Music Based Authentication

Christopher Johnson, Daryl Bennett, Michael Anderson, Terry Kingston
Fall 2015

1 Introduction

A strong password is an essential part of network security. Most passwords used today are ASCII based and ask you to choose a password that contains at least one uppercase letter, one lowercase letter, one number, be at least 8 characters, have no repeated values, and have a special character. ASCII password schemes increase in strength the longer and more random they are, but the drawback is that they are often harder for our human minds to remember. As computers become more powerful the passwords required to thwart brute-force and dictionary attacks need to become longer. We wanted to create a password that would not only be secure, but also easy to remember. We are arguing that using a musical melody as a password is not only easier to remember but also more secure, not to mention more enjoyable! As part of our project, we were interested in the following:

- Would people actually use a music-based authentication system? What features/considerations would they like to have considered in an implementation?
- How secure could a music-based password be?
- What passwords would be considered insecure?

Our contributions involved:

- conducting a small online poll asking about people's' opinions about music based authentication
- creating a simple proof of concept: a small server/client Python program where we authenticate using music
- calculating how secure a potential music based password could be
- determining how susceptible our scheme would be to a dictionary attack.

2 Related Work

We were inspired by movies such as Batman, where Batman plays a few notes on his Grand Piano to gain access to the Bat cave, or on video games such as Zelda, where the main character Link has to play a small melody on his ocarina to gain access to a new part of the level. We found a few studies conducted based on audio-based authentication and nothing on MIDI-based authentication.

3 Adversary Model

Our adversary model is focused on dictionary attacks on password hashes. We assume the communication between the client and server is secure and the adversary cannot gain enough information from either eavesdropping on the communication or by impersonating the client or server. We assume the adversary has stolen the database records and is using a dictionary attack to discover user's passwords from the hashes stored in the database.

4 Methodology

We use a standard strong password protocol (TLS in our implementation) to securely communicate the password to the server once a session key is established. In our implementation, the order of the notes played is recorded and the string of notes, in order, represents the user's password. This string is encrypted with the session key and transmitted to the server. The server then hashes the password string with the stored salt for the given user, and compares the resulting hash to the password hash stored in the database to authenticate the user.

4.1 Online Poll

We all agreed that this project had a "coolness" factor and was quite the novelty. We wanted to expand beyond the confines of our own anecdotal experiences and see if other people would actually use this. We also wanted to know if non-musically inclined people would use this system. We did a short, two-day online poll to get some feedback. We asked our participants to focus on the abstract idea of a music-based password more than on a specific potential implementation. Our sample size was a little biased towards leaning in favor of our system, because we polled many musician and computer science communities on Reddit and Facebook. Out of 234 responses, 54.2% said that they would use a music-based password. 79.1% identified as strongly or a little bit musically inclined. Out of 75 people who identified as not being musically inclined, only 36% said that they would be willing to learn a musical melody to use as their password. We also left an open-ended question to allow people to give feedback and share their opinions of music based authentication. It was gratifying to see how a lot of people had the same ideas. Those in favor said that they found a musical melody easier to remember than an ASCII password, it added a level of personalization, it was more entertaining and enjoyable, and it has potential uses for the blind. The concerns (many of which we addressed in our implementation, which we'll discuss more in length in the implementation section) included someone being able to hear your password, inconvenient to use additional hardware such as a piano keyboard, drawing too much attention, worry about it not being secure enough, common melodies being easy to predict, non-musically inclined people finding it challenging, and the greater convenience of a password manager outweighing the novelty of music-based passwords.

4.2 Exploring the strength of musical notes, as a password

Our Python-based client user interface contains 38 unique notes, enterable with a standard computer keyboard, which notes, when used with our implementation, give $\log_2(38)=5.24$ bits per note. Simple alphanumeric passwords, for comparison, give $\log_2(62)=5.95$ bits per character. Using the full-size keyboard of 88 notes with our current implementation gives $\log_2(88)=6.46$ bit per note. As can be seen, increasing the note count increases the security of the implementation very little. Recognizing note duration with eight-note resolution gives us $\log_2(8)=4$ extra bits of security per note. However, using larger resolution can lead to false positives during password verification. It can also lead to making the passwords more difficult to use if the user doesn't have lots of musical experience. If playing chords is possible for the password then this gives a large increase in security. Chords can be thought of as simple a combination without repetition. Recognition of 5 note chords on an 88-note keyboard gives 88!5!(88-5)!=3.9107possible combinations and $\log_2(3.9107)$ =25.22 bits per chord. Chords however are likely to be played near the same location on the keyboard, so if a brute force attack on the password is done intelligently, this reduces the overall security of the chord implementation. The potential issue with this scheme is that, despite the fact that there is a large password space, users are still likely to pick passwords which might be easy to guess. The strength of this scheme relies primarily on users to pick notes which are unpredictable. However, we feel that the large password space and nature for music to be easily remembered makes this a stronger authentication scheme than traditional passwords. Traditional passwords also suffer from the same weakness of users picking passwords that are easy to predict (e.g. names, dates, words, etc.) In our password scheme, we face the same problem of users picking passwords like popular songs and same-note repetition. For this reason if this scheme is to be used it should also implement password policies which attempt to prevent users from picking common password patterns. The main focus should not be on telling users which passwords they can't use but rather telling users how to pick password schemes which are secure. An example of a secure password scheme with text passwords is using words instead of random characters/numbers. An equivalent example in our case would be to pick random parts of multiple songs as your password. The number of entries in Webster's New World College Dictionary (2004) was approximately 160,000 words [1]. In 2012 there were 26 million songs on iTunes [2]. So using this scheme gives much more security with 24.6 bits per song vs 17.28 bits for a word. A direct comparison between words and songs is not exact because many of the songs might be the same song performed differently but we feel that these factors don't discredit our example. Another possible password scheme would be to simply pick a musical key like C major and try to create long, interesting-sounding patterns on the keyboard. An attack targeting this method would need to identify all common patterns and musical keys and try to try all common note combinations. The strength of this scheme depends mostly on the length of the password but should help to create passwords which are easy to remember. We feel that despite the problems with this scheme (which are not unique to this scheme), it is still a more secure system when used responsibly.

5 Implementation/Experimentation

Our specific implementation was written in the Python programming language and mapped keys on the user's keyboard to different notes between F#3 and G6. The client records these notes in the order that they were played and, upon request, establishes a TLS v1.2 session with the server.

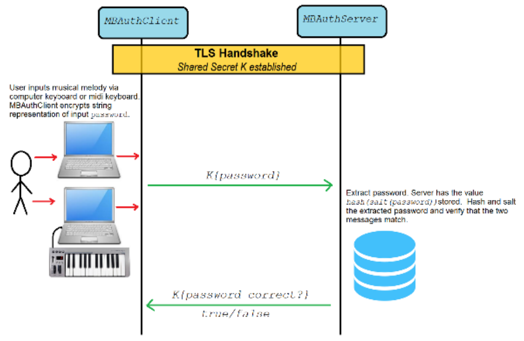
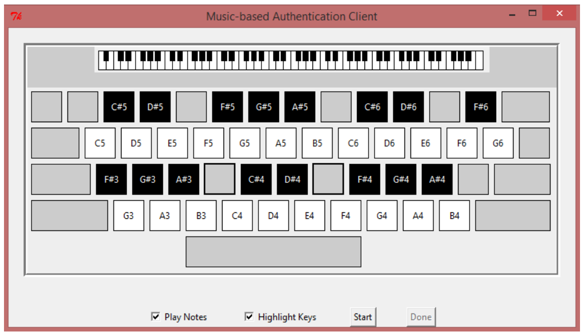

MBAuthClient
MBAuthServer
TLS Handshake
Shared Secret K established
User inputs musical melody via computer keyboard or midi keyboard. MBAuthClient encrypts string representation of input password.
K{password}
Extract password. Server has the value hash(salt(password)) stored. Hash and salt the extracted password and verify that the two messages match.
K{password correct?}
true/false

Diagram of our protocol

Messages are sent between client and server are delimited with a '\n' character. The client first sends the protocol and version in the form 'MBAUTH VERSION 002' and then follows shortly after with a 'START' message followed by the notes recorded. The notes are sent as a string followed by a newline character in the format note|octave e.g. 'F#4\n'. After the client is done sending notes, it ends the password by sending the message 'END'. The server then combines the salt with the clients note sequence and hashes the data. It then compares the hash to the one in the database. If the hashes are equal the server simply replies with 'Authenticated successfully.' on the socket. If the hashes are not equal the server simply sends 'Authentication failed!' In this implementation we didn't implement the ability to authenticate different users so this experimental protocol didn't include messages for specify the identity of the user.
Originally we planned to implement additional features into our protocol such as the ability to recognize note duration and tempo but, because of limitations with some of our python libraries, this was unfortunately not possible. Implementing additional features such as setting passwords, authentication for multiple users, and recognizing chords would require minor modifications to our original protocol such as expanding our protocol command set and using new libraries for key press capture.


Music-based Authentication Client
C#5 D#5 F#5 G#5 A#5 C#6 D#6 F#6
C5 D5 E5 F5 G5 A5 B5 C6 D6 E6 F6 G6
F#3 G#3 A#3 C#4 D#4 F#4 G#4 A#4
G3 A3 B3 C4 D4 E4 F4 G4 A4 B4
Play Notes
Highlight Keys
Start
Done

Our Python Client Interface

6 Conclusion

Our original three questions were, could this application potentially extend beyond just being a novelty, how secure could a music-based password potentially be, and what would a good password scheme consist of within the musical realm? Many people thought that a musical password would enhance their computing experience by making it more personalized to them. Typing out a password is a task, while performing a melody was an act of self-expression. Many were skeptical as to whether this would be useful or practical in real-world situations. Others were afraid that certain implementations would pose a security risk. While we didn't go very into depth in determining what passwords would be secure, we did address the issue a little bit. Our implementation focused on MIDI and computer keyboard input, but there is room for other implementations that would have more robust interfaces, and that would take in other forms of music, such as vocal audio, guitar, etc…

References

[1]M. Agnes, *Webster's New World college dictionary*. Indianapolis: Wiley Pub., 2004.

[2] Apple.com, 'Apple - Press Info - Apple Unveils New iTunes', 2015. [Online]. Available: https://www.apple.com/pr/library/2012/09/12Apple-Unveils-New-iTunes.html. [Accessed: 10- Dec- 2015].