**Date:** Sun, 18 Feb 2001 05:20:43 -0600
**Reply-To:** dotnet discussion <DOTNET@DISCUSS.DEVELOP.COM>
**Sender:** dotnet discussion <DOTNET@DISCUSS.DEVELOP.COM>
**From:** "Jay Freeman (saurik)" <saurik@SAURIK.COM>
**Subject:** ANN: Exemplar - MSIL -> C# Decompiler (was: How to protect the source code from ILDASM?)
**Comments:** cc: Dianne Freeman <sammarrah@saurik.com>,
"Patrick J. Dietzen" <pato@cyberuniverse.com>,
"James M. Gunther" <jmgunther@earthlink.net>
**In-Reply-To:** <01C0757B.57530660.dmb@pfxcorp.com>
**Content-Type:** text/plain; charset="US-ASCII"

OK, if I don't release this today I think I am going to go insane :). Was originally hoping to release this in the first few days of January, and then the first few weeks of January, and then last week, and... well... *sigh*.

Binary: ftp://ftp.saurik.com/pub/net/exemplar/Exemplar-0.1.exe
Source: ftp://ftp.saurik.com/pub/net/exemplar/Exemplar-0.1.src.zip

The output definitely isn't perfect, but I consider it quite good (only reason I don't consider it "amazingly good" is I have heard that there are some pretty awesome Java decompilers out there that produce source code so clean that people purposely compile and then decompile their program just to get the better output...) and it can be made better with more work. I'd be curious to see what David's reaction to the fidelity of the data that can be pulled is :).

The usage is simple, it takes one argument: the filename of the IL executable. Source code is then spit to stdout. If you give it a second argument (anything at all, nothing specific, I'm going to add more intelligent option parsing in a future release) it will kick out information on the operations and statements it pulls out of the IL as well.

The following is a list of problems with my output that actually would keep you from recompiling back to the original executable:

- The '?:' operator results in useless output.
- Some switch statements (a surprising number of the ones in System.Net.DLL, for instance) contain really, really strange constructs that I don't know how to decompile and sometimes even crash my parsing algorithm... right now I'm just hoping that those weren't written in C#.
- No attributes.
- No explicit interface implementation.
- Not as much of a biggie, but I don't setup the "using" directives.
- Occasionally the backup branch->goto conversion goes batty.
- Like ildasm, no resources are extracted.
- Value type initialization sometimes generates initobj instructions, which I am currently printing out a silly placeholder statement for until I figure out whether C# even has a syntax for that or just does it for you.
- Certain load by address statements don't serialize into C# currently.
- Various extended attributes of classes, interfaces, methods, and properties sometimes get lost, should be easy to add them back in, I just haven't dealt with them yet.

I'd very much like to hear comments, suggestions... even flames... just feedback in general tends to be useful :).

Sincerely,

Jay Freeman (saurik)
saurik@saurik.com

-----Original Message-----
From: dotnet discussion [mailto:DOTNET@DISCUSS.DEVELOP.COM]On Behalf Of
David Bennett
Sent: Tuesday, January 02, 2001 6:50 PM
To: DOTNET@DISCUSS.DEVELOP.COM
Subject: Re: [DOTNET] How to protect the source code from ILDASM?

At the risk of flogging a dead horse beyond the Stygian abyss...

In my view, the push-pop code of MSIL and re-generated by ILDASM is no easier and no harder to de-compile than the typical assembly language produced by a modern C compiler. Any way you look at it, turning binary code into source code is a slow, tedious, error-prone business. It's a job only tackled by people with either heaps of money or heaps of time. Given an opponent with enough of both, there is no defence. Period.

Factors which tend to make a difference include:
1. Availability of symbols. A standalone EXE typically has no symbol table. Even a DLL only has the export table. MS.NET executables tend to have more symbolic information, because they carry their metadata with them. In both cases, all imported symbols are available.
2. Optimisation. An optimising C compiler *may* scramble the code significantly, but in most cases it won't. MSIL is produced with very little optimisation, so it's a little easier to read.
3. Subterfuge. C/C++ code may (but usually doesn't) play all sorts of tricks with static data, pointers, casts, unions, etc, much of which is excluded by the C#/VB compilers. If you're desperate, you could always write it in ILASM!

My only area of concern is the symbols and metadata. I haven't yet studied a release assembly to see to what extent it is publishing information I would rather it did not. Apart from this, I don't intend to worry.

After all, anyone who recovers source by decompilation and then uses that source in a way which hurts you is guilty of copyright violation and subject to recovery of open-ended damages through the courts.

Regards
David Bennett
[ POWERflex Corporation     Research Laboratories ]
[ Tel:  +61-3-9888-5833     Fax:  +61-3-9888-5451 ]
[ E-mail: dmb@pfxcorp.com   Web:  www.pfxcorp.com ]