright © 2003 - 2008 Terence Parr. Refer to the ACKNOWLEDGEMENTS.txt file in your SQL Backup installation directory for the full license text.

Why should I obfuscate my code? .............................................................. iii
Obfuscating names ............................................................................. iv
Strings encoding ................................................................................ xi
Obfuscating control flow .................................................................. xii
Using a dynamic proxy for external references .............................. xiii
Pruning code ....................................................................................... xiv
Questions about obfuscation ............................................................. xv

© Red Gate Software Ltd
19 October 2011