Designing and Testing a Secure Storage System

Ryan Stanbury

Student ID: 991 367 311

Email: stanbry@sheridancollege.ca

Sheridan College

1430 Trafalgar Road

1. Abstract

A program that has been designed by using common online resources to build it has been tested for vulnerabilities. These resources, such as online forums, are used to help fix issues and bugs when developing a variety of applications. The focus on these guides however is often to just make sure the program begins to work as intended, without always thinking about how the implementation might be exploited. This program has been built by following these resources as close as possible, and then penetration tested to see how it holds up. This is useful in highlighting common ways in which these guides for developers might fail to teach the developers how to securely write a program while still having all the functionality they desire.

The various attacks that are thrown at this system are Man-in-the-Middle, SQL Injection, uploading malware, and path traversal. Each of these attacks are described in how they work, how an attacker might use them to gain additional access, as well has how to prevent or mitigate these attacks. Finally, it is discussed what the cybersecurity community can do to help prevent these common types of attacks and help show developers the correct way to implement their solutions.

1. Introduction

As computers have taken off as a huge resource in the business world, how that data they generate is transmitted and stored has been increasingly important. Since computers are so flexible in the type of data they can store, any company which uses a computer stores information in one way or another. The types of information stored can have a massive variety in not only the information itself, but also the format of that information. It could range from storing vast amounts of customer information within a database, to webpages on a server, or even a to-do list that is kept on text files. How we store the information can often be determined by the ease of access, and cost (Coughlin, 2017). Another huge factor in how the information is stored which is finally starting to get proper recognition is the security of that data, as spending on cybersecurity has doubled from 2011-2018 (Cybersecurity Spending in the U.S. 2010-2018, 2015). Business tend to have security as a lower priority than most other features of storage because it is perceived as an increased cost, while not having much of a direct return value (Schneier, 2008). While the additional cost of security features and practices might not seem ideal for many companies, the hidden cost of recovery after a data breach could potentially make proper computer security one of the most cost-effective purchases that a company can make.

* 1. Recent Data Breaches

One of the more well-known data breaches that caught the media in a frenzy was the Sony Pictures hack in 2015. This hack became public around the time the movie “The Interview” was about to hit the main screen and included terrorist threats if the movie was broadcast on the release date. The reason for this was that the suspected culprit behind the attacks was the North Korea, as they did not want the movie to be released which makes fun of their leader and their culture (Hesseldahl, 2014). While the attackers used sophisticated tools (Lennon, 2014) in order to break into the computer infrastructure at Sony Pictures, the extremely poor security habits and policies at the company allowed the assailants to gain further control of the system quickly and easily once they got inside. Sony had reportedly stored a massive out of Excel files that contained usernames and passwords for a large variety of services and machines under their control (Williams, n.d.). This login information not only included information on their customers, but also to many computers or other services within the company. This complete lack of focus on security practices and training at their company allowed the hackers to quickly compromise files and accounts which ended up creating even more chaos and concerns. Once all of the dust had settled and it came down to assessing their losses, Sony suffered a loss of $15 million from the breach (Frizell, 2015). Even though this entire event might not of been avoided if they had increased securing their public facing networks, if they had the proper security policies and in-house systems in place it would have drastically stalled out the attackers and would have increased the noise they would have had to create within the network. This delay and noise created would have mitigated the damage the hackers could have done to Sony, which could of ended up reducing the costs from the attack.

Another massive breach which compromised the personal information of nearly 150 million people was when Equifax had their servers hacked. This breach, however, was completely avoidable and only possible due to outdated software (Ken Sweet, 2017). They had been using an open-source package called “Apache Struts” which is a framework used to help develop Java Web applications. This software had a vulnerability that was discovered in March and shortly after had a recommended patch which fixed this security hole. Equifax did not update their version of the package though, as the breach had reportedly begun sometime in May, and continued all the way until June. The hackers were able to obtain Social Security numbers, birth dates, addresses, driver’s license numbers, credit cards, and even some passports data on the customer of Equifax (Sweet Sell, 2019). At the time this report was authored, costs are still adding up for this breach as those effected by the attack may still join the class action lawsuit in order to get compensation for the damages caused to them. Equifax has agreed to pay at least $700 million in compensation fees for those who have been impacted, among other charges. They also must pay $125 per victim to cover the cost of any credit monitoring service they might have paid for upon learning of their compromised data. If every victim ended up joining one of these types of programs, Equifax could end up paying close to $2 billion to compensate every user (Sweet Sell, 2019). These costs also do not include the 30% drop in share price once the news of the breach broke out, or the loss in revenue due to worsened public perception and lowered trust in the company. The saddest part of the story is the vulnerability in their system could have been fixed with a simple patch to their software. While patching is not always straight forward as different systems might also need a patch in order to work correctly with the new version of software, it would certainly be less expensive and time consuming to update these other programs than face the repercussions of leaving their customers data vulnerable to anyone will the skills to exploit it.

Companies are not the only possible target for those ill will intentions however, as even the city of Baltimore was hit with a ransomware attack in May of 2019. This attack was able to shut down a large part of the cities services such as their email server and phone systems. The operators of the ransomware demanded 13 Bitcoin in order to decrypt all of the files there ransomware had scrambled, which is currently approximately $100,000 (Chokshi, 2019). The city refused to pay the ransom under the advisement of the F.B.I. and the secret service to not encourage these types of attack in the future. This attack could have possibly been avoided again if the city increased their budget for Information Security. Their Information Security Manager warned the city that they needed this increase in spending, so they could create better policies and better train the cities staff, but the final budget put forth by the city ignored these remarks and did not prioritize the security of their infrastructure. While fault for this incident can be thrown around at multiple different parties (including the NSA who created the exploit that the ransomware used), it does not help correct the mistake or help fix this issue when future breaches happen.

The root cause of each of these data breaches stemmed from poorly executed or missing security protocols and practices. In the case of Sony Pictures, poor security training for their staff lead to the entire system quickly being compromised once the folder with a bunch of credentials had been found. For Equifax, the lack of updates to the software they use, especially when a critical vulnerability had been found, is going to cost them potentially billions of dollars. The City of Baltimore ignoring their security manager’s request for more resources was their downfall, as like many other companies, they did not see the return on investment in funding security when compared to another project which might give them direct revenue. Each of these have their own cause, ad consequence, but they all have similar stories; those who were in control of the environments did not have cybersecurity as a top concern. That brings in the topic of this paper, if those who are not security aware do not take the proper steps in order to ensure their systems are safe, what can the cybersecurity community do to help make the designers and leaders of tomorrow use the best cybersecurity practices. The process of building an application using public forums and online guides will be put to the test to see where they might lead developers to create a vulnerable product.

1. How can this be fixed?

While fixing the issue might seem simple in principle, it is challenging to bring to reality. Ideally all the programmers would take cybersecurity as a huge concern and have it in mind through each step of the software development process. Following formats such as DevSecOps can help guide the developers to use better security practices. This would force all parts of the program to have security surrounding each part of the program that would help fix security holes in less tested areas. As developers might often work on a particular part of a piece of software, and then move onto the next piece after it, if security is done after the program has been finished then the earlier parts which might not have seemed important to test for might have critical vulnerabilities, where as if security was in the mind of the developer from the start then every time they would test their code for functionality they could also test it for possible holes or abuse cases. While this would not fix all vulnerabilities that are common in software, it would certainly be a step in the right direction towards making sure the applications of tomorrow are safe today.

This change in the development cycle would still have to be approved and practiced by the company making the piece of software, which as previously stated, might seem like a large cost without much return on investment. It would increase the amount of time needed for each stage as the extra security precautions would need to be discussed, however the potential savings from the lack of security could cost the company billions, such as in the Equifax case (Sweet Sell, 2019). This is what the industry must push towards to teach the not security adept individuals why security is such a big concern, and why it needs a respectable amount of resources and attention to protect the company.

While better security education for developers would certainly help improve the security of our created applications, if those that are making the applications do not know how to test for security flaws, or are using direct solutions they find online without thinking about the potential drawbacks it would undermine the whole process. This could potentially mean that even if a company invested an appropriate amount of time at each step ensuring that security was part of the process the entire way through, if those who are in charge of creating the product and testing for the holes do not have the proper resources or are guided incorrectly by solutions they find online then the program would still not properly be secured. The resources that developers use must not only be guides for how to create something that functionally works, but also something that works in a secure manner. This way, even if a company does not invest in the proper resources to make sure their product is as secure as they can make it, the software will still be developed using the best security practices to help harden their systems.

* 1. Industry solutions

There are some companies that are trying to put this plan into action, and one of them is the OWASP (Wyk, 2009). They are a non-profit organization which tries to help developers build their code more securely. They promote using security best practices when writing code so that it is done correctly the first time, rather than doubling the time it takes by having to go back and correct the code once the vulnerability has been found in the software. Not only that, but they also offer intentionally insecure applications such as WebGoat that helps show developers where vulnerabilities might lie in software (Wyk, 2009). This technology is used to help developers with no security background see firsthand how writing insecure code could potentially leave your program open to numerous attacks. They walk the developer through how to exploit some of these attacks, including the code that was used to create the insecure section, as well as tips on how to change the code to preserve the functionality but remove the vulnerability. Another useful feature that they offer is the cross site scripting prevention cheat sheet which helps web developers make sure they are preventing this specific type of attack on their website (Cross Site Scripting Prevention Cheat Sheet, n.d.). Providing this cheat sheet is a simple and easy way in which web developers can ensure they are making their system secure while giving simple explanations on why some code is secure and others aren’t. This way developers don’t need to take a course on security or read a whole bunch of books in order to provide a secure product.

Other popular services that many companies love, even outside the computer field, are top 10 lists. While these lists do not often get into much detail about each item, they are able to give an overall sense on what developers might need to look for when building their program. This list from developer.com for example discusses various ways in which developers might often miss security features when designing their applications (Baig, 2017). Using lists like this one, developers can take a closer look at their own code to see what they might have overlooked. While the lists themselves might not provide a ton of detail about how it can get exploited or how to fix the potential issue, they are a good starting point. Developers can look to see if vulnerabilities that are on the list also reside in their programs and then do further research online for how to properly fix these issues and what the potential cost could be for not being able to. Top ten lists are also very easy to read with quick headlines that should not scare off developers who do not have a security background. They are often simple reads and meant as more of quick bullet points to provoke thought rather than the complete breakdown of how something might work.

Another solution to help developers make their code more secure is by using static analysis tools such as the one designed by Veracode (Veracode Static Analysis, n.d.). These types of tools scan source code to find vulnerabilities and help developers in fixing the issues. This allows any developer to make their code much more resilient against common types of attacks, even if they did not have a security background or followed some bad advice from online forums. These programs also integrate with various platforms and are available to scan multiple languages, making it much easier to be confident in the security of a program if developers are just getting used to it. Automated ticket creation and plugins for various IDEs make it very easy to use and understand the issues and solutions to correctly build the software.

2.2 The Proposed Solution

The attempt to help address this issue in this paper was to try and get in the head of any regular developer who would design some sort of server/client storage system. While these programs would have multiple other factors upon deployment such as firewall rules or Anti-virus software which would help make the system more secure, the storage system itself must be secure. If the program is the weakest link in the chain, it undermines the security of the entire system. To try and mimic a developer who might build a similar system without a background in security, this program was built using whatever code was found online to get a solution working. Even if those suggestions had clear security flaws in them, they would still be implemented to see what a finalized system might look like. Once the system has been finished, it will have multiple attacks thrown at it to see where it fails.

In creating a knowingly weak system and breaking into it, it is revealing where the community are failing developers who are trying to just get their systems to work. Since developers do not all have a security background, the security of their systems in not always a top priority or baking into the design from the beginning. This leaves several parts within the programs they built vulnerable as they might get skipped over when testing or require a massive rework of their program in order to fix the security flaw.

Once this system has been attacked in multiple ways, it will then be secured while describing the ways in which it was vulnerable, as well as the code used to fix the hole. In doing this it will create a better understanding of how the security community should teach and help those who are looking to develop their own types of systems. It will display the ways in which some online forum comments might lead their program to being easily taken over if they do not think about the potential downfalls in what they write. It will also help inform the ways in which we should participate in comments online, suggesting better ways to implement a solution where it is noticed that a bad solution has been put forward. This was the inspiration to design and break into this system, so that it would not only help the discovery of an aspiring penetration tester of where flaws might be in a system, but it would also help guide the develops of tomorrow so that simple implementation flaws become few and far between.

1. Program overview

There is a server which can have connections to multiple different clients, each of which can upload or download files, as well as send a simple text message to the server. The program was designed to be as simplistic as possible, while still having the basic features that any client/server storage system would have so that everything that is discovered is applicable to a standard system. If a bunch of new features were added that were not part of the core concept, the results would only be applicable to those systems that have those specific features, rather than the more generalized approach.

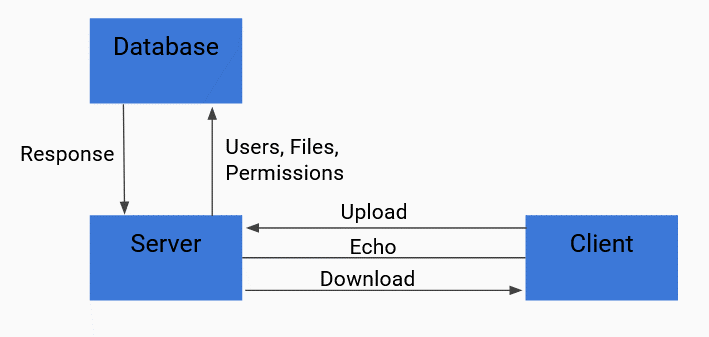


Figure - Program Overview

The program has 3 main parts, each of which will be tested. The first section in the program is the main section, this is used to start the program and launch either the server or client, depending on the options the user specifies. Once successfully launched, the settings will be saved to a text file and loaded on the next launch, so the user does not need to enter the information each time they start the program. There are also multiple validations to ensure that each option fits the desired criteria (e.g. the specified server IP address is an actual IPv4 address).

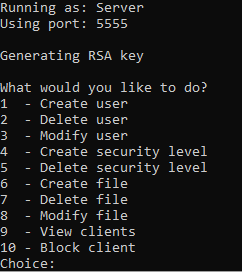


Figure - Server Menu

The next section in the program is the server. This handles all the client connections and interacts with the MySQL database to verify everything that the user attempts are allowed. The server has a simple menu of options in which someone who is controlling the server can create, delete, or modify users, files, or security privileges. The server is set to allow up to 1000 client connections at a time, however the user can specify this number anywhere between 1-1000.

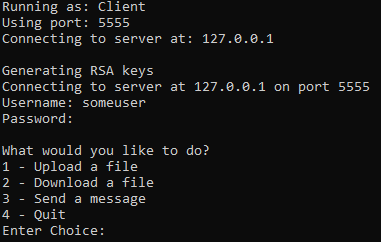


Figure - Client Menu

The final section in the program is the client section. This is what a standard user would use in order to interact with the server that they are attempting to connect to. Upon launch the client must login to the server, and once connected their credentials and security level are stored on the server’s side of the connection to verify every action the client tries to make. Clients can upload files, download files, or send echo messages to the server.

All the connections between the clients and server are encrypted using RSA 4096-bit keys. While this might not be standard in all public storage solutions, the vulnerabilities that are exposed to a system when they send their information in plaintext, or when they use weak and easily broken encryption keys, can be obvious as man in the middle attacks make traffic manipulation trivial. The rest of the program, however, has simple checks and balances which are mainly to handle errors and incorrect input from the user to keep to program functioning, rather than being strictly for security purposes. I believe this is a more accurate representation of the industry, as previously stated, security is often an afterthought in similar types of programs. To keep security with a similar mindset for the development of this program, I have tried to stick to any that I have read online while trying to troubleshoot my issues. This means that for any problem I ran into (e.g. uploading files) I would have looked for ways to get it working, rather than making sure it was done in a secure way such as using checksums to verify the file sent was the one received, or storing the checksum of a file in the database to make sure the next time it has been requested it had not of been modified outside of the program.

For every spot in my program in which I could conceive that some vulnerability might be present, I will document my thoughts before the tests, and if my predictions on what might hold up or break hold true. Things that I can break will also be patched, and I will document the way in which the issues that I come across could be fixed while still allowing full functionality of the program.

* 1. Main

Due to this section of the program having the least amount of logic put into it, as it simply is just used to determine variables to start the program, I expect it to be very stable with not much room for exploitation. This section simply tries to load its previous settings from a text file, then verifies each of these options. If any of these options are invalid, or the file does not exist, the program will ask the user to re-enter the missing information. This information can also be directly entered through arguments while starting the program, but these also get verified through the same process as if it came from a file.

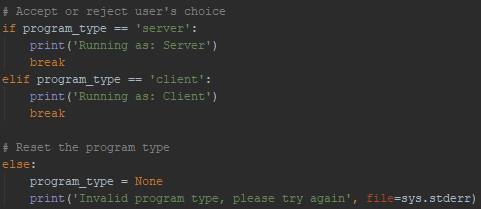


Figure - The program type is input by the user, then goes through a while loop until it meets the correct conditions

The program type is checked first to verify it is set as either a server or a client. This should not have any sort of exploit for it due to the way in which python handles user input. Rather than languages such as Java where the input from the user must match the type of variable that holds it (e.g. numbers will need an integer type to hold their data), python dynamically will change the variables type depending on the data that it contains. Not only that but any input from the user will default to a string, even if the user enters a number. This allows it to handle unexpected input much easier and allows for a simple implementation such as Fig. 1 to be in a while loop until the desired input has been entered.

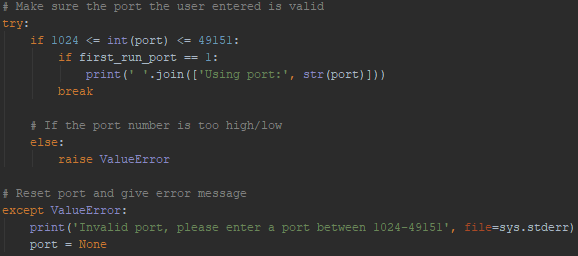


Figure - The port must be a number between 1024 and 49151

Next the port is checked to verify that it matches to correct conditions for a port. It is setup like checking for the program type, where the code in Fig. 2 is sitting in a while loop and only exists once a valid port has been input. If the input from the user is not a number, or the number is outside of the specified range it will raise a value error as remind the user what to input. I think this will also hold up and not be able to be compromised for similar reasons that the program type should be secure. Invalid or unwanted input will be handled the same way and not allow for continuation of the program.

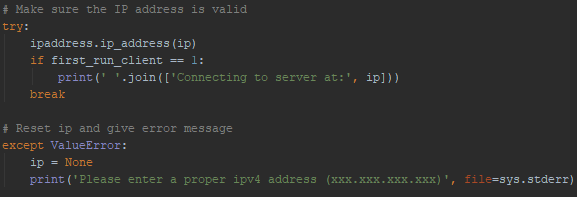


Figure - The IP address is verified and throws an error if it is not valid

Finally, the IP address is checked to verify it is a valid IPv4 address as seen in Fig. 3. This check is only done when the user specifies the program to be run as a client. This is because the server does not need an IP address to connect to, as it is only listening. This check is also setup the same way as the previous two where it is within a while loop. The only issue that I think could come from this check is if the ipaddress library has some vulnerability and does not handle it’s input correctly. If this was the case it would be an easy fix as creating the parsing myself to validate the IP address should not be a difficult task.

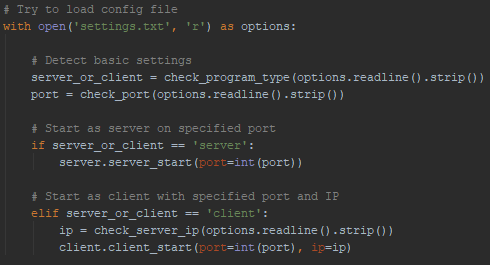


Figure - Loading the settings from the text file

Once the program has run as least once and all the settings have been saved in the text file, they are loaded upon launch as displayed in Fig. 4. All the settings that have been loaded are still sent to the various checks to verify that they have not been corrupted upon saving the file or have been corrupted outside of the program. The issue that I could see this having is if the file is manipulated outside of the program, but still have valid options. If some 3rd party were to manipulate the settings file outside of the program, they could potentially point a client to talk to their malicious server. This could be used to either send malicious information to the client or perform a man in the middle attack. I do currently have some minor mitigation against this however, as seen in Fig. 5 when the program loads valid settings from the text file it will output what those settings are. If the user who is operating the client notices the settings are incorrect, they can stop the program and delete the settings file to set them correctly. The best way that I can think of to fix this potential issue, at least from this program’s perspective, is to not store the settings information in plaintext. If the information is instead stored in a binary file with some possible obfuscation or encryption, it would be much harder to manipulate the program and perform this attack.



Figure - Output of settings read from text file

* 1. Server

The server section in the program is where the bulk of the processing happens. Upon starting the server first creates a new log file to store any information about clients that interact with it. Next it asks the user for the database information so it can attempt to connect to it. Once the information has been verified and the connection to the database has been made, it will verify that the required database name and tables exist and create them if they do not. After that it will ask the user how many clients it will listen for at a time, and then start the main part of the program. It will display a menu to interact with the database, including to create/modify/delete any stored file/user/security level. It also spawns an extra thread which constantly listen for clients that connect to the server. There are 2 possible attack vectors for how the server might get manipulated, if the system the server is operating on is already compromised, or if the client tries to interact maliciously with the server.

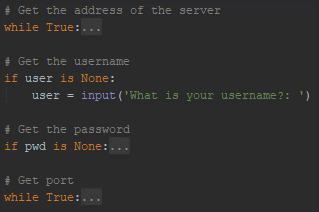


Figure - Getting database information from the user

If the machine that is hosting the server does in fact get compromised, there are very few potential attacks towards the system that the program can stop. This is because similar checks and balances are put into place when compared to getting the information of the program in the main section of the program as shown in Fig. 6. Each step is validated to ensure that correct inputs are entered so the program can handle the information accordingly. If the machine had a keylogger installed on it, then this information will be leaked which would lead to the database being compromised. This possible attack vector could not realistically be prevented within my program, but external programs to not only ensure that attackers do not breach the system, but also not monitor sensitive information that could lead to a data breach. Once this information has been collected from the user, and the server attempts to log into the database as seen in Fig. 7 it could raise a potential issue if the data is sent in plain text. If the data is not encrypted than anyone who sniffs the traffic could easily capture all this information to get access to the database. Also, since there is no validation on the correct format of the user, if the library does not properly handle attempts at SQL injection it might be vulnerable to being manipulated even if a user is not able to log in.



Figure - Using the information provided by the user to connect to the database

Potential fixes if this method is not secure would be to use a different library that has better security features or attempt to build my own library in order to interact with the database. At worst case scenario if the database has no way to decrypt incoming information, I would need to build another server that would sit on the same machine as the MySQL server that would talk to the main server. This way I could ensure that communication outside of the local machine is encrypted, but still pass the information needed back and forth to retain functionality of the program.

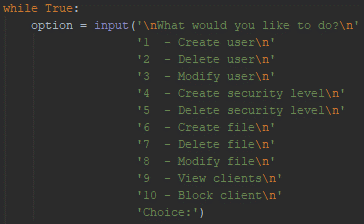


Figure - Menu for interaction on the server

Once the server has finished setting everything up and is able to start being used, the menu from Fig. 8 shows all the options that the server manager would have to interact with it. Selecting one of the options from the menu would automatically be able to send the appropriate information to the database to make the required changes. If a system has been compromised and a malicious user is able to interact with this menu, then they would have full control of the system. There is not a way for the program to determine the type or intent of a user once they reach this screen as all the authentication has already validated their security clearance. The individual options themselves would pose a much greater risk however, as the menu itself does not accomplish anything but asking the user what actions they would like to perform.

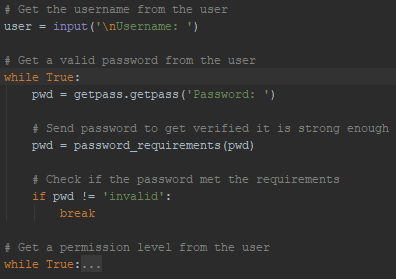


Figure - Asking for information to create a user in the database

The first option in the menu is to create a user in the database. Here we have the similar while loops that only break out once the defined security conditions are met. Keeping the format of checking for security conditions consistent across the program has its pros and cons. On one hand, if the method is implemented poorly and somehow bypassed the entire system is vulnerable, but on the other hand if it is done well, then the entire system is secure. Not only that, but it makes testing for the attacks much simpler. This is because security tests will not need to have 50 variations for 50 different types of implementation, it will streamline the process and make patching much easier should it need to be changed in the future. There are 2 main differences however in this check that could not be used in the previous ones. Fig. 9 shows that passwords entered by the user are sent to verify that the password is strong enough. These requirements as show in Fig. 10, make sure that the user does not enter an extremely weak password that can easily be cracked. These password requirements are not super strong however and could still lead to weak passwords being used and easily cracked. To prevent this possible issue, the passwords entered by the user could get hashed and compared to cracked password online in database such as <https://haveibeenpwned.com/> to make sure the password entered was not reused or likely be in a rainbow table. While hashing and salting passwords is not always practiced in business such as in the Facebook case (Krebs, 2019), when searching how to store passwords in a database some of the tutorials did encourage hashing and salting them. While this does knowingly remove some threat to the system, this practice would likely be used by other developers who are not security experts, so it was left in.

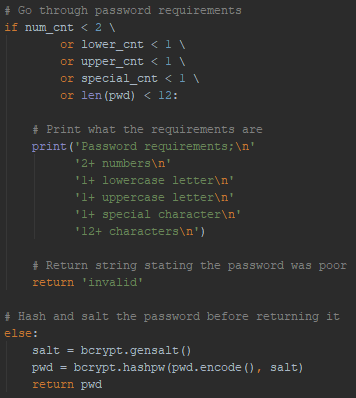


Figure - Password requirements

The second option the server admin has is to remove a user form the database. This is done by simply asking the admin which username should be deleted, then executing the DELETE MySQL query to perform the task. While researching how to use the MySQL libraries in python, there were a couple different statement formats that were suggested to be used. The first format as shown in Fig. 11 does not use prepared statements. This would likely be subject to an SQL injection attack to manipulate the query and insert custom commands into the database. To use this attack, an attacker would still first have to compromise the system the is hosting the server to be able to access this option from the menu. This would still be a valid attack vector to protect however, as they would be able to perform a greater amount of actions towards the database than the program allows for such as querying the users table of the database (not the users table for my server).

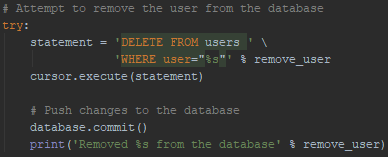


Figure - First MySQL query format

The second format for MySQL queries is what has been used throughout the rest of the program, as show in Fig. 12, due to the better security that it would provide. In using both suggested queries but using the perceived weak one only once, the program should still have much better security overall while still being able to test on the weaker format if it is indeed vulnerable to this attack.

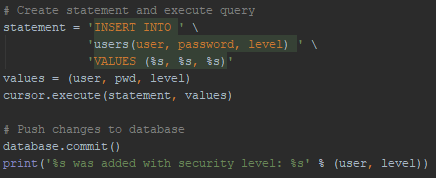


Figure - Prepared MySQL query format

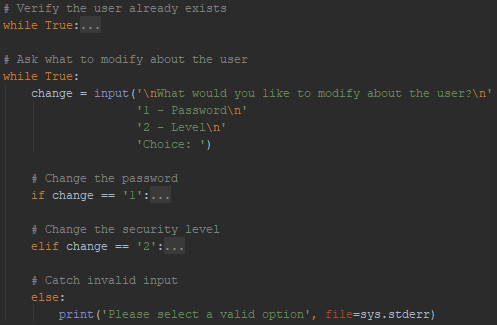


Figure - Modifying a user

The server administrator may also modify the users in the database to adjust their passwords or security levels as display in Fig. 13. This should be secure unless there may happen to be some unknown vulnerability that may occur when fuzzing the input. It is structured like past checks that break out of the while loops once the specific conditions are met for it.

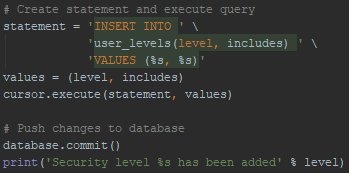


Figure - Creating a security level

The fourth and fifth options on the menu are all about security levels. These levels are tied to user accounts and files to verify that have they correct level of permissions when attempting to access any files. They are also checked when a client uploads a file to the server, so they cannot set the file permissions higher than their own level. The “includes” variable in the statement is for security levels that inherit other security levels. This creates a type of hagiarchy system like the government style of security where they have levels such as “Top Secret  Secret  Confidential  Unclassified”. These should both not pose a threat to the system as they follow the standard pattern of checks and prepared statements.

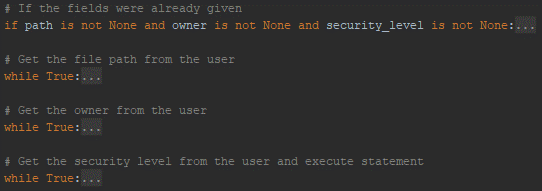


Figure - Creating a file in the database

The server administrator may also create files in the database so they can be downloaded by clients. The database is simply meant to store the path of the file, username of the owner of the file, and the security level of the file. Whenever a new file entry is added the information is stored and checked to verify the file exists, the user exists, and the security level exists. The user who now owns the file is not verified to make sure that they have the correct security level, and the assumption was that whoever is able to run this command is a verified system admin who does not have ill intentions. It is unlikely that a user who owns a file of a higher security privilege than they have would be a vulnerability as all the security level allows a client to do in the system would be to download the file, or overwrite the file if they upload a file of the same path. If it does in fact create some type of hole in the security of the system however, it would be an easy fix to add the check to verify the user would have that permission. Leaving it as is however allows for greater functionality, such as allowing a client to have access to a classified document if they need it for their work, while not giving that client the entire security level to view other documents above their clearance.

Deleting and modifying files are the same as the similar options for modifying or deleting users or security levels. They both user prepared statements would should not be vulnerable to SQL injections, and have various checks in place to ensure the database cannot be tampered with. If any vulnerabilities effect the previous two however, they would also likely effect any of the file manipulations that take place.

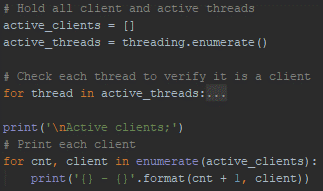


Figure - View active clients

The ninth option that the server administrator has is to view all the active clients as show in Fig. 16. This will display a list of active connections including the client’s IP address and port number they are connected on. This allows system administrators to easily keep track of anyone attempting to contact the server as spot out any IPs that look out of place. This list goes hand in hand with the tenth options available, so any of those suspicious IPs may be blocked and have their connection terminated.

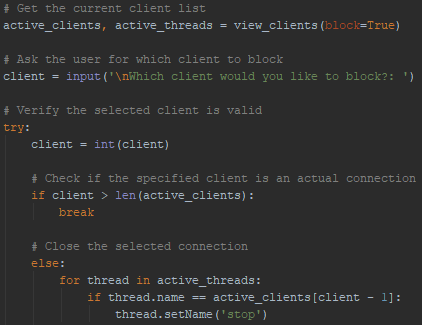


Figure - Blocking clients

Fig. 17 shows the process for how a client’s connection would get blocked. It would simply rename a thread to “stop” which is repeatedly checked and would close the connection for the client. This is highly unlikely to be abused or have some sort of vulnerability in it because a regular client would never be able to access it. The only issue would be if the server machine has already been compromised and the block client’s list is manipulated outside of the program. This would potentially create a denial of service as the list could include all IPv4 addresses, therefore blocking every client that tries to connect.

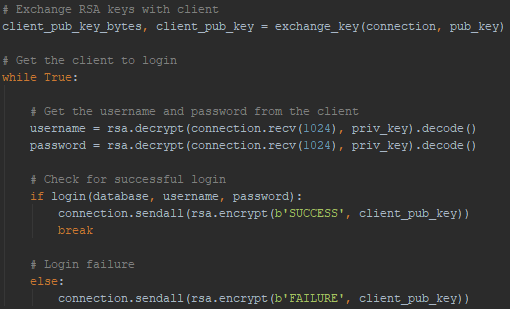


Figure - Starting a connection to a client

While the main menu of the server is certainly usefully especially when setting up or maintain the program, the main functionality comes from what a client can do while connected to it. Fig. 18 displays the first few steps on what the process looks like. First the client and server exchange their public keys in plain text and then the client sends a username and password encrypted with the RSA keys to attempt to log into the server. While sending the keys through plaintext opens the opportunity to have the traffic be sniffed and potentially manipulated with a man in the middle attack, it was the way that was described online in order to exchange keys between the client and server. All communications after the key exchange are encrypted with the RSA keys so that if sniffing were to happen after the key exchange has started, it would not be possible to decode the traffic without first breaking the encryption. They keys used for this process are RSA 4096-bit keys that are generated each time the server or client starts up. This is done to prevent a possible malicious user from logging all the traffic between the client and user to build a large bank of transmissions to mathematically break the encryption. An issue with this implementation, however, is that the server uses a single set of keys for all its connection, so if it is not restarted occasionally then an attacker would still be able to build of the bank and previously mentioned. This potential threat could be mitigated by creating the keys for each connection a client makes, rather than one key that is used for all connections.

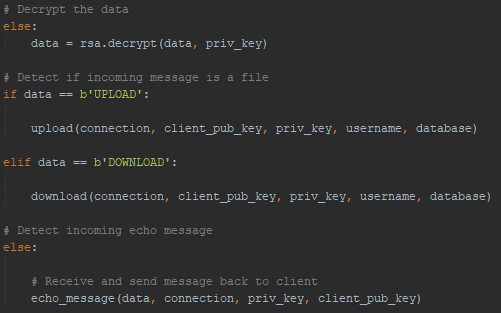


Figure - Determine the client's action

Once the keys have been established and the client logs in, the server waits for the client to tell it what it wants to do. Depending on the message that the client sends, the server will operate with the desired action. As seen in Fig. 19, if the client sends “UPLOAD” or “DOWNLOAD” it will go into the process to allow the client to upload or download files. Otherwise the server will simply echo the message back to the client and act as a basic echo server. This section of code should not be vulnerable as if the client sends some unexpected data, it will simply be treated as an echo message and not be processed. If a client does send some unexpected message that does not properly decode into UTF-8 however, it might cause some error which might cause the server to crash. This is unlikely however as the current error handling in the program should catch failed decoding, and even if it doesn’t it would likely only be the thread that crashes closing the client’s connection, rather than the entire server.

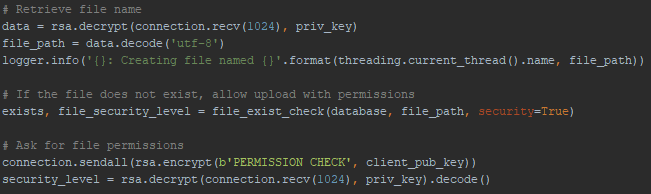


Figure - Start of uploading a file to the server

Fig. 20 displays the first couple communications that occur between the server and client once the client asks to start uploading a file. First the client sends the filename to the server which then gets checked against the data to see if the file already exists or not. After that it asks the client what the desired permissions are for the file. These inputs are then used to see if the client wants to overwrite the file in the database should it exist, and that the client has the correct permissions set the file at their desired security level. Once everything passes and the actual upload of the file to the server starts the server continues to receive data and write it to the file until the client sends the message “ENDED” which closes and saves the file. The biggest issue with this process is that when files are added to the database but not overwritten, they are not checked if they are available on the machine which the server is running on. While this is more of an implementation flaw, it could lead to some serious issues if the client wants to upload a malicious file. A client could upload files which overwrite critical system files, such as files in the C:\Windows\system32 folder if the server is run with permissions to access it. The client could also just upload generally malicious files that are not scanned by the server. This however is still assuming the malicious client can login to the server or find some sort of bypass past the login screen.

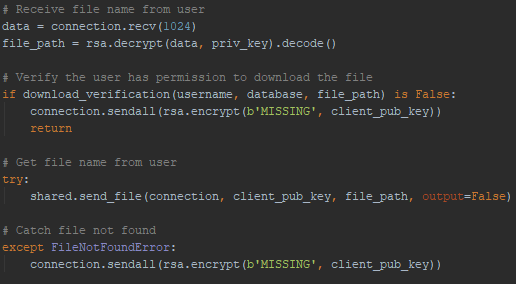


Figure - Allowing a client to download a file

Fig. 21 shows the steps taken when the client wants to download a file from the server. First the client will send the path of the file they wish to download, this will then get checked against the database to verify they have the correct permissions and that it exists in the database. Once both of those checks pass it will start transmitting the file to the client’s machine. If the file was removed outside of the program on the server machine, it will catch the error and notify the client that the file is not available. This section of code should not be vulnerable to many (if any) attacks due to the file needing to exist within the database before it can be downloaded. The only exception to where something malicious might happen at this step of the process is if fuzzing this section reveals some unexpected data can trip up the system, or if the server-client connection manages to get intercepted, decrypted, and manipulated so the client downloads a different file that the one that they have requested.

* 1. Client

The final section in the program is the section which a standard client would interact with. As most of the heavily lifting throughout this project in done at the server level, the client is much lighter in its content and capabilities. Upon launch the client will attempt to connect to the server every few seconds until a successful connection has been made, then exchange RSA keys with the server. This exchange of cryptographic keys is done in the same way that the server attempts to transmit them, in plaintext. While as previously stated this was left intentionally insecure to try and replicate other programmers without a security background who would try and replicate any guides or tutorials that they find online. After the connection was made and the keys have been exchanged, the client is forced to supply a username and password to log into the server. Once a successful login has occurred, the program offers the user a simple selection of options to interact with the server.

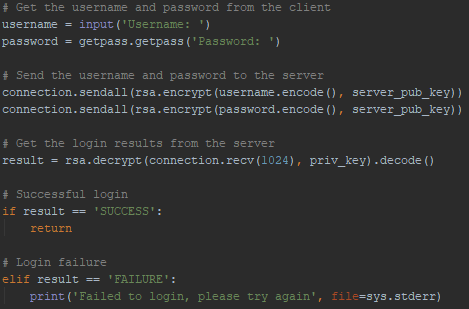


Figure - Client login section

One of the most import parts of the client program is making sure that the client logs into the server before they can have any interaction with it, as displayed in Fig. 22. The code is suck in a while loop that will ask for the username and password until a login is successful. The username and password are sent to the server to get validated, in which the server will simply respond with a “SUCCESS” OR “FAILURE” message. This login process should not introduce any new vulnerabilities into the program, and they are both encrypted with the 4096-bit RSA keys during transmission. The biggest potential threat in this scenario would be if a man-in-the-middle attack was performed and the encryption was broken, as the login information could then easily be viewing by a malicious third party. If a third party is able to decrypt all communications, but do not have a valid login to the server they could technically bypass the login screen on the client by changing the data and sending a “SUCCESS” message rather than a “FAILURE” upon login attempt, however this would only break the program. The server would still be waiting for valid credentials to allow the client to perform any other action, while the client would think that it is logged in an attempt to interact with the server using any one of the various options in the menu. It could potentially create a crash on the system if the client then sends a file which the server thinks is a username, however, this would still likely only crash the potential crash on the thread the connection was using rather than the entire server. This could start to create a denial of service attack on the server if it keeps crashing multiple connection threads or holds up the login queue so that other clients are no longer able to connect.

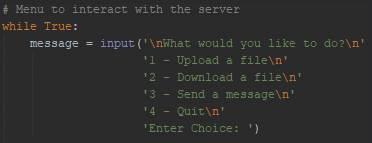


Figure - Client menu for user interaction

The short menu in Fig. 23 are the couple of options the user must exchange data with the server. As this is just a simple menu and does not have any processing itself, it should not be vulnerable attacks.

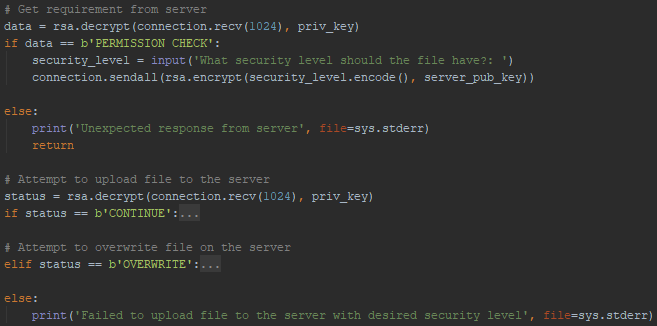


Figure - Uploading a file to the server

In Fig. 24 the few steps that take place if the client selects the option to upload a file to the server are shown. Once the client sends the username to the server, the server returns asking for the security level the file should be set at. Since the client itself does not have access to the database containing this information itself it must send its request from the server and wait for validation. If the client does not inherit the privilege that they asked the file to be set at, they are denied the upload and return to the menu. If they do inherit the permission however, the server response asking if the client would like to overwrite the file in the database should it exist, or just to start uploading the file. Finally, once the file has finished uploading, whether it overwrote a previous file or not, the client receives a status message from the server letting it know if the file has successfully been uploaded and added to the database or not. Due to everything being validated at the server side in this process, the client process is unlikely to have many exploits if the validations are done correctly. The three biggest potential threats to the system would be if the server allows for path traversal across the entire system (which in this case it currently does), if the server-client connection has been intercepted in a man-in-the-middle attack, or if the client decides to be malicious and upload any sort of malware to the server. These three attacks however can be exploited to completely take over the server machine by combining their machine. This attack and its results will later be explored near the end of the paper in the pentest section.

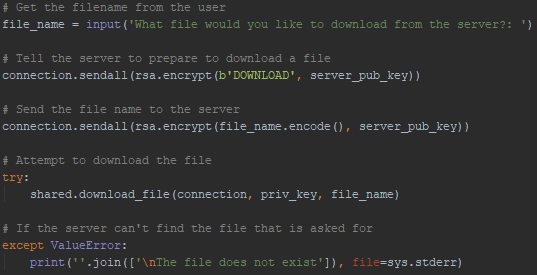


Figure - Client attempting to download a file from the server

The steps that the client would take are laid out in Fig. 25. The user would first input a filename that they would like to download, then that name is passed along to the server. Once the server starts to get ready to download the file, it will prepare it’s checks to verify that the file exists and that the user has permission to access it. If either of the checks fail, the server will respond saying that the file does not exist, and it will boot the user back to the main menu. Otherwise the server will start sending the file to the client to download. Since the client is not changing anything on the server, there can only be two possible parts where this action could turn malicious; if the server is compromised and sending malicious files, or if the connection gets intercepted and the file that was requested to be downloaded has been modified or replaced mid transit. Either of these attacks are like attacks that might be possible in the previously covered sections, so this section of code does not introduce any new possible threats to the system.

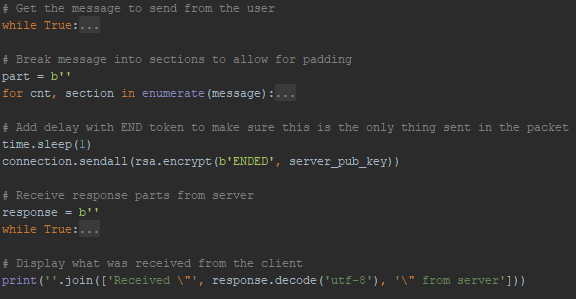


Figure - Client sending echo message to the server

The final option the client has when interacting with the server is to send an echo message and get a reply. The user simply needs to input and message they desire which will then get sent to the server and back, both displaying the message sent and the message received. While a simple echo server is rarely needed for production style programs, it is useful when performing multiple tests to validate parts of the system. Not only can it be useful when sniffing the traffic sent to see what a simple message looks like over the network, but it is also useful as a proof of concept to demonstrate attacks. It can quickly as easily display if the message between the client and the server has been changed mid-route, or to see if the message was properly encrypted once the program sent it to the server. Having this simple tool to help with testing multiple attacks does not introduce any new vulnerabilities, as sending a simple message back and forth is not executing any code, nor uploading or downloading files.

1. Pentest

This section will cover the various ways in which the system was tested for vulnerabilities, as well as the mitigations that were put in place to prevent these attacks. Many of these attacks have been briefly discussed in the program overview section of this paper as likely candidates for compromising the system. While some of these attacks present more of a threat than others, they are all important to detect and mitigate, as the implementation of the software along with the data stored can be highly valuable in many parts of an organization. Protecting and detecting as many attacks as possible reduces the chance at small vulnerabilities compounding to create critical security holes in any system.

* 1. Man-in-the-Middle

The Attack

The first test to check the security of the program is a classic man-in-the-middle attack. This is where and attacker would have their computer “sit” between the client and server connection stream. In doing this, the client would send all their data to the malicious user who would then forward that information to the server, and vice versa. This attack is typically done using a technique called ARP poisoning(Sengar, 2017), but in the case of this program it can be done without it. Not only could a malicious user read all the traffic that has been sent between the two devices, they could also manipulate the data mid-stream to adjust what the two computers say to each other. In doing this, encryption can be bypass if keys are not properly exchanged, or critical information can be manipulated to the malicious user’s benefit without the client realize what has happened.

The way in which this program is particularly vulnerable to this type of attack is if a malicious user has access to the client’s (or server’s) machine in order to manipulate the settings file. In changing the settings file, the client will connect to the malicious user so they can manipulate any traffic between the two as they see fit.

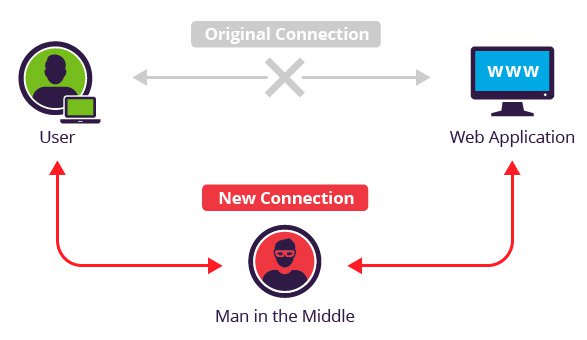


Figure - Man in the middle attack diagram (Man in the middle (MITM) attack, n.d.)

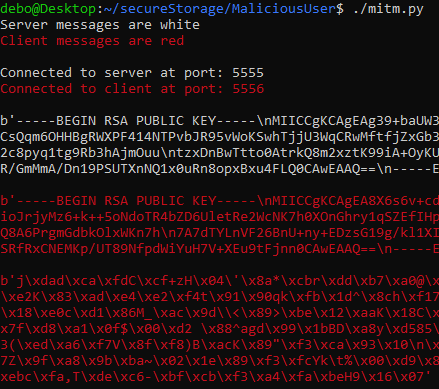


Figure - Man-in-the-middle traffic sniffing

As seen in Fig. 27, sniffing the traffic is simple to see what the machines are sending each other with the server’s messages appearing in white while the client’s messages appear in red. The first message from each machine is their RSA public key, and the following message contains the username and password encrypted with that key. Even though the rest of the communications are encrypted so that we cannot see the plaintext username and password, the information that was in plaintext can be used to break the cryptography.

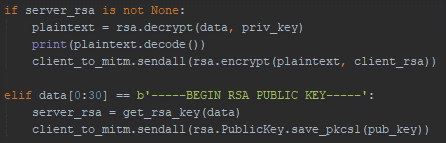


Figure - Replacing the RSA key with the malicious user’s RSA key

Since the public key is sent in plain text, even if the server generated a new key for each client, a malicious user can swap out the key exchanged for their own RSA key. Fig. 28 shows the few lines of code that are needed. The first step is to save the key that has been sent to the attacker, this is so any future communication the machines have with that device able to be decoded and read the message in plaintext. After that the attack send’s their own RSA to the server, which the server will then encrypt their messages with. Now while both the client and the server are still encrypting their messages, and from their ends everything looks to be working as intended, the attacker can do whatever they wish with the rest of their communication.

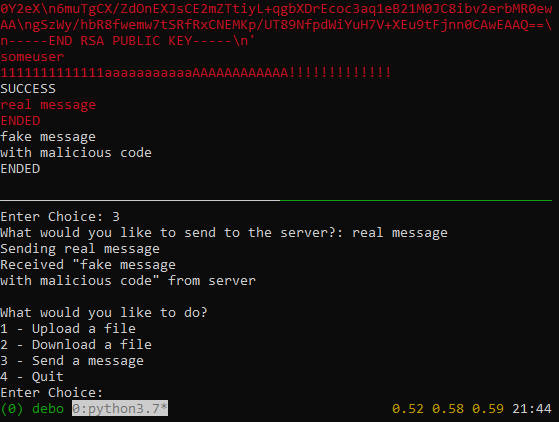


Figure - Manipulating message sent from client

An example of what an attacker could do once they have successfully pulled of this attack is displayed in Fig. 29. In this example the client sends an echo message to the server, which gets manipulated by the attacker’s machine and forwarded to the server. It is also shown on the client’s machine that the server did respond with the changed message from the malicious user. While this example itself is not an exploit to takeover a machine, it shows how someone would be able to do it. Any files that a client uploads or downloads to/from the server could have malware that gets attached to them along the way. While the file requested might seem real and still work the way it was intended, the client could get compromised without even knowing it.

Fixing the Vulnerability

While this attack certainly breaks the security of the entire system, it is not an overly complicated solution in order to correct this mistake. The mistake is in how the keys are transferred, so the solution is to use an algorithm which allows both parties to send their encrypted encryption keys without revealing what the encryption key is over the network. They keys should ideally not be hardcoded into the program itself, as that lead to reverse engineering attacks as well as cryptography attacks if enough transmissions are recorded. The algorithm chosen for this system is the Diffie-Hellman Key Exchange. This allows for both parties to negotiate a key over the network, without revealing what that key will eventually be. Using this 4096-bit key generated by DHE, the RSA key is encrypted by performing a series of XORs and bit shifts, and finally reversing the key and the end before sending it to the client. The code in Fig. 30 shows this process which goes through 64 rounds of operations.

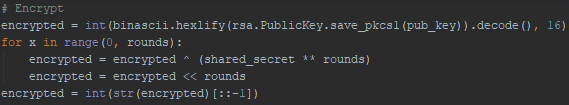


Figure - Encrypting the RSA key with the DHE key

As the DHE is randomly generated in each session with the client, the final size of the encrypted key averages around 80,000 digits. While this key may be larger than needed, it only takes a couple extra seconds to perform this step and transmit the key with the client. If the key is in fact too small and can be broken from some cryptographic attack, the rounds can easily be increased to generate a larger key. Fig. 31 shows what a man-in-the-middle attack would now display after the proper key exchange has been put into place.

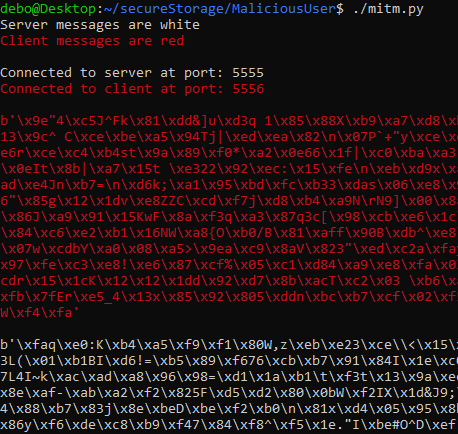


Figure - DHE Exchange

This issue remains however if the server uses the same RSA key for every client that it connects to. If this is the case, a malicious user could pretend to be an actual client, making the DHE exchange, get the public RSA key from the server and hard code it into their man-in-the-middle program to decrypt and future messages the server transmits. This can easily be fixed however by simply generating a new RSA key each time a client connects, rather than having a single key that is used for each session.

There was also the issue of a malicious crafted settings file to point the client to connect to the attacker’s machine rather than directly to the server. While this can be made much more difficult by storing the settings in an encrypted or obfuscated file, the program can still take command line arguments that overrule the settings file, as well as display the server the client is connecting to on startup. Another way to solve this threat would be to not store any settings for the next launch of the program. Users would still go through the menu to input the information to start the program as they wish or use the command line arguments to speed up the process.

* 1. SQL Injection

The Attack

The next attack against this program was SQL injection. This is a very common attack in the industry (Singh, 2019), do due to the differing ways to execute a SQL statement found online, it is best to test how they both work to see if either of them is vulnerable to this attack. SQL injection is an attack where a program asks for input from the user (often for usernames and passwords), and then that information is sent to a SQL database to verify if the information they have entered matches the records the company is storing. If the information sent from the user is not properly handled by the program, a malicious user is able to execute their own SQL statements on the SQL server. This can be done to manipulate the database in ways such as changing user accounts, creating their own administrative account, stealing data, or deleting data.

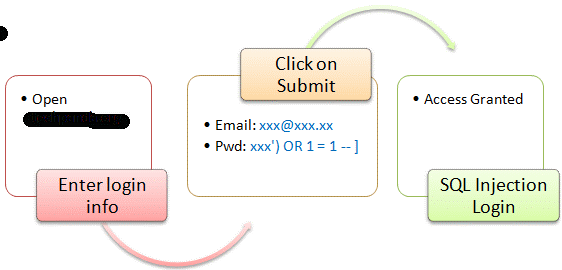


Figure - SQL Injection example (SQL Injection Tutorial: Learn with Example , n.d.)

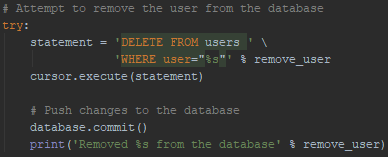


Figure - Non-prepared SQL statement

Fig. 32 was the first suggestion for how to query the MySQL database through python. This is not a prepared statement, so it takes the user the server administrator requested and directly inserts it into the SQL query. This is very dangerous in other programming languages for reasons described above, however, python’s connector to the MySQL database has built in behavior to prevent this type of attack. If the double quotes around the %s placeholder are removed, an SQL injection statement throws an error stating that it has invalid syntax. If these quotes are kept, it will automatically escape the quotation marks before it sends the string to the SQL database.

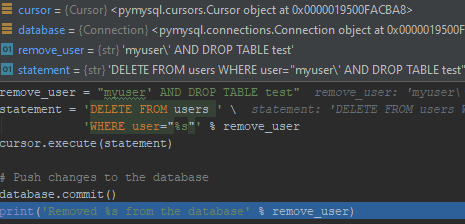


Figure - Automatically escaping strings and successfully executing the statement

Fig. 33 shows the results of trying the SQL injection attack including the debug window showing the values of the variables. The image shows that the quotation marks are automatically dealt with, and that the statement successfully ran without any syntax error.



Figure - Result of attack attempt in MySQL log files

Fig. 34 also shows what the attack attempt looked like on the server’s end. The server treated the entire string as the name of the user, rather than parsing the attack as a separate statement. This result was surprising in a good way, as developers who do not follow best practices will still have their programs protected if they make their SQL statements in python.

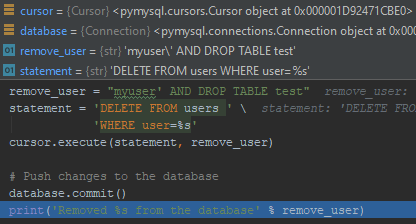


Figure - Using prepared MySQL statements

Fig. 35 was the second way in which MySQL statements were taught online. While this section was originally designed to compare how this format protected against the attack the last format was vulnerable against, it is still interesting to see how the program handles the two differently. Using this proper format, the user input is properly “quarantined” so that it cannot cause an issue. If a developer is using any language other than python (or those that quarantine the user input automatically) they must use the second option with prepared statements, else they risk the possibility that a malicious user can exploit their database using this type of attack.

* 1. Uploading Malware

The Attack

Another potential flaw in the current design of the system is that clients can upload any file they like to the server. These files can contain malicious code but are not properly checked before they are downloaded and kept. This is a difficult problem to solve from strictly the program’s point of view for a few reasons.

The first reason is that the server would first have to download the file in order to upload it and verify the cleanliness of it through online scanners such as virus total. Ideally the server would not download any data which contains any sort of malware in the first place, so while this is a potential solution, it is still not ideal. It could work if the server first downloads the files into a sandboxed environment, which cannot execute any files until they have been validated. This however still creates another possible vulnerability if the sandboxed environment had some loophole that some code could exploit.

A second option for how to fix this issue would be to have the client upload the files to an online scanner before it is able to send the file to the server. This solution would be able to get bypassed from a malicious client as they could intercept the data and force the client to tell the server that the files are always clean. The malicious user would still have to get around the encryption sent between the user and client, however since the client would be able to use their own machine, and not need to compromise a different client’s connection in order to send this data, they would be able to use their own RSA keys and easily break their own encryption. This would still require the client to bypass the login to the server, but is a potential threat, nonetheless.

Fixing the Vulnerability

The most practical solution would be to have other programs deal with any files that clients may upload. These would be in the form of Firewalls or Anti-Virus programs the specialize in dealing with these types of malicious files. This is still not a foolproof solution though, as carefully crafted malicious code can still bypass these types of systems. While it would be nice to have this program take care of everything, in a real-world scenario there is more than just a single program running to protect the network and internals of a business.

* 1. Path Traversal

The Attack

The final attack is all about ways to compromise the server from a malicious client. In this attack, a client could attempt to upload files to the server and overwrite legitimate files with malicious ones. This is a possibility as when the server gets the file name from a client, it will create that file, including any file paths that the client sends it. The current only check on the system to prevent this type of behavior is to detect if the file already exists in the database, and that the user has permission to overwrite it. If the file is new however, it will allow the client to upload the file without checking if it is overwriting a file not in the database, or if the folder the client is attempting to upload to is the regular download location for files on the server.

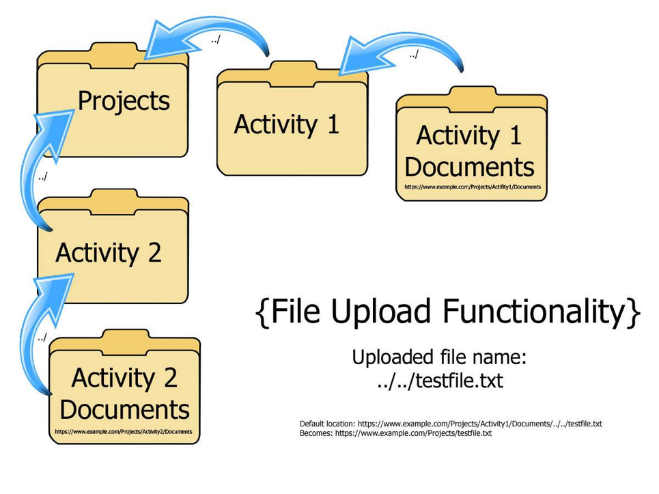


Figure - Path traversal example (Haworth, 2017)

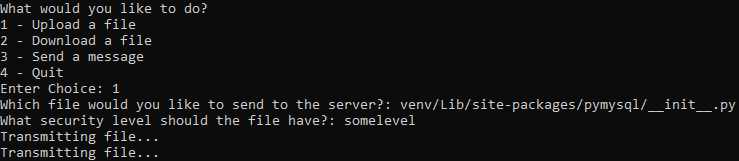


Figure - Uploading a file to the server using path traversal

Fig. 36 shows a malicious client uploading a file using path traversal to overwrite a file stored on the server’s machine. In this case, the venv folder is the python virtual environment that the server uses to import its libraries from. Here the malicious client is uploading the initialization file for the pymysql library that the server would use to talk to the MySQL database. This means that the next time the server starts up and loads the pymysql library, it would use the initialization file that was sent from the client. The part that was changed in the initialization file was a simple print statement stating that it could be malicious code, as seen in Fig. 37.

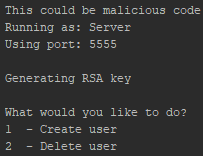


Figure - Server output on its next start after the client uploading the malicious file

While this change does not infect the server or the machine that it resides on, it could have been some malicious code such as a reverse shell or downloading and executing other malicious programs whenever the library is imported. This issue could potentially be combated by using the steps in the previous section to make sure any files that the clients upload does not contain malicious content that could be used to harm the server. This would still not completely stop the problem however, as clients could still attempt to upload files in critical system folders and overwrite important data to potentially break the computer the server is operating on, even if the files they upload do not contain malicious code. Another check should be thrown in to verify a file does not exist on the system before it starts writing the incoming file, even if the file does not exist in the database. While this would stop the server from potentially breaking as it’s critical files would stay intact, a client could still litter the entire system by uploading files in every folder it possibly could.

Fixing the Vulnerability

One reasonable way to solve this issue would be to force all downloads to get stored in the same folder and strip any path traversal that a client may send in their filename. The file names would also have to have some sort of indicator on them, such as prepending the username of the client uploading the file to the file that is stored on the system. This would allow for multiple clients to upload files with the same name without having conflicts. A second way to fix the issue would be to create a folder for each client when they try and upload any files. This would make organizing the files uploaded by the clients easier to manage as it would not have a single folder that contains every file that any client has uploaded to the system. While these do fix this attack, it would stop clients from being able to create their own folders within the system for their own organization.

The most ideal way to solve this problem, would still allow for clients to have path traversal in their files, but deny any attempt to upload files outside of the scope of their own folder. This would still allow them to have organization among their data but prevent the possibility of potentially malicious code making its way where it could harm the system.

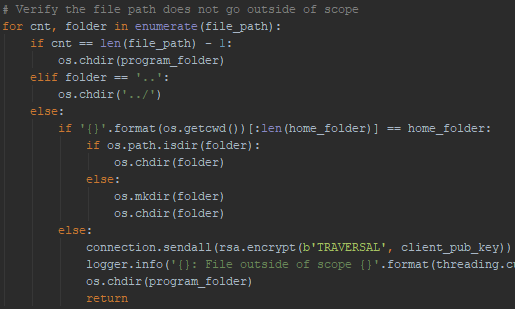


Figure - Fixing path traversal

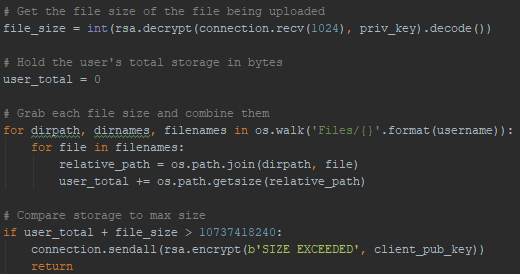
The code in Fig. 38 is what was implemented in order to fix this issue. Here the entire path that was sent from the client gets processed folder by folder to determine if the path is outside of the file storage folder. The file storage folder also contains a folder for each user to have better organization of the files. Users cannot upload files outside of their own folder, but they are able to make any number of folders within their own storage space. If the folder path that they request does not exist, it will get created for them at each step. While this does fix users from putting files wherever they like on the system, it does not prevent them from clogging up storage space on the server. This could potentially be fixed by setting a limit of the amount of data a client would be able to store in their upload folder. To implement this, fix the program could scan the size of the user’s file folder and deny the user from uploading if the size is at the maximum. It could also get the size of the file from the user before it accepts the download, so the maximum file storage size is not exceeded. The code for this implementation is displayed in Fig. 39, where the maximum storage for a single user is set to 10 GB. 

Figure - Limiting users to have 10GB of storage on the server

1. Conclusions

Putting all these attacks together shows just how vulnerable an application can be by following bad advice given in online forums. From the sending encryption keys in plaintext (CatCraftYT, 2019), to not sanitizing SQL statements (user3074823, 2013) and even allowing for file path traversal (Duchess, 2012) creates massive security holes if a system were to implement this application. It would have not been an issue however if other developers were able to point out these mistakes rather than teach incorrect ways of implementing these features. It is a cascading effect in the industry as bad advice leads to bad learning, leading to vulnerable applications and teaching other the same bad habits they learnt themselves. This problem only gets more complex as more functionality is built on top of the already existing code, leading to potentially more vulnerabilities if also done incorrectly. These vulnerabilities also get more and more important to fix depending on the type of data that is being collected by the program and company. While in this circumstance, the damage is limited by the content of the files the clients upload, as well as if the system itself contained any confidential information that could be obtained through combining mention attacks such as man-in-the-middle, path traversal, and uploading malware. If more sensitive information was transmitted however, such as social security numbers or credit card information it heightens the need to secure everything to stop this information from leaking.

Those in the cybersecurity community who are not involved with companies promoting and teaching the developers how to securely write code could still try to contribute to improving security practices from everyone. One way in which we could accomplish this goal is by participating wherever we can in online forums such as stackoverflow.com to try and answer questions so that those who follow the answer would end up with secure code. Writing paragraphs explain every detail on how the answer in secure might be a bit much for those who are just looking for a quick fix but linking resources to read further on the topic helps those who are trying to learn and better their practice. It would also be helpful (although not up to the community) if stack overflow implemented some sort of insecure tag. This way whenever an answer provided is not secure, it can be flagged as such so that other who might be looking to copy the solution for the quick fix are warned that it is not the correct way to fix the problem. This would also allow users to highlight code which is deemed secure so that new developers are able to learn the proper way to solve their problems, rather than finding an answer online that “just works”, without any thought on if the fix for their problem was the right way about solving the issue.

On top of just helping the developers understand how to properly write their code in a secure manner, it would also be useful if companies were mandated to properly train their employees if they are handling critical consumer data. This training would not just extend to the developers who would benefit from the previous help, but also the general employee to ensure they understand why the security implemented is required and how to properly protect the company. This would discourage practices such as storing passwords in text files such as in the Sony Pictures hack, or encouraging them to properly update their software when new security patches have been released such as in the Equifax or Baltimore City hacks. All these attacks could have been prevented from escalating to the scale that they did if simple security guidelines were followed by every employee. Increasing the awareness of how employees that do not follow security policies could potentially cripple the business should help them understand where the security professionals are coming from. While security might seem like only a hassle to some users, if we are able to teach them in a cost-benefit analysis of why these policies exist, they just might come around to practice what we preach.

1. Areas for further study

There are several ways to properly train employees so they have the correct amount of training to reduce the likelihood of a breach, as well as the extent in which the breach might spread. One great resource that may be used is the PCI Data Security Standard (Council, 2014). In order to be compliant with this system, businesses must implement multiple security awareness programs to help their employees understand the importance and follow through with adequate security policies. Using these types of standards take the pressure off the management team to develop their own plan, in which they might overlook important details. This way they have a guide to follow to ensure the correct programs are implemented and policies put in place to improve their security practices.

Another way to help employees build products that are not vulnerable to the attacks highlighted in this report are to follow professional documentation as a “how to” guide for building their applications (S. Milanovic, 2002). This helps the developers when designing their programs so that they have some sort of guideline rather than building their software from scratch. Using these guidelines will also help the developers rely less on the public forums to discuss the issues they are having. This would reduce the likelihood that bad advice will be given, and bad implementation will end up in the final version of the code.

Top ten lists are also something that employees should investigate when testing their applications for vulnerabilities. These lists highlight multiple common ways in which systems have poor design and implementation, so that those who use these list are able to ensure their programs does not fall under the same category (OWASP Top 10 - 2017: The Ten Most Critical Web Application Security Risks, 2017). An important thing to note about these types of lists, however, is that the vulnerabilities change all the time. This would make it smart to not only look at the most recent lists of most common vulnerabilities, but also those of previous years. In doing this the program would end up with a wider range of protection from attacks rather than only focusing on the most recent ones.

1. Bibliography

Baig, A. (2017, July 28). *Ten Security Issues Developers Overlook While Developing Apps*. Retrieved from Developer: https://www.developer.com/security/ten-security-issues-developers-overlook-while-developing-apps.html

CatCraftYT. (2019, August 4). *Python - How can I send a rsa.key.PublicKey object through socket?* Retrieved from Stack Overflow: https://stackoverflow.com/questions/57345647/python-how-can-i-send-a-rsa-key-publickey-object-through-socket

Chokshi, N. (2019, May 22). *Hackers Are Holding Baltimore Hostage: How They Struck and What’s Next*. Retrieved from The New York Times: https://www.nytimes.com/2019/05/22/us/baltimore-ransomware.html

Coughlin, T. (2017, November 8). *Low Cost Digital Storage Options*. Retrieved from Forbes: www.forbes.com/sites/tomcoughlin/2017/11/08/low-cost-digital-storage-options/

Council, S. A. (2014, October). *Information Supplement: Best Security Practices for Implementing a Security Awareness Program.* Retrieved from PCI Security Standards: https://www.pcisecuritystandards.org/documents/PCI\_DSS\_V1.0\_Best\_Practices\_for\_Implementing\_Security\_Awareness\_Program.pdf

*Cross Site Scripting Prevention Cheat Sheet*. (n.d.). Retrieved from OWASP: https://cheatsheetseries.owasp.org/cheatsheets/Cross\_Site\_Scripting\_Prevention\_Cheat\_Sheet.html

*Cybersecurity Spending in the U.S. 2010-2018*. (2015, April 1). Retrieved from Statastia: www.statista.com/statistics/615450/cybersecurity-spending-in-the-us/

*Data Breaches, Digital Storage, and Physical Authenticators*. (2015, June 16). Retrieved from Medium: medium.com/@BlueCheck/data-breaches-digital-storage-and-physical-authenticators-c56a6d487ca5

Duchess, S. (2012, February 21). *Python - Sending files over sockets*. Retrieved from Stack Overflow: https://stackoverflow.com/questions/9385509/python-sending-files-over-sockets

Frizell, S. (2015, February 4). *Sony Is Spending $15 Million to Deal With the Big Hack*. Retrieved from Time: https://time.com/3695118/sony-hack-the-interview-costs/

Haworth, K. (2017, January 5). *TRC Tech Talk: Directory Traversal Attack & Defense – Part 1*. Retrieved from WhiteHat Security: https://www.whitehatsec.com/blog/directory-traversal-attack/

Hesseldahl, A. (2014, December 3). *Sony to Officially Name North Korea as Source of Hack Attack*. Retrieved from Recode: https://www.vox.com/2014/12/3/11633486/sony-to-officially-name-north-korea-as-source-of-hack-attack

Ken Sweet, M. L. (2017, September 14). *Equifax traced the source of its massive hack to a preventable software flaw*. Retrieved from Business Insider: https://www.businessinsider.com/how-did-equifax-get-hacked-2017-9?op=1

Krebs, B. (2019, March 19). *Facebook Stored Hundreds of Millions of User Passwords in Plain Text for Years*. Retrieved from Krebs on Security: https://krebsonsecurity.com/2019/03/facebook-stored-hundreds-of-millions-of-user-passwords-in-plain-text-for-years/

Lennon, M. (2014, December 19). *Hackers Used Sophisticated SMB Worm Tool to Attack Sony*. Retrieved from Security Week: https://www.securityweek.com/hackers-used-sophisticated-smb-worm-tool-attack-sony

*Man in the middle (MITM) attack*. (n.d.). Retrieved from Imperva: https://www.imperva.com/learn/application-security/man-in-the-middle-attack-mitm/

*OWASP Top 10 - 2017: The Ten Most Critical Web Application Security Risks.* (2017). Retrieved from OWASP: https://www.owasp.org/images/7/72/OWASP\_Top\_10-2017\_%28en%29.pdf.pdf

S. Milanovic, Z. P. (2002, August 7). *Building the enterprise-wide storage area network.* Retrieved from IEEE Xplore: https://ieeexplore.ieee.org/abstract/document/937782

Schneier, B. (2008, September 2). *Security ROI: Fact or Fiction?* Retrieved from CSO: https://www.csoonline.com/article/2123096/security-roi--fact-or-fiction-.html

Sengar, S. S. (2017, December 6). *How To Do A Man-in-the-Middle Attack Using ARP Spoofing & Poisoning*. Retrieved from Secjuice: https://medium.com/secjuice/man-in-the-middle-attack-using-arp-spoofing-fa13af4f4633

Singh, S. (2019, April 23). *Common SQL Injection Attacks*. Retrieved from Pentest-Tools.com: https://pentest-tools.com/blog/sql-injection-attacks/

*SQL Injection Tutorial: Learn with Example* . (n.d.). Retrieved from Guru99: https://www.guru99.com/learn-sql-injection-with-practical-example.html

Sweet Sell, K. S. (2019, July 22). *Equifax to pay up to $700M in data breach settlement*. Retrieved from Yahoo: https://ca.news.yahoo.com/equifax-pay-700m-data-breach-112229026.html

user3074823. (2013, December 13). *Python & Mysql : Select statement with variable*. Retrieved from Stack Overflow: https://stackoverflow.com/questions/20559965/python-mysql-select-statement-with-variable

*Veracode Static Analysis*. (n.d.). Retrieved from Veracode: https://www.veracode.com/products/binary-static-analysis-sast

Williams, O. (n.d.). *New Sony Pictures leak appears to contain lists of passwords in plain text, security certificates*. Retrieved from TheNextWeb: https://thenextweb.com/insider/2014/12/03/latest-sony-pictures-leak-paints-picture-poor-security/

Wyk, K. v. (2009, February 3). *OWASP: Helping Web Developers Develop Securely* . Retrieved from Datamation: https://www.datamation.com/entdev/article.php/3800326/OWASP-Helping-Web-Developers-Develop-Securely.htm

1. Appendices

Server Input

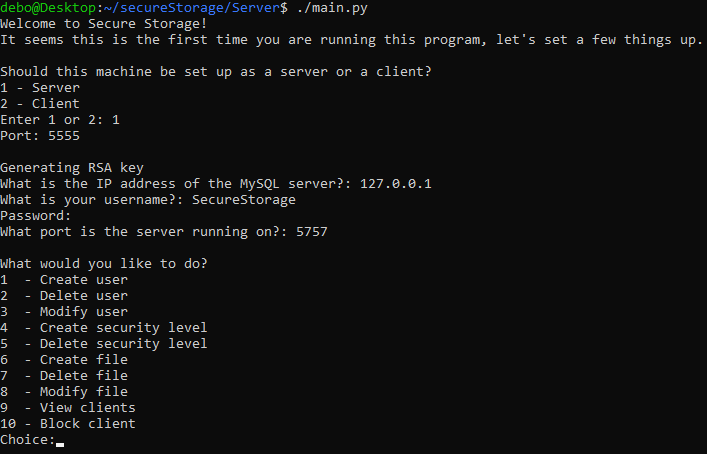


Figure - Setting up the program

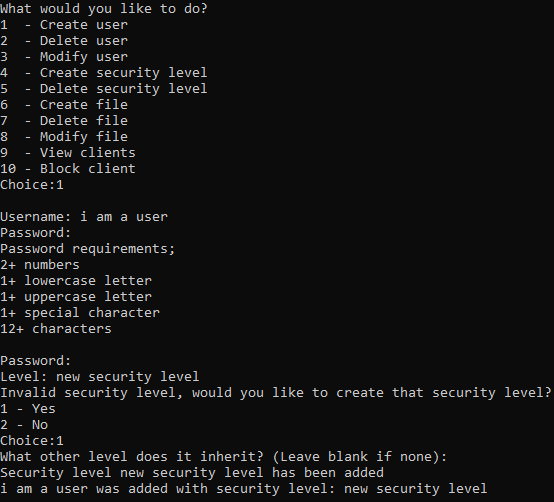


Figure - Creating a new user

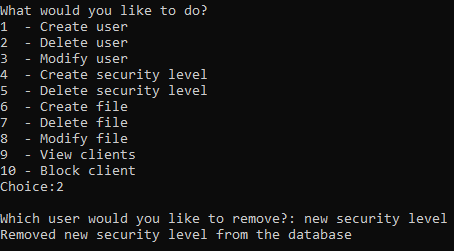


Figure - Removing a user

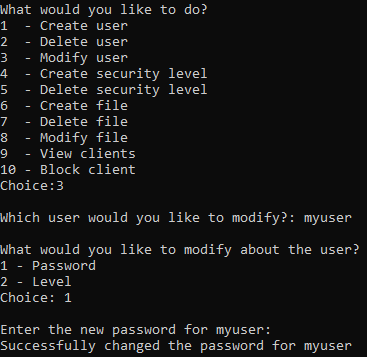


Figure - Modifying a user

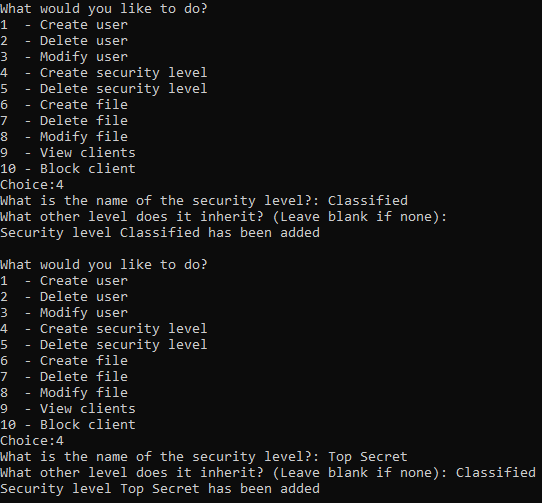


Figure - Creating a security level

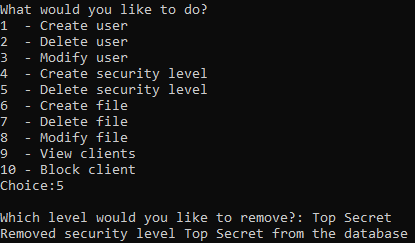


Figure - Removing a security level

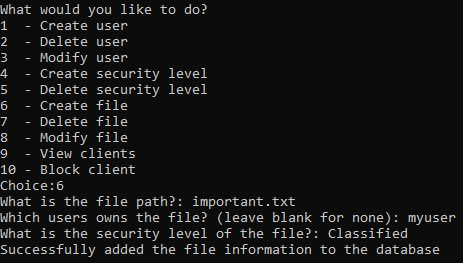


Figure - Adding a file to the database

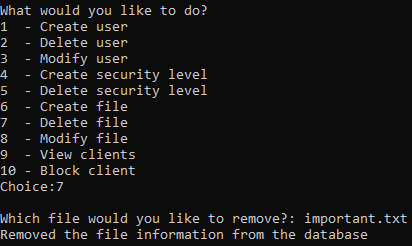


Figure - Removing a file

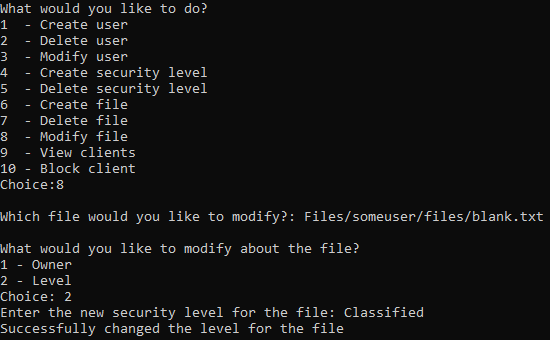


Figure - Modifying a file in the database

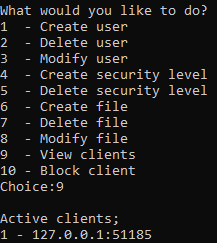


Figure - Viewing connected clients

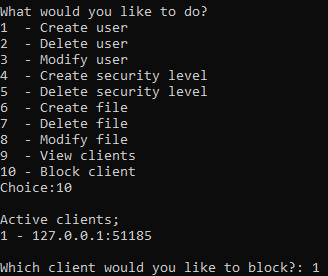


Figure - Blocking clients

Client Input

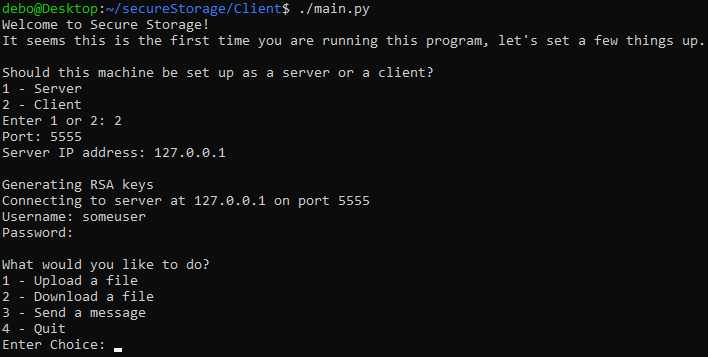


Figure - Setting up the client

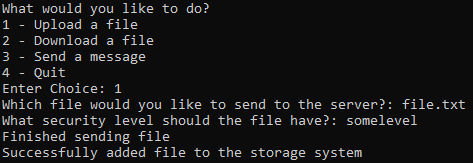


Figure - Uploading a file to the server

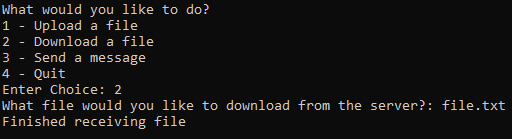


Figure - Downloading a file from the server

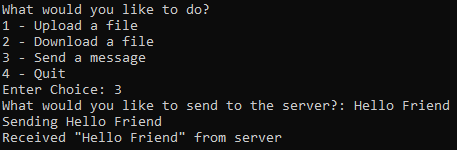


Figure - Echo message

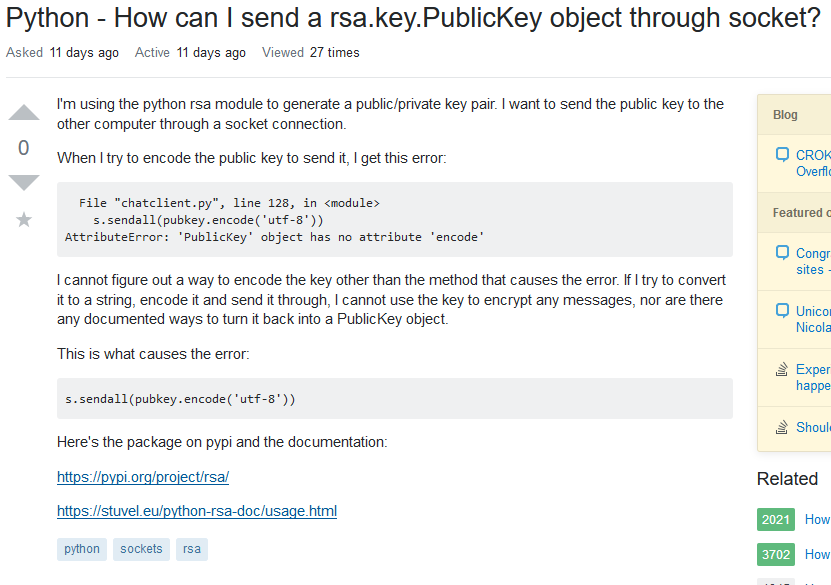


Figure - User asked about sending RSA key in plaintext and no answers talked about encrypting it with a key exchange protocol (CatCraftYT, 2019)

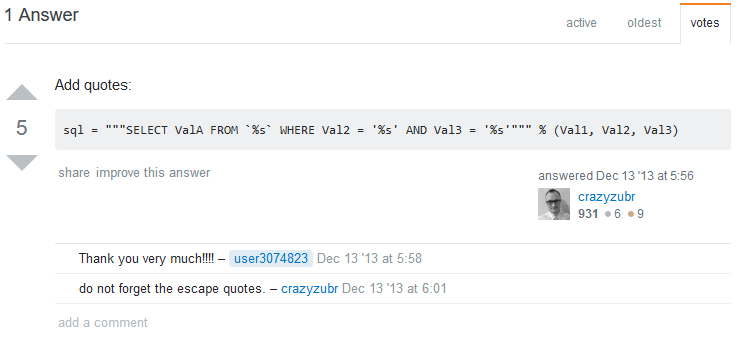


Figure - SQL statement given which could be vulnerable to SQL injection (user3074823, 2013)

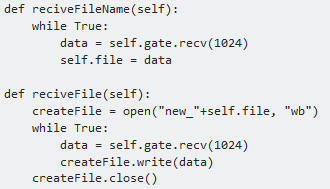


Figure - Part of a question that is vulnerable to path traversal that never got corrected in the answers (Duchess, 2012)