Manipulating FTP Clients Using The PASV Command

mark@bindshell.net
http://bindshell.net/papers/ftppasv
Version 1.0

4 March 2007

Abstract

This paper discusses a common implementation flaw in the File Transfer Protocol (FTP). Several popular FTP clients are affected including web browsers. Some proof of concept code is presented to demonstrate how the vulnerability can be used to extend existing JavaScript-based port scans. Finally, some consideration is given to other ways in which this flaw could present a security risk to other FTP clients.

Contents

1 FTP Client Implementation Flaw

It is possible for malicious FTP servers to cause some popular FTP clients to connect to TCP ports on other hosts. This allows us to extend existing JavaScript-based port scan techniques [spi] in the follow ways:

- Scan ports which modern browsers would not normally connect to [portban]
- Fingerprint services which do not send a banner by timing how long the server takes to terminate the connection
- Perform simple "banner grabbing" to identify services running on other hosts

1.1 Causing An FTP Client To Connect To Another Host

By crafting replies to the FTP PASV (passive) command, FTP servers are able to cause clients to connect to other hosts.

The output below shows the messages typically exchanged between client and server during an FTP connection in passive mode:

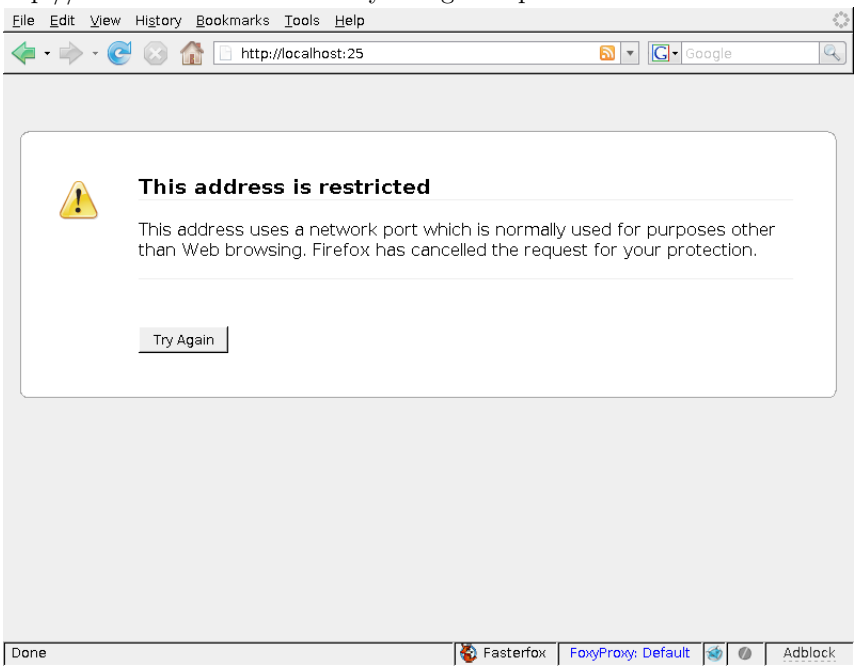
```
$ telnet 192.168.0.1 21
Trying 192.168.0.1...
Connected to localhost.
Escape character is '^]'.
220 FTP Server Ready
USER ftp
331 Please specify the password.
PASS password
230 Login successful.
SYST
215 UNIX Type: L8
PASV
227 Entering Passive Mode (192,168,0,1,84,149)
LIST
150 Here comes the directory listing.
226 Directory send OK.
```

Towards the end of the output above, the client sends the PASV command to the server to indicate that the server should listen for an incoming connection on a TCP port of its choice. The server replies, telling the client it is listening on 192.168.0.1, port 21653 (21653 = 84 * 256 + 149). Normally after the LIST command is sent (asking for a directory listing) the client would connect to the port (21653 in this example) to receive the directory listing. Once the directory listing has been sent, the server closes the connection to signify the end of the listing.

If a malicious FTP server wants the client to connect to a different IP address, it simply needs to specify a different IP address in its reply to the PASV command, e.g. to make it connect to port 22 on 192.168.0.99, it would send:

```
227 Entering Passive Mode (192,168,0,99,0,22)
```

Interestingly, Firefox will connect to whatever is sent in the PASV response, even if the target port is on its banned list - e.g. Firefox 2 wouldn't normally connect to port 25 because it's typically used for email (SMTP), not browser-supported protocols such as HTTP, HTTPS and FTP. If you try and connect to http://localhost:25 in Firefox 2 you'll get response similar to:



The ability to direct the client to a different IP address does not seem to be contrary to the RFC for FTP [rfc959], but doesn't seem to be useful in most real world situations.

1.2 How Widely Used Is Passive Mode?

We've seen above that it's only possible to direct a client to another host when it uses the PASV command - i.e. when it uses passive mode FTP. Passive mode is used by all web browsers when accessing URLs like ftp://ftp.example.com. It must also be used by all clients behind Firewalls or NAT devices unless those devices are able to understand the FTP protocol.

Command-line FTP clients, however typically using active mode (PORT commands) by default.

1.3 Vulnerable FTP Clients

The following web browsers have been found to respond to malformed PASV responses in the way described above:

- Firefox 1.5.0.9
- Firefox 2.0.0.2
- Opera 9.10
- Konqueror 3.5.5

Several command line FTP clients have also been found to be vulnerable. However as the vendors have not been notified (and the author cannot think of an interesting way of exploiting command line clients), they have been omitted from this paper.

1.4 Immune FTP Clients

The following web browsers seem to ignore the IP address returned in PASV responses. They simply connect to the IP address to which the original control connection (21/TCP) was made:

- Microsoft Internet Explorer 7.0.5730.11
- Microsoft Internet Explorer 6.0.3790.0

2 Manipulating Web Browsers

This section will look primarily at how to manipulate Firefox into portscanning and fingerprinting hosts of an attacker's choosing. We assume from now on that we are able to coerce a victim into executing some JavaScript of our choosing - e.g. by viewing a site under our control, or via XSS. The techniques may or may not also apply to Konqueror and Opera - further research is required, see Section 4.

2.1 Portscanning Banned Ports

This is basically a case of automating the process described above. We will need:

- A malicious FTP server to tell the client to connect to the port we want to scan
- A web page to lure unwary victims to
- Some JavaScript to make lots of request for FTP URLs (the engine for our port scanner)

Portscanning using JavaScript is well documented. All that remains is to demonstrate how to scan banned ports. To begin with we'll adapt the portscanner at http://no.spam.ee/scanner/ which is based on AttackApi [attackapi].

FTP Server

The FTP server will need to return a PASV response which contains the IP address and port the browser should connect to. It's easiest if this IP and port is chosen by the JavaScript scanner. A simple way for the JavaScript to pass this information to the FTP server is in the username. So instead of sending "anonymous", the JavaScript will send "10.0.0.1-25" for example if it wants to scan port 25 on 10.0.0.1. An example URL the JavaScript might access is ftp://10.0.0.1-25:anypassword@ftpserver/image.png. Our FTP server would respond with a PASV response as follows:

```
227 Entering Passive Mode (10,0,0,1,0,25)
```

A sample FTP server written in PERL (*ftp-server.pl*) is included in the zip file which accompanies this paper [poc]. Usage instructions are included at the end of this section.

Web Page

For our Proof of Concept we'll use a similar layout to http://no.spam.ee/scanner/. We simply need a text area to report the results and to add some JavaScript to do the dirty work.

A sample web page *ftp-pasv-demo1.html* has been included in the zip file [poc].

JavaScript Scanner

The first modification required to the AttackApi code is to change the image source to an FTP URL with the specially chosen username as discussed above.

Now we consider how to detect open and closed ports. Unless we connect to a port that sends us an image file without us having to send any data first, the image will fail to load. As with the original AttackApi code, the trick is to measure how long it takes for the image to fail to load:

- If the image loading fails quickly, either the port we connected to was closed or it was open, but closed very quickly (e.g. TCP wrapped). We therefore modify AttackApi to report "Closed / TCP wrapped" in this case.
- If the image loading takes a long time to fail, we know that either the port was filtered, or it was open and the service is waiting for us to send some data. We modify this case in AttackApi to report "Open / Filtered".

We can now portscan any port, even ones banned by Firefox 2.

The modified AttackApi code is included in the zip file [poc] as part of the *ftp-pasv-demo1.html* page.

Trying out the Proof of Concept

You'll need a box to run the FTP server, a box to run a copy of Firefox 2 on and a box to scan. You can use the same box for all three purposes if you're willing to my word for it that the demo works on 3 different boxes - scanning yourself hardly demonstrates that the attack works in the general case. First edit the page *ftp-pasv-demo1.html* in a text editor. Find the commented out line "// ftp_server =" and set the IP address of your FTP server (which you'll start in the next step), e.g.:

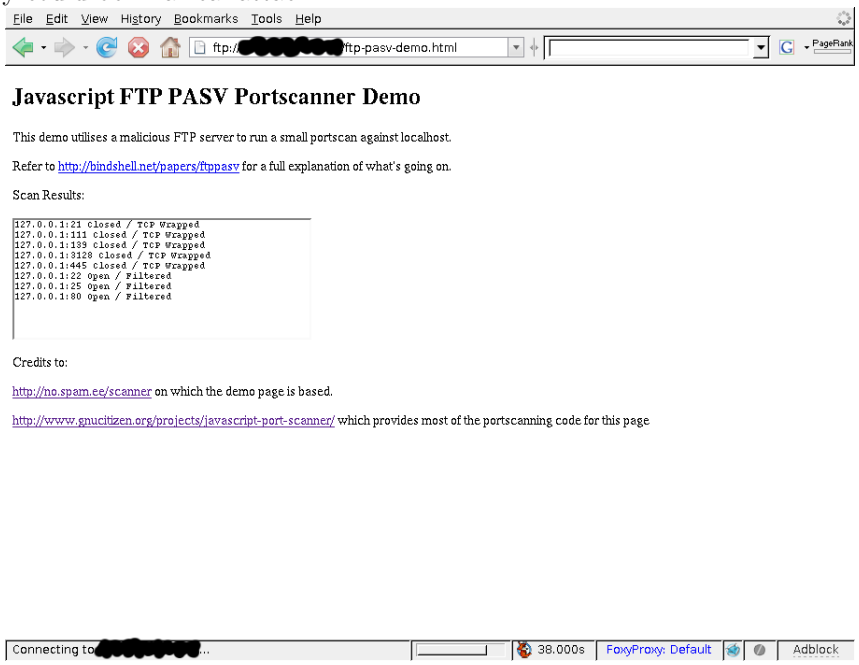
```
ftp_server = '10.0.0.1';
```

Then start the FTP server (as root). The server may work on Windows too, but I haven't tried it:

```
ftp-server.pl <ftp-server-ip> ftp-pasv-demo1.html
```

Ensure that for both steps you use an IP address of the FTP server which is reachable by your Firefox client.

Next, simply browse to ftp://*ftp-server-ip*/ftp-pasv-demo1.html in Firefox 2, making sure that you use the same IP address as above (hostnames won't work because of the same-origin policy). A scan of the host on which Firefox is running will be performed and the results displayed in the text area on the web page. The results of the scan won't be sent anywhere in this demo, although they could be in a real attack.



2.2 Fingerprinting Servers

We can't actually send any data to the ports we scan using the PASV vulnerability - after all we're downloading a file via FTP, not uploading one. This limits what we can acheive by way of fingerprinting.

However, one thing we can do is to measure the time it takes for the service to which we've connected (e.g. a mail server on port 25) to close the connection. Different services take different amounts of time to close the connection an this can help us to fingerprint them. It unlikely we'll get a completely accurate fingerprint this way, but if we want to be able to tell the difference between a Bind Nameserver and MS DNS server, or Apache and IIS, then it's a good start[1].

[1] If you're fingerprinting web servers, check out the methods SPI Dynamics came up with [spi]

Some examples are show below. Exact version number are given for reference - I'm not suggesting we can identify individual version so accurately using this method.

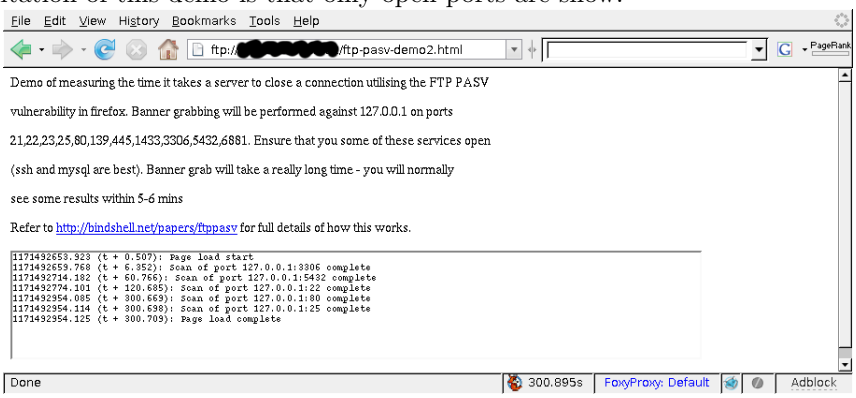
Server Type	TCP Port	Software	Version	Time to close connection (s)
FTP Server	21	vsFTPd	2.0.4	300
SSH Server	22	OpenSSH	4.5	120
DNS Server	53	djbdns	1.05-r17	10
Web Server	80	Apache	2.0.58-r2	300
Database Server	3306	MySQL	5.0.32	5
Database Server	5432	PostgreSQL	8.0.9-r1	60

Note that a variant of the this technique would probably work in other browsers such as IE6. Simply have the JavaScript measure the time it takes for an "img src" of http://target:port/any.img to fail. Most servers will close the connection eventually and some will close it early because they don't understand the HTTP request. These nuances could be used to fingerprint the server. Additionally some services may respond differently to FTP requests (quite short) and HTTP request (much longer). This would be an interesting project, but the author has not had time to investigate it.

Trying out the Proof of Concept

A web page demonstrating how we can measure the time taken for a server to close the connection is available in the file *ftp-pasv-demo2.html* within the zip file [poc]. Matching the time taken to a database of servers (like that shown above) is left as an exercise to the reader.

Run the demo in the same way as above: edit *ftp-pasv-demo2.html*, start the FTP server and browse to ftp://*ftp-server-ip*/ftp-pasv-demo2.html. A limitation of this demo is that only open ports are show.



2.3 Banner Grabbing

If we're patient enough to wait for the server to close its connection, we can actually grab banners from certain types of network service. To achieve this, we simply set an iframe's source to our FTP URL instead setting an image's source. This is kind of a degenerate case of of the "inter-protocol communication" methods discussed in [ipc].

But if we want to read the source of the child iframe (containing the banner) our JavaScript needs to have the same source as the child iframe. This is trivial to acheive. We'll simply download a web page containing our JavaScript port-scanning code from our malicious FTP server, then have the JavaScript spawn more FTP iframes (with the same origin) and the read the contents of them after they've loaded.

Trying out the Proof of Concept

The file *ftp-pasv-demo3.html* in the accompanying zip file [poc] demonstrates banner grabbing. Simply run the demo a similar way to above.

In a real attack, you'd probably want to lure users to a (HTTP) web page and source the FTP URL above into a hidden iframe. This should work fine and won't present any same-origin problems as the JavaScript will still run with an origin of ftp://*ftp-server-ip*.

3 Mitigating the Attack

The obvious recommendation is for FTP clients to behave like IE and ignore IP address in PASV responses. As an extra layer of protection, it may be possible to gain some protection by using certain Proxies and Application-layer Firewalls. Such devices would be in a position to identify the malformed PASV response.

The benefit of browsers running JavaScript for ftp:// sites is also questionable. Maybe this is a feature that should be turned off by default on the grounds that most ftp:// sites don't need JavaScript for normal operation.

Whitelisting websites which are allowed to run JavaScript would help to prevent this attack. IE7 can already do this, as can Firefox with the Noscript addon [noscript].

4 Further Research

This section summarises some of the questions raised in the is paper alongside some ideas for further research.

- Opera 9.10 warns users when it's about to follow a URL containing a username (e.g. ftp://myuser@10.0.0.1/). This makes the attack described on Firefox unsuitable: we can't pass information about which port we'd like Opera to scan in the FTP username. We could hardcode a target IP address and Port into the FTP server, but further research is needed to determine if this sort of attack is useful in practise.

- Konqueror 3.5.5 segfaults if JavaScript tries to read the contents of a child iframe if the iframe has an ftp:// URL [konqcrash]. Maybe Konqueror will be exploitable after this bug is fixed, or maybe there's an alternative way to read the child iframe.

- The author doesn't have access to a Mac, so hasn't been able to determine if Safari is vulnerable or not.

- Some research is required to see if timing how long it takes for an "img src" to fail works in IE. It would be interesting to build up a database of how long it takes various network daemons to close connections in response to a) no request at all, b) an HTTP request and c) and FTP request.

- How successful is this attack when the client is using a proxy or Firewall which understands FTP? It is conceivable that such devices would disallow the PASV response if it contained the IP address of a 3rd party, or foil our attack in some other way. We could certainly run in to trouble if the proxy allowed HTTP, but not FTP for example.

- If google-bot followed ftp links, you might be able to get it spider it's own internal network, then google for the results. It's doesn't, But maybe there are other bots that do follow FTP links. No that such an attack

would work even if the bot didn't render JavaScript (which it probably wouldn't).

- Part of the problem of running a port scan in the browser is knowing what IP addresses to scan. One way to find out the real (e.g. non-NATd) IP address of a client is using a Java applet [javaip]. Proof of concept code which automatically found some hosts on a client's internal network would probably have more impact than the POC presented in this paper. Then again, it may also be illegal in some regions.

- Is there any way to exploit text-based browsers, simple spiders or website mirroring programs if they respond to PASV requests in the way described in this paper? It's hard to see how you could coerce a text-based browser to follow a lot of FTP links (e.g. they won't automatically pull in images). Spiders, however could probably be made to follow lots of FTP links, but how could the spider be made to feed information back to the attacker about whether a port is open or closed? If the spider is single-threaded, maybe monitoring the time between successive requests would help. More research is required, anyway.

5 Credit

The author wishes to acknowledge the input of Wade Alcorn [wade] for helping to formulate a practical attack against web browsers, and Ferruh Mavituna [ferruh] for his help in writing the Proof of Concept. Thanks guys.

6 Disclosure Timeline

2007-01-29	FTP PASV Vulnerability in Konqueror reported to security@kde.org
2007-01-29	FTP PASV Vulnerability in Firefox reported to security@mozilla.org
2007-01-29	FTP PASV Vulnerability in Opera reported at https://bugs.opera.com/wizard/ (bug 249375)
2007-02-03	Crash during read of child ftp:// Iframe bug in Konqueror reported to security@kde.org
2007-02-15	Notified security@kde.org of intention to publish after 22nd Feb
2007-02-15	Notified security@mozilla.org of intention to publish after 22nd Feb
2007-02-15	Notified bug-249375@bugs.opera.com of intention to publish after 22nd Feb
2007-03-04	Publication

7 Vendor Responses

No response was provided by either Mozilla or Opera.

KDE responded and discussed both issues. However, they have yet to be convinced of the severity of the FTP PASV Vulnerability. Unfortunately, providing POC to demonstrate banner grabbing was made harder (impossible?) by the crash during the reading of child FTP iframes. KDE have reproduced the crash and produced a patch [konqcrash].

References

[ipc] Inter-Protocol Communication
http://www.bindshell.net/papers/ipc

[spi] Port Scanning in JavaScript - SPI Dynamics
http://www.spidynamics.com/spilabs/js-port-scan/

[portban] Mozilla Port Blocking
http://www.mozilla.org/projects/netlib/PortBanning.html

[rfc959] RFC 959: FILE TRANSFER PROTOCOL (FTP)
http://www.faqs.org/rfcs/rfc959.html

[attackapi] AttackApi by pdp
http://www.gnucitizen.org/projects/attackapi/

[poc] Proof of concept portscanning code
http://bindshell.net/papers/ftppasv/ftp-pasv-poc-v1.0.zip

[javaip] Finding A Client's Internal IP Address
http://reglos.de/myaddress/MyAddress.html

[noscript] Firefox Noscript Addon
https://addons.mozilla.org/firefox/722/

[konqcrash] Konqueror DoS Via JavaScript Read Of FTP Iframe
http://bindshell.net/advisories/konq355

[ferruh] Ferruh Mavituna, ferruh@mavituna.com
http://ferruh.mavituna.com

[wade] Wade Alcorn, wade@bindshell.net
http://bindshell.net