Morse Ping: A Method of Timing Channel Steganography

Cheri Walker-Owens, Hussain Alhassan, Jacob Helou and Gabriel Aguirre

30 April 2018

Abstract

Cryptography is secure in that it encodes a message to make it unreadable by unintended viewers. However, cryptographic keys always pose the risk of being cracked. Steganography uses a different approach to secure communication by hiding the existence of the message entirely. Steganography can be implemented through different mediums such as images, audio files, and even through the exploitation of protocols. This project uses timing channel steganography to secretly send messages in a manner similar to Morse code.

Problem

Current methods of communication over networks are either completely unsecured or are protected using cryptography. While the current way communication is set up is not a major concern, it is still vulnerable to attacks using packet sniffing. Packet sniffing allows eavesdroppers to see what is being sent between and within networks. If the information contained within the packets, eavesdroppers are immediately able to read it. If the information is encrypted, attackers can still attempt to crack the encryption method used to protect the data.

Approach to Solving the Problem

Our approach to solving the issue is to use steganography methods to communicate. Steganography can be argued to be slightly more secure than cryptography because, while cryptography makes messages unreadable, steganography hides the existence of the message altogether. Our program makes use of a "vulnerability" within the UDP protocol to send messages secretly.

Vulnerability

In essence, networking consists of sending information from one user/machine to another user/machine. Rather than sending the information all at once, information is broken down into packets and sent in a sequence. How and when packets are sent is dependent on the protocol being used to send them. A possible "vulnerability" that can be exploited for secret communication is the manipulation of the timing that packets are sent.

If packets are manipulated to be sent at certain times, this can be used for purposes of sending communications undetected in a way similar to Morse Code. An eavesdropper would be more likely to focus on the information the packets contain, rather than the timing between the

packets, to decipher any secret messages. This method of communication is similar to steganography in that the actual message is not encrypted (other than being converted into some sort of non-numeric or non-alphabetic form), but rather the fact that the communication is happening at all is made to be undetectable.

Background

Current research classifies this method of communication as *network steganography,* which is defined as, "the hiding of information within ordinary network transmissions." Methods of network steganography include exploiting TCP/IP, higher layer applications (such as Skype, Bittorrent and intercepting Google Search suggestions) and cloud computing ("The Growing Threat of Network-Based Steganography"). The main threats that network steganography pose are that it's difficult to detect and can operate for long periods of time (Houmansadr 6).

Network steganography can also be broken down into sub-groups, the one in which this project applies to being *timing channel steganography.* Timing channel steganography is considered to take place when "covert communications is established by the artful modulation of a shared resource over a prescribed time period in order to effect an information exchange" (Collins and Agaian 4). Different methods of timing channel steganography include manipulating packet transmission rates, changing sequence timing, and modifying the order of packets or packet loss (Collins and Agaian 6). Timing analysis can be used to attempt to detect the use of timing channel steganography (Houmansadr 22).

There are two approaches to removing the threat of network steganography: detecting first, then removing (known as the passive warden threat model) and modifying traffic regardless of suspicion (known as the active warden threat model) (Houmansadr 23). Currently, anomaly

detection is being developed to combat this type of steganography (Mazurczyk et al.). However, network steganography methods can also be altered to use shared keys to make their existence harder to detect (Aviv et al. 1).

Our project consists of two separate programs: a client and a server. The client takes a message, converts it to binary, and sends the message to the server through packets. Rather than the packets containing the message within them, the packets contain irrelevant data and the timing between the packets indicate the binary values of the message. It is similar to morse code in that the packets act as the sound tone used to relay the message and the delay between the packets are the equivalent of the length of the tone (dots and dashes). The server then deciphers the message and converts it back to plaintext. Our source code can be viewed here: https://github.com/hussain-alhassan/morse-ping.

How Our Approach Solves the Problem

Our approach solves the problem by disguising the fact that there's a message being sent between two parties at all. The user enters a message on the client side. The message is converted into its binary representation. The program sends one digit at a time to the server. If the digit is a 0, it sends 2 packets with no delay in between. If the digit is a 1, it sends 2 packets with a small delay between each packet. The actual data in the packets we send is mostly irrelevant, but for extra sneakiness, we pull truly random numbers from the website "random.org" into the packets. This way, people sniffing or eavesdropping on the packets will see the numbers, and incorrectly assume that's the message. After the whole message has been sent, the client program sends 3 packets in a row to the server, to signal the end of the message.

Meanwhile, the server receives all the packets, and stores the timestamps of the packet arrivals in an array. When it receives the last 3 packets that signal the end of message, it drops the last three packets, because they're just a marker for the end of message. The server looks at the timing between each 2 packets, and if the timestamps are close enough together, it's a 0. Otherwise, it's a 1. Then, it takes the binary digits, and reconstitutes them into a message, displaying it to the user.

Testing Procedure

Our testing procedure included extensive client and server interaction between two terminal windows on the same computer and between two different computers. After fixing bugs and working through any errors when running our programs, we tested our progress by running our client file and our server file and attempting to send packets from the client to the server. After achieving a successful connection between the client and server, we would ensure that our message was being broken up into binary and the gaps between the 1's and 0's were being correctly analyzed by the server by printing the bits and timestamps. On the server side, we were able to determine when a 1 or a 0 was being sent by the gaps between the packets. Once this was achieved, we made efforts to optimize our code for speed.

Future Work

Future work for our project would include several things. Right now, the program isn't multithreaded, which means that the server side doesn't have a GUI at all. Multithreading would enable GUI functionality for both the client and server. Another way we could improve the code would be to take into account the location of the two users, and compensate for any latency or lag based on their distance. People communicating between two different buildings on Uindy's

campus would experience different levels of latency when compared to two people communicating between different states. We could also enable the user to choose a cover message to send with the message, instead of the random numbers that it currently sends. Then, we could encrypt the cover message. If an eavesdropper sees it, they will waste time attempting to decrypt it, and when they finally do, they will have decrypted the wrong message. We could also encrypt the actual message we're sending before we transmit the packets. Then, the server could decrypt the message on the other side, perhaps using the number of end of message packets it receives as the key.

Workload Distribution

The following outlines the contributions each group member made to the project.

Hussain:

- Created and managed GitHub repo
- Helped teammates on using GitHub professionally
- Wrote most of the server code
- Added GUI to the server (did not work, then removed it)
- Contributed on the client GUI and client code in general
- Organized most of the meetings physically and remotely
- Massive testing and timestamp analyzing

Gabriel:

- Made initial GUI windows
- Integrated client/server code with GUI windows
- Bug fixes in GUIs and in client/server files

- Extensive code testing
- Problem section of presentation
- Mitigation section of proposal
- Final paper contribution

Jacob:

- Wrote a large portion of the proposal
- Helped with presentation and wrote parts of presentation out
- Wrote future work and how our approach solves the problem sections
- Helped with brainstorming and ideas
- Helped with explanation and understanding for other group members

Cheri:

- Client's base code (sends packets with sleep() function)
- Modified client to allow sending spaces and special characters
- Bug fixes
- Vulnerability and Background sections of the proposal
- Approach section of presentation
- Contributed to final paper

Works Cited

Aviv, Adam, et al. "Steganographic Timing Channels." *University of Pennsylvania Department of Computer and Information Science Technical Report,* no. MS-CIS-11-18, 2011, p.p. 1-16, https://repository.upenn.edu/cgi/viewcontent.cgi?article=2010&context=cis_reports.

Collins, James and Sos Agaian. "Trends Toward Real-Time Network Data Steganography." https://pdfs.semanticscholar.org/c9ac/b053d6ccfc21ca608e97e450ce9f90fada37.pdf.

Mazurczyk, Wojciech, et al. "Towards Steganography Detection Through Network Traffic Visualisation." https://arxiv.org/pdf/1208.2861.

"The Growing Threat of Network-Based Steganography." *MIT Technology Review,* 18 July 2014, https://www.technologyreview.com/s/529071/the-growing-threat-of-network-based-steganography/.

Houmansadr, Amir. "Information Hiding: Covert Channels." 2015. Microsoft Powerpoint file, https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=6&ved=0ahUKEwiWy9nC6JTaAhVPMqwKHSLoBXcQFghRMAU&url=https%3A%2F%2Fpeople.cs.umass.edu%2F~amir%2Fcourses%2FCMPSCI660-SP15%2Fslides%2F3-Covert-channels.pptx&usg=AOvVaw0iC78MGxa_dutSuXwdKZLW.

"How Can I Make a Time Delay in Python?" *Stack Overflow* Web forum, https://stackoverflow.com/questions/510348/how-can-i-make-a-time-delay-in-python.

"How to Use Threading in Python?" *Stack Overflow* Web forum,

https://stackoverflow.com/questions/2846653/how-to-use-threading-in-python.

Random.org, https://www.random.org/.

"TKInter." *The Python Wiki*. Python Software Foundation. Web. 30 Apr. 2018, https://wiki.python.org/moin/TkInter.

"UDP Communication." *The Python Wiki*. Python Software Foundation. Web. 30 Apr. 2018, https://wiki.python.org/moin/UdpCommunication.

YatriTrivedi. “How to Forward Ports on Your Router.” How-To Geek, How-To Geek, 3 July 2017, www.howtogeek.com/66214/how-to-forward-ports-on-your-router.