

Forum:[Hack Forums](#)

Thread: [\[C#\] Memory Variable Protection](#)

codepoet - May 1, 2010, 4:24 pm

This just makes any byte[] variable you pass to the function protected in memory.
Cant be debugged.

.NET Already has this function, but i created a nice function that makes it easier to protect and unprotect.

How to use:

```
Code UTF8Encoding utf8 = new UTF8Encoding();
    //Basically create a byte array with some simple text bytes in it.
    byte[] ass = utf8.GetBytes("BYTEARRAY");

    //Protect the variable, by refering to it.
    ProtectVariable(ref ass);

    //Unprotect the variable.
    UnProtectVariable(ref ass);
```

Usings:

```
Code using System.Diagnostics;
using System.Security.Cryptography;
```

Code:

```
Code /// <summary>
    /// Protects the variable.
    /// Created by RUE, Please give credits.
    /// </summary>
    /// <param name="variable">The variable.</param>
    private static void ProtectVariable(ref byte[] variable)
    {
        //Check if the byte array is a multiple by 16.
        if ((variable.Length % 16) != 0)
        {
            /*
            * If byte array is NOT a multiple by 16, we make it go to the closest
            * multiple of 16. To do this we create a temp buffer array the size of
            * our variable plus the difference between it and the closest multiple of
            * 16, we fill the first part of the array with your variable data and the
            * rest remain blank. MAX variable size is < 256.
            */
            #region Check Difference

            //Checks to see the difference between the closest multiple of 16.
            int difference = 0;

            if (variable.Length < 16)
                difference = 16 - variable.Length;
            else if (variable.Length < 32)
```

```

        difference = 32 - variable.Length;
    else if (variable.Length < 48)
        difference = 48 - variable.Length;
    else if (variable.Length < 64)
        difference = 64 - variable.Length;
    else if (variable.Length < 128)
        difference = 128 - variable.Length;
    else if (variable.Length < 192)
        difference = 192 - variable.Length;
    else if (variable.Length < 256)
        difference = 256 - variable.Length;
    else if (variable.Length > 256)
        return;

```

```

#endregion

```

```

byte[] buffer = new byte[variable.Length + difference];
for (int i = 0; i < variable.Length; i++)
{
    buffer[i] = variable[i];
}
variable = buffer;
}

```

```

//protect the variable
ProtectedMemory.Protect(variable, MemoryProtectionScope.SameProcess);
Debug.WriteLine("Variable Protected!");
}

```

```

/// <summary>
/// Unprotects a variable.
/// Created by RUE, Please give credits.
/// </summary>
/// <param name="variable">The variable.</param>
private static void UnProtectVariable(ref byte[] variable)
{
    ProtectedMemory.Unprotect(variable, MemoryProtectionScope.SameProcess);
    Debug.WriteLine("Variable UnProtected!");
}

```

originally written by: RuE

If You Like this tutorial,
please check out my other tutorials by clicking ==> [here](#)

julian - May 1, 2010, 6:10 pm

That's cool. There's something similar specifically for Strings.

The second function can be simplified:

```

Code private static void ProtectVariable(ref byte[] variable)
{
    int remainder = variable.Length % 16;
    int paddedSize = variable.Length + (16 - remainder);

```

```
byte[] paddedVariable = new byte[paddedSize];
Buffer.BlockCopy(variable, 0, paddedVariable, 0, variable.Length);
variable = paddedVariable;
//protect the variable
ProtectedMemory.Protect(variable, MemoryProtectionScope.SameProcess);
Debug.WriteLine("Variable Protected!");
}
```

Buffer.BlockCopy is a fast way to copy data from one array of primitive types to another.

codepoet - May 1, 2010, 6:14 pm

very nice...thanks for shareing