Description

de4dot is an open source (GPLv3) .NET deobfuscator and unpacker written in C#. It will try its best to restore a packed and obfuscated assembly to almost the original assembly. Most of the obfuscation can be completely restored (e.g. string encryption), but symbol renaming is impossible to restore since the original names aren't (usually) part of the obfuscated assembly.

It uses dnlib to read and write assemblies so make sure you get it or it won't compile.

Features

Here's a pseudo random list of the things it will do depending on what obfuscator was used to obfuscate an assembly:

Inline methods. Some obfuscators move small parts of a method to another static method and calls it.
Decrypt strings statically or dynamically
Decrypt other constants. Some obfuscators can also encrypt other constants, such as all integers, all doubles, etc.
Decrypt methods statically or dynamically
Remove proxy methods. Many obfuscators replace most/all call instructions with a call to a delegate. This delegate in turn calls the real method.
Rename symbols. Even though most symbols can't be restored, it will rename them to human readable strings. Sometimes, some of the original names can be restored, though.
Devirtualize virtualized code
Decrypt resources. Many obfuscators have an option to encrypt .NET resources.
Decrypt embedded files. Many obfuscators have an option to embed and possibly encrypt/compress other assemblies.
Remove tamper detection code
Remove anti-debug code
Control flow deobfuscation. Many obfuscators modify the IL code so it looks like spaghetti code making it very difficult to understand the code.
Restore class fields. Some obfuscators can move fields from one class to some other obfuscator created class.
Convert a PE exe to a .NET exe. Some obfuscators wrap a .NET assembly inside a Win32 PE so a .NET decompiler can't read the file.
Removes most/all junk classes added by the obfuscator.
Fixes some peverify errors. Many of the obfuscators are buggy and create unverifiable code by mistake.
Restore the types of method parameters and fields

Supported obfuscators/packers

Agile.NET (aka CliSecure)
Babel.NET
CodeFort
CodeVeil
CodeWall
CryptoObfuscator
DeepSea Obfuscator
Dotfuscator

.NET Reactor
Eazfuscator.NET
Goliath.NET
ILProtector
MaxtoCode
MPRESS
Rummage
Skater.NET
SmartAssembly
Spices.Net
Xenocode

Some of the above obfuscators are rarely used (e.g. Goliath.NET), so they have had much less testing. Help me out by reporting bugs or problems you find.

Warning

Sometimes the obfuscated assembly and all its dependencies are loaded into memory for execution. Use a safe sandbox environment if you suspect the assembly or assemblies to be malware.

Even if the current version of de4dot doesn't load a certain assembly into memory for execution, a future version might.

How to use de4dot

N00b users

Drag and drop the file(s) onto de4dot.exe and wait a few seconds.

Deobfuscate more than one file at a time

When more than one assembly has been obfuscated, it's very likely that you must deobfuscate them all at the same time unless you disable symbol renaming. The reason is that if assembly A has a reference to class C in assembly B, and you rename symbols only in assembly B, then class C could be renamed to e.g. Class0 but the reference in assembly *A* still references a class called C in assembly B. If you deobfuscate both assemblies at the same time, all references will also be updated.

Find all obfuscated files and deobfuscate them

The following command line will deobfuscate all assemblies that have been obfuscated by a supported obfuscator and save the assemblies to `c:\output`

```
de4dot -r c:\input -ru -ro c:\output
```

`-r` means recursive search. `-ru` means it should ignore unknown files. `-ro` means it should place the output files in the following directory. Typically, you'd first copy `c:\input` to `c:\output`, and then run the command. That way all the files will be in `c:\output`, even non-assemblies and non-processed assemblies. When de4dot is finished, you'd just double click the main assembly in `c:\output` and it should hopefully start.

Detect obfuscator

Use the `-d` option to detect the obfuscator without deobfuscating any assembly.

Find all .NET assemblies and detect obfuscator. If it's an unsupported obfuscator or if it's not obfuscated, it will print "Unknown obfuscator".

```
de4dot -d -r c:\input
```

Same as above except that it will only show which files have been obfuscated by a supported obfuscator.

```
de4dot -d -r c:\input -ru
```

Detect obfuscator

```
de4dot -d file1.dll file2.dll file3.dll
```

Preserving metadata tokens

Sometimes in rare cases, you'd want to preserve the metadata tokens. Use `--preserve-tokens` or `--preserve-table`. Also consider using `--keep-types` since it won't remove any types and methods added by the obfuscator. Another useful option is `--dont-create-params`. If used, the renamer won't create Param rows for method parameters that don't have a Param row. That way the ParamPtr table won't be added to your assemblies. Peverify has a bug and doesn't support it (you'll see lots of "errors").
The #Strings, #US and #Blob heaps can also be preserved by using `--preserve-strings`, `--preserve-us`, and `--preserve-blob` respectively. Of these three, `--preserve-us` is the most useful one since `ldstr` instruction and `module.ResolveString()` directly reference the #US heap.
`--preserve-sig-data` should be used if the obfuscator adds extra data at the end of signatures that it uses for its own purpose, e.g. as decryption keys. Confuser is one obfuscator that does this.
`--preserve-tokens` preserves all important tokens but will also enable `--preserve-us`, `--preserve-blob` and `--preserve-sig-data`.
If it's detected as an unknown (unsupported) obfuscator (or if you force it with `-p un`), all tokens are preserved, including the #US heap and any extra data at the end of signatures. Also, no obfuscator types, fields or methods are removed.

Preserve all important tokens, #US, #Blob, extra sig data.

```
de4dot --preserve-tokens file1.dll
```

Preserve all important tokens, #US, #Blob, extra sig data and don't remove types/fields added by the obfuscator

```
de4dot --keep-types --preserve-tokens file1.dll
```

Preserve all important tokens, #US, #Blob, extra sig data and don't create extra Param rows to prevent the ParamPtr table from being created.

```
de4dot --dont-create-params --preserve-tokens file1.dll
```

Preserve all important tokens except the Param tokens.

```
de4dot --preserve-table all,-pd file1.dll
```

Dynamically decrypting strings

Although `de4dot` supports a lot of obfuscators, there's still some it doesn't support. To decrypt strings, you'll first need to figure out which method or methods decrypt strings. To get the method token of these string decrypters, you can use ILDASM with the 'show metadata tokens' option enabled. A method token is a 32-bit number and begins with 06, e.g. 06012345.
This command will load assembly file1.dll into memory by calling `Assembly.Load()`. When it detects calls to the two string decrypters (06012345 and 060ABCDE), it will call them by creating a dynamic method, and save the result (the decrypted string). The call to the string decrypter will be removed and the decrypted string will be in its place.

```
de4dot file1.dll --strtyp delegate --strtok 06012345 --strtok 060ABCDE
```

Since the assembly is loaded and executed, make sure you run this in a sandbox if you suspect the file to be malware.

Forcing detection of a certain obfuscator

`de4dot` isn't perfect. If it fails to detect an obfuscator, you can use the `-p` option to force it to assume it's been obfuscated by it.

Force SmartAssembly

```
de4dot file1.dll -p sa
```

Force unsupported obfuscator

```
de4dot file1.dll -p un
```

For other obfuscator types, see the help screen.

Disabling symbol renaming

Renaming symbols isn't as easy as renaming A to B when reflection is involved. `de4dot` currently doesn't support renaming XAML so if you suspect that it uses WPF (or if it's a Silverlight app) you should disable renaming if the assembly fails to run.

```
de4dot --dont-rename file1.dll file2.dll
```

`--keep-names` can also be used to tell `de4dot` not to rename certain symbols, e.g. "don't rename fields".

Rename everything that should be renamed except properties, events and methods.

```
de4dot --keep-names pem file1.dll
```

Using a different rename regex

The default regexes should be enough, except possibly the one that is used when an unsupported obfuscator is detected. To see all default regexes, start `de4dot` without any arguments and it will list all options and all default values.

Eg., currently the following is the default regex used when Dotfuscator is detected

```
!^[a-z][a-z0-9]{0,2}$&!^A_[0-9]+$&^[\u2E80-\u9FFFa-zA-Z_<{$][\u2E80-\u9FFFa-zA-Z_0-9<>{}$.`-
]*$
```

As you can see, it's not just one regex, it's more than one. Each one is separated by `&` and each regex can be negated by using `!` in front of it. To show it more clearly, these regexes are used:

```
(negated) ^[a-z][a-z0-9]{0,2}$
(negated) ^A_[0-9]+$
^[\u2E80-\u9FFFa-zA-Z_<{$][\u2E80-\u9FFFa-zA-Z_0-9<>{}$.`-]*$
```

To change the regex(es), you must know the short type name of the obfuscator (see help screen). e.g. it's `sa` if it's SmartAssembly, and `un` if it's an unsupported/unknown obfuscator. The option to use is `--TYPE-name` (e.g. `--sa-name` for SmartAssembly and `--un-name` for unknown/unsupported obfuscators):

```
de4dot --un-name "^[a-zA-Z]\w*$" file1.dll
```

Other options

Start `de4dot` without any arguments and it will show all options.