Masquerading a Wired Covert Channel into a Wireless-like Channel*

Mina Guirguis
Computer Science Department
Texas State University
San Marcos, TX 78666, USA
msg@txstate.edu

Jason Valdez
Computer Science Department
Texas State University
San Marcos, TX 78666, USA
jv1150@txstate.edu

Abstract

This paper presents a novel method to implement a covert channel that is based on inducing dynamics to convey a covert message. These dynamics are induced in a manner that emulates the normal operation of a hypothetical virtual channel. As a case study, this paper focuses on a scenario whereby TCP packet losses are induced to change the behavior of the Additive-Increase Multiplicative-Decrease (AIMD) congestion control mechanism of TCP to convey the covert message. From the outside, the TCP connection appears to be a normal connection that is traversing a congested link or a lossy wireless link. However, the sender, through monitoring the packets that get retransmitted, will decode the covert message. Packet losses are induced based on a hashing algorithm with specific hash patterns that are chosen a priori to emulate a specific loss rate. We have assessed the performance of this covert channel through simple analysis, simulation and real Internet experiments. We illustrate the existence of an optimal packet drop rate that maximizes the throughput of the covert channel.

1 Introduction

Motivation and Background: A covert channel is a means for two parties (e.g., Alice and Bob) to exchange information in a hidden manner. Covert channels are typically used to violate security policies. Thus, they present an ongoing threat to almost any entity that has sensitive data and wishes to keep it within its protected boundary. If Alice and Bob decide to use an encryption scheme, it would likely raise suspicion for someone monitoring the traffic between them. In essence, they need to masquerade their data exchange over, what appears to be, a normal communication channel. For the sake of illustration, we assume that Alice resides within a secure entity and wishes to maliciously leak some message (a string of 0's and 1's) to Bob.

Current covert channels can be broadly classified into two categories; storage-based and timing-based. In a storage-based covert channel, Alice hides the covert message in the headers of the packets (TCP and IP headers, for example). Clearly, not all the fields can be used since the use of some fields may prevent the packet from being routed correctly, or may raise a flag at intermediate nodes. If Bob knows which bits in which fields carry the hidden message, he can recover the secret message.[1] Research works have investigated the usability of the different fields in the TCP/IP headers [1, 11, 14], and other protocols like ICMP and HTTP [12, 13] in their potential to carry a secret message. Other studies have focused on detecting those covert channels and scrambling some fields to render those covert channels useless [7].

In a timing-based covert channel [3], Alice and Bob agree beforehand on a timing interval and a starting time (due to clock synchronization issues, the starting time may be an event such as the arrival of the first packet). Each one of them will divide time into intervals. Whenever Alice would like to transmit a 1, it would send a packet to Bob during that interval. To transmit a 0, Alice would remain silent during that interval. Bob, upon observing when he receives the packets, would be able to recover the secret message. The packets used over this transfer are normal ones, in the sense that they do not include any covert data within.

*This work was supported by an internal REP grant from Texas State University - San Marcos.

[1] Hiding information inside the payload falls in the area of steganography which is not our main focus in this paper.

Network delay and packet loss may cause Bob to get a different message than the one intended. With the proper choice of the timing interval, Bob can reduce such probability. Also, with the use of error correcting codes, Bob can recover from some errors. There has also been some works on detecting timing based covert channels [2, 3, 8].

Stealthy Covert Channel: We consider a new approach to implement a covert channel that relies on *dynamics to convey the secret message*. In particular, we consider a normal communication channel between Alice and Bob that uses TCP [4]. Again, Alice will be the sender of the secret message while Bob will be the receiver. Unlike traditional covert channels where the sender is the one who is transmitting the secret message, we consider here the opposite. So Bob would be the one sending packets to Alice, yet Alice would be the one who is sending the secret message. The main reason to implement the covert channel in this manner is to add to the complexity of detecting this covert channel since Alice is actually the receiver in this TCP connection.

The scenario proceeds as follows: Alice and Bob open a normal TCP connection and Bob starts sending Alice data (say a photo of them together taken in the last conference they attended). To transmit a 0, Alice waits until she receives a data packet from Bob that has a particular hash value and drops it. To transmit a 1, Alice waits for a packet that has another hash value and drops it[2]. Notice that the impact of those losses forced by Alice, in essence, resembles a channel that is dropping packets independently from congestion (for example, due to a wireless channel or transient congestion on non-bottleneck links due to cross-traffic). So, in effect, the communication between Alice and Bob resembles a normal TCP connection that traverses a wireless link at Alice's side. Moreover, the particular hash values are chosen carefully, so that the gaps between induced drops would resemble a hypothetical channel with a specific packet loss probability.

Once the TCP connection, at Bob's side, observes these packet losses, TCP would react to them the normal way TCP reacts to any packet loss, by halving its congestion window. Notice that the network may also drop packets (due to congestion), which may contain the specific hash values. This would cause Bob to recover a different message from the one intended. The choice of the hash values will be our first line of defense against this case. The second line of defense will be the use of error correcting codes, so Bob can still recover the message.

[2]We will explain later, how these hash values are computed.

Paper Organization: In Section 2, we describe the overall design of the covert channel. In Section 3, we present implementation results from our evaluation experiments. Finally, we conclude the paper in Section 4 with a summary and directions for future work.

2 The Covert Channel

In this section, we describe the overall design of the covert channel.

2.1 An Overview

Consider a TCP connection between Alice and Bob where Bob is the data sender while Alice is the receiver. Alice has a covert message to send to Bob, and she is going to convey it through dropping some packets. In particular, Alice is going to drop packets that contain specific patterns in their hash values.

We assume that Alice and Bob agree a priori on a specific field, f, in the TCP packet header that is different in different packets, yet the same if a packet gets retransmitted (e.g., the sequence number field). Alice and Bob also agree on a cryptographic hashing function H (e.g., MD5 or SHA-1). Finally, they agree on a specific pattern to indicate a '0' and a specific pattern to indicate a '1'.[3]

Let H_n be a cryptographic hash function that would hash the field f into a hash value h that is n bits long.

$$h = H_n(f) \quad (1)$$

Alice and Bob need to agree on two patterns to differentiate between a '0' and a '1'. To indicate a '0', let $P[0]_i^l$ indicates a bit pattern (of 0's and 1's) that is of length l and starts at location i in the hash value h, where i can vary from 0 to $n-l$. Similarly, let $P[1]_j^l$ be another bit pattern that indicates a '1' and starts at location j in the hash value. For simplicity, we assume that both patterns have the same length l. We also prevent the overlapping between those two patterns. Notice that this is easy to achieve, since the length of the hash values, n, is typically much larger than the length of the pattern l. For example, with an MD5 hash, n is 128 and the patterns that would typically be chosen are around the length 6; hence, it is easy to prevent the overlap between the two patterns.

[3]As we will illustrate later, more than one pattern will be agreed upon in order to tune the performance of the covert channel.

Conveying the covert message: To convey a '0', Alice will hash the field f in the packets she is receiving and is going to examine the hash value for the pattern $P[0]_i^l$. If the pattern is found, Alice would drop this packet. Similarly, to convey a '1', Alice would do the same and examine the hash value for the pattern $P[1]_j^l$. If, however, both patterns are found in the hash of a single packet, Alice will *not* drop this packet. Also, if Alice wants to convey a '0', but receives a packet with its hash value containing the pattern $P[1]_j^l$, Alice will *not* drop this packet. The same case occurs if Alice wants to covey a '1', but receives a packet with its hash value containing the pattern $P[0]_i^l$.

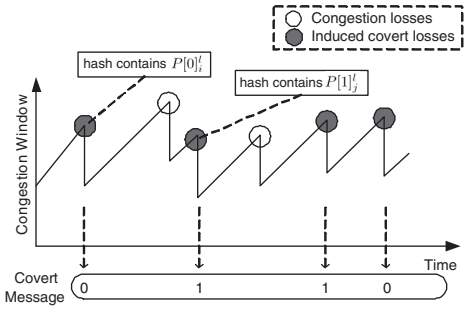

Figure 1. The performance of the TCP AIMD mechanism and recovering the covert message.

Recovering the covert message: For any packet that gets retransmitted, Bob would hash the field f and look for the two patterns in the hash value at their specified locations. If $P[0]_i^l$ was found, then Bob recovers a '0'. If $P[1]_j^l$ was found, then Bob recovers a '1'. If the hash contains both patterns, Bob knows that Alice did not drop this packet, and hence ignores this packet. This packet must have been dropped by the network. Figure 1 illustrates the recovering process.

Notice that the packets that get dropped by the network (due to congestion) may include the chosen patterns. This may result in recovering a longer message from the one intended. As we will illustrate next, the length of the patterns will be chosen carefully, to minimize such probability.

2.2 Choosing the lengths of the patterns

Given a hash value h according to equation (1), the probability that the pattern $P[0]_i^l$ exists is:

$$Prob[h\ contains\ P[0]_i^l] \quad = \quad \frac{1}{2^l} \qquad (2)$$

For example, by choosing $P[0]_0^3$ to be 101, the probability that hash value starts with 101 is $\frac{1}{8}$. Moreover, it would take 2^l packets, on average, before finding a packet that contains that particular pattern.[4] Notice that it would actually take 2^{l+1} packets before finding the *correct* packet to drop based on the covert bit Alice wants to send, assuming the covert message contains an equal number of 0's and 1's on average.

Notice that if l is chosen too small, we will be dropping a lot of packets. While dropping a lot of packets may seem to increase the throughput of the covert channel, it may hurt TCP and cause its throughput to drop, which would consequently reduce the chances of finding enough packets with the specific hash patterns. On the other hand, if l is chosen too large, we may wait for a long time before sending a covert bit. As we will illustrate in Section 3, a small range of values for l exists that would maximize the throughput of the covert channel.

We need to choose l such that 2^{l+1} is around the number of packets transmitted successfully, between packet losses for a potential connection that we would like to masquerade into. For example, if we would like to masquerade into a wireless channel that is dropping 5% of its packets. We have to choose $l+1$ equal to 4 or 5. In the case that we choose l equals to 3, we will drop a packet in every 16 packets, leading to a probability of 6.25%. With the case l equals to 4, we will drop a packet in every 32 packets, leading to a probability of 3.125%. Notice that both choices of l caused inaccuracy.

In order to drop more accurately, we can use a combination of *independent* patterns. Since the hash space is typically very wide, we divide it into a number of blocks (say 16-bit blocks). For an example, with an MD5, we can have 8 16-bit blocks. Now, to emulate a wireless channel that is dropping 5% of its packets, we can use independent patterns of lengths 4, 5 and 7 where each pattern occurs in a different block. This would lead to a probability of $\frac{1}{32} + \frac{1}{64} + \frac{1}{256}$, which is equal to 5.07%. Clearly, this choice is more accurate than just choosing one pattern.

In general, given a loss probability p that we would like to emulate, we would like to express it as a polynomial in the powers of 2:

$$p \quad = \quad \sum a_i 2^{-i} \qquad (3)$$

[4] This assumes that the hash values obtained are random, which is a reasonable assumption due to the use of MD5 and SHA-1 algorithms. We will also verify this in our experimental section.

where each coefficient, a_i, is either a '0' or a '1'.

For practicality concerns, we need to limit our choices of patterns to a small subset (less than the number of blocks we have). Also, we need to account for an allowable error, ϵ, due to representing a float value into a finite binary form. Notice that the problem of finding the lengths of the pattern is similar to the problem of representing a float in a binary system. So, given a p and ϵ, we find the powers of two that can represent the probability p with an error bounded by ϵ by dividing and subtracting powers of 2, until the remainder is within ϵ.

2.3 Impact of other packet drops on the covert channel

In our design of the covert channel, packet losses can be attributed to one of two events: congestion related losses or covert induced by Alice. Let E_c denotes the event that a packet was lost due to congestion. Let E_i denotes the event that a packet was forcibly dropped by Alice. The probability of E_i is p and let the probability of E_c be q. Thus the probability of Bob interpreting the correct lost packet as covert is:

$$Prob[Success] \quad = \quad 1 - p \times q \tag{4}$$

For an example, if Alice is inducing 5% losses and the channel is dropping 2% of the packets due to congestion. There is 0.1% chance that Bob would misinterpret a lost packet that was due to congestion as being induced by Alice. In this case, the hash of the dropped packet contained one of the patterns. The opposite cannot happen since a packet dropped by Alice *must* contain one of the patterns in its hash. Alice may drop other random packets (that do not have hash values containing the chosen patterns) in order to increase the stealthiness of this method as we explain below.

2.4 Discussion

We discuss a few important points regarding the design of this covert channel.

Impact of packet losses induced by Alice on TCP: In general, as a TCP connection experiences more packet losses, its throughput decreases. However, it has been shown that a *small* level of extra packet losses tend not to impact the throughput of the TCP connection. On the contrary, they can actually be beneficial in promoting fairness [9]. That is because when the network is congested, those small levels of losses act as an early congestion signal for the TCP connection to slow down, similar to the effect of a RED router queuing [6].

Stealthiness versus throughput: The covert channel discussed in this paper, while possibly achieving a low throughput, its level of stealthiness is high. The dynamics induced can still occur under normal conditions of operations, which makes the problem of identifying them as potential covert carriers harder. Since a TCP connection would typically traverse multiple links, each with possibly different characteristics/levels of congestion, it becomes challenging to observe packets on a given link and try to infer the behavior on other links (no global view of the network). Notice that to increase the throughput of the covert data while keeping the stealthiness level the same, multiple TCP connections can be used at the same time.

Masquerading into exact distributions of packet losses: In this paper, we have focused on the case where the covert channel is masquerading into another channel based on a specific average packet loss rate. To take this further, the covert channel could also masquerade into another channel not just with a specific average loss rate, but also with a specific variance. The exact parameters could be extracted from wireless traffic log files. Notice that the way packet losses are induced here follows a geometric distribution.[5] Alice could change this distribution by dropping some extra packets (that do not contain the patterns), just to change the distribution of packet losses (for example, to create bursts of packet losses). Including this in the covert channel will make detection much harder.

3 Experimental Evaluation

In this section, we report on our Internet experiments conducted to assess the performance of the covert channel discussed in this paper.

Sanity Checks of Randomness: The method described in this paper hinges on knowing when and how often the opportunity will arise to induce a packet loss. As described in Section 2, this information is based on the probability of finding specific bit patterns of particular lengths in the hash values obtained. The goal of our first experiment is to show that the use of the sequence number field in the TCP header does not bias the hashed values obtained (due to inherent dependencies between them). To that end, we used tcpdump to record the acknowledgment packets sent from one

[5]Since this is the same problem as the problem of coin flips. How many flips it would take until we get a head?

TCP end to the other in a connection that downloaded 560 MB of data.[6] The sequence numbers were hashed using MD5 and SHA-1, in two separate experiments.

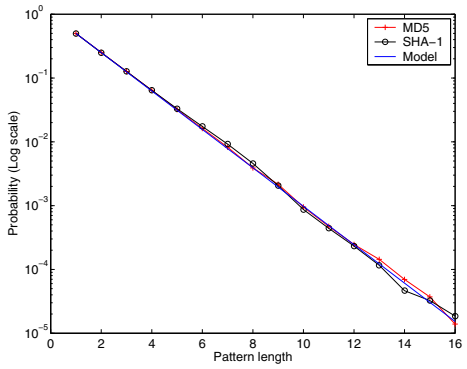

Figure 2. Probability of getting patterns of particular lengths using MD5 and SHA-1 in comparison to the idle value.

Figure 2 illustrates the results. On the x-axis, we vary the lengths of patterns we are looking for in the hashes from 1 to 16, starting from the first location. The y-axis shows the probability of finding those patterns. One can see that both methods have very close matching to the ideal model (complete randomness).

Another set of experiments were performed where we examined different bit fields at different locations in the hash values (to drop packets more accurately) to determine how random they look. The results also confirmed our expectations in being very close to the ideal case. Due to lack of space, we do not present those results here, but we refer the reader to [10].

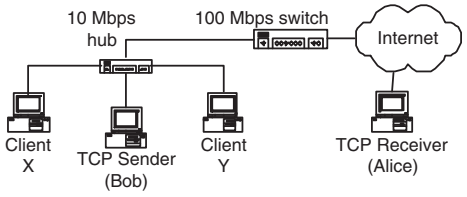

Figure 3. Experimental setup used in our implementation.

Experimental Setup: Figure 3 illustrates the experimental setup used in our implementation. It is composed of 3 machines (X, Y and Bob) connected to a 10 Mbps hub and another machine (Alice) that is connected to the Internet. Machines X and Y are Intel Core 2 Duo CPU running at 2.4 GHz with 4 GB of memory. Machine Bob is an Intel Core 2 Duo CPU running at 2.33 GHz with 4 GB of memory. Machine Alice is Intel Xeon CPU 3.80 GHz with 4 GB of memory. X, Y and Bob run Linux CentOS, version 2.6.18. Alice runs Linux 2.6.8. Alice and Bob communicate using ATOU [5], which is a TCP implementation over UDP. Machines X and Y send UDP traffic to each other to simulate different congestion levels.

Impact of congestion: To assess the performance of the covert channel under different levels of congestion, we varied the UDP traffic rate between machines X and Y from 0 till the maximum rate the network can handle (which was slightly less than the 10 Mbps physical constraint). The length of the pattern was chosen to drop 2.5% of the packets.

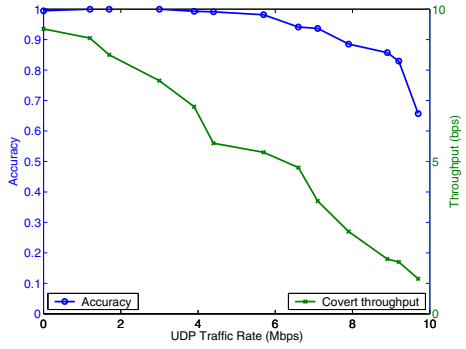

Figure 4. Accuracy and throughput of the covert channel over different levels of congestion.

Figure 4 shows the accuracy of the covert channel (left y-axis) and the covert channel throughput (right y-axis), at different levels of congestion. The accuracy is measured as the percentage of correct bits. The throughput of the covert channel declines as the UDP traffic rate is increased which is not surprising since TCP backs off due to congestion. Notice, however, that the accuracy of the channel remains somehow resilient against congestion (above 65% for the most congested network). This is primarily attributed to the inherent reliability in TCP.

Impact of loss probability: To illustrate the impact of the induced loss probability on the throughput of the covert channel, we ran a covert channel between Alice and Bob and we varied the loss probability induced by Alice (by varying the lengths of the chosen patterns). Each experiment ran for 30 seconds.

[6]Duplicate acknowledgment packets were excluded from the results because each sequence number is only examined once for a possible covert drop.

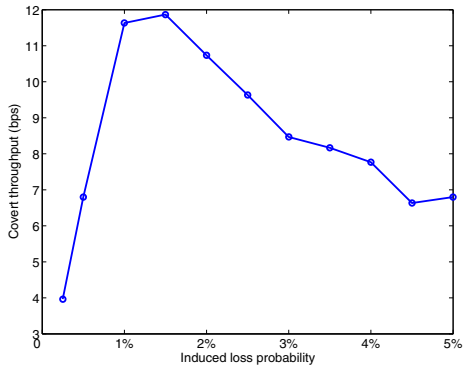

Figure 5. Throughput of the covert channel versus the degree of packet loss induced.

Figure 5 illustrates the throughput of the covert channel versus the percentage of packet loss induced. When the induced packet loss probability is very low, the throughput of the covert channel is low since a small number of packets gets dropped. Also, when the induced packet loss probability is high, the throughput of the covert channel is low, since TCP reacts to those losses by reducing its throughput, which affects the throughput of the covert channel. Notice the existence of a maximum value for the throughput when the induced loss probability is between 2% and 3%.

We refer the reader to [10], where we report on additional results and other simulation experiments.

4 Conclusion

We have presented a new method to implement covert channels that relies on inducing dynamics to convey a covert message. We have exemplified this approach with a TCP channel that can masquerade into a channel that appears to traverse a wireless or a congested link. The main strength of this approach is its resilience from being detectable since the dynamics induced may still occur under normal conditions. Such stealthiness comes at the price of a potentially low throughput for the covert channel. This makes our approach more suitable for short covert messages (e.g., transferring keys, passwords, etc..). For larger covert messages, multiple TCP connections may be used. Based on our Internet experiments, we found that low levels of induced packet losses tend to maximize the throughput of the covert channel while not forcing TCP's throughput to dip sharply. We are currently investigating the use of different error correcting codes to recover the covert message.

References

[1] K. Ahsan and D. Kundur. Practical data hiding in TCP/IP. In *Workshop on Multimedia Security at ACM Multimedia*, December 2002.

[2] V. Berk, A. Giani, and G. Cybenko. Detection of Covert Channel Encoding in Network Packet Delays. *Technical Report TR536, Department of Computer Science - Dartmouth College*, November 2005.

[3] S. Cabuk, C. Brodley, and C. Shields. IP Covert Timing Channels: Design and Detection. In *Proceedings of the 11th ACM conference on Computer and Communications Security (CCS'04)*. Washington, DC, October 2004.

[4] D. Chiu and R. Jain. Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks. *Computer Networks and ISDN Systems*, 17:1–14, 1989.

[5] T. Dunigan and F. Fowler. A TCP-over-UDP Test Harness. *Technical Report, Network Research Group, Oak Ridge National Laboratory*, May 2002.

[6] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *Transactions on Networking*, 1(4):397–413, August 1993.

[7] G. Fisk and M. Fisk T and C. Papadopoulos and J. Neil. Eliminating Steganography in Internet Traffic with Active Wardens. In *Proceedings of the fifth International Workshop on Information Hiding*, 2002.

[8] S. Gianvecchio and H. Wang. Detecting Covert Timing Channels: An Entropy-based Approach. In *Proceedings of the 14th ACM conference on Computer and Communications Security (CCS'07)*. Alexandria, Virginia, October 2007.

[9] M. Guirguis, A. Bestavros and I. Matta. Exogenous-Loss Aware Traffic Management in Overlay Networks: Toward Global Fairness. *The Computer Networks Journal (COMNET): The International Journal of Computer and Telecommunications Networking. Volume 50, Issue 13, Pages 2331-2348*, September 2006.

[10] M. Guirguis and J. Valdez. Masquerading a Wired Covert Channel into a Wireless-like Channel. *Technical Report, Comptuer Science Department, Texas State University*, December 2008.

[11] S. Murdoch and S. Lewis. Embedding Covert Channels into TCP/IP. In *Proceedings of the 7th International Workshop on Information Hiding*, 2005.

[12] P. LeBoutillier. HTTunnel. http://sourceforge.net/projects/httunnel/, 2005.

[13] T. Shon, J. Moon, S. Lee, D. Lee, and J. Lim. Covert Channel Detection in the ICMP Payload Using Support Vector Machine. In *Proceedings of ISCIS*, November 2003.

[14] B. Xu, J. Wang, and D. Peng. Practical Protocol Steganography: Hiding Data in IP Header. In *Proceedings of the First Asia International Conference on Modelling and Simulations*, 2007.