Summary-Invisible Networking: Techniques and Defenses

Lei Wei Michael K. Reiter Ketan Mayer-Patel

University of North Carolina, Chapel Hill, NC, USA
{lwei,reiter,kmp}@cs.unc.edu

***Abstract*—Numerous network anomaly detection techniques utilize traffic summaries (e.g., NetFlow records) to detect and diagnose attacks. In this paper we investigate the limits of such approaches, by introducing a technique by which compromised hosts can communicate without altering the behavior of the network as evidenced in summary records of many common types. Our technique builds on two key observations. First, network anomaly detection based on payload-oblivious traffic summaries admits a new type of covert embedding in which compromised nodes embed content in the space vacated by compressing the payloads of packets already in transit between them. Second, point-to-point covert channels can serve as a "data link layer" over which routing protocols can be run, enabling more functional covert networking than previously explored. We investigate the combination of these ideas, which we term Summary-Invisible Networking (SIN), to determine both the covert networking capacities that an attacker can realize in various tasks and the possibilities for defenders to detect these activities.**

I. Introduction

Due to existing router support for collecting *flow records*, there is increasing attention being devoted to performing network anomaly detection using flow logs. A typical flow record format (e.g., CISCO NetFlow) provides summary statistics (numbers of packets and bytes) for packets sharing the same addressing information (source and destination addresses and ports) in an interval of time. Other, more fine-grained summarization approaches, such as *a-b-t* records [1], further reconstruct connections and characterize their behaviors, e.g., as interactive, bulk file transfer, or web-like, on the basis of packet sizes and interleavings of packets in each direction. Such summarization approaches have proven useful for traffic classification (e.g., [2]) and diagnostics of various types (e.g., [3]), including of some security-relevant anomalies. For example, even simple flow logs have been shown to be useful for finding peer-to-peer traffic masquerading on standard ports (e.g., HTTP port 80) (e.g., [4]), various kinds of malware activities (e.g., [5], [6], [7]), and even for identifying the origin of worms [8]. Indeed, a community of security analysts now holds an annual workshop devoted to the use of flow records for such purposes (http://www.cert.org/flocon/).

In this paper we explore the limits of analysis using such traffic summaries for security purposes, by taking the attacker's perspective and investigating to what extent an attacker who compromises a collection of machines in an enterprise, for example, can perform his activities in a way that is undetectable in summary records. Because we cannot foresee every potential approach to summarization that might be employed, we take an extreme position to ensure that the attacker's activities will remain undetected by *any* summarization technique that does not inspect application payload contents. As such, the attacker is not permitted to alter the flow-level behavior of the network, for example, *at all*. The challenge, then, is to demonstrate what the attacker can accomplish under this constraint.

To provide such a demonstration, we introduce *Summary-Invisible Networking* (SIN), a networking technique that piggybacks on existing traffic to enable data interchange among compromised hosts (SINners). SIN is designed to be invisible in traffic summaries, in the sense that a log of summary records collected in the infected network should be unchanged by the presence of compromised hosts executing SIN. To accomplish this, SIN must operate under stringent constraints:

- **The number of packets between any source and destination must remain the same.** Increasing the number of packets between sources and destinations would be evidenced in flow records that report packet counts, for example. In order to avoid this, a SIN network must perform all signaling and data exchange using packets that the hosts would already send.
- **The sizes of packets must remain the same.** Increasing the sizes of packets would be evidenced in flow records that contain flow or packet byte statistics.
- **The timing of packets must be preserved.** Because some summarization techniques (e.g., [1]) take note of interstitial packet timings, the timing of packets must remain essentially unchanged, and so a SIN network must involve only lightweight processing.
- **SINners must transparently interoperate with uncompromised hosts.** The behavior of SINners as observed by uncompromised hosts must be indistinguishable from that of uncompromised hosts, lest the different behaviors induce uncompromised hosts to behave differently and, e.g., send traffic that would not have otherwise been sent. As such, a SINner must covertly discover other SINners in such a way that does not interfere with regular

interaction with uncompromised hosts.

In order to meet these requirements, SIN builds from two key observations: First, to preserve the size of each original packet it sends, a SINner compresses the original application payload to make room to insert SIN data; the receiving SINner then extracts the SIN data and restores the payload to its original form before delivering the packet. Second, mechanisms for discovering other SINners and embedding data in packets already being sent enables the establishment of a "data link layer", over which we can layer a routing protocol, for example. This routing protocol will need to accommodate the fact that SIN is purely opportunistic: unless the host is already sending a packet to a particular other host, data cannot be sent to that host. In this and other respects, our work can build from prior routing protocols for *delay-tolerant networks* [9], [10].

Using these observations, we design a framework for SIN networking and evaluate it in this paper. Specifically, we show the following:

- Using traces collected on our organization's network, we measure the characteristics of SIN capacity in packets, i.e., the space available by compressing packet payloads. We demonstrate the extent to which SIN capacity is asymmetric between SINners, and the ports (applications) that contribute the most SIN capacity to the network.
- We identify groups of nodes in our organization's network that, if compromised, could carry out a viable SIN network among them. For example, we identify a 114-node group for which the available SIN capacity from any one member to any other is at least 700kB/day and as large as 20MB/day.
- We characterize the performance of broadcast and unicast routing algorithms run among such a group of SINners. For example, we show that broadcast by flooding in the aforementioned 114-node group conveys a 1kB command (as if from a bot-master to his bots) to more than 80 of the SINners in under 2 hours and to more than 100 of the SINners in under 5 hours. To evaluate unicast, we implement and measure the performance of a state-of-the-art delay-tolerant unicast routing protocol. We show that it falls far short of fully utilizing the available SIN capacity, thus highlighting the need for unicast protocols tailored to this domain, but still enables SINners to push more than 52MB/hour to a designated drop-site (as if exfiltrating data to it), for example.

We then turn our attention to approaches that might be used to detect SIN networks. In this direction, we make the following contributions:

- We analyze the possibility of detecting changes in interstitial packet timings that come with the compression and decompression steps intrinsic to SIN processing in the network stacks of compromised computers, since some traffic summarization approaches capture interstitial times (e.g., [1]). Using response-time measurements of our organization's main web server, we demonstrate that this method of detecting SIN processing will be unreliable, at best.
- Though SIN is premised on payload-agnostic detectors, our work provides an incentive to identify lightweight, payload-sensitive measures to find SIN networks, and so we explore what those might be. We show, for example, that monitoring the byte-value distribution of DNS packets permits the reliable detection of a DNS server participating in SIN. However, we show that similar monitoring of web server response packets is ineffective, owing to the diversity of content served from web servers. We further explore testing the compressibility of web-server response packets, though this is not conclusively effective, either, particularly if the SIN nodes restrict the entropy of data they embed in packets. We conclude that even payload-sensitive measures for detecting SIN networks require further study.

To summarize, the contributions of this paper include introducing SIN networking, a novel approach to hiding from summarization-based anomaly detectors, and measuring both its capabilities and its detectability in a modern network. Our evaluation shows that SIN networks pose a viable strategy for a sufficiently patient attacker to coordinate malfeasant activities among compromised nodes, but that routing protocols that better utilize available SIN capacity could benefit from further research. We also show that SIN networking within some protocols can be detected through lightweight payload inspection, but that further research is required to do so in others.

The rest of this paper is structured as follows. We discuss related work in §II, and our goals and assumptions in more detail in §III. We present a framework for SIN networking in §IV. We then populate that framework with particular routing protocols, and evaluate the feasibility and performance of various attack activities using SIN in §V. We study ways to detect SIN networks in §VI, and conclude in §VII.

II. Related work

SIN can be viewed as embedding a covert network within another (physical) network, albeit where the "covertness" of the embedding is subject to the detector taking a limited viewpoint on the traffic (i.e., summarization). This limitation on the detector obviates the need to steganographically embed SIN communication in packets (with one exception, discussed in §IV-A). Nevertheless, SIN draws inspiration from other works focused on embedding one type of service steganographically within another, e.g., a file system within a file system [11], a file system or wiki within a media hosting and sharing service [12], [13], or communication within web counters [14].

Because the opportunity to transmit packets between any two SINners is sporadic and depends entirely on the shape

of the traffic in the legitimate network, the problem of routing in such a context can be modeled as a sort of delay-tolerant network (DTN) as described in [9], [10]. Since that early work, there has been a reasonably large body of work on routing in such environments. A number of routing and forwarding services have been described, including a strategy based on Levy walks [15], a probability-of-delivery-based distance scheme [16], a likelihood-based scheme [17], and epidemic routing [18], to name several.

It may be that all of these schemes can be shown to apply to SIN networking. There are, however, several significant differences between the nature of the opportunistic model in the SIN context and the model used to develop and evaluate these schemes. Those differences may have an impact on to what degree the mechanisms in the DTN literature can be brought to bear. First, because we assume that the participating nodes in the network have been entirely compromised by the attacker, buffer space at the intermediate nodes in the SIN network is not an issue. Many of the schemes assume that buffer in an intermediate node is a constraining resource. A second difference is that the mix of contact opportunities between any two nodes in the network is not based on either mobility or the probability of two sensors choosing the same duty cycle, but instead inherits the contact mix exhibited by the legitimate traffic that drives the SIN network. One could reason that the entropy of the contact mix in the SIN network is probably lower than that of the existing models. A more in-depth study on this question specifically is required to know more. Finally, there is no control over how much can be sent for any given contact opportunity since the size of the SIN payload is dependent on the compression ratio achieved on the original payload content.

Our motivation for studying SIN networking is to understand the limits of network anomaly detection approaches that do not examine packet payload contents. We are not the first to examine these limitations. For example, Collins et al. [19] evaluated the extent to which five proposed payload-agnostic anomaly detectors — Threshold Random Walk [20], server address entropy [21], protocol graph size and protocol graph largest component size [7], and client degree (i.e., number of addresses contacted) — could limit bot harvesting and reconnaissance activities on a large network while maintaining a specified maximum false alarm rate. Here we take a distinctly different perspective than all such past studies of which we are aware, by designing the attacker's communication to avoid detection by *any* payload-agnostic detector, and then demonstrating the attacker's ability to communicate under this constraint.

While our motivation derives from examining limitations of payload-agnostic detectors, some efforts have embraced the need to examine packet contents in order to identify malware outbreaks, e.g., [22], [23], [24], [25], [26], [27], [28]. A significant class of this type of work focuses on finding byte-level similarities in packets that suggest the frequent occurrence of similar content (e.g., [23], [25], [27], [28]). Our SIN network design necessitates no such byte-level similarities, and so we do not expect that these approaches would be effective at detecting SIN networking (nor were they intended for this purpose). Other approaches that strive to detect deviations from past byte-value distributions for an application (e.g., [22], [24], [26]) may be more successful at detecting SIN networks. We will discuss these and other approaches to efficiently examining payload contents that might be more suited to detecting SIN networks in §VI.

III. Goals and assumptions

We consider a network in which monitoring produces summary traffic records. For our purposes, a traffic summary is any log format that is insensitive to the byte values that comprise the application payloads of TCP/IP packets. More precisely, define a characterizing feature vector $c(p)$ defined on TCP/IP packets over a link such that $c(p) = c(p')$ if p and p' differ only in the contents of their application payloads.[1] For the purposes of this paper, we define this characterizing feature vector most generally to be $c(p_i) = \langle t_i, s_i, h_i \rangle$ where t_i is the time the packet i was transmitted, s_i is the size of the packet, and h_i is the header of the packet outside of the application payload. A summary record r is defined by $r \leftarrow f(c(p_1), \ldots, c(p_n))$ for some function f and for packets $p_1, \ldots, p_n$.

A particularly common type of summary record is a *flow record*. A flow record summarizes a collection of packets sharing the same protocol and addressing information (source and destination IP addresses and ports) observed in a short interval of time. Such a record generally includes this information, the number of packets observed, the number of bytes observed, a start time and duration of the flow. Additional header information could also be collected about the flow, such as the logical-or of the TCP flags in the packets that comprise the flow (as is available in NetFlow). However, we require that whatever is collected be invariant to the application payload contents of the packets that comprise the flow (since our techniques change the payloads). Whenever it is convenient to simplify discussion, we will use flow monitoring as an example of traffic summarization.

We assume that an adversary is able to compromise a collection of computers, in such a way that the attacker's malware on each such computer can intervene in that computer's networking functions at the IP layer. That is, we presume that the attacker's malware can intercept each outbound IP datagram and modify it prior to transmission. Similarly, a compromised machine can intercept each inbound IP datagram and modify it prior to delivering it to its normal processing. On most modern operating systems, these capabilities would require a compromise of the O/S kernel.

[1] And, of course, their TCP checksums.

In this setting, the goal of SIN networking is to implement a functional overlay network among compromised machines that is unnoticeable to traffic summarization techniques. Specifically, collected traffic records must be unchanged by the presence of SIN, and consequently SIN should not alter the number or size of packets sent on the network, or the destination of any packet. Subject to this constraint, it should enable communication among compromised computers to the extent enabled by the cover traffic into which this communication must be included.

IV. A SIN Network Framework

In this section we describe a protocol framework for SIN networking. We emphasize, however, that this is one possible approach to SIN, and we believe a contribution of this paper is identifying SIN networking as a challenge for which improved approaches can be developed (and new defenses can be explored). We will populate this framework with particular routing protocols, and evaluate their performance on a variety of tasks, in §V.

We begin in §IV-A with a description of how one SINner discovers its neighbors. We discuss naming SINners in §IV-B. We present data objects, their identifiers, and SIN headers in §IV-C. Finally, we perform an evaluation of the potential for SIN capacity in a modern network in §IV-D.

A. Neighbor discovery

A "neighbor" of one SINner is another SINner to which it sends or from which it receives IP packets directly. Since our requirements for invisibility in traffic summaries requires that a SINner send packets to only destinations to which its host would already send, a SINner must discover which of those hosts are also compromised. To do so, it must piggyback discovery on those already-existing packet exchanges, in a way that does not interfere with the regular processing of those packets, in the event that the neighboring host is not a SINner.

To accomplish this, a SINner can employ any available covert storage channel, such as known channels in the IP or TCP packet header (e.g., [29], [30], [31], [32], [33], [34]). Murdoch and Lewis [33] develop a robust implementation based on TCP initial sequence numbers, for example. This technique generates initial sequence numbers distributed like those of a normal TCP implementation, but that would enable SINners knowing a shared key to recognize as a covert signal. Moreover, since discovery need not be immediate, the signal could be spread over multiple packets. Another alternative would be to exploit any available covert storage channels in application payloads, which would be undetectable in traffic summaries by definition. Covert *timing* channels (e.g., [35]) would risk detection owing to the availability of timestamps $t_1, \ldots, t_n$ to the summarization function f (see §III). They may nevertheless be effective since the information being conveyed is small (effectively one bit) and can be conveyed over the course of multiple packets.

When a SINner receives a packet in which it detects the discovery signal (or, for robustness, multiple packets from the same source bearing the signal), it adds the source IP address of the packet to a *neighbor table*. If it has not yet indicated its own participation in the SIN network to this neighbor, it takes the opportunity to do so in the next packet(s) destined for that address, i.e., by embedding the discovery signal in those packets. Since virtually all traffic elicits some form of response packet, the neighbor relation is typically symmetric: if IP address a_2 is listed in the neighbor table at the SINner with IP address a_1, then a_1 is (or soon will be) listed in the neighbor table at the SINner with address a_2.

After discovering a neighbor, the SINner can estimate its transmission capacity to this neighbor by observing the packets sent to it over time. To estimate this capacity, the SINner compresses each application payload to that neighbor (possibly in the normal course of executing a routing protocol, see §V) and collects the sizes of the vacated space in the packet. These per-packet capacities can be accumulated over whatever interval of time is appropriate, say per day, to determine an estimate of the capacity to that neighbor on each day. Each node stores these estimates for each neighbor in its neighbor table.

Each SINner augments the IP addresses and capacity estimates in its neighbor table with additional information, as described in §IV-B. §V-A and §V-B discuss how this information is used in particular routing protocols.

B. Naming

When a host is compromised, the SIN malware generates an identifier for the SINner that will permit other SINners to name it (e.g., as the destinations for objects). While the SINner could adopt another, existing identifier for the host (e.g., its IP address), we consider a different alternative here. Reusing an existing, well-known host identifier would enable any other SINner in the SIN network to potentially locate all other SINners in the network, if coupled with a link-state routing protocol as we explore in §V. While we do not incorporate robust defenses against SIN network infiltration in our design (e.g., by law enforcement) or against learning other participants in the event of such infiltrations (in contrast to membership-concealing overlays [36]), permitting even a single infiltrated SINner to learn common identifiers for all SIN participants would make SINner location just too easy. (Note, however, that we cannot prevent a SINner from knowing the IP addresses of its neighbors.)

For this reason, the identifier that a SINner generates for itself is a new random value of a fixed length. Since the SINner will transmit this identifier to others in a manner described below and since, as we will see in §IV-D, transmission capacity is at a premium, we opt for identifiers

that are not too long, specifically of length 4 bytes (B). Since identifiers are chosen at random, a 4B identifier should suffice to ensure no identifier collisions with probability at least $1-1/2^{32-2n}$ in a network with up to 2^n SINners, e.g., with probability at least 0.999 for a SIN network of up to $2^{11} = 2048$ nodes.

Once a SINner discovers a neighbor (§IV-A), it transmits its identifier to that neighbor in packets already destined to it. To do so, the sender compresses the existing application payload, and uses the vacated space to insert the identifier. A small header precedes the identifier; it simply indicates that this packet holds the sending SINner's identifier. Each sent packet is made to be exactly the same size of the original packet, which is necessary to ensure that our approach is summarization-invisible.

Upon collecting the identifier for a neighbor, the SINner inserts the identifier into the neighbor table, so that it is associated with the IP address of that neighbor.

C. Data object model

Our design of a SIN network enables the transmission of *objects* from one SINner to another. The object is assumed to begin with its length (e.g., in its first four bytes), and so is self-describing in this respect. An object is transmitted in the SIN network using a collection of point-to-point *messages*, each from one SINner u to a neighbor v. Each such message is embedded in an imminent packet from u to v. Figure 1 shows the transmission of one object in two SIN messages (i.e., separate object byte ranges in each message), each embedded into separate IP packets to a neighbor. Note that the neighbor may or may not be the ultimate destination of the object.

An object has an *identifier* that will be included in each SIN header for a message containing bytes of that object. Data objects are framed at byte granularity; i.e., arbitrary byte ranges can be sent, and so each SIN header also includes the starting byte offset and the length of the byte range being transmitted.

The object identifier for an application data object is a hash of the object contents. In this way, it can serve as both a tag to identify bytes for the same object (i.e., for reassembly), and as a checksum to detect corruptions (though this is admittedly probably unnecessary). A different type of object that can be sent is an acknowledgment for (perhaps byte ranges of) another object. The identifier for an acknowledgment for an object is a hash of: the identifier for the object it is acknowledging, appended with a constant string (e.g., "acknowledgment"). In this way, any SINner holding bytes of an object to forward can recognize an acknowledgment for byte ranges of this object, and remove any acknowledged bytes from its pending forwards.

To summarize, a SIN header for a message consists of the object identifier for the object for which bytes are being sent in the message, the starting byte offset and length of the bytes in the message, and the source and destination identifiers for the object. Since, as will be shown in §IV-D, SIN capacity is at a premium, we want to reduce the header size as much as is reasonable. The SIN header for an IP packet that we employ in the rest of our evaluation is 18B in length: it includes a 4B object identifier for the object of which the SIN payload in this packet is a part; the byte offset (4B) in the object and length (2B) of the SIN bytes in this packet; the 4B identifier for the source SINner of the object, and the 4B identifier of the destination SINner for this object. (See Figure 2.)

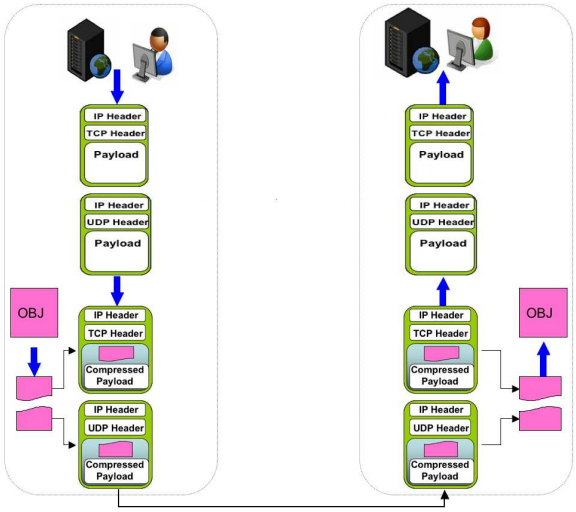


Figure 1. Transmission of an object in two SIN messages

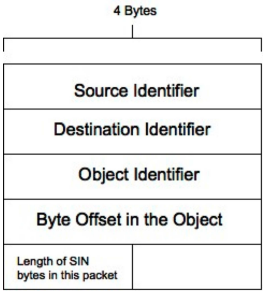


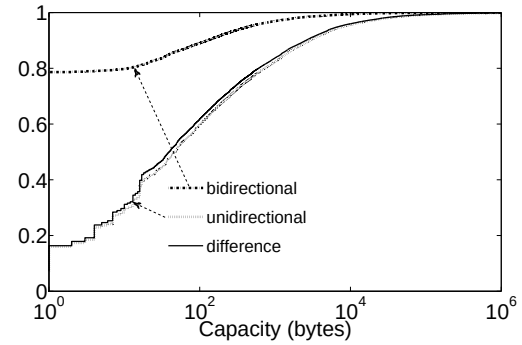
Figure 2. SIN header format in experiments (object length is contained in the first 4 bytes of the object itself)

D. Available Capacity

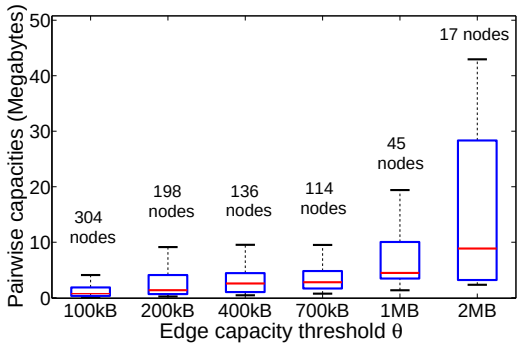
There is reason to be skeptical of the capacity that is offered by compressing packet payloads. Traffic that follows a client-server pattern poses a challenge to two-way covert communication via this technique, since typically only flows in one direction (e.g., downloads from a web server, versus requests to the web server) are sufficiently large offer the possibility of substantial residual capacity after compression.

Peer-to-peer traffic, on the other hand, does not suffer from this limitation, but since it usually consists of media files that are already compressed, its packets might not be very compressible.

In order to understand the capacity offered by modern networks via this technique, we logged traffic on the network at our organization, in particular recording the size to which each packet payload could be compressed using the `zlib` library. We collected a dataset, referred to as Dataset1 in this paper, over several weeks.[2] Only packets that could be compressed were represented in Dataset1, as other packets are useless for our purposes. In particular, this discarded all encrypted packets, despite the fact that SIN capacity might be available in the plaintext protocol (and could be utilized by SINners at the encryption/decryption endpoints).



(a) Direct (all nodes, CDF)



(b) Max-flow (selected nodes)

Figure 3. Pairwise SIN capacities per day in Dataset1.

We used Dataset1 to build a graph consisting of vertices that represent hosts in the network, and directed edges labeled by the average daily capacity in Dataset1 from the source host to the destination host, i.e., capacity available for covert payload due to compressing packets. Figure 3(a) shows the distribution of these capacities, both unidirectionally (i.e., per edge) and bidirectionally (i.e., the smaller of the capacities of the opposing edges between two nodes). Also shown in Figure 3(a) is the distribution of the difference between the capacities of the opposing edges between each pair of nodes. As the difference plot in Figure 3(a) shows, the difference distribution is very close to the unidirectional distribution, suggesting that our intuition regarding the constraining effects of common client-server patterns on two-way communication holds true.

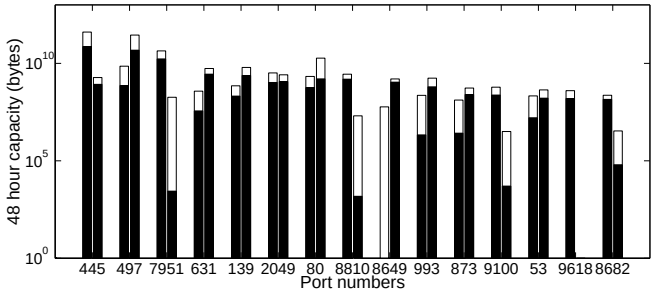


Figure 4. Server ports that contributed the most SIN capacity over 14:56 Thu Sep 3 – 14:06 Sat Sep 5; left bar is inbound traffic, right bar is outbound; bar height denotes total bytes, black area denotes SIN capacity

We have also evaluated what applications contribute the most available SIN capacity. To do so, we recorded additional traces and found all ⟨IP address, port⟩ pairs that acted as servers in our traces, in the sense of accepting initial SYN packets in a three-way TCP handshake that went on to complete. For each pair, we calculated the payload sizes of all inbound packets, the sizes of those payloads compressed (individually), the payload sizes of all outbound packets, and the sizes of those payloads compressed. By summing each of these four categories of values over all ⟨IP address, port⟩ pairs with the same port value, we gain an understanding of how much available capacity each server port contributes. The 15 ports contributing the most available SIN capacity, summed over both inbound and outbound directions, are shown in log scale in Figure 4. This figure also confirms the asymmetry of SIN capacities, in that for most of these ports, there is at least an order of magnitude difference between the SIN capacities in the inbound and outbound directions.

Despite the limitations introduced by client-server communication patterns, the possibility of substantial overall capacity remains for SIN networking since it need not rely on bidirectional point-to-point communication. To illustrate this, for a given capacity threshold θ, we deleted all edges of capacity less than θ in the above graph, and then computed the largest strongly connected component in the graph that remained. Then, we re-inserted all edges between nodes in that strongly connected component, and computed the

[2]Due to artifacts of our collection infrastructure, our traces were typically not contiguous. Dataset1 contains data collected during the intervals Mon Jun 22 15:19 – Wed Jun 24 00:51, Thu Jun 25 20:43 – Sun Jun 28 12:38, Tue Jun 30 19:07 – Wed Jul 1 04:55, Thu Jul 2 09:37 – Sat Jul 4 17:18, Mon Jul 6 19:07 – Tue Jul 7 15:22, Thu Jul 9 04:11 – Sat Jul 11 11:56, and Mon Jul 13 09:50 – Tue Jul 14 11:02, all in 2009. Note that Dataset1 was collected during the summer, the time of lowest network utilization on a university network.

maximum flow [37] sizes in this graph for each ordered pair of nodes, i.e., in which the first node is the "source" and the second node is the "sink". Intuitively, if exactly the nodes in the strongly connected component were SINners, then these maximum flow sizes are theoretically the SIN capacity from one SINner to another.

Figure 3(b) shows the distribution of those maximum flow sizes (over all ordered pairs of nodes in the strongly connected component) for different levels of θ, each represented as a box-and-whiskers plot. The box illustrates the 25th, 50th, and 75th percentiles; the whiskers cover points within 1.5 times the interquartile range. In addition, above each box-and-whiskers plot is the number of nodes in the strongly connected component for that value of θ. As this graph shows, the maximum flow sizes often far exceed the threshold θ, implying that there is much capacity to be gained by routing over multiple paths between pairs of nodes. The extent to which this extra capacity can be utilized by an intelligent a routing protocol is discussed in §V.

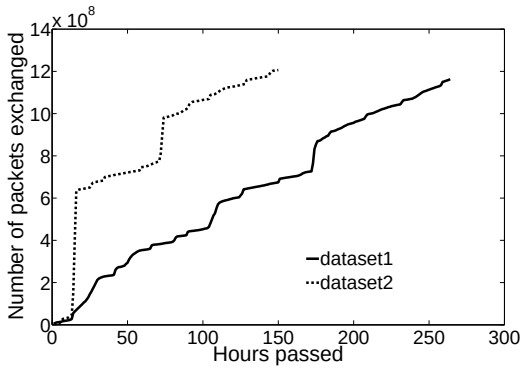


Figure 5. Cumulative packets over time in Dataset1 and Dataset2 between nodes in 114-node component

For the experiments in §V, we will focus on the 114-node connected component determined by $\theta = 700$kB/day, for which the distribution of maximum achievable capacities between nodes is shown in Figure 3(b). We caution the reader that despite the significant capacities illustrated by this distribution (e.g., some pairs with capacity of 10MB/day), one cannot conclude that the latencies of communication between any two of these nodes should be small. The capacity realized by any particular routing protocol might be far less. Moreover, the number of hops between the two nodes might be large, and other workload effects, notably congestion (as we will see), can substantially increase object transmission latencies.

As many of our experiments in subsequent sections will use this 114-node component, in Figure 5 we plot the packet volume of Dataset1 over time, restricted to packets between nodes in this component. A second dataset, Dataset2, is also shown, restricted to packets between nodes in this component; this dataset was recorded after Dataset1 [3] and also includes only compressible packets. So as to adjust for packet volume differences in these datasets, we will often plot results as a function of packets processed from the trace. Figure 5 enables one to translate these results to real time.

V. Evaluation of SIN for Attacker Workloads

The available SIN capacity demonstrated in §IV-D is practically a threat only to the extent that it can be realized by actual routing protocols, and for tasks that an adversary might wish to perform. In this section, therefore, we instantiate the framework described in §IV with a state-of-the-art delay-tolerant routing protocol, namely the Delay Tolerant Link State Routing (DTLSR) protocol [10], [38]. We also use trace-driven simulations to evaluate the performance it achieves in tasks that we believe would be characteristic of activities an attacker might want to perform using a SIN network.

Very briefly, DTLSR is a modification of classical link-state routing to DTNs. As is typical for link-state protocols, DTLSR floods link-state updates (in our terminology, neighbor-table broadcasts) through the network. Each node uses these updates to maintain a graph representing its current view of the network topology, and uses a modified shortest path algorithm to find the best path to the destination. Unlike traditional link-state protocols, however, the link-state information conveyed in DTLSR is used to predict when links should become available and with what capacities (versus to remove unavailable links). Our choice of DTLSR for our evaluation derives from our conjecture that it is well-suited to SIN networking, where we generally expect the connectivity in the cover network (and thus in the SIN network) to be relatively predictable on the basis of history. If true, then neighbor-table broadcasts, which include historically derived capacity estimates, should be useful for making routing decisions. We will briefly evaluate this conjecture later.

A. Flooding and Neighbor Table Broadcast

The neighbor tables described in §IV-A–§IV-B support a primitive form of broadcast communication in the SIN network, i.e., by flooding. For any given broadcast, each SINner holds (i) the byte ranges (and byte values) of the broadcasted object it has received — in the case of the broadcast sender, this is just the entire object — and (ii) for each of its neighbors, the byte ranges of the broadcasted object that its neighbor should already have, because it previously either sent those bytes to or received those bytes from that neighbor. When an IP packet destined to a neighbor is imminent, the application payload is compressed and the next available bytes of the broadcasted object (not

[3] Specifically, Dataset2 consists of records collected during 22:56 Wed Jul 15 – 22:28 Thu Jul 16; 18:04 Sat Jul 18 – 03:22 Tue Jul 21; and 05:12 Thu Jul 23 – 02:28 Sun Jul 26.

already possessed by the neighbor) are included in the vacated space, preceded by the header described in §IV-C with the destination field set to a particular value designating this as a broadcast. As always, any packet is always ensured to be of the same length as the original packet, by padding if necessary.

Neighbor-table broadcast In DTLSR, a key application of flooding is to propagate the neighbor tables themselves to all SINners. This is done by flooding each SINner's neighbor table periodically; we call this a *neighbor-table broadcast*. The SINner's neighbor table consists of SIN capacity estimates to each SIN neighbor per *epoch*, where an epoch is a specified time interval (e.g., 8-9am on weekdays, or on Mondays specifically) and the capacity estimate is derived from historical data for previous instances of that interval. Neighbor tables include epoch capacity updates only for those that have changed significantly from the estimates previously conveyed for them. Initially, before capacity estimates to a neighbor are determined, a SINner simply includes the neighbor's SIN identifier in its next neighbor table update.

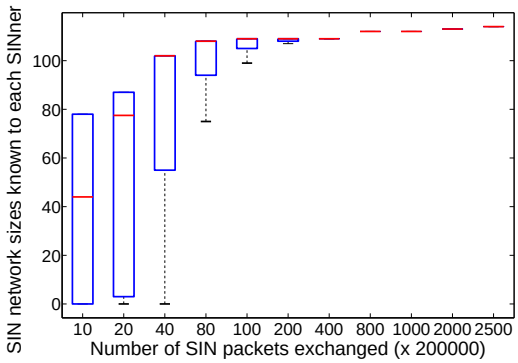


Figure 6. Neighbor-table broadcast latency, 114 SINners (Dataset1)

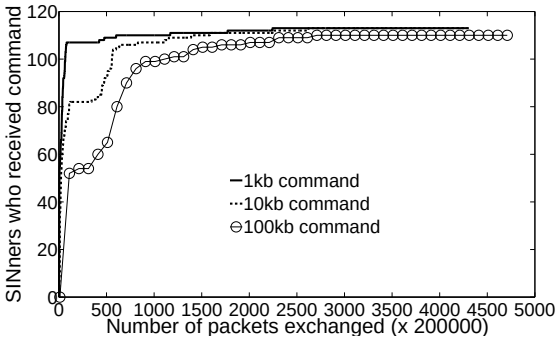
We stress that the IP addresses of a SINner's neighbors are *not* included in this broadcast; only their SIN identifiers and capacity estimates are included. Excluding these IP addresses is motivated by the desire to not disclose the IP addresses of all participants to each SINner, so as to make it more difficult for an infiltrator to passively discover all SINners. Alternatively, IP addresses could be included, so as to short-circuit the discovery process in some cases, but here we employ the more conservative approach.

In Figure 6, we demonstrate the progress of neighbor-table broadcasts performed among the same 114 SINners identified in §IV-D and using Dataset1 restricted to these SINners as the cover traffic. This figure shows progress as a function of the number of packets processed from the trace. The neighbor tables in this test included a capacity estimate per hour of each weekday for each neighbor, i.e., 168 estimates per neighbor. This box-plot shows the distribution of the number of SINners of which each SINner is aware, assuming that each SINner "awakens" at the beginning of the trace and initiates its neighbor-table broadcast. As this graph shows, after about 8×10^6 packets are exchanged among those 114 SINners— or about two hours if translated by using Figure 5 — the median number of SINners about which a SINner knows is over 100.

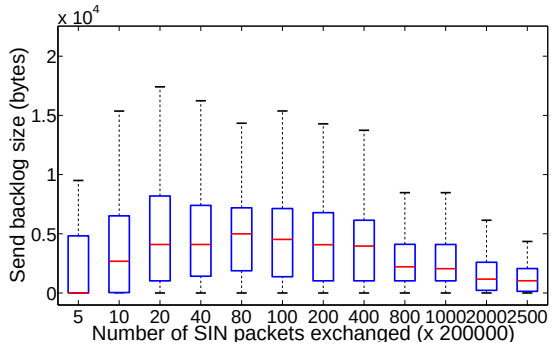
There is, however, a "long tail", with a few SINners becoming known to the whole SIN network much later than others. There are several reasons for this. First, there are several SINners who didn't become active until a day or so in the trace, resulting in a significant delay before their neighbor-table broadcasts really began. The second reason is the asymmetric bandwidth of outbound and inbound directions for some SINners. The timeliness of neighbor-table broadcast depends heavily on the outbound capacity of a SINner. Since some SINners exhibit lower outbound capacities versus their inbound capacities, it takes each one longer to send out a neighbor table, thus lengthening the time required for others to learn of its neighbors.

Application broadcasts The above flooding mechanism could also be used by applications to disseminate a broadcast message, and so we pause here to shed light on the performance of such broadcasts as a function of their size. Consider, for example, a workload in which we view the SINners as bots under control of a bot-master. We consider the latency required for a bot-master command to propagate from one SINner inside the network (presumably, the first to receive it) to the remainder of the SINners. We select the initiating SINner of the broadcast at random, and consider the cases of a 1kB, 10kB, and 100kB broadcasts.

The command broadcast latency is shown in Figure 7(a). These tests used the same 114 nodes identified in §IV-D, and after neighbor-table broadcast disseminated neighbor tables. Each point on a curve in Figure 7(a) depicts the number of SINners who received the complete command throughout the trace-driven simulation. As the figure shows, in the 1kB case, more than 100 SINners learned the command very quickly. The 10kB and 100kB commands take much longer to broadcast because of their larger size. The slower propagation to the last SINners may represent topological effects of the SIN network, in that nodes farther away from the bot master take more time to receive the complete command than those who are closer. It may also be partially explained by node "congestion" that flooding induces, which is represented in Figure 7(b). More specifically, Figure 7(b) shows the distribution on SINners' *send backlog* over time, i.e., the sum of the sizes of all messages that the SINner has pending to send. As Figure 7(b) shows, backlogs swell early and then slowly dissipate. Perhaps surprisingly, some nodes continue to have forwarding obligations some 100 hours later, owing to a lack of traffic on which to piggyback the bytes they need to forward. This suggests that some form of broadcast expiration would be appropriate.



(a) 1kB, 10kB and 100kB broadcast latency



(b) SINner send buffers over time (1kB broadcast)

Figure 7. Command broadcast to 114 SINners (Dataset1)

B. Unicast routing

As in a broadcast, bytes of a unicasted object can be inserted into the spaces vacated by compressing application payloads of IP packets. In each such packet, the object bytes are preceded by the header described in §IV-C. Upon receiving bytes of a unicast object (as an intermediate SINner on the path), the SINner buffers these bytes for forwarding.

We implemented DTLSR routing to use the expected delay of message delivery as the objective to minimize. Each SINner models the network as a graph for each hour of each day of the week (e.g., 1-2pm on Mondays). (This granularity is the default, though we will evaluate other granularities of network modeling in selected tests, as well.) That graph is directed and weighted, where the weight on the edge from SINner u to SINner v represents the expected capacity directly from u to v in the hour the graph represents, based on historical activity in the same hour on the same day of the week. Upon receiving s bytes of an object, u computes the path yielding the minimum expected delay for these s bytes to reach their destination, via a modified Dijkstra's algorithm that builds a shortest path tree from u iteratively. Specifically, v is added to the tree when the expected delay to reach it from any w already in the tree is the minimum among SINners not yet in the tree, where this expected delay is the ratio of s and the estimated capacity from w to v in the hour at which the s bytes would arrive at w (in expectation) traveling along the existing tree. These s bytes are then queued for transmission to the first SINner on that shortest path.

In order to evaluate the performance of DTLSR for unicast routing in a SIN network, we postulate various workloads that might be characteristic of activities that an attacker might desire to perform. The workload scenarios that we consider include point-to-point messaging, bot "response" to a central controller SINner (e.g., a "bot-master" as evaluated in §V-A) and data exfiltration to a drop site.

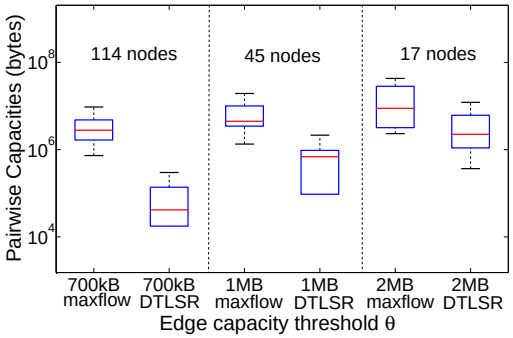
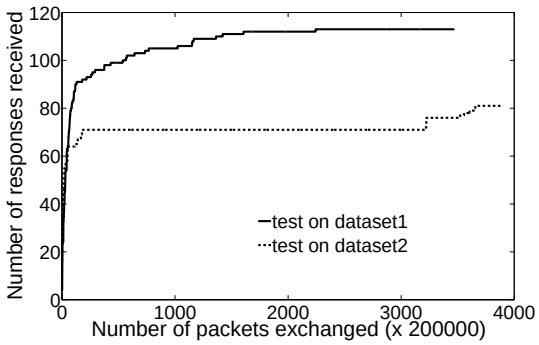


Figure 8. Realized pairwise daily SIN capacities (Dataset1)

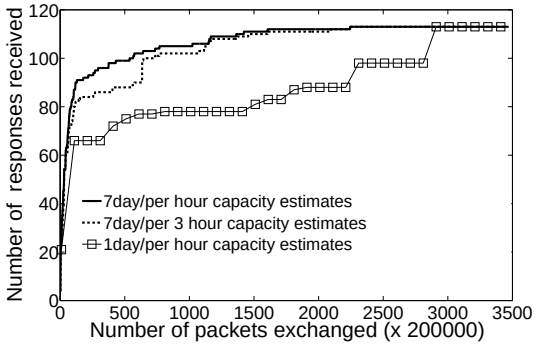
Pairwise capacities The first test we perform using the DTLSR unicast protocol is to examine to what extent it can achieve the pairwise potential capacities shown in Figure 3(b). To do so, we computed a trace-driven simulation of DTLSR per pair of SINners in some of the connected components depicted in that figure. We focused on the smaller components since the number of needed simulations grows quadratically in the number of SINners in the component. So as to perform a fair comparison with Figure 3(b), in these tests we used Dataset1, though restricted to only to its first two days of traffic, to reduce the time consumed per pair of SINners.

The results of this test are shown in Figure 8. This figure includes both the distribution of potential (maxflow) capacities taken from Figure 3(b) and the distribution of realized SIN capacity using DTLSR. This figure shows that the average daily capacities realized by DTLSR are roughly at least an order of magnitude lower than those shown in Figure 3(b). The reasons for this are (at least) two-fold: First, in DTLSR, a SINner forwards bytes for a particular destination to only one of its neighbors in any epoch. (In contrast, a maximum flow is calculated utilizing all links neighboring each node.) Second, due to cover traffic packets departing a SINner prior to SIN bytes reaching that SINner, many opportunities for transmitting SIN data are missed.

Bot response We next consider an attacker workload in which all SINners initiate messages ("responses") simultaneously to one SINner, which might be responses to a



(a) Latencies in Dataset1 and Dataset2 tests



(b) Latencies in Dataset1 tests with various capacity estimation granularities

Figure 9. 1kB response (to broadcast) statistics

broadcast sent by that SINner (e.g., as measured in Figure 7). In our experiment, we use the 114-SINner component from §IV-D, and each SINner sends a 1kB object unicast. We consider two tests, one in which the cover traffic is Dataset1, and one in which Dataset2 is the cover traffic. In the second test, there are differences between the capacity estimates sent in the neighbor tables (computed on the basis of Dataset1) and the capacities available during unicast routing (dictated by Dataset2).

As shown in Figure 9(a), there is a significant difference between performance using Dataset1 and Dataset2. While this is partially due to differences between capacity estimates from Dataset1 and available capacities with Dataset2, a closer look reveals that the massive spike in Dataset2 packet volume in hours 14–16 (see Figure 5), occurs between only two nodes and in a way that is not particularly useful for routing to the chosen destination SINner in this test. As such, this packet burst elongates the Dataset2 curve in Figure 9(a) to the right without facilitating its progress.

Another observation from this figure is that the response latencies generally exceed the latencies of the bot command shown in Figure 7(a) even in the Dataset1 test, and while the bot-master receives responses from over 90 bots in the first 20 hours, the responses come slowly after that. Figure 9(a) hides the fact that in many of these cases, a substantial fraction of the 1kB unicast does arrive at the bot-master in this time. Nevertheless, this latency is much larger, which is at least partially because the DTLSR unicast protocol does not duplicate the message in the network like flooding does, and so is not as aggressive in pushing SIN messages forward. In addition, congestion forms at certain nodes "close to" the destination node, resulting in delays.

To highlight another consideration in determining sending performance, we show in Figure 9(b) the impact of conveying capacity estimates of different granularities in neighbor tables, using the Dataset1 test. As shown in that figure, the best messaging latencies were obtained when capacity estimates were computed and conveyed per hour of each weekday (the default in this paper), though estimates per 3-hour period of each weekday resulted in nearly the same performance. The worst results occur when computing estimates per hour but averaged over all days of the week. This demonstrates a tradeoff between the size of neighbor tables and the resulting performance of the unicast protocol.

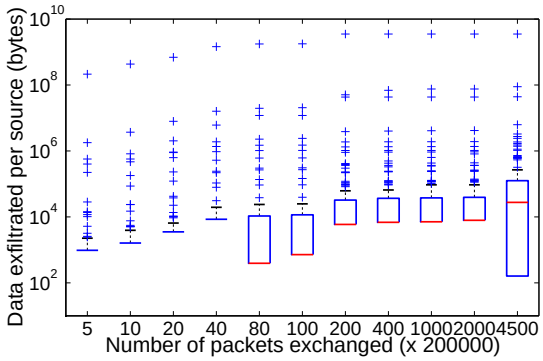


Figure 10. Data delivered to drop site per sender (Dataset2)

Data exfiltration The third attacker workload that we consider is one in which the attackers "stream" as much data as possible to a single SINner by repeatedly performing 10kB unicasts. This workload might reflect a scenario in which the SINners are exfiltrating data covertly from an infected organization, using the unicast destination as a drop site. This test used the same 114-SINner component, and the destination was the same SINner as was chosen as the bot-master in the previous workloads.

In this experiment we witnessed an average throughput of 52.43MB per hour for the first 70 hours to the destination. That said, the amount of data received from each sender varies dramatically. Figure 10 shows the log scale distribution of the amount of data received per sender as a function of the packets exchanged among SINners. As this chart shows, some senders fare far better than others in terms of getting their data to the drop site, and most data came from one outlier that had very large bandwidth to the drop site. This is primarily due to some senders being in

advantageous positions relative to others in the topology, but the fact that some SINners have much smaller outbound capacities than others also contributes to this disparity. (Note that since the medians for the first four boxes are zero, they are not displayable on the log scale. So, the first four boxes show only the 75th percentile.)

VI. Detection of SIN Networks

Exploration of SIN networks naturally raises the question as to what can be done to defend against them. Our design in §IV presents several avenues for detecting a SIN network. For example, one approach would be to try to exploit the covert discovery signal discussed in §IV-A: a host could be set up to passively inspect received packets for the signal, or to actively probe hosts with candidate such signals in the hopes of eliciting a response. In either case, the other communication endpoint would be detected as a SINner. This approach, however, presupposes knowledge of what the candidate signals are, and in this respect is a form of signature-based intrusion detection. A new variant of SIN malware could employ a new signal that would go undetected until discovered and reverse-engineered.

Below we report preliminary results on two more general approaches to attempt to detect SIN networks, and specifically that try to detect the compression of packets. As such, even if these approaches detect SIN networks as explored in this paper, they may not detect SIN networks built from primitive covert communication channels of other types (e.g., timing channels).

A. Using Packet Processing Delays

No matter what SIN routing algorithm is employed, a basic property of SIN networking as we have explored it here is compression of packet payloads to insert SIN data, and corresponding decompression to reconstitute the original packet payload prior to delivery. This compression and decompression adds processing delays to the network stacks of the SINners between which the communication occurs, and so one avenue of detection might be to try to notice these extra delays.

To measure the capability of this detection approach, we performed an experiment that a defender might utilize to detect a SINner, namely by monitoring the delay apparently associated with a node's packet processing. Consider, for example, a network defender that records *a-b-t* records [1] on his network; unlike most payload-agnostic summarization approaches, these records include packet timings. Suppose the defender builds a model of the typical responsiveness of a server to queries, and monitors for a change in the responsiveness that might be indicative of the extra delays associated with decompression of each query and compression of each response. We would like to determine whether such a detector would be effective.

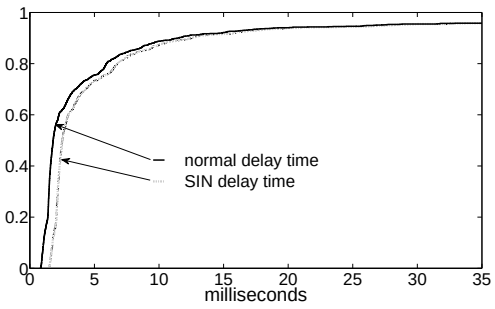


Figure 11. Responsiveness of our organization's main web server with and without request decompression and response compression

To test this, we measured the responsiveness of our organization's main web server as we used `wget` to retrieve the entire contents of that site, i.e., by requesting each URL, and timing the delay to receive the first packet of each server's response. These requests were issued from a host on the campus wired network. We then added to each of the timings the costs of decompressing the payload of the corresponding request packet (after it had been compressed, as if the server were a SIN node decompressing it) and compressing the payload of the corresponding first response packet (as if the server did so). To ensure that our tests were conservative, we did these tests on a campus holiday (September 7, 2009, which was Labor Day in the U.S.) so that the network and server were lightly loaded; we used an inferior machine (a 2.2GHz MacBook Pro laptop, versus the 2.4GHz dual-core AMD Opterons that support the web server) for calculating the compression and decompression costs; and we measured the most aggressive compression supported by `zlib`. The CDFs of the resulting timings (with and without compression and decompression overheads) are shown in Figure 11. The similarities of these CDFs suggest that detecting the web server as a SIN node using server responsiveness would be unreliable.

B. Using Payload Analysis

While SIN networking is premised on the notion that network defenders collect only summary information about their networks, it is worthwhile to consider what degree of payload inspection is useful to detect it. For example, the network monitoring could scan in a signature-based way, e.g., for something that might look like our SIN header in application payloads. However, due to the random selection of identifiers and since object identifiers would typically be hash values, there is perhaps less structure in our header to exploit in this way than it first appears. Moreover, a change of header format or location would also likely sidestep this detection.

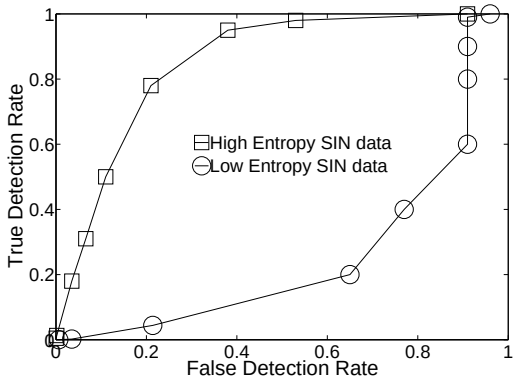
Instead, in this section we consider ways of analyzing packet contents to detect a change in the byte-value distribution for a particular application's communication. Our

compression of application payloads and insertion of SIN data risks changing the byte frequency distribution, and so it would seemingly be detectable by mechanisms that monitor this distribution [22], [24], [26]. In order to test this, we built a detector similar to PayL [24], an intrusion-detection system that detects attempts to infect a server by monitoring byte-value n-grams in server query packets. Instead of limiting our attention to server queries, however, we considered building byte-value n-gram distributions for query packets and response packets independently, to see if the insertion of SIN payloads within those packets would alter the byte-value n-gram distributions of either type of packets in a detectable way (without inducing a large number of false alarms).

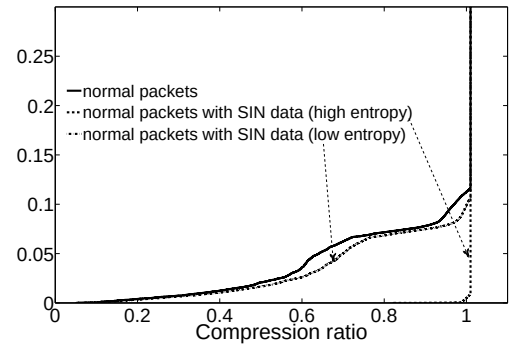
DNS traffic To provide a test of this, we utilized a DNS trace collected over two weeks[4] from a university wireless network used by approximately 50 clients. This dataset comprises approximately 3 million packets amounting to 450MB of data. We separated this dataset into request and response packets, and treated these separately. For each type (request or response) and each test, we used a nonoverlapping 33% of these response packets as testing data and the remaining 66% for training. In each test, we built a byte-value 1-gram distribution of our training packet payloads. Then, for each testing packet, we built a 1-gram distribution for that packet, computed the L1 distance between the trained distribution and this packet's distribution, and then raised an alarm if the L1 distance exceeded a threshold. We did this test for each testing packet in its original form, each testing packet compressed with low-entropy SIN payload added (all zeroes), and each testing packet compressed with high-entropy SIN payload added (random bytes). In these tests, we found an equal error rate of zero — the SIN packets were perfectly identifiable using the 1-gram detector — because DNS is a protocol in which the bytes of many fields typically range over a relatively restricted subset of values. Even bytes of domain names can include only letters, digits between "0" and "9", and "-". For this reason, it is likely that compressing a DNS packet will introduce byte values that would otherwise be highly uncommon, resulting in a large L1 distance between a SIN packet and the distribution for normal DNS packets.

Web responses To illustrate a type of packet in which SIN data is more difficult to detect, we turned to web traffic, as we conjectured that the richer content and media types available on web servers would make it much more difficult to identify individual SIN packets. For web sites hosting a variety of media formats (videos, images, text), the byte-value n-gram distribution calculated over all response packets would generally differ too much from that of individual packets (which typically include content of only one media type) to make it a useful detector. Instead, for web server responses we borrowed an approach used in Anagram [39], in which we simply record which n-grams occurred in the training data for that site, rather than tracking a distribution for them.

To conduct this test, we used `wget` to retrieve the contents of our organization's main web server, and built training and testing data from server response packets. The results reported below are the average of five tests, each using a nonoverlapping 20% of these response packets as testing data and the remaining 80% for training. Our decision to use 80% for training (vs. 100%) reflects two observations. First, training a network-based SIN detector, even one with a different model per server IP address and port, on the entire contents of every server that it witnesses, would likely be untenable; more likely, it would train on what is actually retrieved from each site during the training period, which would typically be less than the entirety of the site. And second, for a site that is more frequently updated than our organization's, there will be a gap between the current content of the site and the content on which the detector was trained.



(a) ROC curve for 4-gram detector



(b) CDF of compression test

Figure 12. Detection of SIN packets from our organization's main web server

For testing, we determined the number of n-grams in a

[4]This trace was collected in 2007 between Sat Aug 11 11:16 and Sun Aug 26 20:48.

response packet that was not seen during training, and raised an alarm on that packet if a threshold number of such n-grams had not been seen in training. Typically all possible 1-grams, 2-grams and 3-grams were observed in training, rendering the model useless, and so we conducted tests using 4-grams. We tested on both response packets from our organization's web server drawn from our corpus as described above, and on these packets after SIN contents were added. For the latter, in one test we embedded low-entropy SIN data (all zeroes) and in one test we embedded high-entropy SIN data (random bytes). The ROC curves for these tests are shown in Figure 12(a). It is evident from this plot that SIN detection based on 4-gram detection was quite poor when tested on high-entropy SIN data — the equal error rate is approximately 20% — and is abysmal when tested on low-entropy SIN data.

The failure of our 4-gram detector suggests perhaps trying 5-grams, as was done in Anagram [39]. Anagram, however, recorded 5-grams from *requests* to an academic web server; since such requests are far less varied in content type than web responses, it was feasible to record the 5-grams witnessed in requests. For web responses, we project that well more than 100GB would be needed to record the 5-grams witnessed, which would require on-disk storage for any computer available to us. Such an architecture would preclude monitoring efficiently, however. A second deterrent is that any benefit gained from using higher n-grams to detect SIN data would presumably trigger false positives on any new high-entropy data object not seen during training. As such, it is unclear whether using n-gram analysis would effectively detect SIN communication for servers with dynamic content.

As such, we turned to another natural idea for detecting SIN traffic in web-server responses: a SIN network presumably is required to increase the entropy of the application payloads on which it piggybacks, since packets carry both the information of the original application and the SIN information. We are unaware of research on evaluating application payload entropies at line speeds,[5] but a pragmatic alternative is to monitor the compression ratios of payloads. To explore this avenue, we repeated the above tests, except calculating the compression ratios of the packet payloads from our organization's web server, both without and with SIN data embedded (and in the latter case, for both low-entropy and high-entropy SIN data). Since we do not believe a per-packet detector is feasible for this measure — e.g., a single video response packet will be uncompressible — we plot the CDF of the compression ratios to gain insight into whether a monitor that computes an aggregate compression ratio might work. (Because our dataset does not reflect real browsing, it is difficult to project what sort of aggregate measure might work, however.) The results, shown in Figure 12(b), suggest that a web server that sends high-entropy SIN data might be detectable, though it is more questionable whether a server sending low-entropy SIN data would be. This presents a tradeoff to the adversary between conveying information as compactly as possible and keeping its traffic as undetectable as possible. We plan to further investigate this tradeoff in future work.

[5]In contrast, entropies of header fields (which our approach does not affect) have been used to monitor for anomalies (e.g., [40], [41], [42]) and streaming algorithms have been proposed for estimating these entropies (e.g., [43], [44], [45]).

VII. Conclusion

In this paper we introduced Summary-Invisible Networking (SIN), a type of covert networking that strives to remain invisible to anomaly detection systems that examine traffic summaries. To be undetectable by such detectors, we set a stringent requirement for a SIN network, namely that it leave unchanged the results of any network monitoring that is oblivious to the application payload content. Consequently, a SIN network is not permitted to initiate new packets or even alter the timing or size of additional packets. To accomplish this while implementing a functional network, SIN builds from two key observations. First, network anomaly detection based on payload-oblivious traffic summaries admits a new type of covert embedding in which compromised nodes embed content in the space vacated by compressing the payloads of packets already in transit between them. Second, point-to-point covert channels can serve as a "data link layer" over which routing protocols can be run, enabling more functional covert networking than previously explored.

We presented a framework for SIN networks and showed the potentially achievable networking capacity that this framework permits (§IV). We then considered instantiating this framework with particular routing protocols and evaluated these protocols on tasks suggestive of what an adversary might like to attempt with a SIN network (§V). This study demonstrates that while latencies and throughputs on a SIN network are orders of magnitude worse than existing networks today, the network is adequately functional that a sufficiently patient attacker might find the covertness of this approach worth the performance price. We believe our work opens up new research directions in improving the performance of SIN networks.

We also examined approaches to detect SIN networks (§VI). We first tested the possibility of using the timing perturbations resulting from SIN processing at SINners, and found this to be unreliable. Second, we considered efficient means of payload analysis to detect SIN packets, with mixed results. For a protocol like DNS, we found that monitoring byte-value n-gram distributions of packets can effectively detect SIN packets, but neither this nor monitoring compression rates of packets produced compelling results for detecting a web server sending SIN payloads. As such, we believe there is much more to investigate in how to detect SIN networking in general.

Finally, as an initial paper in SIN network, this paper introduces at least two additional directions for subsequent research. First, the SIN network protocols we proposed will not scale to a very large SIN network. Scaling these systems might involve the creation of an inter-SIN-domain routing protocol, i.e., an analog to BGP for our SIN networks. Scale may also bring the need for the attacker to better monitor the viability of his SIN network, along with analogs of other challenges of network operations. Second, we have taken an extreme position in stating the requirements of a SIN network, namely that it cannot change traffic summaries at all. Even small relaxations of this requirement may provide substantial benefits for the SIN network, though any such relaxation must be carefully considered to determine whether it opens a practical method of detection.

Acknowledgements

We are grateful to Alex Everett, James Gogan, Fabian Monrose, and Sid Stafford for their assistance in collecting the network traffic traces utilized in this research. We are also grateful to John McHugh and Nikola Vouk for helpful discussions on this research. This work was supported in part by NSF awards CT-0756998 and CT-0831245.

REFERENCES

[1] F. Hernández-Campos, A. B. Nobel, F. D. Smith, and K. Jeffay, "Understanding patterns of TCP connection usage with statistical clustering," in *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Sep. 2005, pp. 35–44.

[2] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "BLINC: Multilevel traffic classification in the dark," in *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Aug. 2005.

[3] J. Terrell, K. Jeffay, F. D. Smith, J. Gogan, and J. Keller, "Exposing server performance to network managers through passive network measurements," in *2008 IEEE Internet Network Management Workshop*, Oct. 2008, pp. 1–6.

[4] M. P. Collins and M. K. Reiter, "Finding peer-to-peer file-sharing using coarse network behaviors," in *Computer Security – ESORICS 2006: 11th European Symposium on Research in Computer Security*, Sep. 2006, pp. 1–17.

[5] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagl, K. Levitt, C. Wee, R. Yip, and D. Zerkle, "GrIDS – a graph based intrusion detection system for large networks," in *19th National Information Systems Security Conference*, 1996, pp. 361–370.

[6] Y. Gao, Y. Zhao, R. Schweller, S. Venkataraman, Y. Chen, D. Song, and M.-Y. Kao, "Detecting stealthy attacks using online histograms," in *15th IEEE International Workshop on Quality of Service*, Jun. 2007.

[7] M. P. Collins and M. K. Reiter, "Hit-list worm detection and bot identification in large networks using protocol graphs," in *Recent Advances in Intrusion Detection, 10th International Symposium, RAID 2007*, 2007, pp. 276–295.

[8] Y. Xie, V. Sekar, D. Maltz, M. K. Reiter, and H. Zhang, "Worm origin identification using random moonwalks," in *2005 IEEE Symposium on Security and Privacy*, May 2005, pp. 242–256.

[9] K. Fall, "A delay-tolerant network architecture for challenged internets," in *2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2003, pp. 27–34.

[10] S. Jain, K. Fall, and R. Patra, "Routing in a delay tolerant network," in *2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2004, pp. 145–158.

[11] R. Anderson, R. Needham, and A. Shamir, "The steganographic file system," in *Information Hiding, Second International Workshop, IH'98*, 1998, pp. 73–82.

[12] A. Baliga, J. Kilian, and L. Iftode, "A web based covert file system," in *11th Workshop on Hot Topics in Operating Systems*, 2007.

[13] A. Baliga and J. Kilian, "On covert collaboration," in *9th Workshop on Multimedia & Security*, 2007, pp. 25–34.

[14] X. Luo, E. W. W. Chan, and R. K. C. Chang, "Crafting web counters into covert channels," in *IFIP International Federation for Information Processing, Volume 232, New Approaches for Security, Privacy and Trust in Complex Environments*, 2007, pp. 337–348.

[15] M. Shin, S. Hong, and I. Rhee, "DTN routing strategies using optimal search patterns," in *ACM MobiCom Workshop on Challenged Networks*, Sep. 2008.

[16] V. Erramilli and M. Crovella, "Forwarding in opportunistic networks under resource constraints," in *ACM MobiCom Workhop on Challenged Networks*, Sep. 2008.

[17] R. Ramanathan, R. Hansen, P. Basu, R. Rosales-Hain, and R. Krishnan, "Prioritized epidemic routing for opportunistic networks," in *1st International MobiSys Workshop on Mobile Opportunistic Networking*, Jun. 2007.

[18] A. Vadhat and D. Becker, "Epidemic routing for partially connected ad hoc networks," Department of Computer Science, Duke University, Tech. Rep. CS-200006, 2000.

[19] M. P. Collins and M. K. Reiter, "On the limits of payload-oblivious network attack detection," in *Recent Advances in Intrusion Detection, 11th International Symposium, RAID 2008*, Sep. 2008, pp. 251–270.

[20] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan, "Fast portscan detection using sequential hypothesis testing," in *2004 IEEE Symposium on Security and Privacy*, May 2004.

[21] A. Lakhina, M. Crovella, and C. Diot, "Mining anomalies using traffic feature distributions," in *2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2005, pp. 217–228.

[22] C. Kruegel, T. Toth, and E. Kirda, "Service specific anomaly detection for network intrusion detection," in *Symposium on Applied Computing*, Mar. 2002.

[23] H. A. Kim and B. Karp, "Autograph: Toward automatic distributed worm signature generation," in *13th USENIX Security Symposium*, Aug. 2004.

[24] K. Wang and S. J. Stolfo, "Anomalous payload-based network intrusion detection," in *Recent Advances in Intrusion Detection, 7th International Symposium, RAID 2004*, 2004, pp. 203–222.

[25] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated worm fingerprinting," in *6th USENIX Symposium on Operating System Design and Implementation*, Dec. 2004.

[26] K. Wang, G. Cretu, and S. J. Stolfo, "Anomalous payload-based worm detection and signature generation," in *Recent Advances in Intrusion Detection, 8th International Symposium, RAID 2005*, 2005, pp. 227–246.

[27] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatically generating signatures for polymorphic worms," in *2005 IEEE Symposium on Security and Privacy*, May 2005.

[28] V. Karamcheti, D. Geiger, Z. Kedem, and S. Muthukrishnan, "Detecting malicious network traffic using inverse distributions of packet contents," in *2005 ACM SIGCOMM Workshop on Mining Network Data*, 2005, pp. 165–170.

[29] T. G. Handel and M. T. Sandford II, "Hiding data in the OSI network model," in *Information Hiding, First International Workshop*, 1996, pp. 23–38.

[30] C. H. Rowland, "Covert channels in the TCP/IP protocol suite," *First Monday*, vol. 2, no. 5, 1997.

[31] K. Ahsan and D. Kundur, "Practical data hiding in TCP/IP," in *Workshop on Multimedia and Security at ACM Multimedia '02*, Dec. 2002.

[32] J. Griffin, R. Greenstadt, P. Litwack, and R. Tibbetts, "Covert messaging through TCP timestamps," in *Privacy Enhancing Technologies, Second International Workshop, PET 2002*, 2003, pp. 194–208.

[33] S. J. Murdoch and S. Lewis, "Embedding covert channels into TCP/IP," in *Information Hiding, 7th International Workshop, IH 2005*, 2005, pp. 247–261.

[34] N. B. Lucena, G. Lewandowski, and S. J. Chapin, "Covert channels in IPv6," in *Privacy Enhancing Technologies, 5th International Workshop, PET 2005*, 2006, pp. 147–166.

[35] S. Cabuk, C. E. Brodley, and C. Shields, "IP covert timing channels: Design and detection," in *11th ACM Conference on Computer and Communications Security*, 2004, pp. 178–187.

[36] E. Vasserman, R. Jansen, J. Tyra, N. Hopper, and Y. Kim, "Membership-concealing overlay networks," in *16th ACM Conference on Computer and Communications Security*, Nov. 2009.

[37] L. R. Ford, Jr. and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, pp. 399–404, 1956.

[38] M. Demmer and K. Fall, "DTLSR: Delay tolerant routing for developing regions," in *Proceedings of the 2007 Workshop on Networked Systems for Developing Regions*, 2007, pp. 1–6.

[39] K. Wang, J. J. Parekh, and S. J. Stolfo, "Anagram: A content anomaly detector resistant to mimicry attack," in *Recent Advances in Intrusion Detection, 9th International Symposium, RAID 2006*, 2006, pp. 226–248.

[40] Y. Gu, A. McCallum, and D. Towsley, "Detecting anomalies in network traffic using maximum entropy estimation," in *2005 Internet Measurement Conference*, Oct. 2005, pp. 345–350.

[41] A. Wagner and B. Plattner, "Entropy based worm and anomaly detection in fast IP networks," in *14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*, 2005, pp. 172–177.

[42] K. Xu, Z.-L. Zhang, and S. Bhattacharyya, "Profiling internet backbone traffic: Behavior models and applications," in *2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, 2005, pp. 169–180.

[43] A. Chakrabarti, K. D. Ba, and S. Muthukrishnan, "Estimating entropy and entropy norm on data streams," in *23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS 2006)*, 2006, pp. 196–205.

[44] S. Guha, A. McGregor, and S. Venkatasubramanian, "Streaming and sublinear approximation of entropy and information distances," in *17th ACM-SIAM Symposium on Discrete Algorithms*, 2006, pp. 733–742.

[45] A. Chakrabarti, G. Cormode, and A. McGregor, "A near-optimal algorithm for computing the entropy of a stream," in *18th ACM-SIAM Symposium on Discrete Algorithms*, 2007, pp. 328–335.