12-16-2013

A Dynamic, Broad Spectrum Approach to Covert Channels

Christopher Hoffman

A Dynamic, Broad Spectrum Approach to Covert Channels

Computer Security and Information Assurance

Christopher Hoffman
12/16/13

Masters of Science Computer Security and Information Assurance

From

The College of Computing and Information Science

At

Rochester Institute of Technology

Department of

Networking, Security and System Administration

Committee:

Daryl Johnson

Peter Lutz

Bo Yuan

Table of Contents

1 Abstract

In most covert channel systems, bandwidth is sacrificed for covertness. By using a dynamic, broad spectrum approach it was possible to use multiple covert channels to create a larger pipeline for data transmission. This approach utilized a monitor to determine how much data the network would be able to support before suspicion would be drawn to the change in traffic utility. The system then dispersed this traffic to each protocol proportionately using their monitored utility as a guide. A feedback channel was also utilized to determine if channel transmissions were successful and disabled any channels for future use that the network security blocked. In this way, a robust channel system was created that also increased bandwidth of communication.

2 Introduction

Covert channels were first proposed by Lampson (Lampson, 1973) to solve the Prisoners' Problem (Simmons, 1983). In the Prisoners' Problem, two parties wished to communicate with each other, however all communication travels through a third party, the warden. The prisoners wanted to communicate messages in a way that the warden would not be able to interpret. To do this, they must hide their secret communications within the overt messages the warden has the ability to see. This ability to hide the second level of communication is the purpose of covert channels.

Lampson stated that the purpose of a covert channel was to circumvent security policies. The most common way is to use resources in an unusual way. Examples of this are regulating processor utility or altering network packet header information in fields that are not checked.

Even though these are not designed to transfer information, they can be leveraged to carry out malicious activity. In this way it is possible to hide that communication is taking place.

2.1 Purpose of Covert Channels

Unlike cryptography, covert channels are used to hide the fact that any communication is taking place (Zander, Armitage, & Branch, 2007). Since covert channels attempt to conceal end points, it is harder to connect that the parties are communicating. In some cases, the fact that two parties are communicating with cryptography can raise suspicion and provoke further investigation (Newman, 2007). If cryptology is in use, it is showing that the parties want to hide what they are saying. An unencrypted channel would not draw as much attention since it would not appear that there is something to hide.

2.2 Bandwidth vs. Covertness

Covert channels have the inherent problem of sacrificing bandwidth for covertness (Cabuk, Brodley, & Shields, IP Covert Timing Channels: Design and Detection, 2004) (Cabuk, Brodley, & Shields, IP Covert Channel Detection, 2009) (Moskowitz, Chang, & Newman, 2002). For this reason, the two must be balanced. If one wanted to increase the bandwidth of a channel, they could send a big block of plain text, but this would not be covert. Conversely, the most covert a transmission could be is not sending any information at all which would be an unacceptable level of throughput.

2.3 Types of Covert Channels

There are three types of covert channels: storage, timing, and behavior. Storage channels utilize some form of medium where one party has permission to write and another party can read (Cabuk, Brodley, & Shields, IP Covert Channel Detection, 2009). This type of channel has the ability to be non-real time, the parties do not need to be actively connected for successful transmission. Eggers et. al. (Eggers & Mallett, 1988) also identifies that covert channels which utilize network protocols are a type of storage channel. This type of channel will be used in this paper.

Timing channels utilize patterns based on clocks (Cabuk, Brodley, & Shields, IP Covert Timing Channels: Design and Detection, 2004). These can be slightly more secure since they require active monitoring to detect the channel since patterns may be difficult to notice. They have the disadvantage of the two parties needing their clocks to be synchronized.

The last type of channel is based on behavior. In these, messages are sent by one party acting in a certain way and the action is interpreted based on a predetermined alphabet (Johnson, Yuan, & Lutz, 2009). These are harder to detect because actions need to be analyzed to determine if they fit the normal behavior of the party. If there is deviation, it then needs to be determined whether that deviation was intended or random change. The difficulty is selecting an appropriate alphabet to use to be able to maintain an acceptable bandwidth while maintaining normal behavior.

2.4 Proposed Idea

The dynamic, broad spectrum covert channel system incorporates many covert channels working together as one. The main property of covert channels is that an increase of bandwidth means a decrease of covertness. Depending on the desired utility of the channel, low bandwidth can be problematic. In traditional thought, covertness would be sacrificed to increase the bandwidth and vise versa, but this paper proposes a covert channel system that maintains appropriate covertness while increasing bandwidth. This will provide a higher capability as defined by Moskowitz, *et. al.*, that detection is taking into account with bandwidth when analyzing covert channels (Moskowitz, Chang, & Newman, 2002).

Much work has been done to create covert channels in many different protocols (Khan, Javed, Mirza, & Khayam, 2009) and to optimize their covertness and bandwidth. By using many of these channels together this covert channel system could be covert while maintaining a higher bandwidth.

One way of increasing bandwidth would be to push the covert channels more frequently. Even if a covert channel is used with an existing service, it could still be possible to over-saturate the network with channel traffic to the point of suspicion. A single protocol covert channel under control would not increase usage enough to be noticeable. But to increase the bandwidth by pushing it more would raise suspicion.

If multiple channels are used together, each on different protocols, it would be possible to increase the load of all the protocols together to keep the traffic proportions similar to the network fingerprint. This method would effectively hide the covet traffic in itself.

3 Related Work

"Towards Adaptive Covert Communication Systems" (Yarochkin, Dai, Lin, Huang, & Kuo, 2008) proposed a similar solution of monitoring and adapting to the specific network, however their approach focused on creating a robust channel. The channel would start by monitoring the network to determine the type of traffic being used. It then ranked the channels it was bundled with to determine what would be the optimal choice, the one that would be least likely detected on the monitored network. A feedback mechanism was in place to make sure the channel was successfully passing through to its endpoint. If the feedback showed that the channel was broken, it would select the next optimal channel. In this way, the channel remained robust under active wardens that might block some channels. The proposed channel performed similarly for the training and channel selection although it selected multiple channels to use together. This paper focused on increasing bandwidth where theirs was to increase robustness.

"Embedding a Covert Channel in Active Network Connections" also used multiple protocols, but in a different way. Unlike the previous channel, Khan *et. al.* used preexisting connections for its transfer instead of generating new traffic (Khan, Javed, Mirza, & Khayam, 2009). Most timing channels rely on the timed difference of arrivals of packets, this channel used the pattern of packets. This solved the problem of networks that fluctuate in speed over time. The two parts of the channel are the sender, which controlled the network connections of a computer system and the receiver, which monitored the incoming connections and detected the pattern. This improved typical timing channels by taking away the problem of unsynchronized timing. Since it used multiple protocols, normal timing detection techniques would be harder because not all the protocols needed to be used meaning a set theory is required to detect the channel. Similar to this channel, the proposed channel needed to synchronize the pattern, but the proposed system used storage mechanisms for message transfer as opposed to strictly the pattern. The pattern used in the proposed system related to how the message was broken down to get distributed across the channels.

"Capacity is the Wrong Paradigm" (Moskowitz, Chang, & Newman, 2002) stated that the metric so many rely on to measure covert channels (in their case steganographic channels) is capacity, but they propose that it should be capability. They differentiate covert and steganographic channels in that stegonagraphy has a finite transmission time and that they only exist when they are hidden. One of the factors in the capability measurement is capacity, but it also includes detectability and robustness of the channel. Normal communication has a capacity whose limit is the amount of information that can be transferred error-free. This is important in covert communication in general, it doesn't matter how much information a channel can carry if it is easily detected or breaks down with the smallest noise. That is why multiple vectors were used, each being covert with capability, but working together to generate a higher throughput for better capacity.

4 Dynamic, Broad Spectrum Approach

4.1 Problem it Solves

In a real world network, there will be a specific baseline of traffic usage that is unknown from the outside environment. When adding a covert channel, it is difficult to determine how much traffic can be added before it would become noticeable. For example, if there was a significant increase in traffic usage, such as DNS using 25% of traffic when the baseline is only 2.5%. It would also be suspicious if an unused protocol randomly appeared on a network that did not previously use that protocol.

There are multiple types of covert recognizers in practice, such as protocol analyzers to make sure that header information is legitimate since this is a common place to hide information in network traffic. Another recognizer could log traffic to see if there is a detectable pattern that could be a timing channel. This paper is based on a different detection mechanism that uses service utility to detect possible covert channels. This work is concerned that adding traffic will cause a spike in usage of a service which will signal possible malicious behavior in the network and provoke investigation.

The network monitor works off the theory that in a network, traffic proportions will stay largely similar. As total usage increases, the utility of each protocol increases proportionally to the baseline as opposed to only one or two protocols increasing in usage.

Two possible problems a covert channel could face in a network are over saturation and operation on an unused protocol. An outside observer does not know how much traffic is being used inside a network. They could access the network, configure a monitor, come back at a later point in time, and configure a covert channel to use what was present. This would require the attacker gain access to the network a few times to carry out all the operations and additional times if the channel does not work the first time. It would be much easier if a single application could carry out the operations and possibly safer since the attacker does not need to keep entering.

A system assisted by a monitor could help protect the covert channel system against both of these issues. By monitoring how much services get used in the network, it would be possible to calculate how much data can be added by the system before it became noticeable. A channel could provide a large bandwidth if it transmitted continuously as fast as it could but would be noticeable. It must be throttled to where its usage would not trigger the alarm. The monitor is the solution for this.

The monitor would also assist in knowing what protocols the network was using. If an attacker took a look at the network during the day and configured an HTTP channel to run continuously at an acceptable level from the small window observed, it could be possible once employees leave for the day that HTTP usage goes down and the channel would stand out. Conversely, if an attacker looks at night they might miss a wealth of HTTP traffic to hide in during the day.

The network landscape will continually be changing and the covert channel system must be able to handle this change or the chance of detection will increase.

The other problem faced by covert channels is that they are low bandwidth. Since this cannot be solved by pushing the channel more with one pipe as shown in Figure 1, this thesis proposes using a group of small pipes that together will increase bandwidth such as in Figure 2. Using more than one channel will help solve the bandwidth problem as well as the network usage problem. If multiple services raise their usage (packets/hour), it might be possible to protect their proportionate network usage (%). This would effectively hide the channel in itself while increasing its bandwidth.

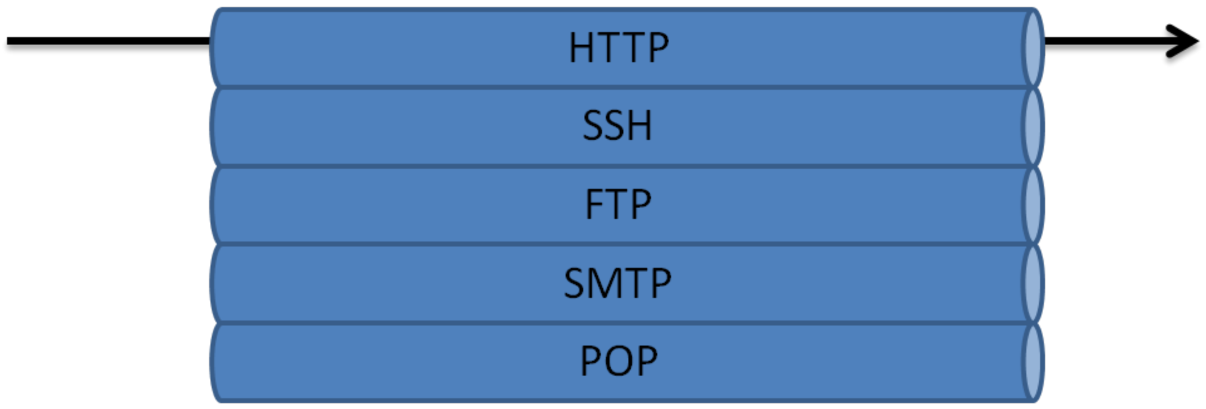

Figure 1: Single channel system

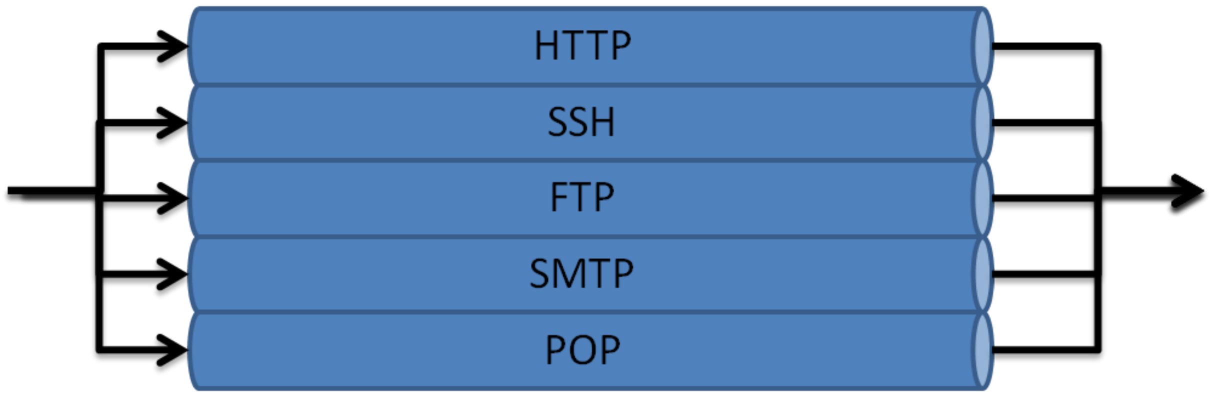

Figure 2: Multiple channel system

To note, this monitoring approach is only related to network traffic IDS as defined above. A host based IDS will be able to detect the increase of network traffic from the computer it is on and trigger alerts. The purpose of this experiment is to avoid detection by modeling a broad spectrum channel based on monitored network statistics.

4.2 Monitor Network

To increase the covertness of the communication, the system first monitored the network it was on to detect normal network usage and to pair the communication scheme with the network fingerprint. The system will only be successful once the capability of the environment has been discovered as stated by Cabuk *et. al.* (Cabuk, Brodley, & Shields, IP Covert Timing Channels:

Design and Detection, 2004). Each network will have a different fingerprint from IP ranges, servers, protocols used, topology, and most importantly security. A covert channel can be optimized to hide information deep inside a protocol, but it will be unsuccessful if its usage patterns do not match the network's, triggering a security alert to the presence of the channel. If a certain service does not get used or gets used rarely in the network, the covert channel on that protocol would stand out.

This information gets stored for later use deciding which covert channels to use and how much traffic can be added to each protocol before becoming significant. The intricate information that could be discovered inside the network made this covert channel system more dangerous than if an attacker only monitored from the outside (Bertino & Ghinita, 2011).

4.3 Channel and Bandwidth Selection

To determine the amount of traffic the system is able to add to the network, statistical analysis was used. Using the standard deviation of the network from the monitoring phase and a predetermined t-value, the t-test equation was used to calculate the greatest amount of traffic that can be added before it became significant.

Once the system knew how much total traffic was available and the proportions of that traffic, the channels can be selected. Since this is based off ingress and egress traffic, an appropriate channel should be chosen which matches the ingress to egress ratio of the added traffic. By matching these statistics the greatest amount of added traffic can be utilized.

4.4 Data Communication

To help with the covertness of the channel, the message was broken up between all the channels. If a single channel contains consecutive message pieces in a row, the discovery of the channel would have a readable message and might be interpreted as a communication. A readable message would raise suspicion where a broken message would look more random and therefore less suspicious.

A timed window was used to determine that the channel was working properly for checking but this is not a timing channel. The clocks do not need to be exactly the same as long as they are relatively synchronized. Also, there is no definite pattern that is followed, it will change based on the network usage during that time period. The chosen time window is an hour for each data segment.

4.5 Danger of Channel System

"Capacity of Steganographic Channels" (Harmsen & Pearlmean, 2005) discuss that a covert channel is only successful if it arrives at the destination undetected in an understandable format. This covert channel system faces problems meeting both of these requirements. By having more than one covert channel being run at the same time, the chance of discovery increases. There is also a danger of the control message encountering a problem in transit which could cause the covert message to be misinterpreted. If either of these things happened the channel would fail for the current time window and would have to restart during the next.

4.6 Channel Detection Mechanism

The network had baseline statistics to match against. When the covert system was run, it monitored the statistics. Once the time frame was over, it ran a statistical t-test on the data to determine if there was a significant increase in usage. Total traffic could also be checked, but a significant increase does not mean there is a problem, it could be due to adding or removing a user in the network. Since the amount of traffic will be added proportionately to each protocol in use, there would not be any spikes to alert a network administrator to a problem point.

If one of the protocols was found to be significant, either from being used more by the channel or lowing usage by not being used, the channel system can be discovered. If a single channel is discovered, network security could be changed to block it, leaving the rest of the system untouched. As more channels are discovered, the whole system has a higher chance of being discovered.

5 Structure of the System

There are three main parts to the covert channel system: an initial monitoring system, an internal controller, and an external controller.

For this demonstration, the channels themselves were not important. Since other covert channel recognizers were not in place, there would not be added value to have real channels as long as they transmitted at a similar capacity. The ‘channels’ transferred on a service protocol channel and sent only a few bytes at a time as a typical covert channel would behave.

5.1 Test Environment

The testing environment was made up of two networks in a virtual environment, 192.168.1.0 being the attacked network and 192.168.2.0 being the network data was being sent to. Each of these had a DNS server, mail server, ftp server, ssh server, two clients, and a router to connect to the Internet. Another router connected the two network routers in the 192.168.0.0 network and forwarded traffic between the two test networks. The network topology is shown in Figure 3.

Client machines were equipped with user scripts to simulate network traffic. This traffic was a necessity while testing since the system needed to monitor the current state of the network to decide how the system should perform. The traffic influences what channels and how much of each channel will get used. The traffic was also necessary while testing to make sure the system worked as expected in a real network and didn't trigger any alarms in the data transfer.

The user scripts were run at random intervals around a base to attempt to match the proportions found in the DARPA 98 dataset.

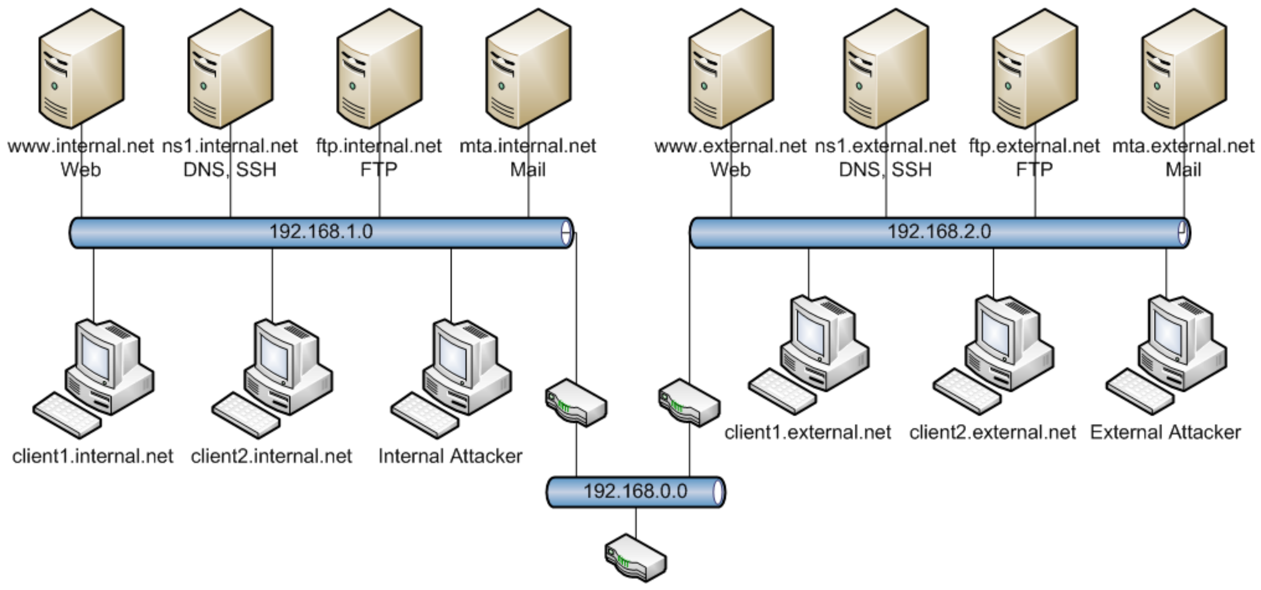

Figure 3: Network topology for test environment

5.2 Learning Engine

The monitor was designed to run passively for a week before data transfer to determine usage statistics. To determine usage statistics for the hour quicker, detection was made of four 15 minute segments. It would be safer to gather statistics over the course of a couple weeks, but there was a balance of safety vs data transfer.

The monitor used the tshark command line utility to capture traffic in 15 minute intervals. Tshark display options were used to pick out only necessary data fields which could get piped into the Java monitor program. The passed fields were source IP, source port, destination IP, destination port, and the IP protocol. This removed the need for Java to parse the network packet itself.

The Java program needed to have the network address and subnet mask to determine whether traffic was ingress or egress based on the addresses match to the given network information. The ports on the addresses were used to determine what protocol the traffic was. The data was kept in a Map until the end of the time frame at which point it was moved to storage.

The data was stored in a JSON data file for easy storage and recall between iterations of the system. At the end of the hour, the mean and standard deviations were calculated and also stored into the JSON file.

5.3 Channel Selection

For the purposes of testing, actual channels did not need to be chosen. The purpose of this process was the learning and scheme processing that would raise usage statistics uniformly. For this reason, specific channels were not chosen, each channel used was traffic on a specific service port which only sent a small amount of data at a time, similar to how a covert channel would behave. To maintain a standard between channels, each channel sent 1 byte per transaction.

The channels were created using simple Java network connections. Actual protocol behavior was observed using Wireshark protocol analyzer. The number of ingress and egress packets were counted for the TCP connection including the handshakes. The Java server/client programs were then generated to mimic the packet transfer behavior of real channel to send the same amount of packets per transaction while only sending 1 byte of data from the whole transaction (handshake to handshake). This was to maintain accuracy by simulating real world behavior.

After the week of network monitoring and statistical analysis, the channel system got called to run. The system ran in one hour windows, each time updating its schema based on the statistics from the monitor.

The first computation was to determine how much traffic can be added to the network before it becomes significant.

The selector started by determining how much traffic could be added to the network before it becomes significant. For the test, a t-value of 2.00 was used instead of 2.35 which would be the t-value with alpha 0.05 and df=4 (for each of the 15 minute segments). This is similar to Moskowitz and Miller (Moskowitz & Miller, 1992) protecting against a noisy network for timing channels. This way, if the network went through a cycle of high usage, it had less of a chance to trigger an alarm. The t-test was run with this predetermined t-value to determine the amount of traffic that would make the system significant.

From there, each of the channels received a portion of the added traffic. Using the mean and additional traffic values, it determined if each protocol was significant. If any protocol was significant, the additional traffic was decreased by 5% and the process was attempted again until acceptable values were found.

The system attempted to find a covert channel for each of the protocols present in the network as this would theoretically produce the largest throughput. Each of the stored channels had a set ingress to egress traffic ratio. This ratio was used to select channels to match closely to the available traffic based on the calculated ratio of ingress to egress. The channel with the closest proportion was used since this utilized the most traffic available.

Once the channels were selected, the network proportions were rechecked to make sure the network would still be able to theoretically support the additional traffic. If a problem was found, it decreased the amount of allowed traffic by 5% and re-picked the channels. This helped ensure the channel stayed within the safety range. The pseudocode for this can be found in Figure A-1, located in Appendix A.

5.4 Channel Information

The channels were stored as JSON objects, with the name of the channel being used as the key. The other information stored in the object was the channel ID, the protocol it operated on, the amount of data per transaction, the number of packets the channel uses in each direction, and whether the channel had been blocked from use.

The name of the channel was used to pick the appropriate class to use as the channel sender/receiver. When the controller was starting the channel, it called the class name to instantiate the channel.

The internal connection was the client (sender) and the external had the server (receiver). Although the system could open server connectors on the inside, most of the well known ports require super user privileges to listen on.

The controller used the amount of data to break the message up into pieces. The number of ingress and egress packets were used to help calculate the additional traffic being added to the network to test if a level of significance had been reached.

The blocked field was used to tell the system not to use a certain channel if a problem had been detected with it. This helped the system be more robust by not reusing anything that had been found unreliable or detected.

5.5 Opening Control Message

The external controller opened all of its channel receivers to listen for the control message since it did not know what channel would be used for control. The internal controller generated the control message which consisted of ordered pairs of channel id and number of transfers.

The schema needed to be transferred to the outside so it knew how to reconstruct the message. The transfer of the scheme outside the network took place on a single channel. The channel with the highest data/hour amount was chosen as the control channel. This feedback channel was also used at the end for the closing transactions.

Since this channel would have additional usage in the process, its allocated usage for the message was decreased to protect it from overstepping the significance level. There was a timeout in place in case the first channel chosen did not work, it could move onto another channel, blocking the first. This continued until a successful channel was found.

5.6 Data Transfer

Starting the data transfer section, the first thing that was done was breaking the message apart for the different channels. If a channel would get discovered and read and there is a readable message, it becomes much more suspicious than if the data appeared random. This method was better than keeping message segments whole and using encryption (Newman, 2007).

To break the data up, the channels were sorted in order by number of transactions from most to least. An ArrayList was used to hold the order the message was broken into. The List was populated by the largest channel first, filling in one index for every transfer with the channel ID. For the remainder of the channels in order, an ID was placed every other index until all the channels were used. Population of the List proceeded for each progressive channel starting where the previous left off. Message ordering pseudocode can be found in Figure A-2 located in Appendix A.

Each thread took part of the message as a List of bytes. The channel sender got called taking the bytes it was going to send in that transaction. The channel kept getting called until it had sent the entirety of its message.

A thread was created for each of the channels being used and set to run until there were 5 minutes left in the hour. The final 5 minutes were left for the control channel to communicate the closing. Transactions were spaced equally out across the sending time period to appear more natural. If there was a burst of data, there could have been raised suspicion. See Figure A-3 for pseudocode in Appendix A for internal controller process.

Each channel thread called a channel based on the name from its object. The name was the name of the channel sender class. The sender class uses an interface made of 5 methods: run(), addData(ArrayList<Byte>), getData(), close(), and isSuccess(). The last was for the control thread to check if any of the channels were having a problem, the rest is self explanatory. Pseudocode for channel control can be found in Figure A-4 located in Appendix A.

If at any point a channel dropped a message, the thread used a Boolean to notify the controller that a problem had occurred. At this point, the controller shut down all the channels to protect the system. If one channel failed, the traffic it was supposed to be adding to the network would not exist and could cause one of the other protocols to become significant. By having all the channels shut down, control was maintained by the system.

This does not negatively affect the performance of the system. Once one message piece was dropped, the external controller would drop the message in the end anyway. This way there was less traffic to be detected. If the channel was blocked and the rest continued on, it could cause imbalance in the network. The imbalance could cause the proportions to fall out of the baseline and bring more attention to the channels.

While the internal controller was getting the message split and threads created, the external controller was getting the receivers prepared. The channel receivers worked on a similar interface as the sending end except it didn't have an isSuccess() method. The receiving end was passive; it listened until the end of the time period so it didn't need to actively check on success. Figure A-5 in Appendix A shows pseudocode for the external controller process.

This time, only the channels requested in the control message were started. They passively waited, receiving traffic as it was transmitted. Once started, the external controller waited until the end of transfer period which was X:55 of the respective hour. When this time was reached, the controller gathered the data from the channels and closed them.

This framework could be a one-to-many configuration. During testing, all the traffic travelled to the central point outside the network, but it could travel to different addresses outside the network. This would provide some additional security to the channel since it would break up the connectivity of the channel. If a covert channel was discovered to a location, chances are that address would be monitored more closely. If different external addresses were used, it would make it harder to track other channels once one was discovered.

5.7 Closing Control Message

Once the sending time period had finished, the closing control message started.

Using the data received during the control thread, the channels were checked for completion. This was done by comparing the amount of data received with the number of transfers expected and the amount of data per transfer. If any channels did not send the appropriate amount of data, the system had failed and the offending channels were blocked from further use.

If all transfers were deemed successful, the external controller created the same message mapping that was used to break the message into pieces in the internal controller. This time it worked through the mapping List and takes from the appropriate channel in order.

The channel that control data was received on was restarted. This time the channel was set to send a message back based on the results of the channel system. If the channel was successful, the message was '0'. If the channel failed at any point, the message started with an '1' and had all the channels that failed appended to the end of the message.

The internal controller started the control thread used in the beginning, this time using a blank message. The interaction communicated to the internal controller how the channel performed.

If the first byte of the message was 0, the controller updated the index the message was at and saved the channel file with any blocking updates necessary. If the first byte was an 1, then the index remained the same and the indexes in the rest of the message were blocked from future use.

This was needed to ensure that this channel system remained robust. Sooner or later the system would become known and pieces would be secured. By logging the blocked channels, they will be avoided from future use.

5.8 Network Monitor

The last part to be developed was to assist in testing the system. The monitor utilized baseline statistics collected by the baseline monitor. These statistics were then stored in the detector data directory for use during monitoring.

To detect possible illicit activity, the monitor utilized a process similar to the one from the covert channel system, except it ran for a full hour. At the end of the hour, it tallied the traffic to determine the network usage statistics. It then preformed a t-test for each of the protocols and determined significance using the mean and standard deviation from the monitor and t=2.35 as opposed to the 2.00 used by the covert system. If there were any protocols that were significant, the system threw an alert. It would also issue an alert if there was a protocol that was present during the test that wasn't during the baseline.

6 Covert System Results

6.1 Testing Procedure

To test, the system was run in the test environment. The first step was to determine the baseline network traffic with just the user scripts running. For this, the covert system monitor was utilized to collect network data for a few hours and this was stored in the baseline data directory.

The covert channel system was then started to monitor the network for one hour. Once the hour was complete, it was started as though it were the next week and it was starting its data exfiltration phase. The system was run multiple times, decreasing the number of channels used each time to see how the system reacted to the change. The traffic it was tested on was kept the same as the baseline traffic.

6.2 Data Throughput

The testing of the covert channel system was successful. The monitor yielded the results shown in Table 1 and

Table 2. Tests started with the system running all channels. For each consecutive test, a channel was removed so one less protocol was in use to observe the change. The more channels in use, the higher throughput was observed as shown in Figure 4. Since the throughput is determined by the standard deviations of the channels, there would be a balance of increased t-values from usage being above and below the mean baseline proportions.

Table 1: Monitored ingress traffic

	Mean	Std. Dev.
SSH	31.10%	1.86%
SMTP	11.40%	0.81%
FTP	1.99%	0.27%
HTTP	43.83%	4.02%
POP	11.68%	1.45%
Total(Packets)	43351	5991

Table 2: Monitored egress traffic

	Mean	Std. Dev.
SSH	30.69 %	1.78%
SMTP	11.51%	0.86%
FTP	2.02%	0.28%
HTTP	44.04%	3.96%
POP	11.73%	1.50%
Total(Packets)	43250	5940

Based on Figure 4, the greater number of channels being used, the greater throughput is achieved. When five covert channels were in use, 938 bytes of data were sent over the course of the hour. Four channels only dropped throughput slightly to 934 bytes in the hour. The greatest change in throughput was observed when the number of channels dropped from three to two. When the system was down to one channel, 232 bytes were sent, a 4x decrease from when all present protocols had a working channel.

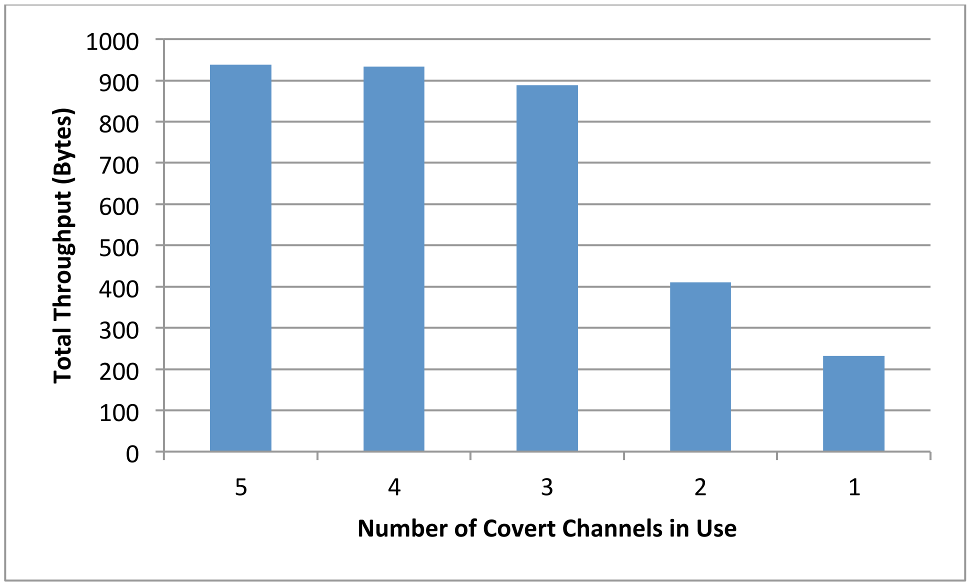

Figure 4: Total throughput from covert channels as channels are removed.

6.3 Channel Blocking

The system handled a blocked channel mid transport. Under this condition, the network stayed under the significance level. If one channel was blocked and the rest continued on for the rest of the time block it could cause a significance trigger. Since it dropped all the channels, the levels were able to stay under control.

6.4 Staying Under Significance Levels

The channel was determined to be successful if for all the channels a level of significance was not reached. When all the channels were used, the levels of significance stayed close to 0. As channels were removed, the levels increased. There was not a noticeable pattern of how the t-values fluctuated since the network usage was also in flux.

Below are the statistics gathered from the network while the system was running. Table 3 and Table 4 show the traffic that was collected during the system and what the baseline was. The amount of data for all the protocols were increased relative to its monitored proportion. This caused a much greater increase for some protocols.

Table 3: Ingress traffic of running system

	Baseline	System	Percent Increase
FTP	861	965	12.03%
HTTP	19000	21739	14.42%
POP	5065	5947	17.41%
SMTP	4941	5799	17.38%
SSH	13484	14667	8.78%
Total	43351	49117	13.30%

Table 4: Egress traffic of running system

	Baseline	System	Percent Increase
FTP	874	994	13.79%
HTTP	19049	21469	12.71%
POP	5075	5961	17.47%
SMTP	4979	5681	14.10%
SSH	13274	15012	13.09%
Total	43250	49117	13.57%

The proportions of traffic are displayed below in Figure 5 and Figure 6. As it can be seen, the proportions of network traffic are relatively similar even though the total traffic increased by

5766 packets for ingress and 5867 packets for egress. Since all the protocols increased together, the additional traffic helped mask the existence of the covert channel system.

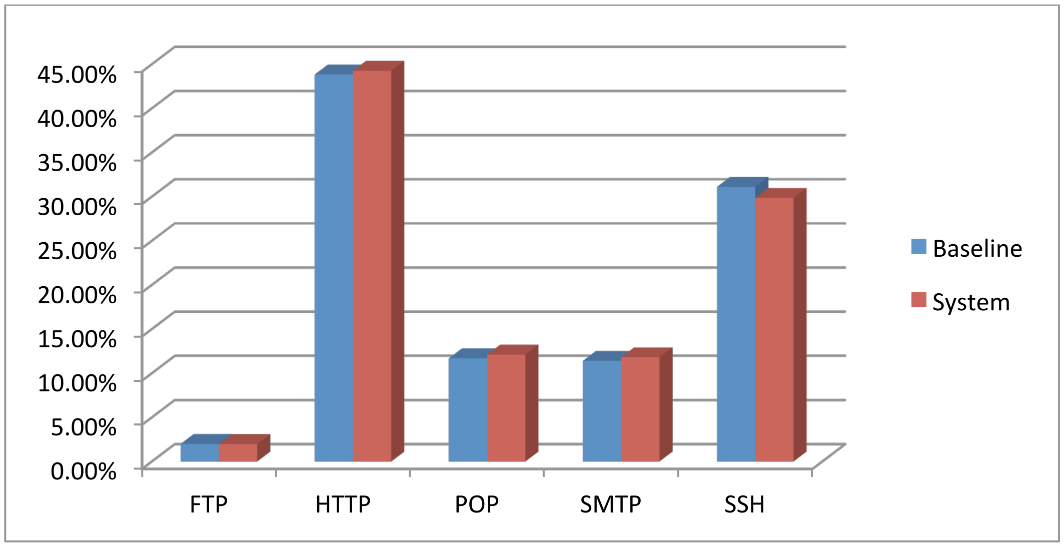

Figure 5: Comparison of protocol proportions of ingress traffic

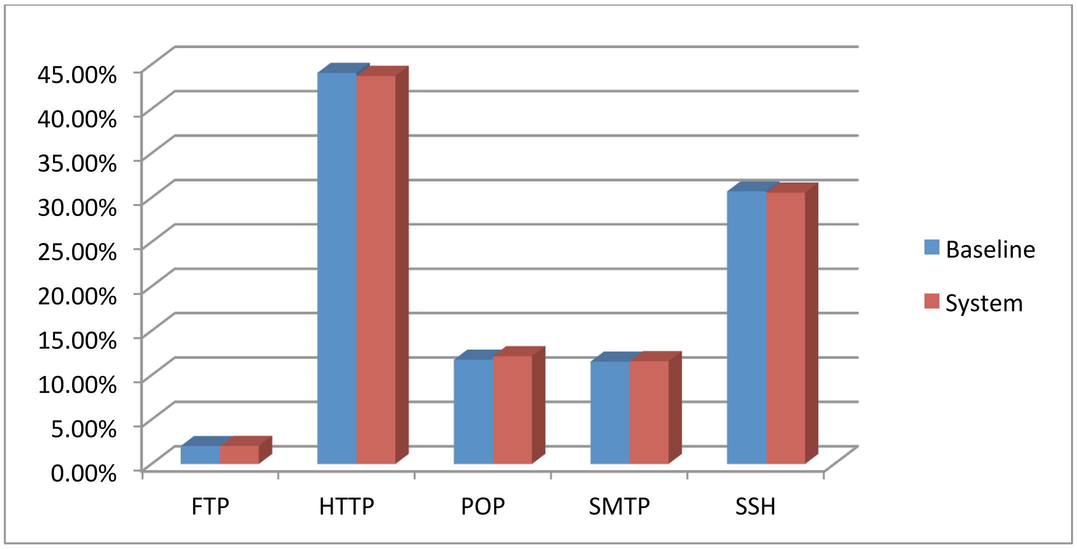

Figure 6: Comparison of protocol proportions of egress traffic

As can be seen above, proportions remained largely similar between the baseline and the system executing. There is some variability due to the network being variable and the full amount of available ingress/egress traffic will not get utilized. Individual increases are significant, but when viewed in the frame of all network traffic increasing, increases are small and insignificant. For example, ingress HTTP traffic increased from 19,000 packets per hour to 21,739 packets per hour. Even though this is a 2,739 packet increase, which by itself is a 14% increase, only increased its utility by 1.46% in the total traffic proportions. All channels had similar behavior of large individual increases, but insignificant increases when viewed in the whole system.

6.5 Changes in Throughput

The affect of standard deviation was tested by artificially increasing the standard deviation of each protocol and observing the behavior. As the standard deviation increases, the available throughput increased as well. This was due to a higher amount of change available before the level of significance was reached. As shown in Figure 7 and Figure 8 there was a positive linear correlation of channel throughput.

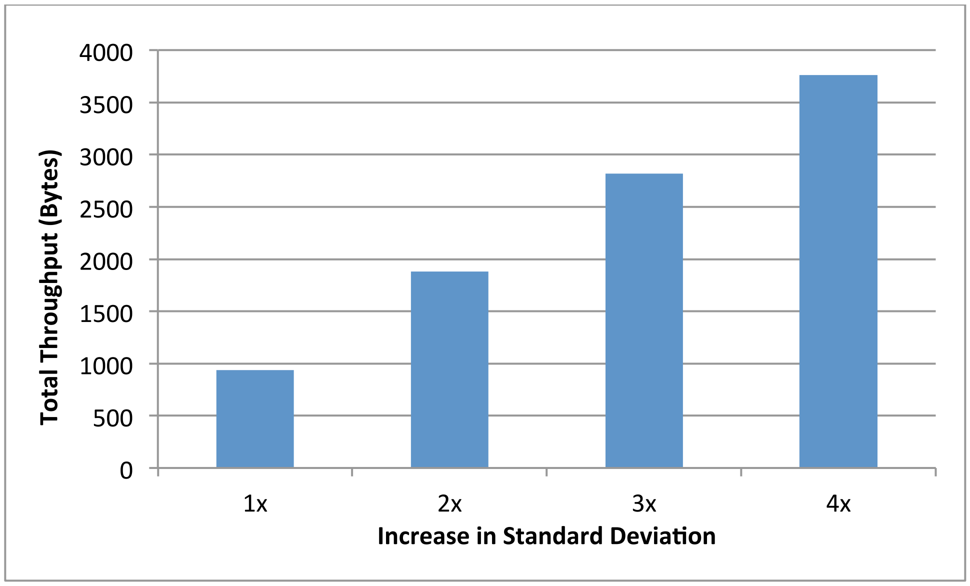

Figure 7: Total throughput from covert channels as standard deviation increases.

Figure 7 shows the effect of the change of standard deviation on the total system while Figure 8 shows the effect of the change of standard deviation on the individual channels. As it can be seen, there is a direct relation between standard deviation and throughput. For HTTP traffic, the base standard deviation allowed for 517 bytes. An increase to 2x standard deviation yielded a 1035 byte throughput, 3x was 1553 bytes, and 4x was 2071 bytes. Based on this evidence, the greatest throughput will be achieved when there is greater standard deviation and is not affected by the mean amount of traffic.

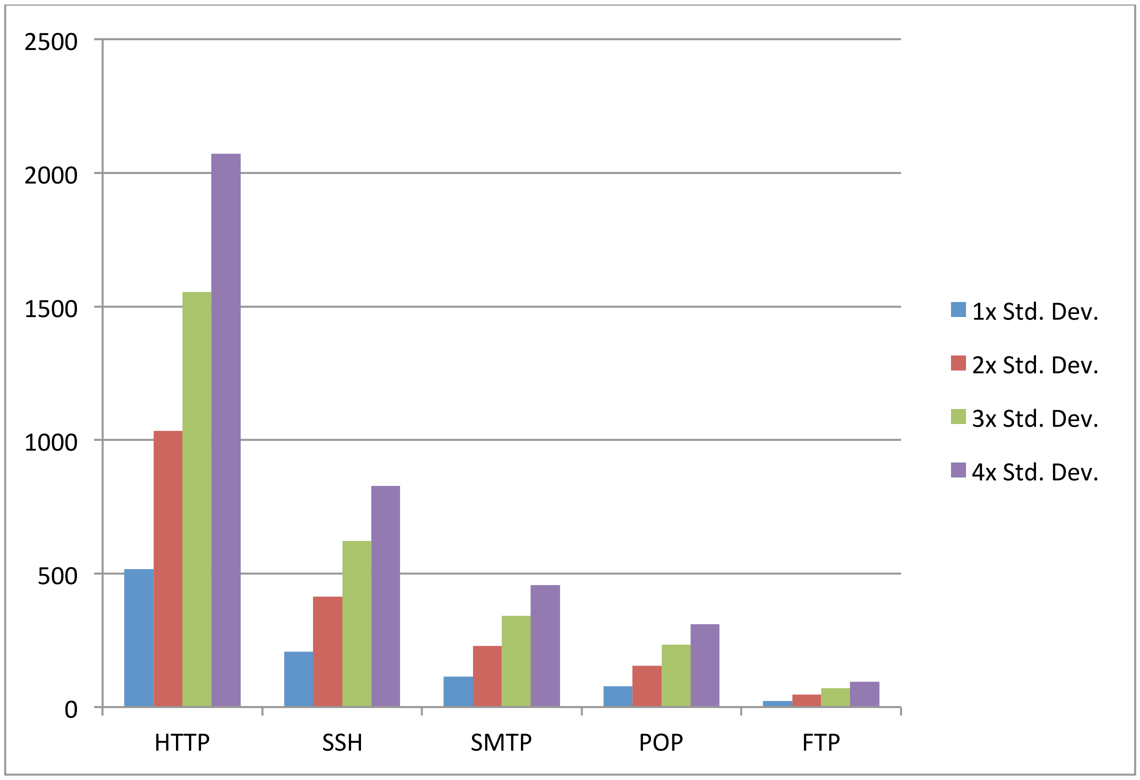

Figure 8: Traffic per protocol as standard deviation is increased

7 Channel Analysis

7.1 Throughput

Throughput of the system was variable based on the collected network statistics. The more variable the baseline of the network was, the higher the throughput of the total system; this is due to the t-test calculation being based off standard deviation.

Having more channels will also increase the throughput of the covert channel system. The greatest throughput will occur when there is a covert channel for each protocol present in the network. If there is a protocol that does not have a covert channel, it will restrict the amount of traffic that can be added.

7.2 Robustness

Robustness mechanisms are built into the system to keep it running. If a channel is blocked during one iteration (hour), it will not be used later. For that hour, the data will be discarded because it is unknown where the error took place so the data is unreliable. The channel system would also stop transmitting for the remainder of the timed window to help insure the traffic remains within acceptable levels. Each hour will be able to start new, taking into account any previous issues.

As discussed by Jadhav and Kattimani (Jadhav & Kattimani, 2011), hybrid channels are more robust as they are harder to detect. Even though there are more pieces to detect, this covert channel system had the ability to recover from error and it would not mean an end to the system.

7.3 Detection

Detection of the system is directly related to the covertness of the channels the system is using. Since the broad spectrum approach would use channels that are already optimized for covertness, this system should be as secure as the covert channels used.

If a channel is found, the end points will be known since the system will use generated traffic as opposed to using hijacking traffic for message transfer. This will break the channel on the internal side since the host will be cleaned. The owner of the external system will be able to remain more anonymous by running the controller in a public cloud with false credentials.

7.4 Prevention

Preventing the covert channel system would be dependent on the prevention mechanisms of the individual channels. Millen (Millen, 1999) discussed that the best prevention mechanism would be to rebuild the affected system with the covert channel in mind. In this case, prevention would be very difficult as this covert channel system affects many systems. However, it would be possible to develop a detection mechanism to determine if a more in depth analysis is needed to find a covert channel. This fits with Zander *et. al.* (Zander, Armitage, & Branch, 2007) that says if a covert channel cannot be removed, knowledge of its existence should still be audited to track its behavior.

A possible detection mechanism could be to add artificial traffic on the network, this would involve adding a finite amount of traffic to the network using a protocol that is not naturally present. If the amount of that protocol traffic deviates from the standard, it can be determined that malicious behavior is taking place.

8 Future Work

At the moment, the covert channel system is only for data exfiltration. It can be used for data infiltration as well at which point it could be a command/control channel for an attack pivot. This would create a powerful tool to pivot inside a network.

Another addition for future work is to pair the monitor with the sending system more closely. This way, when the system is transmitting, the monitor can determine if the base usage statistics have changed from week to week. This will create a more robust system that will require less attention over time.

A process to add channels over time would also be a good addition. This way, as the attacker produces more covert channels they can be loaded dynamically instead of needing to replace the system.

Data transmission can also be dispersed between internal hosts. This could be done either by giving a different covert channel to multiple clients, or by breaking up the traffic for each channel to multiple clients in an attempt to not trigger a host based IDS. The data can also be sent to multiple hosts outside to not stand out in a connection tracer.

9 Conclusion

This paper has shown that by using a dynamic, broad spectrum approach, it is possible to increase the bandwidth of covert communication while maintaining acceptable covertness by running multiple covert channels side by side. The most bandwidth was observed when there were larger standard deviations and more channels in use. By monitoring the network before use, the system was able to make better decisions on the channels and amount of data to send.

10 References

Bailey, D. V., Boneh, D., Goh, E.-J., & Juels, A. (2007). Covert Channels in Privacy-Preserving Identification Systems. In *Proceedings of the 14th ACM conference on Computer and communications security* (pp. 297-306). ACM.

Bertino, E., & Ghinita, G. (2011). Towards Mechanisms for Detection and Prevention of Data Exfiltration by Insiders: keynote talk paper. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security* (pp. 10-19). ACM.

Cabuk, S., Brodley, C. E., & Shields, C. (2009). IP Covert Channel Detection. *ACM Trans. Inf. Syst. Secur. , 12* (4), 22:1-22:29.

Cabuk, S., Brodley, C. E., & Shields, C. (2004). IP Covert Timing Channels: Design and Detection. In *Proceedings of the 11th ACM Conference on Computer and Communications Security* (pp. 178-187). New York City, NY: ACM.

Eggers, K. W., & Mallett, P. W. (1988). Characterizing Network Covert Storage Channels. In *Aerospace Computer Security Applications Conference* (pp. 275-279).

Harmsen, J. J., & Pearlmean, W. A. (2005). Capacity of Steganographic Channels. In *Proceedings of the 7th Workshop on Multimedia and Security* (pp. 11-24). ACM.

Jadhav, M. V., & Kattimani, S. L. (2011). Effective Detection Mechanism for TCP Based Hybrid Covert Channels in Secure Communication. In *Emerging Trends in Electrical and Computer Technology (ICETECT), 2011 International Conference on* (pp. 1123-1128).

Johnson, D., Yuan, B., & Lutz, P. (2009). Behavior-Based Covert Channel in Cyberspace. *Intelligent Systems and Knowledge Engineering* .

Khan, H., Javed, Y., Mirza, F., & Khayam, S. A. (2009). Embedding a Covert Channel in Active Network Connections. In *Global Telecommunications Conference, 2009. GLOBECOM 2009. IEEE* (pp. 1-6). IEEE.

Lampson, B. W. (1973 йил Oct). A Note on the Confinement Problem. *Communications of the ACM* , pp. 613-615.

Millen, J. (1999). 20 years of covert channel modeling and analysis. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on* (pp. 113-114). IEEE.

Moskowitz, I. S., & Miller, A. R. (1992). The Channel Capacity of a Certain Noisy Timing Channel. *Information Theory, IEEE Transactions on , 38* (4), 1339-1344.

Moskowitz, I. S., Chang, L., & Newman, R. E. (2002). Capacity is the Wrong Paradigm. In *Proceedings of the 2002 Workshop on New Security Paradigms* (pp. 114-126). New York City, NY: ACM.

Newman, R. C. (2007). Covert Computer and Network Communications. In *Proceedings of the 4th Annual Conference on Information Security Curriculum Development* (pp. 12:1-12:8). New York City, NY: ACM.

Simmons, G. J. (1983). The Prisons Problem and Subliminal Channel. In *Proc. Advances in Cryptology* (pp. 51-67). Springer-Verlag.

Yarochkin, F. V., Dai, S.-Y., Lin, C.-H., Huang, Y., & Kuo, S.-Y. (2008). Towards Adaptive Covert Communications Ssytem. In *Dependable Computing, 2008. PRDC '08. 14th IEEE Pacific Rim International Symposium on* (pp. 153-159).

Zander, S., Armitage, G., & Branch, P. (2007). A Survey of Covert Channels and Countermeasures in Computer Network Protocols. *Communications Surveys Tutorials, IEEE , 9* (3), 44-57.

A. Appendix

```
Channel Selector
for each ingress[egress]
  reverse t-test to find amount of traffic that can be added
  ingress[egress]AdditionalTraffic
  decrease additional by 10% to help protect the system

Label A:
  for each protocol P from the monitor
    ratio=ingressAdditionalTraffic*P.ingressProportion/
P.egressAdditionalTraffic*egressProportion
    select channel from preloaded bank that has closest ratio

  for each channel
    test ingress[egress] with t-test for significance
    if significant, decrease additional traffic by 5% and restarted from Label A
```

Figure A-2: Pseudocode for channel selection process

```
Message Ordering
sort channels based number of transactions
for channel[0].transactions
  add channel ID to messageOrdering
for each remaining channel c
  index=0
  for i=0 ; i<c.transactions
    insert ID at index of messageOrdering
    index= index+2 mod messageOrdering.length
```

Figure A-1: Pseudocode for message ordering

```
Internal Controller Processing
Label A:
sort channels by amount of data to send
largest gets selected as feedback thread

if the feedback fails
  block channel restart at Label A

initialize channel threads

wait until x:05

start channel threads

sleep until x:55
  check for problems
  if problem occurs
    close all channels

ping the external controller to check for transfer problems
if return good
  update message index
if return bad
  block channels that had problems
```

Figure A-3: Pseudocode for internal controller

```
Channel Controller
receive message to send from controller
create channel based on the channel name as the class
on start:
  sleepTime = (time until x:55) / (number of transaction)
  for each transaction
    extract the number of bytes the channel sends from its message
    run the channel
    sleep for sleepTime

optional:
 check for success

on error:
  close the socket
```

Figure A-4: Pseudocode for channel controller

```
External Controller Processing
start all available channels
listen for channel information to be sent

initialize channel threads using the information sent
start channels threads

listen until x:55 for all the data to be sent

collect information from the threads and close the channels

if the appropriate amount of traffic has been sent
  reassemble the message
else
  create message for the channels to send back
```

Figure A-5: Pseudocode for external controller