IP Covert Channel Detection

SERDAR CABUK
Hewlett-Packard Laboratories
CARLA E. BRODLEY
Tufts University
and
CLAY SHIELDS
Georgetown University

A covert channel can occur when an attacker finds and exploits a shared resource that is not designed to be a communication mechanism. A network covert channel operates by altering the timing of otherwise legitimate network traffic so that the arrival times of packets encode confidential data that an attacker wants to exfiltrate from a secure area from which she has no other means of communication. In this article, we present the first public implementation of an IP covert channel, discuss the subtle issues that arose in its design, and present a discussion on its efficacy. We then show that an IP covert channel can be differentiated from legitimate channels and present new detection measures that provide detection rates over 95%. We next take the simple step an attacker would of adding noise to the channel to attempt to conceal the covert communication. For these noisy IP covert timing channels, we show that our online detection measures can fail to identify the covert channel for noise levels higher than 10%. We then provide effective offline search mechanisms that identify the noisy channels.

Categories and Subject Descriptors: C.2.0 [**Computer-Communication Networks**]: General—*Security and Protection (e.g., firewalls)*; H.1.1 [**Models and Principles**]: Systems and Information Theory—*Information Theory*

General Terms: Security

Additional Key Words and Phrases: Network covert channels, information hiding, channel detection

ACM Reference Format:
Cabuk, S., Brodley, C. E., and Shields, C. 2009. IP covert channel detection. ACM Trans. Inf. Syst. Secur. 12, 4, Article 22 (April 2009), 29 pages. DOI = 10.1145/1513601.1513604.
http://doi.acm.org/10.1145/1513601.1513604.

Author's address: C. Shields, Georgetown University; email: clay@cs.georgetown.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from the Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2009 ACM 1094-9224/2009/04-ART22 $5.00 DOI: 10.1145/1513601.1513604.
http://doi.acm.org/10.1145/1513601.1513604.

1. INTRODUCTION

Consider a computer system in which an attacker, Eve, manages to insert malicious code inside a secure area or network belonging to Alice. Eve would like to be able to exfiltrate confidential information from the secure area to a lower-security area. She faces potential challenges in meeting this goal. There might not be any means of direct communication between the high and low security area. Furthermore, she might be concerned that the confidentiality-violating information that she is sending will be detected and her code located and neutralized. In these cases, she might look for alternate means of communication that allow information to leak from the trusted, higher-security area to the untrusted, lower-security area. One alternative is to find and subvert some resource that is shared between the high and low security areas but which is not usually used for communication.

A *covert channel* is a communication channel that violates a security policy by using shared resources in ways for which they were not initially designed. In this article we address network covert channels in which the timing of packet arrivals encodes the message; an arrival within a certain interval indicates a high bit, and the absence of an arrival a low bit. To utilize this channel, Eve must be able to modify the transmission times of packets departing from the high-security area along a path that crosses a low security area that she can observe. For example, if Alice communicated with Bob over a virtual private network (VPN) that crossed a public network, Eve could alter the timing of packets as they were transmitted, then monitor the timing of the packets as they crossed the public network. She would then be able to receive data from the covert channel even if the packets and headers were encrypted. Though we use the specific example of a VPN as a motivator in the article, there are a wide variety of situations where the leakage of confidential information is a risk. These include not only systems or networks that implement some security policy, but any networked system in which confidential information is available at one point along the transmission path, but not at another, including anonymity networks in which the first hop along an anonymized path might know and leak the initiators identity.

In this article, we present the first public implementation of a true high-bandwidth IP covert channel. While such channels have been discussed in a theoretical manner in the past, we show that there are several subtle issues in implementing such a channel that are not present in single-system covert channels, particularly in dealing with network packet delay and loss. Because there have been no prior public implementations of such a channel, ours is also the first work to discuss their detection based on real network traffic, and we introduce two regularity-based measures that use time-series analysis techniques for online and offline detection. We extend our initial covert channel to include noisy channels, in which an attacker attempts to mask the regularity of the covert channel by introducing noise in the form of added legitimate traffic. These would be the natural next steps an attacker would take to defeat our initial approaches, and we present the first analysis and detection of these types of channels. We evaluate the efficacy of our detection schemes using standard machine learning experiments.

Overall, we show that against a defender using our scheme, an attacker must significantly reduce the rate of transmission over the covert channel to avoid detection, either by making intervals long or by adding significant noise to the channel. This article represents a significant extension to our prior work [Cabuk et al. 2004].

The remainder of this article is organized as follows: In Section 2, we provide background and related work on network covert channels and covert channel detection. In Section 3, we present our design of an IP covert channel that we implement both as a storage and a timing channel and discuss the factors that affect its efficacy. We introduce our measures for channel detection in Section 4 and provide an empirical evaluation for both noiseless and noisy IP covert channels. In Section 5, we focus on noisy IP covert channels and present methods to locate covert channels hidden inside noisy data sets locally. We discuss the limitations of our approach and areas of future work in Section 6, then present our concluding remarks in Section 7.

2. BACKGROUND

Covert channels can be characterized in a variety of different ways: depending on the type of the shared resource used (storage or timing), the noise level (noiseless or noisy), and the type of the system in which they are used (single or network). Traditionally, a *storage covert channel* involves a storage location to which the covert channel sender writes and from which the receiver reads, usually indirectly [Kemmerer 1983; Department of Defense 1985]. A *timing covert channel* is established when the sender can modulate the receiver's observed response time in a way that can provide information [Kemmerer 1983; Department of Defense 1985]. It has been shown that these definitions are ambiguous [Millen 1999; Wray 1991]. A nonambiguous definition has been devised that defines a storage covert channel as a communication channel "where the output alphabet consists of different responses all taking the same time to be transmitted," and a timing covert channel as a channel "where the output alphabet is made up of different time values corresponding to the same response" [Moskowitz and Kang 1994]. Please note that in this article we comply with this definition; thus, we might label some channels as storage channels that the reader might expect to be called timing channels.[1] Classic covert channels include:

(1) A storage *file-lock covert channel* exploits systems with shared file descriptors by observing a read-only file's locked/unlocked status in fixed timing intervals.

[1] Although in our earlier work [Cabuk et al. 2004] we defined our channel as a timing channel (in order to differentiate it from covert shells that use packet headers), it really is a storage channel according to Moskowitz's definition.

(2) A storage *disk-arm covert channel* [Schaefer et al. 1977; Karger and Wray 1991] exploits KVM/370 systems with a security kernel and a shared read-only disk drive by observing the position of the disk arm in fixed timing intervals.
(3) A timing *bus-contention covert channel* [Hu 1992] exploits multiprocessor systems running on a single shared bus by observing the busy status of the bus in multiple timing intervals.

A *hybrid channel* (also known as a mix channel) uses multiple time values with multiple responses. A covert communication channel does not assume a direct channel between the sender and the receiver, which separates it from related constructs such as the *subliminal channel* described in Simmons' prisoner's problem [Simmons 1984] and which is most commonly related to steganography. In these subliminal channels, there is legitimate communication that conceals data within the message itself.

If a shared resource is not solely used by the covert channel parties but also by other legitimate users, the resulting channel is a noisy communication channel. In particular, a *noisy covert channel* is a communication channel in which both legitimate and covert events are observed [NCSC 1993]. Note that the channel noise described here is not *signal noise*, but is *contention noise* caused by the contention for the shared resources [Moskowitz 1991]. In contrast, in a *noiseless covert channel*, the shared resource is used solely by the parties involved in the covert communication. That is, the events observed by Eve on the receiving end are the same as the events generated by Alice. In this article we investigate both noiseless and noisy channels when we analyze the efficacy of our detection measures.

2.1 Network Covert Channels

A *network covert channel* is a covert channel in which the shared medium is the network environment (i.e., transmission lines, firewalls, routers, etc.). Although initial research in covert channels focused on single systems, our focus here is on network covert channels which have become a threat to trusted distributed systems by allowing undetected leaks of confidential data or information. Network covert shells have been used by attackers to communicate with compromised hosts, particularly in distributed denial of service attacks [Henry 2000]. Many tools exist for setting up these shells using a variety of protocols including TCP, IP, HTTP, ICMP, AODV, and MAC [Abad 2001; Ahsan 2000; Ahsan and Kundur 2002; Bauer 2003; Castro 2003; Daemon9 1996, 1997; Giffin et al. 2002; Girling 1987; Handel and Sandford 1996; Hauser 1999; Li and Ephremides 2004; Murdoch and Lewis 2005; Rowland 1997; Rutkowska 2004; Smith 2000]. Common practice is to encapsulate information in a legitimate protocol to bypass firewalls and content filters.

The data section of packets is the easiest place to convey covert information, due to its large size and because it is relatively unstructured compared to headers. Modifying the packet payload is outside the scope of this article as it falls in the realm of steganography or watermarking, and as such is a subliminal rather than covert channel. Unused header fields that are either designed for future protocol improvements or in general go unchecked by firewalls and network intrusion detection devices may convey information in the form of a covert channel. Many covert shells have been illustrated in TCP and IP protocols using packet header fields [Abad 2001; Ahsan 2000; Ahsan and Kundur 2002; Bauer 2003; Castro 2003; Daemon9 1996, 1997; Giffin et al. 2002; Hauser 1999; Rowland 1997; Rutkowska 2004; Smith 2000]. Rowland illustrates three examples of TCP/IP covert shells [Rowland 1997]: the first encodes the messages into the identification field (ID) of an IP header, the second embeds the messages into the initial sequence number (ISN) field of a TCP header, and the third uses the TCP ACK sequence number field. An improved version of Rowland's TCP ISN covert shell, Nushu, is introduced in Rutkowska [2004]. The main difference between Rowland's TCP ISN shell and Nushu is that the latter encrypts the TCP ISN field to make it look like a random number in order to better obscure the channel. A smart attacker can even devise means to use some of the header fields that do fall under scrutiny, such as the IP checksum field. Abad illustrates that the sender can send any message using hash collision in the checksum field [Abad 2001]. Sorting covert channels have also been devised for the TCP/IP protocol, in which it is the ordering of the packets that transmits the covert information to the receiver side [Ahsan 2000; Ahsan and Kundur 2002]. However, for this channel to work, the receiver needs to know the correct ordering of the packets. Therefore, these channels require the use of the IPSEC protocol, which provides the original order in which the packets were sent. The authors in Murdoch and Lewis [2005] argue that the previous efforts to hide covert channels in packet header fields alter the behavior of these fields. They introduce Lathra, which is a covert shell that uses the TCP ISN field while trying to mirror the ISN generation process as closely as possible to avoid detection.

TCP and IP are not the only protocols used for covert shell exploitation. *Reverse WWW Shell* is a covert shell that uses the HTTP request parameters to leak information [Hauser 1999]. In particular, the sender issues an HTTP `get` request with the specified parameters, bypasses the HTTP-only firewall, and leaks the information to the receiver side. Another HTTP covert shell is illustrated for privacy purposes [Bauer 2003]. In this channel, the sender wants to remain unlinkable, hence she redirects the communication via an unwitting client's browser. The author shows that this is possible using redirects, cookies, active content, etc. ICMP packets have also been used to host covert shells. The Loki covert shell runs on ICMP and uses the data portions of the `icmp_echo` packet to send a command to a remote receiver [Daemon9 1996, 1997]. The receiver back door then executes this command on the victim computer, and returns the result embedded in the data portion of the `icmp_echoreply` packet. A list of similar covert shells developed by the black hat community can be found in Smith [2000].

Other interesting examples of covert shells are illustrated in the OSI protocol stack [Handel and Sandford 1996], in LAN networks [Girling 1987; Dogu and Ephremides 2000], and in ad-hoc wireless networks [Li and Ephremides 2004]. In LAN networks, the destination ID, packet length, time between successive transmissions [Handel and Sandford 1996], and the number of collisions in a collusion resolution period (in MAC protocol) [Dogu and Ephremides 2000] were used. In ad-hoc networks, [Li and Ephremides 2004] identify four covert channels in the AODV routing protocol. The authors consider the timing between the route requests, the source sequence number, the lifetime field, and the destination ID.

2.2 Covert Channel Detection

Detection is a common practice in secure systems in order to monitor malicious activity [Common Criteria 1998]. In the literature, covert channel detection is often mixed with covert channel identification (as in Bishop [2002] and Helouet et al. [2003]). The main difference between the two is that the latter tries to uncover a shared resource that can potentially be used as a covert channel variable [Denning 1976; Kemmerer 1983; Kemmerer and Porras 1991], while the former tries to detect an actively running covert channel by examining its output event sequence (i.e., event traffic). Detecting covert channels can be desirable for two reasons. First, detection provides a mechanism to discourage the use of these channels [NCSC 1993]. Second, covert channel elimination can be costly for high-assurance systems [Moskowitz and Kang 1994], or may be completely overlooked. In either case, allowing these channels to exist but monitoring their activity is crucial. Alternatively, detection mechanisms can run at the front end of elimination protocols such that the elimination schemes are activated only if there is a suspicious activity in the network. This architecture helps reduce the negative performance impact of the elimination schemes and additional benefits are possible if the detection schemes can identify the parties involved in covert communication channel and ideally discover the covert message. This can provide a valuable tool to authorities by which they can identify malware and classified information that have been exposed to insecure parties.

Previously, it has been shown that unusual traffic patterns may lead to discovery of covert shells that employ packet headers. For example, multiple ping requests within a small time interval may indicate the presence of an ICMP covert shell such as Loki [Daemon9 1996]. In addition, covert shells can be detected by observing an anomaly in unused packet header fields [Handley and Paxson 2001]. Others illustrate an offline detection scheme that uses support vector machines (SVM) to detect ICMP (payload) and TCP/IP header (IP ID and TCP ISN) channels [Sohon et al. 2003a, 2003b]. Still others argue that TCP and IP specifications exhibit sufficient structure to define what is normal, and that learning methods such as SVM are overkill [Murdoch and Lewis 2005], and they present a suite of tests to detect whether the IP ID and TCP ISN generating processes are in compliance with the specifications.

These schemes have been fairly successful in detecting covert shells but none can detect the type of covert channels that use packet timings in distributed systems. This is because in such systems packet headers are already protected by policy (i.e., Eve cannot access any information that is transmitted using Alice's network packet headers) and none of the above schemes address detecting anomalous traffic patterns that can potentially be created by a covert channel.

2.3 Attacker Model

Our research specifically focuses on network covert channels that arise in distributed TCP/IP systems even when the transmission lines between network nodes are controlled. In our example, Alice and Bob share the same security class and use a shared communication channel to transmit sensitive data. The particular threat we consider here, without loss of generality, is malicious software that can be planted on Alice's host to steal data on Eve's behalf, which is then sent via a covert channel on the shared channel.[2] This clearly results in a policy violation if Eve is an eavesdropper that is capable of monitoring the shared line and is not authorized access to the leaked information. We assume that Eve can detect the presence of the packets traveling over the shared network from Alice to Bob, although direct access to the packets (including packet headers) is prevented by encryption. This can happen if Alice is sending data to Bob, but it can also happen if Bob is sending data to Alice. In this situation, the malware can modify the timing of the TCP/IP acknowledgment packets sent by Alice to notify the successful receipt of a packet sent by Bob. If Eve is additionally capable of isolating the network packets originated from Alice and destined to Bob, the resulting channel is a noiseless IP covert channel. Otherwise, the channel is noisy as it contains additional legitimate traffic.

Because direct access to the packets is not permitted, the only information shared between the malicious software on Alice's system and Eve is the timing of the packets traveling from Alice to Bob. We assume use of the standard IP four-tuple of source and destination address and source and destination port to differentiate between data streams but do not use them to identify covert traffic. Note that this rules out the possibility of transmitting covert information using the packet headers because this is disallowed by the mechanisms that control access; in practice, the rewriting of the headers by the VPN. Hence, the aforementioned covert shells do not apply to our scenario.

3. IP SIMPLE COVERT CHANNELS

Within a distributed system that properly follows an access control policy, an untrusted observer may still be able to observe the presence of network traffic without having access to the encrypted contents. Assuming that Alice is a trusted sender whose system is compromised by Eve's malware; Bob is a trusted receiver communicating with Alice over an encrypted link; and Eve is an untrusted observer, then the only information shared between the malware on Alice's system and Eve is the timing of the packets traveling from Alice to Bob. The IP simple covert channels (SCC) we present in this section use this timing information to covertly transmit a secret message to Eve by adjusting the transmission intervals of the network packets Alice sends to Bob.

[2] Additional threats could include Alice herself being an insider attacker who is trying to leak information; a compromised router or gateway that can alter the timing of its packet forwarding; or in a more extreme case, malicious hardware in the network interface.

In this section, we detail the covert channel operation and implementation for both storage and timing IP SCCs and provide a discussion on channel efficacy.

3.1 Covert Channel Operation

IP SCCs transmit a covert message to Eve bit by bit; hence the covert channel symbol set is composed of zeros and ones. The lone event in the system is `packet_arrival`, which is controlled by Eve's malware on Alice's system and observed by Eve. To send consecutive bits to Eve, the malware generates a sequence of packets each separated by a timing interval, thus affecting the interarrival time. Depending on how this timing interval is chosen, a storage or a timing channel can be devised. In particular, if Eve's malware uses a single timing interval τ to denote the interarrival times between the packets throughout the covert communication, then the resulting channel is a storage IP SCC using Moskowitz's definition [Moskowitz and Kang 1994]. Alternatively, if it uses a set of timing intervals τ_1, τ_2, ..., τ_n for the same purpose, the resulting channel is a timing IP SCC, again using Moskowitz's definition. While the underlying operation is similar, we show that the latter channel protocol is easier to implement and is more robust in the presence of timing (i.e., synchronization) errors. We describe the operation of each type of channel below.

The storage IP SCC operates as follows: Eve's malware and Eve agree in advance on a constant timing interval (τ) and the starting protocol (either a particular time or in response to a network event, such as the first packet sent). To send a zero, the malware maintains silence throughout the interval τ and does not send any packets to Bob, perhaps by delaying transmission of a legitimate packet. To send a one, the malware transmits a single packet to Bob in the middle of τ. Eve monitors the transmission line between Alice and Bob. She records a zero if no packets are observed throughout the interval τ, a one otherwise. By observing each τ consecutively, Eve records the entire binary string transmitted over the covert channel.

The timing IP SCC operates as follows: Eve's malware and Eve agree on a set of timing intervals $\tau_1, \tau_2, ..., \tau_n$, a set of associations for each timing interval ((τ_1, zero), (τ_2, one), etc.), and the starting protocol. For simplicity, we assume that they agree on two timing intervals τ_1 and τ_2 with associations (τ_1, zero) and (τ_2, one). To send a zero, the malware sleeps for τ_1 units of time and sends a packet at the end of this interval. To send a one, it sends a packet after sleeping for τ_2 units of time. On the receiving end (of the covert channel), Eve monitors the transmission line between Alice and Bob. Upon the observance of a network packet, Eve 1) records the interarrival time τ, 2) compares τ to both τ_1 and τ_2, and 3) records a zero or one depending on the previous comparison. Again by observing consecutive interarrival times, Eve records the entire binary string transmitted over the covert channel.

We list our observations and assumptions in designing both types of IP SCCs as follows:

(1) The original payloads of the network packets transmitted over the direct channel between Alice and Bob are legitimate and allowed. Indeed, the

covert message sent over the indirect channel is independent of the original packet payloads.

(2) The raw data that flows across the covert channel is binary but the actual interpretation of the binary string is up to Eve.

(3) Additional bits may also be included in the transmission for three reasons: additional parity bits may be appended to the data to add redundancy for error correction due to transmission errors such as when a packet is lost or delayed; additional bits may be added for purposes of maintaining synchronization between the malware and Eve's event clocks; and the covert data may be encrypted in order to add a further layer of privacy and obfuscation. In this paper, we employ error-correcting codes and leave the self-synchronizing codes and encryption for future work.

(4) The covert message for transmission is subdivided into smaller blocks of binary data, which we refer to as *frames*. A frame consists of data bits, synchronization bits, and error-correcting bits. All the frames are of equal length. The actual length, as well as the interval between frames, is influenced by parameters of the encoding scheme and the network. Our frames are composed of the eight-bit ASCII encodings of each character, a start of frame (SOF) bit (used only in storage IP SCCs), and optional six bits for error correction using (7-4)-Hamming codes. We explain the use of the SOF bit in Section 3.2 and detail the use of Hamming codes in Section 3.3. We note that many different error correction codes may be used; we selected the Hamming code for ease of implementation.

(5) We assume a unidirectional communication model for the covert channel. Note that only the indirect covert channel is assumed to be unidirectional; the direct channel itself is still bidirectional and the TCP/IP packets are ACKed. Assuming a unidirectional channel means that Eve cannot communicate with the malware using the covert channel itself. Restricting the channel to be unidirectional increases the difficulty in implementing an error-free channel. In particular, Eve cannot: acknowledge the correct receipt of covert packets; rate limit the malware; or indicate when to resynchronize.

3.2 Covert Channel Implementation

We implemented our covert channels using the C Berkeley socket library for our communication protocol, and Python version 2.3 to encode/decode the data transmitted over the channel and as a wrapper that called the C library functions. The software was developed for and ran under Red Hat Linux 9.0 kernel version 2.4.22.

The channel implementation differs slightly for the storage and timing IP SCCs. The storage IP SCC software uses both blocking and nonblocking Berkeley sockets and alternates between them to transmit a covert message to Eve. In particular, Eve's malware uses a blocking socket to wait for the SOF packet to arrive at the beginning of each frame. Once this packet is observed, the malware switches to a nonblocking socket to record the remaining bits of the frame and reverts back to the blocking socket to wait for the SOF packet of the next frame. In contrast, the timing IP SCC software uses a single blocking socket to record the packet arrival times and does not employ an SOF bit. Note that the implementation of the latter is more straightforward and is less prone to timing (i.e., synchronization) errors.

Lastly, both IP SCCs can be configured to run on any application port. Because the traffic pattern is expected to vary based on the application, choice of the protocol in which to hide the channels can affect detection ability.

3.3 A Discussion on Channel Efficacy

Two factors affect covert channel efficacy. The first factor is *contention noise*, which denotes the amount of legitimate traffic observed in the covert channel by Eve [Moskowitz 1991]. The second factor is *jitter*, which may result in loss of synchronization between the malware's and Eve's packet clocks.

In our study, we investigated both noiseless and noisy IP SCCs. In the former case, the covert channel between Eve and her malware is free of contention noise. In other words, Eve is capable of isolating the packets generated by Alice and destined to Bob from the ones generated by or destined to other users in the system (i.e., legitimate channels). In the latter, we investigate the cases where other legitimate users can utilize the transmission line between Alice and Bob, and Eve cannot tell which IP packets belong to the covert channel.

Recall that IP SCCs leak information using the timings of packets generated by Alice's compromised system and observed by Eve. In distributed systems, there may be a delay between the first time a packet is transmitted and when it is received. If the delay is constant, Eve can simply adjust her packet clock to compensate for this delay. However if the delay is varying, that is the time it takes for a packet to travel from Alice's system to Eve changes, then the interarrival times observed by Eve are no longer exactly the same as the ones generated by Alice's system. This effect is called jitter, and in a network is mostly due to varying network conditions that result in slight increase or decreases in the latency between Alice and Eve. This can potentially affect the timing of network packets traveling from Alice to Bob which are observed by Eve, thus reduce channel efficacy.

One important problem for IP SCCs related to jitter is the synchronization of the malware's and Eve's packet clocks. For example, jitter can cause Eve to record packets as arriving in a time interval before or after the intended one. For a storage IP SCC, this skew can result in an individual bit flip, a zero-insertion, or a zero-deletion as we illustrate in Figure 1. Individual bit-flips can be corrected using error-correcting codes; however, insertion and deletion errors can potentially cause an entire series of transmission to be shifted. Note that one-insertions are only possible when a packet is fragmented or duplicated in the network. One-deletions can occur if two packets overlap and arrive in the same time interval. Further, insertion and deletion errors only apply to storage IP SCCs. This is because in timing IP SCCs, each network packet has a one-to-one correspondence with a single bit of information (except in cases fragmentation and duplication occurs), whereas in a storage IP SCC a packet

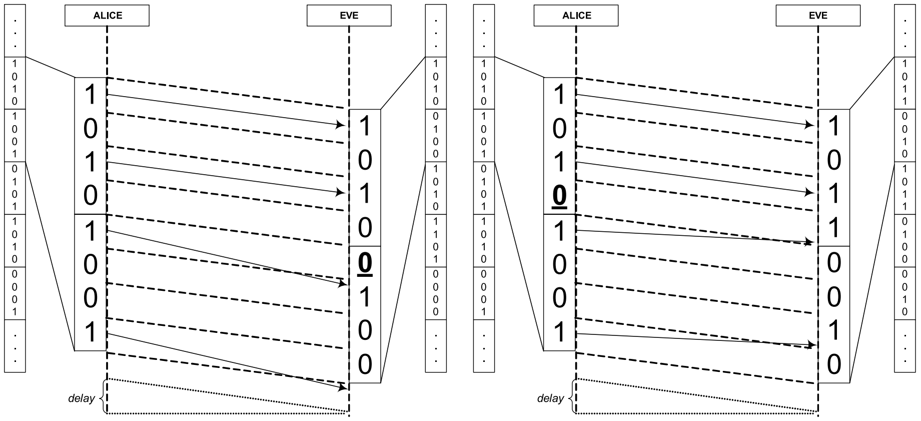

Fig. 1. A zero-insertion and a zero-deletion example for the storage IP SCC.

delay can cause additional zeros to be inserted. Therefore, in timing IP SCCs bit errors are independent from each other and are not carried to the next bit as in the storage case, where one insertion can potentially shift the entire bitstring. This advantage over the storage IP channels makes the timing channel more suitable for one-bit error-correction which we discuss next.

One way to remedy jitter-related errors is to use error-correcting codes (ECC) to correct the individual bit errors. Error-correcting codes add additional bits to the message that can be used to detect and correct some errors in transmission [Berrou et al. 1993; Moon 2005]. The code rate of an ECC is defined as the ratio between the length of an original message word and the length of its codeword. A drawback of ECCs is that they increase the amount of information to be sent. In our study, we employed Hamming ECCs, which are computationally simple to encode and decode and are capable of correcting one error per codeword [Moon 2005]. Our choice of Hamming codes is motivated solely by implementation simplicity and more sophisticated ECCs can be employed for increased accuracy. The choice of which ECC does not affect our detection ability as we do detection based on packet timings (see Section 4).

An additional scheme we employed to prevent synchronization errors for storage IP SCCs is to use an SOF bit at the beginning of each frame and to switch to a blocking socket every time a frame is completely transmitted. This scheme aims at isolating transmission errors within individual frames and stopping the propagation of errors to the following frames by aligning the malware's and Eve's timing windows every time an SOF bit is transmitted. More advanced solutions for combating synchronization errors caused by clock skew are self-synchronizing codes and phase-locked loops (PLL). The former method employs encoding schemes that are specifically designed to detect the loss of synchronization and to recover from this state (i.e., resynchronization). The latter method is a closed-loop feedback circuit that is designed to track or synchronize an output signal with an input signal in frequency and phase [Best 2003]. Both methods were proved to be effective in communications and we will employ them in future work.

3.4 Storage or Timing?

We investigated storage and timing IP SCCs separately and noted subtle differences between them. Some of these differences are not universal but are due to our design and implementation decisions (e.g., the usage of the SOF bit for synchronization purposes). Other differences can be generalized to all storage and timing channels. For example in capacity analysis, similar to the one in Moskowitz and Kang [1994], one needs to use a constant timing value τ for the storage channel and the expected value $E(\tau)$ for the timing channel.

Despite these differences, storage and timing IP SCCs are related constructs. For example, given a storage channel, one can provide a reduction to show that a timing channel equivalent can be constructed and vice versa. To see this, consider a timing IP SCC with a single state (response) `packet_observed` and timing intervals τ_1 and τ_2. Using this channel, we can construct a storage IP SCC with states `packet_present` and `packet_absent` (both derived from the single state `packet_observed`), and $\tau = \tau_1 + \tau_2$. Similarly, given a storage IP SCC with timing interval τ, we can construct a timing IP SCC with $\tau_1 = \tau$ and $\tau_2 = 2\tau$.

Additionally, both IP SCC types are equivalent in terms of the detection methods we introduce in this article, which use packet interarrival times to build measures that detect regularity. Consider a storage IP SCC with timing interval τ that generates k different interarrival time values.[3] In terms of interarrival times, this storage channel is the equivalent of a timing IP SCC with k different timing intervals.

In conclusion our detection schemes equally apply to storage and timing IP SCCs because of the above equivalence. In the following sections in which we discuss channel detection, we focus only on the storage IP SCC with a fixed timing interval as it yields higher number of interarrival times, say k different time values. Our results will then equally apply to all timing IP SCCs that use k or less timing intervals. That is, we can detect a storage IP SCC that yields k interarrival times if and only if we can detect a timing IP SCC that employs k or less timing intervals.

4. NOISELESS IP SCC DETECTION

For a storage IP SCC, there must be a prespecified timing interval between consecutive network packets sent by Alice and observed by Eve. Similarly, a timing IP SCC requires prespecified timing intervals. This behavior implies regularity in terms of packet interarrival times in comparison to packet interarrival times generated by a legitimate process. We argue that the packet traffic generated by IP SCCs is more regular than the traffic generated by legitimate processes, such as WWW traffic, and is therefore anomalous and detectable. To show an example, we plot the interarrival times from simulated timing and storage channels while transmitting a sample bit-string from Alice to Bob (Eve) in Figure 2. For the storage IP SCC, we observe that the packet

[3] Note that k is equal to the number of different number of zeros we can have between two ones in a codeword (e.g., for the set $\{11, 101\}$, $k = 2$).

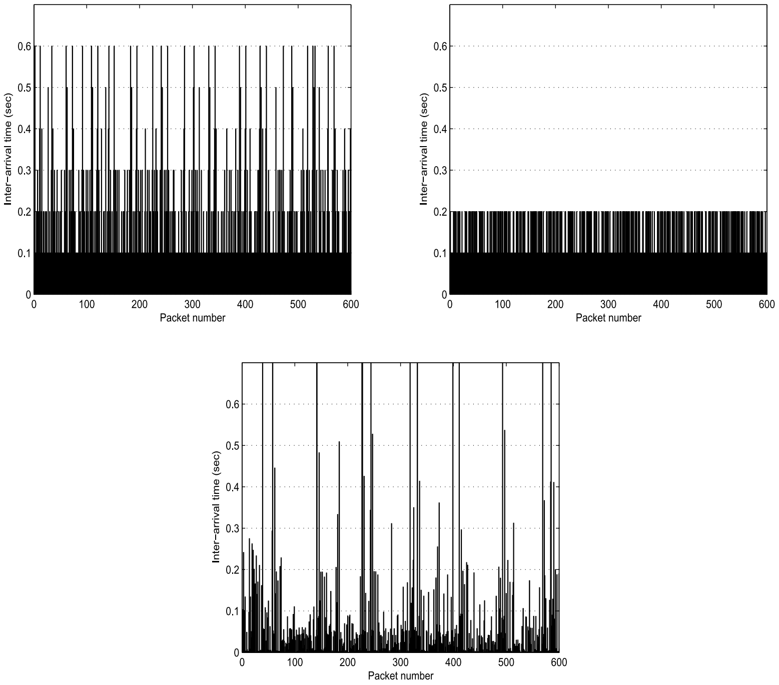

Fig. 2. Interarrival times for different traffic types. (a) A simulated IP storage SCC with τ = 0.2 secs (no clock skew). (b) A simulated IP timing SCC with τ_1 = 0.1 secs and τ_2 = 0.2 sec (no clock skew). (c) An example WWW traffic.

interarrival times can be grouped into six clusters. Similarly, the interarrival times for the timing IP SCC can be grouped into two clusters. Figure 2(c) shows that no such clustering is obvious for a sample WWW traffic.

Identifying this type of a covert channel requires being able to capture such regularity and distinguishing it from legitimate traffic. In this section, we introduce two such measures and empirically evaluate the efficacy of both in terms of false positive (FP) and false negative (FN) rates. In Section 5, we repeat the same study for noisy IP SCCs and evaluate the efficacy of both measures in the presence of contention noise.

4.1 Measures for Detection

4.1.1 *ϵ-Similarity.* In our original work, we developed a measure called ϵ-similarity for detecting IP simple covert channels [Cabuk et al. 2004]. While we refer readers to that article for details and performance results, we briefly note how ϵ-similarity is computed. First, we sort the interarrival times we collected using IP SCCs. From this sorted list we compute the relative difference between each pair of consecutive points. For example, the relative difference between P_i and P_{i+1} is computed as $\frac{P_{i+1}-P_i}{P_i}$. We illustrate this process in Figure 3 for a compilation of interarrival times we collected using our implementation of a storage IP SCC we ran between Purdue and Georgetown

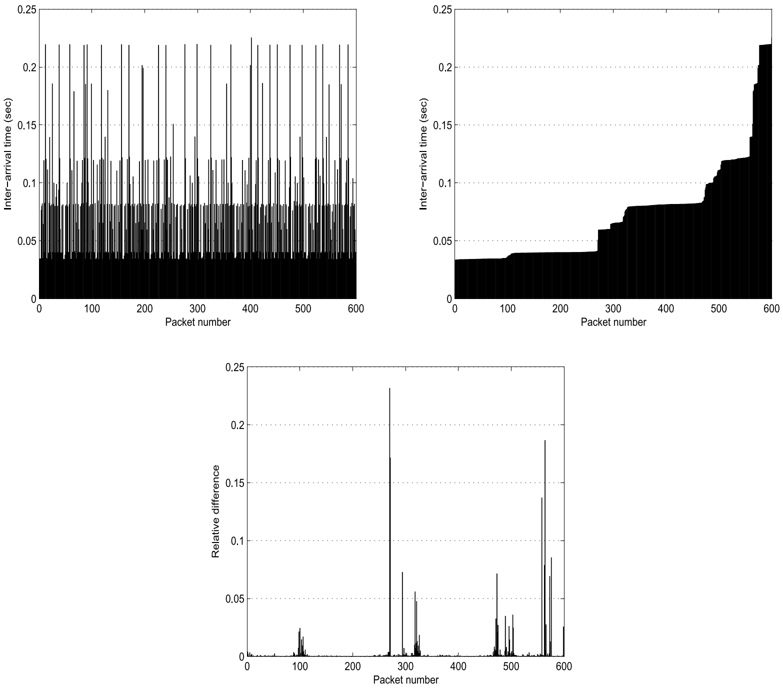

Fig. 3. Interarrival times from an actual run of a storage IP SCC with τ = 0.04 secs. (a) Actual values. (b) Sorted values. (c) Relative differences.

universities. Note that due to network jitter that creates clock skew, the interarrival times no longer create perfect clusters as in Figure 2. From the sorted interarrival times we compute a measure of similarity, which we call ϵ-*similarity*, by computing the percentage of relative differences that are less than a constant number ϵ. We have shown [Cabuk et al. 2004] that for noiseless IP SCCs the majority of the pairwise differences in the sorted interarrival time list will be small, as it is large only for jumps in the step function. Thus, the similarity scores for IP SCCs will be high. We have also shown that the IP SCC ϵ-similarity scores will be comparably higher than the legitimate channel scores.

4.1.2 *Compressibility*. As we will show, while ϵ-similarity works well for noiseless channels, it is less accurate on noisy channels. This led us to develop alternative detection mechanisms.

The Kolmogorov complexity of a string S is defined as the shortest universal computer program that produces this string (see Li and Lampson [1997]). Denoted as $K(S)$, this notion provides a lower bound on the representation of the string. In other words, $K(S) = C$ provides the maximum available compression on S, hence producing the shortest possible compressed string C. Kolmogorov complexity is not computable [Li and Lampson 1997]; however, off-the-shelf compression algorithms can be used as an approximation [Li et al. 2003].

To derive our second similarity measure, let $S \in \Sigma^s$ be a string we want to compress with any off-the-shelf compressor, where Σ is the alphabet of symbols from which the string is drawn, and s is the length of S. Let the resulting compressed string be $C \in \Sigma^c$. We define the *compressibility* of S as the compression rate we obtain by dividing the length of S by the length of C. Formally,

$$\forall S \in \Sigma^s, \exists C \in \Sigma^c \text{ such that}$$

$$C = \Im(S), \text{ and } \kappa(S) = \frac{|S|}{|C|}$$

where $\Im$ is the compressor (e.g., gzip), $\kappa(S)$ is the compressibility of S, and $|.|$ is the length operator. The compressibility of the interarrival packet times generated by a noiseless IP SCC will be higher than the ones generated by legitimate channels. Note that interarrival times are numerical values whereas compression works on strings. In Section 4.2, we detail how we convert these numbers to strings as a pre-processing step to compression.

4.1.3 *A Discussion of Other Approaches.* We additionally investigated several approaches that were not fruitful, but were more obvious from a statistical point of view.

Indicies of dispersion of a point process have been used as a tool in network characterization [Gusella 1991; Rosenberg et al. 1995]. In particular, an *index of dispersion for intervals (IDI)* can be used to qualitatively compare the interarrival times of a point process with the Poisson process serving as the basis (for which the IDI is unity) [Cox and Lewis 1966]. IDI provides a finer measure for defining the *variability* of the process than does a second order moment analysis. Gusella defines the variability, or the burstiness, of the network traffic as “the changes in the variance of the sum of consecutive interarrivals” [Gusella 1991]. Although this measure appears promising, it requires a number of assumptions including stationarity which need to be made for the correct interpretation of the results. In this study, we do not impose such assumptions on the distributions of covert or legitimate traffic.

Another avenue we examined was statistical nonparametric tests similar to those used in network traffic analysis [Paxson and Floyd 1995; Balakrishnan et al. 1997; Claffy et al. 1993]. Applications of these tests have mainly concentrated on network traffic characterization and modeling. The goal is often to determine whether two streams come from the same empirical distribution, for example, using a nonparametric goodness-of-fit test such as the Kolmogorov-Smirnov test. In our research, we are not seeking to model either the legitimate or the covert network traffic. Our goal is to define measures that differentiate covert from legitimate traffic, therefore, these methods are not directly applicable to the detection of IP SCCs.

4.2 Empirical Evaluation

In this section, we empirically show that compressibility measures were successful in detecting IP SCCs. Comparison with our earlier work shows that compressibility performed better than ϵ-similarity. In our experiments, we applied these measures on both legitimate channels and IP SCCs. Our legitimate traces were extracted from the second version of NZIX data sets (NZIX-II) which is a collection of TCP (WWW, FTP-Data, and SSH) and UDP traces collected by the WAND research group [WAND Research Group 2001]. We used twenty different traces for each legitimate traffic type. To collect the interarrival times for the covert channel, we ran three storage IP SCCs (with τ set to 0.04, 0.06, and 0.08 secs) that transmitted a sample text between Purdue and Georgetown universities. The bit string was the ASCII representation of “All human beings are born free and equal in dignity and rights. They are endowed with reason and conscience and should act towards one another in a spirit of brotherhood [EOL]” [United Nations 1948].

For each trace, we first eliminated values larger than one second in order to eliminate large interarrival time values from the data that occur when there is no packet transfer. This allows us to deal only with short values, therefore concentrate on the network effects on the interarrival times rather than the user effects (e.g., time gaps between HTTP requests). Our goal is not to model or identify a traffic distribution, but to determine whether we can accurately detect a covert channel in a short window (e.g., for online detection). Therefore, in our experiments we report the results for windows of size 2,000. Note that although we ran the IP SCC on a real network between Purdue and Georgetown, for the legitimate traffic we used the recorded interarrival times in the data sets. A drawback is that we cannot have the same network conditions (e.g., number of hops, same jitter), but excluding the case of jitter this does not impact our results. This is because none of our measures look at absolute interarrival time values, but rather compute measures of regularity in terms of the relative differences between these values.

4.2.1 *Compressibility Results.* Before applying the compressibility measure, we needed to convert our numerical data set into a set of strings because compression works on strings. To this end, we first smoothed each data set by taking two significant digits (and rounding the third digit), and added a letter at the beginning depending on the number of zeros after the decimal point. For example, we transformed 0.00247 to B25, and 0.0247 to A25, etc. This way, we could still identify the same values without using repeating zeros, which would create an advantage in compressibility.[4]

After this added preprocessing phase, we performed our experiments using `gzip` as the compressor [GNU 2003]. We report the mean compressibility score for each traffic type in Figure 4 averaged over twenty traces for the legitimate channels. Recall that the more regular the data set, the higher the compressibility. Our results show that the compressibility scores for the SSH, WWW, and FTP-Data data sets are much less than IP SCC compressibility scores. For example, the compressibility score for the storage IP SCC with $\tau = 0.06$ is 6.76, whereas it is 3.27 for WWW on average. In addition, we observe that as we increase the timing interval for IP SCC, compressibility increases. For

[4] Even though B25 and A25 compress better than B25 and B31, this effect is evenly distributed among all data sets, whereas repeating zeros favor data sets with small data points.

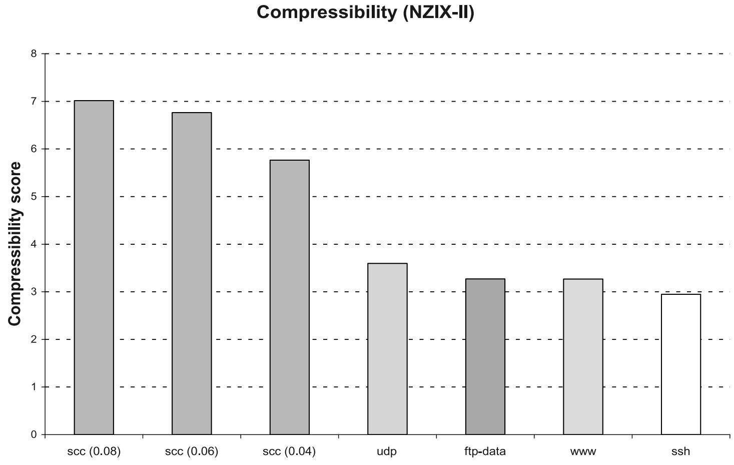

Fig. 4. Compressibility scores for various types of covert and legitimate channels.

example, the compressibility score for a 0.04 IP SCC is 5.76, whereas it is 7.01 for a 0.08 IP SCC. Recall from Section 3 that IP SCC becomes less prone to error and more accurate as we use larger timing intervals.

4.2.2 *Automatic Detection.* We evaluated the efficacy of both measures in terms of FN and FP rates. Our procedure works as follows: for each legitimate traffic type (WWW, FTP-Data, and UDP[5]), we randomly choose ten traces out of twenty for training. We reserve the remaining ten independent traces for testing. In the training phase, we calculate the ϵ-similarity or compressibility scores for each training trace and compute the mean μ and the standard deviation σ to determine our thresholds $\mu + 1.5\sigma$ and $\mu + 2\sigma$. The difference between ϵ-similarity and compressibility is that in the former we calculate seven different learned thresholds for each of seven ϵ values (0.005, 0.008, 0.01, 0.02, 0.03, 0.1, and < 0.1) that are chosen arbitrarily. For compressibility we calculate only one threshold.

To classify a testing trace as legitimate or covert, we calculate its ϵ-similarity (for each ϵ value) or compressibility score that we compare to the learned threshold. A data set is classified as covert if its combined score is above the learned threshold and legitimate otherwise. For ϵ-similarity, the combined score is determined by taking a majority vote over seven different ϵ values. For example, we classify the trace as covert if the scores for four out of seven ϵ values are over the threshold. For compressibility, the combined score is simply the compressibility score.

In our results, we report the averages of 100 runs of the above procedure. Each run uses a different partition of the twenty legitimate traces into training and testing. We report the average FP and FN rates for the ϵ-similarity and

[5] We omit the SSH results because NZIX-II did not contain enough number of SSH traces for training and testing.

Table I. Detection Efficacy for Both Measures with Thresholds $\mu + 1.5\sigma$ and $\mu + 2\sigma$

THRESHOLD	APPLICATION	FP	FN(*tau*=0.04)	FN(*tau*=0.06)	FN(*tau*=0.08)
ϵ-SIMILARITY					
$\mu + 1.5\sigma$	WWW	0.06	0.00	0.00	0.00
	FTP-DATA	0.04	0.00	0.00	0.00
	UDP	0.14	0.65	0.19	0.00
$\mu + 2\sigma$	WWW	0.04	0.08	0.00	0.00
	FTP-DATA	0.00	0.28	0.00	0.00
	UDP	0.08	0.90	0.66	0.38
COMPRESSIBILITY					
$\mu + 1.5\sigma$	WWW	0.14	0.00	0.00	0.00
	FTP-DATA	0.09	0.00	0.00	0.00
	UDP	0.12	0.19	0.00	0.00
$\mu + 2\sigma$	WWW	0.07	0.00	0.00	0.00
	FTP-DATA	0.03	0.00	0.00	0.00
	UDP	0.09	0.63	0.02	0.00

compressibility measures in Table I. We list our observations and conclusions as follows:

(1) The results suggest that we can distinguish IP SCC traces from WWW and FTP-Data traces (i.e., FN = 0%) with low false alarm rates using both ϵ-similarity and compressibility measures with thresholds $\mu + 1.5\sigma$ and $\mu + 2\sigma$ respectively.

(2) UDP results with high FN rates suggest that we cannot distinguish IP SCC traces from UDP traces. UDP is a best-effort protocol which transmits packets as soon as they are ready and there is no flow control. Therefore, UDP interarrival times can be regular under stable network conditions, thus generate higher ϵ-similarity and compressibility scores as compared to TCP interarrival times.

(3) The compressibility measure fares better than the ϵ-similarity measure in terms of the FN rates with comparable FP rates. Combining this with the fact that compressibility uses only one measurement whereas ϵ-similarity requires a majority vote over seven measurements, we conclude that compressibility is a superior measure.

(4) As shown before, IP SCCs with higher τ values are easier to detect. This is because using higher timing intervals provides more accurate channels that generate more regular interarrival times.

(5) The FP rates for both measures can be lowered by increasing the number of data sets we use in the training phase. Using only ten data sets for training was an experimental shortcoming as we were limited by the available legitimate data. In a real detection environment, hundreds of such traces can be used to train the classifier, potentially reducing FP rates further.

5. NOISY IP SCC DETECTION

A noisy IP SCC is a network covert channel in which we observe both legitimate and covert traffic. This channel may arise in cases where Eve cannot isolate the network packets originating from Alice due to the shared use of the transmission line that is being tapped. In addition, introducing noise into the channel artificially has been shown to be effective in terms of jamming the covert channel and limiting its bandwidth [Giles and Hajek 2003]. In our study, we assume that the malware on Alice's system injects noise into the transmission line intentionally to increase the irregularity of the packet interarrival times and thwart the regularity-based detection schemes.

In this section we first empirically show that if channel noise reaches a certain level, the regularity-based detection measures fail to identify the covert channel. As a result Eve's malware can potentially hide the covert channel by mixing it with legitimate traffic. To detect such hidden IP SCCs we require more localized measures to focus on the suspicious segments of the data set. To do so, we employ compression-based similarity measures we use with a sliding window algorithm to search for covert channels in mix data sets.

5.1 Empirical Evaluation

We first show that regularity-based measures fail to detect noisy IP SCCs for noise levels higher than 10%. We apply the ϵ-similarity and compressibility measures to mix data sets that we formed by randomly mixing packet interarrival times extracted from WWW, FTP-Data, and UDP data sets with the ones generated by IP SCCs. As an example, to create a 10% FTP-Data mix data set with 2000 data points, we pick 200 consecutive data points from a FTP-Data data set and randomly mix it with 1800 data points we select from the IP SCC data set. Note that how we mix these data points does not matter for our detection measures because neither measure pays attention to the relative ordering of the individual data points; both measures rearrange the data points as a pre-processing step (e.g., the ϵ-similarity measure sorts the data points).

In our experiments we used the same setup and data sets as in Section 4. We illustrate the average scores for the ϵ-similarity measure for each mix data set with noise levels at 10%, 25%, and 50% in Figure 5. The ϵ-similarity scores for the mix data sets are comparably lower than the ones we calculated in Section 4. For example, the mean 0.005-similarity score for the pure IP SCC (τ = 0.06) is 52 while a 25% FTP-Data mix data set generates a score of 34.

The average scores for the compressibility measure yield a similar trend which we illustrate in Figure 6. The compressibility scores for the mix data sets are substantially lower than the ones we calculated for pure covert channels in Section 4. As an example, the mean compressibility score for the pure IP SCC (τ = 0.06 secs) is 6.76 while a 25% FTP-Data mix data set generates a score of 4.98.

To determine how automatic detection with both measures would fare in the presence of noise, we repeated the automatic detection study we presented in Section 4 to compute the FN and FP rates. In particular, we chose ten training traces out of twenty legitimate data sets to compute the mean μ and the standard deviation σ to determine our thresholds $\mu + 1.5\sigma$ (for ϵ-similarity) and $\mu + 2\sigma$ (for compressibility). We reserved the remaining ten independent traces for testing. We averaged the results over 100 runs each time using a different partition of the twenty legitimate traces into training and testing.

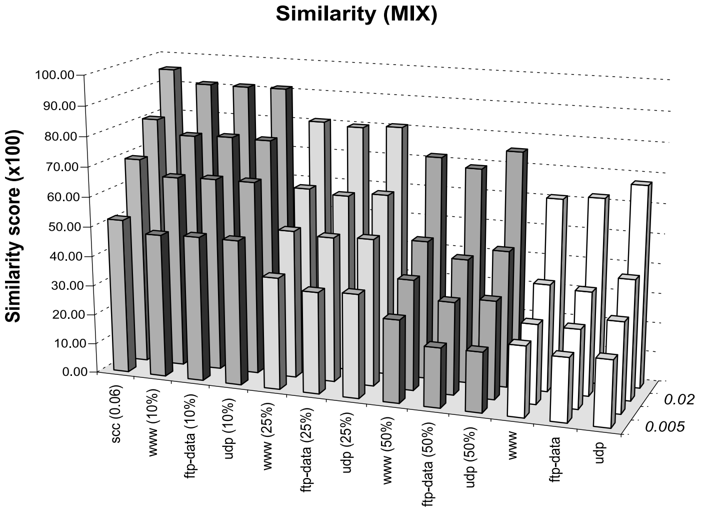

Fig. 5. ϵ-similarity scores for the noisy IP SCC with different traffic types and noise levels set at 10%, 25%, and 50%.

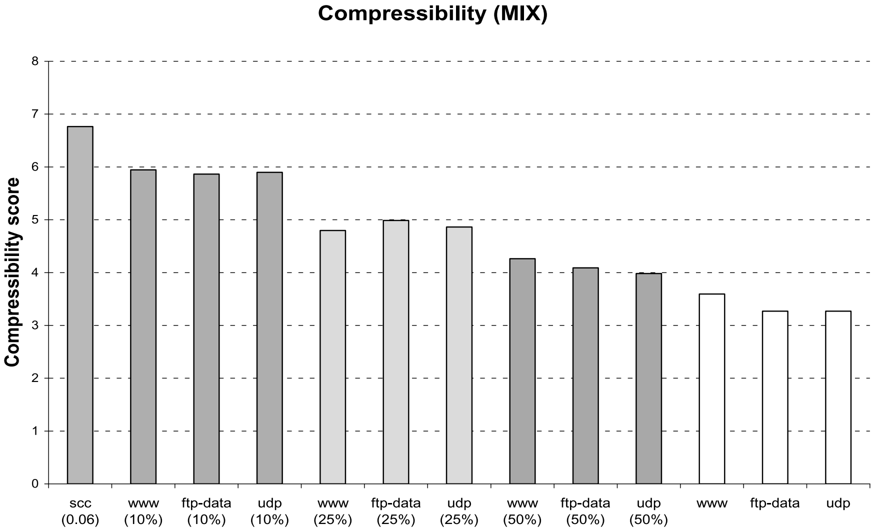

Fig. 6. Compressibility scores for the noisy IP SCC with different traffic types and noise levels set at 10%, 25%, and 50%.

We report the results in Table II and list our observations and conclusions as follows:

(1) The results suggest that both measures can detect WWW and FTP-Data mix data sets up to 10% noise level. We conclude that our measures work for low noise levels.

Table II. Detection Efficacy for Both Measures when Detecting Noisy IP SCCs with Thresholds $\mu + 1.5\sigma$ (for ϵ-similarity) and $\mu + 2\sigma$ (for Compressibility)

Application	Noise	FP	FN(0.04)	FN(0.06)	FN(0.08)
ϵ-SIMILARITY					
WWW	10%	0.06	0.00	0.00	0.00
	25%		0.76	0.23	0.00
	50%		1.00	0.98	0.80
FTP-DATA	10%	0.04	0.04	0.00	0.00
	25%		0.99	0.64	0.00
	50%		1.00	1.00	1.00
UDP	10%	0.15	0.81	0.45	0.08
	25%		0.99	0.95	0.80
	50%		1.00	1.00	0.99
COMPRESSIBILITY					
WWW	10%	0.07	0.00	0.00	0.00
	25%		0.00	0.00	0.00
	50%		0.95	0.92	0.74
FTP-DATA	10%	0.04	0.00	0.00	0.00
	25%		0.52	0.00	0.00
	50%		1.00	0.96	0.95
UDP	10%	0.09	0.71	0.48	0.22
	25%		0.94	0.90	0.84
	50%		0.99	0.99	0.99

(2) Compressibility further detects WWW and FTP-Data mix data sets up to 25% noise level for which the ϵ-similarity measure completely fails. We conclude that compressibility is a more robust measure when detecting noisy IP SCCs.

(3) Neither measure can detect 50% mix data sets regardless of the mix type and UDP mix data sets regardless of the noise level.

(4) Lastly, as mentioned in Section 4 the FP rates for both measures can be lowered by increasing the number of data sets we use in the training phase.

5.2 Measures for Detection

The previous section suggests that our regularity-based detection measures alone fail to detect the presence of IP SCCs given a mix data set. To detect noisy IP SCCs, we take the approach of identifying and eliminating the noise from the mix data set instead of trying to find a measure that is robust in the presence of noise. This requires a more localized methodology that focuses on the suspicious segments of the data set individually and also in relation to each other. To do so, we employ a sliding window methodology that walks over the mix data set using windows of size w and strides of size s. The idea is to collect local information about each window individually, and also in relation to each other. To collect individual and relative information about each window, we introduce *compressibility-walk* and *CosR-walk* methods, respectively.

Once we identify the windows that are potentially generated by a covert channel, we eliminate the remaining and apply our measures only to these ones. Because our regularity-based measures work effectively for low noise levels, this local approach can potentially generate better detection rates as compared to the cases for which we apply the measures globally.

5.2.1 *Compressibility-Walk.* Compressibility-walk walks over the mixed data set in jumps of size s and computes the compressibility score for each window of size w. The windows are allowed to overlap in cases where $s < w$. The compressibility scores of the data sets generated by IP SCCs are higher than the ones generated by legitimate channels, therefore the windows that contain covert channels will reveal themselves as peaks when plotted on a graph. Note that we could also devise an ϵ-similarity-walk approach, but compressibility is a superior measure to ϵ-similarity, especially in the presence of noise, as we showed in Section 4.

5.2.2 *CosR-Walk.* In the following section we show that examining individual windows alone is not sufficient for covert channel identification. Therefore we need to investigate the relation between two consecutive windows, which requires a metric that measures the similarity between the two.

Compression can be used as a similarity metric, which gives an approximation of the conditional Kolmogorov complexity [Sculley and Brodley 2006]. Given two strings S and T, the conditional Kolmogorov complexity $K(S|T)$ is defined as the shortest representation of S given T. Intuitively the more information S and T share, the shorter the representation of S given T.

Compression-based similarity metrics have been used in various research areas [Li et al. 2003; Keogh et al. 2004; Cilibrasi et al. 2004; Cilibrasi and Vitanyi 2005; Wehner 2004; Loewenstem et al. 1995]. Let $\Im$ be a compressor that approximates Kolmogorov complexity. In Keogh et al. [2004], the conditional compression of S given T is approximated by the compression-based distance measure (CDM), defined as:

$$CDM(S,T) = \frac{\Im(S|T)}{\Im(S) + \Im(T)}$$

where | is the concatenation operator. Note that CDM is a dissimilarity measure. In Sculley and Brodley [2006], the authors show that the *CosR* metric performs better than CDM, which is defined as:

$$CosR(S,T) = 1 - \frac{\Im(S) + \Im(T) - \Im(S|T)}{\sqrt{\Im(S)\Im(T)}}$$

Combining this metric with our sliding window approach, CosR-walk walks over the mixed data set in a similar fashion to compressibility-walk, only this time computing the CosR scores for consecutive windows. Again, we allow the windows to overlap. CosR is a similarity measure; the score when one window contain data points generated by a legitimate channel and the other generated by a covert channel will be lower than the score when both windows contain data points generated by a legitimate channel.

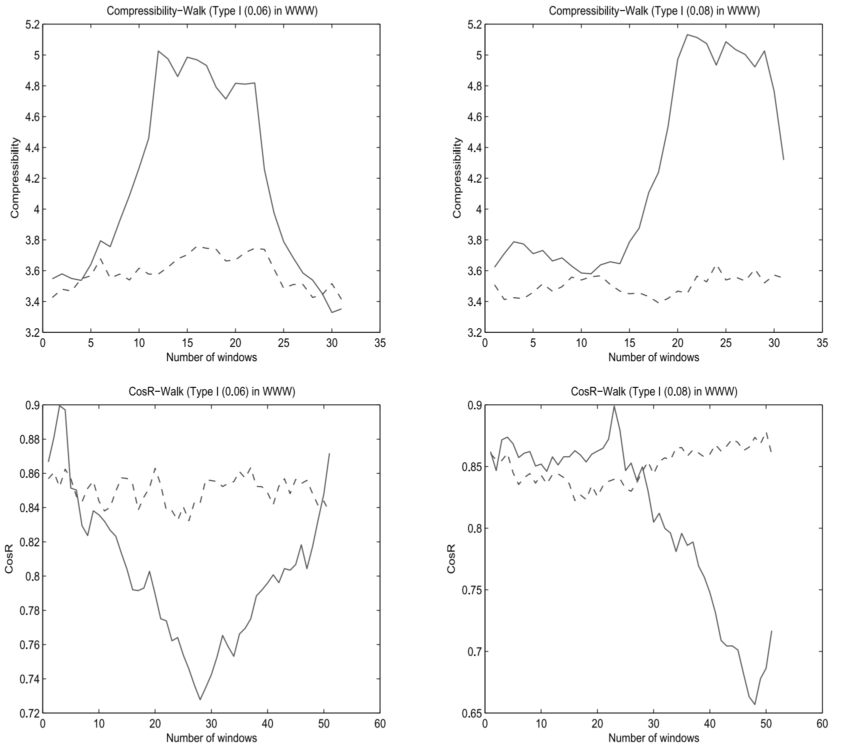

Fig. 7. (Top row) Compressibility-walk results for (dashed) a pure WWW channel, (solid) a 50% WWW mix channel. (Bottom row) CosR-walk results for (dashed) a pure WWW channel, (solid) a 50% WWW mix channel.

5.3 Empirical Evaluation

In this section we empirically show that both compressibility-walk and CosR-walk have successfully identified hidden covert channels in mix data sets, with CosR-walk being superior (i.e., was able to identity the covert channel in more cases). In our experiments we used the same setup and data sets as in Section 5.1. To demonstrate the sliding window approach, we ran compressibility-walk on a WWW data set with 2,000 data points with w and s set to 500 and 20. The dashed lines in the top two graphs in Figure 7 illustrate the cases where the data set was purely generated by a WWW channel. The solid line on the same graph illustrates the results when we ran the same walk on a 50% WWW mix data set. The peaks in each of the upper two graphs show the location of each hidden covert channel. We conclude that IP SCCs will reveal themselves as peaks as long as the compressibility of the legitimate and covert channels are distinguishable. The resulting plots for FTP-Data are similar and we omit them.

Similarly we ran our conditional compressibility method CosR-Walk on the same data sets with w and s set to 500 and 20. The dashed lines in bottom two graphs in Figure 7 show the results when the procedure was run on pure

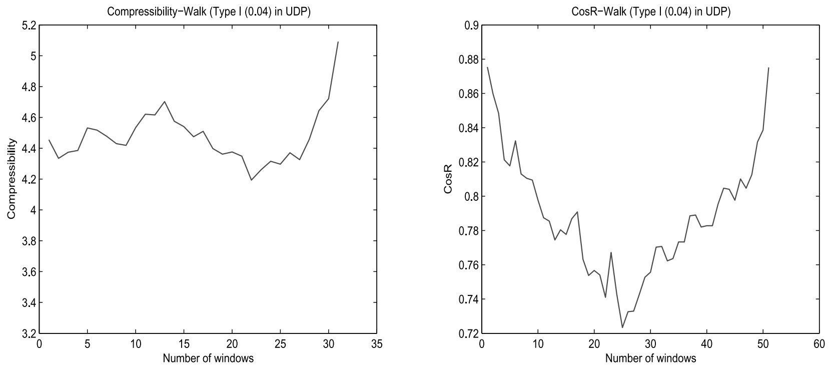

Fig. 8. (Left) Compressibility-walk fails to detect the hidden channel in a 50% UDP mix data set. (Right) CosR-walk detects the hidden channel.

WWW data sets. The solid line on the same graph again illustrates the same walk on a 50% WWW mix data set. This time a V-shaped curve in each bottom graph suggests the existence of a covert channel. As expected, the locations of these channels are consistent with the compressibility graphs. Again the graphs for FTP-Data data sets show a similar trend and we omit them.

To show that CosR-walk is superior to compressibility-walk, we ran both walks on a 50% UDP mix data set. In Figure 8 on the left, we illustrate one unsuccessful attempt by compressibility-walk to locate the covert channel. The curve on this graph appears to be flat suggesting that no covert channel exists, even though there is one hidden in the middle section. This phenomenon occurs because the compressibility scores for the UDP data sets are as high as IP SCC scores, which causes compressibility-walk to fail. (Recall that compressibility-walk works only when the compressibility score of the covert channel is significantly higher than the legitimate channels' scores.) In contrast CosR-walk is a relative measure for which the individual compression scores do not matter. Indeed, as illustrated in the right-hand graph that CosR-walk identifies the covert channel is shown by the signature V-shape.

The time complexity of both methods is a function of data points n, window size w, slide size s, and the running time of the compression algorithm $C(.)$. Compressibility-walk performs one compression per window and CosR-walk performs three. Hence the running time for both algorithms is $O((n/s)C(w))$. Practically compression is an expensive operation and our implementation of the search algorithms compress each window independently. However, the performance can be improved if some compression information is carried over to the current window from the previous window when windows are overlapping.

6. LIMITATIONS

We conclude with a discussion on the limitations of our search measures in detecting noisy IP SCC, particularly in the face of attacker countermeasures.

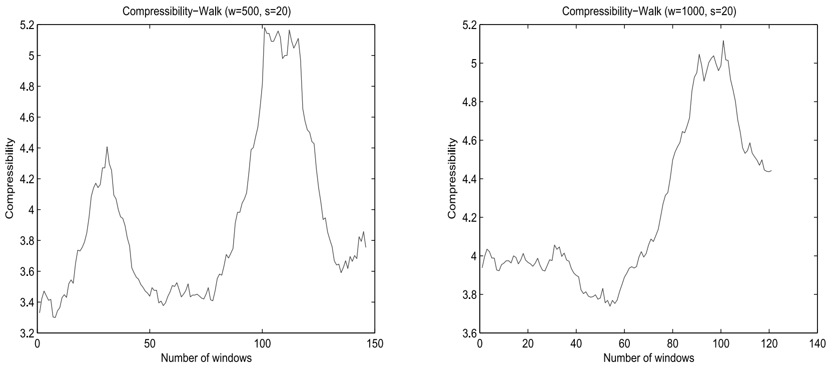

Fig. 9. (Left) Compressibility-walk detects both channels when w = 500. (Right) Compressibility-walk detects only one channel when w = 1000.

6.1 Parameter Choice and Complexity

In our experiments we used window and slide sizes that gave us the optimal results. However, in reality both search methods are sensitive to the choice of parameters w and s. In particular, choosing a window size larger than the covert channel size may not reveal the channel at all. We illustrate one such example for compressibility-walk in Figure 9. On the left graph, both covert channels are identified by setting w = 500, whereas on the right only one covert channel is visible by setting w = 1000. Because the size of the covert channel is not known at the time of detection, the methods need to be run with different sets of parameters to identify the hidden covert channels. This would require additional hardware or processing time, and is best done in an offline manner.

6.2 Attacker Countermeasures

An attacker that suspects our methods are being used to identify network covert channels could try and take action to hide the channel. There are at least three ways that this could be done.

The most direct method would be to alter the covert channel such that it appears similar to regular traffic with respect to our measures. This presents some significant challenges for the attacker. In deployment the detection mechanism would be trained with network traffic specific to the domain, and the detection parameters set based on that traffic. The attacker would either have to access to the training set to determine the correct parameters to use for the camouflaged traffic, or know that training is occurring and be able to influence the training set as to negatively affect the parameters. These capacities far exceed our attacker model. Without knowing what the detection parameters are, the attacker would have a difficult time of producing timings that match legitimate traffic, particularly because the covert channel is one way, and timings must be agreed on in advance.

Another method of thwarting detection would be to hide short bursts of covert channel communication within legitimate traffic to create balanced windows in terms of compression scores and similarity. A challenge is that the attacker must know the window size used and make balanced bursts within the window, which requires notifying the receiver in advance as to what the correct pattern would be. Even if this were possible and did occur, the rate at which the data was sent would be significantly downgraded, as transmission would no longer be continuous. There is also the possibility of detecting the differences in traffic, which we can examine in future work.

A third method would be to add significant noise to the channel. Although this type of a noisy IP SCC would be hard to detect, note that by forcing the covert channel parties to introduce noise into the channel we rate-limit the covert channel. For example, if Alice uses a 50% mix data set to avoid detection, in reality this means that the covert channel bandwidth is halved.

7. CONCLUSIONS AND FUTURE WORK

Detection is a common practice in secure systems in order to monitor malicious activity [Common Criteria 1998]. In this article, we investigated the detection of a class of network covert channels that use the timing of IP packets to transmit a secret message over the network. We argued that such channels can potentially arise in distributed systems given the possibility that a malicious user can monitor the presence of packets traveling between two legitimate users even though direct access to the packets is limited.

We performed an empirical study to justify the hypothesis that the traffic generated by this class of covert channels will be more regular in terms of interarrival times as compared to the traffic generated by legitimate channels because of the way the former generate network packets. To show this, we introduced a new measure for detection called compressibility and showed that it performed better than the ϵ-similarity measurement we had introduced previously. Our experiments illustrated that we were able to distinguish the covert channel traffic from WWW, FTP-Data, and SSH traffic with more than 95% detection rates; although, UDP results were not that promising due to the fact that UDP regularity was similar to that of the covert channel.

We additionally investigated the cases in which a malicious user might try to hide the covert activity by injecting noise into the channel to increase the irregularity of its interarrival times. Under this scenario, our detection measures failed to locate the covert channel hidden inside the mix traffic with noise rates higher than 25%. To counter this deficiency, we investigated two methods named compressibility-walk and CosR-walk to search for hidden covert channels locally in mix data sets. These search techniques effectively revealed the presence of covert channels in mix data sets at the expense of increased algorithmic complexity. We conclude that these search methods are more suitable for offline detection in which one can run these algorithms on large collections of interarrival time log files to search for potential covert activity whenever there is a suspicion.

In future work, we will repeat our empirical study on various network collections that were recorded under different traffic conditions to determine how our measures would fare under these conditions. Additionally, we will extend our work on noisy IP SCCs and investigate ways to make the search algorithm more efficient. One possible avenue for improvement is to retain compression information when compressing overlapping windows instead of compressing each separately. Lastly, we will further investigate the dependency of each search method on parameter selection.

REFERENCES

ABAD, C. 2001. IP checksum covert channels and selected hash collision. Tech. rep., University of California.

AHSAN, K. 2000. Covert channel analysis and data hiding in TCP/IP. M.S. thesis, University of Toronto.

AHSAN, K. AND KUNDUR, D. 2002. Practical data hiding in TCP/IP. In *Proceedings of the Workshop on Multimedia Security (MMSEC'02)*, 63–70.

BALAKRISHNAN, H., STEMM, M., SESHAN, S., AND KATZ, R. H. 1997. Analyzing stability in wide-area network performance. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'97)*, 2–12.

BAUER, M. 2003. New covert channels in HTTP: Adding unwitting web browsers to anonymity sets. In *Proceedings of the ACM Workshop on Privacy in the Electronic Society (WPES'03)*, 72–78.

BERROU, C., GLAVIEUX, A., AND THITIMAJSHIMA, P. 1993. Near Shannon limit error-correcting coding and decoding: Turbo codes. In *Proceedings of the IEEE International Conference on Communications (ICC'93)*, *2*, 1064–1070.

BEST, R. E. 2003. *Phase-Locked Loops: Design, Simulation and Applications*, 5th Ed. McGraw-Hill Professional.

BISHOP, M. 2002. *Computer Security: Art and Science*. Addison Wesley Professional.

CABUK, S., BRODLEY, C. E., AND SHIELDS, C. 2004. IP covert timing channels: Design and detection. In *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS'04)*, 178–187.

CASTRO, S. 2003. CCTT: Covert channel tunneling tool. Tech. rep., The Gray-World Team.

CILIBRASI, R. AND VITANYI, P. M. B. 2005. Clustering by compression. *IEEE Trans. Inf. Theory 51*, 4, 1523–1545.

CILIBRASI, R., VITANYI, P. M. B., AND WOLF, R. D. 2004. Algorithmic clustering of music based on string compression. *Comput. Music J. 28*, 4, 49–67.

CLAFFY, K. C., POLYZOS, G. C., AND BRAUN, H.-W. 1993. Application of sampling methodologies to network traffic characterization. In *Proceedings on Communications Architectures, Protocols and Applications (SIGCOMM'93)*, 194–203.

COMMON CRITERIA. 1998. Common criteria for information technology security evaluation, version 2.0 Ed. ISO/IEC Standard 15408.

COX, D. R. AND LEWIS, P. A. W. 1966. *The Statistical Analysis of Series of Events*. Chapman and Hall.

DAEMON9. 1997. Loki2 (the implementation). *Phrack 51*, 6.

DAEMON9. 1996. Project Loki. *Phrack 49*, 6.

DENNING, D. E. 1976. A lattice model of secure information flow. *Comm. ACM 19*, 5, 236–243.

DEPARTMENT OF DEFENSE. 1985. Trusted computer system evaluation criteria, 5200.28-STD Washington: Government Publishing Office.

DOGU, T. M. AND EPHREMIDES, A. 2000. Covert information transmission through the use of standard collision resolution algorithms. In *Proceedings of the 3rd International Workshop on Information Hiding (IH'00)*, 419–433.

GIFFIN, J., GREENSTADT, R., LITWACK, P., AND TIBBETTS, R. 2002. Covert messaging through TCP timestamps. In *Proceedings of the Workshop on Privacy Enhancing Technologies (PET'02)*, *2482*, 194–208.

GILES, J. AND HAJEK, B. 2003. An information-theoretic and game-theoretic study of timing channels. *IEEE Trans. Inf. Theory 48*, 9, 2455–2477.

GIRLING, C. G. 1987. Covert channels in LANs. *IEEE Trans. Softw. Eng. SE-13*, 2, 89–101.

GNU. 2003. GNU zip utility. http://www.gzip.org.

GUSELLA, R. 1991. Characterizing the variability of arrival processes with indexes of dispersion. *IEEE J. Select. Areas Comm. 9*, 2, 203–211.

HANDEL, T. AND SANDFORD, M. 1996. Hiding data in the OSI network model. In *Proceedings of the 1st International Workshop on Information Hiding (IH'96)*. Springer-Verlag, 23–38.

HANDLEY, M. AND PAXSON, V. 2001. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *Proceedings of the 10th USENIX Security Symposium (SECURITY'01)*, 9.

HAUSER, V. 1999. Placing backdoors through firewalls. Tech. rep., The Hacker's Choice.

HELOUET, L., JARD, C., AND ZEITOUN, M. 2003. Covert channels detection in protocols using scenarios. In *Proceedings of Workshop on Security Protocols Verification (SPV'03)*, 21–25.

HENRY, P. A. 2000. Covert channels provided hackers the opportunity and the means for the current distributed denial of service attacks. Tech. rep., CyberGuard Corporation.

HU, W. M. 1992. Reducing timing channels with fuzzy time. *J. Comput. Secur. 1*, 3–4, 233–254.

KARGER, P. A. AND WRAY, J. C. 1991. Storage channels in disk arm optimization. In *Proceedings of the IEEE Computer Society Symposium of Research in Security and Privacy (SP'91)*, 52–61.

KEMMERER, R. A. 1983. Shared resource matrix methodology: An approach to identifying storage and timing channels. *ACM Trans. Comput. Syst. 1*, 2, 256–277.

KEMMERER, R. A. AND PORRAS, P. A. 1991. Covert flow trees: A visual approach to analyzing covert storage channels. *IEEE Trans. Softw. Eng. 17*, 11, 1166–1185.

KEOGH, E., LONARDI, S., AND RATANAMAHATANA, C. A. 2004. Towards parameter-free data mining. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDDM'04)*, 206–215.

LI, M., CHEN, X., LI, X., MA, B., AND VITANYI, P. 2003. The similarity metric. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms (SIAM'03)*. Society for Industrial and Applied Mathematics, 863–872.

LI, M. AND LAMPSON, W. 1997. *An Introduction to Kolmogorov Complexity and Its Application* 2nd Ed. Springer.

LI, S. AND EPHREMIDES, A. 2004. A network layer covert channel in ad-hoc wireless networks. In *Proceedings of the 1st IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks (SECON'04)*, 88–96.

LOEWENSTEM, D., HIRSH, H., YIANILOS, P., AND NOORDEWIER, M. 1995. DNA sequence classification using compression-based induction. Tech. rep., DIMACS.

MILLEN, J. 1999. 20 years of covert channel modeling and analysis. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'99)*. 113–114.

MOON, T. K. 2005. *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley-Interscience.

MOSKOWITZ, I. S. 1991. Variable noise effects upon a simple timing channel. In *Proceedings of the IEEE Symposium on Security and Privacy (SP'91)*. 362–372.

MOSKOWITZ, I. S. AND KANG, M. H. 1994. Covert channels - Here to stay? In *Proceedings of the 9th Annual Conference on Computer Assurance (COMPASS'94)*. National Institute of Standards and Technology, 235–244.

MURDOCH, S. J. AND LEWIS, S. 2005. Embedding covert channels into TCP/IP. In *Proceedings of the Workshop on Information Hiding (IH'05)*, *3727*, 247–261.

NCSC. 1993. A guide to understanding covert channel analysis of trusted systems. Tech. Rep. Library No. S–240,572, National Computer Security Centre.

PAXSON, V. AND FLOYD, S. 1995. Wide area traffic: The failure of Poisson modeling. *IEEE/ACM Trans. Netw. 3*, 3, 226–244.

ROSENBERG, C., GUILLEMIN, F., AND MAZUMDAR, R. 1995. New approach for traffic characterization in ATM networks. In *Proceedings of the IEEE International Conference on Communications (ICC'95)*, *142*, 87–90.

ROWLAND, C. 1997. Covert channels in the TCP/IP protocol suite. Tech. rep., First Monday.

RUTKOWSKA, J. 2004. The implementation of passive covert channels in the Linux kernel. Tech. rep., Chaos Communication Congress.

SCHAEFER, M., GOLD, B. B., LINDE, R., AND SCHEID, J. 1977. Program confinement in KVM/370. In *Proceedings of the ACM Annual Conference (ACM'77)*, 404–410.

SCULLEY, D. AND BRODLEY, C. E. 2006. Compression and machine learning: A new perspective on feature space vectors. In *Proceedings of the Data Compression Conference (DCC'06)*. IEEE Computer Society, 332–332.

SIMMONS, G. J. 1984. The prisoner's problem and the subliminal channel. In *Advances in Cryptography*, 51–67.

SMITH, J. C. 2000. Covert shells. Tech. rep., SANS Institute Information Security Reading Room.

SOHN, T., MOON, J., LEE, S., LEE, D. H., AND LIM, J. 2003a. Covert channel detection in the ICMP payload using support vector machine. In *Proceedings of the International Conference on Information and Communications Security (ICS'03)*, 828–835.

SOHN, T., SEO, J.-T., AND MOON, J. 2003b. A study on the covert channel detection of TCP/IP header using support vector machine. In *Proceedings of the International Conference on Information and Communications Security (ICS'03)*, 313–324.

UNITED NATIONS. 1948. Universal declaration of human rights. 217A, 3.

WAND RESEARCH GROUP. 2001. NZIX-II trace archive. University of Waikato Computer Science Department. http://pma.nlanr.net/Traces/long/nzix2.html.

WEHNER, S. 2004. Analyzing network traffic and worms using compression. Tech. rep., Centrum Wiskunde and Informatica.

WRAY, J. C. 1991. An analysis of covert timing channels. In *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy (SP'91)*, 2–7.

Received March 2006; revised July 2008; accepted November 2008