David Im and Alper Karslioglu

CS370

May 1, 2018

Password Authentication Device

Introduction

In this growing age of technology, it has become more important than ever to be aware of cyber security attacks and identity theft. So much of our personal information and personal assets, that are crucial to our everyday lives, are linked to a simple string of characters easily cracked. Thus it becomes more important that companies and consumers apply stronger methods of authentication to ensure information is not easily stolen. This leads to the attempt by Karslioglu and Im into testing how applicable and viable password authentication devices are.

Problem

The most popular industry standard of authentication is a simple character string password. Benefits of this system is the simplicity, access is granted to those who know the password. Everyone can handle remembering a password. However the major downside and security flaw is based on the same aspect, the simplicity. Through a number of technique, passwords can be stolen. Social engineering, key loggers, fake websites, or even brute force with the power of the modern day computers are just some of the methods that can be used to target accounts. The simpler the password, the easier it is to crack. According to Huffington Post, “17 percent of the 10 million passwords analyzed, was ‘123456’” with others amongst the top ten including “password” at eighth and “qwerty” at third [1]. So the problem not only lies in the length of the problem but also uniqueness. Furthermore, this then becomes a balance between usability and security because the simpler the password, the weaker it is; the stronger the password, the more difficult it becomes to remember. We are attempting maintain appeal for the average user while increasing the strength and the uniqueness of login systems.

Solution

This leads into Karslioglu and I’s attempt at researching and discovering if other types of authentication can be used in place of a password system. Our particular focus utilizes a model 3 Raspberry pi in conjunction with a test server as a proof of concept in order to study if a password authentication device can provide the same level of simplicity in use while increasing the security. A password authentication device is seen in consumer markets today. They are a small device that communicates with the login server. When a user wants to login into the system, the server sends a password to display onto the authentication device which the user will then use to log into the server. There are several variation to this methodology with some giving a randomized password, others refreshing the password after a certain period of time, and interactive graphic interfaces [2]. The benefit of having such a device in a security system is that it increases the factor of authentication. There are four factors of authentication: something the user knows such as a password; something the individual possesses like the password authentication device; something the user is like fingerprint recognition; and something the user does like voice pattern [2]. Adding a password authentication device would mean that the user needs both something they know (username or secondary password) along with something they have (the device itself) to be granted access. Increasing factors of authentication means a more secure system and more difficulty for a possible attacker to try and steal information. Our development project is an educational look at the difficulties involved in implementing such a device, ease of use for the user, and analysis on the increase of security it provides. Our project in particular looks at the difficulties of creating a system built around the password authentication and attempts to go further in making the system secure by doing randomly generated passwords from a large word bank and encrypting the data being sent between the devices.

Implementation

Our implementation involves a two way communication between a computer, which is a computer science desktop that represents the server and a Raspberry Pi which represents the device. Since this project is a proof of concept and security insight, we focused less on making the product as marketable as possible and more on making the security aspects as strong as possible. On real marketable device, there would be a smaller screen for display and input be more user friendly instead of an entire keyboard but network problems also prevented us from making the device more usable as detailed in the Roadblock section. The device is a Raspberry Pi Model 3 which was selected based on its variety of documentation, low cost, and the built in Bluetooth and wifi capabilities. Connected to the device is a monitor for displaying the password, Ethernet cord for connection to the network, mouse for input, and keyboard for input. The keyboard and mouse were required to be able to run the demo properly on the school network. On the server side, the computer could be any computer that is connected to the same network and is able to run C++ code. For ease of use, the server is represented as one of the computers in the 315 lab where the demo is held.

On the software side of the implementation, we had to separate parts. The code itself is in C++. Because it is just a simulation of a login server, the whole program runs on the terminal without an interface. The program welcomes the user and asks for username and password. This sets of credentials represent the traditional method of authentication and represents the factor of authentication of what the user knows. To get access to enter in the password authentication device input, one must first be granted access into the server itself. This is for adding another layer of security and making it more difficult for a potential attacker to brute force the server because it prompts for a layer of protection. Once a user enters in the correct credentials, the program prompts the user for the type of encryption they would like to use. This aspect represents a developer feature as it shows that the system can be used many types of encryption which in this project we use two types. Once the encryption type is selected, then the program prompts the user for another password which is from the password authentication device. Once the correct password is answered then the user is prompted with a success message representing access into the system.

Behind the scenes of the user interface, the program is split into two parts, main.cpp which runs the user interface and Cipher.cpp which runs the actual encryption algorithms. For the program to run, it requires a key file which must be a key of exact length of 8 bytes required by the block algorithm. It also requires a word bank to draw randomly draw new passwords which has a max size of 200 words. The word bank itself is encrypted in block cypher algorithm so that it cannot be easily identified. On the topic of encryption algorithms, our project implements two different algorithms as a proof of concepts that the software could be replaced with any type of encryption algorithm. Our algorithms were based on the same ones we implemented for our CS356 Computer Security class [3]. The first algorithm is a stream cipher using a key. Given an input string, the first bit of the input string is XOR with the first bit of the key. Because the key is a strict 8 byte character string, if the key is exhausted then the index for the key is reset to 0. This process repeats until the whole input string is encrypted. The result is an output file of unreadable characters. Decryption of the algorithm, which would be on the device side, is actually the same process of the encryption algorithm giving to simplicity the algorithm.

The other algorithm that is implemented is a block cypher [2]. Instead of encrypting the entire input string at once, block cyphers shifts the string in blocks first. The blocks are sized 64 bit or 8 bytes so if a block of data is less than a multiple of 8 bytes then it is padded the 0x80 that will be removed later. Then the blocks are XOR with the key just like in the stream cipher. But then the algorithms rearranges the blocks with two variables that hold indexes for the input text, start is the first index and end is the last. Each bit in the key is modulo with their asci value by 2. If the resulting value is 0 then do not swap anything in the input key and increment start but if the resulting value is 1, then swap the bit of that the start and end indexes refer to. If the algorithm reaches the key size, reset to 0 and continue this process until the start and end pointers collide. Partitioning the input key into blocks and rearranging within the blocks themselves adds extra layers of complexity. This makes it harder to crack the original input string even if the key is known. Additionally the decryption of the algorithm is the same as the encryption but in reverse: swap the blocks first, run XOR with the input key, and then remove padding. These two algorithms are representation of the variety of algorithms possible for encrypting the data that is being sent between the devices and shows that any other method of encryption could replace them.

For the device side of the software, it is done in Python code. The device itself runs the python code on the terminal. The code searches the current directory for an output file that is the encrypted message it should have received from the server in a process outlined below. If it cannot find such a file, it will display that it is waiting for a file and sleep for a few seconds. This process repeats until the program finds the output file where it will decrypt the algorithm. It will know which algorithm to use because in the output file, the first character in the file is an unencrypted character that mentions the type of encryption it is. Once decrypted in the methods outlined above, it will display the password for the user to enter back at the server which has been waiting for the right password.

Methodology, Imports, and Road Blocks

Our interest in creating something related to security came from our interest in our cs356 Computer Security class and the project was a good opportunity to blend those two fields together. We began by deciding to make the server side C++ because there was good support for communication through executing commands to the terminal which is what we thought we needed for device communication. The first task then was implementing the two algorithms both encryption and decryption also both in C++ and python. Once that was done, Karslioglu focused on creating the server interface while Im focused on setting up the raspberry pi interface and working on the device communication. For the code itself, the two algorithms were learned from the CS356 class and subsequently the CS356 textbook [2] [3]. All the other code used C++ and python main libraries without importing any parts.

This is where the biggest roadblock of the project was. Eventually we got to the point where we had every part done except for the communication between the server and the device. The major roadblock was that we could SSH into the raspberry pi when it was not the computer science network. But because the demo was in the 315 lab and that the computer science department set up the network in a certain way, we could get internet access through the Ethernet however the other network computer could not find the raspberry pi meaning even SSH was not possible. We had followed the guidelines to setting up the static IP address on the rubric but this semester the network had changed meaning the old method was no longer possible as confirmed by the Sean and Matt. Eventually, with their help, we were able to SSH into the device in the 315 lab but only after a special command and right before the deadline. So to keep in line of keeping this as proof of concept, the communication between the two devices is through SFTP to transfer the encrypted message to the device. This is also the reason why the device still requires a keyboard and mouse is to execute the static IP address commands because on restart, the IP address resets. Though this is not a realistic way to implement this in a product nor is it a properly located in the code, we wanted to show that every part was functional except the network connection from the differing network setup.

Security Flaws

Being that the project is centered on a security topic, we thought it most appropriate to analyze the security flaws within the system. The device, with having the server send the password to the device for the user to enter, is itself a layer of security because any attacker must have the device to be able to get into the system. On top of this, we added traditional login credentials which boosted the system to two factor authentication. Then the password itself is generated randomly from the word bank each time the user tries to enter the system thus 200 combinations of passwords within such a small window makes it hard for an attacker to randomly guess the password from the word bank. This is also why the word bank itself is encrypted with a block cipher because then the attacker would have to crack the encryption on the word bank to start randomly brute forcing combinations. Finally, the strongest security measure we implemented is the fact the data between the system and device is encrypted meaning that even if an attacker were to access between the traffic server and device, they would have to decrypt the algorithm which is unreadable ciphered characters.

Though those systems in place are strong measures to dissuading attackers, there are still a plethora of methods of attack and vulnerabilities still present in the system. Majority of these were found during the process but were unable to be addressed due to time constraints. The largest flaw in the system we created is the fact that the demo is done on the computer science network and with a computer science machine. In practice, transferring sensitive information like a password should be done on a private network so that no one could have immediate access or do a man in the middle attack to listen in on the data. This also poses a problem because the files are hosted on the computer science machines itself. Because of the nature of the demo, it was easiest to have the server on the network computers but this means that the files for the program exist on the network; leaving itself open to anyone who is able to enter our files. A further problem that comes with this is that the program files and key files were not encrypted themselves. Again due to time restraints and the nature of the demo, it would have been very difficult to run the program file while also keeping it encrypted. There is also the problem that the files would be encrypted with a key which then means that key file is vulnerable and needs to be encrypted with a key which feeds into an endless loop. A proper implementation would asymmetric encryption with public keys because each time a user wanted to access the system or in our case we want to encrypt the system, we need to create “only a public key and private key”. This means that when we want encrypt a system, we lock it with a public key and only those who have the proper private key would be able to access the files; keeping everything contained with one key [4].

If we were to be given more time, there are several security additions that could be added to address the various flaws. The first as mentioned before is a public and private key to encrypt both device and server files and make sure that they are not just open for anyone with access to the system. Also mentioned before, in a realistic scenario, the entire system would be on its own controlled network unlike the open computer science network. There are smaller additions that we considered like a message system to text message a number when the system is being accessed along with lock out after a certain number of attempts. The biggest aspect that we mentioned we were going to attempt during the progress report but ran out of time because of the networking issues is the ability for the device to signal back to the server and generate a new password. This would allow for additional security by refreshing the password frequently and reduce time for brute force.

Conclusion

It is essential with how integrated technology is becoming that society find new ways to secure our information over such fragile systems currently in place. What we attempted to explore was different ways of authenticating users in more secure ways while keeping the simplicity of use that simple passwords provide. By exploring a proof of concept of a password authentication device with a raspberry pi, we have learned that the device itself is sufficient at providing more security. For the user, it is still as easy as entering a password with just the added cost of keeping a device around but for the benefit of being more resistant to brute force and identity theft attacks. However the downside is a large amount of added overhead to the system designers because there are multiple requirements. There is added need to secure both the server and the device from interference and theft. Additionally the data itself must be sent in a secure way. With a host of additional attack vectors shows that though we need to move away more from the reliance on outdated and insecure methods, we must be willing to undertake the increase cost of added security.

Bibliography

1. Golgowski, Nina. “The Most Common Passwords In 2016 Are Truly Terrible.” The Huffington Post, TheHuffingtonPost.com, 18 Jan. 2017, www.huffingtonpost.com/entry/2016-most-common-passwords\_us\_587f9663e4b0c147f0bc299d.
2. “Chapter 3 User Authentication.” Computer Security: Principles and Practice, 3rd Edition, by William Stallings and Lawries Brown, Pearson, 2014, pp. 72–110.
3. Gersch, Joseph. “Programming Assignment 1.” Programming Assignment 1, CS356 Computer Security, www.cs.colostate.edu/~cs356/assignments/PA1.php.
4. Young, Bill. “Foundations of Computer Security Lecture 44: Symmetric vs. Asymmetric Encryption.” [PowerPoint slides]. Retrieved from https://www.cs.utexas.edu/users/byoung/cs361/lecture44.pdf