Prakhar Saxena

Professor Kain

CS 472

November 9, 2020

Assignment 4

1. What are the security considerations with port_mode? With pasv_mode? Why would I want one or the other (think about some of the problems that you had with the client and the server – and who calls who)? Think of the conversation between client and server. Think about how NAT changes this – is it a good thing that an application knows about IP addresses?

   FTP, originally, was not designed to be a secureprotocol. There were/are many known weaknesses, especially with the port_mode and pasv_mode. One vulnerability with the port_mode is port stealing, this is where anyone with malicious intent could easily take advantage of port mode, just by requesting random ports to send data. They can guess the next open port or even act as the middleman. One can also sniff packets if you know the port number of the port being used to send data. Also, not much can be done about the issues caused by colliding services among the common ports like 8080. And Lastly, one has to understand that at the end of the day, a client is waiting for the server, one can't monitor much because of all the activity in various ports.

   Pasv mode was introduced to attempt to fix these security issues caused by the client waiting for the server. Although, this mode has plenty of issues as well, but on the server side. In Passive mode, a random port is initialised for use, which then accepts a connection from the client, to send/receive data through it. But a big problem with this mode is that it doesn't have any sort of authentication or verification whether it's the right client. It assumes that once it receives a connection from a client on a port, it's good to send/receive data through it. This is a big problem because any 3rd client could easily accept the connection on the specified port and compromise the data transmission. Not to mention the

vulnerability of anyone packet sniffing any data intended for the serer, basically the same vulnerability as Port mode.

One would want one over the other depending on the situation. For instance if the environment is a secure infrastructure with reliable firewall or lower availability of ports, I would recommend passive here, because one wouldn't want to waste time and resources sending data to invalid ports. If such is not the case, then port mode is good, you would leave the port responsibilities onto the client. However, NAT changes some things, it changes functionalities on both passive and port modes. For passive mode, if the server is behind an NAT, then it'll need the external IP address so that it can provide it to the client with the PASV command, otherwise it would lead to a failure to transmit/receive. It's the same with port mode, unless the client knows the IP address, it's not

2. Think about the security implications of a fixed pathname (although probably in a system directory when it would be deployed as a system service) rather than a relative (like out of the current directory). Describe the "it depends" of these two approaches.

   There is an it depends answer, but in my experience, specifying a directory within code is never a good idea, it's always better to specify it in a config file of sorts. One advantage to that is that one can ensure the program can run on different environments.

   Both have their pros and cons. If you use a relative path, someone with access to the code can manipulate it easily to cause harm.

   Whereas with absolute path, though a person with malicious intent won't be able to harm the system directly, they would learn some information about the environment, as to how things are set up.

   Bottom line is that don't hard code any path in your code. Use config files.

3. Why is logging an important part of security?

   Logging is extremely important, not just for security, but even for any diagnostic check and/or debugging. For all the major benefit of logging is that there is always a trail that can be followed, to understand whatever is exactly happening. It reports what occurred at what time, issues, warnings, errors, general info, anything

pertinent to the program can and should be logged. It acts like an all-seeing surveillance system of any program.

In terms of security, it follows the similar routine; if there's a failure or a hack in, or if the system is compromised in any way, a well implemented logging enables us to see all sorts of information like what all users were logged in, what command executions were attempted, what IP addresses and ports requested what, the timestamps, locations, essentially logging can give us a great overall understanding of what happened. It helps us determine the root cause of almost all problems, and therefore helps us fix it.

4. Do you see any problems with concurrent servers and log files? (dealing with multiple processes or threads writing to the log file at the same time)? How can you solve this problem to keep one logfile for the server even though there are multiple threads/processes trying to write?

   I had a tough time implementing that, any time I tried creating multiple threads, with many connections on the server, my server logs were constantly overwritten every single time. A practical solution is to instantiate the log file immediately when the server is run. This object can then be used globally accessible, any method can access it and use it to log any message without being overwritten or erased, only appended.

5. What are the different issues with securing the connections with IMPLICIT mode (which is a separate server listening with TLS always at port 990 by default) and EXPLICIT mode (which is the same server waiting on port 21 for special commands to turn TLS on and off)? What are the "it depends" part of the tradeoffs? Think of the data that you're transporting, both on the command channel and data channel.

   There are some issues with IMPLICIT mode, for one, it is considered more strict that explicit. Before requesting any command execution, an SSL handshake is required, and the entire FTP session must be encrypted. The issues that arise with this mode are varied and they depend on the use-case. For instance, the stricter rules lead to loss of flexibility and options when acquiring a connection. No commands can be run, unless an SSL handshake is successfully completed and authenticated. Additionally, the encryption can lead to some extra computation that one might not

necessarily require, again it depends on what the use case is. A significant problem is that IMPLICIT mode expects data to be encrypted through SSL/TLS and once the client connects, it does not have any control over what's encrypted. Because explicit mode does not have some of these issues, it's considered to be the standard, whereas IMPLICIT is depreciated. An SSL connection is required only once with the explicit mode, when the connection is made between the client and server. Moreover explicit mode allows the client to toggle the SSL/TLS connection with certain commands.

It must be noted that this functionality can an issue further, because it essentially relies on the client for when to encrypt the data. As known, each way has its own pros and cons. It can be a serious security vulnerability, if compromised, the data might not be encrypted, this defeating the entire purpose of encryption.

So to answer the "it depends" aspect of things, if you want the responsibility of securing specific data

6. Do you think there are events that you've logged which show that someone is trying to break into your server? Do you have to add other log entries and checking to make sure that attacks aren't happening? Brainstorm some attacks against your FTP server. For at least one of them, implement additional code to check for them and log appropriate entries.

   Yes, but I am not certain; I don't think I've built a system that covers every route of someone attempting to break into my server. I did implement authentication for when a user logs in, for which the program logs it at that time. The program also does log the authentication process, including its outcome of whether the user was authenticated or not. If a user attempts login with invalid username/password, then the program logs that entire scenario, including why the user was denied, what credentials were passed etc.

   Although, I do think I should add other logging methods to, in the worst case

scenario, report any attacks. I think there's a lot that can be done to address man in the middle attacks, proper IP addresses and ports must be verified and used. So if there are any attacks, one can go through the logs to diagnose the issues.

Extra Credit

Think of the conversation of FTP and compare it to other file transfer protocols

- SFTP – offers the service on port 22 and data and commands share the same channel – better or worse?
  In my opinion, it is better. We don't have to send/receive plaintext form of data because of the use of a secure protocol. This therefore averts the risk of packet sniffing, because the data is encrypted, even if someone acquires the file, it would mean nothing to them. Additionally, the idea of using the same channel for commands and data transfer averts the risk of man in the middle attack. This security does come with certain drawbacks, for one things get much slower, a user is now required to have an account in order to log into the server, it takes away anonymity, but in my opinion it's well worth it.
- BitTorrent – offers files from a large number of hosts.
  BitTorrent is different from FTP since it is a Peer-to-Peer (P2P) protocol. The idea is to find users with files (data) that other users want, and then basically share that piece of file/data. This happens not just for/from one user but from multiple users, thus resulting in faster transmission rates. However, this comes at the cost of many security concerns and vulnerabilities. First and foremost, the client app on your machine is downloading data from unknown and untrusted nodes and clients. This is a big vulnerability, because at any time any of these users can take advantage of the assumption of your client's trust, for one, they can extract your IP address, and use it to send all sorts of nasty undesirable things. Furthermore, all the data you have on that client is visible to the BitTorrent users and can easily be tracked by law enforcement. Using a VPN may help, but you can't be certain.

- What are the good points and bad points of each approach (FTP, SFTP, BitTorrent)?
    - FTP
        - FTP is the simplest one of the three. It is easy to setup and use (contrary to how much I struggled with the last 2 assignments) It is perfect for a lightweight file transfer scenario, such as a small business of less than 10 people. Security is not a big concern when working with a small number of nodes assuming trust. But again, that's also its problem, it's insecure, and susceptible to attacks; and the lack of encryption doesn't help.
    - SFTP
        - It follows the FTP, but it's much harder to setup, it requires account authentication from its users, but is reliable for security. Data transfer is encrypted and therefore is not susceptible to packet sniffing or pure interference. It's ideal for scenarios with more than 10 people, and trust isn't a given.
    - BitTorrent
        - BitTorrent as discussed earlier is desirable of trying to download big files, as there are multiple "seeders". However, it comes with huge security risks. It's P2P, and you are connecting with and trusting unknown nodes and clients. This trust can be easily violated since all the information is visible to BitTorrent users. Any entity with a malicious intent can extract your information like IP address and send you undesirable content.