Listen and Whisper: Security Mechanisms for BGP

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

[*] *University of California, Berkeley* {lakme,istoica,randy}@cs.berkeley.edu

[++] *Fraunhofer Institute, Germany* vroth@igd.fhg.de

[+] *ICSI, Berkeley* shenker@icsi.berkeley.edu

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. 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 [25]. 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:

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.).

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

[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.

corresponding routing path advertised in the control plane. Mao et al. [27] show that for nearly 8% of Internet paths, the control plane and data plane paths do not match.

Two primary sources of invalid routes are misconfigurations and deliberate attacks. While these are the only sources of invalid routes in the control plane, data plane invalidity can occur additionally due to genuine reasons (e.g. intra/inter-domain routing dynamics [27]). The fact that a sizable fraction of Internet routes are invalid in the data plane motivates the need for separately verifying the correctness of routes in the data plane and not merely focusing on the control plane. Prior works on securing BGP focus primarily on the control plane.

Misconfigurations occur in several forms ranging from buggy configuration scripts to human errors. In the control plane, Mahajan et al. [26] infer 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, 35], 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). Colluding adversaries have the additional ability to tunnel route advertisements and fake additional links in the topology.

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: Two colluding nodes can always pretend the existence of a direct link between them by tunneling packets/ advertisements. In the absence of complete knowledge of the Internet topology, these fake links cannot be detected even using heavy-weight security solutions like Secure BGP [24]. While these fake links enable colluding adversaries to propagate invalid routes without being detected, we show that if BGP employs *shortest-path* routing then a large fraction of the paths with fake links can be avoided. On the contrary, colluding adversaries can exploit the current application of BGP policies to mount a large scale attack. To deal with this problem and yet support policy-based routing, we suggest simple modifications to the BGP policy engine which in combination with Whisper can largely restrict the damage that colluding adversaries can cause.

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 deployment and security analysis. In Section 7, we discuss the case of colluding adversaries and finally present our conclusions in Section 9.

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 [26, 36]. While these works

identify two forms of misconfigurations (origin and export misconfigurations), a fundamental limitation with analyzing BGP streams: *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 but not when the path is modified.

Mao *et al.* [27] build an AS-traceroute tool to detect the AS path in the data plane which can be used for data-plane verification. While this tool can detect several forms of invalid routes in the data plane, it is useful for diagnostic purposes only once a problem is detected. Padmanabhan *et al.* [30] propose a secure variant of `traceroute` to test the correctness of a route. However, this mechanism requires a prior distribution of cryptographic keys to the participating AS's to ascertain the integrity and authenticity of traceroute packets. In the context of feedback based routing, Zhu *et al.* [37] 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.

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.* [33], Murphy *et al.* [28], Kent *et al.* [25], and recent work on *Secure Origin BGP* [29]. All these protocols make extensive use of digital signatures and public key certification. More lightweight approaches based on cryptographic hash functions have been proposed *e.g.*, by Hu *et al.* [21, 23] in the context of secure routing in ad hoc networks. However, these 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 [17], Ellison and Schneier [18]. The PKI model has been criticized based on technical grounds, on grounds of a lack of trust and privacy, as well as on principle [17, 18, 16]. Building an Internet wide PKI infrastructure incurs huge costs and has a high risk of failure. Secure-BGP, despite the push by a 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.* [20] 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. Second, the origin authentication of the database contents again demands a public key infrastructure [29]. Third, access to such databases relies on the very infrastructure that it is meant to protect, which is hardly an ideal situation.

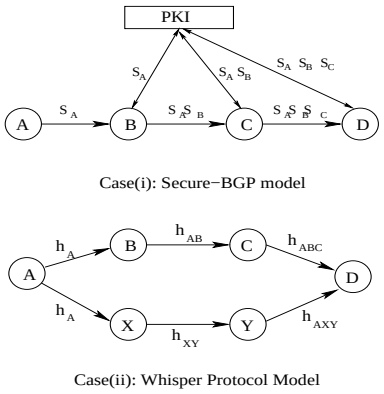

Figure 1: Comparison of the security approach of Whisper protocols with Secure BGP

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.

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].

Whisper 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 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 different flavors of route consistency tests of increasing complexity which offer different security guarantees. Conceptually, these constructions introduce a *signature* field in every BGP UPDATE message which is

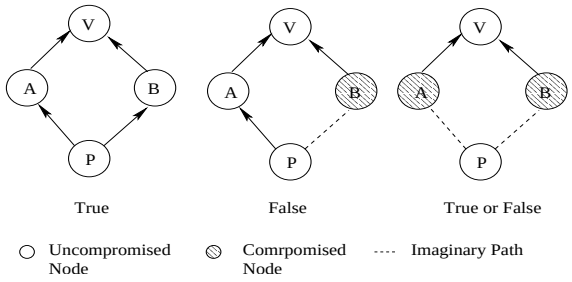

Figure 2: Different outcomes for a route consistency test. In all these scenarios, the verifying node is V. The verifying node checks whether the two routes it receives to destination P are consistent with each other.

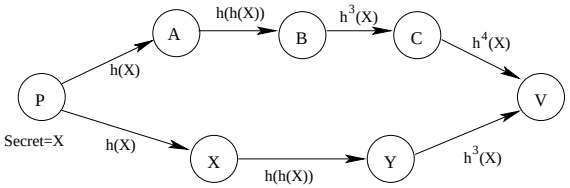

Figure 3: Weak-Split construction using a globally known hash function $h()$

updated by every AS along a path and is used for performing the route consistency test. The origin AS (the originator of a route announcement) of a destination prefix initiates the signature field and every intermediary AS that is not the origin of a destination prefix is required to update the signature field using a cryptographic hash function.

Weak-Split Whisper(WSW): 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.* [22, 21] 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.* However, weak split whisper cannot offer path integrity since an adversary can modify the AS numbers along a path without affecting the path length.

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.

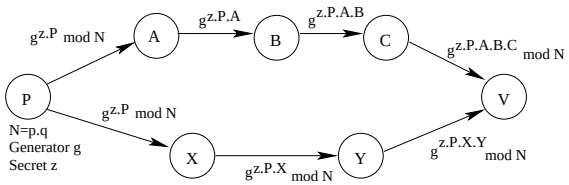

Figure 4: Basic Strong-Split construction using exponentiation under modulo N where $N = p \times q$, a product of two large primes.

3.2.1 RSA-based Strong Split Whisper

Figure 4 shows a hash construction of the whisper signature using the RSA-based strong split whisper(SSW). We use a minor modification of the illustrated example. We will elaborate the three basic operations for this protocol:

generate-signature: The origin AS computes three 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. g is a generator in the prime group Z_p and Z_q and z is a random seed. The signature generated is a tuple $(N, g^z \text{mod} N)$ where only the origin 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.

update-signature: Every AS is associated with a unique AS number which is specified in the path. Assume AS A receives an advertisement with a signature (N, y). A updates this signature to $(N, y^A \text{mod} N)$. In Figure 4, the route announcement for the AS path P, A, B, C, has the signature $(N, g^{z.P.A.B.C} \text{mod} N)$.

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 outputs the routes to be consistent if:

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

SSW is similar to the MuHASH construction proposed by Bellare *et al.* [12] for incrementally hashing signatures. The key observation with this construction is: given N and given $g^x \text{mod} N$, an adversary cannot compute $x^{-1} \text{mod} \phi(N)$ (given $N = p \times q$, $\phi(N) = (p-1) \times (q-1)$) and hence cannot remove the signature of previous nodes in the AS path. Additionally, in practice, for every AS B, we use $h(A, B, C)$ in the exponentiation (instead of B) where A and C are the predecessor and successors of B in the route and $h()$ is a one-way hash function. This way, B sets up a binding between itself and its neighbors in the signature. Hence adversaries upstream can neither remove B nor change its position in the AS path.

Common Modulus Problem: One attack on the RSA-based strong split whisper mechanism is the common modulus attack on RSA encryption [32]. If an AS learns two RSA signatures of the form $s^d mod N$ and $s^e mod N$, an adversary can decipher s provided $gcd(d, e) = 1$. This implies that the RSA-based split whispers will not be able to guarantee *path integrity* in the face of a common modulus attack. This is a fundamental problem with the RSA group and not of the whisper construction. A generalization to the whisper construction requires an Abelian group $(G, \odot)$ which should satisfy two properties: (a) it is computationally infeasible to find the inverse a^{-1} of a given group member a; (b) G does not have the common modulus problem ***i.e.,*** Given $s \odot x$ and $s \odot y$, one should not be able to infer s. While the RSA-group satisfies the first property, it does not satisfy the second. While other Abelian groups like Elliptic curves [13] exist, the choice of an appropriate group that satisfies both properties is an open issue.

We will now describe two alternate whisper constructions which offer path-integrity and do not have the common modulus problem.

3.2.2 SHA-based Strong Split Whisper

The SHA-based Strong split whisper is an emulation of the previous RSA construction where the exponentiation step is replaced with the SHA one-way hash function [32]. Unlike RSA-based SSW, SHA-based whisper signatures can only be verified by the originator since it does not satisfy the commutativity property of RSA.

Let $h_S()$ represent the SHA one-way hash-function which takes an arbitrary string as input and outputs a 160-bit hash value. A SHA-SSW signature of a route R consists of two parameters: (a) the hash-value of the path; (b) public-key of the originator (as published in the route announcement). In SHA-based SSW, an origin A initiates an announcement to its neighbor B with the signature $h_S(Z, (A, B))$ where Z is a 160-bit nonce and P, its public key. Every intermediary AS B along a path that receives an update from a neighbor A with a SHA signature Y generates the signature $h_S(Y, (A, B, C))$ to its successor C along the path. Hence, a SHA whisper signature is simply a hash signature of the initiator's nonce Z and all neighbor bindings (A, B, C) along a path. The path integrity of SHA signatures follows because the SHA signatures are not: (a) invertible given the one-way hash function property; (b) reproducible by an adversary. Additionally, SHA does not face the common modulus problem.

Consistency Testing: Unlike RSA, only the origin A can verify the correctness of the SHA signature of a path. A node V that receives two routes R, S to origin A performs the following operations for consistency testing. First, if the public keys advertised in routes R and S are inconsistent, then the routes are obviously inconsistent. Second, if the public-keys are the same, V chooses R as its routing path (by fixing its routing table) and sends the encrypted form

of S's SHA signature to A querying whether the signature matches the path. A is supposed to send its response to V and signs it with its private key so that V can verify whether A indeed generated the message. Similarly, by setting S as the chosen route, A can verify R's signature. If A responds positively, the routes are deemed consistent. Note that the above test does not make any assumption about the nature of the path from A to V (*i.e.*, symmetric routing is not necessary) since A signs its response using its private key. However, it assumes that at least one valid reverse path exists from A to V. In summary, SHA-based SSW guarantees path-integrity but has the additional complexity of a pair of message exchanges between the verifier and the originator. From an implementation perspective, these messages are routed using normal IP routes and the only modification necessary is an additional signature field in the BGP UPDATE message. We leverage two optimizations to reduce message overhead: (a) The public key of an origin needs to be communicated only once provided future updates use the same consistent public key. (b) Given that the set of distinct routes to a destination AS is relatively stable over time as well as small [15], the SHA signature verification needs to be done only once for each distinct AS path.

3.2.3 Loop Whisper

Loop Whisper is a simple consistency testing strategy which uses AS-level traceroute to check correctness. A verifier V that receives two route advertisements R and S to the same destination A can form an AS-loop involving itself and AS's in R and S. If R and S are completely vertex-disjoint (except the origin A), then the AS-loop is simply $R^{-1}S$ where R^{-1} is the inverse AS-path of R.

Given an AS-loop, the verifier generates a special control message (like an ICMP message) with a nonce and the AS-loop and *source-routes* the message along the loop to test whether the loop exists (nonce is used as a packet identifier). Routing such control messages requires: (a) Each AS should have an additional control mechanism in the routers to handle these specific packets and route them to the neighbor as specified in the source route. (b) Each AS should forward control messages to a neighbor only when a genuine neighbor exists. The second constraint guarantees that if an adversary generates an invalid route with a non-existent path, the loop-test will never succeed. If a loop-test succeeds, two routes are deemed consistent. In summary, while loop whisper guarantees path integrity, it requires at least one router in each AS to support AS-level traceroute.(Note that not all routers need to be modified). From a deployment perspective, SHA-SSW signature based mechanism is easier to deploy than loop whisper.

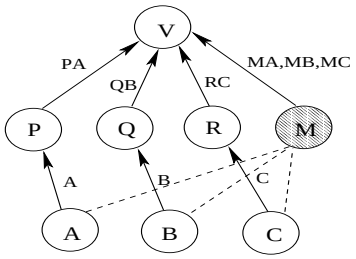

Figure 5: Detecting Suspicious AS's: In this example, M is a malicious AS that propagates 3 invalid routes to 3 different destinations A,B,C. The AS paths in the routes propagated are indicated along the links. The verifier V assigns penalty values of 3,1, 1, 1 to M, A, B, C respectively.

3.3 Containment: Penalty Based Route Selection

Route consistency testing only provides the ability to trigger alarms whenever a node propagates invalid route announcements. We append consistency testing with *penalty based route selection*, a simple containment strategy that attempts to identify suspicious candidates and avoid routes propagated by them. The strategy works as follows: A router counts across destinations how often an AS appears on an invalid 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. The route selection strategy is to *choose the route to a destination with the lowest penalty value.*

Consider the topology in Figure 5, where M is a malicious node that propagates 3 invalid route announcements with AS paths MA, MB, MC. By choosing the minimum penalty route, the verifier V can avoid the invalid routes through M since they have a higher penalty value. 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.

Penalties should primarily be viewed as a reasonable first response to detect suspicious candidates and not as a foolproof mechanism. In the presence of an isolated adversary, penalty based filtering can ensure that the effects of the adversary are contained. We believe that penalties is a good mechanism to detect malicious adversaries in customer AS's but should be applied with caution when involving AS's in the Internet core. In particular, penalties are not a good security measure in the presence of colluding adversaries or when the number of independent adversaries is large. For example, multiple adversaries can artificially raise the penalty of an innocent AS by including its AS number in the invalid route.

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) incrementally deployable and standalone solution with no modifications to BGP; (c) quick detection of reachability problems for popular prefixes; (d) low overhead.

The basic form of the protocol described in this section is vulnerable to port scanners generating many incomplete connections. In Section 6.2, 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 where t is 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 6.2, 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.

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 their overhead characteristics.

5.1 Whisper Implementation

In this section, we will only focus on the implementation of the strong split whisper protocol (RSA variant). The SHA variant requires a modification to the hash function we use in our code. [2] 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.

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.

5.1.1 Whisper Library

The structure of a basic Whisper signature is:

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

[2]The additional control messages in SHA-based SSW are data-plane messages and are not incorporated in the code.

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 3.2.1. The main advantage of separating the whisper library from the whisper-BGP interface is modularity. The whisper library can be used 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. BGP uses $32-bit$ community attributes which are options within UPDATE messages that can be leveraged for embedding the signature attributes. 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. An RSA-SSW uses 2048 bits per signature field, while SHA-SSW needs $1184 = 160 + 1024$ bits for the SHA signature and public key.

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).

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.

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 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 an expensive operation and can consume 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 and are invoked only when a prefix observes a combination of both incomplete and complete connections. 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, can it detect reachability problems?
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 6.1, questions (2),(3) in Section 6.2. Our evaluation methodology is two-fold: (a) empirically evaluate the security properties of Whisper; (b) use a real-world deployment to determine usefulness of Listen. To evaluate the security properties of Whisper, it is necessary to determine the effects of the worst-case scenario which is better quantified using an empirical evaluation.

We collected the Internet AS topology data based on BGP advertisements observed from 15 different vantage points over 17 days including Routeviews [8] and RIPE [7]. The

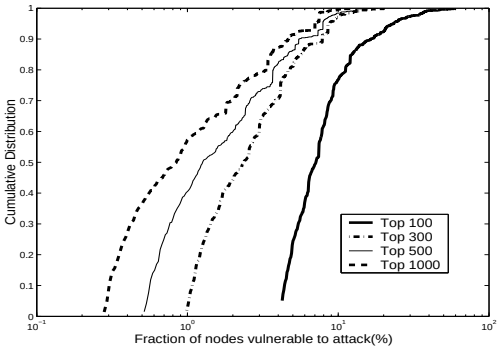

Figure 7: Effects of penalty based route selection

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 [34].

6.1 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 SHA-based 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 route selection 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 [19]. 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.

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

Table 2: Listen: Summary of Results

With AS's deploying penalty based route selection 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 route selection. We consider the scenario where the top n ISPs deploy penalty based route selection (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 use penalties. 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 (repeat the same analysis without SSW) to a randomly chosen destination AS.

Second, in the worst case, a single AS can at most affect 8% of the destination AS's for $n = 1000$. 8% is a limit imposed by the structure of the Internet topology since it represents the size of the largest connected without the top 1000 ISPs. One malicious AS in this component can potentially affect other AS's within the same component.

Third, if all provider AS's use penalties for route selection, the worst case behavior can be brought to a much smaller value than 8%. Additionally, there is very little benefit in deploying penalty based route selection 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 route selection can guarantee that a randomly placed isolated adversary propagating invalid routes can affect at most 1% of the AS's in the Internet topology.

6.2 Listen: Experimental Evaluation

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

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

we can drastically reduce the probability of such false negatives due to such connections. Second, we 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. We detected reachability problems to 578 different prefixes with a very false negative probabilities of 0.95% and 0.37% respectively due to spurious outbound and inbound connections.

We will now describe our deployment experience in greater detail. In our testbed, we use three active probing tests to verify the correctness of results obtained using Listen: (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. These tests are activated for every incomplete connection. We classify an incomplete connection as having a reachability problem only if all the three probing tests 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 a $/24$ network for over 40 days. In reality, we found that nearly 10% of the connections are incomplete of which a large fraction of these connections are spurious (91% inbound and 63% outbound). 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 being 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 propose defensive measures to reduce the probability of false negatives due to such connections.

6.2.1 *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 that 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.

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 $1/pop(P)$ [31]. 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 from 91.83% 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.

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 and by setting $N = 2$, we tend to omit these cases. In practice, the value of N is dependent on the diversity of traffic to a 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.

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.

6.2.2 *Detected Reachability Problems*

Among the reachability problems detected by Listen, two specific types of routing problems (as detected by active probing) include: *routing loops* and *forwarding errors* due to unknown IP addresses. We detected routing loops using traceroute and inferred forwarding errors using the routing table entries at the University exit router. A forwarding error arises when the destination IP address in a packet is a genuine one but the router has no next hop forwarding entry for the IP address. This can potentially arise due to staleness of routes. Table 4 summarizes the number of prefixes affected by each type of problem. In particular, we observe routing loops to 51 different prefixes and forwarding errors to 64 different prefixes. Additionally, Listen detected 463 prefixes having other forms of reachability problems.

To cite a few examples of reachability problems we observed: (a) 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 due to a routing loop. (b) The route to Yahoo-NET prefix 207.126.224.0/20 was fluctuating. During many periods, the route was detected as unavailable.

7 Colluding Adversaries

Additional to acting as a group of isolated adversaries, colluding adversaries can tunnel advertisements and secrets between them and create invalid routes with fake AS links without being detected by the Whisper protocols. These invalid routes are not detectable even with a PKI unless the complete topology is known and enforced. Despite the limitation, we can provide protective measures for avoiding these invalid routes.

Given the hierarchical nature and the skewed structure of the Internet topology, the invalid paths from colluding adversaries not detectable by the Whisper tend to be longer in AS path length. This is because, a normal route would traverse the Internet core (tier-1 + tier-2 ISPs) once while a consistent invalid route through 2 colluding adversaries traverses the Internet core twice (since the adversary cannot remove any AS from the path). Hence, by choosing the shortest path we have a better chance of avoiding the invalid route. Figures 8, 9 and 10, illustrates this effect of colluding adversaries for 3 scenarios: (a) the current Internet with

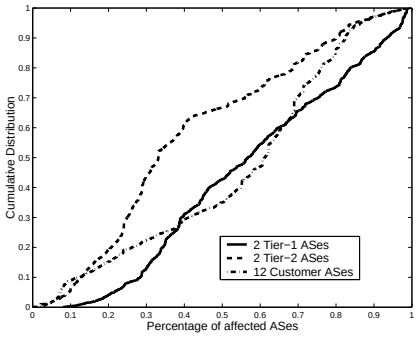

Figure 8: The effects of colluding adversaries in the current Internet.

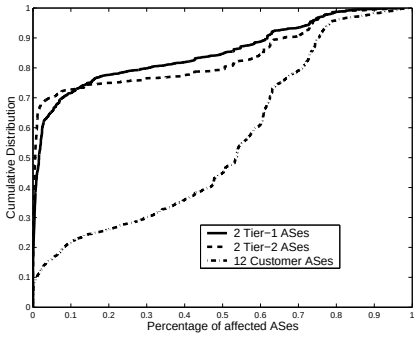

Figure 9: Effects of colluding adversaries with whisper + policy routing.

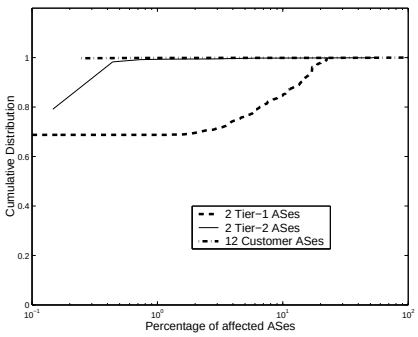

Figure 10: Effect of colluding adversaries with whisper + shortest path routing

no protection; (b) whisper protocols with policy routing; (c) whisper protocols with shortest path routing. All these graphs show the cumulative distribution of the vulnerability metric (defined in Section 6.1) for a set of colluding malicious adversaries. We specifically consider three cases: (a) 2 colluding tier-1 AS's; (b) 2 colluding tier-2 AS's (c) 12 colluding customer AS's.

We make two observations. First, 12 randomly compromised customer routers can inflict the same magnitude of damage as that of two tier-1 nodes illustrating the effect of colluding adversaries in the current Internet. Typically, customer AS's are easier to compromise since many of them are unmanaged. Second, whisper protocols with shortest path routing drastically reduces the possibility of colluding adversaries (in comparison to policy routing) propagating invalid routes without triggering alarms. In particular, even when 12 customer AS's are compromised, the effect on the Internet routing is negligible.

Whisper protocols with policy routing offers much lesser protection since BGP tends to choose routes based on the *local preference*. The typical policy convention based on stable routing and economic constraints is to prefer customer routes over peer and provider routes [19]. This preference rule increases the vulnerability of BGP to pick consistent invalid routes from customers over potentially shorter routes through peers /providers. In principle, this problem also exists in S-BGP. To strike a middle ground between the flexibility of policy routing and this vulnerability, we propose a simple modification to the policy engine: *Do not associate any local preference to customer routes that have an AS path length greater than* 2 (any route from a pair of colluding route should 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 policies (emulating shortest path routing) can largely restrict the damage of colluding adversaries.

8 Discussion

We now discuss three specific issues not covered earlier.

Hijacking unallocated prefixes: With the deployment of Whisper, a malicious adversary can still claim ownership over unallocated address spaces without triggering alarms by propagating bogus announcements. One way of dealing with this problem is to request ICANN [3] to specifically advertise unallocated address spaces with its own corresponding Whisper signatures whenever it notices an advertisement for an unallocated prefix. Additionally, to avoid a DoS attack on ICANN for such prefixes, routers should not maintain forwarding entries for these prefixes.

Route Aggregation: Whenever an AS aggregates several route advertisements into one, it is required to perform one of the following operations to maintain the consistency of the aggregated route: (a) Append the individual signatures corresponding to each advertisement so that an upstream AS can match at least one of the signatures with the whisper signatures for alternate routes to sub-prefixes. (b) If the AS owns the entire aggregated prefix (common form of aggregation in BGP), ignore the whisper signatures in the sub-prefixes and append its own whisper signature.

Other types of security attacks: Other than propagation of invalid routes, one can imagine other forms of routing attacks or misconfiguration errors which may result in routing loops, persistent route oscillations or convergence problems. Such problems are out of the scope of this paper.

9 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 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 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 changes to BGP, while Whisper can be integrated with BGP without changing the packet format.

Acknowledgments

The anonymous reviewers and Amin Vahdat, our shepherd provided us with invaluable feedback which helped substantially towards improving the quality of the paper. Tom Anderson, Alexandros Batsakis, Anand Desai, Nick Feamster, Mark Handley, Chris Karlof, Ratul Mahajan, Satomi Okazaki, Vern Paxson, Adrian Perrig, Jennifer Rexford, Dawn Song, Doug Tygar and David Wagner provided several technical comments on this work. Krishna Gummadi and Konstantina Papagianakki provided us with valuable data for empirically evaluating our Listen algorithm. Several students in Berkeley and Johns Hopkins University read earlier drafts of this paper and provided very useful feedback. The authors would like to thank them all.

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] D. F. Chang, R. Govindan, and J. Heidemann. Temporal and topological characteristics of BGP path changes. In *IEEE ICNP*, 2003.

[16] 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`.

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

[18] 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`.

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

[20] 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.

[21] 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.

[22] 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.

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

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

[25] 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.

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

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

[28] 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.

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

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

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

[32] B. Schneier. *Applied Cryptography*. John Wiley & Sons, Inc., 1996.

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

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

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

[36] 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.

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