

Лабораторна №7

Мета роботи: Вивчення можливостей існуючих обфускаторів для захисту програмних продуктів.

Огляд обфускаторів для .NET

- Безкоштовні обфускатори дуже слабкі та придатні лише для простого перейменування. Про control flow знаю їх лише деякі;
- Існують дуже хороші рішення (control flow, MSIL encryption) вартістю до \$500;
- Дорослі рішення коштують близько 5000, але, на жаль, для багатьох є розпакувальники. Деякі з них крякнуті. Крякнули обфускатор — отже, зрозуміли його систему захисту. На смітник такий обфускатор.
- Є рішення «проти бидлохакерів» - складання шифрується повністю і розшифровується на льоту. Зламати символьним налагоджувачем таке складання простіше простого.

Складемо таблицю найбільш популярних обфускаторів.

Назва та URL	Вартість	Control flow	Шифрування MSIL	Детальніше...
.NET Reactor https://www.eziriz.com/	\$180	+	+	Шифрує код, зламати його досить важко, але можливо, є розпакувальник
{ SmartAssembly } https://www.red-gate.com/	\$795	+	-	Використовується RedGate. Крякнутий.
Salamander.NET http://www.remotesoft.com/	\$800	-	-	На прикладі, наведеному на

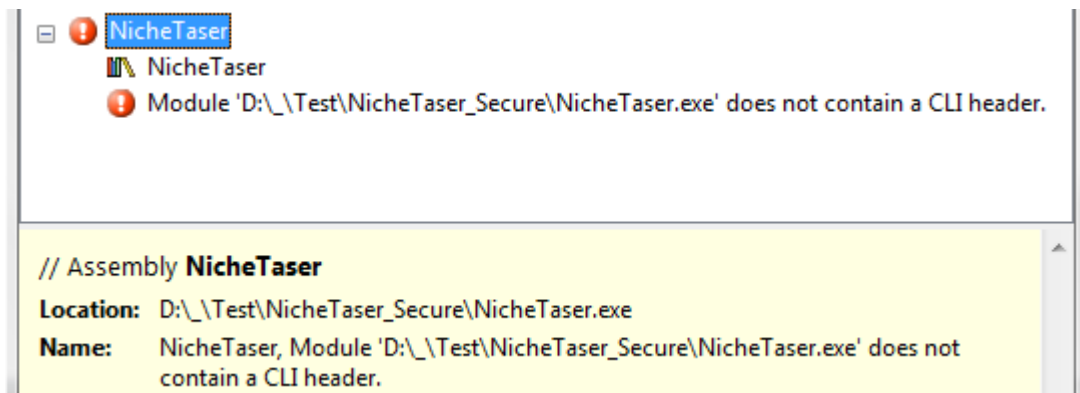
				сайті, рефлексор, звичайно, лається, підозрюючи засідку, але дизасемблить на ура
Babel https://www.babelfor.net/	\$250	+	?	Щось шифрує, але в рантаймі можна виконати DumLL, можливо, повна версія працює добре
XHEO CodeVeil http://xheo.com/	\$900	+	+	Відомий тим, що палиться в антивірусах. В цілому, гарна штука
dotNetProtector http://dotnetprotector.pvlog.com/	\$500	+	+	Додав до проекту більше 4 MB своїх DLL
Spices.NET https://spices.net/	\$400	-	±	Шифрує всю збірку цілком, що погано
Goliath.NET https://goliath.net/	\$115	+	±	Складання рефлексором не відкривається, але у WinDbg видно весь вихідний код
Eazfuscator.NET https://eazfuscator.net/	Free	-	-	Простий rename

VMWare ThinApp www.vmware.com	> \$5000	+	+	Ця програма може запускатися навіть без .NET. Серйозна штукавина
Xenocode PostBuild	> \$1000	+	+	Сам обфускатор крякнуто, що наводить на не дуже добрі думки

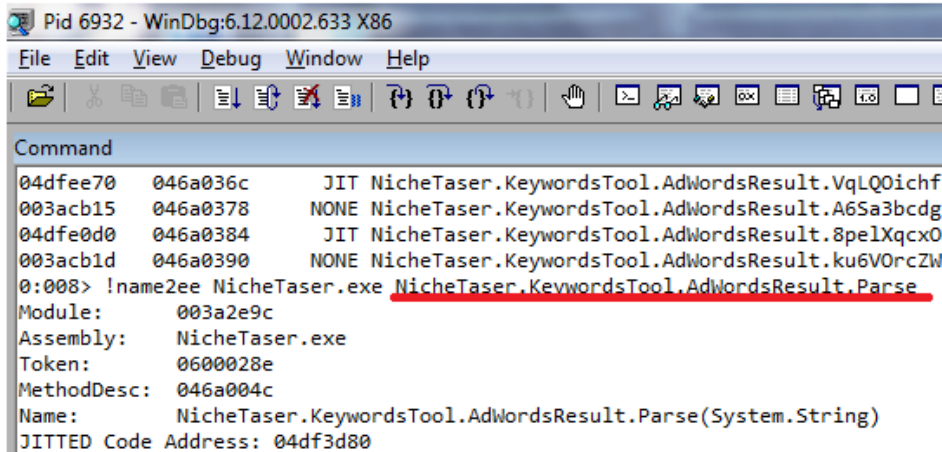
NET Reactor

Шифрує код за допомогою NecroBit (назва їхньої технології), на форумах ходять чутки про те, що NecroBit успішно крякнуть. Можливо, просто цього не знайшов, т.к. особливих зусиль не докладав. Код розібрати не вдалося, WinDbg теж нічого не знайшов.

Reflector:



Щось витягти за допомогою WinDbg можна, але IL-код методів не віддається.



SmartAssembly

Його недавно купила компанія RedGate. Чесно кажучи, вибору RedGate я не зрозумів, не вміє навіть шифрувати MSIL. Посидівши з налагоджувачем, можна розібратися в коді. Я не раджу використати цей обфускатор, ціна \$750 явно не відповідає якості. Все, що робить цей обфускатор з кодом - це обфускування control flow приблизно такому вигляді: Reflector в C# код не розбирає (хоча це зробити нескладно), але в IL - відмінно:

L_1 br.s L_4

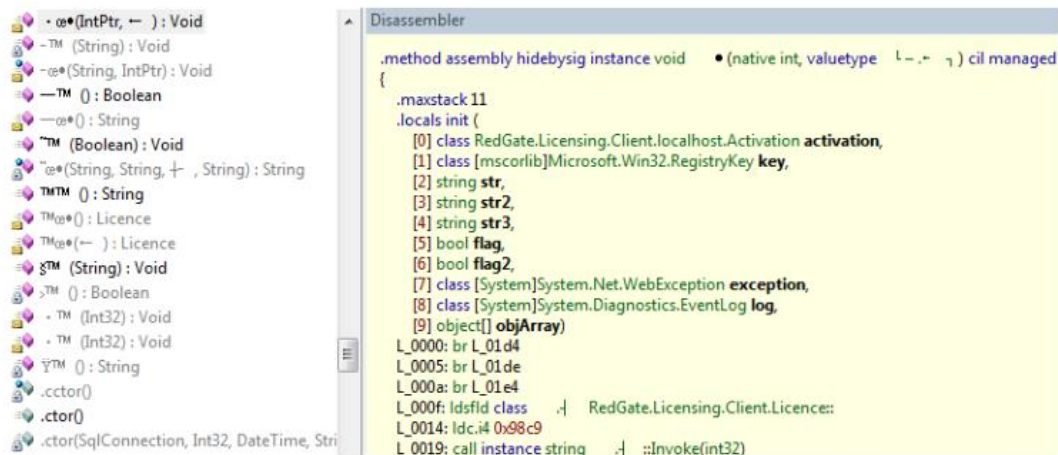
L_2 br.s L_3

L_3 ret

L_4 push

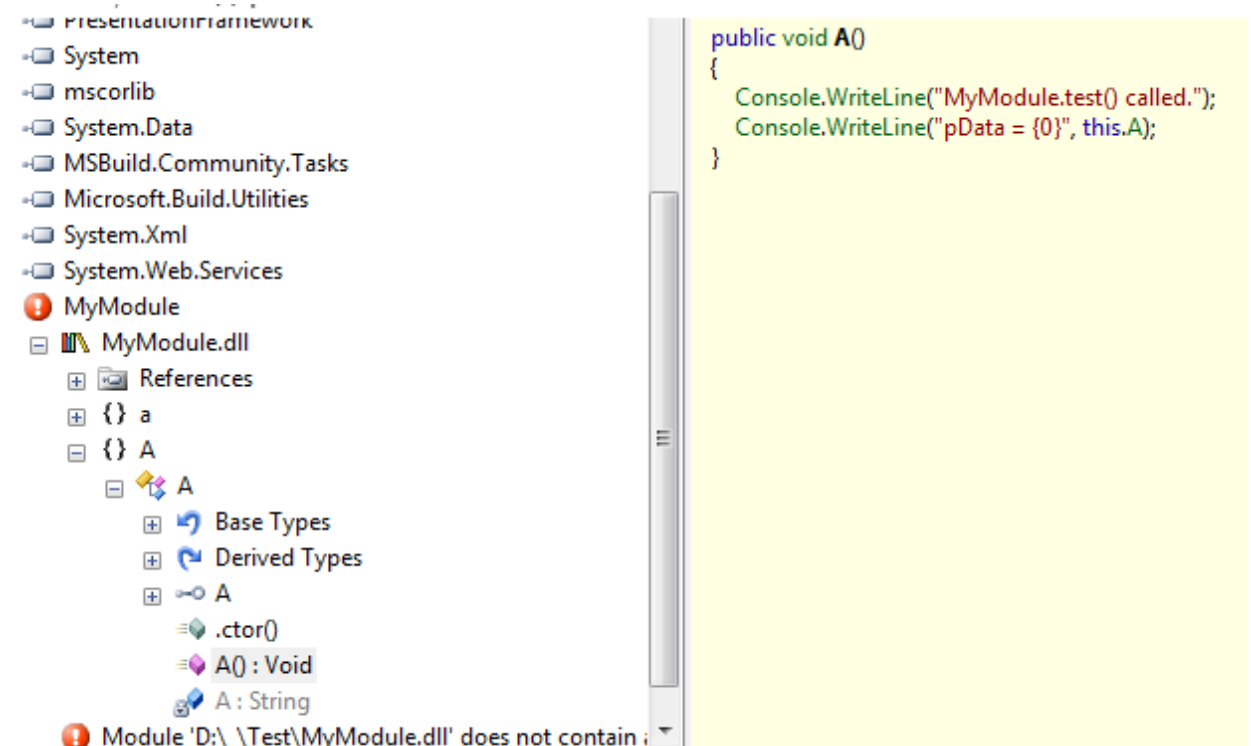
L_5 ldc.i4.1

L_6 br.s L_2



Salamander.NET

Завантажив супер-приклад з їхнього сайту. Обіцяли, що Reflector не відкриє. Reflector, насправді, лаявся, але відкрив:



Babel (той, що не free), CodeWall, dotNetProtector

Продукти одного класу, я вважаю, що дуже підходять, якщо потрібен захист коду від подальшого рефакторингу та зайвої цікавості. Babel:

```
.method private hidebysig static string • (string • , string • , string • ) cil managed
{
    .maxstack 5
    .locals init (
        [0] string str)
    L_0000: br.s L_0004
    L_0002: 0x00A8 // Unknown IL instruction.
}
```

Однак WinDbg з успіхом показав, що всередині збірки, зробленої Babel (ймовірно, тому, що у мене була community версія; повну, на жаль, я не перевіряв):

```

Command
066dfd34 64db5a08 clr!Thread::intermediateThreadProc+0x4b
066dfec0 64db59f6 clr!Thread::intermediateThreadProc+0x39, calling clr!_all
066dfed4 76f43677 KERNEL32!BaseThreadInitThunk+0xe
066dfee0 77469d72 ntdll!_RtlUserThreadStart+0x70
066dff20 77469d45 ntdll!_RtlUserThreadStart+0x1b, calling ntdll!_RtlUserThr
0:013> !name2ee NicheTaser.exe NicheTaser.KeywordsTool.AdWordsResult.Parse
Module:      00152e9c
Assembly:    NicheTaser.exe
Token:       06000046
MethodDesc:  001570f4
Name:        NicheTaser.KeywordsTool.AdWordsResult.Parse(System.String)
JITTED Code Address: 00474f50
0:013> !DumpIL 001570f4
ilAddr = 00c95888
IL_0000: br.s IL_0004
IL_0002: unused
IL_0003: unused
IL_0004: ldarg.0
IL_0005: br.s IL_0045
IL_0007: brfalse.s IL_003b
IL_0009: br.s IL_0029
IL_000b: ldc.i4 61991
IL_0010: br.s IL_001a
IL_0012: ldstr "...
C# Source Code Obfuscator

```

На сайті пишуть, що працює з вихідними джерелами, при цьому виходить заобфусцований вихідний код. Дуже цікавий підхід, але, на жаль, скачати сам обфускатор не можна. Мінус — за такого підходу неможливе шифрування MSIL та невірні інструкції.

XNEO CodeVeil

На попередню версію цього обфускатора було багато кряків, які з'являлися дуже швидко. Що говорить про те, що хакери знають, як працює цей обфускатор.

На останню версію кряка я не бачив, але є погана проблема:

- Програми не подобаються антивірусу, обфускатор зашифровує складання і записує в себе (що радує, складання шифрується по шматочках)
- Після цього обфускатора, додатки треба дуже добре тестувати, баги можуть впасти в найнесподіваніших місцях.

Використовувати можна, але чекайте на сюрпризи.

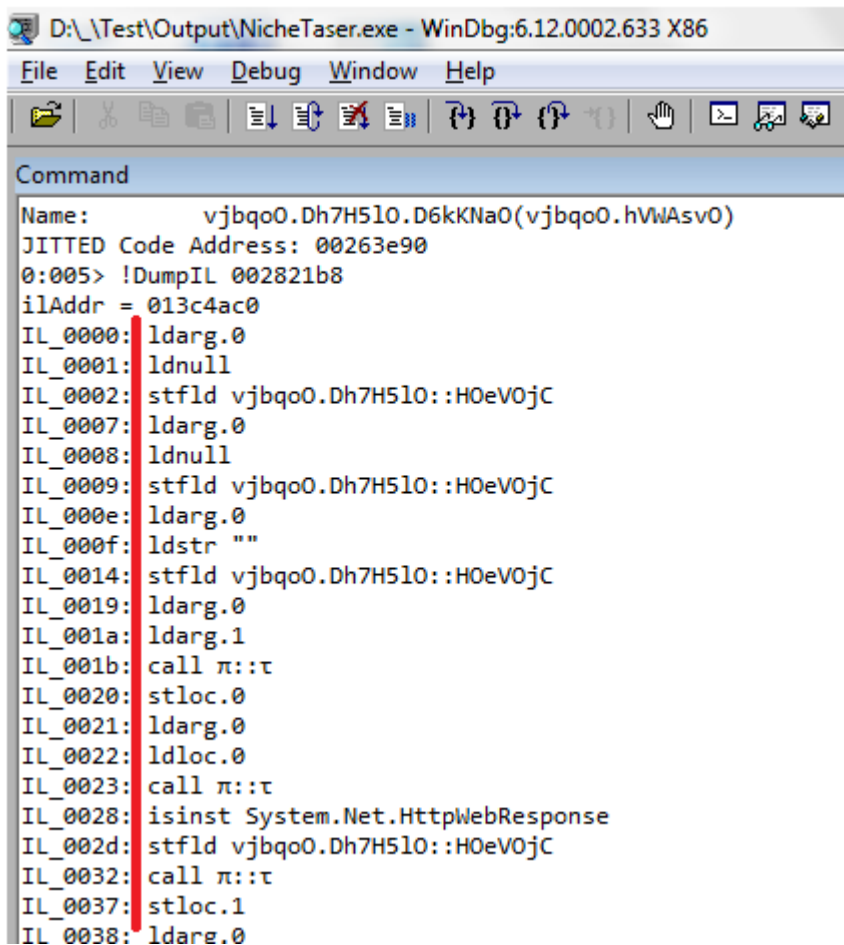
dotNetProtector

Збірка вийшла досить хороша, але на мене чекала засідка - зі збіркою прийшли 4 МБ DLL-ек від цього обфускатора:

```
public string ò  
{  
    [MethodImpl(MethodImplOptions.NoInlining)]  
    get  
    {  
        throw new ApplicationException();  
    }  
}
```

Spices.NET

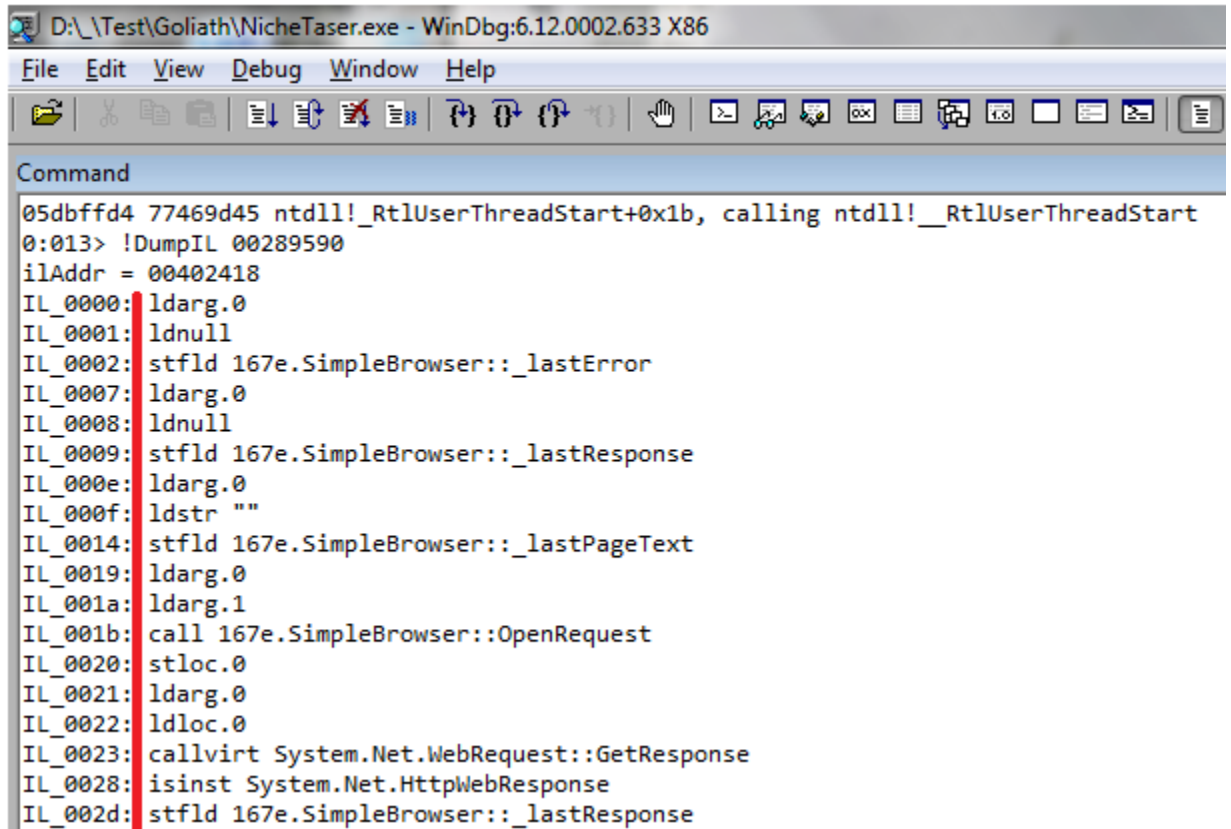
Дуже відомий продукт, я був здивований, що отримані програми так просто зламати:



```
D:\Test\Output\NicheTaser.exe - WinDbg:6.12.0002.633 X86  
File Edit View Debug Window Help  
Command  
Name: vjbqo0.Dh7H510.D6kKNa0(vjbqo0.hVwAsv0)  
JITTED Code Address: 00263e90  
0:005> !DumpIL 002821b8  
ilAddr = 013c4ac0  
IL_0000: ldarg.0  
IL_0001: ldnull  
IL_0002: stfld vjbqo0.Dh7H510::H0eV0jC  
IL_0007: ldarg.0  
IL_0008: ldnull  
IL_0009: stfld vjbqo0.Dh7H510::H0eV0jC  
IL_000e: ldarg.0  
IL_000f: ldstr ""  
IL_0014: stfld vjbqo0.Dh7H510::H0eV0jC  
IL_0019: ldarg.0  
IL_001a: ldarg.1  
IL_001b: call π::τ  
IL_0020: stloc.0  
IL_0021: ldarg.0  
IL_0022: ldloc.0  
IL_0023: call π::τ  
IL_0028: isinst System.Net.HttpWebResponse  
IL_002d: stfld vjbqo0.Dh7H510::H0eV0jC  
IL_0032: call π::τ  
IL_0037: stloc.1  
IL_0038: ldarg.0
```

Goliath.NET

Рефлектором складання не відкрити, вона зашифрована. Косяк у тому, що збірка повністю зашифрована, і розшифровується вона після того, як запуситься програма. Після розшифровки жодного захисту у пам'яті:



```
D:\Test\Goliath\NicheTaser.exe - WinDbg:6.12.0002.633 X86
File Edit View Debug Window Help
Command
05dbffd4 77469d45 ntdll!_RtlUserThreadStart+0x1b, calling ntdll!_RtlUserThreadStart
0:013> !DumpIL 00289590
ilAddr = 00402418
IL_0000: ldarg.0
IL_0001: ldnull
IL_0002: stfld 167e.SimpleBrowser::_lastError
IL_0007: ldarg.0
IL_0008: ldnull
IL_0009: stfld 167e.SimpleBrowser::_lastResponse
IL_000e: ldarg.0
IL_000f: ldstr ""
IL_0014: stfld 167e.SimpleBrowser::_lastPageText
IL_0019: ldarg.0
IL_001a: ldarg.1
IL_001b: call 167e.SimpleBrowser::OpenRequest
IL_0020: stloc.0
IL_0021: ldarg.0
IL_0022: ldloc.0
IL_0023: callvirt System.Net.WebRequest::GetResponse
IL_0028: isinst System.Net.HttpWebResponse
IL_002d: stfld 167e.SimpleBrowser::_lastResponse
```

Eazfuscator.Net

Обфускатор безкоштовний. Прикро, що вміє робити лише простий rename. Що може порадувати (особисто мене більше цікавлять консольні або MSBuild-версії) - досить простий процес обфускування, все зводиться до

перетягування файлу збирання. Ось що виходить у результаті

```
Disassembler

public static string ȳ (string ȳ , Func<Image, string> ȳ )
{
    if (string.IsNullOrEmpty(ȳ ))
    {
        throw new ArgumentNullException(ȳ ȳ (-1543723580));
    }
    string str = null;
    string str2 = null;
    for (int i = 0; i < 5; i++)
    {
        if (ȳ ȳ == null)
        {
            ȳ ȳ = new ȳ ȳ ();
            ȳ ȳ .ȳ ȳ (new ȳ ȳ (- ȳ ȳ (-1543723528)));
            ȳ ȳ .ȳ ȳ (true);
        }
        try
        {
            ȳ ȳ = new ȳ ȳ (ȳ ȳ (ȳ , str, str2));
            try
            {
                ȳ ȳ .ȳ ȳ ȳ (ȳ ȳ );
            }
        }
    }
}
```

VMWare ThinApp, Xenocode PostBuild

Обфускатори побудовані за схожим принципом, можуть виконувати код, впроваджуючи в додаток предкомпилированые збірки .NET, що включає можливість перехоплення викликів JIT-компіляції.

Отримані програми можуть запускатись навіть без встановленого .NET на машині. Розмір програми, що вийшов, — 10..50 МБ, залежно від того, які бібліотеки будете використовувати. Коштують ці рішення дуже дорого. Але, на жаль, на PostBuild ходять кряки (навіть на останній). Ймовірно, у відомих колах є й готові розпакувальники.

Висновки

Який пакувальник слід обрати? Однозначної відповідь на це питання не дати, все залежить від того, що для нас є цінністю в коді або що саме ми хочемо приховати:

- **Весь код разом** — важлива не одна «фішечка» в коді, а код повністю. Зашифруй код якимось простим обфускатором, краще зашифрувати MSIL. Якщо важливий дійсно весь код, розшифрувати його повністю буде складніше за написання заново, і ніхто цим не займатиметься;

- **Окрема частина** - наприклад, перевірка ключа. Я порадив би такий код взагалі в паблік не давати, краще різати функціонал у trial-версії. У повній версії ключ перевірити обов'язково, але ризик крадіжки менший. Проте я порадив би використати обфускатор серйозніше.

Тести

1. Б
2. Б, В
3. А
4. А, В
5. А
6. Б
7. В
8. Г
9. А, Б
10. А, Б