Secure Split Assignment Trajectory Sampling: A Malicious Router Detection System

Sihyung Lee Tina Wong Hyong S. Kim
ECE Department and CyLab
Carnegie Mellon University
sihyunglee@cmu.edu, tinawong@cmu.edu, kim@ece.cmu.edu

Abstract

Routing infrastructure plays a vital role in the Internet, and attacks on routers can be damaging. Compromised routers can drop, modify, mis-forward or reorder valid packets. Existing proposals for secure forwarding require substantial computational overhead and additional capabilities at routers. We propose Secure Split Assignment Trajectory Sampling (SATS), a system that detects malicious routers on the data plane. SATS locates a set of suspicious routers when packets do not follow their predicted paths. It works with a traffic measurement platform using packet sampling, has low overhead on routers and is applicable to high-speed networks. Different subsets of packets are sampled over different groups of routers to ensure that an attacker cannot completely evade detection. Our evaluation shows that SATS can significantly limit a malicious router's harm to a small portion of traffic in a network.

1. Introduction

Routers are crucial to the Internet. Unfortunately, attacks aimed directly at routers are prevalent and on the rise. According to CERT/CC, there are lists of thousands of compromised routers being traded underground [5]. There are hacker tools, openly available on the Web, to scan, identify and eventually exploit routers with weak passwords and default settings. More recently, Lynn [14] presented Cisco IOS's security flaws, which allow hackers to seize control of Cisco routers. Compromised routers are being used as platforms to send spam, launch Denial-of-Service (DoS) attacks, intercept sensitive traffic, and carry out illegal yet profitable activities. In general, since routers are considered trusted entities in a network, their power can be easily exploited once they are compromised.

This work was funded in part by KISA, MIC, Samsung, ARO and Carnegie Mellon CyLab.

In order to secure the Internet routing infrastructure, the two main planes of network functionality (i.e. control and data) must be protected. The control plane runs intradomain and interdomain routing protocols to build forwarding tables at routers. The data plane forwards (or drops) packets according to forwarding tables built by the control plane. Recently, considerable research and industrial efforts have addressed securing routing protocols, e.g. securing Border Gateway Protocol (BGP), the de facto glue for Internet interdomain connectivity [9]. A secure version of BGP provides path and prefix attestations, which prevent propagation of illegitimate routes. Even in the presence of a secure control plane, however, a compromised router can disregard decisions made by the control plane and act autonomously and maliciously on the data plane. It can modify, drop, delay, reorder, mis-forward valid packets or permit otherwise prohibited packets. Such misbehavior would not be prevented by any secure routing protocol.

This paper presents *Secure Split Assignment Trajectory Sampling (SATS)*, a system that detects packet modification, substitution, dropping, reordering and forwarding loop attacks carried out by subverted routers. SATS detects attacks if the observed paths of packets are not consistent with the predicted ones, and pinpoints a set of suspicious routers. As SATS is designed for high-speed networks, it relies on packet sampling, and so routers only need to do additional processing on a subset of packets. While sampling decreases monitoring overhead at each router, the accuracy of detection depends on how samples are selected. If the samples do not contain compromised packets, attacks would not be detected. Thus, SATS uses *Split Range Assignment* that prevents attackers from biasing sample selection. To sample packets and to observe paths that the selected packets followed, SATS can be integrated with the Trajectory Sampling traffic measurement system [2]. Secure communication between routers and the backend measurement engine is also needed. SATS has low impact on router processing and memory usage: it only applies a modular hash function on each packet and a cryptographic hash function on selected packets. The rest of the detection process is done externally on a backend measurement engine.

2. Related Work

Related work on detecting and locating data-plane misbehavior falls into two categories: active probing and passive monitoring. SATS is a passive monitoring scheme.

Secure Traceroute [13] sends probing packets to detect packet drop, modification and mis-forwarding attacks. Once a human operator notices performance degradation on a path, Secure Traceroute is initiated and routers respond to the probing packets. Stealth Probing [1] sends probing packets through an encrypted channel with normal packets to make probing packets indistinguishable from normal packets. To protect all packets, every packet has to be encrypted in this scheme. In general, in active probing, the probing packets must be similar to the packets that are being attacked. However, it is not easy to determine beforehand which packets will be attacked. It is also not clear when to initiate probing and where to probe.

The Conservation of Flow (CF) [8] is a passive monitoring scheme. CF analyzes traffic volume at various observation points in a network. Discrepancies between the ingress traffic volume and the egress traffic volume at different points indicate potential problems. CF can discover dropping of packets but fails to detect modification of packets if traffic volume remains the same. Hughes et al. [8] address several ways to fool the CF algorithm. Fatih [11] considers other types of attacks including packet modification, substitution, mis-forwarding and reordering. Routers compute hashes of packet content to validate integrity of content as well as ordering. To reduce the overhead on routers, Fatih proposes a path-level detection algorithm while increasing the size of the suspicious set of routers. For both CF and Fatih, most of detection processing is done on routers. In SATS, only a subset of packets are sampled by routers and the sampled packets are examined on an external measurement engine.

Listen [15] and Feedback-based Routing [16] detect connectivity problems between two end-hosts by monitoring TCP packets but does not aim to localize the problems. Finally, Herzberg and Kutten [7] proposes a protocol that detects delaying and possibly dropping of packets by using timeouts and acknowledgments.

3. Background

In this section, we briefly describe previous works on flow-level measurement and Trajectory Sampling (TS). We discuss only material relevant to SATS. Flow-level measurement provides more fine-grained information about network traffic than the traditional link-level SNMP approach. A router periodically sends records about its flows to a collection machine that processes the records. Cisco's Netflow as well as flow-level measurement solutions from other vendors are widely deployed in networks. These tools are used to calculate traffic matrices to provision a network, understand traffic mix in terms of application types, and find reasons behind sudden traffic spikes.

For high data rate interfaces, packet sampling is necessary to scale flow-level measurement. Otherwise, a router quickly runs out of processing cycles and memory while trying to examine every passing packet, and its forwarding performance is severely degraded. Netflow uses a simple 1-out-of-N sampling method but not without problems [4]. Improved methods have been proposed, such as the novel Trajectory Sampling (TS) proposed by Duffield and Grossglauser [2]. The main idea of TS is that a packet is either sampled at every router along its path, or not sampled at all. The IETF is working on standardizing various aspects of flow-level measurements including TS to ensure multi-vendor compatibility and industry-wide acceptance. TS works as follows:

- A router applies a *selection hash function*, $h_{selection}()$, to compute a *hash value* over the invariant portion of a packet. The source and destination IP addresses, port numbers, protocol and payload remain the same as a packet travels across the network, and thus are included in the calculation. On the other hand, the TTL, ECN, TOS and CRC checksum are not included in the calculation as they can be changed.

  In order to achieve unbiased and uniform sampling, this hash function must generate values that appear statistically independent of its input. Using large packet traces from a Tier-1 ISP network, [2] show that modular arithmetic with prime moduli satisfies this property.

- If the hash value falls into the predetermined *sampling range*, the packet is sampled. The size of this sampling range, N_{small}, divided by the total size of the *hash range*, N_{total}, is the effective sampling rate, p_{samp}. The actual sampling rates used vary by networks, and we have seen 1/100, 1/1000 and smaller.

- For each sampled packet, a *label hash function*, $h_{label}()$, computes another hash value over the invariant portion of a packet. The hash value is called the *label* of the packet since the label provides a unique ID of the packet. This label is then reported back to a machine for processing – this is called the *backend engine* in TS. A different hash function is used to make the label small, yet unique during a measurement interval. An ingress router also reports a *key*, which contains raw header information of a flow, for each sampled packet.

- The backend engine reconstructs the path a sampled packet traversed in the network. This path is called the packet's *trajectory*. The backend engine sets a timer when it first receives a label l. When the timer expires, the backend engine gathers the routers that sent label l and reconstructs the packet's trajectory. It assumes that the network topology is known. A possible timeout value

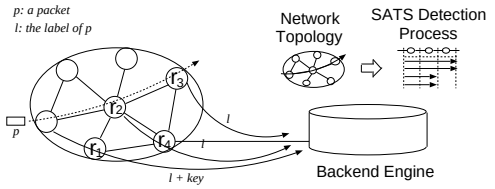


Figure 1. Overview of SATS.

is the sum of the upper bounds of all possible delays: the delay within the label buffer before a label is exported, and the propagation and queuing delay a packet experiences from a reporting router to the backend engine. The backend engine also provides storage, query and visualization functions.

- $h_{selection}()$, $h_{label}()$ and the sampling range must be identical at all routers during a measurement interval. This ensures that a packet is sampled at all routers on its path or at none. If each router randomly samples a subset of packets as in Netflow's 1-in-N method, a packet's path cannot be reconstructed. To adjust the sampling rate, an out-of-band mechanism is used to change the sampling range on all routers.

Besides the benefits of fine-grained flow-level measurements and reduction of overhead on router performance through sampling, TS has a number of additional advantages. It is a direct observation method: a packet's or flow's path is measured directly, without needing to know IGP and BGP routing state. On the other hand, Netflow-style data requires routing state, in order to derive the original path of a packet or flow. This indirect method is not as accurate as the direct observation, as there are synchronization issues during the computation. Another advantage is that the reconstructed trajectories can be used to passively measure link performance, without the injection of active probe traffic into a network.

4. SATS

4.1. Overview and System Model

SATS extends upon TS to detect malicious routers. In this paper, SATS is limited to routers within a single administrative domain, such as ISP, university campus or enterprise network, or multiple cooperating domains that are open to share measurement data with one another. We also assume that the network topology is known. SATS can detect one or multiple consecutive malicious routers on a path if there are non-malicious routers at both ends of the malicious routers. Thus, we assume that the first and the last router on a path are correct. This assumption is necessary and also stated in all related work. SATS cannot detect mis-forwarding behavior that does not manifest itself as a forwarding loop.

Figure 1 illustrates the SATS design, which consists of the following functions:

Split Range Assignment: SATS assigns multiple overlapping hash ranges to routers to minimize the probability that an attacker can completely evade detection. In contrast, TS uses a single universal sampling range for every router in a network, which is vulnerable to attacks. We call our scheme *Split Range Assignment* and TS's *Single Range Assignment*.

Report Collection: A router samples a subset of packets based on its assigned sampling range and reports the hash labels and keys of sampled packets to the backend engine. Report collection in SATS is the same as TS. SATS (as in TS) is designed with router performance in mind, and we aim to keep router state and processing to a minimum. The next three functions are carried out by the backend engine, not by routers.

Reconstruction and Aggregation of Trajectories: At the end of each measurement interval, the backend engine reconstructs trajectories of sampled packets using reports from routers. Trajectories with the same ingress router and destination routing prefix pair are in turn aggregated. SATS detects anomalies based on this aggregation unit instead of a single trajectory.

Pinpointing Malicious Routers: In each aggregation, SATS looks for inconsistent trajectories that are different from their predicted trajectories. If inconsistent trajectories are found, SATS locates a set of suspicious routers that are responsible for the attack.

4.2. Split Range Assignment

Let us consider a malicious router that not only misbehaves, but also tries to avoid detection by attacking packets that are not being sampled. In Single Range Assignment, the malicious router knows that the hash range it has been assigned is the only sampling range being sampled throughout the network. In Split Range Assignment, we vary sampling ranges from router to router. Different sampling ranges are assigned by the backend engine through encrypted and authenticated channels to ensure that the sampling range assigned to a router is unknown to other routers. Thus, only the backend engine has knowledge of the entire range assignments.

Figure 2 illustrates Split Range Assignment. A long vertical rectangle at a node represents the entire hash range of the selection hash function. Within the entire hash range, the small gray rectangles depict the hash values in the sampling range at the node. As opposed to Single Range Assignment in Figure 2.(a), Split Range Assignment in Figure 2.(b) has sampling ranges that varies from router to router. However, the sampling rate $p_{samp} = N_{small}/N_{total} = 6/31 = 0.19$ at a router remains the same. If a malicious router, r_4, ever attacks packets outside of its sampling range $\{25, 15, 28, 14, 12, 8\}$ (e.g., hash value 4) the attack will soon be noticed by $\{r_2, r_6\}$. Note that Split Range Assignment is not random sampling (as in Cisco Netflow) since more than one router would be as-

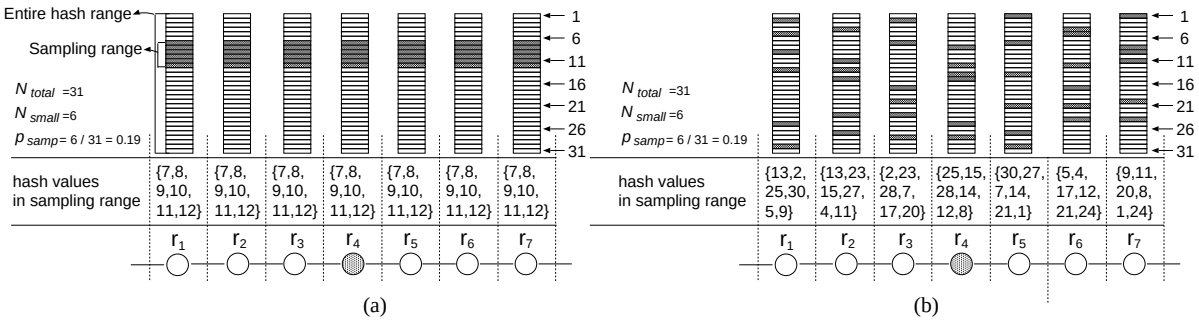


Figure 2. Hash ranges in Single Range Assignment (a) and Split Range Assignment (b).

$\mathcal{S}_{total}$: a set of all the routers in a network
$\mathcal{PR}$: a set of prime numbers, where $prime_i \in \mathcal{PR}$ is the i-th element
N_{router}: the number of routers in $\mathcal{S}_{total}$

$N'_{small} = N_{router} - 1$; $N'_{total} = N'_{small}/p_{samp}$;
$N_{total} = min_i(prime_i),\ prime_i \geq N'_{total}$;
N_{small}=$\lfloor (N_{total} \times p_{samp}) \rfloor$;
for each pair of routers (r_i,r_j), $i < j, r_i \in \mathcal{S}_{total}, r_j \in \mathcal{S}_{total}$:
select one hash value h_i out of N_{total} hash values in the entire hash range at random
assign h_i to (r_i,r_j)

Figure 3. Split Range Assignment algorithm.

signed the same sampling range, such as the hash value 4 in $\{r_2, r_6\}$ in Figure 2.(b). If all the routers have different sampling ranges (as with random sampling), we cannot compare samples from one router with samples from any other routers.

We assign different sampling ranges as follows. A router shares one hash value with each router in a network. Thus, if there are N_{router} routers in the network, $N_{small} = N_{router} - 1$ hash values are assigned to a router. Given p_{samp}, we find N_{small} and N_{total} for the selection hash function as shown in Figure 3. For example, with $p_{samp} = 6/31$ and $N_{router} = 7$ routers in the network (Figure 2.(b)), $N_{total} = 31$, which is the smallest prime number greater than or equal to $(N_{router} - 1)/p_{samp} = 31$. Then $N_{small} = \lfloor N_{total} \times p_{samp} \rfloor = 6$ hash values are assigned to each router in a way that one router has one hash value in common with each of the other 6 routers. For instance, r_1 shares the hash values 13, 2, 25, 30, 5 and 9 with r_2, r_3, r_4, r_5, r_6 and r_7, respectively. We evaluate the detection rate of this assignment method in Section 6.1 and we show that the probability that an attacker can avoid detection is fairly low.

Split Range Assignment also ensures that SATS is able to trace packets in case routing changes. On any given path, each pair of routers have at least one common hash value. Thus, we can compare samples from one router with samples from any other routers.

4.3. Inconsistent Trajectories

Split Range Assignment prevents malicious routers from tampering with the sampling process. Sampled packets are then reported to the backend engine, where the trajectories of the packets are reconstructed. The trajectories are then fed into the SATS detection process. We first provide the main premise of the SATS detection process. We then elaborate each step in detail in the following sections.

The main premise of the SATS detection process is that if packets are manipulated, the resulting trajectories will not be consistent with the predicted trajectories. Figure 4 shows the predicted and resulting trajectories of a packet p, which is supposed to go from an ingress node r_{i-2} to an egress node r_{i+3}. The label of p is $l = h_{label}(p)$. The predicted trajectory of p is shown in Figure 4.(a). If p is dropped at a node r_i (Figure 4.(b)), the trajectory of p, t_{normal}, ends at r_i before reaching its egress node. If p is modified to p' at r_i (Figure 4.(c)), the trajectory for the label l, t_{normal}, also ends prematurely at r_i since the label is changed to $l' = h_{label}(p')$ before p is forwarded to r_{i+1}. In the case of modification, a new trajectory, t_{orphan}, for the new label l' starts from r_{i+1} if $h_{selection}(p')$ falls in the sampling range. We refer to the new trajectory as an *orphan trajectory* as opposed to a *normal trajectory* since the new trajectory has no origination point, that is, no ingress router reported the trajectory's label, l'. When t_{orphan} is discarded, the two early ended normal trajectories, t_{normal}'s, in (b) and (c) look the same. Consequently, we use the early ending of a normal trajectory as a clue to detect both packet dropping and modification. Orphan trajectories are discarded since we already have a way to detect anomalies. Substitution of a packet for another packet yields two modified packets and is thus detected as well.

In Split Range Assignment, a packet p sampled from a hash value h is not sampled at all the routers on the path. Thus, the trajectory t of p has holes on the routers with hash values other than h. If t has a hole in any of the routers where h is assigned, the trajectory is inconsistent. t is an orphan trajectory if t has a hole in the first router where h is assigned on p's path. In [10], we have shown that SATS can perform traffic measurement

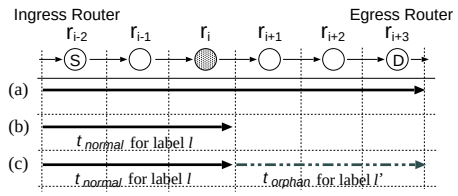


Figure 4. The predicted trajectory of a packet p (a) and the resulting trajectories of p when p was dropped at node r_i (b) and when p was modified at node r_i (c).

functions of TS even with holes in trajectories.

Because of space, we focus on the detection of packet dropping, modification and substitution in the rest of this paper. The detection of routing loops and reordering attacks is described in detail in the technical report version of this paper [10]. Briefly, if trajectories intersect themselves, we report them as routing loops; we detect packet reordering attacks by comparing the time-ordered list of labels from a router with the list from another router with the same sampling range.

4.4. Aggregation of Multiple Trajectories

At the end of each measurement interval, trajectories are aggregated into the same flow, which is defined as the trajectories with the same ingress router and destination routing prefix pair. Based on each *aggregation* rather than each trajectory, we make a decision concerning anomalies. The aggregation is done mainly for two reasons. First, running one detection process for each set of aggregated trajectories scales better than running one detection process for each trajectory, especially in a high load situation where thousands of packets are sampled each second. Second, the aggregation helps us use a threshold to differentiate between legitimate packet drops and malicious attacks. We do not raise an alarm for each inconsistent trajectory, which might be caused by congestion or an error in the packet header[1]. Instead, we raise an alarm only when the number of inconsistent trajectories in an aggregation is more than the threshold. Setting the threshold depends on the network. For example, legitimate packet losses in wireless networks are much more likely than in wired networks. Thus, the threshold must be determined accordingly. Also, the threshold has to be changed adaptively according to the average traffic volume. The number of samples collected from a link and the link capacity can help to predict the legitimate packet loss ratio and can thus help to determine the threshold.

In order to identify the flow of a normal trajectory t, we need to know both the ingress router and the destination routing prefix. We use the key reported from the ingress router since the key includes the destination routing prefix. However in Split Range Assignment, t may have a hole in the ingress router. Thus, we have all routers, not only the ingress routers, report a key for each sampled packet. The key has to include the source IP and the destination routing prefix. We then identify the ingress router of t through another trajectory t' that does not have a hole in its ingress router. The ingress router of t' is the same one for t if t' has the same source IP and destination prefix as those of t.

$t_i[r_j]$: If a trajectory t_i has a report from r_j, $t_i[r_j] = 1$. If not, $t_i[r_j] = 0$.
$\mathcal{H}(i)$: a set of N_{small} hash values in the sampling range assigned to r_i
$C_{prim}[h_j][r_i]$: the primary counter for the hash value h_j of r_i
$\mathcal{S}_{suspect}$: a set of suspicious routers
// Aggregation algorithm
for each normal trajectory t_i sampled from a hash value h in a flow:
  for each router r_j on the path:
    if ($t_i[r_j] = 1$ and $h \in \mathcal{H}(i)$) **then** $C_{prim}[h][r_j] = C_{prim}[h][r_j] + 1$;
// Detection and Pinpointing algorithm
for each node r_l in the flow:
  for each hash value $h_k \in \mathcal{H}(l)$:
    if $((C_{prim}[h_k][r_l] - C_{prim}[h_k][r_r]) > TH_{prim})$ **then**
    // r_r is the closest node to r_l in the flow, where $\mathcal{H}(r) \ni h_k$ and $r > l$
      $\mathcal{S}_{suspect} = \mathcal{S}_{suspect} \cup \{r_s : l \leq s \leq r\}$;

Figure 5. Aggregation and Detection algorithms.

Figure 5 shows the pseudo code of the aggregation algorithm. The aggregation process can be thought of as overlaying of trajectories one after another. In the backend engine, we maintain a *primary counter*, C_{prim}, for each hash value assigned to each node on the path. $C_{prim}[h]$ counts the number of normal trajectories sampled from a hash value h in a node. Thus, if some of the trajectories end prematurely at a node r, then C_{prim} decreases for nodes beyond the node r in the path. If the decline is more than the threshold TH_{prim}, the aggregation is marked as an anomalous one. Dropping of packets below the threshold can avoid detection, but the attack would not be more significant than temporary congestion.

Figure 6 illustrates an example of the aggregation algorithm. The number in the parentheses next to a trajectory is the hash value where the trajectory is sampled. The primary counter values are shown at the bottom. We assume that the hashes of the six trajectories are distributed into three different hash values, 3, 5 and 6. A trajectory sampled from a hash value h has "holes" on the routers where h is not assigned. However, the aggregation of trajectories shows the complete path followed by the packets in the flow, which is the *predicted trajectory* of the packets. A malicious router, r_3, modifies half of the packets, b, d and f. The modified packets result in three orphan trajectories, $\{t'_b, t'_d, t'_f\}$. The orphan trajectories are then discarded during the aggregation, since we already have a way to detect anomalies as shown by the early ending of nor-

[1]The loss of report packets en route to the backend engine can also create incomplete trajectories. [3] provides a way to infer the packet loss rate from the report loss rate. The report loss rate is estimated from the sequence numbers carried by report packets.

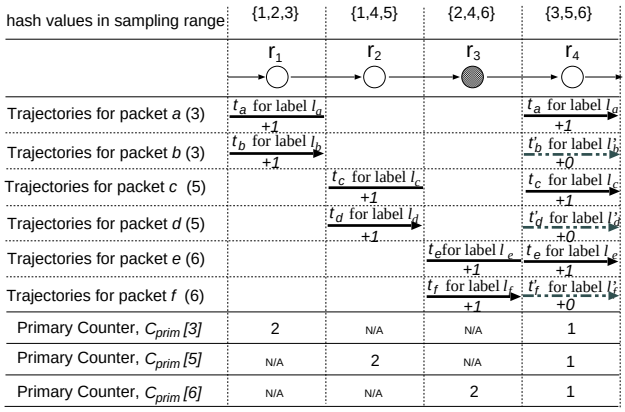


Figure 6. Aggregation of Trajectories.

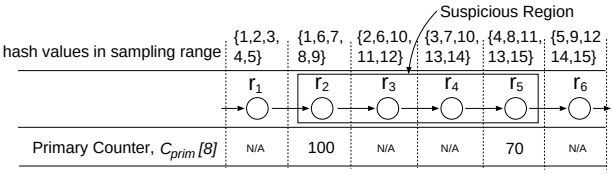


Figure 7. An example of a suspicious region.

mal trajectories, $\{t_b, t_d, t_f\}$. As the normal trajectories of b, d and f, $\{t_b, t_d, t_f\}$, end at r_3, they do not increment C_{prim}'s for nodes beyond r_3. Thus, when the normal trajectories are aggregated, C_{prim} decreases beyond r_3.

C_{prim} represents the number of normal trajectories sampled from each router. Note that C_{prim} is maintained in the backend engine, not in each router, and updated when trajectories are aggregated in the backend engine.

4.5. Pinpointing Malicious Routers

Once trajectories are aggregated, we check for inconsistent trajectories in each aggregation and pinpoint a set of suspicious routers. We show the algorithm in Figure 5. To find inconsistent trajectories, we compare C_{prim}'s for two routers $\{r_i, r_j\}$ $(i < j)$ where the same hash value h is assigned. We compare the samples of the same packets. If the decrease in C_{prim} for $\{r_i, r_j\}$ is more than TH_{prim}, we pinpoint all the routers between r_i and r_j including $\{r_i, r_j\}$ to be *the suspicious region* that includes a set of routers $\mathcal{S}_{suspicious} = \{r_k : i \leq k \leq j\}$. Figure 7 shows an example of a suspicious region. Since r_2 and r_5 have the same hash value 8, the $C_{prim}[8]$'s for $\{r_2, r_5\}$ are compared. Observing a decrease in the $C_{prim}[8]$'s for $\{r_2, r_5\}$, we conclude that $\{r_2, r_5\}$ and the routers between $\{r_2, r_5\}$ are suspicious. Thus, $\mathcal{S}_{suspicious} = \{r_2, r_3, r_4, r_5\}$.

The reason for choosing $\mathcal{S}_{suspicious}$ as suspects is based on three possible scenarios: (In Figure 7, r_i and r_j correspond to r_2 and r_5, respectively.)

Scenario 1: r_i is malicious. The packets were dropped at r_i. r_j reported correctly that it had not seen the packets. r_j did not report the dropped packets to the backend engine. Alternatively, when there are consecutive malicious routers, one of the previous nodes $r_l \in \{r_n : 1 \leq n < i\}$ dropped the packets, the following nodes from r_{l+1} to r_i misinformed the backend engine that they had observed the packets and finally r_j reported correctly.

Scenario 2: r_j is malicious. r_i forwarded the packets correctly, but r_j did not report the packets to the backend engine. r_j may or may not have dropped the packets.

Scenario 3: One of the routers between r_i and r_j (i.e., $r_m \in \{r_n : i+1 \leq n \leq j-1\}$) dropped the packets. The hashes of the dropped packets do not fall in the sampling ranges of any routers between r_m and r_j, $\mathcal{S}_{non} = \{r_n : m \leq n \leq j-1\}$. In other words, the packets under attack are not supposed to be sampled at any of the routers in $\mathcal{S}_{non}$. Thus, C_{prim}'s for $\mathcal{S}_{non}$ do not decrease. On the other hand, the hashes of the packets fall in the sampling range of r_j, and C_{prim} for r_j decreases.

In all three scenarios, one of the routers in $\mathcal{S}_{suspicious}$ must be faulty. To further reduce the size of $\mathcal{S}_{suspicious}$, we assign the hash value h that is common in the sampling ranges of $\{r_i, r_j\}$ to the routers between $\{r_i, r_j\}$ [10].

Consecutive Malicious Routers: Although there might be other colluding routers before r_i as shown in scenario 1, we first examine $\mathcal{S}_{suspicious}$. Among all the consecutive colluding routers, the last one must be in $\mathcal{S}_{suspicious}$. If the routers before r_i continue to behave maliciously, they will be eventually detected one by one. In summary, if there is a correct node at the end of the colluding routers, all colluding routers are detected.

Incremental Deployment: Between r_i and r_j, there might be a set of routers, $\mathcal{S}_{non}$, that do not have deployed SATS. In this case, the routers in $\mathcal{S}_{non}$ are also included in the suspicious region since C_{prim} decreases at the same node r_j if any one of $\mathcal{S}_{non}$ have manipulated the packets.

Equal-Cost Multi-Paths and Routing Changes: We may observe an aggregation where a node, r_i, branches into $n(> 1)$ nodes, from rb_1 to rb_n, if there are equal-cost multi-paths or routing changes. In this case, we compare C_{prim} for r_i with the sum of the C_{prim}'s for all the nodes branching from r_j. Thus, if $((C_{prim}[h][r_i] - \sum_{k=1}^{n} C_{prim}[h][rb_k]) > TH_{prim})$, we suspect the set $\{r_i\} \cup \{rb_k : 1 \leq k \leq n\}$. Then, from each of rb_k, we resume the detection process. Figure 8 shows an example where node 4 branches into node A and node D. The number below a node represents the primary counter C_{prim} for the node. We assume that node 4, A, and D have the same hash value. The detection process is running at node 4. If $(100 - (30 + 40) > TH_{prim})$, both node A and D along with node 4 are reported as suspicious. We then resume the detection process from node A and D.

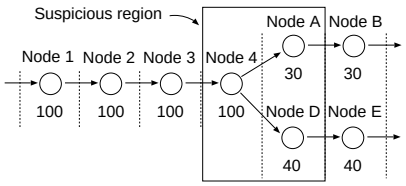


Figure 8. Suspicious region when an aggregation branches into two paths.

5. Security Analysis

In this section, we address a variety of possible scenarios that could be used to circumvent or confuse SATS. For each scenario, we show how SATS detects the attack and suggest some countermeasures, if necessary.

5.1. Dropping of Report Packets and Misreporting

Although report packets are sent to the backend engine through encrypted and authenticated channels, a malicious router may not forward the reports of other routers. This behavior is detected as a normal packet drop as reports are transmitted in packets and can thus be sampled. In an extreme case (Figure 1), a malicious router, r_4, on the way to the backend engine can drop all the report packets from a set of routers, $\{r_1, r_2, r_3\}$. If the backend engine does not receive any report from a router r_i, the routers on the way from r_i to the backend engine are assumed to be suspicious and examined. In this example, $\{r_1, r_4\}$, $\{r_2, r_4\}$ and $\{r_3, r_4\}$ are in the suspicious regions. Among the routers in the suspicious regions, the router that overlaps the most, router r_4, is examined first. If r_4 keeps dropping report packets, it will be the first router to be examined and detected.

If a malicious router does not sample some of the packets in its sampling range, or samples wrong packets, the corresponding labels are not reported. It results in an inconsistent trajectory as well.

5.2. Label Collision Attack

A malicious router, r, can modify a packet p with label l into another packet p' with the same label l, if r can find p' where $h_{label}(p') = h_{label}(p) = l$. Then, the trajectory of p will not end early at r. Thus, the trajectory appears complete, and r can go undetected. In order to prevent this attack, we use a 2nd-preimage resistant hash function (for a given value x, it is computationally infeasible to find a $x' \neq x$ such that $h(x') = h(x)$.) to generate labels. Popular checksum algorithms, such as MD-5 [2] or a universal one-way hash function with stronger security and statistical properties [12] can be used. Although we use a cryptographic label hash function $h_{label}()$, the selection hash function, $h_{selection}()$, remains the same as that of TS.

[2]Note that 2nd-preimage resistance is different from collision resistance, where x is not given, and both of x and x' are to be chosen such that $h(x') = h(x)$. Although MD-5 is not collision resistant, it is still 2nd-preimage resistant.

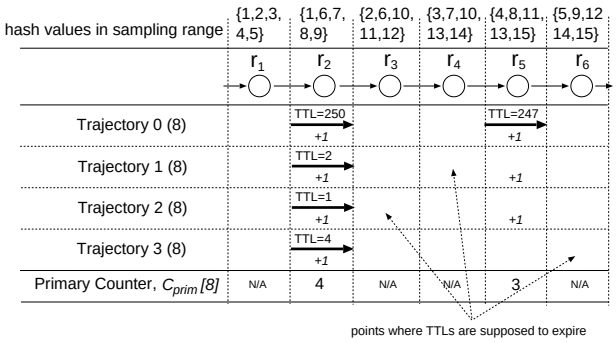


Figure 9. Solution to reduce the false positives due to TTL adjusted packets.

5.3. TTL Modification

Changing the TTL value to a very low number could cause a packet to be dropped by other legitimate downstream routers. This behavior would divert attention away from the real attacker. Such changes would not be caught by C_{prim} as $h_{label}()$ does not operate over variant fields including TTL. This behavior can be prevented by reporting the TTL value, along with a label, at each sampling node. If the difference between the TTL values from two nodes are more than the number of hops between the nodes, these nodes are suspects.

Some legitimate packets may also be detected as anomalous, especially the packets where TTL is adjusted so that they can be dropped prematurely (e.g., traceroute packets). If the ratio of these packets is high enough to cause false positives, we can compensate for the effect by incrementing the C_{prim}'s for the nodes after the point where the TTL is supposed to expire. Figure 9 illustrates such an example. We assume all the trajectories are sampled from the hash value 8. Thus, we only add to $C_{prim}[8]$'s. Since trajectories 1 and 2 end where the TTL's are supposed to expire, the additions to C_{prim}'s are the same as trajectory 0 when they are aggregated. On the other hand, trajectory 3 indicates that it ended earlier.

6. Evaluation

We evaluated the detection rate and incremental deployment property of SATS using simulation and present the results here. We used 14 topologies with various node degrees and network diameters: four are published by real networks, and the rest are generated by Georgia Tech's topology generator using either random or Waxman models [6]. We did

Table 1. Summary of topologies used in our evaluation.

Topology	No. of Nodes	No. of Edges	Node Degree	No. Nodes on Shortest Path	Link Metric?
ATT	54	144	1, 1.5, 6	5, 7, 9	No
CENIC	25	54	1, 2, 4	5, 7, 9	Yes
Berkeley	43	112	1, 1, 6.5	4, 5, 6	Yes
CMU	102	250	1, 1, 4.3	4, 6, 6	No
Waxman1	100	286	1, 3, 5	4, 6, 8	No
Random1	100	374	2, 3, 6	3, 5, 6	No
Random2	100	712	4, 7, 10	3, 4, 4	No
Waxman2	100	1004	6, 10, 14	2, 3, 4	No

not use the transit-stub model because this evaluation is for SATS in a single autonomous system. Eight of the 14 topologies are summarized in Table 1. The three numbers under the "Node Degree" and "No. Nodes on Shortest Path" columns denote the 10th-percentile, median and 90th-percentile values, respectively. In our simulation, for each topology, we compute the shortest path between each pair of edge routers to be the path taken between the routers. If link metrics are available for the topology, we use it in the shortest path calculation. Otherwise, we assume all links have equal weights.

6.1. Probability of Complete Avoidance

In Single Range Assignment, a malicious router can completely evade detection by targeting packets whose hashes fall outside one universal sampling range. In this section, we show that it is very difficult for such an attack to go undetected in Split Range Assignment.

We assume that a malicious router, r_m, randomly selects p_{attack} percentage of hash values out of N_{total} values of $h_{selection}()$. Then, the number of hash values selected by the malicious router r_m is $N_{attack} = p_{attack} \times N_{total}$. Let $\mathcal{H}_{attack}(m)$ denote a set of the N_{attack} hash values chosen by r_m. r_m targets the packets whose hashes fall in $\mathcal{H}_{attack}(m)$. Thus, on average, r_m attacks p_{attack} percentage of all the packets.

We define *complete avoidance* to be the event where all the hash values in $\mathcal{H}_{attack}$ are not assigned to any router in a given path. In the case of complete avoidance, the packets being attacked are not sampled at any router and the attack cannot be detected. We vary p_{attack} from 1% to 100% in 10% increments and derive the probability of complete avoidance, p_{avoid}. With higher p_{attack}, the malicious router attacks more packets, but the attack is less likely to go undetected leading to lower p_{avoid}.

Figure 10 shows the derivation of the probability of complete avoidance. We illustrate the derivation through the example in Figure 2.(b). In this example, we assign 6 hash values to a router as the sampling range out of 31 different hash values.Thus, N_{small} = 6, and N_{total} = 31. The 4-th router, r_4, is a malicious router on a 7-hop path. r_4 selects

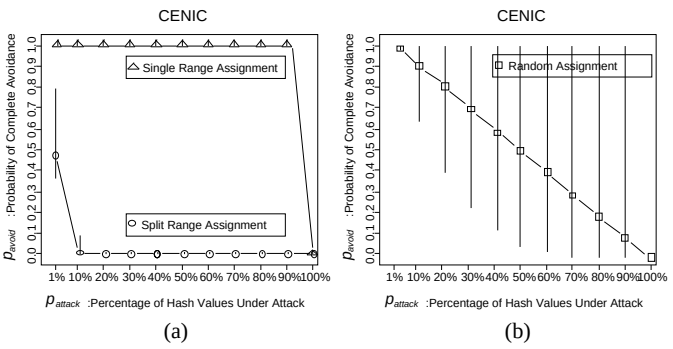
h_i: the i-th hash value among N_{total} hash values of $h_{selection}()$
$\mathcal{H}(i)$: a set of N_{small} hash values in the sampling range assigned to r_i
$\mathcal{H}_{enclose}(m) = \{h_i : h_i \in \mathcal{H}(l), h_i \in \mathcal{H}(r), 1 \leq l < m < r \leq T\}$: a set of hash values assigned to any pair of routers that enclose a malicious router, r_m, on a T-hop path
$N_{enclose}(m)$: the number of hash values in $\mathcal{H}_{enclose}(m) \cup \mathcal{H}(m)$
$\mathcal{H}_{attack}(m)$: a set of hash values chosen by the malicious router r_m
$N_{attack} = p_{attack} \times N_{total}$: The number of hash values in $\mathcal{H}_{attack}(m)$
P(complete avoidance of r_m) = $p_{avoid}(m)$ = $P(\mathcal{H}_{attack}(m) \cap \mathcal{H}_{enclose}(m) = \phi) = C_{N_{attack}}^{N_{total}-N_{enclose}(m)} / C_{N_{attack}}^{N_{total}-N_{small}}$

Figure 10. The calculation of the probability of complete avoidance, p_{avoid}.

p_{attack} = 20% of the N_{total} hash values ($31 \times 0.2 \approx 6$ hash values) out of $N_{total} - N_{small} = 31 - 6 = 25$ hash values in $\{h : 1 \leq h \leq 31\} - \{25, 15, 28, 14, 12, 8\}$. r_4 does not choose the 6 hash values from its own sampling range, $\{25, 15, 28, 14, 12, 8\}$, since these values can also be assigned to other routers. The number of possible choices is thus C_6^{31-6}. r_4 attacks packets whose hashes fall in the 6 chosen hash values, $\mathcal{H}_{attack}$. If any one of the 6 hash values in $\mathcal{H}_{attack}$ is in $\mathcal{H}_{enclose} = \{30, 5, 9, 27, 4, 11, 7, 17, 20\}$, the attack will be detected - the attack on the hash value 30, 5, 9, 27, 4, 11, 7, 17, or 20 will be detected by $\{r_1, r_5\}$, $\{r_1, r_6\}$, $\{r_1, r_7\}$, $\{r_2, r_5\}$, $\{r_2, r_6\}$, $\{r_2, r_7\}$, $\{r_3, r_5\}$, $\{r_3, r_6\}$, or $\{r_3, r_7\}$, respectively. The probability of complete avoidance of r_4, which is the probability that all of the 6 chosen hash values in $\mathcal{H}_{attack}$ are not in $\mathcal{H}_{enclose}$, is $C_6^{31-6-9}/C_6^{31-6} \approx 0.045$.

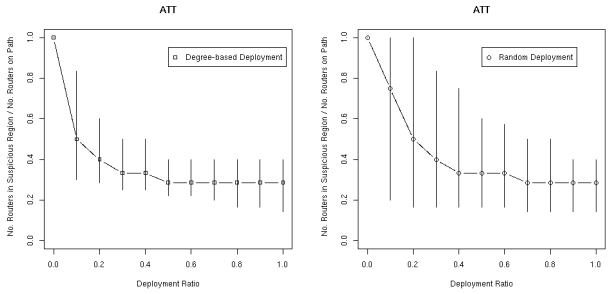
We assign sampling ranges using three different methods, Split Range Assignment (Figure 3), random assignment and Single Range Assignment. In the random assignment, we assign $N_{small} = N_{router} - 1$ hash values to a router. The hash values are randomly chosen out of $N_{total} = N_{small}/p_{samp}$ hash values of $h_{selection}()$. For each node in a topology, we compute the probability of complete avoidance considering all the shortest paths where the node is present. We run 100 simulations with the sampling ratio p_{samp} = 1/1000.

In Figure 11, we only show the results from the topology of CENIC as the results from other topologies are very similar. The left graph shows results for both Single Range Assignment and Split Range Assignment, and the right one shows results for random assignment. The median, 10th-percentile and 90th-percentile values of the ratio are shown. In Single Range Assignment, unless a malicious router attacks all the packets, the malicious router can avoid detection with 100% probability. Random assignment can reduce p_{avoid} but not significantly. In Split Range Assignment, p_{avoid} becomes negligible as p_{attack} approaches 10%. Thus, attacks on more than 10% of packets can hardly circumvent SATS. An attack with p_{attack} less than 1% has almost 50% chance of being undetected. However, the resulting attack is limited. The attacker has to significantly limit its ability to attack only a small por-



(a) (b)

Figure 11. Probability of Complete Avoidance.



(a) (b)

Figure 12. Suspicious region ratio.

tion of packets. In addition, we can further reduce p_{avoid} by periodically re-launching new sampling ranges. Let us assume that the range assignment in each new launch of sampling ranges is independent of other launches. A malicious router also changes its target hash values, $\mathcal{H}_{attack}$, independently in each new launch. Then, the probability that the malicious router avoids detection in n consecutive launches of sampling ranges is ${p_{avoid}}^n$. After 5 such re-launches ($n = 5$), the probability of complete avoidance decreases to $(50\%)^5 \approx 3\%$ when p_{attack}=1%.

6.2. Incremental Deployment

In this section, we discuss the size of a suspicious region detected by SATS. We aim to answer the following questions: What is the size of a suspicious region relative to the diameter of a network? does SATS work in different topologies? What is the incremental deployment property of SATS? Is it necessary to have complete deployment before seeing most of the benefits? Is a sophisticated deployment scheme needed to put SATS on strategic routers?

We evaluate two deployment methods: random and degree-based. The degree-based method chooses routers with the highest number of neighbors first during deployment. It gives priority to securing the more critical routers. We also vary the fraction of routers using SATS, from 0.1 to 1, in 0.1 increments. 80 different combinations of parameters are based on 10 deployment ratios $\times$ 2 deployment methods $\times$ 4 real topologies. 100 simulation runs are made in each combination.

Figure 12 shows the evaluation results only from the topology of ATT as the results from other topologies are very similar. The right graph show results for random deployment and the left one degree-based. The median, 10th-percentile and 90th-percentile values of the ratio are shown. The y-axis plots the suspicious region ratio, computed as the number of nodes in the pinpointed suspicious region divided by the total number of nodes on the shortest path. A node without SATS is considered to be suspicious. The median, 10th-percentile and 90th-percentile values are shown. The x-axis plots the deployment ratio.

All graphs show "knees" in the performance curves: after reaching a certain deployment ratio, from 0.1 to 0.4 depending on the topology and deployment method, the curves flatten out. In other words, we start seeing most of the benefits of SATS before complete deployment on all routers in a network. These "knees" happen because of the hierarchical nature of networks. There are a handful of well-connected routers that are on most of the shortest paths between pairs of edge routers. Once these routers deploy SATS, the size of a suspicious region can be reduced dramatically.

We see that variability is much lower for the degree-based deployment. In particular, the 90th-percentile values of the suspicious region ratio are much lower for all the topologies. The results also show that a practical yet extremely simple method of deploying SATS on routers with the largest degrees first can yield significant improvements over a random method. A sophisticated deployment method may not be necessary. It is more important to secure first the routers with higher degrees of connectivity. We focus on the results of the degree-based deployment below.

When the deployment ratio is zero, the suspicious region ratio is 1 since all of the nodes on the shortest path is considered as suspicious. With half of the nodes in a topology deploying SATS, the median values of the suspicious region ratio range from 0.3 to 0.4, depending on the topology. In other words, we can pinpoint 3 to 4 routers as suspicious routers on a 10-hop path in the presence of malicious routers with the deployment ratio of 0.5.

7. Implementation Costs

In this section, we discuss the implementation costs associated with SATS in a fictional network. This network has 100 routers (N_{router}) and 300 links. The link rate is 10 Gbps and all links are 95% utilized. All packets are 500B long. The sampling rate p_{samp} is 1/1000. N_{samp}, the number of sampled packets per second on a link, is $10Gbps \times 95\% \times$

$0.001/(500 \times 8) = 2375$. Let us assume samples are not sent to the backend engine until they fill up a 1500B packet. Less than 1500B are stored on each router until they are flushed to the backend engine. The computational cost on a router is dominated by $h_{selection}()$, which is computed for each packet. Modular arithmetic of $h_{selection}()$ can be implemented using simple integer arithmetic in hardware or software. Current technology can compute such a function for each packet that arrives at 20 Gbps or even higher [2].

To evaluate the quantity of data sent to the backend engine, we first derive the size of a report packet. For a sampled packet, we report raw header information of the packet such as the source, the destination IP addresses and the TTL field. We also add a 26-bit label of the packet [2] and the hash value where the packet is sampled. Since the total number of hash values of $h_{selection}()$ is about 10^5 as shown in Section 4, we need $\lceil \log_2(10^5) \rceil = 17$ bits to represent a hash value. Thus, the size of a report packet $l_{report} = (20$ (IP header)$+4+4+1) \times 8 + 26 + 17 = 275$ bits. Finally, each link sends $N_{samp} \times l_{report} \approx 653Kbps$ to the backend engine. With 300 such links, the total report traffic consumes less than 2% of the link rate. Note that if multiple samples are reported in a single IP packet, we can further reduce the communication overhead.

In the backend engine, we maintain $T-1$ counters for each router that is traversed by each flow, where T is the length of the path. With 1.7 million flows [4] and network diameter of 20, we need up to $1.7 \times 20(20-1) \times p_{samp} \approx 0.6$ million counters, which corresponds to several MB of memory. For each sampled packet p, the backend engine reads counter values of p's flow from memory. The memory of a flow is indexed by using a hash function $h_{index}(x) = x \bmod 2^{20}$. $h_{index}()$ has a 64-bit input (source and destination IP pair) and a 20-bit output to accommodate a million flows. One of the counters is incremented and the value is compared with another counter value in the same flow. The new counter values are then stored back in memory. Load and store operations in DRAM take around 60 nanoseconds each and it takes up to 100 cycles to compute $h_{index}()$. Thus, such operations can be processed by a current 1GHz processor in less than 300 nanoseconds, which is far less than the time we have for each sample, $1/(N_{samp} \times N_{router}) \approx 4.2$ microseconds.

8. Conclusions

We presented SATS, a data plane method to detect malicious routers. SATS uses Split Range Assignment, which varies sampling ranges on each router, to maintain integrity of the packet sampling process. Using simulation, we showed that the probability that a malicious router can avoid detection is less than 5% and even lower when new sampling ranges are periodically reloaded onto the routers. SATS can be deployed incrementally as most of the benefits of SATS can be achieved when only 10% to 40% of the routers in a network deploy SATS depending on the topology. Although SATS is initially designed for a single administrative domain, we believe it can be extended to multiple non-cooperative domains by sharing common hash values in border routers, without revealing internal traffic measurement to an external network. We leave this as future work.

References

[1] I. Avramopoulos and J. Rexford. Stealth Probing: Securing IP Routing through Data-Plane Security. Technical Report TR-730-05, Princeton University Department of Computer Science, June 2005.

[2] N. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. *IEEE/ACM Transactions on Networking*, 9(3):280–292, June 2001.

[3] N. Duffield and M. Grossglauser. Trajectory sampling with unreliable reporting. In *IEEE INFOCOM*, Hong Kong, March 2004.

[4] C. Estan. *Internet Traffic Measurement: What's going on in my network?* PhD thesis, University of California, San Diego, Oct. 2003.

[5] B. Greene and K. Houle. Isp security – real work techniques ii. NANOG 26 presentation, October 2002. http://www.nanog.org/mtg-0210/ispsecure.html.

[6] *Georgia Tech Internet Topology Model*. http://www-static.cc.gatech.edu/projects/gtitm.

[7] A. Herzberg and S. Kutten. Early detection of message forwarding faults. *SIAM J. Comput.*, 30(4):1169–1196, 2000.

[8] J. Huges, T. Aura, and M. Bishop. Using Conservation of Flow as a Security Mechanism in Network Protocols. In *IEEE Symposium on Security and Privacy*, May 2000.

[9] S. Kent, C. Lynn, and K. Seo. Secure border gateway protocol (secure-bgp). *IEEE Journal on Selected Areas in Communications*, 18(4), April 2000.

[10] S. Lee, T. Wong, and H. S. Kim. Secure Split Assignment Trajectory Sampling: A Malicious Router Detection System. Technical Report CMU-CK-05-01, Carnegie Mellon University, July 2005. Available at `http://www.ece.cmu.edu/~sihyungl/CMU-CK-05-01-SATS.pdf`.

[11] A. T. Mizrak, Y. Cheng, K. Marzullo, and S. Savage. Fatih: Detecting and isolating malicious routers. In *Dependable Systems and Networks*, Yokohama, Japan, June 2005.

[12] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *ACM Symposium on Theory of Computing*, Seattle, WA, May 1989.

[13] V. Padmanabhan and D. Simon. Secure traceroute to detect faulty or malicious routing. *Sigcomm Computer Communications Review*, 33(1):77–82, Jan 2003.

[14] B. Schneier. Cisco Harasses Security Researcher, July 2005. http://www.schneier.com/blog/archives/2005/07/cisco_harasses.html.

[15] L. Subramanian, V. Roth, I. Stoica, S. Shenker, and R. Katz. Listen and whisper: Security mechanisms for bgp. In *USENIX/ACM Symposium on Networked System Design and Implementation (NSDI)*, San Francisco, CA, March 2004.

[16] D. Zhu, M. Gritter, and D. R. Cheriton. Feedback based routing. *Sigcomm Computer Communications Review*, 33(1):71–76, Jan 2003.