Proceedings of the 8th World Congress on Electrical Engineering and Computer Systems and Sciences (EECSS'22)
Prague, Czech Republic - July 28- 30, 2022
Paper No. CIST 110
DOI: 10.11159/cist22.110

Covert Communications via Spotify Playlists

Claire Casalnova[1], Calista Gasper[1], Grace Lombardi[1], Daryl Johnson[1]
[1]Rochester Institute of Technology
1 Lomb Memorial Drive, Rochester NY, USA
ccc2876@rit.edu; cmg5435@rit.edu; grl1764@rit.edu; daryl.johnson@rit.edu

Abstract **-** In this paper, we utilize a music streaming platform called Spotify to create two distinct covert channels. The usage for these covert channels is based on the ability for various users to access publicly available playlists. Users can use these playlists to send and receive covert messages. The first method uses the first letter of an album name to construct a message, and the second uses song's explicit tags to create binary messages. We concluded that these covert channels were successful. This is because they could send messages with adequate obscurity that outside actors would not be privy to the covert message contained.

Keywords**:** covert communication, music streaming platform, encoding, decoding

1. Introduction

The need for secure and secret communication is always present in our world, which is why discovering new ways to accomplish this is a necessity. At the same time, the popularity of music streaming platforms has increased tremendously. Hence, finding users who actively listen to playlists on public platforms is not an abnormality in our society. Pairing together the covert communication channel and the public music streaming platforms creates a covert communication channel that is hidden in plain sight.

Considering the nature of the music platform, the sender can start conveying the message by simply playing their playlist, with high confidence that it will not be detected by adversaries. A shared secret between the sender and receiver is all that is needed to understand what to look for during transmission. We decided to create this covert channel using the platform Spotify.

In our implementation, we created two different methods for message relay and retrieval. The first method builds messages using album names for songs in a playlist. As most users are more concerned with song names than album names, we saw this frequently overlooked area as an ideal place to embed messages. The second method utilizes the presence of explicit tags to craft binary strings which will decode to reveal the secret messages.

2. Related Work

According to J. Miller, "Covert channels are a means of communication between two processes that are not permitted to communicate, but do so anyway, a few bits at a time, by affecting shared resources [1]." Covert channels in social media are a quickly growing method to send secret messages. Some examples of covert channels in social media include image steganography on Instagram, WhatsApp messages, and music streaming services such as Spotify [2] [3] [4]. We were inspired to implement this covert channel based off of the first word of a song encoding and hex encoding that a group of RIT students originally created. In their method, they used the Spotify API to create a playlist where the first word of each song in a playlist created a message. Their second implementation used hexadecimal characters embedded in the track ID to send a message [5]. Compared to previous work, we have used different methods to create a novel covert channel using Spotify.

3. Background

3.1. Spotify

Launched in 2006 with an IPO date of April 2018, Spotify is the world's largest audio streaming and media service provider with over 381 million monthly users [6]. Spotify has two user options: premium users (paying) or standard (non-paying/free) users. Both user options allow streaming of roughly 70 million songs and 2.9 million podcasts. Users also have the ability to create their own playlists, title them whatever they want, add cover photos, and share them with other users.

3.2. Friend Feed

Spotify contains the ability to friend other users which allows you to see the public playlists the user has created. This also adds the friend to the live friend activity feed which shows what song they last listened to, the album or playlist the song is in, and how long ago they listened to it.

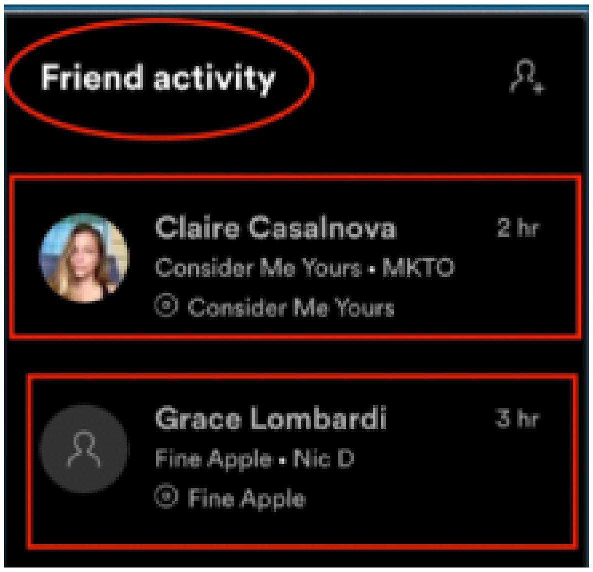


Fig. 1: Example of Spotify Friend Feed.

4. Design

4.1. Album Letter

For our first method of implementation, we decided to create a covert channel using the first letter of an album name. We decided it would be harder to recognize a formatted message if we used the name of albums as compared to song names. This is because when users open a playlist, more often than not they are more interested in the individual songs than the albums they come from. With this understanding, we decided our covert channel would have a Low Probability of Detection (LPD) where LPD represents the ability for a channel to be detected by adversaries.

4.2. Binary Explicit Tag

For our second method of implementation, we wanted to create a covert channel that was even more inconspicuous than the Album Letter method. We accomplished this through the use of explicit tags associated with some songs. Unlike the Album Letter, this Explicit Tag method does not use plain-text ASCII letters to represent a message - encoding and decoding values is required.

We decided to let the presence of an explicit tag represent a 1, and the absence represent a 0. With this defined, we would create binary strings that when converted to base 10, would equate to an ASCII character's numerical value. This is how entire messages are built and disguised.

5. Proof of Concept

5.1. Spotify API

Spotify has an open-source API (Application Programming Interface) that we leveraged in order to create playlists, add songs, and start playing the playlists.

5.2. Spotify Buddylist

The current Spotify API does not have support for an API endpoint to return data from the Friend Feed. However, we leveraged an open-source tool called Spotify Buddylist to return data regarding the Friend Feed [4]. This open-source tool was written in JavaScript and returns information such as the user, current song, and playlist information. Since our proof of concept was developed in Python we utilized a package called the Naked ToolShed Shell module [7]. This module allows us to execute JavaScript code within a Python script [8].

5.3. Encoding

5.3.1 Album First Letter

The Album First Letter Encoder method is a very simple method for encoding the messages into playlists. This method works by encoding each letter of the message into the first letter of the name of the album. To make this method more covert, we ensure that the album name being used is longer than a single character. This method is completed by initially asking the user for the message that he or she would like to send.

After the message is created, a new playlist is then created. A random word generator is used to name the playlist. This allows the channel to be more covert and not stand out as an anomaly as most Spotify playlists have unique names. After the playlist is named, albums are then searched for matching the letters of the message using the Spotify API /search endpoint. Then, once an album is found that matches the criteria, the number of songs in that album is determined using the API. A random number is generated between 0 and the number of songs and the song at this index of the playlist is chosen and added to the playlist.

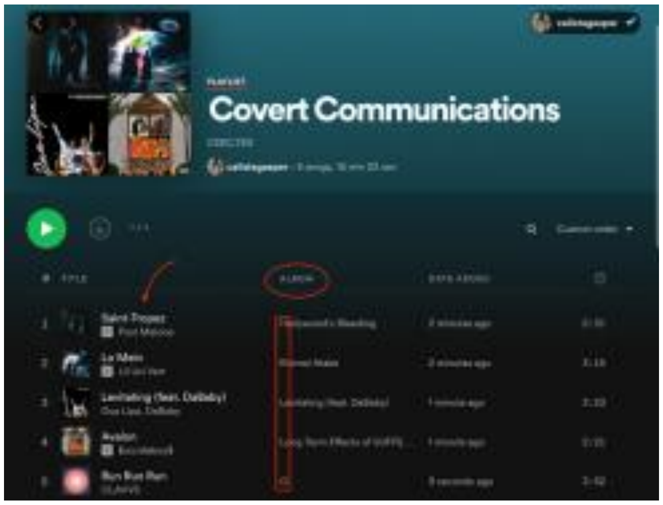
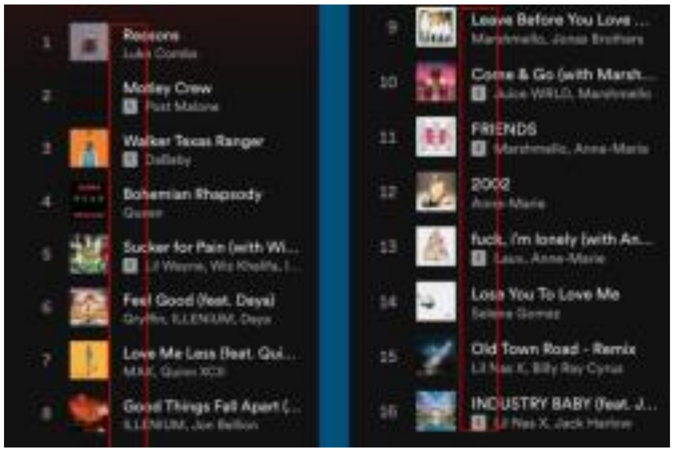


Fig. 2: Example of album name method sending message "hello"

5.3.2. Binary Explicit Tag

The Binary Explicit Tag method is a slightly more complex and more covert method than the prior method. This method utilizes the explicit tag of songs to encode the message. The explicit tag in Spotify means that a song contains expletive words. Utilizing the tag as binary we determine that if the song has an explicit tag that is a 1 and if there is not an explicit tag that is a 0. To encode the message, we convert it to binary and then use the Spotify API to search for songs. The songs are searched for using a random word generator and from the results list a random song is chosen to be added. If the song is explicit and the expected binary value is 1, the song is added to the playlist. In the same way, if the song is not explicit and the expected value is 0, the song is added. If the tag and expected value do not match, another search is performed.



1 Recsons
Luke Combs
2 Motley Crew
E Post Malone
3 Walker Texas Ranger
E DaBaby
4 Bohemian Rhapsody
Queen
5 Sucker for Pain (with Wi...
E Lil Wayne, Wiz Khalifa, I...
6 Feel Good (feat. Daya)
Gryffin, ILLENIUM, Daya
7 Love Me Less (feat. Qui...
MAX, Quinn XCII
8 Good Things Fall Apart (...
ILLENIUM, Jon Bellion
9 Leave Before You Love ...
Marshmello, Jonas Brothers
10 Come & Go (with Marsh...
E Juice WRLD, Marshmello
11 FRIENDS
E Marshmello, Anne-Marie
12 2002
Anne-Marie
13 fuck, i'm lonely (with An...
E Lauv, Anne-Marie
14 Lose You To Love Me
Selena Gomez
15 Old Town Road - Remix
Lil Nas X, Billy Ray Cyrus
16 INDUSTRY BABY (feat. J...
E Lil Nas X, Jack Harlow

Fig. 3: Example of binary method sending message "hi"

5.3.3. Playing the Playlist

For both of the aforementioned methods, the playlist begins playing in the same way. For a user that does not have a Premium account, the Spotify API does not support playing music. To work around this, we utilize the Python Selenium [9] package. This package is a web browsing automation tool [10]. We simply have the webdriver open the Spotify web player, log in to the user account, and begin playing the specified playlist. If the user sending the message does have a Premium subscription, the playlist is started by an API endpoint call.

5.4. Decoding

Decoding works similarly for both methods of encoding. The receiver begins the decoding process, and the program waits for the sender's user to appear within the Friend Feed. Once the user appears in the friend feed, the appropriate decoding method is used based on how the message was encoded. For the Album First Letter method, each song is read, and the first letter of the album is added to a message string. For the Binary Explicit Tag method,

every song from the playlist is read. If the song is explicit, then a 1 is added to a string of binary. If the song is not explicit, a 0 is added. The binary string is then converted to ASCII. The message is then outputted to the terminal for the receiver.

6. Analysis

6.1. Robustness

Robustness is described as the survivability of a covert channel [4]. Both the album name and binary encoding channels are considered noiseless because the message that is sent is exactly the same as the message that is received [7]. The album name channel lacks in covertness due to the message being showcased in plain text. If another user noticed that the album names spelled out a sentence or phrase, this would severely impact the level of covertness of the channel. However, one of the communicating parties would still be protected as the third-party user would not know who was supposed to be the intended receiver. The binary encoding method is far more covert than the album name method. Since the message is encoded into the explicit tags, a third-party user would not be able to decode the message just by looking at the playlist. Although it adds complexity to encode this method, it allows the channel to be more robust.

6.2. Detectability

One method of detection that could inhibit our channel would be irregular behavior. When the playlist is created from the message, the songs are added relatively quickly compared to how fast a human user would be able to add the songs. This could possibly be detected by Spotify and could block the user from usage of Spotify. The covert channel would then no longer be a viable form of communication.

6.3. Limitations

One of the main challenges that was encountered during development was working with the Friend Feed. Users do not show up in the Friend Feed unless they are listening to the music on a mobile device or the desktop application. This creates the need for user interaction to start the playlist to send the message to the intended receiver. Additionally, the Friend Feed information that is pulled down does not indicate if the user is currently listening to the song or was listening to it in the past. Due to this, further work would need to be done in performing a handshake to communicate that a message has been received.

7. Future Work

As of right now, the receiver is not equipped to constantly be monitoring for message transmission updates. So this means that the receiver is not capable of consistently running in the background, ready to receive information at any time. The sender and receiver programs work on a one-time basis and would have to be re-run for every new message. We would be interested in extending constant polling functionality to the receiver to circumvent this.

A second idea for future work is implementing handshake communication similar to that of TCP network packets. Currently, the communication between sender and receiver mimics UDP network communication. There is no established communication between the sender and receiver that signifies a message was successfully transmitted and received, and the receiver is ready for more information. The sender currently begins transmitting the message without acknowledgement from the receiver that it’s listening. So, in the event that further work be done on this covert communication channel, we believe incorporating a handshake communication structure would improve functionality.

Finally, we propose incorporating a genre filter for the sender when randomly selecting songs during playlist creation. Currently the sender’s program randomly selects a song that matches current specifications such as album title name length, the first letter of the album, and occasionally the presence of the explicit tag. We thought it a good idea to incorporate a genre filtering system, so playlists didn’t have, for example, Skrillex beside The Wiggles. This would help make playlists look even more convincing and inconspicuous.

8. Conclusion

In conclusion, we were able to successfully implement both proposed methods for our covert communication: the Album First Letter and the Binary Explicit Tag. Several challenges were encountered while building this channel, but nothing that derailed our implementation completely. Overall, the Binary Explicit Tag method proves to be a strong covert channel with high level of covertness but higher complexity, whereas the Album Name method providers lower covertness and lower complexity.

References

[1] Millen J. 20 years of covert channel modeling and analysis. In: Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No.99CB36344); 1999. p. 113-4.

[2] Ning J, Singh I, Madhyastha HV, Krishnamurthy SV, Cao G, Mohapatra P. Secret message sharing using online social media. In: 2014 IEEE Conference on Communications and Network Security; 2014. p. 319-27.

[3] Gasimov V, Mustafayeva E. Implementing Covert Channels to Transfer Hidden Information Over WhatsApp on Mobile Phones. American Journal of Engineering and Applied Sciences. 2020 01;6:32-5.

[4] Brown E, Yuan B, Johnson D, Lutz P. Covert channels in the HTTP network protocol: Channel characterization and detecting man-in-the-middle attacks; 2010.

[5] Egbert C, Alhenaki F, Johnson D. Leveraging a Music Streaming Platform in Establishing a Novel Storage Covert Channel. In: 2020 IEEE 45th Conference on Local Computer Networks (LCN); 2020. p.437-40.

[6] About Spotify; 2021. Available from: https://newsroom.spotify.com/company-info/.

[7] MD NCSCFGGM. A Guide to Understanding Covert Channel Analysis of Trusted Systems. Defense Technical Information Center; 1993. Available from: https://books.google.com/books?id=7JmaAQAACAAJ.

[8] Toolshed: Shell;. Available from: http://naked.readthedocs.io/toolshed$$shell.html.

[9] Valeriangalliat. Spotify-Buddylist: Fetch the friend Activity Spotify feed.. Valeriangalliat;. Available from: https://github.com/valeriangalliat/spotify-buddylist.

[10] Selenium with python;. Available from: https://selenium-python.readthedocs.io/.