Peer Pressure:
Exerting Malicious Influence on Routers at a Distance

Max Schuchard[1], Christopher Thompson[2], Nicholas Hopper[1], and Yongdae Kim[3]

[1]Department of Computer Science and Engineering, University of Minnesota, Twin Cities
[2]Department of Electrical Engineering and Computer Science, University of California, Berkeley
[3]Department of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST)

***Abstract*—Both academic research and historical incidents have shown that unstable BGP speakers can have extreme, undesirable impacts on network performance and reliability. Large amounts of time and energy have been invested in improving router stability. In this paper, we show how an adversary in control of a BGP speaker in a transit AS can cause a victim router in an *arbitrary* location on the Internet to become unstable. Through experimentation with both hardware and software routers, we examine the behavior of routers under *abnormal* conditions and come to three conclusions. First, that unexpected but *perfectly legal* BGP messages can place routers into those states with troubling ease. Second, that an adversary can implement attacks using these messages to disrupt the function of victim routers in arbitrary locations in the network. And third, modern best practices do not blunt the force of these attacks sufficiently. These conclusions lead us to recommend more rigorous testing of BGP implementations, focusing as much on protocol correctness as on software correctness.**

I. Introduction

Routers are a critical piece of the Internet infrastructure. They provide path discovery and selection services needed for hosts on the Internet to communicate with each other. We say that a router is stable when it exhibits three properties: high up-time, long lived BGP sessions, and a converged view of the network. It is easy to see that a router meeting these criteria will be able to provide IP layer forwarding services. A router demonstrating instability, on the other hand, will fail to demonstrate one or more of these qualities. It is well known that routers suffering from instability will be unable to perform their duties. Historical incidents, such as the Code Red and Slammer worm events [4, 12, 23], cable cuts [20], and improper configurations [19, 25], only serve to emphasize this fact. For the most part, however, these accidents have been rare and the overwhelming majority of the time routers on the Internet are stable.

While routers function well under *normal* conditions, there is one obvious question: What happens if one router forces another to operate under *abnormal* conditions? In this paper, we will demonstrate that an adversary in control of a router can cause an *arbitrary* honest router on the Internet to fail, even if the adversary is *not directly connected to the victim*. We present the results of a collection of experiments on hardware and software routers running the Border Gateway Protocol (BGP) to illustrate three key points. First, that unexpected but *perfectly legal* BGP messages can place routers into unstable states with troubling ease. Second, that an adversary can implement attacks using these messages to disrupt the function of victim routers in arbitrary locations in the network. And third, that modern best practices do not blunt the force of these attacks sufficiently.

Through experimentation on hardware and software routers, we examined what happens when modern routers find themselves without free memory. In all cases, we found that routers fail to handle this scenario gracefully. We witnessed a variety of failure modes, ranging from severe performance degradation to the unrecoverable failure of all active routing sessions. Given the negative impact of these unstable states, one might be tempted to believe that placing a router in such a state is difficult. Sadly, it turns out that the opposite is true. We found it relatively easy to send BGP messages to a router that would directly place it into one of these states. We focused on exploring corner cases that are unlikely to be found "in the wild" but are still perfectly valid in the eyes of BGP. We will examine examples of these messages later in the paper.

While it is unlikely a router would see any of these messages normally, they are easily generated by an adversary. By utilizing these BGP messages, an adversary who controls a BGP speaker is capable of launching powerful attacks against other routers. We will show how our adversary can manipulate honest routers into propagating these malicious BGP messages across the network, allowing our adversary to attack routers not directly connected to himself. In addition, by triggering loop detection mechanisms, the attacker can contain attack updates so that they are only seen by routers on paths between the malicious router and victim.

It might be comforting to assume that deployed routers could be hardened to these attacks via proper configuration. We demonstrate that the commonly accepted best practices would do little to slow these attacks. We examine four options in detail: prefix filtering, prefix aggregation, prefix limits, and AS path length limits. In each case, we use observations based on the contents of real world routing

tables to reason about both the extent to which these best practices are used and the degree to which these practices could prevent our attacks.

The contributions of this paper are fourfold. First, we provide experimental evaluations of both hardware and software routers in abnormal operating conditions. We validate and expand upon previous work looking at memory issues in routers and investigate causes of CPU exhaustion. Second, we examine the response of multiple BGP implementations to unexpected inputs. We present clear scenarios, along with experimental evidence, that illustrate the implementation failings of two commonly used BGP daemons. Third, we present clear scenarios to underscore how an adversary might take advantage of these unexpected inputs. Fourth, we look at why current best practices provide an insufficient defense against these attacks.

The rest of this paper is organized as follows. In Section II, we will discuss background material relevant to understanding this work. Then, in Section III, we will examine how an adversary can attack the functionality of a victim router to which he can directly send messages to. In Section IV we will expand upon these attacks, showing how they can be applied not just to directly connected peers, but to victims located at arbitrary positions in the network. In Section V, backed up with experimental observations, we will examine why operator best practices fail to blunt these attacks.

II. Background

A. Routers

Routers are network hosts tasked with building paths to end destinations in layer three networks, most notably the Internet. In order to accomplish their task, routers exchange reachability information with other routers using a routing protocol. We will discuss BGP, the routing protocol we focus on in this work, in Section II-B. Routers are often, but not always, responsible for forwarding data plane traffic using the paths they have built. Routers can be broadly partitioned into two categories: hardware routers and software routers.

Hardware routers are constructed using high-performance, highly specialized components in order to cope with the task of forwarding millions to billions of packets per second. Hardware routers are costly pieces of equipment and represent a large capital investment by operators. Because of this, hardware routers typically have little in the way of spare resources. A clear example of this is the route processor's memory. Modern routers generally have between 256 and 4096 MB of memory [2, 11]. In contrast to highly specialized hardware routers, software routers are built using commodity hardware, and have access to the same level of resources any desktop computer does. However, they lack the high performance line cards and switching fabric of hardware routers, preventing them from handling packet volumes typically found on today's data plane.

B. BGP

Throughout the course of this paper we will focus on routers running the Border Gateway Protocol, or BGP [15]. BGP is the current de facto standard routing protocol spoken between a pair of routers in different Autonomous Systems, or ASes; this key role makes it vitally important. BGP is a path vector routing algorithm with policies. These policies are used to augment the route selection process, allowing decisions to be made based on business relationships rather than path length.

Neighboring routers connect to each other and establish a *BGP session*. A router can advertise a path to any other router it currently has a BGP session with. To do this, it sends a BGP UPDATE message containing the block of IP addresses reached by the path and a collection of path attributes. The receiving router then stores this information in a Routing Information Base, or RIB, and recalculates the best path to the listed block of IP addresses from available paths. Update messages are required to have certain attributes: the path (by Autonomous System number) to the destination, whether the route was learned from a peer inside or outside of this AS, and the next hop in the path. Updates can also contain optional attributes, such as Community Attributes [14].

In order to understand why instability in BGP speakers is so problematic, we must introduce the concept of convergence. BGP is a distributed routing protocol, meaning that routers do not have global knowledge; information about the network, therefore must come from other nodes. When routers have settled on a consistent view of the network we call this *convergence*. When BGP speakers have converged, network traffic will flow correctly. It is a well-studied fact that this guarantee does not hold when the network has not converged. Why this lack of convergence causes the data plane to fail has been studied extensively in the work of Feamster et al. [7], Wang et al. [22], Pei et al. [17], and others. When a BGP session fails, routers are forced to withdraw all routes learned via that session, remove the routes from their forwarding tables, recalculate best routes to affected prefixes, and send out updated advertisements. In otherwords, when router become instable, the network is no long er converged, and ceases to function correctly.

III. Crashing Routers

In this section we will outline the mechanisms an adversary in control of a BGP router can use to disrupt the functionality of victim routers. We will examine how an adversary can consume all of a target's free memory, causing the target to crash. As covered in Section II-A, routers have two different types of memory, small amounts of high speed memory for line card operation, and larger amounts of general purpose memory for control plane operation. We are interested in the latter, consequently for the rest of the paper when we refer to *memory* we mean general purpose

memory. Previous research by Chang et al. [1] examined what happens when a router runs out of free memory. They found that when a router exhausts its free memory, the BGP process crashes, causing the failure of all BGP sessions and the halting of data plane operation.

Our observations in this and later sections come from experiments run on hardware and software routers. The hardware router we had access to was a CISCO 7603 series router. It is important to note that in this work we are experimenting with the behavior of a router's *software*, not its hardware. Our 7603 is an acceptable test router since it runs the same version of IOS deployed on many larger and more powerful CISCO routers. For a software router we selected the Quagga suite. In order to ensure isolation, our software routers were run on Qemu virtual machines running Linux. Our experiments in this section were done with a simple topology where attacking routers, in this case BGP injectors, were directly connected to the victim router. In Section IV we will expand to more complicated topologies to explore how our attacks function on distant targets.

While we will not discuss it here, it should be noted that in our experimention we also came across several ways to exhaust the CPU resources of a router. The methods we experimented with included crafting single updates that, when propagated through the network, will cause the targeted router to destroy one or more active BGP sessions with other honest peers and algorithmic DOS. In both the attacks against BGP sessions and the algorithmic DoS attacks, we utilize valid advertisement messages that are designed to take advantage of slow or buggy code designed to handle odd "edge cases" found in BGP. Full details of these attacks can be found in our tech report [1].

A. Available Router Memory

The immediate question that springs to mind is: how difficult is it to force a router to exhaust its supply of memory? Clearly this is dependent on how much free memory a router has. The amount of total memory varies widely based on exactly what model of router is used and where in the network it is deployed. As discussed in Section II-A, routers commonly have between 256 MB to 4 GB of *total* memory. However, what we are interested in is the *free* memory. The main demand placed on a BGP router's memory comes from storage of the Routing Information Bases, RIBs, which are tables of known valid routes, currently used routes, and currently advertised routes. It is easy to see now how a router's position in the network will alter the demands on its memory. If a router is located in the dense core of the Internet, where it has a large number of peers, the majority of which are advertising a global BGP table, it will have higher memory usage compared to a router that exists on the fringes of the Internet, where it might only receive one or two global BGP tables along with a collection of very small tables.

Because of this diversity, building an exact model of router memory usage is impossible. We can build a rough estimate by examining both how much memory a single global routing table requires and how many of those tables a router could in theory receive. The first quantity can be measured directly. We collected global routing tables from April of 2012 via RouteViews [21] and performed a series of experiments involving our routers. We advertised global routing tables to both types of router and measured their memory usage. The plot of measured memory usage versus the total number of prefixes can be seen in Figure 1a. We then examined a sample of representative routers, noting how many line cards the routers can have, presenting an upper bound on the number of full tables a router could receive [2], and their total memory. We can combine this data and the results in Figure 1a to build an upper bound on the amount of free memory a router could have as a function of the fraction of its line cards receiving a global table from a neighbor, shown in Figure 1c. While this figure does not give us a definitive target for memory consumption, it does give us an idea about the state of free router memory. Smaller routers, like the 7603, will more than likely have little to no free memory available, while larger, more powerful routers will not have free memory in abundance, likely no more than 1 GB.

B. Consuming Memory via Updates

With a ballpark figure in mind, we can examine how an adversary would consume the memory of a target router. Our basic approach is simple: an adversary will send BGP advertisements to the victim router, which will in turn store those advertised routes in its RIB. The challenge for the adversary is to craft BGP routes that take up as much space as possible. It is important to note that the routes the adversary sends need not be considered the best by the victim, they need only be considered valid. We set out to establish what is the maximum amount of memory an adversary could consume per BGP route accepted. Routes crafted using various strategies were advertised to the target router, and its resulting memory load measured.

An starting point for building an update construction strategy would be to revisit the experiments done by Chang et al. [1], where injectors simply advertised successive IP blocks with a single hop AS path. We re-ran their experiments with our CISCO router, the results of which can be seen in Figure 1b. We estimate that it would take on the order of 2 million routes in order to exhaust the free memory on

[1] http://www.cs.umn.edu/research/technical_reports.php?page=report&report_id=11-030

[2] Routers could in theory have more BGP sessions than they have line cards, via multi-hop BGP sessions, however these are not commonly used in practice.

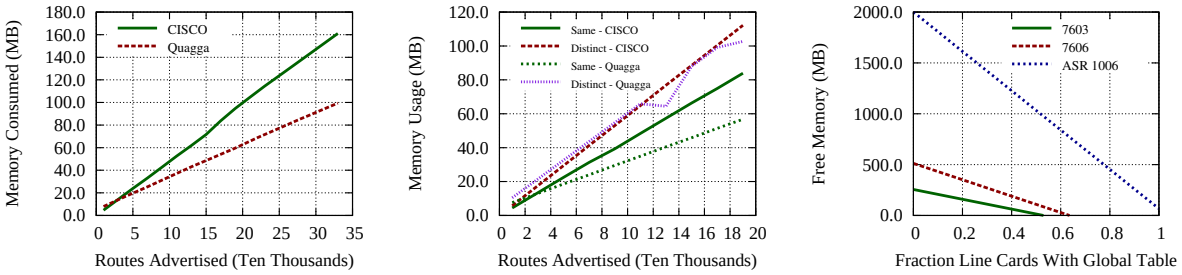


Figure 1: Measured memory consumption of BGP process as a function of the number of real world routes advertised to it in Figure 1a and repeating the Chang et al. experiments compared to distinct routes in Figure 1b. Figure 1c shows an estimate of the upper bound of free memory in various models of CISCO router as a function of the number of line cards receiving full global routing tables from peers.

modern routers using their methods, but can an adversary do better?

The first thing to notice is that all of the AS paths being advertised in the first set of experiments are identical, simply the ASN of the advertising router. In an effort to shrink the memory footprint of RIBs, routers only store identical AS paths once. In fact, if one compares the memory usage in Figure 1b, where all AS paths are the same, to the memory usage from storing real world routes in Figure 1a, where there is some degree of path distinctness, one can quickly see that the small amount of path distinctness results in increased memory load. Therefore, if the adversary can ensure all of the routes being advertised have distinct paths, we should see a larger increase in memory usage. Figure 1b shows memory consumption when routes have distinct AS paths compared to routes with identical AS paths. This increase in memory load of 33% is a start, but clearly the adversary still needs to consume more memory per update for the attack to become realistic.

The next observation is that we can make the AS paths longer, forcing the router to spend more memory storing them. While the BGP RFC allows AS level paths of any length so long as they are packaged in the update message correctly, software bugs related to long AS paths and the practice of limiting AS paths (discussed in Section V-E) constrain our adversary's ability to create *arbitrarily* long paths. Of course, paths longer than the single hop used by Chang et al. are still possible. We ran another series of injections using distinct AS paths of varying lengths. The per path memory consumption as a function of path length can be seen in Figure 2a. While the Quagga router allocates memory proportionally based on AS path length, we can see that the CISCO router instead allocates memory in a fixed size block for AS paths longer than 21 and switches to a proportional allocation only for paths longer than 120. This means that by advertising AS paths of length 22, the adversary can consume the same amount of memory he could by advertising paths of length 120. This is important, as AS paths of length 22 are smaller than limits imposed by current best practices, something we will cover in more detail in Section V-E.

Our last observation is that our simple update, even with an above normal size path, only takes up a small fraction of the total available bytes in a BGP update message. The adversary can fill the remainder of that space with Community Attributes [14] in an effort to consume more memory. Community Attributes are well known optional transitive attributes which allow operators to specify arbitrary path properties. The impact these attributes have on memory usage are highly similar to AS paths: each unique community attribute needs to be stored in a receiving router's RIB, and increasing the number of community attributes increases memory consumption. We repeated our route injection experiments, padding the update messages with unique community attributes. A plot of memory usage as a function of accepted routes can be seen in Figure 2b. Memory values beyond the capacity of our CISCO router, are extrapolated. We can see that the combination of distinct AS paths, increased modestly in length, and surrounded with unique community attributes increases the per route memory consumption by a factor of 7.48. The change is dramatic. Instead of needing more than 2.2 million routes to consume 1 GB of memory, 300 thousand routes can accomplish this task, a fraction of the size of today's global routing table. In fact the *total* memory of our CISCO router could be consumed with 76 thousand routes.

C. Exhaustion Through I/O

An alternative tactic for exhausting a router's free memory is by increasing the demands from its I/O buffers. We observed crashes that resulted from running out of memory in our victim router when its *neighbors* became CPU starved. To understand why this occurs, we must take a look at what happens when the rate of incoming updates to a router exceeds its computational capacity. In this case the receiving router will have to buffer the unprocessed updates. We found that both our Quagga and CISCO router will only buffer a fixed number of BGP messages. When those limits have been reached, the BGP process will stop fetching packets

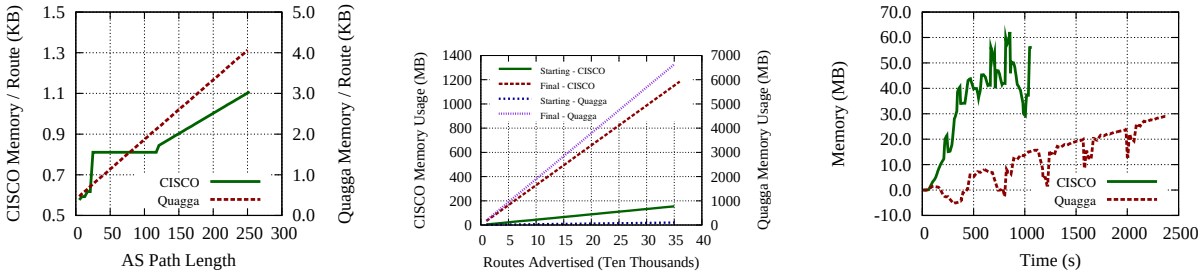


Figure 2: Figure 2a shows the per update memory usage as a function of path length for Quagga and CISCO routers for both unique and identical sets of paths. Note how Quagga's memory allocation is always a function of the path length, while CISCO allocates fix size blocks for all paths between 24 and 108 ASes in length. Figure 2b shows the measured memory consumption when applying path distinctness, increased AS path length, and distinct community attributes compared to Chang et al.'s attack. Figure 2c shows the increase in memory load for a Quagga and CISCO router when advertising to a CPU starved peer compared to a normal peer, demonstrating exhaustion of memory via I/O backlog.

from the operating system's buffers. Network buffers are of fixed size as well—when the receiving router's network buffer is full, it will send TCP ZERO WINDOW messages to the advertising router, preventing new packets from being placed on the wire. New packets are then buffered in the *sender's* network buffers. When those fill, the updates are buffered inside the advertising router's BGP process. These buffers are *unbounded* in size. We term this behavior *back pressure*.

We performed a simple experiment to illustrate back pressure involving three routers: an injector, a victim router, and a potentially computationally starved peer. The injector was directly connected to the victim router, as was the CPU strained router. All routers started with empty tables, and the injector would proceed to advertise routes taken from RouteViews to the victim router, which would in turn attempt to advertise them to its other peer. Runs were performed when the third peer was both CPU starved and when it had sufficient computational resources. The *difference* in memory loads of the victim router when it's peer was CPU strained versus when it was not can be seen in Figure 2c. The CISCO router experienced an increase in memory consumption of more than 40 MB, with spikes over 60 MB. Quagga saw an increase of 30 MB by the end of our experiment. It is important to note that this was for a single CPU strained peer, and that these added memory costs are *per peer*, meaning that routers with more peers are more susceptible to this attack.

This increase in memory usage was not the strangest behavior that resulted from update back pressure. On the CISCO router, we noticed that if the amount of back pressure was large enough, the processes controlling BGP I/O started to fail. Specifically, it ceased interacting correctly with the peers responsible for the back pressure. The memory-starved router ceased attempting to send BGP related packets to these peers. We assumed tearing down the BGP session, which would result in a new TCP session, would solve this I/O issue. It did not. While the back pressure causing peers could complete a TCP handshake, no response to their BGP OPEN message came from the CISCO router. This I/O issue was limited to BGP, however, as we could initiate a telnet console with the CISCO router from the Linux box hosting the troubled peer. This problem was only fixed when the CISCO router was restarted.

Of course in order for back pressure to exist a router's neighbors need to be in a CPU starved state. This can occur for a variety of reasons. The most obvious one is as a result of nearby router failures. As discussed in Section II-B router failures push the network out of convergence, forcing routers to spend processing power recomputing the best paths to large swaths of the IP address space.

IV. Attacking Distant Routers

The attacks of Section III provide a peek into how a malicious router can force a victim into a non-functional state via legitimate BGP messages. However, one might feel skeptical about these examples, as they were tested with an adversary that is directly connected to its victim. In this section, we will provide examples of how an adversary in control of a router could force other routers *at arbitrary locations in the topology* into an unstable state. We will lay out how our adversary can launch this attack by convincing honest routers to forward these malicious updates through the network to the victim while at the same time minimizing his direct impact on other routers in the system.

A. Threat Model

Our threat model focuses on legitimate BGP speakers in transit ASes [3] that have become malicious. These adversaries are the result of either an autonomous system electing to act in an adversarial manner or an outside entity compromising one or more BGP routers. We focus on transit ASes since stub ASes have very limited abilities within the BGP network. Our chosen threat model gives our adversary two key capabilities.

[3]By transit AS, we mean any AS that has other ASes as customers.

First, the adversary can send BGP messages to other routers. The malicious router cannot simply send arbitrary messages to *any* router, however; it can only directly send BGP messages to its legitimate peers. This is an issue for our attacker, as his previously stated goal is to disrupt *arbitrary* routers or BGP sessions, not simply those he is directly connected to. In order to have malicious update messages reach arbitrary routers, our adversary will need to convince honest peers to propagate those updates in such a manner that the intended victims receive them. We will cover how our adversary does that in Section IV-B.

The second ability our adversary has is the capacity to act in a non-standard, or even protocol non-compliant manner. Our adversary can, for example, locally ignore paths with loops, use non-standard path selection, not apply best practices, and advertise paths in a manner that does not conform to Valley Free Routing. However, our adversary again runs into the issue that *only* he can act in this way; honest nodes will act normally and can use best practices. We will cover how these attacks work in relation to best practices in Section V.

B. Propagating Malicious Updates

In our threat model the adversary only has the capacity to send update messages directly to his legitimate peers. In order to get malicious updates to targeted routers, the adversary will need to convince routers that lie on paths between him and his victim to forward the updates. Honest routers only re-advertise the routes they consider "best". This is an issue for our adversary because, as we have seen in Section III, some of the malicious updates will have a longer than average AS path length. Because AS path length is one of the key path selection metrics, this will make it less likely that the malicious updates will be considered best if there is an alternative.

If our attacker could advertise IP blocks that have no competing paths, the malicious routes would be the best by default. To do this, our attacker will take advantage of the fact that BGP considers more specific IP prefixes to be distinct. For example, BGP considers the IP block `123.101.0.0/16` to be distinct from `123.101.128.0/17` and `123.101.0.0/17`. Since these blocks are considered distinct, path selection for `123.101.128.0/17` will be done separately from `123.101.128.0/16`. Our adversary simply couples his malicious advertisements to highly specific IP blocks (e.g. `123.101.128.0/24`) for which there are not pre-existing routes. Due to this forced de-aggregation, his malicious updates will have no competition and will be the best. We discuss how this tactic interacts with best practices such as aggregation and prefix length filtering in Section V.

C. Building Attack Flows

The adversary cannot blindly send these "best paths" out to all of his peers in the hope that they eventually reach his target. Our adversary will take advantage of the fact that honest routers operate in a predictable manner in order to construct "flows" of updates from himself to the victim. When determining whether to propagate a best path to its neighbors, a router takes into account the customer/provider relationships it has both with its neighboring routers and with the route's next hop. A well known set of policies called Valley Free Routing [8] are applied based on these relationships. Valley Free Routing states that a path will be advertised if and only if: a) the party being advertised to is the AS's customer or b) the route was learned from a customer. While the AS relationships are technically private information, a large amount of work has been done to infer them. By building a topology based on a data set of these relationships between ASes [21] and applying Valley Free Routing policies to this topology, a model for how the malicious routes will travel through the network can be built. Central to this construction is the concept of a *customer cone*. An AS's customer cone is the set of all of its customers, plus all of its customer's customers, and so on.

With this model of path propagation in mind, the adversary can construct a directed graph based on Internet topology and inferred AS relationships. The adversary starts with an edge-less AS graph. The adversary then adds a directed edge between his AS and every AS he is directly connected to, this represents his ability to send advertisements to any peer he is directly connected to. For each of the ASes added to the connected component containing the adversary that are part of the adversary's customer cone, an edge is added from that node to its customers. This represents the "A" clause of Valley Free Routing. If instead the AS added to the connected component is the adversary's provider, an edge is added from that AS to all of its customers, peers, and providers, the "B" clause of Valley Free Routing. If a path exists from the adversary to the victim, that means that there is some neighbor of the adversary which, if the adversary sends an update, will start up a chain of advertisements that will end with the update reaching the victim. We call such a path through this directed graph an *attack flow*. In a densely connected graph such as the Internet, there will typically be multiple paths, which will allow the adversary to load balance his malicious updates. We will examine the prevalence of attack flows in Section IV-D.

Once attack flows are found, it might be in the attacker's interest to contain updates to only their assigned flow. The adversary must prevent routers that are next to the attack flows from accepting the malicious routes. To do this, our attacker can use loop detection to his advantage. In BGP, loop detection is achieved by scanning the AS

path for the router's ASN. If the router detects itself in the path, it considers the route in-feasible, neither storing it in memory nor propagating it. It is important to note that when loop detection is triggered it does *not* result in a BGP NOTIFICATION. The first step for our adversary is to define the "borders" of each attack flow. These are all of the nodes in the directed graph which are *not* part of the attack flow, but have an edge leading from a node in the attack flow to themselves, we call these nodes *bystander nodes*. These are the nodes that will see the malicious updates, but have nothing to do with the actual propagation of the updates to the victim. He then ensures that the ASNs of all neighbors of the attack flow are included inside the fabricated AS path of any update utilizing that flow. Since the neighbors of the flow will not propagate the malicious updates, the routers behind those neighbors will never see the updates.

D. Experimental Observations

In order to validate that our attack works, we utilized our software router test-bed from Section III. Our topology was a small subset of the AS level topology of the Internet built by doing an expansion from a node chosen at random. An example topology including AS relationships can be seen in Figure 3a. We launched the attack we have just described from random attackers to random victims and monitored the memory of routers in the test-bed. We set a goal, based on our findings in Section III-A, of 1 GB of additional memory consumption. The routers in the test-bed fall into one of three different groups. First, there are routers that are on the attack flows from the adversary to the victim. Those routers will see increased memory utilization, but *not* enough to push them over the 1 GB threshold. Second, there is the victim, who should see memory consumption *over* the 1 GB threshold if the attack is successful. The last group of routers are those directly connected to the victim or an attack flow, but not actually part of the flow. Our loop detection technique should cause them to see *no* additional memory load.

A collection of memory traces from an example run can be seen in Figure 3b. In this run two attack flows were used by the attacker to push malicious updates. The injection of malicious routes to directly connected neighbors starts at time 0 and completes at roughly time 2900. As can be seen, the attack flow routers show an increase in memory that ends with them having less than the 1 GB barrier, exactly what we expected to see. The victim router's memory load lags behind the average memory load of the attack flow routers, a result of it being the last router in the chain that the updates propagate to. At time 5200 we see the victim router cross the 1 GB threshold. In our case the router did not crash as the virtual machine had added memory to handle this load. The last set of routers, those protected by loop detection, showed zero increased memory load, as all of the incoming routes were discarded as loops.

1) Counting Attack Flows.: Another interesting question to ask is how many attack flows an adversary is likely to have available for a given victim. To answer this question, we examined the AS level topology (considering only those ASes that meet the threat model discussed in Section IV-A) and calculated the attack flows between each pair of ASes, counting the number of flows. The results of this calculation appear in Figure 3c.

As the figure shows, in the worst case a random adversary has a 73.5% chance of possessing multiple attack flows to a random victim; almost half of all pairs have at least 3 attack flows. Furthermore, adversaries in the "transit core" of the Internet are even more advantageously situated: for example, around 10% of tier 2 routers can access 20 or more attack flows when targeting a tier 1 router; and if an adversary is in control of a tier 1 router, at worst he has roughly 60 attack flows open to him, and on average far more. This supports our contention that our attacks can indirectly target arbitrary routers in the AS topology.

V. Defeating Best Practices

In this section, we examine how various "best practices" interact with the proposed attack of Section IV. We focus on the following practices: prefix length limits, prefix filters, prefix aggregation, prefix count limits, and AS path limits. We will show that each of these fails to disrupt the adversary's actions to any sizeable extent. A summary can be seen in Table I. While not covered here, in our tech report [4] we also consider how a global deployment of BGPSec [16] would impact our adversary.

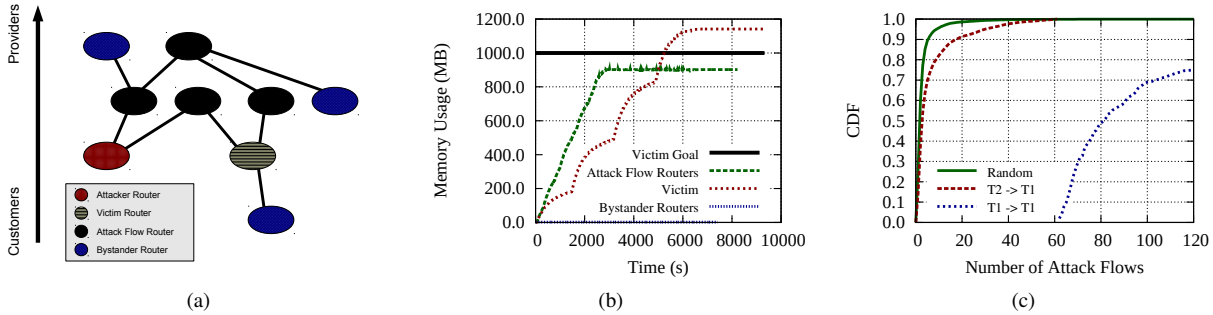
A. Prefix Length Limits

One commonly applied best practice is to drop updates for highly specific prefix blocks. Filtering in this manner is done in an effort to control the size of routing tables. This policy is an issue for our attacker because, as discussed in Section IV-B, our adversary relies on advertising very specific prefix blocks which do not have pre-existing paths. Two questions are raised because of this practice.

First, how specific of a prefix can our adversary advertise without it being filtered? To answer this, we examined what length of prefixes we can *actively observe* being forwarded by various Autonomous Systems based on RouteViews data, the results of which can be seen in Figure 4a. What we found was straightforward: 88.5% of transit ASes forwarded prefixes that were /24s or shorter, while 6.8% forwarded prefixes longer than this. Thus, in the majority of cases, our adversary can advertise routes containing a /24 or shorter successfully.

This leads us to our next question: Given that we can advertise no more specific a prefix than a /24, can our adversary find enough un-advertised prefixes to complete

[4] http://www.cs.umn.edu/research/technical_reports.php?page=report&report_id=11-030

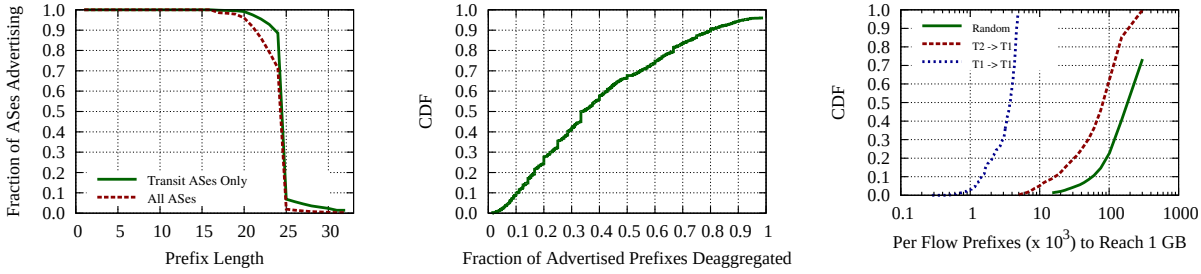


(a) (b) (c)

Figure 3: A graphical representation of one topology configuration used for our experiments is shown in Figure 3a. AS relationship is represented by vertical placement. Figure 3b is a trace of memory loads for BGP processes of various classes of routers during an update based memory exhaustion attack. Memory load for attack flow routers and bystander routers is an average over all routers in that class. Our 1 GB goal is highlighted on the graph for clarity. A CDF of the number of attack flows existing between various tiers of transit routers based on AS level topology can be seen in Figure 3c.

Best Practice	Why It Does Not Help	Experimental Evidence
Prefix Length Limits	Limits still give the attacker access to millions of prefixes	/24s advertised by 88.5% of transit ASes (Fig 4a)
Prefix Aggregation	Not done to routes from transit ASes	Observation of hole punches and non-aggregated IP blocks (Fig 4b)
Prefix Count Limits	Malicious updates target receives based on sum of victim prefix limits	Prefix limits applied on a per connection basis combined with AS level topology (Fig 4c)
AS Path Limits	Weakened by generous path limits and how Routers allocate memory	CISCO routers allocate memory in fixed size blocks (Fig 2a)

Table I: Summary of best practices we considered, why they fail to stop the attack from Section IV, and what experimental evidence backs each conclusion.



(a) (b) (c)

Figure 4: The fraction of ASes observed view RouteViews advertising prefixes of a given length can be seen in Figure 4a. Figure 4b is a CDF of the fraction of prefixes originated by transit ASes that could be aggregated into more general prefixes. Lastly, Figure 4c is a CDF of the number of malicious updates required *per attack flow* to reach 1 GB of memory consumption for different classes of attacker and target nodes.

his attacks? This can be answered with a quick back of the envelope calculation. There are approximately 1.6×10^7 prefix blocks of length /24, and of those 98% correspond to routable IPs (the other 2% are un-routable bogons [5]). The current size of the full Internet routing table is roughly 4×10^5 prefixes [10], meaning that if all IP blocks advertised were /24s then there would still be over 1.5×10^7 /24s un-advertised. Clearly this means that our adversary can find a sufficient number of un-advertised /24 blocks to utilize.

B. Prefix Filtering

Modern routers have the ability to filter incoming updates based on a combination of the CIDR being advertised and the AS sending the update. This capability, if widely and correctly deployed could have an impact on our adversary's strategy as outline in Section IV. Sadly, it is *not* a forgone conclusion that prefix filtering is deployed in such a manner. Prefix filtering must be done manually, and changes in filters requires a reconfiguration of routers. Additionally, establishing exactly what blocks of IP addresses an AS should or should not advertise is a non-trivial task. Measuring exactly how many ASes deploy prefix filtering correctly would require operators to share confidential information, specifically the active configurations of their routers. However, we can gain an intuition as to how widely and correctly prefix filters are deployed by examining several historical incidents of prefix leaks and hijackings [19, 25, 3] which were only been possible because prefix filtering is not correctly and widely used. As an example, in 2010 China Telecom accidently sent advertisements for 50 thousand blocks of IP addresses that it did not own.

C. Prefix Aggregation

Tied closely with the subject of filtering long prefixes is the concept of prefix aggregation. Upstream routers have the ability to aggregate multiple advertisements from downstream peers into a single, less specific, advertisement which they pass on to their peers. This again presents an issue for our adversary, as aggregation could cause his attack updates to be merged into a small number of aggregated routes. However, this issue is actually a non-factor for our adversary for several reasons.

First, we have to take into account how, where, and why aggregation is and is not done. Aggregation must be manually configured, and while it is fairly straightforward to aggregate updates from non-transit ASes, as these routes are essentially static stubs, this is not the case for routes from transit providers, where our attacker was assumed to be. In fact, a commonly used traffic engineering trick called *hole punching* assumes that transit providers do not forcibly aggregate each other's announcements. In hole punching, a router announces a path to both a prefix and a different path to a more specific prefix contained in the first. In this way the router can hint at different policies for this specific destination or can encourage load balancing. Using RouteViews data, we observed 569 core transit ASes actively using hole punching. The fact that hole punching is actively done is of great value to our adversary, as the way in which he builds prefixes makes them appear identical to hole punches.

Moreover, one can examine RouteViews to see exactly how many ASes aggregate routes at all. By scanning RouteViews for ASes that advertise easily aggregatable blocks, for example `1.2.1.0/24` and `1.2.0.0/24`, we can quickly get a sense for how much aggregation is actually done *in practice*. We found that 100% of transit providers are observed advertising trivially aggregatable prefixes. Figure 4b shows a CDF of the fraction of advertised prefixes seen from transit providers that could be aggregated.

D. Prefix Count Limits

A different best practice that directly impacts our adversary is limiting the number of prefixes one router will accept from another. While there have been historical incidents that call into question whether a majority of ASes actually do this [19], let us assume the best case: that all ASes follow this practice. While some of the attacks covered in Section III center around sending a single malicious path to a target, others require the adversary to send sets of paths. Therefore, prefix limiting might present an issue for our attacker: if prefix limits prevent him from sending enough paths, his attack could fail. However, when examined more closely, this turns out not to be an issue. There are two different sets of prefix limits that will interact: those that the adversary's neighbors have set for it, and those that the victim has set with his neighbors. Somewhat counter-intuitively, the actual number of prefixes the attacker can push to a victim several hops away can be higher than the number of prefixes he can push to his neighbors. This is because the attacker can set up multiple attack paths *that utilize the same first hop*. In this case the maximum number of malicious updates the attacker can send to the victim is the *sum* of the *victim's* prefix limits. We can examine our results from Figure 3c, where we examined the estimated number of attack flows an adversary would have to a given target, to get an estimation of the number of *per BGP session* advertisements an adversary would need to send for various attackers and targets. The results of this can be seen in Figure 4c. Even in the topologically worst case of a randomly chosen attacker and target, the median number of routes required is less than half of the routes in the current global routing table. This is well within the resources of our threat model. In the case of a tier 1 attacker against a tier 1 target, on average only 5000 prefixes per flow are sufficient.

Another reason prefix limits do not have a large impact on the attacker is how large these prefix limits might be. The value of prefix limits depends on where the victim sits within the Internet topology. In general the victim falls into one of two places: either on the fringe of the network or not. If the victim is on the fringe of the network, then he is expecting to receive full BGP tables from a single digit number of providers, in which case his prefix limits are set at full table size (on the order of hundreds of thousands of updates). If the victim is not on the fringe, he might be expecting smaller amounts of updates from each individual peer, ranging from tens of thousands of prefixes up to full tables. However, victims who are not on the fringe of the Internet also have an increase in their number of peers of an order of magnitude or more [21] compared to their counterparts in the fringes. This means that, even if we assume the core victim has prefix limits on the order of tens of thousands of routes, his aggregate route acceptance will be equivalent to that of the fringe victim, since the core victim has more peers. Lastly, it is advised practice to keep a safety margin of as large as 25% on prefix limits, so as to not accidentally exceed them [6]. This means our adversary can allow normal operation to continue, while using that safety margin to advertise his malicious routes.

E. Path Length Filtering

Recommended best practice is to limit the maximum accepted AS path length. Again, recent historical incidents call into question whether this is actually done [25]. Even if routers set a small AS path length limit (the current recommendation is 100 or less), we can recall back to Figure 2a in Section III-B and see that the length of the path is not a dominating factor for memory consumption in CISCO routers. With a path length of 22, each update accepted takes up 0.8 KB of memory. With a full path of

length 253, we only see a marginal improvement to 1.1 KB of memory per update accepted. In our memory consumption attack, we cause a much greater memory consumption by adding community attributes to the updates.

VI. Related Work

Previous works have examined the ability of adversaries to propagate BGP updates to routers multiple hops away. Two such examples are Pilosov and Kapela from Defcon 16 [18] and Goldberg et al. from SIGCOMM 2010 [9]. In these works the authors examine how to propagate updates for individual IP blocks, which the adversary does not own, with the goal of hijacking traffic bound for hosts inside those IP blocks. We share some techniques with each of these works. For example, Pilsov uses the technique of adding ASNs to the path in an effort to hide the updates from the victim AS. We expand upon this idea in Section IV-C, using prepending ASes to build attack flows. In addition, the goals of our attacks different greatly from these works. In prior work the adversary's goal was to use the accepted routes to hijack BGP traffic, our adversary's goal on the other hand is to use the side effects of the accepted routes to disrupt the victim router.

Issues with memory load has also been covered from different perspectives in prior work. As mentioned earlier, Chang et al. [1] examined the impact of large BGP RIBs on router functionality. While Chang et al. covered the direct impacts of a router running out of memory, they neither examine what makes routers more likely to run out of memory nor how and adversary can leverage router memory allocation to induce stability issues. In contrast, our work examines both of these items in detail. Routing table growth has also been covered extensively by the networking community [13]. These works are concerned with the growth of routing tables by non-malicious means, specifically the natural expansion of the Internet and traffic engineering techniques. Our work instead examines how an adversary can artificially expand routing table size. Possible solutions to these issues have been proposed as well [24], however these solutions focus nearly exclusively on line card memory load, and not the RIB, our adversary's target.

VII. Conclusion

In this paper we demonstrated how an adversary in control of a BGP router can disrupt victim routers located across the Internet. We have shown through experimentation with hardware and software routers three key points. First, that an adversary in control of a BGP speaker can use unexpected but *perfectly legal* BGP messages to crash victim routers. Second, that such attacks can be launched against victims located at arbitrary locations in the network by taking advantage of the natural behavior of honest routers. And third, modern best practices do not prevent these attacks.

References

[1] D.-F. Chang, R. Govindan, and J. Heidemann. An empirical study of router response to large BGP routing table load. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, IMW '02, pages 203–208, New York, NY, USA, 2002. ACM.

[2] CISCO Systems. CISCO Systems - Routers. http://www.cisco.com/en/US/products/hw/routers/index.html.

[3] J. Cowie. China's 18-minute mystery. Renesys Corp., http://www.renesys.com/blog/2010/11/chinas-18-minute-mystery.shtml, 2010.

[4] J. Cowie, A. Ogielski, B. Premore, and Y. Yuan. Global routing instabilities during Code Red II and Nimda worm propagation, 2001.

[5] T. Cymru. Team Cymru Bogon List. http://www.team-cymru.org/Services/Bogons/bogon-dd.html.

[6] T. Cymru. Team Cymru Secure BGP Template. http://www.team-cymru.org/ReadingRoom/Templates/secure-bgp-template.html.

[7] N. Feamster, D. G. Andersen, H. Balakrishnan, and M. F. Kaashoek. Measuring the effects of Internet path faults on reactive routing. In *Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, SIGMETRICS '03, pages 126–137, New York, NY, USA, 2003. ACM.

[8] L. Gao and J. Rexford. Stable Internet routing without global coordination. *IEEE/ACM Trans. Netw.*, 9:681–692, December 2001.

[9] S. Goldberg, M. Schapira, P. Hummon, and J. Rexford. How secure are secure interdomain routing protocols. *SIGCOMM Comput. Commun. Rev.*, 41:87–98, August 2010.

[10] G. Huston. BGP Routing Table Analysis Reports. http://bgp.potaroo.net/.

[11] Juniper Networks. Juniper Network Routing Solutions. http://www.juniper.net/us/en/products-services/routing/.

[12] M. Lad, X. Zhao, B. Zhang, D. Massey, and L. Zhang. An analysis of BGP update surge during Slammer attack. In *Proceedings of 5th International Workshop on Distributed Computing*, 2003.

[13] X. Meng, Z. Xu, B. Zhang, G. Huston, S. Lu, and L. Zhang. Ipv4 address allocation and the bgp routing table evolution. *SIGCOMM Comput. Commun. Rev.*, 35(1):71–80, Jan. 2005.

[14] Network Working Group. RFC1997 - BGP Communities Attribute. http://tools.ietf.org/rfc/rfc1997.txt, August 1996.

[15] Network Working Group. RFC4271 - A Border Gateway Protocol 4 (BGP-4). http://tools.ietf.org/html/rfc4271, January 2006.

[16] Network Working Group. BGPSEC Protocol Specification. https://http://tools.ietf.org/html/draft-ietf-sidr-bgpsec-protocol-04, July 2012.

[17] D. Pei, X. Zhao, D. Massey, and L. Zhang. A study of BGP path vector route looping behavior. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, ICDCS '04, pages 720–729, Washington, DC, USA, 2004. IEEE Computer Society.

[18] A. Pilosov and T. Kapela. Stealing the internet. In *Defcon 16*, 2008.

[19] A. Popescu, B. Premore, and T. Underwood. Anatomy of a leak: AS9121. Renesys Corp., http://www.renesys.com/tech/presentations/pdf/renesys-nanog34.pdf, 2005.

[20] A. Popescu, B. Premore, and E. Zmijewski. Middle east meltdown: A global BGP perspective. Renesys Corp., http://www.renesys.com/tech/presentations/pdf/apricot-plenary-08.pdf, 2008.

[21] ieee RouteViews. RouteViews Dataset. http://www.routeviews.org/.

[22] F. Wang, Z. M. Mao, J. Wang, L. Gao, and R. Bush. A measurement study on the impact of routing events on end-to-end Internet path performance. *SIGCOMM Comput. Commun. Rev.*, 36(4):375–386, 2006.

[23] L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, A. Mankin, S. F. Wu, and L. Zhang. Observation and analysis of BGP behavior under stress. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, IMW '02, pages 183–195, New York, NY, USA, 2002. ACM.

[24] X. Zhao, Y. Liu, L. Wang, and B. Zhang. On the aggregatability of router forwarding tables. In *Proceedings of the 29th conference on Information communications*, INFOCOM'10, pages 848–856, Piscataway, NJ, USA, 2010. IEEE Press.

[25] E. Zmijewski. Reckless driving on the internet. Renesys Corp., http://www.renesys.com/blog/2009/02/the-flap-heard-around-the-worl.shtml, 2009.