


Unfolding Remcos RAT- 4.9.2 Pro

 infosecwriteups.com/unfolding-remcos-rat-4-9-2-pro-dfb3cb25bbd1

Osama Ellahi

November 26, 2023



Executive Summary

SHA256 hash:

```
| 2e5c4d023167875977767da513d8889f1fc09fb18fdadfd95c66a6a890b5ca3f
```

Remcos is a commercially available Remote Access Tool (RAT) marketed for legitimate use in surveillance and penetration testing. However, it has been leveraged in various unauthorized hacking initiatives. When deployed, Remcos establishes a backdoor, allowing comprehensive remote control over the affected system. The tool is a product of BreakingSecurity, a company specializing in cybersecurity solutions.

Hackers are getting smarter by using **tricks like hiding their code and adding fake code**, which makes it harder for security experts to figure out how their attacks work. They're using things like image files and compression to disguise their activities.

YARA signature rules are attached in Appendix A. Malware sample and hashes have been submitted to VirusTotal for further examination.

High-Level Technical Summary

Remcos is an advanced remote access tool that breaks into computers using a series of hidden codes, starting with a malicious file which can be delivered from mail or dropper. It **cleverly disguises its next steps within an image file**, and then uses another DLL to make sure it stays on the computer even after it's restarted. Remcos can record keystrokes to steal passwords and other

private information, **which it logs into a file**. It stays in contact with the hacker's server to send out this stolen information and to get new orders, allowing the hacker to keep a close watch and control over the infected computer.

Malware Composition

This composition of remcos consists of the following components:

```
2e5c4d023167875977767da513d8889f1fc09fb18fdadfd95c66a6a890b5ca3f
```

```
Embedded_Remcos.exe
```

In a C# dropper, there's a sneaky way that malware developers are hiding bad code. They put this code inside **the InitializeComponent()** method. This method is normally used just for setting up how the app looks, like buttons and menus. But now, it's being used to hide something harmful. The tricky part is that this bad code looks just like regular setup code, so it's hard to spot. It's like hiding something bad inside something good, so people don't notice it.

This makes it hard to find and fix the problem. It's a clever trick by hackers, and it shows how they can use parts of an app we usually trust to do sneaky things.

It is extracting a byte array from a resource, possibly a file or other data embedded in the application and generating another byte array from a hard coded string.

The code is setting up a user interface for a form and then performing an operation on a **data resource ("SHP")** using a **generated key**.

The `_data` before the encryption looks like this.

The for loop processes the `Data_` array in a complex way. It goes through each byte of `Data_` and modifies it based on a calculation involving both `Data_` and `KeyGen`.

The calculation inside the loop involves bitwise XOR (^), addition, and modulo operations. It appears to be some form of data manipulation or encryption/decryption, where `Data_` is being altered using the `KeyGen` byte array.

First, a `MethodBase` object named `methodBase` is assigned the value `kb`. The `MethodBase` class in C# is part of the reflection namespace and is used to discover information about methods (like constructors and other methods) at runtime.

Then, an array of objects named `array` is created and initialized with string values. This array includes `this.VC`, `this.VR`, and the literal string "Boilerplate". `VC` and `VR` are private string fields of the class, initialized to "57775972" and "6C7978", respectively. Therefore, the array contains these two strings along with "Boilerplate".

Finally, the `Invoke` method on `methodBase` is called, passing `obj` and `array` as arguments. This means the method represented by `methodBase` is being executed with `obj` as the target and the string array as the parameters.

Before the inoke there was binary loaded successfully in modules.

And if look closely it in kb.Fullname it is calling dr,hA.wP method in Ben dll.

Ben DLL

By adding breakpoint after loading from module we catch the debugger.

The code performs image processing, uses reflection to invoke a method, and dynamically loads an assembly from a byte array. This kind of operation is typical in applications that need to manipulate images, dynamically execute code, and potentially load plugins or modules at runtime.

Sleep for 16sec

- new MemoryStream(array2): This creates a new MemoryStream object using array2 as its buffer. array2 is assumed to be a byte array (byte[]) that contains data compressed using the GZip algorithm. The MemoryStream is a stream based on a memory buffer, allowing for reading from and writing to memory.
- new GZipStream(...): This creates a new GZipStream object. The GZipStream class is used to compress and decompress data in the GZip data format. In this case, it's constructed with the previously created MemoryStream and the CompressionMode.Decompress. This indicates that the GZipStream should be used for decompression, i.e., to decompress the data contained in array2.

It's part of a process involving dynamic loading and reflection. It reads and possibly processes data from a MemoryStream, uses that data to load an assembly or access its contents, and then retrieves a specific type from that assembly.

Rd is designed to dynamically load a .NET assembly from a byte array, denoted as `\u0020`. It employs a nested, infinite loop structure with a switch statement for control flow. Initially, it attempts to load the assembly using `Assembly.Load(\u0020)`. The code's flow is influenced by the result of `global::dr.hA.EV()`, a method call whose purpose is unclear. If EV() returns a non-null value, the method exits the loop prematurely via a go to statement. The method's coding style, characterized by unconventional variable naming and complex looping, suggests a potential for obfuscation, possibly to conceal the actual functionality or make reverse-engineering more challenging.

ReactionDiffusion

After loading assembly we get a new binary in modules with the name of **ReactionDiffusion**.

Then it disposes the "memorystream" which means the work of the memory stream is done here. Probably it will now move on to the next binary.

After that it also dispose the gzip stream which was used to get the binary.

Now let's track where it would go next in **ReactionDiffusion**. If we investigate the object where it is pointing its type show us the destination namespace and class.

Since there were no method calls from previous binary. So, we created break point at constructor at it hit exact on it.

There was nothing useful in **ReactionDiffusion** there, maybe it was all decoy code. Let's see what next the **Ben binary** does, in case 8 it gets bitmap from resources.

RS Method

The RS method in C# is designed to retrieve a **Bitmap image from resources using reflection and obfuscated code patterns**. It starts by declaring a **ResourceManager to access embedded resources**, using a dynamically constructed resource name from the first-string parameter, `\u0020`. This parameter, along with a similarly named second parameter, is used in a nested, infinite loop structure with a switch statement. Bitmap is obtained by the method `global::dr.hA.rY`, which likely extracts the image from the resources. The control flow includes checks with `global::dr.hA.EV()` and `global::dr.hA.m3()`, whose purposes are unclear, but they seem to influence the flow and decision-making within the method. The use of obfuscated names (like `\u0020`) and complex control flow suggests an intent to mask the code's functionality or purpose.

Loading the assembly from byte array

1. It defines a private static method named `Rd` that takes a byte array `\u0020` as its parameter.
2. It initializes an integer variable `num` with the value 1.
3. Inside an infinite loop (for (;;)), the code performs the following actions: a. It declares a variable `num2` and assigns it the value of `num`. b. It enters another loop (for (;;)). c. Within the inner loop, there is a switch statement with two cases:
 - Case 1:
 - It attempts to load an assembly using `Assembly.Load(\u0020)`, where `\u0020` represents the byte array passed as a parameter to the method.
 - If the assembly is successfully loaded, it sets `num2` to 0.
 - It then checks whether `global::dr.hA.EV()` is not null. If it's not null, the code proceeds to the `Block_1` label.
 - If `global::dr.hA.EV()` is null, it effectively exits the loop and returns the loaded assembly.
 - Default case:

If none of the cases match, it returns the assembly variable, which would have been assigned earlier in the code.

d. The `Block_1` label is used to indicate the point where the code should continue if `global::dr.hA.EV()` is not null. It doesn't contain any specific code logic in the provided snippet.

Tyrone

It looks like another binary is coming. **Another DLL loaded in modules with the name Tyrone.**

Invoking `AJBqklj3Jn` from `tyorne { YcMqTyPiynJnoycycL.MhMHeAYqAZ6AJWSu3o}`

This is more obfuscated than previous binaries.

Checking for the presence of a named mutex, which may be used by malware for synchronization or coordination purposes. **“wnmJOXavioKPdkNYG”**

It tried to open but since if there is no **mutex** it goes to exception. If it exists it will end itself in second line.

Creating Mutex

It creates a new Mutex object with the name **“wnmJOXavioKPdkNYG”**. Mutexes are synchronization primitives used to control access to shared resources among multiple threads or processes.

This was all to get path of appdata and then append it with **“EiHjExP.exe”**.

“C:\Users\username\AppData\Roaming\EiHjExP.exe”

Check if not there Copy it.

Change Directory Permission

It adds access control entries to the **directorySecurity** object using the **MhMHeAYqAZ6AJWSu3o.PR6qMi9p2U** method. These entries seem to define permissions for specific file system rights (e.g., Read, **ReadAndExecute**, Delete, Write, etc.) with different access control types (e.g., Allow, Deny). The permissions are set for various inheritance flags and propagation flags, which determine how permissions are inherited by child objects.

It removes **“currentuser”** security to change file and write permission.

As you can see the permission are denied now

Remcos is doing this because it makes it safe from being changed or deleted from disc.

Then it gets a base64 encoded text fetched from modules of this **tyrone** binary with this code.

I decode this string from <https://www.base64decode.org/> and it turns out that it is xml.

There is code for decoding also in the remcos.

Then this function is called to play with Microsoft Security. This function decodes the text which was fetched from module.

It then creates a new process, assign a **new stratinfo** with it and give file name **“powershell”** which it gets from the module. In arguments of process, it gives **@”Add-MpPreference -ExclusionPath “”C:\Users\shaddy\AppData\Roaming\EiHjExP.exe”””**

Set process’s window hidden.

Windows Exclusion

It will be added to the exclusion but keep in mind that I was running it from admin, if not performing analysis from admin it will be able to add since so far there was not privilege escalation performed.

Path.GetTempFileName(); it will return a string that represents a unique temporary file name. This file name is generated using a combination of a temporary directory path and a unique identifier, making it highly unlikely to clash with other temporary files in the system.

It gets the identity of current user, exe path to update the xml. In the breakpoint it is updating the xml and saving it in text variable.

The clean xml code.

Persistence

After that it is writing all xml in tmp file.

It then loads the command of scheduling task from modules and **sets startupinfo** of process. Process is executed with window style hidden, Filename **“schtask.exe”** and with following arguments.

```
@"/Create /TN ""Updates\EiHjExP"" /XML  
""C:\Users\shaddy\AppData\Local\Temp\tmp66E3.tmp""
```

This command appears to be creating a new scheduled task with the name “Updates\EiHjExP” and configuring it using an XML file located at **“C:\Users\shaddy\AppData\Local\Temp\tmp66E3.tmp.”**

It is triggering the exe after every system restarts.

Then it deletes the tmp file.

After that it loads new assembly “xF7siMsac” from its resource manager.

It is injecting this final binary and executing it. Let’s see its injection inside process hacker.

Another binary which is extracted and DE obfuscated from resources.

Remcos / 5thstage

After saving the binary from **lu0020** it looks exactly like client agent built from the original **remcos** agent from <https://breakingsecurity.net/remcos/>. The logo is also the same, but its signature was not present in any online threat intelligence.

This final stage was developed in c++ language. And before analysis when we perform strings filter there was something linking to remcos, this pattern comes almost in every remcos rat.

Now let’s start the debugger to look more into it. We can see some more identifications.

It starts with calling **GetAddrInfoW API** which is pointing to **rungmotors20.ddns.net:60247**.

GetAddrInfoW is a Windows API function that is used for network operations. It's part of the Windows Sockets (Winsock) API and is typically called to resolve network addresses or to perform name resolution, converting a hostname like a domain or a URL into an IP address that can be used to establish network connections.

If running from admin privileges, it creates a directory [C:\\ProgramData\\remcos] using CreateDirectoryW API.

CreateDirectoryW is a function in the Windows API that is used to create a new directory. The W at the end of CreateDirectoryW indicates that this function uses wide characters (Unicode), as opposed to CreateDirectoryA, which uses ANSI characters.

After creating Directory, it creates file with name logs.dat using **CretaeFileW** api.

There are privileges check also it is handling both cases smoothly. It is just paths which it used separately.

While executed from admin it uses [C:\\ProgramData\\remcos folder]. It creates thread and that thread in loop performs these steps.

If executed from normal permission, it uses

[C:\\Users\\username\\Local\\VirtualStore\\ProgramData\\remcos\\logs.dat]

It sets its mark on the system in registry. It sets exepath, licence and time for thread.

Patching TLS

All traffic was encrypted so we must check what is being sent. There was TLS check which was on in our client rate.

Since we cannot see what it is sending to server, because of TLS flag is on. It will send all the traffic encrypted. After patching this, we can analyze the traffic.

After finding the check I was able to turn off the TLS and see all the traffic clearly. It was sending the device identification after every few seconds to server.

This was sample data that rat was sending.

```
$ KRemoteHost||DESKTOP-002IHON/shaddy||US||Windows 10 Enterprise (64
bit)|||8588939264||4.9.2
Pro||C:\\ProgramData\\remcos\\logs.dat||C:\\Users\\shaddy\\Desktop\\5thstage.exe|||5thstage.exe
— PID: 3308 — Module: 5thstage.exe — Thread: Main Thread 6232 — x32dbg
[Elevated]||1||47||48556593||1||rungmotors20.ddns.net||Rmc-
ZT6SIL||0||C:\\Users\\shaddy\\Desktop\\5thstage.exe||12th Gen Intel(R) Core(TM) i7-
12700KF||Exe|||
```

Clipboard and Process recording

Inside the thread it was performing three major activities because the one who built it, he/she only want to record clipboards, records keylogging and setting some registries. It records all the clipboards data inside the same logs.dat file. Only it appends [Text copied to clipboard] at initial and [End of clipboard] at end.

It also keeps recording the process which spawns, its architecture, its user access and all the keystrokes also.

Rules & IOCs

Yara Rules

```
rule remcos_pro_4_9_2
{
meta:
author = "Osama Ellahi"
description = "Remcos RAT 4.9.2 pro version from breakpoint"
strings:
$string_match1 = "© by P.J. Plauger, licensed by Dinkumware, Ltd. ALL RIGHTS RESERVED"
ascii fullword
$string_match2 = "\tRemcos v" ascii fullword
$string_match3 = "BreakingSecurity.net" ascii fullword
$string_match4 = "4.9.2 Pro" ascii fullword
$string_match6 = "[Text pasted from clipboard]" ascii fullword
$string_match7 = "[End of clipboard]" ascii fullword
$string_match8 = "[End of clipboard]" ascii fullword
$string_match9 = "[Text copied to clipboard]" ascii fullword
$string_match11 = "Offline Keylogger Started" ascii fullword
$string_match12 = "Offline Keylogger Stopped" ascii fullword
$string_match13 = "Online Keylogger Started" ascii fullword
$string_match14 = "Online Keylogger Stopped" ascii fullword
$string_match15 = "Remcos restarted by watchdog!" ascii fullword
$string_match16 = "Watchdog module activated" ascii fullword
$string_match17 = "Watchdog launch failed!" ascii fullword
$string_match18 = "[Chrome StoredLogins not found]" ascii fullword
$string_match19 = "[Chrome StoredLogins found, cleared!]" ascii fullword
$string_match20 = "[Chrome Cookies not found]" ascii fullword
$string_match21 = "[Chrome Cookies found, cleared!]" ascii fullword
$string_match22 = "[Firefox StoredLogins not found]" ascii fullword
```

\$string_match23 = “[Firefox Cookies not found]” ascii fullword

\$string_match24 = “[Firefox cookies found, cleared!]” ascii fullword

\$string_match25 = “[Firefox StoredLogins Cleared!]” ascii fullword

\$string_match26 = [IE cookies not found] ascii fullword

\$string_match27 = [IE cookies cleared!] ascii fullword

\$string_match28 = [Cleared browsers logins and cookies.] ascii fullword

\$string_paths1 = “\\AppData\\Local\\Google\\Chrome\\User Data\\Default\\Cookies” ascii fullword

\$string_paths2 = “\\AppData\\Roaming\\Mozilla\\Firefox\\Profiles\\” ascii fullword

\$string_paths3 = “Software\\Microsoft\\Windows\\CurrentVersion\\Explorer\\User Shell Folders” ascii fullword

\$string_paths4 = “Software\\Microsoft\\Windows\\CurrentVersion\\Run\\” ascii fullword

\$string_paths5 = “\\AppData\\Local\\Google\\Chrome\\User Data\\Default\\Login Data” ascii fullword

\$string_paths6 = “Software\\Microsoft\\EventSounds\\Sounds” ascii fullword

\$string_paths7 =
“System\\CurrentControlSet\\Control\\MediaProperties\\PrivateProperties\\Joystick\\Winmm”
ascii fullword

\$string_commands1 = “CreateObject(“WScript.Shell”).Run “cmd /c \\”” ascii fullword

\$string_commands2 =
“CreateObject(“Scripting.FileSystemObject”).DeleteFile(Wscript.ScriptFullName)” ascii fullword

\$string_commands3 = “\\AppData\\Local\\Google\\Chrome\\User Data\\Default\\Login Data”
ascii fullword

\$string_commands4 = “/k %windir%\\System32\\reg.exe ADD
HKLM\\SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Policies\\System /v EnableLUA /t
REG_DWORD /d 0 /f” ascii

\$string_url1 = “http://geoplugin.net/json.gp” ascii fullword

\$string_url2 = “rungmotors20.ddns.net” ascii fullword

condition:
uint16(0) == 0x5a4d and filesize < 600KB and filesize >200KB

```
and
(
any of ($string_url*)
or
3 of ($string_paths*)
or
5 of ($string_match*)
)
}
```

Callback URLs

URL: rungmotors20.ddns.net Port: 60247

URL: hxxp://geoplugin.net/json.gp Port: 443

IOC

1st

SHA256 —

2e5c4d023167875977767da513d8889f1fc09fb18fdadfd95c66a6a890b5ca3f

2nd

MD5 —

3125f77575829f3b710f5a15912dec20 *stage2.dll

SHA256 —

1cc58fba1d1b4c7e0b9d752ea7f03fa3c312ae2fc53796d5b3acea98e6ea3c0e *stage2.dll

3rd

SHA256 —

d01f3dea3851602ba5a0586c60430d286adf6fcc7e17aab080601a66630606e5 *stage3.dll

MD5 —

579197d4f760148a9482d1ebde113259 *stage3.dll

4th

SHA256 —

c5928572e371b0a5d3109d0a7431ca9e064216beb858f04dc8d0140ccaf44b84 *Tyrone.dll

MD5 —

dd76e11ff9b96efdcf3cd377126c8d96 *Tyrone.dll

5th

SHA256 —

f55fc4f4e1bcbe957d20750f56cd98869c717c18c14c8b6d42698557b254ad51 *5thstage.mal

MD5 —

dc05d4f2864dfafa9b91e8e0d79840e3 *5thstage.mal

References

<https://www.joesandbox.com/analysis/1339230/0/html>

<https://www.jaiminton.com/reverse-engineering/remcos#part-2-decompiling-binary>