

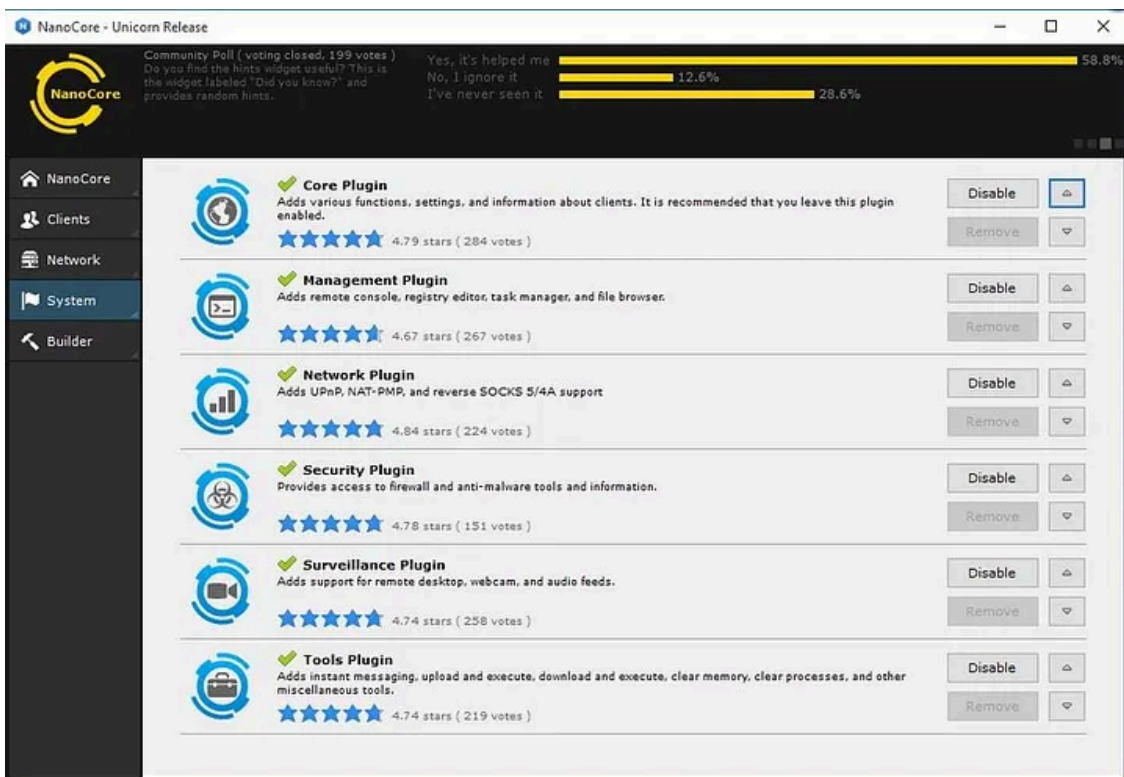
Decrypting NanoCore config and dump all plugins

By NexusFuzzy

Published: 2020-09-10 · Archived: 2026-04-05 14:12:16 UTC



Press enter or click to view image in full size



Even after the arresting of the developer NanoCore remains relevant due to its cracked versions

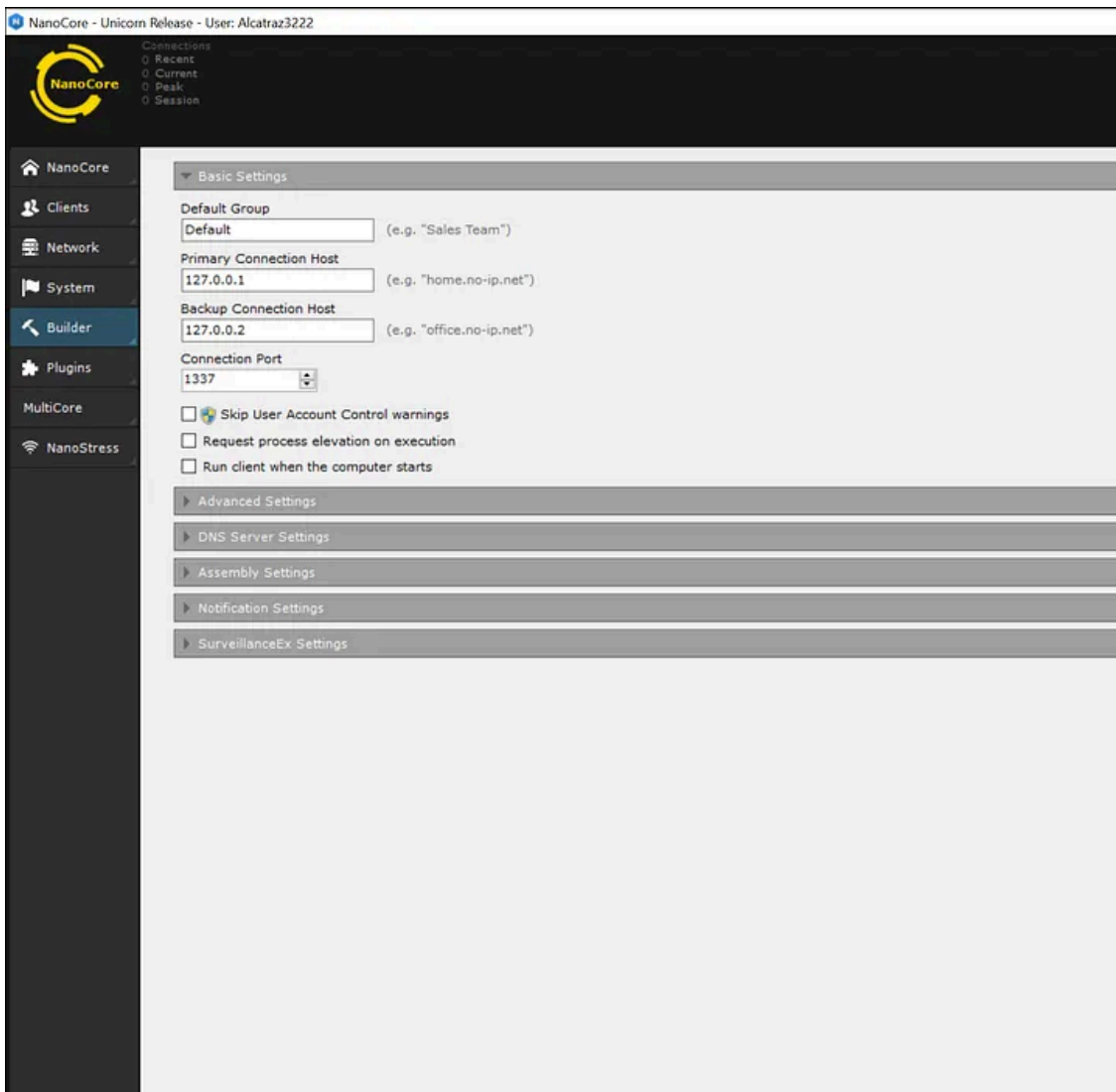
While the original author of NanoCore was arrested back in 2017 and [plead guilty](#), pirated copies of his creation keep floating around the internet making it available to even the most amateurish “threat actors”.

As stated in [my other article](#) about AgentTesla, the main focus for an analyst should be to squeeze out the IOCs (Indicators of compromise) as fast as possible to be able to move on to more interesting threats. That’s why I created a decryptor for NanoCore which does not only dump the config but also all used plugins.

The dissection

NanoCore comes with a comfortable to use Server/Builder which doesn’t need any technical experience at all:

Press enter or click to view image in full size



The builder for NanoCore

In cases of those builders, most of the time the “client” which gets distributed to victims remains the same, just the configuration gets embedded in one way or another so this is the first step in our process: Finding out how the plugins and config are getting embedded in the client.

For this reason, I built my own client and used de4dot to deobfuscate the created file:

Press enter or click to view image in full size

```
λ de4dot NC2_Client.exe
de4dot v3.1.41592.3405 Copyright (C) 2011-2015 de4dot@gmail.com
Latest version and source code: https://github.com/0xd4d/de4dot

Detected Eazfuscator.NET 3.3 (C:\Users\vladimir\Downloads\Unconfirmed 780070\NanoCore 1.2.2.0_Cracked By Alcatraz3222\NC2_Client.exe)
Cleaning C:\Users\vladimir\Downloads\Unconfirmed 780070\NanoCore 1.2.2.0_Cracked By Alcatraz3222\NC2_Client.exe
Renaming all obfuscated symbols
Saving C:\Users\vladimir\Downloads\Unconfirmed 780070\NanoCore 1.2.2.0_Cracked By Alcatraz3222\NC2_Client-cleaned.exe
```

As you may notice on the path I’m using one of the cracked versions of NanoCore

Afterwards you can open up the sample in dnSpy and begin reversing the sample as you normally would. During this process I found an interesting function:

Press enter or click to view image in full size

```
58 // Token: 0x060000CF RID: 207
59 [DllImport("kernel32.dll")]
60 public static extern IntPtr FindResourceEx(IntPtr intptr_0, int int_0, int int_1, short short_0);
61
62 // Token: 0x060000D0 RID: 208
63 [DllImport("kernel32.dll")]
64 public static extern IntPtr LoadResource(IntPtr intptr_0, IntPtr intptr_1);
65
66 // Token: 0x060000D1 RID: 209
67 [DllImport("kernel32.dll")]
68 public static extern int SizeofResource(IntPtr intptr_0, IntPtr intptr_1);
69
70 // Token: 0x060000D2 RID: 210
71 [DllImport("kernel32.dll")]
72 public static extern IntPtr LockResource(IntPtr intptr_0);
73
```

So what does this mean? Well, it's a strong indicator that the client is accessing its native resource(s) which could be the perfect place to embed a dynamic configuration. Let's dig deeper...

Get NexusFuzzy's stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

After analyzing where those functions are used I stumbled across this function:

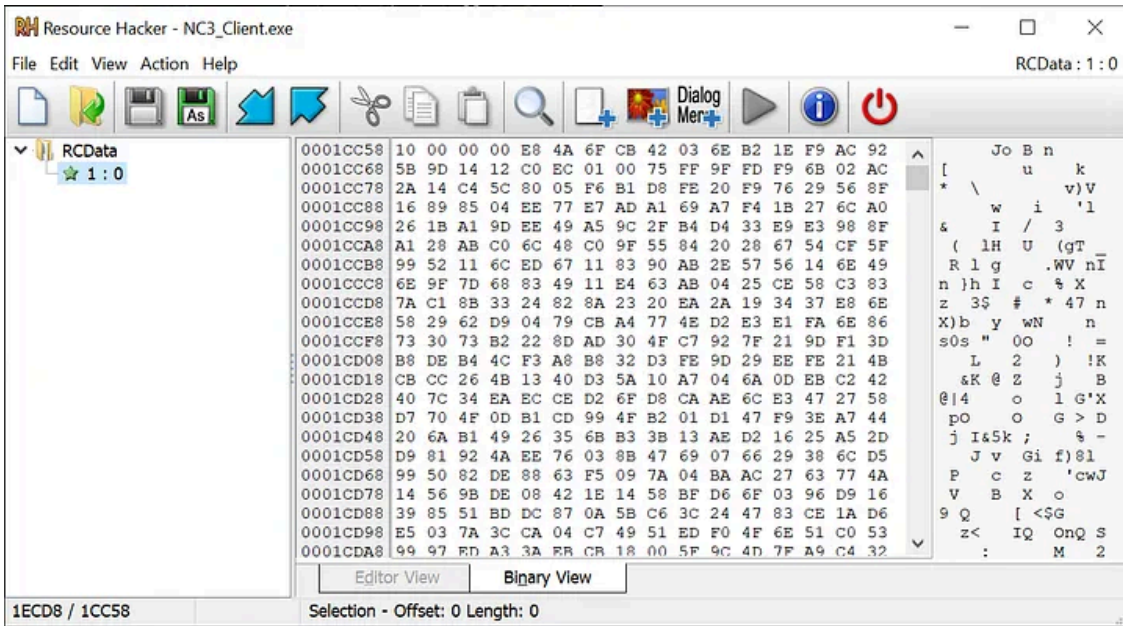
Press enter or click to view image in full size

```
429 // Token: 0x0600006D RID: 109
430 private static byte[] GetByteArrayFromResourceFile()
431 {
432     IntPtr intPtr = NativeFunctions.FindResourceEx(IntPtr.Zero, 10, 1, 0);
433     if (intPtr == IntPtr.Zero)
434     {
435         return null;
436     }
437     IntPtr intPtr2 = NativeFunctions.LoadResource(IntPtr.Zero, intPtr);
438     if (intPtr2 == IntPtr.Zero)
439     {
440         return null;
441     }
442     int num = NativeFunctions.SizeofResource(IntPtr.Zero, intPtr);
443     if (num == 0)
444     {
445         return null;
446     }
447     IntPtr intPtr3 = NativeFunctions.LockResource(intPtr2);
448     if (intPtr3 == IntPtr.Zero)
449     {
450         return null;
451     }
452     byte[] array = new byte[num - 1 + 1];
453     Marshal.Copy(intPtr3, array, 0, array.Length);
454     return array;
455 }
```

Looks like our assumption was right!

What this function does, is to find a resource by its number and afterwards loading its content into a byte array. Let's have a look at this resource with Resource Hacker, maybe we can find some clear text strings for a quick win:

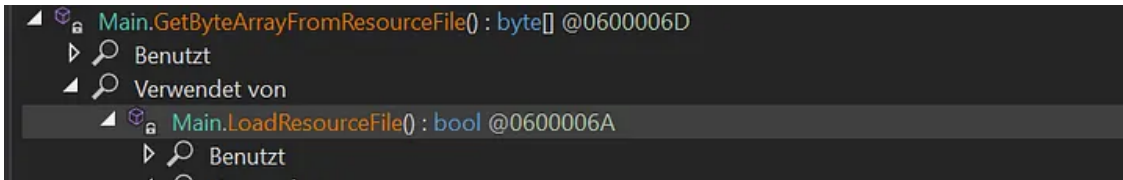
Press enter or click to view image in full size



Doesn't look really readable to be honest

Could it be that the resource is encrypted? Let's have a look! To see this we follow the rabbit hole and see where our function to load the resource is referenced:

Press enter or click to view image in full size



Don't mind that my functions have other names than the original binary

If we open up this function we can see some interesting things:

Press enter or click to view image in full size

```

356 // Token: 0x0600006A RID: 106
357 private static bool LoadResourceFile()
358 {
359     byte[] byteArrayFromResourceFile = Main.GetByteArrayFromResourceFile();
360     if (byteArrayFromResourceFile != null)
361     {
362         MemoryStream input = new MemoryStream(byteArrayFromResourceFile);
363         BinaryReader binaryReader = new BinaryReader(input);
364         byte[] byte_ = binaryReader.ReadBytes(binaryReader.ReadInt32());
365         Guid guidAttribute = Main.GetGuidAttribute(Assembly.GetExecutingAssembly());
366         Main.DecryptionKey = Main.Decrypt(byte_, guidAttribute);
367         Class13.InitializeEncryptorDecryptor(Main.DecryptionKey);
368         byte[] byte_2 = binaryReader.ReadBytes(binaryReader.ReadInt32());
369         object[] array = Class13.smethod_2(byte_2);
370         int num;
371         object[] array2 = new object[(int)array[num] - 1 + 1];
372         num++;
373         Array.Copy(array, num, array2, 0, array2.Length);
374         num += array2.Length;
375         object[] array3 = new object[(int)array[num] - 1 + 1];
376         num++;
377         Array.Copy(array, num, array3, 0, array3.Length);
378         Main.SetBuilderSettings(array3);
379         Main.AddPlugins(array2);
380         return true;
381     }
382     return false;
383 }

```

Let's examine the interesting parts:

First, it's getting the Guid of itself and passing this guid together with [4 byte integer](#) to some sort of decryption method:

Press enter or click to view image in full size

```

469 // Token: 0x06000070 RID: 112
470 private static byte[] Decrypt(byte[] byte_3, Guid guid_0)
471 {
472     Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(guid_0.ToByteArray(), guid_0.ToByteArray(), 8);
473     return new RijndaelManaged
474     {
475         IV = rfc2898DeriveBytes.GetBytes(16),
476         Key = rfc2898DeriveBytes.GetBytes(16)
477     }.CreateDecryptor().TransformFinalBlock(byte_3, 0, byte_3.Length);
478 }

```

The returning byte array is used to initialize a DES encryptor & decryptor using the byte array as Key and IV:

Press enter or click to view image in full size

```

45 // Token: 0x06000100 RID: 256
46 public static void InitializeEncryptorDecryptor(byte[] byte_0)
47 {
48     DESCryptoServiceProvider descryptoServiceProvider = new DESCryptoServiceProvider();
49     descryptoServiceProvider.BlockSize = 64;
50     descryptoServiceProvider.Key = byte_0;
51     descryptoServiceProvider.IV = byte_0;
52     Class13.Encryptor = descryptoServiceProvider.CreateEncryptor();
53     Class13.Decryptor = descryptoServiceProvider.CreateDecryptor();
54 }

```

This decryptor is then used to decrypt the config and plugins.

Let's summarize: First, it's getting its guid and uses it to get the key itself which was used to encrypt the resource file. The fact, that the Guid is created by the server randomly for every new client which is build lead me to the initial conclusion that you have to extract the Guid from the sample to dynamically dump the config — Well, I

was wrong. The guid changes but the key remains the same so you could skip this step but I left it in my tool since this may change from version to version and I would like to keep it dynamically.

After having realized how those steps are working together I was able to come up with a decryptor which you can find in [my GitHub repo called NanoDump](#).

Press enter or click to view image in full size

```
79 static void ProcessFile(string input, string output)
80 {
81     Guid guid;
82     Assembly a; = null;
83
84     // Extract the Guid from the sample which is used as Key and IV to get the actual decryption key from the
85     // resource file. This is not really necessary because the Guid differs from build to build but the underlying
86     // key remains constant. Since I could only test with one version it seems safer to do this step if this
87     // key ever changes
88     try
89     {
90         a = Assembly.LoadFile(input);
91         guid = new Guid(((GuidAttribute)a.GetCustomAttributes(typeof(GuidAttribute), false)[0]).Value);
92         Console.WriteLine("Successfully extracted Guid from file: " + guid.ToString());
93     }
94     catch (Exception ex)
95     {
96         Console.WriteLine("Couldn't extract Guid from sample: " + ex.ToString());
97         return;
98     }
99
100     // We now load the native resource
101     byte[] resource = GetResourceFromExecutable(input, "1", "RCData");
102     if (resource != null)
103     {
104         MemoryStream ms = new MemoryStream(resource);
105         BinaryReader binaryReader = new BinaryReader(ms);
106         byte[] byte_ = binaryReader.ReadBytes(binaryReader.ReadInt32());
107         byte[] key = GetKeyFromResourceFile(byte_, guid);
108         InitializeEncryptorDecryptor(key);
109
110         byte[] byte_2 = binaryReader.ReadBytes(binaryReader.ReadInt32());
111
112         object[] array = smethod_2(byte_2);
113         int num = 0;
114         object[] plugins = new object[(int)array[num]];
115         num++;
116         Array.Copy(array, num, plugins, 0, plugins.Length);
```

Source of NanoDump

As you can see, I replicated the necessary steps to reach the goal to decrypt and dump the details of NanoCore.

If you have any questions, feel free to drop me a [message on Twitter](#) or open a GitHub issue!