CS 472 HW 4

Jason R. Carrete (jrc394@drexel.edu)

Relevant log file sample output can be found in `samples/ftpserver/logs`

I implemented an auto-disconnect feature for question 6 when too many failed login attempts are made and log the disconnect when it occurs.

Part A

Question 1

When the server is in port_mode, it will allow the client to send PORT/EPRT commands to the server. When the server receives such a command, it is told what host and port to initiate a connection to on the client. For port mode, the server initiates a connection to the client on an arbitrary port chosen by the client. It is possible for the client to have a firewall setup such that all inbound connections are rejected unless explicitly allowed through. This poses a problem for port mode because if the server tries to initiate a connection to the client, the connection attempt may be blocked by the client's firewall. Assuming the connection is allowed through, then there is a problem where now a third party could connect to the client posing as the server and send malicious data to the client without the client knowing. Because FTP goes over plain text, it wouldn't be difficult to figure out what port and address to connect to because that information is specified in the port command. An attacker could read that packet and make the connection in place of the server.

When the server is in pasv_mode, it will allow the client to send PASV/EPSV commands to the server. When the server receives such a command, it is told to listen for a connection and reply with the host and port to connect to. For passive mode, the client initiates the data connection to the server. A problem with passive mode exists for the server similar to the problem caused by port mode on the client. The server now has to make sure the connection attempt will be allowed to pass through any firewall setup in order for the connection to succeed. This would require allowing a range of ports to be forwarded to the server opening up the possibility for exploits of connections made to those ports. Like with port mode on the client, the server could receive a connection from a third party instead of the intended client. This third party could then send malicious data through the data connection without the server knowing.

Generally passive mode is better for clients because the client is always initiating the connections and a user using the client doesn't need to worry about forwarding any ports. It is likely that an administrator creating an FTP server is expecting connections from various hosts and can forward a range of ports thus allowing clients to connect without having to worry about this detail. The FTP RFC allows for a client to tell a server to connect to a different FTP server

via a port command. For this scenario, it makes sense to use the port command because the client wants the server to send/receive data with another server and not with the client. One server would be in passive mode while the other server would be in port mode.

When connecting through NAT, the conversation doesn't change. The FTP protocol is the same and is agnostic to the fact that it may be communicating over a NAT. As far as the protocol is concerned, the NAT device is just another host. If it looks like a host and acts like a host, then it is a host even though it is likely a router or gateway that understands where to forward FTP packets to and from.

I think it is necessary for applications to know about IP addresses. How else are applications on two remote hosts supposed to communicate with each other if they don't know each other's address? DNS would be the abstraction that would allow applications to communicate with each other without needing to know each other's IP address. Ideally, an application trying to communicate over a network should only need to identify other hosts by their domain name instead of their IP address. Sending IP address information over a plain text channel I would say is a bad thing. The only hosts who should know how to connect to each other should be the only ones to have each other's IP address. FTP doesn't do this with commands like PORT and PASV because they send connection details including IP addresses directly in the response. I think it would be interesting if these commands were updated to allow passing hostnames instead of raw IP addresses so that way applications wouldn't need to be aware of the network layer addressing scheme.

Question 2

Using a relative path to configuration files makes assumptions about where the program currently is within a filesystem. The location of the configuration files will vary from system to system and it might be possible that a malicious configuration file could be read in from a non-standard location relative to the where the program was run. With a fixed pathname, an application can make assumptions about the access to a file based on where it is. Standard system-wide configuration files usually go in /etc on linux based systems and this directory is only writable by root. An application can be certain that only users with root access can modify these files thus making the fixed path more secure than the relative path in this case. A fixed path that doesn't point to a standard secure location for configuration files, however, is just as bad as a relative path because the non-standard location might have a malicious configuration file.

Relative paths are more convenient for development because the developer doesn't need to worry about having the right user permissions to install application settings in the appropriate locations. In either case (relative or absolute), it is important to properly sandbox data and disallow an application to navigate to parts of the filesystem that it shouldn't be able to navigate to. This is a potential problem for an FTP server where a client may try to change directory outside of the root of the FTP server directory and access system files. This is especially

dangerous if the server is run as root (like most system services are) because that allows an FTP client to download sensitive data files like /etc/passwd or /etc/shadow.

Part B

Question 3

Logging is important for security because it allows an administrator to look into what a service is doing while it is running. It provides live debugging of a service without having to know the technical details of the service implementation. A good logging system will have information about abnormal or unexpected behavior that occurs during execution that an administrator can digest. If a service is being abused, a log file can give insight into what is happening so an admin can make a decision about how to handle the abnormal (and potentially malicious activity). Without logging, an administrator is blind to the service's state and any potential problems that may be occuring.

Logging also keeps a history of the execution of a program. As long as the file exists, an admin can always look back and have an archive of what happened during a service's execution. This can be useful for reproducing issues with a service and ultimately help an admin fix any faulty behavior due to misconfiguration etc.

Question 4

In my time spent working on the FTP server, I never had any concurrency issues writing data to my log file. This is likely because I never really had two clients trying to communicate at the exact same time with the server. One possible problem of having multiple processes writing to the same log file is matching log messages with a specific process. Because each subprocess is writing to the file, there needs to be a way to figure out which process is actually doing the writing. Otherwise, an admin reading the file can't determine which connection is having issues or which message goes with which connection/process. I solved this problem in my server application by numbering each of the forked processes and including that number when logging messages for that process. By doing this, I could see the sequence of messages sent between any client and server process in order even if they are interspersed with messages from other connections.

It turns out that, "The adjustment of the file offset and the write operation are performed as an atomic step." (write(2)). Any call to write will be atomic meaning that the write will happen "all at once" and two calls to write cannot happen at the same time. So solving the problem of keeping one log file is as simple as marking which process is doing the writing each time a message is logged in a process. As long as the processes writing data to the log file write an entire log message at once, there is no fear of having garbled log messages in the log file. The order of log messages may vary between program runs, however. Including a way to ID a process along

with a timestamp of when the message is logged is a surefire way to keep track of multiple processes from one log file.

Part C

Question 5

Securing an FTP connection with implicit mode makes the assumption that both the server and client will be talking over TLS right from the beginning. This assumption would make the FTP server less compatible with other clients that may not support TLS. Implicit mode enforces security at the cost of being less compatible with less secure implementations. Starting with TLS, usernames and passwords sent over the control channel will be encrypted from the beginning which will prevent the sensitive information contained in those commands to be readable by third-parties. Potentially sensitive information sent over the data channel will be encrypted by default as well. This approach isn't flexible because it enforces TLS which depending on the use case, may be a good or bad thing. Good because you can be sure all data-in-transit is always secure and bad because both client and server have to support TLS even though some simple commands may not warrant the need to be encrypted like SYST or QUIT.

Securing an FTP connection with explicit mode doesn't make the assumption that a client supports TLS. If a client does support TLS, then the client can choose when to switch to communicating over TLS. Explicit mode allows the server to be compatible with more clients while also allowing insecure communication with those clients that choose not to use TLS. Explicit mode is more flexible because the client can control when and if they want to communicate over TLS. One downside about explicit mode is that a client might forget to turn TLS mode on when they ought to e.g. when sending over the username and password. This downside can also be a good thing because maybe all a client wants to do is see what kind of system is running the FTP server or maybe just wants a list of files. For those use cases, the client may not even need to authenticate at all and having the data go over TLS would be overkill.

Part D

FTP is a basic protocol for transferring files over the internet. It has basic security features that don't go beyond username/password based authentication. It utilizes two separate socket connections: one for sending and receiving control commands and the other for sending and receiving data (e.g. files). FTP is a simple protocol and therefore is simple to set up and easy to use. An fully implemented FTP client can send, retrieve, and remove files from a remote FTP server.

SFTP (SSH File Transfer Protocol) extends FTP by securing the data sent and received by the client and server. Unlike FTP, SFTP only has one communication channel and thus only has to encrypt data on a single channel. Having only one communication channel makes SFTP better at navigating across NATs because less ports need to be forwarded. Having one channel also requires less overhead because connections do not need to be opened and closed everytime a new data transfer command is specified. One disadvantage to having only one channel is that only one file can be downloaded at time. This may or may not be that bad depending on the bandwidth of the connection. There is no point downloading multiple files at once from the same server if one of the connections is already exhausting all of the client or server's bandwidth. And of course, multiple instances of a client could be created to allow for concurrent downloads.

SFTP and FTP have similar conversations after they are connected. The significant difference occurs during the handshake, when both applications start communication. In FTP, this "handshake" is simply a TCP 3-way handshake with little to no security. FTP adds a thin authentication layer with the USER and PASS commands but the fact that these commands go over plain-text makes the whole process insecure. SFTP on the other hand makes use of SSH keys to encrypt traffic over a single channel that is always encrypted. There is a secure process in place in the beginning of the connection for both client and server to negotiate encryption technologies and protocols.

BitTorrent is a P2P file transfer protocol that works very differently from FTP and SFTP. BitTorrent works by having a network of peers that each host pieces of files that others want to download. This differs from FTP and SFTP because there is no central server where files are downloaded from. Instead, BitTorrent users can download a file called a torrent file or click on a magnet link. The torrent file contains information to help guide the client to the right peers to download the desired file. A magnet link takes this indirection a step further by telling the client where to find the torrent file. The main purpose of BitTorrent is to share files and as a result, things like listing directories or deleting remote files don't make sense here.

BitTorrent is potentially faster than FTP and SFTP because there may be many peers (seeders) that can share the file with a BitTorrent client versus a single server that may be very slow. With BitTorrent, as long as there are enough peers, it is very likely to have a speedy download. Another advantage of BitTorrent is that files don't need to be hosted centrally. The bandwidth and storage is shared amongst the peers of the network. One disadvantage of BitTorrent is that you are less sure about the content of the files you are downloading. Depending on where a user gets their torrent/magnet links, the torrent file may refer to data that isn't actually what the user wants but instead *looks* like what the user wants. A BitTorrent user therefore has to be more careful and inspect the files they are downloading to make sure they are correct. With FTP and SFTP, the integrity of the file is less of an issue because you are likely connecting to a known server that you have authenticated with.

Part E

Question 6

In my server implementation, I don't have specific log messages that can tell an admin if someone is trying to break into the server. An admin could infer that a client is trying to break into the server by inspecting the log file and seeing if some client is entering an incorrect username/password or trying to execute commands before authenticating. It is up to the admin viewing my log file to determine when something fishy is happening. One thing I could do to improve my logging is to recognize when these events occur and make it obvious to an admin viewing my log file that they are occurring.

I mentioned previously that it may be possible to have a third-party client connect to the server data transfer socket before the actual client does. It would be possible to prevent this from happening by checking the connecting IP address and matching it with the IP address of the peer socket on the control connection. If the IP addresses do not match, then the server can conclude that the connecting host is a third-party and refuse its connection attempt. This attack affects the server when it is in passive mode i.e. listening for a data connection. It's also worth mentioning that this behavior could be intended because two FTP servers may be configured to communicate with each other by an external FTP client as per RFC 959.

I could also disconnect clients that fail to authenticate after a number of connections. If the same client repeatedly tries to gain access to the server, then that could be a sign someone is trying to break into the server and it would probably be a good idea to disconnect that client and log the event. I noticed that vsftpd did something similar whenever I would enter an incorrect username.