

Forum:[Hack Forums](#)

Thread: [Executing Shellcode In C# \(No Imports Required\)](#)

atom0s - May 3, 2017, 12:59 pm

Someone on Tuts4You made a request to make a way to invoke shellcode within C# without any imports to things such as VirtualAlloc / VirtualProtect etc. I came up with this method as all of C#'s built-in memory allocations via the Marshal class and similar all create with no execution access for protection purposes.

This method makes use of a memory mapped file which can be created with any protections that you want instead. This can be done all via the .NET frameworks built-in classes and such with no imports via DllImport. This is a simple example of creating a function via shellcode to get the PEB of a process. (Cheers to evlncrn8 for adjusting the shellcode from my original post to be more optimized.)

PHP Code

```
<?php
```

```
/**
 * C# Shellcode Example
 * (c) 2017 atom0s [atom0s@live.com]
 *
 * Demonstrates how to invoke shellcode within C# using a memory mapped file.
 */
```

```
namespace ShellcodeExample
```

```
{
    using System;
    using System.IO.MemoryMappedFiles;
    using System.Runtime.InteropServices;

    class Program
    {
        /// <summary>
        /// Function delegate to invoke the shellcode.
        /// </summary>
        /// <returns></returns>
        private delegate IntPtr GetPebDelegate();

        /// <summary>
        /// Shellcode function used to obtain the PEB of the process.
        /// </summary>
        /// <returns></returns>
        private unsafe static IntPtr GetPeb()
        {
            var shellcode = new byte[]
            {
                0x64, 0xA1, 0x30, 0x00, 0x00, 0x00,    // mov eax, dword ptr fs:[30]
                0xC3                                   // ret
            };
        }
    }
}
```

```

MemoryMappedFile mmf = null;
MemoryMappedViewAccessor mmva = null;

try
{
    // Create a read/write/executable memory mapped file to hold our shellcode..
    mmf = MemoryMappedFile.CreateNew("__shellcode", shellcode.Length,
MemoryMappedFileAccess.ReadWriteExecute);

    // Create a memory mapped view accessor with read/write/execute permissions..
    mmva = mmf.CreateViewAccessor(0, shellcode.Length, MemoryMappedFileAccess.
ReadWriteExecute);

    // Write the shellcode to the MMF..
    mmva.WriteArray(0, shellcode, 0, shellcode.Length);

    // Obtain a pointer to our MMF..
    var pointer = (byte*)0;
    mmva.SafeMemoryMappedViewHandle.AcquirePointer(ref pointer);

    // Create a function delegate to the shellcode in our MMF..
    var func = (GetPebDelegate)Marshal.GetDelegateForFunctionPointer(new IntPtr(pointer),
typeof(GetPebDelegate));

    // Invoke the shellcode..
    return func();
}
catch
{
    return IntPtr.Zero;
}
finally
{
    mmva?.Dispose();
    mmf?.Dispose();
}
}

/// <summary>
/// Entry point.
/// </summary>
/// <param name="args"></param>
static void Main(string[] args)
{
    var peb = GetPeb();
    Console.WriteLine("PEB is located at: {0:X8}", peb.ToInt32());
}
}
}

```

I am aware of the various assembly projects made for C# such as FASM and AssemblyJitter etc. however these projects all require external imports which was the goal of this idea, to not need any.

N3oNight - May 3, 2017, 1:04 pm

Short and simple piece of code but works well
good release buddy

gigajew - May 3, 2017, 7:17 pm

Didn't know you were on here atom0s. Nice post and also I like your Lua stuff from back in the day

Cecilia - May 4, 2017, 5:34 am

(May 3, 2017 07:17 PM)gigajew Wrote: [\[CITE\]](#) Didn't know you were on here atom0s. Nice post and also I like your Lua stuff from back in the day
Oh, hello there giga :)

Yes, good to have a completely managed solution.

HFDone - May 4, 2017, 8:23 am

the thing is that its work just with unsafe code so you cant use it with vb net
but theres a trick to get it in vb too
and needed framework 4

atom0s - May 4, 2017, 12:29 pm

(May 3, 2017 07:17 PM)gigajew Wrote: [\[CITE\]](#) Didn't know you were on here atom0s. Nice post and also I like your Lua stuff from back in the day
I pop on from time to time, haven't posted much at all lately though.

(May 4, 2017 08:23 AM)HFDone Wrote: [\[CITE\]](#) the thing is that its work just with unsafe code so you cant use it with vb net
but theres a trick to get it in vb too
and needed framework 4
I don't code in Vb.NET since it is basically trash. Plus the point of this was to be done in C# anyway. I suggest you move to a better language rather than sticking it out with Vb.NET if that is what you code in.

HFDone - May 6, 2017, 12:55 pm

i code in many languages. i say it cuz its the net section and even with C# you need to use unsafe code option.
as i said theres a trick that you can use it without unsafe code anyway its good to have option to do it without using apis.

Ast0r - May 7, 2017, 11:39 am

I actually didn't know that this was possible. Good release!

0xDEAD10CC - May 7, 2017, 1:18 pm

You are forgetting to release the acquired pointer within a CER region of your code. There's other issues with this if you tested on various operating systems, this is specific to x32 processes only; PEB isn't at fs:[0x30] in x64.

edit:

```
Code private delegate IntPtr GetPebDelegate();
```

```
private static unsafe IntPtr GetPeb()  
{  
    byte *pCode = null;  
    MemoryMappedFile mmf = null;
```

```

MemoryMappedViewAccessor mmva = null;

var asm = new[]
{
    new byte[]
    {
        0x64, 0xA1, 0x30, 0x00, 0x00, 0x00, // mov eax, dword ptr fs:30
        0xC3 // ret
    },
    new byte[]
    {
        0x65, 0x48, 0x8B, 0x04, 0x25, 0x60, 0x00, 0x00, 0x00, // mov rax, qword ptr gs:0x60
        0xC3 // ret
    }
};

var shellcode = asm[IntPtr.Size / 8];
RuntimeHelpers.PrepareConstrainedRegions();
try
{
    mmf = MemoryMappedFile.CreateNew("__PEB_Address", shellcode.Length,
MemoryMappedFileAccess.ReadWriteExecute);
    mmva = mmf.CreateViewAccessor(0, shellcode.Length,
MemoryMappedFileAccess.ReadWriteExecute);
    mmva.WriteArray(0, shellcode, 0, shellcode.Length);
    mmva.SafeMemoryMappedViewHandle.AcquirePointer(ref pCode);

    if (pCode != null)
    {
        return ((GetPebDelegate)Marshal.GetDelegateForFunctionPointer(new IntPtr(pCode),
typeof(GetPebDelegate)))(0);
    }
}
catch (Exception ex)
{
    Debug.WriteLine(ex.ToString());
}
finally
{
    if (mmva != null)
    {
        if (pCode != null)
        {
            mmva.SafeMemoryMappedViewHandle.ReleasePointer();
        }
        mmva.Dispose();
    }
    mmf?.Dispose();
}

return IntPtr.Zero;
}

```

atom0s - May 7, 2017, 5:00 pm

(May 7, 2017 01:18 PM)0xDEAD10CC Wrote: [\[CITE\]](#) You are forgetting to release the acquired pointer within a CER region of your code. There's other issues with this if you tested on various operating systems, this is specific to x32 processes only; PEB isn't at fs:[0x30] in x64.

I'm aware of the 32bit limitation, the purpose of the example was just to demonstrate the abilities on how to do this. And was pointed out on my Tuts4You post but forgot to mention it here. Granted I'm sure people that understand how this works and would understand how to use it can put the brain cells together to figure that out themselves.

As for full error checking and all that junk, that's up to the user to implement and more less anti-copy/pasta for the derps that land up erroring things.