Модификация кода .NET в рантайме

Игорь Чевдарь

Модификация кода .NET в рантайме

https://github.com/homuroll/CodeModificationSamples

План

- Применение
- Mono.Cecil
- Хук метода
- Profiling API

Применение

- Диагностика приложения
- Профайлеры
  -  dotTrace
  -  ANTS performance profiler
- Code coverage tool’ы
  -  dotCover
  -  OpenCover
- Библиотеки mock’ов
  -  TypeMock
- Reverse engineering

Модификация кода

- Статическая
- Динамическая

Статическая модификация кода: Mono.Cecil

- Парсит .NET сборку без загрузки в адресное пространство
- Позволяет модифицировать и сохранить на диск
- Модифицировать можно не только код
- По сути, можно реализовать все AOP

Применение в Reverse Engineering

1. С помощью дизассемблера находим кусок кода, который нужно изменить
2. Модифицируем с помощью Mono.Cecil

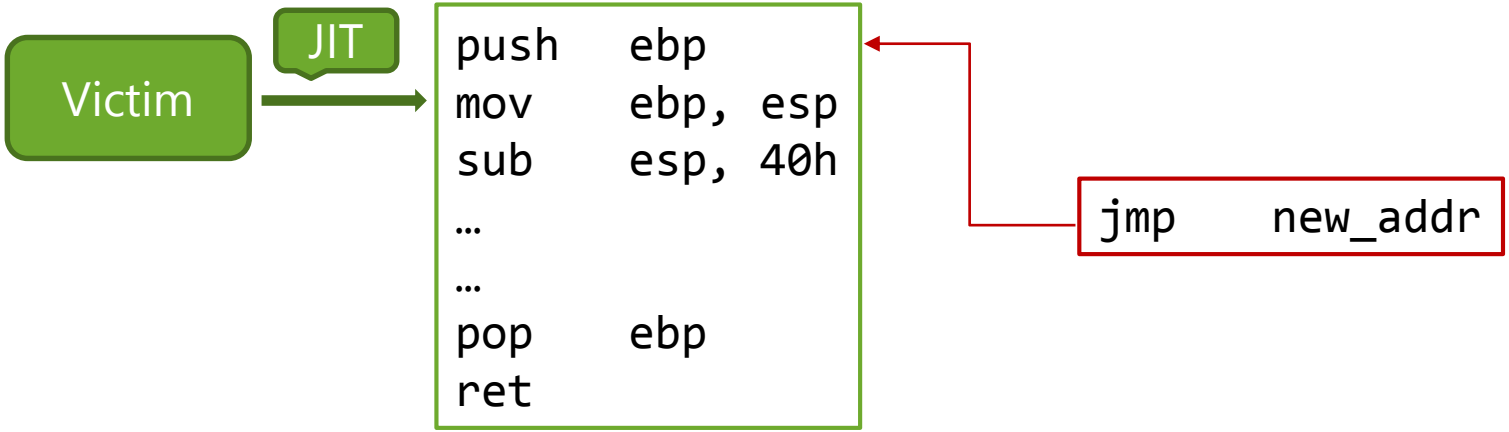
Пример: взлом игры “сапер”

```
// Инициализируем readerParameters
var readerParameters = new ReaderParameters(…) { … };
var assembly = AssemblyDefinition.ReadAssembly("Minesweeper.exe", readerParameters);

var type = assembly.MainModule.Types.First(type => type.Name == "ProductLicense");
var method = type.Methods.First(method => method.Name == "get_TrialDays");
var instruction = method.Body.Instructions.First(
                    instr => instr.OpCode == OpCodes.Ldc_I4_S
                             && (sbyte)instr.Operand == 30);

instruction.OpCode = OpCodes.Ldc_I4;
instruction.Operand = 1000000000;

assembly.Write("Minesweeper.exe");
```

Динамическое модифицирование кода

- Хук метода
- .NET profiling API
- Хук JIT-компилятора

Хук метода


Victim
JIT
push ebp
mov ebp, esp
sub esp, 40h
…
…
pop ebp
ret
jmp new_addr

Хук метода

- RuntimeHelpers.PrepareMethod гарантирует JIT-компиляцию метода
- MethodHandle.GetFunctionPointer() возвращает адрес метода

Хук метода

- Библиотека GrEmit для генерации и модификации IL-кода
  https://github.com/homuroll/GrEmit
  https://www.nuget.org/packages/GrEmit

- `MethodUtil.Hook` ставит хук по предыдущей схеме
- Класс `MethodBody` для манипуляций с телом метода

Изолированное тестирование и mock

```
public interface IDataReader
{
    int[] Read(int column);
}

public class DataProcessor
{
    private IDataReader dataReader;

    public int FindAverage(int column)
    {
        var data = dataReader.Read(column);
        var sum = data.Sum(x => (long)x);
        return (int)(sum / data.Length);
    }
}
```

Задача: mock статического метода

```
public class DataProcessor
{
    public int FindAverage(int column)
    {
        var data = DataReader.Read(column);
        var sum = data.Sum(x => (long)x);
        return (int)(sum / data.Length);
    }
}
```

Хук метода

Варианты использования

- Целевой метод уже подготовлен

```
public void Test()
{
    var dataProcessor = new DataProcessor();
    using (var mock = new Mock<int, int[]>(x => DataReader.Read(x)))
    {
        mock.Set(MockedRead);
        Assert.AreEqual(5, dataProcessor.FindAverage(42));
    }
}

private static int[] MockedRead(int x)
{
    if (x == 42) return new[] {1, 4, 7, 8};
    throw new InvalidOperationException();
}
```

Что подменяем

Новый DataReader.Read

```
public class Mock<T, TResult> : IDisposable
{
    private readonly MethodInfo victim; // Получаем в конструкторе
    private Action unhook;

    public void Set(Func<T, TResult> func)
    {
        Action curUnhook;
        if (!GrEmit.MethodUtil.Hook(victim, func.Method, out curUnhook))
            throw new InvalidOperationException("Unable to hook method");
        if (unhook == null)
            unhook = curUnhook;
    }

    public void Dispose()
    {
        if (unhook != null) unhook();
    }
}
```

```
public void Test()
{
    var dataProcessor = new DataProcessor();
    using (var mock = new Mock<int, int[]>(x => DataReader.Read(x)))
    {
        mock.Set(MockedRead);
        Assert.AreEqual(5, dataProcessor.FindAverage(42));
    }
}

private static int[] MockedRead(int x)
{
    if (x == 42) return new[] {1, 4, 7, 8};
    throw new InvalidOperationException();
}
```

```
public void Test()
{
    var dataProcessor = new DataProcessor();
    using (var mock = new Mock<int, int[]>(x => DataReader.Read(x)))
    {
        mock.Set(x =>
            {
                if (x == 42) return new[] { 1, 4, 7, 8 };
                throw new InvalidOperationException();
            });
        Assert.AreEqual(5, dataProcessor.FindAverage(42));
    }
}
```

```
public void Test()
{
    var dataProcessor = new DataProcessor();
    using (var mock = new Mock<int, int[]>(x => DataReader.Read(x)))
    {
        mock.Set(<>c.<>9__0_0);
        Assert.AreEqual(5, dataProcessor.FindAverage(42));
    }
}

private sealed class <>c
{
    public static readonly <>c <>9 = new <>c();
    public static Func<int, int[]> <>9__0_0 = new Func<int, int[]>(<>c.<>9.<Test>b__0_0);

    internal int[] <Test>b__0_0(int x)
    {
        if (x == 42) return new[] { 1, 4, 7, 8 };
        throw new InvalidOperationException();
    }
}
```

Что это??

Вот оно!

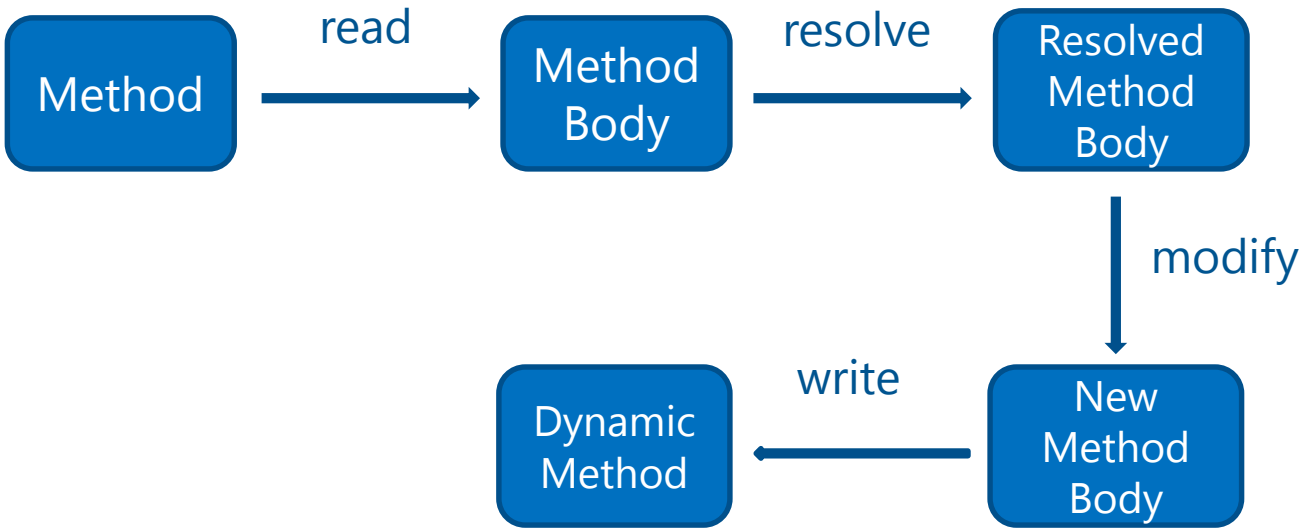
Метод нестатический

Хук метода

Варианты использования

- Целевой метод уже подготовлен
- Целевой метод нужно создать на основе другого

Копирование и модификация метода



Metadata tokens

- У многих IL-инструкций аргументы – токены

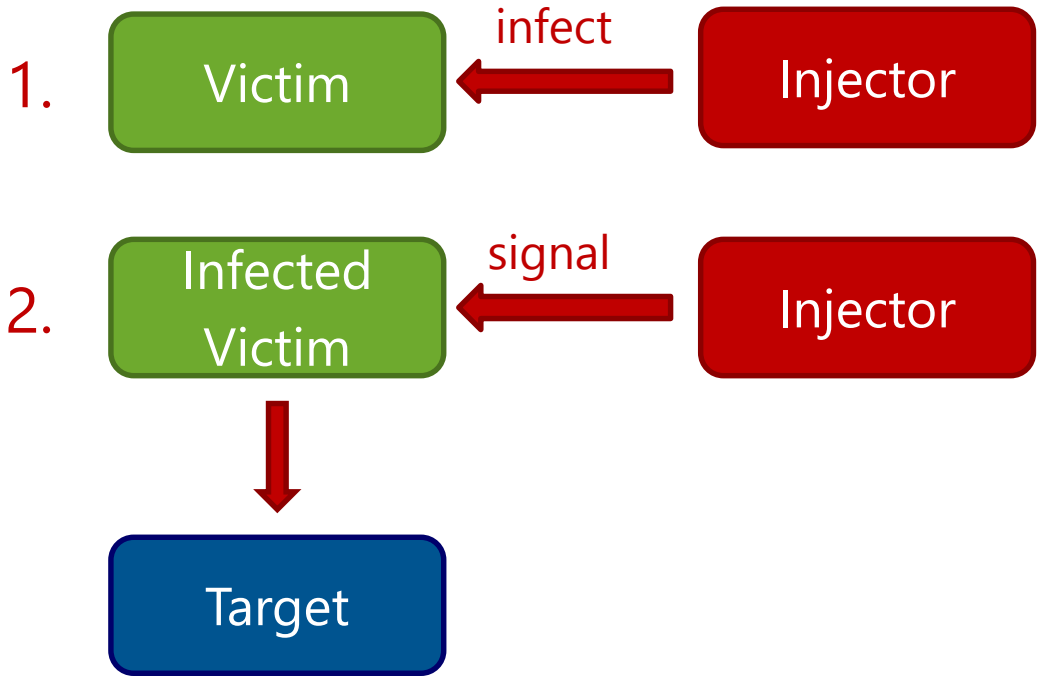
```
Console.WriteLine("zzz");
IL_0000: 72 01 00 00 70       ldstr [String:0x0001]
IL_0005: 28 11 00 00 0A       call [MemberRef:0x0011]
```

- Metadata token – ссылка на метаданные, описанные в модуле
- Токен валиден только внутри модуля

```
public class Mock<T, TResult> : IDisposable
{
    private readonly MethodInfo victim; // Получаем в конструкторе
    private Action unhook;
    private Func<T, TResult> curFunc;

    public void Set(Func<T, TResult> func)
    {
        if (!func.Method.IsStatic)
            func = Clone(func);
        curFunc = func;

        Action curUnhook;
        if (!MethodUtil.Hook(victim, func.Method, out curUnhook))
            throw new InvalidOperationException("Unable to hook method");
        if (unhook == null)
            unhook = curUnhook;
    }
}
```

```
public class Mock<T, TResult> : IDisposable
{
    …
    private static Func<T, TResult> Clone(Func<T, TResult> func)
    {

        var body = MethodBody.Read(func.Method, true);
        foreach (var instruction in body.Instructions)
        {
            switch (instruction.OpCode.Code)
            {
                case Code.Ldarg_0:
                    throw new InvalidOperationException("Method has wrong signature");
                case Code.Ldarg_1:
                    instruction.OpCode = OpCodes.Ldarg_0;
                    break;
            }
        }

        return body.CreateDelegate<Func<T, TResult>>();
    }
}
```

Инъекция кода в чужой .NET процесс

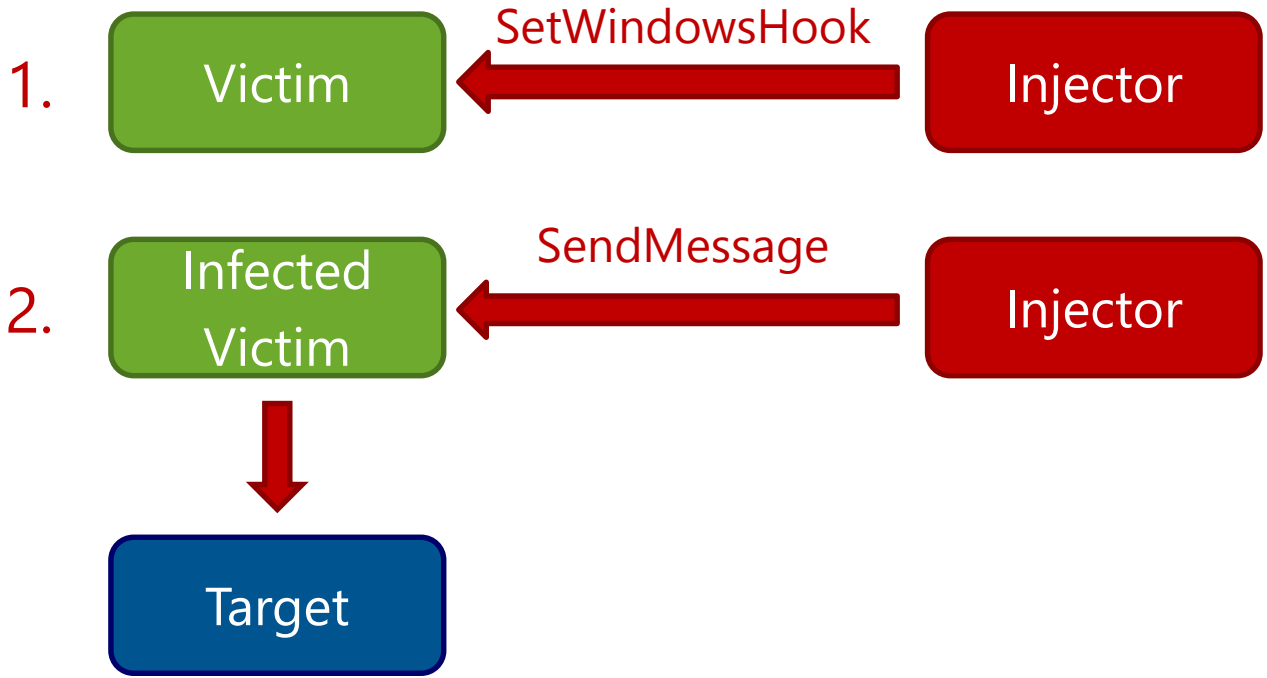


Инъекция кода в чужой .NET процесс

У процесса-жертвы есть GUI

- Заплатка – windows hook
- Signal – SendMessage

Инъекция кода в чужой .NET процесс



Инъекция кода в чужой .NET процесс

```
var injectMessage = RegisterMessage("Inject");
SetWindowsHookEx(WH_CALLWNDPROC, &Inject);
SendMessage(victim, injectMessage, target);
```

```
[DllExport]
public static void Inject(…)
{
    if (message == "Inject")
    {
        var assembly = Assembly.Load(target.Assembly);
        var type = assembly.GetType(target.Type);
        var method = type.GetMethod(target.Method);
        method.Invoke();
    }
}
```

Как экспортировать функцию из .NET

- Пишем на C++/CLI и помечаем __declspec(export)
- Пишем на C#, используем библиотеку UnmanagedExports и помечаем метод атрибутом DllExport

- https://www.nuget.org/packages/UnmanagedExports

Пример: инъекция в игру “сапёр”

Трассировка/логирование на лету

```
private static void Main(string[] args)
{
    var process = Process.GetProcessesByName("devenv")[0];
    Injector.Launch(process.MainWindowHandle, typeof(Program).Assembly.Location,
                    "Injection.Program", "Hook");
}
```

```
public static void Hook()
{
    var method = typeof(FileStream).GetMethod("Init", …);
    var body = MethodBody.Read(method, true);
    body.Instructions.Insert(0, Instruction.Create(OpCodes.Ldarg_1));
    var debugWriteLineMethod = typeof(Debug).GetMethod("WriteLine", …);
    body.Instructions.Insert(1, Instruction.Create(OpCodes.Call, debugWriteLineMethod));

    del = body.CreateDelegate(method);
    Action unhook;
    if(!MethodUtil.Hook(method, del.Method, out unhook))
        Debug.WriteLine("Unable to hook method");
}
```

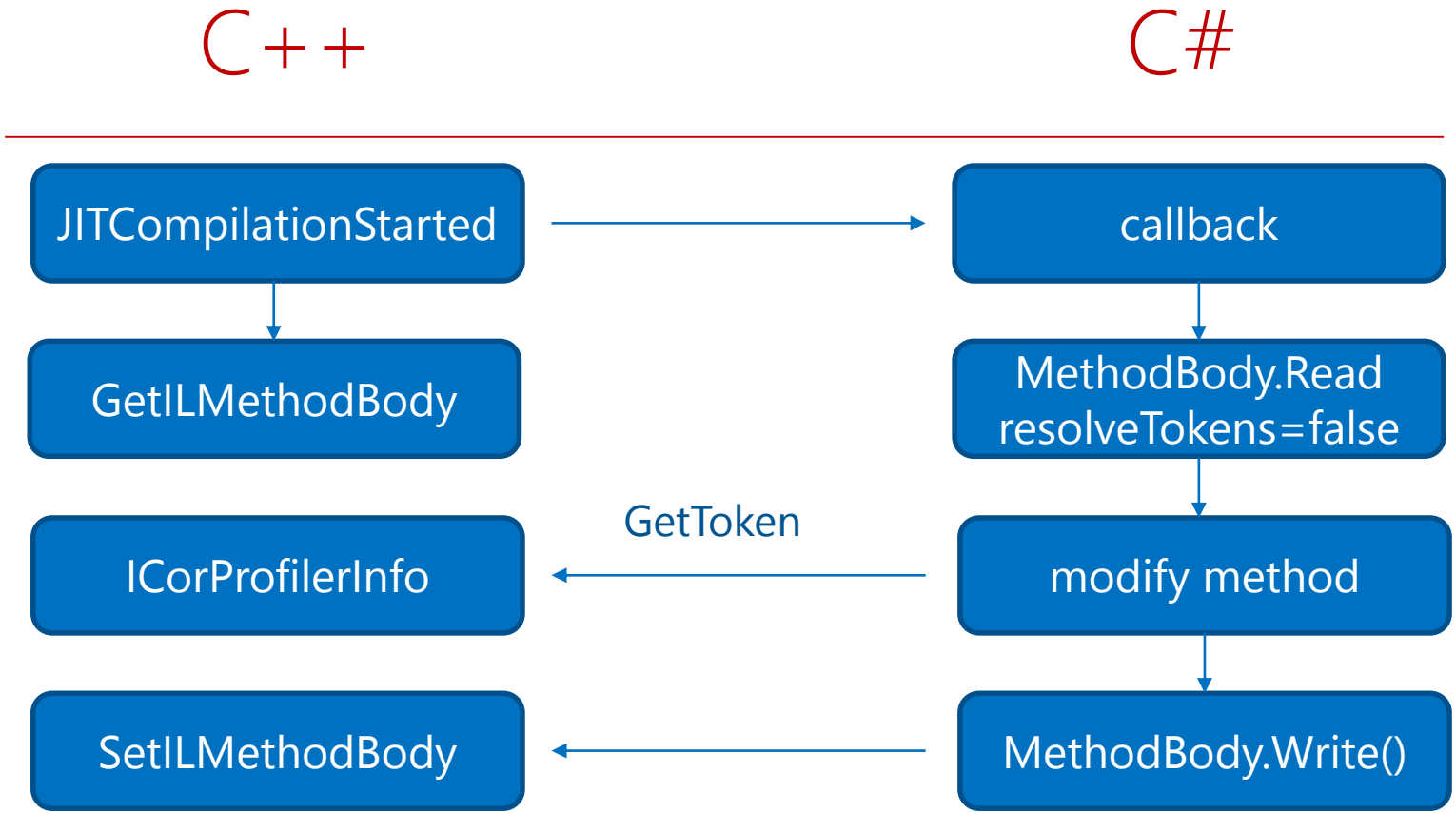
Недостатки хука метода

- Метод по сути не модифицируется, а копируется
- Не документированный способ
- При возникновении исключения стэктрэйс ломается

.NET profiling API

- На старте CLR можно указать через переменные среды, что нужен профайлер:
  - COR_ENABLE_PROFILING = 1
  - COR_PROFILER = {clsid}
  - COR_PROFILER_PATH = path
- Профайлер представляет собой COM-объект, реализующий интерфейс ICorProfilerCallback
- При инициализации нужно указать, что именно профилировать
- Точка входа – C++
- Но с помощью UnmanagedExports можно большую часть написать на C#

```
CorProfiler::Initialize(…)
{
    this->corProfilerInfo = …; // получаем ICorProfilerInfo

    this->corProfilerInfo->SetEventMask(COR_PRF_MONITOR_JIT_COMPILATION);
    // Загрузка Managed-куска профайлера
    auto lib = LoadLibrary("ManagedTrace.dll");
    this->callback = GetProcAddress(lib, "InstallTracing");
}

CorProfiler::JitCompilationStarted(FunctionID functionId, …)
{
    // Получение по functionId имен сборки/модуля и токена метода
    …
    if (assembly == "ManagedTrace.dll")
        return;
    char* methodBody = this->corProfilerInfo->GetILFunctionBody(…);
    char* changed = this->callback(assembly, module, token, methodBody);
    if (changed != nullptr)
        this->corProfilerInfo->SetILFunctionBody(…, changed);
}
```


C++
C#
JITCompilationStarted
callback
GetILMethodBody
MethodBody.Read
resolveTokens=false
GetToken
ICorProfilerInfo
modify method
SetILMethodBody
MethodBody.Write()

.NET profiling API

Использование токенов при модификации кода

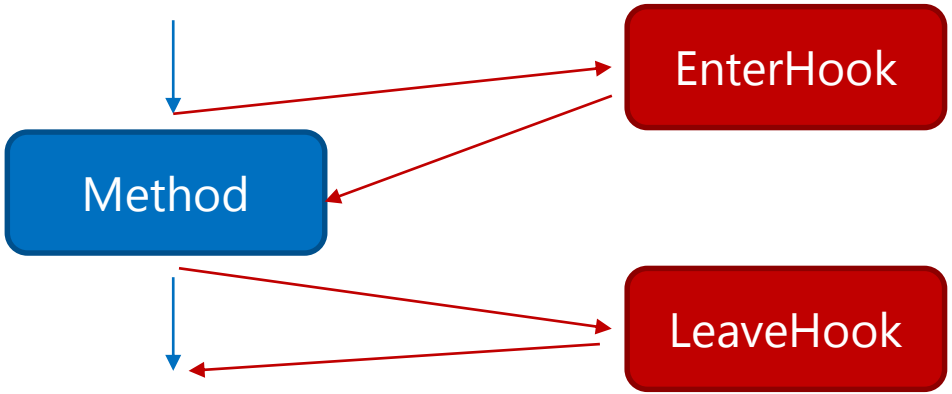
- .NET позволяет получить метаданные по токенам
- .NET позволяет создавать по метаданным токены только для динамически генерируемого кода (`MethodBuilder` и `DynamicMethod`)
- ICorProfilerInfo позволяет добавлять новые метаданные в любой модуль

Пример. Трассировщик

Трассировка

- Модификация кода
- Leave/Enter hooks

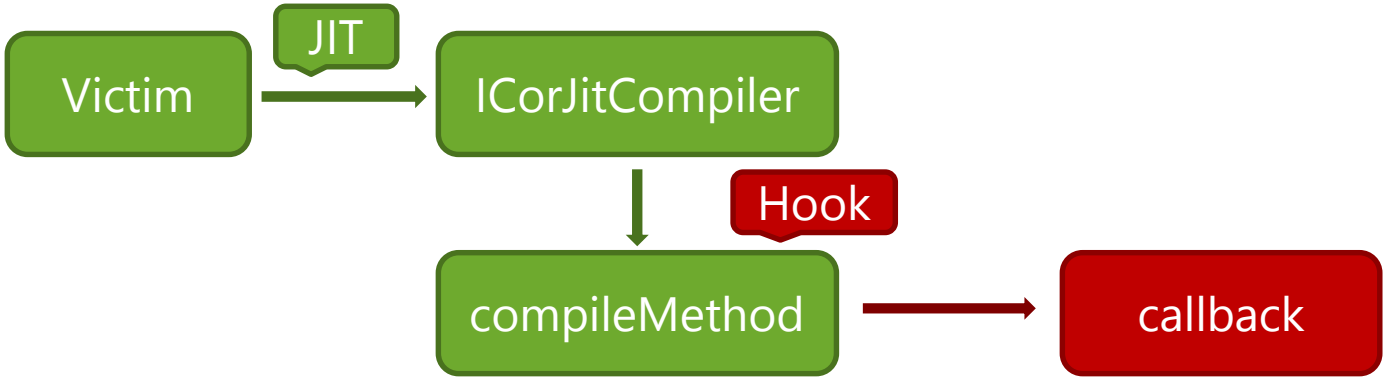
Leave/Enter hooks



Практический пример

- Трассирующий профайлер на продакшене
- Использует обе техники модификации кода:
  - profiling API для обычного кода
  - Хук метода для динамически сгенерированного кода
- Замедление 5-15%
  - Достигается тем, что профилируется только user-код
  - Простые методы не профилируются
  - Есть конфигурирование

Хук JIT-компилятора



http://www.codeproject.com/Articles/463508/NET-CLR-Injection-Modify-IL-Code-during-Run-time

Вывод

- Применять модифицирование кода нужно осторожно
- Предпочтительнее статическая модификация
- Для динамической модификации проще сделать хук метода
- Если нужен профайлер – используем profiling API

ВОПРОСЫ?

tech.skbkontur.ru

Игорь Чевдарь

github.com/homuroll

ichevdar@kontur.ru

homuroll@yandex.ru