Listen and Whisper: Security Mechanisms for BGP

Lakshminarayanan Subramanian*, Volker Roth+, Ion Stoica*, Scott Shenker*+, Randy H. Katz*

*U.C. Berkeley +ICSI, Berkeley

Abstract

BGP, the current inter-domain routing protocol, assumes that the routing information propagated by authenticated routers is correct. This assumption renders the current infrastructure vulnerable to both accidental misconfigurations and deliberate attacks. To reduce this vulnerability, we present a combination of two mechanisms: *Listen* and *Whisper*. Listen passively probes the data plane and checks whether the underlying routes to different destinations work. Whisper uses cryptographic functions along with routing redundancy to detect bogus route advertisements in the control plane. These mechanisms are easily deployable, and do not rely on either a public key infrastructure or a central authority like ICANN.

The combination of Listen and Whisper eliminates a large number of problems due to router misconfigurations, and restricts (though not eliminates) the damage that deliberate attackers can cause. Moreover, these mechanisms can detect and contain isolated adversaries that propagate even a few invalid route announcements. Colluding adversaries pose a more stringent challenge, and we propose simple changes to the BGP policy mechanism to limit the damage colluding adversaries can cause. We demonstrate the utility of Listen and Whisper through real-world deployment, measurements and empirical analysis of their worst-case behaviors. For example, a randomly placed isolated adversary, in the worst case can affect reachability to only 1% of the nodes.

1 Introduction

The Internet is a collection of autonomous systems (AS's), numbering more than 14,000 in a recent count. The inter-domain routing protocol, BGP, knits these autonomous systems together into a coherent whole. Therefore, BGP's resilience against attack is essential for the security of the Internet. BGP currently enables peers to transmit route announcements over authenticated channels, so adversaries cannot impersonate the legitimate sender of a route announcement. This approach, which verifies *who* is speaking but not *what* they say, leaves the current infrastructure extremely vulnerable to both unintentional misconfigurations and deliberate attacks. For example, in 1997 a simple misconfiguration in a customer router caused it to advertise a short path to a large number of network prefixes, and this resulted in a massive black hole that disconnected significant portions of the Internet [14].

To eliminate this vulnerability, several sophisticated BGP security measures have been proposed, most notably S-BGP [24]. However, these approaches typically require an extensive cryptographic key distribution infrastructure and/or a trusted central database (*e.g.,* ICANN [3]). Neither of these two crucial ingredients are currently available, and so these security proposals have not moved forward towards adoption.[1] In this paper we abandon the goal of "perfect security" and instead seek "significantly improved security" through more easily deployable mechanisms. To the end we propose two measures, Listen and Whisper, that require neither a public key distribution nor a trusted centralized database. We first describe the threat model we address and then summarize the extent to which these mechanisms can defend against those threats.

1.1 Threat Model

The primary underlying vulnerability in BGP that we address in this paper is the ability of an AS to create *invalid* routes. There are two types of invalid routes:

1. **Invalid routes in the Control plane:** This occurs when an AS propagates an advertisement with a fake AS path (i.e., one that does not exist in the Internet topology), causing other AS's to choose this route over genuine routes. A single malicious adversary can divert traffic to pass through it and then cause havoc by, for example, dropping packets (rendering destinations unreachable), eavesdropping (violating privacy), or impersonating end-hosts within the destination network (like Web servers etc.).
2. **Invalid routes in the Data Plane:** This occurs when a router forwards packets in a manner inconsistent with the routing advertisements it has received or propagated; in short, the routing path in the data plane does not match the corresponding routing path advertised in the control plane. Mao et al. [26] show that for nearly 8% of Internet paths, the control plane and data plane paths do not match. The prevalence of such a high mismatch ratio motivates the need for separately verifying the correctness of routes in the data plane and not merely focusing on the control plane.

Invalid routes can be caused by either accidental misconfigurations or deliberate attacks. Misconfigurations occur in several forms ranging from buggy configuration scripts to human errors. In the control plane, Mahajan et al. [25] infer

[1] There is much debate about whether their failure is due to the lack of a PKI and trusted database, or onerous processing overheads, or other reasons. However, the fact remains that neither of these infrastructures are available, and any design that requires them faces a much higher deployment barrier.

that misconfigurations produce invalid route announcements to roughly $200 - 1200$ prefixes every day (roughly $0.2 - 1\%$ of the prefix entries in a typical routing table). Stale routes (not propagating new announcements) and forwarding errors at a router (*e.g.,* lack of forwarding entry) are two other data plane misconfigurations causing invalid routes. While AS's might act in malicious ways on their own, the biggest worry about deliberate attacks comes from adversaries who break into routers. Routers are surprisingly vulnerable; some have *default passwords* [10, 34], others use standard interfaces like telnet and SSH, and so routers share all their known vulnerabilities. For our purposes in this paper, the only difference between a misconfiguration and an attack is that attackers can take active countermeasures (by, for instance, spoofing responses to various probes) while misconfigured routers don't. Deliberate attacks can involve an *isolated adversary* (i.e., a single compromised router) or *colluding adversaries* (i.e., a set of compromised routers). It is particularly difficult to secure against colluding attackers.

The spectrum of problems we address in this paper can be described, in order of increasing difficulty, as *misconfigurations, isolated adversaries* and *colluding adversaries*. We now describe the extent to which Listen and Whisper provide protection against these threats.

1.2 Level of Protection

Listen detects invalid routes in the data plane by checking whether data sent along routes reaches the intended destination. Whisper checks for consistency in the control plane. While both these techniques can be used in isolation, they are more useful when applied in conjunction. The extent to which they provide protection against the three threat scenarios can be summarized as follows:

Misconfigurations and Isolated Adversaries: Whisper guarantees *path integrity* for route advertisements in the presence of misconfigurations or isolated adversaries; *i.e.,* any invalid route advertisement due to a misconfiguration or isolated adversary with either a fake AS path or with any of the fields of the AS path being tampered (*e.g.,* addition, modification or deletion of AS's) will be detected. Path integrity also implies that an isolated adversary cannot exploit BGP policies to create favorable invalid routes. In addition, Whisper can identify the offending router if it is propagating a significant number of invalid routes. Listen detects reachability problems caused by errors in the data plane, but is only applicable for destination prefixes that observe TCP traffic.

However, none of our solutions can prevent malicious nodes already on the path to a particular destination from eavesdropping, impersonating, or dropping packets. In particular, countermeasures (from isolated adversaries already along the path) can defeat Listen's attempts to detect problems on the data path.

Colluding Adversaries: None of our techniques can prevent two colluding nodes from pretending there is a direct link between them by tunneling packets. Moreover, colluding nodes can exploit the current usage of BGP policies to create large scale outages without being detectable by either Listen or Whisper. To deal with this problem, we suggest simple modifications to the BGP policy engine which in combination with Whisper can largely restrict the damage that colluding adversaries can cause. In the absence of complete knowledge of the Internet topology, these two problems also exist in the case of heavy-weight security solutions like Secure BGP [23].

The rest of the paper is organized as follows. In Section 2, we discuss related work. In Sections 3 and 4, we describe the whisper and the listen protocols. In Section 5, we present our implementation of Listen and Whisper. In Section 6, we will evaluate several aspects of Listen and Whisper using real-world experiences, empirical evaluation and security analysis. In Section 7, we discuss the case of colluding adversaries and finally present our conclusions in Section 8.

2 Related Work

In this section, we will present related work as well as try to motivate our work in comparison to previous approaches to this problem. We classify related work based on the threat model.

2.1 Misconfigurations

Traditional approaches to detecting misconfigurations involves correlating route advertisements in the control plane from several vantage points [25, 35]. They identify two forms of misconfigurations: origin and export misconfigurations. In an origin misconfiguration, the destination AS corresponding to a prefix is wrongly specified. An export misconfiguration occurs when a router violates a BGP export policy and forwards certain routes to a neighboring AS that it is not supposed to forward. From our definition of an invalid route, an export misconfigured route is still a valid route. Our paper does not deal with this type of misconfiguration. One limitation with analyzing BGP streams is: *the lack of knowledge of the Internet topology*. Since the topology is not known, these techniques can pinpoint invalid routes only when the destination AS is wrongly specified.

Mao *et al.* [26] build an AS-traceroute tool to detect the AS path in the data plane. Using this tool, one can detect several forms of invalid routes in the data plane. While this tool is useful for diagnostic purposes once a problem is detected, one cannot pro-actively use it to search for invalid routes since it actively probes the data paths.

Within the context of feedback based routing, Zhu *et al.* [36] proposed a data plane technique based on passive and active probing. The passive probing aspect of this work shares some similarities to our Listen method. Padmanabhan *et al.* [29] propose a secure variant of `traceroute` to test the correctness of a route. This mechanism requires both a PKI and prior distribution of cryptographic keys to the participating AS's to ascertain the integrity and authenticity of traceroute packets.

2.2 Dealing with Adversaries

Techniques dealing with adversaries can be classified as *Key distribution based* or *Non-PKI based.*

Key-distribution based: One class of mechanisms builds on cryptographic enhancements of the BGP protocol, for instance the security mechanisms proposed by Smith *et al.* [32], Murphy *et al.* [27], Kent *et al.* [24], and recent work on *Secure Origin BGP* [28]. All these protocols make extensive use of digital signatures and public key certification. S-BGP roughly uses fifteen different certificate types for route verification [31]. More lightweight approaches based on cryptographic hash functions have been proposed *e.g.*, by Hu *et al.* [20, 22] in the context of secure routing in ad hoc networks. One variant of the Whisper protocol is conceptually similar to their work. However, their mechanisms require prior secure distribution of hash chain elements.

Why not use a PKI-based infrastructure? Public key infrastructures impose a heavy technological and management burden, and have received a fair share of criticism *e.g.,* by Davis [16], Ellison and Schneier [17]. The PKI model has been criticized based on technical grounds, on grounds of a lack of trust and privacy, as well as on principle [16, 17, 15]. Building an Internet wide PKI infrastructure is a major project with huge costs and a high risk of failure. Secure-BGP, despite the push by a major tier-1 ISP, has been deployed only by a very small number of ISPs after 5 years (though an IETF working group on Secure-BGP exists).

Non-PKI approaches: Non-PKI based solutions offer far less security in the face of deliberate attacks. Some of these mechanisms assume the existence of databases with up to date authoritative route information against which routers verify the route announcements that they receive. The *Internet Routing Registry* [4] and the *Inter-domain Route Validation Service* proposed by Goodell *et al.* [19] belong to this category. Here, the problem is to ascertain the authenticity, completeness, and availability of the information in such a database. First, ISPs only reluctantly submit routing information because this may disclose local policies that the ISPs regard as confidential (this is not an issue in [19] because each AS keeps its own route validation service). Second, the origin authentication of the database contents again demands a public key infrastructure. Third, access to such databases relies on the very infrastructure that it is meant to protect, which is hardly an ideal situation.

3 Whisper: Control Plane Verification

In this section, we will describe the whisper protocol, a control plane verification technique that proposes minor modifications to BGP to aid in detecting invalid routes from misconfigured or malicious routers. In this section, we restrict our discussion to the case where an isolated adversary or a single misconfigured router propagates invalid routes. We will discuss colluding adversaries in Section 7.

The Whisper protocol provides the following properties in the presence of isolated adversaries:

1. Any misconfigured or malicious router propagating an invalid route will always a trigger an alarm.
2. A single malicious router advertising more than a few invalid routes will be detected and the effects of these spurious routes will be contained.

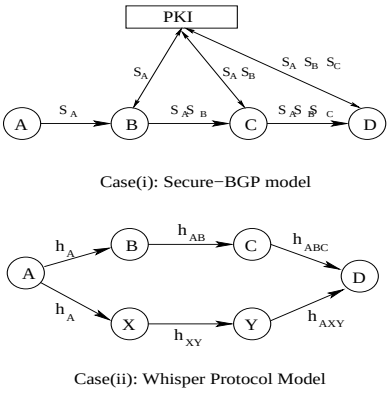

Figure 1: Comparison of the security approach of Whisper protocols with Secure BGP

3.1 Triggering Alarms vs Identification

The main distinction between our approach and a PKI-based approach is the concept of *triggering alarms* as opposed to *identifying the source of problems*. In Secure-BGP, a router can verify the correctness of a single route advertisement by contacting a PKI and a central authority to test the validity of the signatures embedded in the advertisement . For example, in Figure 1 (Case(i)), each AS X appends an advertisement with a signature S_X generated using its public key. Another AS can use a PKI to check whether S_X is the correct signature of X. In this case, any misconfigured/malicious AS propagating an invalid route will not be able to append the correct signatures of other AS's and can be *identified*.

Without either of these two infra-structural pieces, a router cannot verify a single route advertisement in isolation. The Whisper model is to consider two different route advertisements to the same destination and check whether they are consistent with each other. For example, in Figure 1 Case(ii), each route advertisement is associated with a signature of an AS path. AS D receives two advertisements to destination A and can compare the signatures h_{ABC} and h_{AXY} to check whether the routes (C, B, A) and (Y, X, A) are consistent. When two routes are detected as *inconsistent*, the Whisper protocol can determine that at least one of the routes is invalid. However, it cannot clearly pinpoint the source of the invalid route. Upon detecting inconsistencies, the Whisper protocol can *trigger alarms* notifying operators about the existence of a problem. This method is based on the composition of well-known principles of *weak authentication* as discussed by Arkko and Nikander [11].

The whisper protocol does not require the underlying Internet topology to have multiple disjoint paths to every destination AS. As long as an adversary propagating an invalid route is not on every path to the destination, whisper will have two routes to check for consistency: (a) the genuine route to the destination; (b) invalid path through the adversary.

3.2 Route Consistency Testing

A *route consistency test* takes two different route advertisements to the same destination as input and outputs *true* if

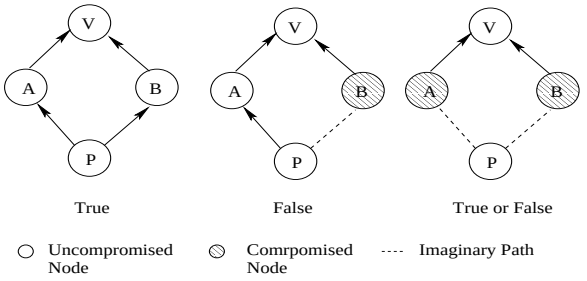

Figure 2: Different outcomes for a route consistency test. In all these scenarios, the verifying node is V. The verifying node checks whether the two routes it receives to destination P are consistent with each other.

the routes are consistent and outputs *false* otherwise. Consistency is abstractly defined as follows:

1. If both route announcements are valid then the output is *true*.
2. If one route announcement is valid and the other one is invalid then the output is *false*.
3. If both route announcements are invalid then the output is *true* or *false.*

The key output from a route consistency test is *false*. This output unambiguously signals that *at least one* of the two route announcements is invalid. In this case, our protocols can raise an alarm and flag both the suspicious routes as potential candidates for invalid routes. If the consistency test outputs true, both the routes could either be valid or invalid. Figure 2 depicts the outcomes of a route consistency test for various examples of network configurations.

We will now describe two whisper consistency tests, namely *Weak Split Whisper* and *Strong Split Whisper (SSW)*, of increasing complexity offering different security guarantees. We primarily use Weak Split, a simple hash chain based construction, to motivate the construction of SSW. All the results presented in this paper are based on the strong split variant. SSW is the variant that offers *path integrity* in the presence of misconfigurations or isolated adversaries.

Conceptually, both these constructions introduce a *signature* field in every BGP UPDATE message which is used for performing the route consistency test. There are three basic operations that are allowed on the signature field:

1. *Generate-Signature:* The origin AS (the originator of a route announcement) of a destination prefix generates a signature and initializes this field in the BGP UPDATE message and forwards it to its neighbor. The origin AS uses different initial signatures for every prefix it owns.
2. *Update-Signature:* Every intermediary AS that is not the origin of a destination prefix is required to update the signature field using a cryptographic hash function. This operation is only performed by one router in every AS (typically at the entry point of an AS).
3. *Verify-Signature:* Any intermediary router that receives two different routes (with different AS paths) can compare whether the signatures in the two different routes are consistent with each other.

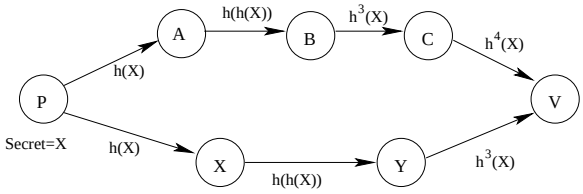

Figure 3: Weak-Split construction using a globally known hash function $h()$

The path integrity property requires the whisper protocol to satisfy two properties: (a) a malicious adversary should not be able to reverse engineer the signature field of an AS path; (b) any modification to the AS path or signature field in an advertisement should be detected as an *inconsistency* when tested with a valid route to the same destination.

3.2.1 Weak Split Whisper

Figure 3 illustrates the weak-split construction using a simple example topology. Weak-Split whisper is motivated by the hash-chain construction used by Hu *et al.* [21, 20] in the context of ad-hoc networks. The key idea is as follows: The origin AS generates a secret x and propagates $h(x)$ to its neighbors where $h()$ is a globally known one-way hash function. Every intermediary AS in the path repeatedly hashes the signature field. An AS that receives two routes r and s of AS hop lengths k and l with signatures y_r and y_s can check for consistency by testing whether:

$$h^{k-l}(y_s) = y_r$$

The security property that the weak-whisper guarantees is: *An independent adversary that is N AS hops away from an origin AS can propagate invalid routes of a minimum length of $N-1$ without being detected as inconsistent.* An AS that is N hops away from the origin knows the value $h^N(x)$ but cannot compute $h^k(x)$ for any $k < N$ since $h()$ is a one-way hash function. Such an AS also is not supposed to reveal its hash value to other nodes (unless the AS colludes with other AS's). However, the adversary can forward any fake path of length $N-1$ and forward $h^N(x)$ along with the path.

Hence, weak-split whisper does not provide strong forms of security guarantees. In particular, it cannot ensure path integrity i.e. a malicious AS could modify the AS numbers of a path without affecting the AS path length.

3.2.2 Strong Split Whisper

The strong split whisper protocol uses a more sophisticated cryptographic check and can provide *path integrity* in the presence of independent adversaries *i.e.,* If an adversary removes or changes any entry in the AS path, the strong split whisper will always detect an inconsistency.

Figure 4 shows a construction of the basic SSW using the RSA mechanism. We use a minor modification of the illustrated example. We will elaborate the three basic operations for this protocol:

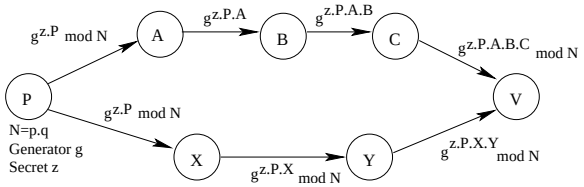

Figure 4: Basic Strong-Split construction using exponentiation under modulo N where $N = p.q$, a product of two large primes.

1. *generate-signature:* The origin AS computes three basic parameters:. N, g, z. N is chosen as $p \times q$ where p and q are two large primes of the form $2p' + 1$ and $2q' + 1$ where p' and q' are also prime. It then computes a generator g in the prime group Z_p and Z_q. Finally, it chooses a random number z and computes $g^z \text{mod} N$. The signature generated is a tuple $(N, g^z \text{mod} N)$. While the origin AS publicly announces N, only it knows the prime factors of N. Similar to RSA, we rely on the fact that an adversary cannot factor N to determine its prime factors.
2. *update-signature:* Every AS is associated with a unique AS number which is specified in the path. Let AS A that receive an advertisement from a neighboring AS with a signature (N, y) where y is of the form $g^D \text{mod} N$ for some value of D. AS A updates this signature to $(N, y^A \text{mod} N)$. In other words, the AS exponentiates using its AS number. In Figure 4, the route announcement contains an AS path P, A, B, C, the corresponding signature of the route is $(N, g^{z.P.A.B.C} \text{mod} N)$.
3. *verify-signature:* We will describe verify-signature using the example in Figure 4. The verifier,V, receives two signatures (N, s_1) and (N, s_2) where $s_1 = g^{z.P.A.B.C} \text{mod} N$ and $s_2 = g^{z.P.X.Y} \text{mod} N$. Given these values and the corresponding AS paths, the verifier checks whether:

$$s_1^{X.Y} = s_2^{A.B.C}$$

If so, the routes are said to be consistent.

SSW is similar to the MuHASH construction proposed by Bellare *et al.* [12] for incrementally hashing signatures. A formal proof of the security guarantees offered by MuHASH is also applicable in our context to show that SSW offers path integrity. The key observation with our construction is: given N and given $g^x \text{mod} N$, an adversary cannot compute $x^{-1} \text{mod} N$ and hence cannot remove the signature of previous nodes in the AS path.

This construction has three problems: (a) an adversary can permute entries in a path due to *commutative* property of multiplication *i.e.,* $A.B = B.A$; (b) the *factoring* property *i.e.,* $8 = 4 \times 2$ implies an AS path $(2, 4)$ can be replaced by (8); (c) More importantly, an adversary can *add AS's to the AS path* without being detected.

Preventing commutativity and factoring: To prevent commutativity and factoring problems, we define a *pseudo-AS number* for every AS which depends on the position of the AS in a given AS path. If an AS X appears in position p in the AS path, the following function

$$f(X, p) = 2^{16} \times p + X$$

will produce unique values for all AS's in different positions in an AS path (since 16 bits are sufficient to express AS numbers). To avoid the problem of commutativity, an AS updates a signature using $f(X, p)$ instead of using its AS number X.

To avoid the factoring problem, we use prime numbers. Given a number y, one can determine the $q(y)$ as the y^{th} lowest prime number. Prime numbers are not factorable and these numbers can be precomputed. Hence, given an AS X appearing at position p, we use the exponent to be $X' = q(f(X, p))$ to avoid both commutativity and factoring problems. We refer to X' as the *psuedo-AS number* of AS X when it appears in position p. The pseudo-AS numbers for a given AS are computable by other routers as well. Hence, we only use pseudo AS numbers for computing the signature but do not change AS numbers in the AS path.

Preventing Addition of new AS numbers: The key to preventing an adversary from adding AS numbers is to associate a *link identifier* to represent an AS link between two AS's. If AS A forwards a route to AS B, let $link(A, B)$ be a uniquely computable identifier which is a function of the AS numbers A and B. An AS A that received an advertisement (N, y) should propagate the advertisement with the signature:

$$(N, y^{A' \times link(A,B)})$$

where A' is the pseudo-AS numbers of A. Since the identifier $link(A, B)$ is added to the signature by A, B cannot remove this portion from the signature. This implies B cannot convert an AS path (B, A) to (B, C, A). However, if B adds an AS at the end of a path (*e.g.,* (C, B, A)), then the neighbor receiving the advertisement will notice that the neighbor it received the announcement from (*i.e.,* B) does not match the first AS in the path (*i.e.,* C). Hence it will not accept the announcement. One simple way to define a link identifier is:

$$link(A, B) = 2^{32} \times A' + B'$$

where A' and B' are the pseudo-AS numbers of A and B. $link(A, B)$ will be unique for all AS pairs A, B. Note that pseudo-AS numbers are always less than 2^{32} since $f(X, p) < 2^{21}$ for all AS paths less than 32 hops in length.

Generalized SSW construction: In this section, we only described the SSW construction using the basic RSA group structure. Alternatively, one can build SSW using elliptic curve cryptography [13]. The main distinction between RSA and ECC is the number of bits necessary for the signature field. While RSA requires 1024 bit signatures, ECC only requires 256 bits to provide the same level of security.

3.3 Containment: Penalty Based Filtering

Using route consistency testing, we suggest *penalty based filtering*, a simple containment strategy against independent adversaries. The strategy works as follows: A router counts across destinations how often an AS appears on an invalid

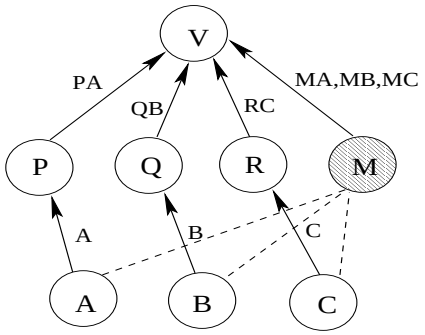

Figure 5: Detecting Suspicious AS's: In this example, M is a malicious AS that propagates 3 invalid routes to 3 different destinations A,B,C. The AS paths in the routes propagated are indicated along the links.

route, and assigns this count as a *penalty* value for the AS. The more destinations an adversary affects the higher becomes its penalty and the clearer it stands out from the rest. In penalty based filtering, an AS applies this strategy: *Choose the route to a destination with the lowest penalty value.*

We will motivate this using an example. In Figure 5, we show a simple scenario where M is a malicious node that propagates 3 invalid route announcements with AS paths MA, MB, MC. The verifier V receives three genuine announcements PA, QB and QC. V notices that the announcement pairs (PA, MA), (QB, MB), (RC, MC) are not consistent with each other. Hence it assigns the penalty values 3, 1, 1 and 1 to AS's M, P,Q and R. By choosing the minimum penalty route, the verifier can avoid the invalid routes through M.

One key assumption used in this technique is: *The identity of an AS propagating invalid routes is always present in the AS path attribute of the routes.* The identity of every AS is verified by the neighboring AS which receives the advertisement. For example, Zebra's BGP implementation [2] explicitly checks for this constraint for every announcement it receives. BGP should use shared keys across peering links to avoid man in the middle attacks i.e. if X and Y peer with each other, an adversary Z should not be able to hijack the peering connection by pretending as Y.

However, one must exercise caution in applying penalties. If we assume that there is only one isolated adversary then the penalty of the adversary will always be at least as high as the penalty of any other AS. If multiple adversaries advertise suspicious routes to different destinations then they may raise the penalty of innocent AS's higher than their individual penalties. This may happen even if the adversaries do not actively collude.

Penalties should be viewed as a reasonable first response until close investigation reveals the cause of alarms. Penalties are useful for detecting and containing isolated adversaries but is not a good security measure against colluding adversaries. In the absence of any additional knowledge about two paths, one can exercise penalties in choosing the path. We will evaluate the performance of penalty based filtering in the presence of isolated adversaries in Section 6.2.

4 Listen: Data Plane Verification

In this section, we will present the Listen protocol, a data plane verification technique that detects reachability problems in the data plane. Reachability problems can occur due to a variety of reasons ranging from routing problems to misconfigurations to link failures. Listen primarily signals the existence of such problems as opposed to identifying the source or type of a problem.

Data plane verification mechanisms are necessary in two contexts: (a) connectivity problems due to stale routes or forwarding problems are detectable only by data plane solutions like Listen. (b) Blackhole attacks by malicious adversaries already present along a path to a destination. However, proactive malicious nodes can defeat any data plane solution by impersonating the behavior of a genuine end-hosts. The attractive features of Listen are: (a) passive (b) a standalone solution that can be incrementally deployed without any modifications to BGP; (c) quick detection of reachability problems for popular prefixes; (d) low overhead.

Listen relies on passively observing TCP connections to verify the correctness of routes. However, the basic form of the protocol described in this section is vulnerable to port scanners generating many incomplete connections. In Section 6.3, we use propose defensive measures against port scanners and motivate them using real world measurements.

4.1 Listening to TCP flows

The general idea of Listen is to monitor TCP flows, and to draw conclusions about the state of a route from this information. The forward and reverse routing paths between two end-hosts can be different. Thus we may observe packets that flow in only one direction. We say that a TCP flow is *complete* if we observe a SYN packet followed by a DATA packet, and we say that it is *incomplete* if we observe only a SYN packet and no DATA packet over a period of 2 minutes (which is longer than the SYN timeout period).

Consider that a router receives a route announcement for a prefix P and wishes to verify whether prefix P is reachable via the advertised route. In the simplest case, a router concludes that the prefix P is reachable if it observes at least one complete TCP flow. On the other hand, the router cannot blindly conclude that a route is unreachable if it does not observe any complete connection. Incomplete connections can arise due to reasons other than just reachability problems. These include: (a) non-live destination hosts; (b) route changes during the connection setup of a single flow i.e. SYN and DATA packets traverse different routes. (c) port scanners generating SYN packets.

Under the assumption that port scanners are not present, detecting reachability problems would be easy. To deal with non-live destinations, a router should notice multiple incomplete connections to N different distinct destination addresses (for a reasonable choice of N). The problem of route changes can be avoided by observing flows over a minimum

time period T. Hence, a router can conclude that a prefix is unreachable if during a period t it does not observe a complete TCP flow. In summary, a router must wait for a time t defined as the *maximum* between: (a) the time taken to observe N or more incomplete TCP flows with different destinations within prefix P; (b) a predefined time period T.

The basic probing mechanism described above suffers from two forms of classification errors: (a) false negatives; (b) false positives. A false negative arises when a router infers a reachable prefix as being unreachable due to incomplete connections. A false positive arises when an unreachable prefix is inferred as being reachable. A malicious end-host can create false positives by generating bogus TCP connections with SYN and DATA packets without receiving ACKs. In Section 6.3, we show how to choose the parameters N and T to reduce the chances of incomplete connections causing false negatives.

4.1.1 Dealing with False Positives

Malicious end-hosts can create false positives by opening bogus TCP connections to keep a router from detecting that a particular route is stale or invalid. Adversaries noticing route advertisements from multiple vantage points (*e.g.,* Routeviews [8]) can potentially notice mis-configurations before routers notice reachability problems. Such adversaries can exploit the situation and open bogus TCP connections.

We propose a combination of *active dropping* and *retransmission checks* as a countermeasure to reduce the probability of false positives.

1. *Active dropping:* Choose a random subset of m_1 packets within a completed connection (or across connections), drop them and raise an alarm if these packets are *not* retransmitted. Alternatively, one can just delay packets at the router instead of dropping them.
2. *Retransmission check:* Sample a different random subset of m_2 packets and raise an alarm if more than 50% of the packets are retransmitted.

An adversary generating a bogus connection cannot decide which packets to retransmit without receiving ACKs. If the adversary blindly retransmits many packets to prevent being detected by Active dropping, the Retransmission check notices a problem. We set a threshold of 50% for retransmission checks assuming that *most* genuine TCP connections will not experience a loss-rate close to 50%.

Consider an adversary that has transmitted k packets in a TCP connection without receiving ACKs to retransmit a fraction, q, of these packets. Let $C(x,y) = \frac{x!}{(x-y)!.y!}$ represent the binomial coefficient for two values x and y. The probability with which the adversary is able to mislead the active dropping test is given by $\frac{C(k\cdot q,m_1)}{C(k,m_1)}$. The probability with which the retransmission check cannot detect an adversary is given by the tail of the binomial distribution $(1-(\sum_{l=m_2/2}^{m_2} C(m_2,l)q^l(1-q)^{m_2-l}))$. Hence the overall probability, p_e, that our algorithm does not detect an adversary is:

$$\frac{C(k\cdot q,m_1)}{C(k,m_1)} \times (1-(\sum_{l=m_2/2}^{m_2} C(m_2,l)q^l(1-q)^{m_2-l}))$$

For a given prefix, the overhead of active dropping can be made very small. By choosing $m_1 = 6$ and dropping only 6 packets across different TCP flows, we can reduce the probability of false positive, p_e, to be less than 0.1%.

This countermeasure is applied only when we notice a discrepancy across different TCP connections to the same destination prefix, *i.e.,* number of incomplete connections and complete connections are roughly the same. In this case, we sample and test whether a few complete connections are indeed bogus.

procedure **LISTEN(P,T,N)**

Require: Prefix P, time period T, number of unique destinations N

```
t_0 = time at which first SYN packet observed
wait until |flows with distinct dest. in P| ≥ N
wait till clock time > t_0 + T
{Clean the data-set}
For every pair of IP addresses (src, dst) observed
if at least a single connection has completed then
    Add sample (src, dst, complete)
else
    Add sample (src, dst, incomplete)
end if
{Constants C_h, C_l must be determined in practice}
if fraction of complete connections > C_h then
    return "route is verifiable"
end if
if at least one connection completes then
    if fraction of complete connections < C_l then
        {Test for false positive}
        sample 2 future complete TCP flows towards P
        apply active dropping and retransmission checks
        if test is successful then
            return "route is verifiable"
        else
            return "route is not verifiable"
        end if
    end if
end if
```

Figure 6: Pseudo-code for the probing algorithm.

4.1.2 Detailed Algorithm

Figure 6 presents the pseudo-code for the listen algorithm. The algorithm takes a conservative approach towards determining whether a route is verifiable. Since false positive tests can impact the performance of a few flows, the algorithm uses the constant C_h and C_l to trade off between when to test for false positives. When the test is not applied, we use the fraction of complete connections as the only metric to determine whether the route works. The setting of C_h, C_l depends on the popularity of the prefixes. Firstly, we apply

the false positive tests only for popular prefixes *i.e.,* $C_l = 0$ for non-popular prefixes. For a popular prefix, we choose a conservative estimate of C_h (closer to 1) *i.e.,* a large fraction of the connections have to complete in order to conclude that the route is verifiable. On the other hand, if we observe that a reasonable fraction of combination of incomplete connections, we apply the false positive test to 2 sampled complete connections. The user has choice in tuning C_l based on the total number of false positive tests that need to be performed. For non-popular prefixes, the statistical sample of connections is small. For such prefixes, we set the value of C_h to be small.

5 Implementation

In this section, we will describe the implementation of Listen and Whisper and also discuss their overhead characteristics.

5.1 Whisper Implementation

In this section, we will only focus on the implementation of the strong split whisper protocol. The whisper implementation contains two basic components: (a) a stand alone whisper library which performs the cryptographic operations used in the protocol. (b) a Whisper-BGP interface which integrates the whisper functions into a BGP implementation. We implemented the Whisper library on top of the *crypto* library supported by OpenSSL development version 0.9.6b-33. We integrated this library with the Zebra BGP router implementation version 0.93b [2]. Our Whisper implementation works on Linux and FreeBSD platforms.

5.1.1 Whisper Library

The structure of a basic Whisper signature is:

```
typedef struct {
  BIGNUM *seed;
  BIGNUM *N;
}Signature;
```

BIGNUM is a basic data structure used within the OpenSSL crypto library to represent large numbers. The whisper library supports these three functions using the Signature data structure:

1: generate_signature(Signature *sg);
2: update_signature(Signature *sg, int asnumber, int position);
3: verify_signatures(Signature *r, Signature *s,int *aspath_r, int *aspath_s);

These functions exactly map to the three whisper operations described earlier in Section 3.2.2. The main advantage of separating the whisper library from the whisper-BGP interface is modularity. The whisper library can be in isolation with any other BGP implementation sufficiently different from the Zebra version.

5.1.2 Integration with BGP

The Whisper protocol can be integrated with BGP without changing the basic packet format of BGP. Specifically, we do not need any additional field for the Whisper signature. BGP uses community attributes within UPDATE messages that can be leveraged for embedding the signature attributes. Community attributes are 32 bit values which are optional BGP attributes that are mainly used for community-based routing mainly for multi-homing ISPs.

This design offers us many advantages over updating a version of BGP. First, a single update message can have several community attributes and one can split a signature among multiple community attributes. Second, a community attribute can be set using the BGP configuration script to allow operators the flexibility to insert their own community attribute values. In a similar vein, one can imagine a stand-alone whisper library computing the signatures and a simple interface to insert these signatures within the community attributes. Third, one can reserve a portion of the community attribute space for whisper signatures. In today's BGP, community values can be set to any value as long as they are interpreted correctly by other routers.

Our implementation uses the following semantics for the community attribute: if the first 8 bits of an attribute are set to $0xF0$ and $0xF1$, then the remaining 24 bits refer to a portion of the $seed$ and N attributes in the signature. An RSA based Whisper signature uses 2048 bits per signature field - 1024 bits for the *seed* and 1024 bits for N. Such a signature uses 88 community attributes. An ECC based Whisper implementation uses 512 bits per signature and hence uses only 22 community attributes.

5.2 Listen Implementation

We implemented the passive probing component of *Listen* (i.e. without active dropping) in about 2000 lines of code in C and have ported the code to Linux and FreeBSD operating systems. The current prototype uses the *libpcap* utility [5] to capture all the packets off the network. This form of implementation has two advantages: (a) is stand-alone and can be implemented on any machine (need not be a router) which can sniff network traffic; (b) does not require any support from router vendors. Additionally, one can execute *bgpd* (Zebra's BGP daemon [2]) to receive live BGP updates from a network router. For faster line-rates (e.g. links in ISPs), *listen* should be integrated with hardware or packet probing software like Cisco's Netflow [1]. The current implementation cannot support false positive tests since the code can only passively observe the traffic but cannot actively drop packets (since this does not perform the routing functionality).

In our implementation, the complexity of listening to a TCP flow is of the same order as a route lookup operation. Additionally, the state requirement is $O(1)$ for every prefix. We maintain a small hash table for every prefix entry corresponding to the (src,dst) IP addresses of a TCP flow and a time stamp. While a SYN packet sets a bit in the hash table, the DATA packet clears the bit and record a complete connection for the prefix. Using a small hash table, we can

Operation	512-bit	1024-bit	2048-bit
update_signature	0.18 msec	0.45 msec	1.42 msec
verify_signatures	0.25 msec	0.6 msec	1.94 msec
generate_signature	0.4 sec	8.0 sec	68 sec

Table 1: Processing overhead of the Whisper operations on a 1.5 Ghz Pentium IV with 512 MB RAM.

crudely estimate the number of complete and incomplete connections within a time-period T. Additionally, we sample flows to reduce the possibility of hash conflicts. This implementation uses simple statistical counter estimation techniques used to efficiently maintain statistics in routers. Hence, the basic form of Listen can be efficiently implemented in the fast path of today's routers.

Deployment: We deployed our *Listen* prototype to sniff on TCP traffic to and from a $/24$ prefix within our university. Additionally, we received BGP updates from the university campus router and constructed the list of prefixes in the routing table used by the edge router. The tool only needs to know the list of prefixes in the routing table and assumes a virtual route for every prefix. The Listen tool can report the list of verifiable and non-verifiable prefixes in real time. Additionally, the *Listen* algorithm is applied only by observing traffic in one direction (either outbound or inbound).

5.3 Overhead Characteristics

Overhead of Whisper: One of the important requirements of any cryptography based solution is low complexity. We performed benchmarks to determine the processing overhead of the Whisper operations. Table 1 summarizes the average time required to perform the whisper operations for 3 different key sizes: $512-$ bit, $1024-$bit and $2048-$bit. As the key size increases, the RSA-based operations offer better security. Security experts recommend a minimum size of 1024 bit keys for better long-term security.

We make two observations about the overhead characteristics. First, the processing overhead for all these key sizes are well within the limits of the maximum load observed at routers. For 2048 bit keys, a node can process more than $42,000$ route advertisements within 1 minute. In comparison, the maximum number of route advertisements observed at a Sprint router is 9300 updates every minute [9]. For 1024 bit keys, Whisper can update and verify over $100,000$ route advertisements per minute. Second, *generate_signature()* is typically a very expensive operation and in many cases consumes more than 1 sec per operation. However, this operation is performed only once over many days.

Overhead of Listen: By analyzing route updates for over 17 days in Routeviews [8], we observed that 99% of the routes in a routing table are stable for at least 1 hour. Based on data from a tier-1 ISP, we find that a router typically observes a maximum of 20000 active prefixes over a period of 1 hour *i.e.,* only 20000 prefixes observe any traffic. If the probing mechanism uses a statistical sample of 10 flows per prefix, the overhead of probing at the router is negligible. Essentially, the router needs to process 200000 flows in 3600 sec which translates to monitoring under 60 flows every second (equivalent to $O(60)$ routing lookups). Even if the number of active prefixes scales by a factor of 10, current router implementations can easily implement the passive probing aspect of Listen.

Active dropping and retransmission checks are applied only in the IP slow path. These tests are invoked only when a prefix observes a combination of both incomplete and complete connections. In order to minimize the additional overhead of these operations, we restrict these checks to a few prefixes.

6 Evaluation

In this section, we evaluate the key properties of Listen and Whisper. Our evaluation is targeted at answering specific questions about Listen and Whisper:

1. How much security can Whisper provide in the face of isolated adversaries?
2. How useful is Listen in the real world? In particular:
   (a) Can it detect reachability problems?
   (b) How many routes can be verified using Listen?
   (c) What is the response time?
3. How does Listen react in the presence of port scanners? How does one adapt to such port scanners?

We answer question (1) in Section 6.2, questions (2(a)),(3) in Section 6.3 and questions (2(b)), (2(c)) in Section 6.4.

6.1 Evaluation Methodology

Our evaluation methodology is three-fold: (a) empirically evaluate the security properties of Whisper; (b) use real-world deployment to determine usefulness of Listen; (c) use data sets from external sources to infer additional properties of Listen. The motivation for this three-fold approach is that some questions that can be answered empirically cannot be answered using deployment. Since our deployment setup (described earlier in Section 5.2) is limited to a small network. Hence we use additional data sets to answer specific questions about Listen.

We will now describe the data sets used for empirically evaluating Listen and Whisper.

FlowStat: We obtained two-sets of aggregated statistics of flows collected at an edge router of a tier-1 provider. This data covers a period of 114 hours in 2002 in one-hour and $5-$minute intervals. The first statistics provided the number of active prefixes (prefixes observing at least one flow) across both time intervals. The second statistics provided the number of flows observed within a prefix over 5 minute and 1 hour time intervals. This data is provided only for prefixes that contribute 80% of the traffic.

Internet Topology: We collected Internet AS topology data based on BGP advertisements observed from 15 different vantage points over 17 days including Routeviews [8] and RIPE [7]. The policy-based routing path between a pair of AS's is determined using customer–provider and peer–peer relationships, which have been inferred based on the technique used in [33].

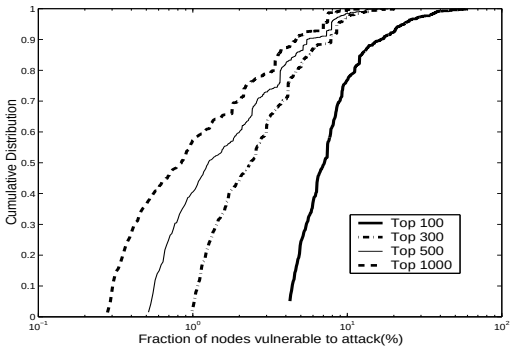

Figure 7: Effects of penalty filtering.

6.2 Whisper: Security Properties against Isolated Adversaries

In this section, we quantify the maximum damage an isolated adversary can inflict on the Internet given that Strong Split Whisper is deployed. Since SSW offers path integrity, an isolated adversary cannot propagate invalid routes without raising alarms unless there exists no alternate route from the origin to the verifier (i.e. adversary is present in all paths from the origin to the Internet).

Given an adversary that is willing to raise alarms, we analyzed how many AS's can one such adversary affect. In this analysis, we exclude cases where the adversary is already present in the only routing path to a destination AS. We use penalty based filtering as the main defense to contain the effects of such invalid routes. We assume that in the worst-case, an adversary compromising a single router in an AS is equivalent to compromising the entire AS especially if all routers within the AS choose the invalid route propagated by the compromised router.

Let M represent an isolated adversary propagating an invalid route claiming direct connectivity to an origin AS O. AS V is said to be *affected* by the invalid route if V chooses the route through M rather than a genuine route to O either due to BGP policies or shorter hop length. Based on common practices, we associate all AS's with a simple policy where customer routes have the highest preference followed by peers and providers [18]. Given all these relationships, we define the *vulnerability* of an origin AS, O, as $V(O, M)$ to be the maximum fraction of AS's, M can affect. Given an isolated adversary M, we can quantify the worst-case effect that M can have on the Internet using the *cumulative distribution* of $V(O, M)$ across all origin AS's in the Internet.

With AS's deploying penalty based filtering as a defense, we expect the vulnerability $V(O, M)$ to reduce. We study how the cumulative distribution of $V(O, M)$ for a single adversary M varies as a function of how many AS's deploy penalty based filtering. We consider the scenario where the top n ISPs deploy penalty based filtering (based on AS degree). Figure 7 shows this cumulative distribution for for different values of $n = 100, 300, 500$ and 1000. These distributions are averaged across all possible choices for M.

We make the following observations. First, a median value of 1% for $n = 1000$ indicates that a randomly located adversary can affect at most 1% of destination AS's by propagating bogus advertisements assuming that the top 1000 ISPs deploy penalty based filtering. This is orders of magnitude better that what the current Internet can offer where a randomly located adversary can on an average affect nearly 30% of the routes to a randomly chosen destination AS. The value 30% was determined by repeating the same analysis on the Internet topology without using SSW.

Second, in the worst case, a single AS can at most affect 8% of the destination AS's for $n = 1000$. For $n = 1000$, 8% is a limit imposed by the structure of the Internet topology. If we remove the top 1000 AS's from the topology, the size of the largest connected component contains roughly 8% of the AS's. One malicious AS in this component can potentially affect the other AS's within the same component.

Third, if all provider AS's deploy penalty based filtering, the worst case behavior can be brought to a much smaller value than 8%. Additionally, there is very little benefit in deploying penalty based filtering in the end-host networks since they are not transit networks and typically are sources and sinks of route advertisements. Hence, any filtering at these end-hosts only protects themselves but not other AS's.

To summarize, the Whisper protocol in conjunction with penalty based filtering can guarantee that a randomly placed isolated adversary propagating invalid routes can affect at most 1% of the AS's in the Internet topology.

	Number of Reachability Problems	Probability of False Negatives
Outbound	235	0.93%
Inbound	343	0.37%

Table 2: Listen: Summary of Results

6.3 Listen: Experimental Evaluation

In this section, we describe our real-world experiences using the Listen protocol. We make two important observations:

1. In reality, we found that a large fraction of incomplete TCP connections are *spurious i.e.,* not indicative of a reachability problem. We show that by adaptively setting the parameters T, N of our listen algorithm we can drastically reduce the probability of such false negatives due to such connections.
2. We are able to detect several reachability problems using Listen including specific misconfiguration related problems like forwarding errors.

Table 2 presents a concise summary of the results obtained from our deployment. First, we were able to detect reachability problems to 578 different prefixes from our testbed. Second, we reduce the false negative probability due to spurious connections to 0.95% and 0.37% respectively for outbound and inbound connections.

We will now describe our deployment experiences in greater detail. In our testbed, we additionally use active probing to verify the correctness of results obtained using Listen. It is

Number of end-hosts behind $/24$ network	28
Number of days	40
Total No. of TCP connections	994234
No. of complete connections	894897
No. of incomplete connections	99337
Average Routing Table Size	123482
Total No. of Active Prefixes	11141
Average No. of Active Prefixes per hour	141
Average No. of Active Prefixes per day	2500-3000
Verifiable Prefixes	9711
Prefixes with perennial problems	42

Table 3: Aggregate characteristics of Listen from the deployment

activated for every failed TCP connection. We use three different techniques: (a) ping the destination; (b) traceroute and check whether any IP address along in the path is in the same prefix as the destination; (c) perform a port 80 scan on the destination IP address. We classify an incomplete connection as having a reachability problem if all the three techniques fail. We classify an incomplete connection as a *spurious connection* if one of the probing techniques is able to detect that the route to a destination prefix works. A spurious TCP connection is an incomplete connection that is not indicative of a reachability problem.

Table 3 presents the aggregate characteristics of the traffic we observed from our deployment. We observed the traffic to and from a $/24$ network for over 40 days. In reality, we found that many incomplete connections do occur and a large fraction of these connections are spurious. Nearly 10% of the TCP connections we observed in our testbed were incomplete. Of these, nearly 91% of inbound connections and 63% of outbound connections are spurious. A more careful observation at the spurious connections showed that nearly 90% of spurious inbound connections are due to port scanners and worms. The most prominent ones include the Microsoft NetBIOS worm and the SQL server worms [6]. Spurious outbound connections occur primarily due to failed connection attempts to non-live hosts and attempts to access a disabled ports of other end-hosts (*e.g.*, telnet port being disabled in a destination end-host).Given this alarmingly high number of spurious connections, we now propose defensive measures to reduce the probability of false negatives due to such connections.

6.3.1 Defensive Measures to reduce False Negatives

In this section, we show that one can adaptively set the parameters N, T in the listen algorithm to drastically reduce the probability of false negatives due to spurious TCP connections. In particular, we show by adaptively tuning the minimum time period, T, one can reduce false negatives due to port scanners and by tuning the number of distinct destinations, N, one can deal with non-live hosts.

Given the nature of incomplete connections in our testbed, we use outbound incomplete connections as a test sample for non-live hosts and inbound connections as the test sample for port scanners and worms. In both inbound and outbound, we restricted our samples to only those connections which are known to be false negatives.

Type of problem	Number of Prefixes
Routing Loops	51
Forwarding Errors	64
Generic (forward path)	146
Generic (reverse path)	317

Table 4: The number of prefixes affected by different types of reachability problems.

Setting T: One possibility is to choose an interval T large enough such that the router will notice at least one genuine TCP flow during the interval. Such a value of T will depend on the popularity of a prefix. The popularity of a prefix, $pop(P)$, is defined as the mean time between two complete TCP connections to prefix P. We can model the arrival of TCP connections as a Poisson process with a mean arrival rate as $1/pop(P)$ [30].[2] Given this, we can set the value of $T = 4.6 \times pop(P)$ to be 99% certain that one would experience at least one genuine connection within the period T. To have a 99.9% certainty, one needs to set $T = 6.9 \times pop(P)$. For prefixes that hardly observe any traffic, the value of T will be very high implying that port scanners generating incomplete connections to such prefixes will not generate any false alarms.

From our testbed, we determine the mean separation time between the arrival of two incoming connections to be $pop(P) = 34.1$ sec. By merely setting $T = 156.8$ to achieve 99% certainty, we could reduce the probability of false negatives in Listen to 0.37%. Throughout the entire period of measurement, only during 8 periods of 156 seconds each did we verify incorrectly that the local prefix is not reachable. In the process, we could reduce the number of spurious connections that we considered from 91.83% to 0.35%. Note that if the volume of traffic observed is higher (*e.g.*, in a tier-1 ISP), the value of T will be much smaller.

Setting N: The choice of an appropriate value of N trades off between minimizing the false negative ratio due to non-live hosts and the number of reachability problems detected. In our testbed, we noticed that by merely setting $N = 2$, we can significantly reduce the false negative ratio in outbound connections from 63% to less than 1%. However, Listen reported only 35 out of 663 potential prefixes to have routing problems. For several $/24$ prefixes, we observed TCP connections to only a single host. By setting $N = 2$, we tend to omit these cases. In practice, the value of N is dependent on the destination prefix and the traffic concentration at a router. For many $/24$ prefixes, we need to set $N = 1$. For $/8$ and $/16$ prefixes, one can choose larger values of $N = 4$ or $N = 5$ provided the prefix observes diversity in the traffic.

6.3.2 Detected Reachability Problems

After using the defensive measures against false negatives, Listen is able to detect several reachability problems with

[2]While arrival of TCP connections can be modeled as a Poisson process, packet arrival times cannot be modeled using a Poisson process.

	5-min interval	1-hour interval
Active prefixes	1004–5061	4201–13007
Mean active	2908	7663
Elephant	318	625
Mice	2590	7048
No traffic	127 K	122 K

Table 5: Number of active, elephant, mice prefixes observed in FlowStat. We calculated the number of prefixes with no traffic as the difference between the mean number of active prefixes and the number of elephant prefixes. The routing table size has approximately 130 K entries.

a low false negative ratio. Two particular forms of reachability problems which we can detect are: *routing loops* and *forwarding errors* due to unknown IP addresses. While Listen signals the existence of the problem, we use traceroute to check whether the routing path to a destination IP has a routing loop or not. A forwarding error due to an unknown IP address arises when the destination IP address specified in a packet corresponds to a genuine prefix but does not have a forwarding entry in the routing table. This can potentially arise due to staleness of routes. Table 4 summarizes the number of prefixes which are affected by each type of problem. In particular, we observe routing loops along the data paths to 51 different prefixes. Additionally, 64 different prefixes were affected by forwarding errors. Listen detected 463 other prefixes having other forms of reachability problems. We classify these into two categories: *forward path* and *reverse path*. A forward path problem implies that the path an end-host (initiator of TCP connection) from our network to a destination prefix is problematic. Similarly, reverse path is used in the context of inbound connections. We observed more reverse path than forward path problems due to a larger diversity of prefixes in inbound connections.

To cite a few examples of reachability problems we observed:

1. A BGP daemon within our network attempted to connect to another such daemon within the destination prefix 193.148.15.0/24. The route to this prefix was perennially unreachable. For a few days, we observed routing loops in the path to this prefix.
2. The route to Yahoo-NET prefix 207.126.224.0/20 was fluctuating. During many periods, the route was detected as unavailable.
3. We detected several local outages ranging from 10 minutes to an hour within the campus network. During these periods, none of the prefixes are verifiable.

6.4 Listen: Empirical Evaluation

In this section we answer two questions about Listen: (a) How many routes are verifiable in practice using Listen? (b) What is the time to detect a problem?

Number of Verifiable Routes: The number of routing table entries verifiable using *listen* is dependent on two parameters: (a) how frequently does a route change; (b) do we observe any traffic for a given prefix during the stable period

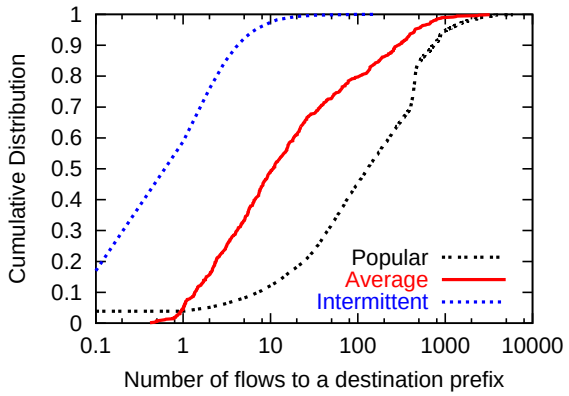

Figure 8: Number of flows observed within a prefix over an interval of 5 minutes at an access router of a tier-1 ISP. We consider only 625 elephant prefixes in this case.

of a route. By analyzing route updates for over 17 days in Routeviews, we observed that 99% of the routes in a routing table are stable for at least 1 hour. To analyze the number of verifiable routes, we set the maximum stable time period as 1 hr (a conservative estimate).

Table 5 shows the aggregate statistics of the number of verifiable routes in the FlowStat over two separate time-periods: 5 min and 1 hr. We make three observations. First, only 10% of the prefixes observe any traffic over the period of 1 hour. While probing is not applicable for the remaining 90% of the prefixes, reachability problems along these routes will have no impact on the traffic. Second, the FlowStat dataset given to us classifies prefixes into *elephant* and *mice* prefixes. Elephant prefixes together account for 80% of the traffic. We infer that the number of elephant prefixes is very small compared to the total number of prefixes (in the order of 300-700 for the tier-1 ISP). Third, the number of verifiable prefixes within $5-$ minute intervals constitute only 25-40% of the number of verifiable prefixes in 1 hour.

Response Time: The time required to detect a reachability problem for a route depends on the frequency of new TCP flows observed for the corresponding prefix. Based on this, we can classify elephant prefixes into two categories: *popular* and *intermittent*. We define a prefix to be *popular* if it observes at least one flow for a large fraction ($> 90\%$) of the measurement intervals. We classify other prefixes as *intermittent*. Figure 8 shows the cumulative distribution of the average number of flows observed by an elephant prefix in 5 minute intervals and contrasts a popular prefix from an intermittent one.

Additionally, we need to observe multiple flows within a time-period T in order to conclude whether a route is reachable or not. The listen mechanism can provide a fast detection time for elephant prefixes in particular popular prefixes. In the best case, we observed more than 1000 flows to one popular prefix within certain intervals while in the mean case we observe roughly 200 flows. In the mean case, an access router may observe at least 10 flows for a popular prefix within 15 seconds. Hence the router can notice within seconds when such a prefix becomes unreachable. In the aver-

age case, the detection time is in the order of minutes for elephant prefixes. For mice prefixes, the detection time can be much higher. The important observation is that our Listen mechanism works best for those prefixes that attract the largest amount of traffic.

6.5 Summary of Results

We summarize the important conclusions from our evaluation results:

1. The Whisper protocol in conjunction with penalty based filtering can restrict the damage a randomly placed isolated adversary can cause to less than 1% of the AS's in the Internet topology.
2. Listen is useful in detecting many reachability problems. We detected reachability problems to 578 different prefixes of which 51 prefixes had routing loops and 64 of them were affected by forwarding errors.
3. An adaptive choice of parameters for the Listen algorithm can largely reduce the probability of false negatives due to spurious connections. In our testbed, the probability of false negatives is reduced to 0.37% in the presence of port scanners and worm traffic.
4. Listen can provide fast detection of reachability problems for popular prefixes (in the order of 15 seconds).

7 Colluding Adversaries

Colluding adversaries can perform three types of attacks additional to acting as isolated adversaries. First, colluding adversaries can tunnel advertisements and secrets between them and attempt to defeat the security measures. For example, without complete knowledge of the Internet topology, it is impossible to detect a fake AS link between two colluding adversaries. Second, colluding adversaries can target the same set of destination AS's by propagating invalid routes. By doing so, these nodes can inflict the maximum damage on specifically chosen destination AS's. Third, within the context of BGP, colluding adversaries can exploit BGP policies to propagate invalid routes.

While we cannot deal with the problem of tunneling advertisements, we can provide protective measures against the other two problems. In Figures 9, 10 and 11, we contrast 3 scenarios: (a) effect of colluding adversaries on the current Internet; (b) effect of colluding adversaries with whisper protocols and policy routing; (c) effect of colluding adversaries with whisper protocols and shortest path routing. All these graphs show the cumulative distribution of the vulnerability metric (defined in Section 6.2) for a set of colluding malicious adversaries. We specifically consider three cases: (a) 2 colluding tier-1 ASes; (b) 2 colluding tier-2 ASes (c) 12 colluding customer ASes.

Firstly, the amount of damage 12 randomly compromised customer routers can inflict is of the same magnitude as that of two tier-1 nodes. This describes the seriousness of the problem of colluding adversaries. Secondly, while Whisper protocols provide some level of protection against colluding adversaries, the advantage quickly diminishes as the number of adversaries increases in the presence of policy routing. Finally, whisper protocols with shortest path routing offers the maximum protection. The difference between shortest path routing and policy routing is strikingly high in the case of colluding adversaries in customers. In the case of shortest path routing, the damage that 12 customers can inflict is negligible.

Colluding Adversaries exploiting BGP policies: BGP allocates higher importance to *local preference* than the AS path length. The local preference of a route is based on which neighboring AS advertised it. The typical policy preference of an AS is: Customer routes have a higher preference than peer routes which are more preferred than provider routes [18].

Consider a simple attack scenario: a single adversary compromises several routers in different customer networks. Such an attacker can exploit the fact that customer routes typically receive a higher local preference even if other shorter paths are available. As an example, consider 10 customers of 10 different tier-1 ISPs. By tunneling advertisements between them, these nodes can exploit local preference and virtually hijack the routes to all other customers of these 10 ISPs (These nodes set up a virtual tier-1 plane between themselves). This is not a weakness of our protocols, but it is a loop hole in the current application of BGP policies. In principle, this problem exists also in S-BGP.

Our analysis in Figure 11 shows that shortest path routing offers much better protection in the face of colluding adversaries especially when the compromised nodes are in customer ASes. However, for economic reasons, policy routing is necessary In order to strike a middle ground between economic considerations and the greater protection offered by shortest path routing, we propose a simple modification to the calculation of local-preferences: *Do not associate any local preference to customer routes that have an AS path length greater than* 2. Since whisper protocols provide path integrity, an invalid route that is tunneled across two colluding adversaries will have a minimum path length of 3. We believe that this modification to BGP policies should have little impact on current operation since most customer routes today have a path length less than 3.

To summarize, whisper protocols in combination with the modified application of policies (emulating shortest path routing) can largely restrict the damage of colluding adversaries.

8 Conclusions

In this paper we consider the problem of reducing the vulnerability of BGP in the face of misconfigurations and malicious attacks. To address this problem we propose two techniques: Listen and Whisper. Used together these techniques can detect and contain invalid routes propagated by isolated adversaries, and a large number of problems (such as unreachable prefixes) due to misconfigurations. To demonstrate the utility of Listen and Whisper, we use a combination of real world deployment and empirical analysis. In particular, we show that Listen can detect unreachable prefixes with a low

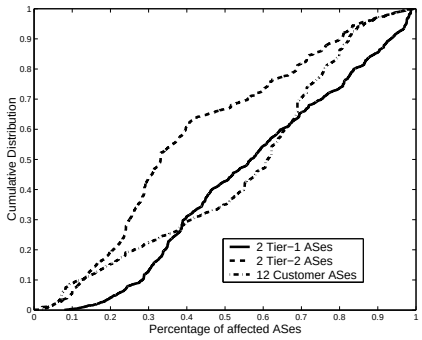

Figure 9: The effects of colluding adversaries in the current Internet.

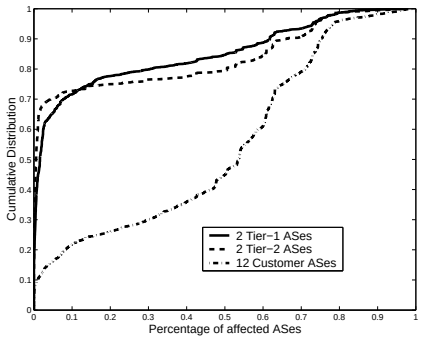

Figure 10: Effects of colluding adversaries with whisper protocols + policy routing.

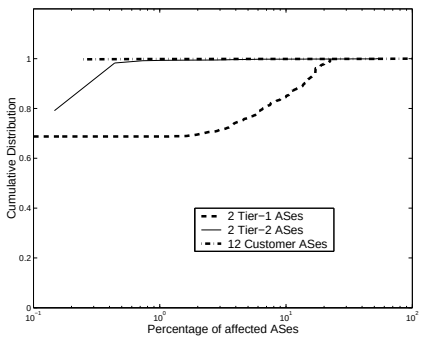

Figure 11: Effect of colluding adversaries with whisper protocols + shortest path routing

probability of false negatives, and that Whisper can limit the percentage of nodes affected by a randomly placed isolated adversary to less than 1%. Finally, we show that both Listen and Whisper are easy to implement and deploy. Listen is incrementally deployable and does not require any BGP changes, while Whisper can be integrated with BGP without changing the packet format.

References

[1] Cisco ios netflow. `http://www.cisco.com/warp/public/732/Tech/nmp/netflow/index.shtml`.

[2] Gnu zebra router implementation. `http://www.zebra.org/`.

[3] Internet Corporation for Assigned Names and Numbers. `http://www.icann.org/`.

[4] Internet routing registry. `http://www.irr.net/`. Version current January 2003.

[5] libpcap utility. `http://sourceforge.net/projects/libpcap`.

[6] Microsoft port 1433 vulnerability. `http:/lists.insecure.org/lists/vuln-dev/2002/Aug/0073.html`.

[7] Ripe ncc. `http://www.ripe.net`.

[8] Routeviews. `http://www.routeviews.org/`.

[9] Sprint IPMON project. `http://ipmon.sprint.com/`.

[10] Trends in dos attack technology. `http://www.cert.org/archive/pdf/DoS_trends.pdf`.

[11] J. Arkko and P. Nikander. How to authenticate unknown principals without trusted parties. In *Proc. Security Protocols Workshop 2002*, April 2002.

[12] M. Bellare and D. Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. volume 1223 of *Lecture Notes in Computer Science*. Springer Verlag, 1997.

[13] I. Blake, G. Serossi, and N. Smart. *Elliptic Curves in Cryptography*. Cambridge University Press, 2000.

[14] V. J. Bono. 7007 explanation and apology. `http://www.merit.edu/mail.archives/nanog/1997-04/msg00444.html`.

[15] R. Clarke. Conventional public key infrastructure: An artefact ill-fitted to the needs of the information society. Technical report. `http://www.anu.edu.au/people/Roger.Clarke/II/PKIMisFit.html`.

[16] D. Davis. Compliance defects in public key cryptography. In *Proc. 6th USENIX Security Symposium*, 1996.

[17] C. Ellison and B. Schneier. Ten risks of PKI: What you're not being told about public key infrastructure. *Computer Security Journal*, 16(1):1–7, 2000. Available online at URL `http://www.counterpane.com/pki-risks.html`.

[18] L. Gao and J. Rexford. Stable internet routing without global coordination. In *IEEE/ACM Transactions on Networking*, 2001.

[19] G. Goodell, W. Aiello, T. Griffin, J. Ioannidis, P. McDaniel, and A. Rubin. Working around BGP: An incremental approach to improving security and accuracy of interdomain routing. In *Proc. of NDSS*, San Diego, CA, USA, Feb. 2003.

[20] Y. Hu, D. B. Johnson, and A. Perrig. SEAD: Secure efficient distance vector routing for mobile wireless ad hoc networks. In *Proc. of WMCSA*, June 2002.

[21] Y. Hu, A. Perrig, and D. B. Johnson. Wormhole detection in wireless ad hoc networks. Technical Report TR01-384, Department of Computer Science, Rice University, Dec. 2001.

[22] Y. Hu, A. Perrig, and D. B. Johnson. Efficient security mechanisms for routing protocols. In *Proc. of NDSS'03*, February 2003.

[23] S. Kent, C. Lynn, and K. Seo. Design and analysis of the Secure Border Gateway Protocol (S-BGP). In *Proc. of DISCEX '00*.

[24] S. Kent, C. Lynn, and K. Seo. Secure Border Gateway Protocol (Secure-BGP). *IEEE Journal on Selected Areas of Communications*, 18(4):582–592, Apr. 2000.

[25] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP misconfigurations. In *Proc. ACM SIGCOMM Conference*, Pittsburg, Aug. 2002.

[26] Z. Mao, J. Rexford, J. Wang, and R. H. Katz. Towards an accurate AS-level traceroure tool. In *ACM SIGCOMM*, 2003.

[27] S. Murphy, O. Gudmundsson, R. Mundy, and B. Wellington. Retrofitting security into Internet infrastructure protocols. In *Proc. of DISCEX '00*, volume 1, pages 3–17, 1999.

[28] J. Ng. Extensions to BGP to support Secure Origin BGP (sobgp). Internet Draft draft-ng-sobgp-bgp-extensions-00, Oct. 2002.

[29] V. N. Padmanabhan and D. R. Simon. Secure traceroute to detect faulty or malicious routing. In *Proc. HotNets-I*, 2002.

[30] V. Paxson and S.Floyd. Wide area traffic: Failure of poisson modeling. In *Proc. ACM SIGCOMM*, 1994.

[31] K. Seo, C. Lynn, and S. Kent. Public-Key Infrastructure for the Secure Border Gateway Protocol (S-BGP). In *Proc. of DISCEX '01*, volume 1, pages 239–253, 2001.

[32] B. Smith and J. Garcia-Luna-Aceves. Securing the Border Gateway Routing Protocol. In *Proc. Global Internet '96*, London, UK, November 1996.

[33] L. Subramanian, S.Agarwal, J.Rexford, and R. H. Katz. Characterizing the Internet hierarchy from multiple vantage points. In *IEEE INFOCOM*, New York, 2002.

[34] R. Thomas. `http://www.cmyru.com`.

[35] X. Zhao, D. Pei, L. Wang, D. Massey, A. Mankin, S. F. Wu, and L. Zhang. An analysis of BGP multiple origin AS (MOAS) conflicts. In *ACM SIGCOMM IMW*, 2001.

[36] D. Zhu, M. Gritter, and D. Cheriton. Feedback based routing. In *Proc. of HotNets-I*, October 2002.