Summary-Invisible Networking: Techniques and Defenses

Lei Wei, Michael K. Reiter, and Ketan Mayer-Patel

University of North Carolina at Chapel Hill, 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.

Keywords: network anomaly detection, flow records, covert channels, botnet command and control.

1 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 [17], 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., [20]) and diagnostics of various types (e.g., [32]), 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., [30,11,5,40]), and even for identifying the origin of worms [38]. Other anomaly detection techniques couple examination of summary records and limited payload inspection to find botnet command-and-control (C&C) channels [15,13,39]. Indeed, a

M. Burmester et al. (Eds.): ISC 2010, LNCS 6531, pp. 210–225, 2011.
© Springer-Verlag Berlin Heidelberg 2011

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 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 invisible to any summarization technique that does not inspect application payload contents. As such, the attacker will be invisible to any *summary-dependent* detector, i.e., for which detecting the attacker's behavior in summary records is a *necessary* ingredient, even if it employs payload inspection in other stages of its processing. For example, several botnet detectors leverage anomalous behavior exhibited in summary records, either to focus the attention of subsequent analysis [39] or to correlate with anomalies from other sources [14,15,13] to find infected machines. Suppressing evidence of botnet activities in the summary records, then, provides an avenue for potentially circumventing detection.

To implement this invisibility in summary records with certainty, we disallow altering the flow-level behavior of the network *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., [17]) 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 a SINner as observed by uncompromised hosts must be indistinguishable from that of an uncompromised host, lest the different behaviors induce uncompromised hosts to behave differently or detect the SINner outright. In particular, a SINner must covertly discover other SINners in such a way that does not interfere with regular interaction with uncompromised hosts.
- **SINners must satisfy application demands faithfully, even for each other.** Hosts perform application-level tasks (e.g., serving or retrieving content), interference with which can affect applications and, in turn, the behavior of the system as viewed in

event logs (including summary records) or by human users. Thus, a SINner must continue to faithfully perform tasks requested of it by its user(s) and other nodes.

In order to meet these requirements, SIN builds a covert network that piggybacks on existing network traffic, leveraging 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* [9,18].

Using these observations, we design a framework for SIN networking and evaluate it in this paper. Our evaluation primarily focuses on the use of SIN as a command-and-control (C&C) channel for botnet-like activity that would evade detection by summary-dependent anomaly detectors. Our evaluation shows that SIN networks should enable 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 then turn our attention to approaches that might be used to detect SIN networks. 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. The results 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.

2 Related Work

Our motivation for studying SIN networking is to understand the limits of network anomaly detection approaches that are dependent on traffic summaries. We are not the first to examine these limitations. For example, Collins et al. [6] evaluated the extent to which five proposed payload-agnostic anomaly detectors — Threshold Random Walk [19], server address entropy [24], protocol graphs [5], 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 past studies of which we are aware, by focusing on the ability of attacker's nodes to communicate with each other (e.g., to conduct botnet C&C) while avoiding detection by *any* payload-agnostic detector.

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., [23,22,36,29,34,27,21]. 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., [22,29,27,21]). 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., [23,36,34]) may be more successful at detecting SIN networks. We will discuss such approaches to efficiently examining payload contents that might be suited to detecting SIN networks in §6.

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) [9,18]. Since DTNs were introduced, there has been a large body of work on routing in such environments (e.g., [33,8]). Many of these schemes might 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. First, buffer space at the intermediate nodes in the SIN network is perhaps less constrained that has been assumed in previous DTN frameworks, since we generally assume participating nodes in the SIN network have adequate resources and moreover are compromised. A second difference is that the mix of contact opportunities between any two nodes in the network is not based on, e.g., mobility, but instead inherits the contact mix exhibited by the legitimate traffic that drives the SIN network. 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.

3 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 such that $c(p) = c(p')$ if p and p' differ only in the contents of their application payloads (and, of course, their TCP checksums). 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. Other 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 some versions of 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 payloads). When 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.

The goal of SIN networking is to implement an overlay network among compromised machines that is unnoticeable to traffic summarization techniques. Collected traffic records must be unchanged by the presence of SIN, and so 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.

4 A SIN Network Framework

In this section we describe a protocol framework for SIN networking. We emphasize, however, that this is only 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 protocols, and evaluate their performance on various tasks, in §5.

4.1 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., [16,28,1,12,26,25]). Murdoch and Lewis [26] 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. An alternative would be to exploit covert storage channels in application payloads, which would be undetectable in traffic summaries by definition. Covert *timing* channels (e.g., [3]) would risk detection owing to the availability of timestamps $t_1, \ldots, t_n$ to the summarization function f (see §3). 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 would typically be 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 §5) and collects the size 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 other information, as described in §4.2. §5.1 and §5.2 discuss how this information is used in particular routing protocols.

4.2 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 §5. While we do not incorporate robust defenses against SIN network infiltration (e.g., by law enforcement) in our design or against learning other participants in the event of such infiltrations, permitting 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 §4.4, 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 $2^{11} = 2048$ nodes.

Once a SINner discovers a neighbor (§4.1), 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.

4.3 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.

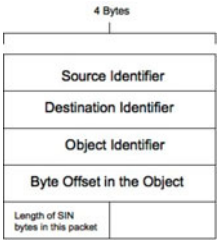

Fig. 1. SIN message header in experiments (object length is contained in the first 4 bytes of the object itself)

An object has an *identifier* that will be included in each SIN header for a message containing bytes of that object. 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). 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. Since, as will be shown in §4.4, 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. (See Figure 1.)

4.4 Available Capacity

There is reason to be skeptical of the capacity offered by compressing packet payloads. Client-server traffic 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 server) are sufficiently large to 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.

To understand the capacity offered by modern networks via this technique, we logged traffic over two days on our department network, recording the size to which each packet payload could be compressed using the `zlib` library. Only packets that could be compressed were kept, 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). After recording these traces, we 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 2. 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 capacity remains for SIN networking since it need not rely on bidirectional point-to-point communication. To demonstrate this, we used a larger dataset collected over three weeks, referred to as Dataset1 in this paper. This data was collected during the summer of 2009; summer is the time of lowest network utilization on a college campus, and thus we believe our capacity results to be conservative. Dataset1 includes traffic from over 8000 hosts, including both internal and external addresses. Since the IP addresses were anonymized, however, we are unable to further characterize particular nodes in the dataset.

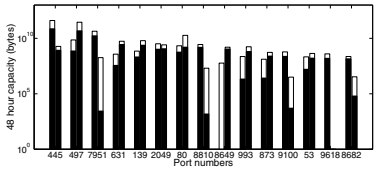

Fig. 2. Server ports that contributed the most SIN capacity over 14:56 Thu Sep 3 – 14:06 Sat Sep 5; left bar is inbound traffic, right bar is outbound; bar height denotes total bytes, black area denotes SIN capacity

We 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. Then, 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. Finally, we re-inserted all edges between nodes in that strongly connected component, and computed the maximum flow [10] 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.

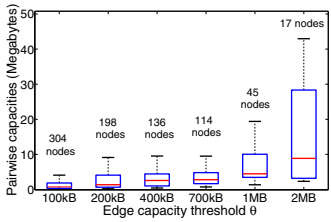

Fig. 3. Max-flow (selected nodes)

Figure 3 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. 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,

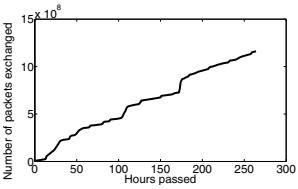

Fig. 4. Cumulative packets over time in Dataset1 between nodes in 114-node component

implying that there is much capacity to be gained by routing over multiple paths between pairs of nodes. That said, even when there are significant capacities between nodes, one cannot conclude that the latencies of communication between them 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, e.g., congestion, can substantially increase object transmission latencies.

In §5, we will focus on the 114-node connected component determined by $\theta =$ 700kB/day. Though botnets typically consist of more hosts, we limit our attention to this 114-host component as a relatively well-connected core that, e.g., could potentially bridge other infected but sporadically connected hosts in the network. (Botnets like Storm, Nugache and Waledac are structured hierarchically, with relatively well-connected bots playing such roles [31,2].) In Figure 4 we plot the packet volume of Dataset1 over time, restricted to packets between nodes in this component. We will plot results as a function of packets processed from this trace, and so Figure 4 enables one to translate these results to real time.

5 Evaluation of SIN for Attacker Workloads

The available SIN capacity demonstrated in §4.4 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 §4 with a state-of-the-art delay-tolerant routing protocol, namely the Delay Tolerant Link State Routing (DTLSR) protocol [18,7]. We use trace-driven simulations to evaluate the performance it achieves in tasks that are characteristic of activities an attacker might want to perform using a SIN network. We will focus on tasks motivated by botnet C&C.

Very briefly, DTLSR is an adaptation of classical link-state routing to DTNs. Like link-state protocols, DTLSR floods link-state updates (in our parlance, neighbor-table broadcasts) through the network. Each node uses these updates to maintain a graph representing its view of the network topology, and uses a modified shortest path algorithm to find the best path to the destination. Unlike typical 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 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.

5.1 Flooding and Neighbor Table Broadcast

The neighbor tables described in §4.1–§4.2 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 already possessed by the neighbor) are included in the vacated space, preceded by the header described in §4.3 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.

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.

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 to short-circuit the discovery process in some cases, but here we employ the more conservative approach.

In Figure 5, we demonstrate the progress of neighbor-table broadcasts performed among the same 114 SINners identified in §4.4 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 4 — the median SINner knows about 100 SINners.

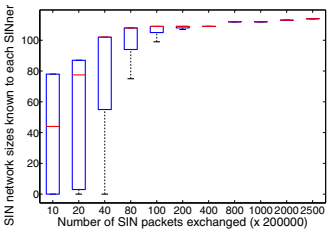

Fig. 5. Neighbor-table broadcast latency, 114 SINners (Dataset1)

5.2 Unicast Routing

A type of botnet activity conducted through a C&C channel is bots sending or responding to commands; for this, a point-to-point unicast protocol is needed. 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 §4.3. 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). (We examine other granularities in our accompanying technical report [37].) 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.

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. 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, 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.

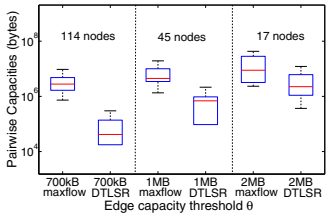

Fig. 6. Realized pairwise daily SIN capacities (Dataset1)

The results of this test are shown in Figure 6. This figure includes both the distribution of potential (maxflow) capacities taken from Figure 3 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. 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.

Data exfiltration. The next 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 selected arbitrarily from this component.

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. 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.

6 Detection of SIN Networks

If used to implement a botnet C&C channel, for example, SIN networking will interfere with the detection of this channel by summary-dependent detectors. This naturally raises the question as to what can be done to defend against SIN networks. One approach would be to try to prevent SIN networking by first patching all hosts to compress all outbound traffic (and decompress inbound traffic), so that a summary-based detector trains on summaries with no "room" for extra SIN payload. While potentially a long-term solution, this approach poses obvious incremental deployment difficulties.

We evaluated two approaches to detect SIN networks by trying to detect packet compression. Our first attempt involved trying to detect the delays associated with compression and was at best unreliable [37]. While SIN networking is premised on the notion that network defenders collect only summary information about their networks, our second approach nevertheless involved 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 [23,36,34]. In order to test this, we built a detector similar to PayL [36], 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).

Here we report the results of trying to detect SIN data in web traffic, as we conjectured that the rich content and media types available on web servers would make it difficult to identify individual SIN packets. (Tests on DNS traffic are described in our technical report [37].) For web sites hosting various 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 [35], in which we simply record which n-grams occurred in the training data for that site, rather than tracking their distribution.

To conduct this test, we used `wget` to retrieve all the contents of `www.unc.edu`, and built training and testing data from server response packets using K-fold cross-validation with $K = 5$. As such, 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. We also believe that our choice of 80% for training represents reality, in the following two senses. 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. Second, for a site that is updated more than `www.unc.edu`, there will be a gap between the current content of the site and the content on which the detector was trained.

For testing, we determined the number of n-grams in a 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 `www.unc.edu` 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 7. This plot shows that SIN detection based on 4-gram detection was quite poor when tested on high-entropy SIN data (the equal error rate is roughly 20%) and was abysmal when tested on low-entropy SIN data.

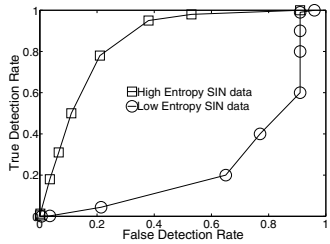

Fig. 7. ROC curve for 4-gram detector

The failure of our 4-gram detector suggests perhaps trying 5-grams, as was done in Anagram [35]. 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. Such an architecture would preclude monitoring efficiently, moreover. A second deterrent is that using higher n-grams to detect SIN data would presumably still trigger false positives on any new high-entropy data object not seen during training. As such, n-gram analysis might not be effective for servers with dynamic content.

We thus 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. One way to detect this 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 `www.unc.edu`, both without and with SIN data embedded (and in the latter case, for both low-entropy and high-entropy SIN data).

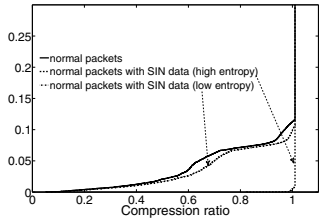

Fig. 8. CDF of compression test

Since we do not believe a per-packet detector is feasible for this measure — e.g., a single video response packet will be incompressible — 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 8, 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.

7 Conclusion

Summary-Invisible Networking (SIN) is a type of covert networking that strives to remain invisible to anomaly detection systems that examine traffic summaries. We presented a framework for SIN networks and showed the potentially achievable networking capacity that this framework permits. We evaluated its performance on tasks suggestive of what an adversary might attempt with a SIN network. We also examined approaches to detect SIN networks by payload analysis, with mixed results.

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. This work was supported in part by NSF awards CT-0756998 and CT-0831245.

References

1. Ahsan, K., Kundur, D.: Practical data hiding in TCP/IP. In: Workshop on Multimedia and Security at ACM Multimedia 2002 (December 2002)
2. Borup, L.: Peer-to-peer botnets: A case study on Waledac. Master's thesis, Technical University of Denmark (2009)
3. Cabuk, S., Brodley, C.E., Shields, C.: IP covert timing channels: Design and detection. In: CCS, pp. 178–187 (2004)
4. Collins, M.P., Reiter, M.K.: Finding peer-to-peer file-sharing using coarse network behaviors. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 1–17. Springer, Heidelberg (2006)
5. Collins, M.P., Reiter, M.K.: Hit-list worm detection and bot identification in large networks using protocol graphs. In: Kruegel, C., Lippmann, R., Clark, A. (eds.) RAID 2007. LNCS, vol. 4637, pp. 276–295. Springer, Heidelberg (2007)
6. Collins, M.P., Reiter, M.K.: On the limits of payload-oblivious network attack detection. In: Lippmann, R., Kirda, E., Trachtenberg, A. (eds.) RAID 2008. LNCS, vol. 5230, pp. 251–270. Springer, Heidelberg (2008)
7. Demmer, M., Fall, K.: DTLSR: Delay tolerant routing for developing regions. In: Workshop on Networked Systems for Developing Regions, pp. 1–6 (2007)
8. Erramilli, V., Crovella, M.: Forwarding in opportunistic networks under resource constraints. In: ACM MobiCom Workshop on Challenged Networks (September 2008)
9. Fall, K.: A delay-tolerant network architecture for challenged internets. In: SIGCOMM, pp. 27–34 (2003)
10. Ford Jr., L.R., Fulkerson, D.R.: Maximal flow through a network. Canadian J. Mathematics 8, 399–404 (1956)
11. Gao, Y., Zhao, Y., Schweller, R., Venkataraman, S., Chen, Y., Song, D., Kao, M.-Y.: Detecting stealthy attacks using online histograms. In: 15th IEEE Intern. Workshop on Quality of Service (June 2007)

12. Giffin, J., Greenstadt, R., Litwack, P., Tibbetts, R.: Covert messaging through TCP timestamps. In: Dingledine, R., Syverson, P.F. (eds.) PET 2002. LNCS, vol. 2482, pp. 194–208. Springer, Heidelberg (2003)
13. Gu, G., Perdisci, R., Zhang, J., Lee, W.: BotMiner: Clustering analysis of network traffic for protocol and structure independent botnet detection. In: USENIX Security (2008)
14. Gu, G., Porras, P., Yegneswaran, V., Fong, M., Lee, W.: BotHunter: Detecting malware infection through ids-driven dialog correlation. In: USENIX Security (August 2007)
15. Gu, G., Zhang, J., Lee, W.: BotSniffer: Detecting botnet command and control channels in network traffic. In: NDSS (February 2008)
16. Handel, T.G., Sandford II, M.T.: Hiding data in the OSI network model. In: Anderson, R. (ed.) IH 1996. LNCS, vol. 1174, pp. 23–38. Springer, Heidelberg (1996)
17. Hernández-Campos, F., Nobel, A.B., Smith, F.D., Jeffay, K.: Understanding patterns of TCP connection usage with statistical clustering. In: MASCOTS, pp. 35–44 (September 2005)
18. Jain, S., Fall, K., Patra, R.: Routing in a delay tolerant network. In: SIGCOMM, pp. 145–158 (2004)
19. Jung, J., Paxson, V., Berger, A.W., Balakrishnan, H.: Fast portscan detection using sequential hypothesis testing. In: IEEE Symp. Security and Privacy (May 2004)
20. Karagiannis, T., Papagiannaki, K., Faloutsos, M.: BLINC: Multilevel traffic classification in the dark. In: SIGCOMM (August 2005)
21. Karamcheti, V., Geiger, D., Kedem, Z., Muthukrishnan, S.: Detecting malicious network traffic using inverse distributions of packet contents. In: Workshop on Mining Network Data, pp. 165–170 (2005)
22. Kim, H.A., Karp, B.: Autograph: Toward automatic distributed worm signature generation. In: USENIX Security (August 2004)
23. Kruegel, C., Toth, T., Kirda, E.: Service specific anomaly detection for network intrusion detection. In: Symp. Applied Computing (March 2002)
24. Lakhina, A., Crovella, M., Diot, C.: Mining anomalies using traffic feature distributions. In: SIGCOMM, pp. 217–228 (2005)
25. Lucena, N.B., Lewandowski, G., Chapin, S.J.: Covert channels in IPv6. In: Danezis, G., Martin, D. (eds.) PET 2005. LNCS, vol. 3856, pp. 147–166. Springer, Heidelberg (2006)
26. Murdoch, S.J., Lewis, S.: Embedding covert channels into TCP/IP. In: Barni, M., Herrera-Joancomartí, J., Katzenbeisser, S., Pérez-González, F. (eds.) IH 2005. LNCS, vol. 3727, pp. 247–261. Springer, Heidelberg (2005)
27. Newsome, J., Karp, B., Song, D.: Polygraph: Automatically generating signatures for polymorphic worms. In: IEEE Symp. Security and Privacy (May 2005)
28. Rowland, C.H.: Covert channels in the TCP/IP protocol suite. First Monday 2(5) (1997)
29. Singh, S., Estan, C., Varghese, G., Savage, S.: Automated worm fingerprinting. In: OSDI (December 2004)
30. Staniford-Chen, S., Cheung, S., Crawford, R., Dilger, M., Frank, J., Hoagl, J., Levitt, K., Wee, C., Yip, R., Zerkle, D.: GrIDS – a graph based intrusion detection system for large networks. In: 19th National Information Systems Security Conf., pp. 361–370 (1996)
31. Stover, S., Dittrich, D., Hernandez, J., Dietrich, S.: Analysis of the Storm and Nugache trojans: P2P is here. USENIX;login 32(6) (2007)
32. Terrell, J., Jeffay, K., Smith, F.D., Gogan, J., Keller, J.: Exposing server performance to network managers through passive network measurements. In: IEEE Internet Network Management Workshop, pp. 1–6 (October 2008)
33. Vadhat, A., Becker, D.: Epidemic routing for partially connected ad hoc networks. Technical Report CS-200006, Department of Computer Science, Duke University (2000)
34. Wang, K., Cretu, G., Stolfo, S.J.: Anomalous payload-based worm detection and signature generation. In: Valdes, A., Zamboni, D. (eds.) RAID 2005. LNCS, vol. 3858, pp. 227–246. Springer, Heidelberg (2006)

35. Wang, K., Parekh, J.J., Stolfo, S.J.: Anagram: A content anomaly detector resistant to mimicry attack. In: Zamboni, D., Krügel, C. (eds.) RAID 2006. LNCS, vol. 4219, pp. 226–248. Springer, Heidelberg (2006)
36. Wang, K., Stolfo, S.J.: Anomalous payload-based network intrusion detection. In: Jonsson, E., Valdes, A., Almgren, M. (eds.) RAID 2004. LNCS, vol. 3224, pp. 203–222. Springer, Heidelberg (2004)
37. Wei, L., Reiter, M.K., Mayer-Patel, K.: Summary-invisible networking: Techniques and defenses. Technical Report TR09-019, Department of Computer Science, University of North Carolina at Chapel Hill (2009)
38. Xie, Y., Sekar, V., Maltz, D., Reiter, M.K., Zhang, H.: Worm origin identification using random moonwalks. In: 2005 IEEE Symp. Security and Privacy, pp. 242–256 (May 2005)
39. Yen, T.-F., Reiter, M.K.: Traffic aggregation for malware detection. In: Zamboni, D. (ed.) DIMVA 2008. LNCS, vol. 5137, pp. 207–227. Springer, Heidelberg (2008)
40. Yen, T.-F., Reiter, M.K.: Are your hosts trading or plotting? Telling P2P file-sharing and bots apart. In: ICDCS (2010)