Global Information Assurance Certification Paper

SANS SECURITY ESSENTIALS GSEC PRACTICAL ASSIGNMENT
Version 1.2d
Author: Stephen Todd Redding

Why FTP may forever be a security hole, and what you can do about it.

Introduction

In researching a paper on FTP security issues, (over 40,000 of them listed on a Google search for "FTP and Vulnerabilities") it became apparent that either FTP programmers were all collectively writing poor code, RFC959 was not written with security in mind, or there is some inherent difficulty in implementing a "secure" FTP solution. Further research shows that all three may be true, but it became apparent reading the RFC for FTP that File Transfer Protocol is not intended to be secure.

RFC959 clearly states what the purposes of FTP are:

> "1. …to promote sharing of files…
> 2. …to encourage indirect or implicit (via programs) use of remote computers,
> 3. to shield a user from variations in file storage systems among hosts, and
> 4. to transfer data reliably and efficiently." (RFC959, J.Postel, J. Reynolds, Oct 1985)

The purposes of FTP do not include any mention of security, and therefore it should come as no surprise that FTP has been exploited over and over again. Indeed, FTP was designed and implemented to make sharing of files easy for programs and users. While we hopefully were all taught to share by our parents, in the world of information security, sharing is usually insecure.

It is the thrust of this paper to encourage you to reevaluate your use of FTP on your network. Most FTP implementations (i.e. programs, server software, daemons) are largely complaint with the Request For Comments that describes FTP. The examples below will show how a combination of factors has led to the onslaught of FTP vulnerabilities we see today. A consumer demand (perceived or real) has produced "extras" that produce problems, such as the "glob attack." A Request for Comments written without regard for security that may be outdated has led to such things as the "Bounce" attack and others. PASV-FTP and poorly thought out logic in stateful firewall configurations has led to compromised firewalls and networks. FTP needs to be reevaluated by the RFC committees and you may want to reevaluate what you are using FTP for in the mean time.

FTP Construction/Design

FTP is based on two connections. The first is a "control connection" which "follows the Telnet Protocol." (RFC959, Postel and Reynolds) The user initiates this connection so that the server may be told what is being requested of it. According to the RFC, FTP commands may specify the parameters for the "data connection" that is used for the

actual transmittal of data. The problem with letting the user define parameters in any protocol is easy to see here. Currently, many of the most common FTP vulnerabilities are using the "FTP Bounce Attack" (CERT Advisory CA-1997-27). The Bounce attack involves the user (attacker) opening a control connection with an FTP server.

How Does a Bounce Attack Work?

One of the most alarming things about a Bounce Attack is that it is RFC complaint. (CERT Advisory CA-1997-27) Recall from the earlier section that FTP was promoted to be easy and robust to use, not secure. By allowing the user (in this case attacker) to define the parameters of the data connection, the writers of the RFC (and many subsequent programs implementing FTP) have left themselves open to this attack.

Bounce Attacks occur when a user (attacker) is connected to an FTP server by his control connection and uses the ftp PORT command to get the FTP server to open what the FTP server believes is a data connection on the specified port. Any other machine receiving a "data" connection on TCP port 23, however, will probably believe that it is receiving a telnet request. Also, it should be kept in mind that FTP was specifically written to allow transfer between two hosts, both remote to the "user." (RFC959, p 8) So, the RFC clearly explains that a compliant implementation will allow for redirection of the data connection to a different host than the one that initiated the control connection!

Since the RFC limits us here, this author's best idea for controlling this vulnerability is to limit the exposure of your network (by putting a firewall or other packet filtering device between the ftp server and your internal network) and by removing anonymous FTP access if at all possible. Furthermore, it should be noted that most Bounce Attacks involve the intruder uploading (and subsequently using) a script or program onto the FTP server. Careful configuration of the FTP server, therefore, is also key. CERT offers configuration ideas for anonymous FTP configuration at ftp://ftp.cert.org/pub/tech_tips/anonymous_ftp_config.

OTHER NASTIES FROM THE PORT COMMAND

The PORT command in Active FTP connections is perhaps the biggest problem with any FTP server you will encounter. In this author's opinion, an extranet or internet facing FTP server should never be implemented without some sort of firewall between it and your internal network. According to RFC959, "If this command is used, the argument is the concatenation of a 32-bit internet host address and a 16-bit TCP port address. This address information is broken into 8-bit fields and the value of each field is transmitted as a decimal number (in character string representation)." (RFC959,p28)

A logical extension of the above leads us to understand that simply by connecting to an FTP server, you can tell that FTP server to connect to another host, and on what port to connect. In the RFC, there are NO restrictions on this behavior. For this reason, this

author sees a future filled with FTP PORT vulnerabilities. It is easy to see, for example, how easy it would be to complete a port scan of a network by using an FTP server that was strictly following the RFC. Or how a server on a LAN might be convinced to "respond" to a "connection" from an external host. By providing us with functionality that is simple and robust, the RFC has provided almost all networks with big holes in security.

BAD CODING AND WHAT IS A GLOB VULNERABILITY?

Filename "globbing" is the common practice of using wildcard characters (like "*" in Unix/Linux) to perform operations on lots of files with common strings in their names. The essence of this problem is that, although not required by the RFC, many FTP implementations allow for file name "globbing." While in itself this is not a bad thing, it can be exploited by creating very large amounts of data being passed to the main command processing routines. This can lead to buffer overflows. Depending on how the system is "trained" to handle the overflow, arbitrary code can be run on the server at this time. (Using the permissions of the FTP process daemon)
While this vulnerability has been fixed on a large amount of FTP packages at the time of this writing, it illustrates another vulnerability commonly found in FTP software – the software itself. Again, this demonstrates the need (since most of us do not have the time or knowledge to pour through source code) to keep our servers well patched, but should further to drive home the point that externally facing FTP servers do not belong anywhere but on a DMZ.

STATEFUL FIREWALLS AND FTP

CheckPoint Firewall-1 is of particular interest to this writer, as it is a major thrust of my daily duties. Firewall-1 operates on the "stateful inspection" principle, which commonly examines the source and destination addresses and port numbers. In the case of PASV FTP, CheckPoint is required to keep track of another bit of information: the PASV port number sent to the client from the FTP server. (CheckPoint Secure Knowledge 47.0.2638567.2540926, no date given)

During a normal PASV-FTP connection, the user (client) sends the FTP server a PASV command. This is related to the PORT command in that it is used to determine which TCP port the data connection will be established on. The difference here is that when the PASV command is issued, the SERVER now is responsible to identify a non-default data port. In a common FTP situation, the client (on TCP port 21) initiates the control connection. The client then proceeds (on TCP port 20) to open the data transfer connection. It is perhaps easier to think of active FTP sessions as "Client-Driven" and PASV-FTP sessions as "Server-Driven". Normally, the client creates all connections, and a stateful inspection type firewall can "watch" the source port in the first packet the firewall lets through. In any event, with PASV-FTP the CheckPoint firewall must watch for the PASV port that is passed from the server to the client, and dynamically allow that port number through. This behavior is on by default in Checkpoint 4.0, as many

common programs (like Microsoft Internet Explorer and Netscape Communicator) use PASV-FTP in their FTP implementations.

CheckPoint Firewall-1, then, looks for the string "227" (an FTP message code meaning "Entering Passive Mode") and extracts the destination IP and port given in the packet payload. The issue is that the destination IP and port can be that OF THE FIREWALL and there is no logic in the firewall to prevent this. A crafty person, therefore, can take control of your firewall by using this vulnerability to run arbitrary code on your firewall. (Most famous appears to be a ToolTalk vulnerability in unpatched Solaris 2.6 implementations, which can be seen in the Black Hat briefings at http://www.dataprotect.com/bh2000/blackhat-fw1.html

Again, the blame for this PASV-FTP problem is perhaps half CheckPoint, half RFC, but that will not help those of us with FTP servers on our networks. It should also be pointed out that there are various patches for CheckPoint, Operating systems as well as FTP implementations that will reduce or eliminate this particular vulnerability.

Another couple of bad things that can happen to your FTP implementation

In November of 2000, Mr. Michael Sparks submitted a paper to SANS on just two vulnerabilities in FTP which will allow the attacker to gain root access on the FTP server. It is not a far stretch to imagine that once one machine on your network is "rooted," every machine has at that point been compromised. The first exploit Mr. Sparks delineated is as simple as compiling some C code (freely and easily available on the Internet) and running it. It exploits a

Moving Files Without FTP

Most, if not all, networks of any size have internet-facing or extranet-facing FTP servers. Frequently, these servers also sit on LANs or are even someone's workstation. In an age where a simple search yields 14,000+ web pages of FTP vulnerabilities, it is easy to see that FTP has some issues that could be very costly to an organization if not addressed. While many solutions to the FTP vulnerabilities explained above are simple and effective, data is still in plain view as it traverses the Internet and (probably more importantly, certainly easier to capture) your LAN or the client's LAN. In recent months, we have seen an increase in "SFTP" being used by folks to transfer data to and fro. Sftp is a secure, interactive ftp. Scp is "Secure Copy" which works much like remote copy (rcp).

Note: SFTP as used here is "Secure File Transfer Protocol" as used in RedHat Linux and some BSD implementations. It is not "Simple File Transfer Protocol as defined in RFC913.

SCP and SFTP are excellent alternatives to FTP for business partners, virtual office users, and any other well-known trusted source. DSA keys are created and distributed to server and client machines that you want on your ftp server. As is easy to see by the flurry of VPN buying activity, encryption and key use is very popular. This is because it is

effective at keeping prying eyes off of your data. It is also effective in deterring unauthorized entrance to your network and/or its resources. SFTP allows for authentication by key sharing and passphrases. SecureFTP operates over encrypted SSH tunnels, which use TCP port 22. This makes the server much less vulnerable to enumeration and "script kiddie" type attacks. Furthermore, there are currently SSH Windows clients available, making this implementation practical for "User-type" home office or remote clients. The SFTP server daemon runs readily on RedHat, OpenBSD and others, and therefore is a supported/supportable platform. RedHat and OpenBSD are both also free (depending on how you obtain them) and therefore can be cost-justified with relative ease.

HOW TO RID YOUR LAN OF (the most common and dangerous) FTP issues.

The following is a partial list of ideas that can be used to help reduce or eliminate your risk to FTP issues.

1. **Reduce or eliminate all Anonymous FTP access.** It has to at least be somewhat difficult for the attacker to obtain a control connection.
2. **Reduce or eliminate all Internet facing FTP servers, isolate whatever is left.** While this is impractical for people wishing to freely distribute things, it can be replaced with http-download sites, or at least having these servers well-patched, well-secured and well-away from your LAN (on a DMZ or even external to the firewall behind another firewall, for example).
3. **Replace FTP with SecureFTP wherever possible.** It is a well-known fact that most network breaching begins and/or ends as an "inside job" so using these programs even within an organization may be a good idea.
4. **Consider eliminating PASV-FTP access through your stateful firewall.** This may be difficult if the bulk of your users are using Internet Explorer or the like to FTP files around, but you also must weigh the dangers of PASV-FTP. Remember: This is on BY DEFAULT in older CheckPoint products, so be wary!

While the above list is certainly not all-inclusive, the first two should produce a much more secure network than most have today. With an increasingly hostile internet and increasingly more important data being sent across on a daily basis, we must remember to reevaluate even our most faithful programs like FTP. While most vulnerabilities are "patched" or "worked-around" there are some which remain issues at the core of the protocol, the RFC. FTP remains a very useful and robust method of moving files around dissimilar systems, but it was written for a time before we were forced to take our security so seriously.

SOURCES

CheckPoint Secure Knowledge "47.0.2638567.2540926"
http://support.checkpoint.com/ (No date available)

"Firewall-1 FTP Server Vulnerability" McDonald, John
http://packetstorm.securify.com/0002-exploits/fw1-ftp.txt

CheckPoint Tech Support "Passive FTP Vulnerability" Copyright 2001
http://www.checkpoint.com/techsupport/alerts/pasvftp.html
CERT Advisory CA-2001-07 "File Globbing Vulnerabilities in Various FTP Servers" May 9, 2001
http://www.cert.org/advisories/CA-2001-07.html

CERT Coordination Center "Problems With the FTP PORT command or Why You Don't Want Just Any PORT in a Storm" February 12, 1999
http://www.cert.org/tech_tips/ftp_port_attacks.html

CERT Advisory CA-1997-27 "FTP Bounce" December 10, 1997
http://www.cert.org/advisories/CA-1997-27.html

CERT Advisory CA-2000-13 "Two Input Validation Problems in FTPD" November, 2000
http://www.cert.org/advisories/CA-2000-13.html

"WU-FTP Your Way To Root" Sparks, Michael, 21 November, 2000
http://www.sans.org/infosecFAQ/threats/wu-ftp.html

RFC959, Postel and Reynolds, October 1985
http://www.cis.ohio-state.edu/Services/rfc/rfc-text/rfc0959.txt

"A Stateful Inspection of Firewall-1" Lopatic, McDonald, Song, 2000
http://www.dataprotect.com/bh2000/blackhat-fw1.html