Covert Channels

Matthew Kusman

Overview

The covert channel I am researching and conceptualizing involves the implementation of file systems and their relationship with processes. More specifically, the Linux operating system and attempting to open a file for any mode (Read, write, append). My experience in systems programming and operating systems programming gave me the idea, learning how processes read and write to files. The general idea is that one process (P1) with higher privileges maintains control of a file as the sender. When P1 opens the file and P2 attempts to open it, it fails. P1 can then communicate by opening/closing the file. This represents a form of binary where P2's ability to open the file is a 1, the inability is a 0. This creates a noisy shared resource covert channel with low bandwidth because communication happens bit by bit, and can be time intensive[1].

- Noisy: the file may be opened by another process, thus resulting in an incorrect transmission. The opening may fail for other reasons[2].
- Low bandwidth: while the actual speed varies based off system hardware, in general opening/closing files is rather time intensive and would make this method slow.
- Shared resource: this channel utilizes the file's shared status as a means to communicate, making it a shared resource channel.

Implementation

Below is pseudo code for the implementation:

Sender:

```
File file;
bool message;

if (message){ //to send 1 bit
   while(true){
      open(file);
   }
}
else{ //to send 0 bit
   close(file);
}
```

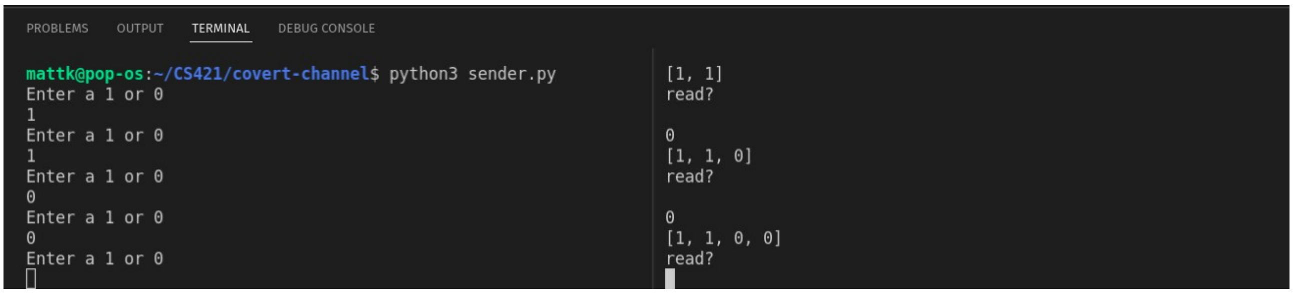
Reader:

```
File file;
bool response;

Message = open(file)
```

Explanation:

The sender attempts to open the file continuously if it is sending a 1 bit. Otherwise, it closes the file. The receiver/reader attempts to open the file. If successful, it adds the bit to the whole message.



```
mattk@pop-os:~/CS421/covert-channel$ python3 sender.py
Enter a 1 or 0
1
Enter a 1 or 0
1
Enter a 1 or 0
0
Enter a 1 or 0
0
Enter a 1 or 0
```

```
[1, 1]
read?

0
[1, 1, 0]
read?

0
[1, 1, 0, 0]
read?
```

Refer to attached source code

Defense
One possible form of defense would be to allow multiple processes to open a file at once. This however leads to the possibility of data corruption and larger issues. Another perhaps more viable defense is to not notify the process if the file was opened successfully. Rather, enter the process into a queue waiting for the file.

Counter: One way to counter this defense is to modify how the message is send/read. Rather than basing it off the success of opening a file, base it off the success of reading or writing to the file. This does not mean reading/writing content to the file, but rather the capability to do so.

Time Log

Task	**Time**
Research	~2 hours
Implementation	~0.5 hours
Documentation	~2 hours
Research (finding idea)	~1.5 hours
TOTAL:	6 hours

References:

1. https://www.ibm.com/docs/en/aix/7.2?topic=channels-bandwidth-guidelines
2. https://arxiv.org/pdf/1306.2252.pdf#:~:text=Effect%20of%20Noise&text=With%20covert
%20channels%2C%20each%20symbol,rates%20in%20a%20communication%20channel.