

# PowerPoint and Custom Actions

POSTED ON FEBRUARY 23, 2016 | BY **SEAN WILSON** IN **PHISHING**

SHARE THIS:



We've recently observed a Phishing attack which uses PowerPoint Custom Actions instead of macros to execute a malicious payload. Although using PowerPoint attachments is not new, these types of attacks are interesting as they generally bypass controls that assert on macro enabled Office attachments.

- An attacker creates a new PowerPoint presentation and inserts a malicious script/executable. The inserted file is embedded as an OLE object.
- A Custom Action is created set to trigger 'With Previous' with the actionfigur 'Activate Contents' to execute the embedded OLE object.

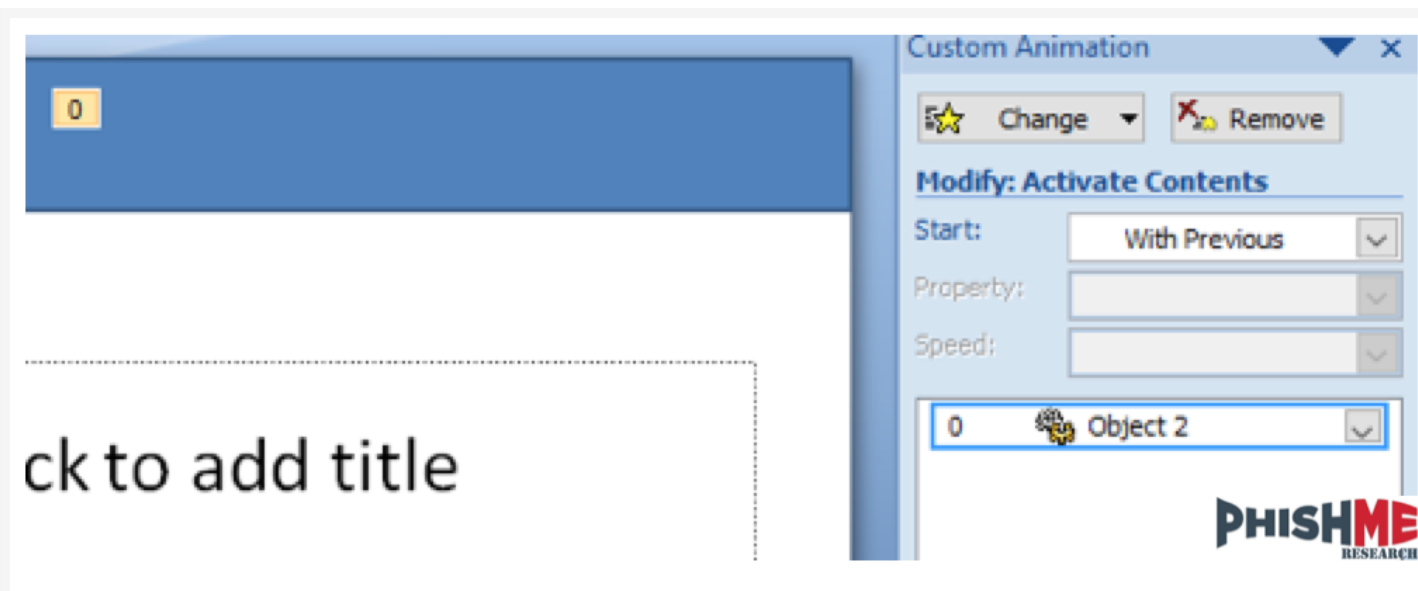


Figure 1. Custom Action

- The presentation is saved as a PowerPoint Show file, so that once opened it immediately opens in slide show view.

When a user opens the presentation, it opens in “show mode” displaying the first slide to the user; this triggers the custom action and executes the embedded payload. When the embedded content is executed the user will be prompted with a security warning asking if they want to open/execute the file.

In samples we’ve observed the script was named Powerpoint.vbe likely to further trick users into executing the malicious payload.

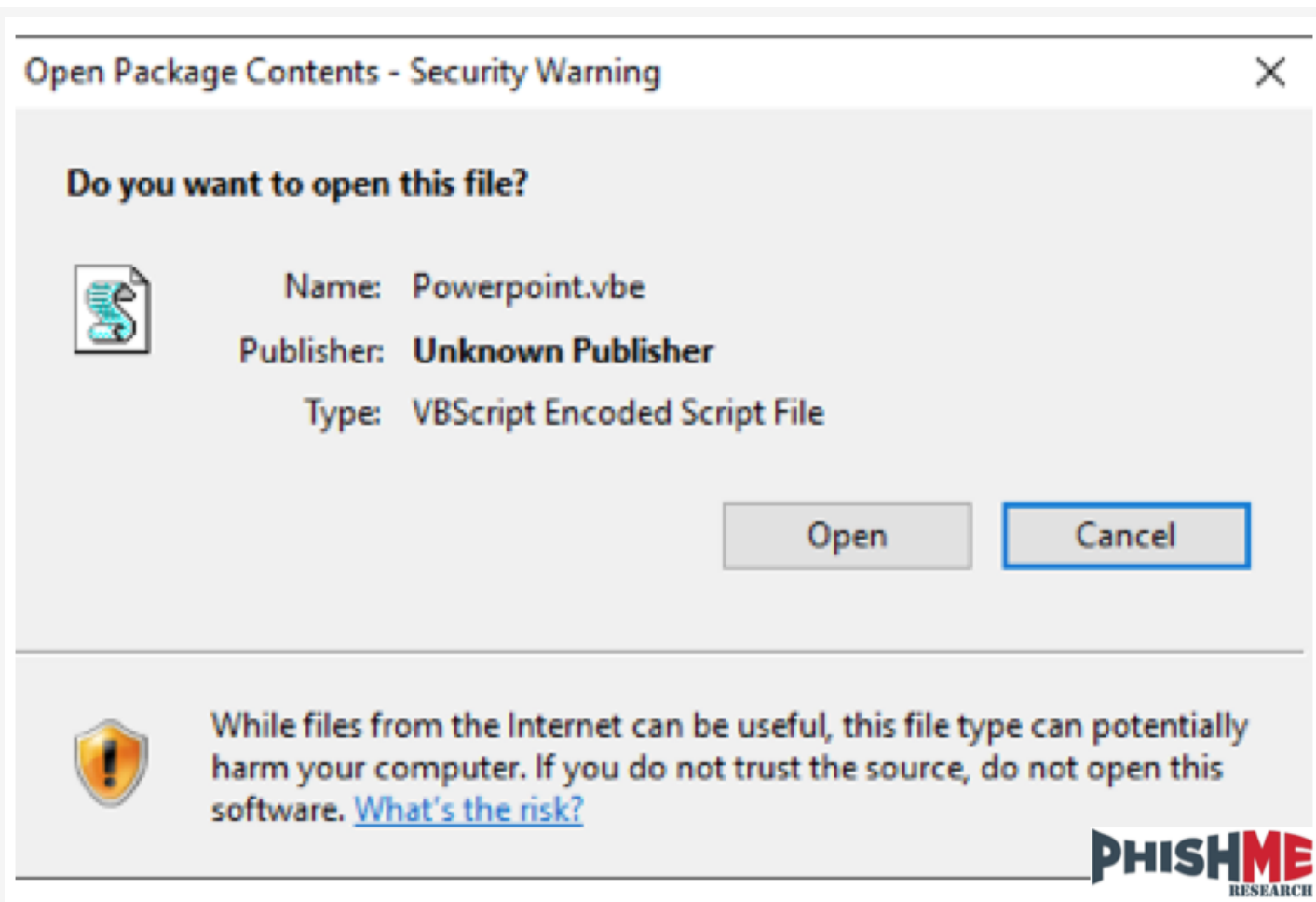


Figure 2. Security Warning

Analyzing the presentation contents, it's evident that some steps were taken by the attacker to hide the inclusion of the script from the user. An image set to look like the presentation header is placed in the foreground covering the embedded Object icon and can easily be moved for further inspection.

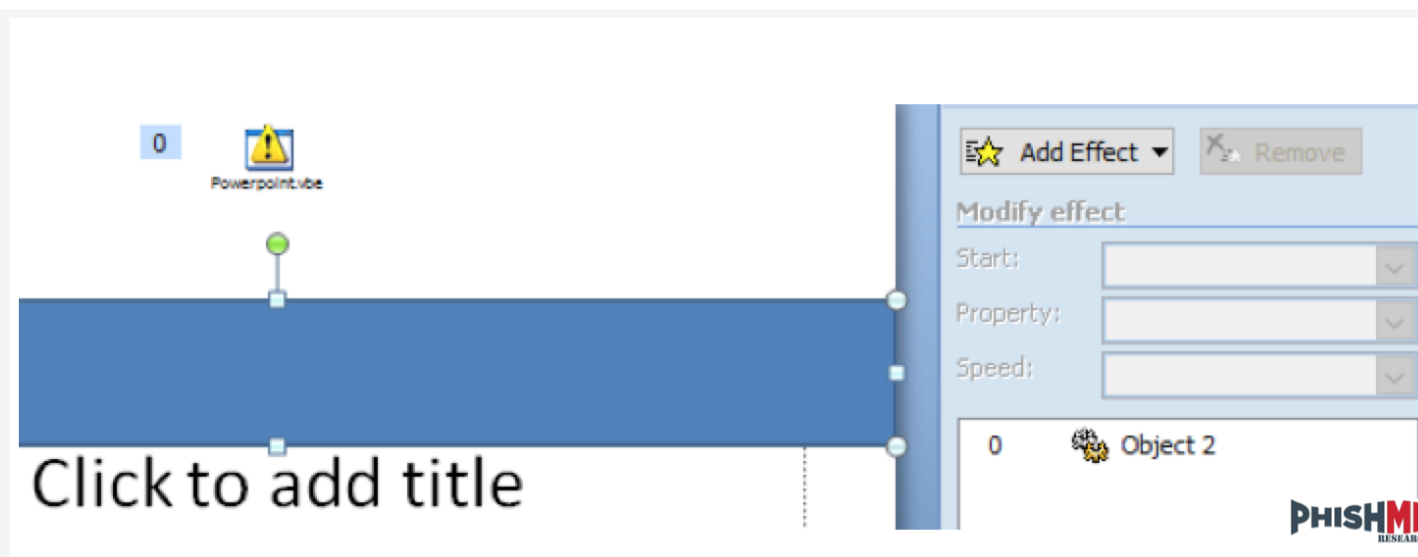



Figure 3. oleObject Reference

By default an inserted object is stored as oleObject1.bin within the ppt/embeddings directory of the archive and will increment based on the number of inserted objects.

Size	Compressed	Name
3384	529	[Content_Types].xml
738	261	_rels/.rels
607	251	ppt/slides/_rels/slide1.xml.rels
976	275	ppt/_rels/presentation.xml.rels
3228	589	ppt/presentation.xml
3399	1223	ppt/slides/slide1.xml
311	190	ppt/slideLayouts/_rels/slideLayout5.xml.rels
311	190	ppt/slideLayouts/_rels/slideLayout8.xml.rels
311	190	ppt/slideLayouts/_rels/slideLayout10.xml.rels
311	190	ppt/slideLayouts/_rels/slideLayout11.xml.rels
311	190	ppt/slideLayouts/_rels/slideLayout9.xml.rels
311	190	ppt/slideLayouts/_rels/slideLayout6.xml.rels
1991	286	ppt/slideMasters/_rels/slideMaster1.xml.rels
311	190	ppt/slideLayouts/_rels/slideLayout1.xml.rels

311	190	ppt/slideLayouts/_rels/slideLayout2.xml.rels
311	190	ppt/slideLayouts/_rels/slideLayout3.xml.rels
311	190	ppt/slideLayouts/_rels/slideLayout7.xml.rels
3064	939	ppt/slideLayouts/slideLayout11.xml
12066	1935	ppt/slideMasters/slideMaster1.xml
4531	1036	ppt/slideLayouts/slideLayout4.xml
4260	1150	ppt/slideLayouts/slideLayout3.xml
2785	838	ppt/slideLayouts/slideLayout2.xml
4176	1074	ppt/slideLayouts/slideLayout1.xml
311	190	ppt/slideLayouts/_rels/slideLayout4.xml.rels
7042	1385	ppt/slideLayouts/slideLayout5.xml
1714	675	ppt/slideLayouts/slideLayout7.xml
2840	861	ppt/slideLayouts/slideLayout10.xml
4476	1191	ppt/slideLayouts/slideLayout9.xml
4621	1255	ppt/slideLayouts/slideLayout8.xml
2052	720	ppt/slideLayouts/slideLayout6.xml
292	190	ppt/drawings/_rels/vmlDrawing1.vml.rels
7004	1667	ppt/theme/theme1.xml
1366	612	ppt/drawings/vmlDrawing1.vml
3706	1075	ppt/media/image1.wmf
3584	1304	ppt/embeddings/oleObject1.bin
7168	7168	docProps/thumbnail.jpeg
287	170	ppt/presProps.xml
182	172	ppt/tableStyles.xml
860	407	ppt/viewProps.xml
1224	521	docProps/app.xml
655	343	docProps/core.xml
<hr/>		
97699	32192	41 files, 0 folders



**PHISHME**  
RESEARCH

Figure 4. oleObject1.bin

Using the [psparser.py](#) tool to inspect the embedded file we can view the object metadata and extract the malicious payload.

```
[*] Analyzing file....
[*] File is an OLE file...
[*] Processing Streams...
[*] Found Ole10Native Stream...checking for packager data
[*] Stream contains Packager Formatted data...

Header:      0200
Label:       Powerpoint.vbe
FormatId:    00000300
OriginalPath: C:\Users\Administrator\Downloads\SCAN\Powerpoint.vbe
Extract Path: C:\Users\ADMINI~1\AppData\Local\Temp\Powerpoint.vbe
Data Size:   962
Data (SHA1): ad51c15b572694f462b05922ccad3b0e46b0fdac
```




Figure 5. Packager Object Data

In this sample the attacker embedded a script (Powerpoint.vse) which will download and execute the file 'updater.exe' ([c098a36309881db55709a759df895803](https://c098a36309881db55709a759df895803)) from the remote site `hxxp://secureemail[.]bz/updater.exe`.

The following script available in the Microsoft TechNet Gallery can often be used to decode these VB encoded scripts: <https://gallery.technet.microsoft.com/Encode-and-Decode-a-VB-a480d74c>

### Detecting Malicious Presentations

Attackers looking to leverage custom actions as a delivery vector need to ensure two things:

- An action is triggered when the slide deck starts.

- The action executes an embedded payload.

Additionally, attackers will often attempt to obfuscate the payload name to lure users into allowing execution. Incident responders can use these properties to help identify malicious presentation files.

For a presentation to run automatically as a slide show, the attacker needs to save the file as a PowerPoint Show (ppsx) document, which is identified within the [Content Types].xml file as:

- application/vnd.openxmlformats-officedocument.presentationml.slideshow

*Note: Attackers can bypass the inclusion of this content type by renaming the file to use the legacy .pps extension. This will trigger PowerPoint to open the document in Slide View even though it's not in the binary format. Renaming to the modern ppsx extension will cause PowerPoint to throw an error...*

The attacker will also need to embed content to execute once the action is triggered and is often a script or executable file. Either of these will be embedded within the presentation as an OLE object handled by the legacy Packager server (packager.dll).

By default the embedded object will be contained in a graphicFrame and referenced by an oleObj node in the slide XML markup. Note: the oleObj tag could be contained within other objects if the attacker modifies the output and will contain the tag embed indicating an embedded object

In cases where the embedded content is a script or executable, the progId will be "Package" identifying that there is no native server to handle the content.

```

<p:graphicFrame>
  <p:nvGraphicFramePr>
    <p:cNvPr id="4" name="Object 3"/>
    <p:cNvGraphicFramePr>
      <a:graphicFrameLocks noChangeAspect="1"/>
    </p:cNvGraphicFramePr>
    <p:nvPr/>
  </p:nvGraphicFramePr>
  <p:xfrm>
    <a:off x="3810000" y="1066800"/>
    <a:ext cx="819150" cy="604837"/>
  </p:xfrm>
  <a:graphic>
    <a:graphicData uri="http://schemas.openxmlformats.org/presentationml/2006/ole">
      <p:oleObj spid="_x0000_s1026" name="Packager Shell Object" r:id="rId4" imgW="818640" imgH="604800" progId="Package">
        <p:embed/>
      </p:oleObj>
    </a:graphicData>
  </a:graphic>
</p:graphicFrame>

```



Figure 6. Embedded Object Markup

As seen in the [past](#), attackers will often move to use [legacy Office formats](#) as a way to further obfuscate the content of a document. The modern Office file formats are a standardized XML markup that can be easily analyzed by incident responders and researchers. In contrast the legacy format is a binary file consisting of a number of OLE Streams that are [documented](#) within the [MS-PPT](#) Specification.

The specification is lengthy at ~650 pages, but luckily we don't need to fully review the entire implementation. Instead we want to know how OLE Objects are referenced and stored within the format.

Analyzing samples and the specification leads to the following record types that are used to reference embedded or linked OLE objects.





Figure 7. Reference to embedded objects

The existence of the [RT\\_ExternalOleEmbed](#) container and the Atom types [RT\\_ExternalOleEmbedAtom](#) and [RT\\_ExternalOleObjectAtom](#) indicates that there is an embedded or linked OLE object and are strong indicators that the file should be further analyzed for malicious content.

## Incident Response

The following primary indicators can be used to identify artifacts of the initial infection. Analysts and Incident responders can also use the included [Yara rules](#) to help identify malicious documents.

## Sample Hashes

- d7c6e591c0eb1e7ab23c036fd1c93003
- 2968fb5744433a7a8fabf65228f57801
- f4abbd6f97f035cfadd43962ee5c0e20

### Primary Network Indicators

- hxxp://secureemail[.]bz/updater.exe
  - c098a36309881db55709a759df895803
- hxxp://secureemail[.]bz/pending.exe
  - 982a2161673245c3eaa80313238f4037

For existing **PhishMe Triage** customers these malicious documents are detected using the following rules:

- PM\_PPT\_With\_OLEObject
- PM\_PowerPoint\_Show\_Embedded\_OLE
- PM\_PowerPoint\_Single\_Slide\_Presentation

### Conclusion

This is another example demonstrating how attackers leverage existing application features to bypass security controls. Instead of relying on macros, the attackers are leveraging slide animation and a custom action to trigger the embedded payload. Additionally, unlike traditional macros the user doesn't need to take action to enable execution of a scripting language, instead they confirm they want the payload to run. This provides attackers additional options to craft executable or script names, in an effort to trick end users.

## Leave a Reply

Your Comment\*

Your Name\*

Your Email\*

Your Website

☐ Save my name, email, and website in this browser for the next time I comment.

**POST COMMENT**

Cofense Headquarters



1602 Village Market Blvd, SE #400  
Leesburg, VA 20175  
Tel: 703.652.0717



## Sitemap

Overview

Management Team

Board Of Directors

Careers

Contact

Sitemap