

Forum:[Hack Forums](#)

Thread: [C# Mono.Cecil Builder \(Easy!\)](#)

coh0rt - May 27, 2019, 11:55 am

Let's pretend we have two people using a RAT, Bob and Alice. Bob wants to have his client connect back on port 7777, while Alice wants her client to connect on port 5555. The RAT can let the user pick which port the client connects back to. This can be done by creating a builder. A builder basically allows Bob and Alice to choose whatever settings they want. There can be Anti-VM, Persistence, and other settings. The RAT can then generate a new client based on the settings the user chooses. Every generated client has the settings the user picked, but everything else is the same. To do this, the RAT can have a blueprint of the client (the stub) and replace the ip, port, and other settings in that blueprint with whatever the user chooses. The final file generated with all of the user's settings is called the client.

Before we start, let's define some terms.

Builder - Provides the user a user interface to change settings. The builder changes variables in the stub and makes a new client with the settings. The stub and the client basically share the same code, the only thing that is different are the settings that the user chose.

Client - Executed on the victim's computer. Contains the settings (like ip and port) that the user chose in the builder.

Stub - The stub is nothing but a blueprint for the final client.

Let's start! To keep this post short, this is just going to be a POC. It will not be an actual RAT. Code will be written in C# inside of Visual Studio.

Start of by making two .NET framework console app projects. Name them StubProject and BuilderProject.

In StubProject, copy and paste this code in the Main function.

```
Code string message = "Insert message here";  
Console.WriteLine(message);  
Console.ReadLine();
```

In a RAT, message would be one of the settings. It's current value is just a placeholder. Our builder is going to replace the value of message with something else.

Switch to BuilderProject now. We are going to install Mono.Cecil, Visual Studio will automatically reference the DLL after your install it using NuGet Packet Manager Console.

In Visual Studio click "Tools" -> "NuGet Packet Manager" -> "Packet Manager Console"
Then type in "Install-Package Mono.Cecil -Version 0.10.3" and press enter.

Now that you have installed Mono.Cecil you can use it in your project. Add this code to your project. Code using Mono.Cecil;

Now we can use Mono.Cecil. We're going to read StubProject's .exe file and change the value of message.

Before we start, you need to run StubProject at least once so it's compiled into a .exe

Add the code below to your Main function.

First let's read the assembly,

Code AssemblyDefinition module = AssemblyDefinition.ReadAssembly(pathToFile);

Replace pathToFile with the path to StubProject's executable file.

Pay attention to the this next code because it's the main part. I've provided comments inside the code.

```
Code foreach (TypeDefinition type in module.MainModule.Types)
{
    // Iterate through each TypeDefinition, we only care about the Program type though.
    if (type.ToString().Contains("Program"))
    {
        // Now we are in in the program type. Let's iterate through all of the
        // methods until we get to the method we care about - in this case that's the
        // Main method.
        foreach (var method in type.Methods)
        {
            if (method.ToString().Contains("Main"))
            {
                // We found the main method.
                foreach (var instruction in method.Body.Instructions)
                {
                    // Iterating through method.Body.Intructions
                    // method.Body.Instructions basically contains the body of a method.\

                    // We are checking if the value of one of the instructions is a string and is the same
placeholder value we gave message
                    // if it is then we change it to "It works!"

                    if (instruction.Opcode.ToString() == "ldstr" && instruction.Operand.ToString() ==
"Insert message here")
                    {
                        instruction.Operand = "It works!";
                    }
                }
            }
        }
    }
}
```

Now we just need to add one more line of code.

Code module.Write(pathToWrite);

Change pathToWrite to whatever path you want (for example:

C:/Users/YourName/Desktop/app.exe). A new .exe file with the new message (the client) will be created in that path.

If you plan on using this in an actual application make sure that the placeholder values in your stub is unique for each setting.

Congratulations! We're done. Run the generated .exe, you should see the message "It works!"

I'd love some feedback on how I can do better. Am I going about this in the right way? If I'm not what should I do instead? Don't just tell me that this isn't the right way to code it. Tell me WHY and how I could improve. Thanks!

HFDone - May 27, 2019, 12:03 pm

<https://github.com/0xd4d/dnlib>

NYANxCAT - May 27, 2019, 1:09 pm

Great share. Thank you coh0rt.

Washi - May 27, 2019, 3:12 pm

Interesting way you have for finding the method you want to edit. It's also quite inefficient, as both the `TypeDefinition.ToString` and `MethodDefinition.ToString` are actually quite complex for something to just compare the name with. Furthermore, you do not break out of the loop once you found the method. If your assembly contains lots of methods and/or types, this can result in a lot of extra unnecessary work to be done by the program. It's also completely useless to convert the opcode to a string as well when you can just compare to the `Mono.Cecil.Cil.Code` enum value.

Instead, try to use the properties that Cecil actually defines, and perhaps use some Linq to make it actually more readable and reduce indentation:

PHP Code

```
<?php
using Mono.Cecil;
using Mono.Cecil.Cil;
...
var declaringType = module.MainModule.Types.First(t => t.Name == "Program");
var main = declaringType.Methods.First(m => m.Name == "Main");

// Or, if you're looking for the main method, just use:
// var main = module.EntryPoint;

foreach (var instruction in main.MethodBody.Instructions)
{
    if (instruction.OpCode.Code == Code.Ldstr && instruction.Operand is string s && s ==
"<your marker>")
    {
        // ...
    }
}
```

Nonetheless, nice read.

SaketSaket - May 28, 2019, 1:28 pm

Nice share... keep sharing... keep rocking...

tweety - May 29, 2019, 5:04 am

this is actually a popular method for generating builds for most software already, its not new.
i think you'd want to include tup/washi's suggestion into your op, as said above, its not really read friendly lol.

i had my fair share of using cecil a few years ago, used to use this method before but its redundant to use strings/booleans to let your build know what options to execute. a smart way to go about this method is to inject classes/methods that the user selects in the builder, and call them upon execution. why have 20 classes, tons of lines of code in your build if its not used at all? just baggage.

inject whole classes and methods instead of adjusting a string.

JUSTIN - May 29, 2019, 7:08 am

(May 27, 2019 12:03 PM)HFDone Wrote: [\[CITE\]](#) <https://github.com/0xd4d/dnlib>
why? cecil is perfectly fine for this...

(May 29, 2019 05:04 AM)tweety Wrote: [\[CITE\]](#) i had my fair share of using cecil a few years ago, used to use this method before but its redundant to use strings/booleans to let your build know what options to execute. a smart way to go about this method is to inject classes/methods that the user selects in the builder, and call them upon execution. why have 20 classes, tons of lines of code in your build if its not used at all? just baggage.

inject whole classes and methods instead of adjusting a string.

+1

example of a bad job can be found here:

<https://github.com/quasar/QuasarRAT/blob/master/Quasar.Server/Build/ClientBuilder.cs>

tweety - May 29, 2019, 8:07 am

(May 29, 2019 07:08 AM)JUSTIN Wrote: [\[CITE\]](#) why? cecil is perfectly fine for this...

+1

example of a bad job can be found here:

<https://github.com/quasar/QuasarRAT/blob/master/Quasar.Server/Build/ClientBuilder.cs>

i mean that source is something that walked out of 2003 lol

PHP Code

```
<?php
if (!r.Perform())
    throw new Exception("renaming failed");
```

you can tell its not well done lol..

most people link others to sources, its not always best... actually its never the best way to learn at all since people will pick up these kind of habits. regardless of its malware, webapps or whatever the case, should always follow the best standard to scale and have maintainable.

client a; yes the software says renaming failed... what do i do, help me debug
dev; uhh yes it means it failed, i can neither tell you why or have an available stack track to examine the issue, it just failed deal with it
dev2; hold on il try reproduce, ah it works fine here must be your issue

-- incoming 2 hours of teamviewer, precious time and money being wasted as well as inconvenience

its actually one of the reasons why i stepped away from hf and moved to other platforms that have good devs to talk to, hardly anyone competent left here tbh with a few exceptions ie tuppy/washie, 0x..

HFDone - May 29, 2019, 10:53 am

(May 29, 2019 07:08 AM)JUSTIN Wrote: [\[CITE\]](#) why? cecil is perfectly fine for this...

"I'd love some feedback on how I can do better. Am I going about this in the right way? If I'm not what should I do instead?"

JUSTIN - May 30, 2019, 6:29 am

(May 29, 2019 10:53 AM)HFDone Wrote: [\[CITE\]](#) "I'd love some feedback on how I can do better. Am I going about this in the right way? If I'm not what should I do instead?"

Mono.Cecil or dnlib are both "going about this the right way."

dnlib does not magically make this any better or worse...

(May 29, 2019 08:07 AM)tweety Wrote: [\[CITE\]](#) its actually one of the reasons why i stepped away from hf and moved to other platforms that have good devs to talk to, hardly anyone competent left here tbh
recommendations?

Aristocles - May 30, 2019, 4:06 pm

(May 29, 2019 08:07 AM)tweety Wrote: [\[CITE\]](#) its actually one of the reasons why i stepped away from hf and moved to other platforms that have good devs to talk to, hardly anyone competent left here tbh with a few exceptions ie tuppy/washie, 0x..

lol how pedent ... just if we look at your post activity you don't seems to be that away from hackforums, on the contrary ! ...

deiski - May 31, 2019, 2:48 pm

(May 29, 2019 08:07 AM)tweety Wrote: [\[CITE\]](#) i mean that source is something that walked out of 2003 lol

PHP Code

```
<?php
if (!r.Perform())
    throw new Exception("renaming failed");
```

you can tell its not well done lol..

most people link others to sources, its not always best... actually its never the best way to learn at all since people will pick up these kind of habits. regardless of its malware, webapps or whatever the case, should always follow the best standard to scale and have maintainable.

client a; yes the software says renaming failed... what do i do, help me debug
dev; uhh yes it means it failed, i can neither tell you why or have an available stack track to examine the issue, it just failed deal with it
dev2; hold on il try reproduce, ah it works fine here must be your issue

-- incoming 2 hours of teamviewer, precious time and money being wasted as well as inconvenience

its actually one of the reasons why i stepped away from hf and moved to other platforms that have good devs to talk to, hardly anyone competent left here tbh with a few exceptions ie tuppy/washie, 0x..

We're in the .NET subforum, can't expect much better being HackForums anyway.

Washi - June 1, 2019, 12:11 am

(May 29, 2019 05:04 AM)tweety Wrote: [\[CITE\]](#) ...

inject whole classes and methods instead of adjusting a string.

Yes this would be true if a feature is optional, and does not require any adjustment before injection.

However, some features either are always going to be present, or need special attributes set by the compiler (think of an API key or a custom message for a message box). For these scenarios, I think adjusting the existing (instead of generating) the CIL is perfectly fine.

I guess you could also go for another approach by embedding a configuration file in your application (such as in resources), and let the injected code read its options from there. I think choosing between those two is a matter of preference, and neither are going to be "the best" in my opinion.

Dolippo - June 1, 2019, 2:22 pm

Pretty good tutorial ! :D Yhank's a bunch, nice read