Free Nitro - CTF writeup

NahamCon CTF 2022

Summary

sha256 : 7A83115AB46BA6A3C237D78F32BD3386FF4D4D7CD7B06AD731FE8071B2246278

Free_nitro.exe is a .net binary that was part of the malware section on NahamCon CTF 2022. It has three infection stages:

- Free_nitro.exe: The binary contains two principal payloads represented as hexadecimal data, each payload is read, decrypted, loaded and executed from memory.

  Lib Array (first payload - ManeySubLib.dll) contains data in PE32+ executable (DLL) format and File Array (second payload - Client.exe) contains data in PE32 executable (GUI) format.

- ManeySubLib.dll: Data is loaded by free_nitro.exe using reflective load and it runs the File Array (Client.exe) payload.
- Client.exe: Contains flag for this challenge.

In this report I just will be explaining how I got the flag for this challenge. **The real malware behavior in out of scope.**

Important Note: Free_nitro.exe binary executes real malware behavior therefore, detons it at your on risk!

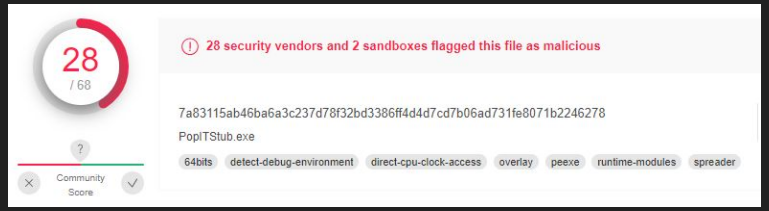
Malware Composition

Free_nitro.exe consist of the following components:

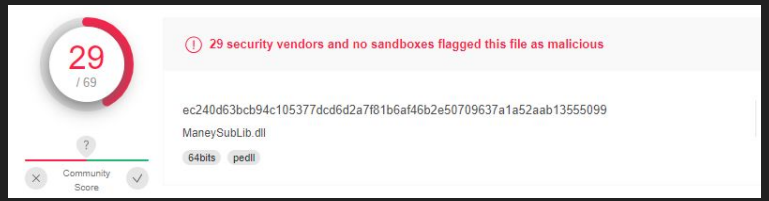
File Name	Sha256 hash
free_nitro.exe	7A83115AB46BA6A3C237D78F32BD3386FF4D4D7CD7B06AD731FE8071B2246278
ManeySubLib.dll	EC240D63BCB94C105377DCD6D2A7F81B6AF46B2E50709637A1A52AAB13555099
Client.exe	98155C900F39FAAC6A42133850329CAABC8BF4EBCC90D5037F481AAA86C7240F

Basic Static Analysis

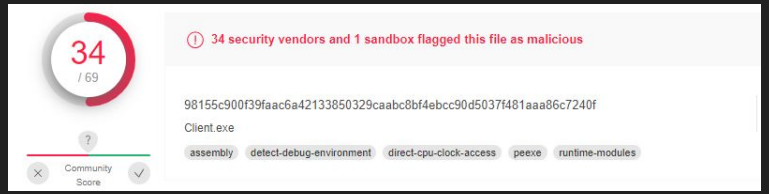
Free_nitro.exe (PopITStub.exe)



ManeySubLib.dll



Client.exe



Basic Static Analysis

1) Let's use floss to extract strings from the binary.

```
C:\Users\Flare\Desktop
λ FLOSS.exe -n 8 free_nitro.exe > output.txt
```

The output shows a large hexadecimal data (ManeySubLib.dll and Client.exe), passwords for decrypting hexadecimal data and regex patterns:

```
<libArr>0B9F1C141414141014D8696368CFD101EC5DC2A62871CDE698B95888F
<fileArr>D84CCFCF75A491A5C7C4A3A0B5F5C72B9DA8B3DB12B2B83CA1310A2B2B
<pass1>276</pass1><pass2>199</pass2><autorun>false</autorun>
```

```
pX4v8jNB2NMVUYaPjR.rEito1YK0HtJZSVnBy
rEiYto1K0
<pass1>(.*?)</pass1><pass2>(.*?)</pass2><autorun>(.*?)</autorun>
<libArr>(.*?)</libArr><fileArr>(.*?)</fileArr>
```

Advanced Static Analysis

1) Let's use ILSpy to decompile free_nitro.exe (PopITStub) and check the most important pieces of code:

   The following code reads the binary data using File.ReadAllBytes(perem.path) and use regex pattern to extract the right piece of hexadecimal data. getStr() is use to extract the decryption password and getArr is use to extract the libArr (ManeySubLib.dll) and fileArr (Client.exe) data.

```
public class readers
{
    public static string getStr(int massive)
    {
        byte[] bytes = File.ReadAllBytes(perem.path);
        return Regex.Match(Encoding.ASCII.GetString(bytes), "<pass1>(.*?)</pass1><pass2>(.*?)</pass2><autorun>(.*?)</autorun>").Groups[massive].Value;
    }

    public static byte[] getArr(int massive)
    {
        byte[] bytes = File.ReadAllBytes(perem.path);
        return decrypters.hextobyte(Regex.Match(Encoding.ASCII.GetString(bytes), "<libArr>(.*?)</libArr><fileArr>(.*?)</fileArr>").Groups[massive].Value);
    }
}
```

Advanced Static Analysis

```
public class decrypters
{
    public static byte[] hextobyte(string hex)
    {
        int length = hex.Length;
        byte[] array = new byte[length / 2];
        for (int i = 0; i < length; i += 2)
        {
            array[i / 2] = Convert.ToByte(hex.Substring(i, 2), 16);
        }
        return array;
    }

    public static byte[] decrypt(byte[] data, int pass)
    {
        for (int i = 0; i < data.Length; i++)
        {
            data[i] = (byte)(data[i] ^ pass);
        }
        using MemoryStream stream = new MemoryStream(data);
        using GZipStream gZipStream = new GZipStream(stream, CompressionMode.Decompress);
        using MemoryStream memoryStream = new MemoryStream();
        gZipStream.CopyTo(memoryStream);
        return memoryStream.ToArray();
    }
}
```

Hextobyte() takes the hexadecimal data gotten from getArr() (see previous slide) and convert it an array of bytes.

decrypt() is use to decrypt the array of bytes gotten from Hextobyte() and xoring each byte with password extracted from getStr() (see previous slide.)

Advanced Static Analysis

```
public class oncole : get_main
{
    public override void unload()
    {
        MethodInfo method = Assembly.Load(perem.lib).GetType("pX4v8jNB2NMVUYaPjR.rEito1YK0HtJZSVnBy").GetMethod("rEiYto1K0");
        try
        {
            method.Invoke(null, new object[4]
            {
                0,
                perem.file,
                true,
                true
            });
        }
        catch
        {
            method.Invoke(null, new object[4]
            {
                1,
                perem.file,
                true,
                true
            });
        }
    }
}
```

This code use system.reflection.assembly.load to load and execute the library code in memory even running the fileArr (Client.exe) payload.

Advanced Static Analysis

2) I wrote a PoC to extract, decrypt and recreate the ManeySubLib.dll and Client.exe files (see Poc code at /PoC).

3) Let's use file command to check our two files:

```
C:\Users\Flare\Desktop
λ file ManeySubLib.dll
ManeySubLib.dll: PE32+ executable (DLL) (console) x86-64 Mono/.Net assembly, for MS Windows

C:\Users\Flare\Desktop
λ file Client.exe
Client.exe: PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows
```

Advanced Static Analysis

4) After spending some time checking our two files, I got a Flag string on Client.exe binary. But the flag was encrypted using AES256.

```
public static string Flag = "mZzroGSIkpZlwvCwLG0PHQMXzjphDowlbeBayjWJhmYPJ5KiQeUAbcv9SzTnLGpr3uYQ0VvZ02rGlxz71tOXMemdK1DKKY6uX2QfUJW+WlDPcLi1u48xBrhmDcpRaK1G";
```

5) Analysing the source code I saw how Client.exe decrypts data:

- Create a master Key.

```
public static string Key = "S1hNZ2tQdFJlRkVIWXhKczRMZEIwRmRQVmg3WGxDNEQ=";
```

```
public Aes256(string masterKey)
{
    if (string.IsNullOrEmpty(masterKey))
    {
        throw new ArgumentException("masterKey can not be null or empty.");
    }
    using Rfc2898DeriveBytes rfc2898DeriveBytes = new Rfc2898DeriveBytes(masterKey, Salt, 50000);
    _key = rfc2898DeriveBytes.GetBytes(32);
    _authKey = rfc2898DeriveBytes.GetBytes(64);
}
```

Advanced Static Analysis

- Use a master key to decrypt data:

```
public byte[] Decrypt(byte[] input)
{
    if (input == null)
    {
        throw new ArgumentNullException("input can not be null.");
    }
    using MemoryStream memoryStream = new MemoryStream(input);
    using AesCryptoServiceProvider aesCryptoServiceProvider = new AesCryptoServiceProvider();
    aesCryptoServiceProvider.KeySize = 256;
    aesCryptoServiceProvider.BlockSize = 128;
    aesCryptoServiceProvider.Mode = CipherMode.CBC;
    aesCryptoServiceProvider.Padding = PaddingMode.PKCS7;
    aesCryptoServiceProvider.Key = _key;
    using (HMACSHA256 hMACSHA = new HMACSHA256(_authKey))
    {
        byte[] a = hMACSHA.ComputeHash(memoryStream.ToArray(), 32, memoryStream.ToArray().Length - 32);
        byte[] array = new byte[32];
        memoryStream.Read(array, 0, array.Length);
        if (!AreEqual(a, array))
        {
            throw new CryptographicException("Invalid message authentication code (MAC).");
        }
    }
    byte[] array2 = new byte[16];
    memoryStream.Read(array2, 0, 16);
    aesCryptoServiceProvider.IV = array2;
    using CryptoStream cryptoStream = new CryptoStream(memoryStream, aesCryptoServiceProvider.CreateDecryptor(), CryptoStreamMode.Read);
    byte[] array3 = new byte[memoryStream.Length - 16 + 1];
    byte[] array4 = new byte[cryptoStream.Read(array3, 0, array3.Length)];
    Buffer.BlockCopy(array3, 0, array4, 0, array4.Length);
    return array4;
}
```

Advanced Static Analysis

6) I wrote another PoC to decrypt the flag (see Poc code at /PoC) and I got the flag!

```
C:\Users\User\source\repos\flagDecrypter\flagDecrypter\bin\Debug>flagDecrypter.exe
flag{8b988b859588f2725f0c859104919019}
```