Department of Computer Science
and Engineering
University of Minnesota
4-192 Keller Hall
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 11-030

Taking Routers Off Their Meds: Unstable Routers and the Buggy BGP Implementations That Cause Them

Max Schuchard, Christopher Thompson, Nicholas J. Hopper, and Yongdae Kim

November 30, 2011

Taking Routers Off Their Meds:
Unstable Routers and the Buggy BGP Implementations That Cause Them

Max Schuchard, Christopher Thompson, Nicholas Hopper, Yongdae Kim
{schuch,cthomp,hopper,kyd}@cs.umn.edu

Abstract

Both academic research and historical incidents have shown the impact of unstable BGP speakers on network performance and reliability. A large amount of time and energy has been invested 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 four conclusions. First, routers placed in certain states behave in anything but a stable manner. Second, unexpected but *perfectly legal* BGP messages can place routers into those states with disconcerting ease. Third, an adversary can use these messages to disrupt a victim router to which he is not directly connected. Fourth, modern best practices do little to prevent these attacks. These conclusions lead us to recommend more rigorous testing of BGP implementations, focusing as much on protocol correctness as software correctness.

1 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 can define a router as stable when it exhibits three qualities: the router itself has high uptime, routing sessions with peers demonstrate longevity, and its view of the network is consistent with that of its peers. 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 previously mentioned 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 [7, 18, 32], cable cuts [25], and improper configurations [24, 35], 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 routers placed in certain states fail to provide their necessary functionality. Second, that unexpected but *perfectly legal* BGP messages can place routers into those states with troubling ease. And third, that an adversary can implement attacks using these messages to disrupt the function of victim routers in arbitrary locations in the network.

In this paper, we will explore three different unstable states that a router can find itself in. In the first, a router encounters a message that it views as invalid or corrupt. In the second, a router exhausts its available memory. In the last, a router finds itself starved for CPU cycles. Through experimentation on hardware and software routers, we observed what happens when routers find themselves in one of these states. In all cases, we found that routers fail to handle these scenarios gracefully. We witnessed a variety of failure modes, ranging from severe performance degradation to the unrecoverable failure of all active routing sessions. We also observed that these unstable states are strongly interrelated. We found that a router placed into one of these states would more than likely cause its peers to enter one or more of these states as well.

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 unstable 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 have developed a few example attacks in order to illustrate this point. These attacks allow an adversary to send advertisements from their compromised BGP speaker that would disrupt the operation of a target router. 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.

It might be comforting to assume that deployed routers might 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, investigate the impacts of CPU exhaustion, and provide evidence of unstable states in one router causing unstable states in neighboring routers. Second, we examine the response of multiple BGP implementations to highly 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 demonstrate why current best practices provide an insufficient defense against these attacks.

The rest of this paper is organized as follows. First, in Section 2, we will discuss background material relevant to understanding this work. Then, in Section 3, we will examine abnormal states that routers can find themselves in and the consequences of these states. In Section 4 we will construct a variety of atypical BGP messages, demonstrating through experimentation how they place routers into hazardous states. We will expand upon these BGP messages in Section 5, developing a collection of example attacks that could be used by an adversary to disrupt BGP speakers on the Internet. We compare this paper to related works in Section 6. Lastly, in Section 7, we discuss how operator best practices fail to blunt our adversary's attacks.

2 Background

2.1 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 2.2. 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 a second. The downside of hardware routers can be summed up in a single word: cost. 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 2048 MB of memory, with the largest routers tending to have 4096 MB [6, 17]. The current logic is that this amount of memory is sufficient for normal conditions, and providing larger amounts of memory undermines cost savings.

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. Software routers are generally a routing daemon running on top of a general purpose operating system. Software routers have the distinct advantage of being cost efficient, costing several orders of magnitude less than hardware routers. 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. Because of these qualities, software routers are typically deployed in specialized roles. One of the most common examples of this would be a route reflector—a specialized router that offers an alternative to the logical full-mesh requirement of internal Border Gateway Protocol.

2.2 BGP

Throughout the course of this paper we will focus on routers running the Border Gateway Protocol, or BGP [20]. 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 [19].

2.3 Instability and Failed BGP Sessions

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 [12], Wang et al [30], Pei et al [23], and others. At a high level, when BGP speakers are out of convergence, they do not have guarantees that they are routing based on fresh and valid information. As a result, packets fail to reach their destination for reasons as complex as the formation of transient forwarding loops or as simple as being routed toward a path that no longer exists. These problems only disappear when the network reconverges. This of course raises the question: what events force BGP speakers out of convergence?

To answer that question, consider what happens after a BGP session dies. 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. Over the course of our experiments, we saw our routers go through this sequence of behavior frequently. This series of withdrawals and re-advertisements forces other routers who used impacted routes to repeat this process, triggering multiple waves of path re-calculations and advertisements. In most cases routers will automatically rebuild the failed BGP session, resulting in a new round of update messages to announce the return of the previously available routes.

2.4 BGP Notifications

The BGP specification defines what does and does not constitute a valid message. It also covers how routers are supposed to respond to invalid messages. When one router sends a malformed or invalid message to another router, no matter how minor, the response is a BGP NOTIFICATION message. This is, in essence, simply an error message. When a BGP NOTIFICATION is sent, both sides of the session, regardless of the reason for the BGP NOTIFICATION, will *always* tear down the session and disconnect from each other. In other words, routers voluntarily step out of convergence in response to any error message. Routers are explicitly not allowed to use mechanisms that mitigate the effects of session failure, like BGP Graceful Restart [21], after a BGP NOTIFICATION.

This stance of "fail early, fail loudly" is powerful: it rapidly draws attention toward issues in the network. However, with great power comes great responsibility. BGP speakers need to ensure that they are not sending error messages often, as the consequence of each error message is a large amount of instability. We found two major issues with how BGP speakers perform error checking and use BGP NOTIFICATION messages. First, routers fail to contain the scope of errors. As a BGP speaker, it would be a simple enough matter for the router to check if it would consider the message it is about to send as valid *before* it sends it. This would be advantageous, as the router could handle the error message locally without causing the failure of its BGP sessions. We found that this is not currently done: routers send BGP messages to other peers that they know are malformed. Second, as we will show in Section 4, routers respond to some perfectly valid but oddly formed or unexpected BGP messages with a BGP NOTIFICATION. While it is normally acceptable to send error messages to odd or unexpected messages, this logic does not hold when the consequences of an error message are as potentially severe as it is in BGP.

3 Unstable States

As stated previously, routers function well under normal operating conditions. We were interested in how routers fair under *abnormal* conditions. A 2002 study by Chang, Govindan, and Heidemann [4] looked at what happens when a router runs out of free memory. In this work, the authors advertised routes to three different routers: a CISCO 7000 running IOS 11.1, a CISCO 12008 GSR running IOS 12.0, and a Juniper M20 running JUNOS 4.3 and 4.4, until they ran out of memory. They reported that, in situations where the space required to store the advertised routes in a RIB exceed the available memory, one of two things happened. One behavior that the routers exhibited was that they disconnected from *all* BGP sessions, regardless of the number of routes learned via each session. The second behavior seen was that the router became unresponsive, requiring a hard reboot in order to resume function. Either behavior is highly undesirable: the data plane will cease to function when a router is restarting, and the failed session will have all of the consequences mentioned previously in Section 2.3.

In this section, we revisit Chang et al's work on memory exhaustion using more modern routers, and present additional results on the effects of CPU exhaustion and how unstable states in one router can often force other routers near it into these states as well. To investigate this, we performed a collection of experiments with two routers: a CISCO 7603 series router, affectionately named "Patsy", and a Quagga software router. For a full explanation of our experimental setup see Appendix A.

3.1 Memory Exhaustion

Repeating Chang et al's experiment using more modern routers, we found that their results still hold today: routers fail when their memory is exhausted. In the case of both the hardware and software router, when the BGP process requested more memory than the host system could provide, the BGP process was killed. The death of the BGP process itself resulted in the failure of all BGP sessions currently active. Since the BGP process died, BGP Graceful Restart could not mitigate the effects of the session failure. In the case of Quagga, the BGP process died silently. No error messages were logged, nor was an explanation of session failure sent to any peer. The BGP process required outside intervention in order to restart. On the other hand, Patsy successfully logged the death of its BGP process via `rsyslog` and sent BGP NOTIFICATION messages to its peers. Its BGP process restarted automatically in all cases without the need for outside intervention.

While the BGP process being killed was the most graceful way routers handled this situation in our tests, we also saw a number of experiments where a different process, such as an internal routing protocol, was the first to request more memory after the router ran out. This resulted in that process dying *in addition* to the BGP process dying. In the case of Patsy, the CISCO Express Forwarding Engine [29] occasionally asked for memory that could not be given. In that case *all* network-based services temporarily failed.

The most troubling part is the ease with which a normal router can become memory exhausted. As covered in Section 2.1, hardware routers do not possess an abundance of memory. More importantly, routers posses an even smaller amount of *free* memory. In Section 4 we will show a variety of atypical path properties that can quickly exhaust free memory and in Section 5 we will develop an attack which utilizes those properties to disrupt a target router.

3.2 CPU Exhaustion

In general, both hardware and software routers will use all available CPU cycles to handle BGP control traffic. This behavior is desired, as it minimizes the convergence time. But how exactly do heavy CPU loads affect the operation of a router? Chang et al [4] examined advertising full tables and session flapping over time when exploring the effects of memory exhaustion. We instead looked at the effect that heavy load from a separate injecting peer has on common tasks for hardware and software routers.

Table 1 shows the effect that heavy load on a router has on the time required to process 20,000 routes from a full BGP table from RouteViews. When individual updates take longer then normal to process, or other processes or tasks on the router require the majority of CPU cycles, the convergence time increases dramatically. When a router's RIB is not converged, we no longer have guarantees of proper IP traffic forwarding (see Section 2.3). Heavy load also increases the time needed to establish BGP sessions. Session failure is a common occurrence during periods of instability, and this only increases the time it takes for the network to recover. Additionally, it is common to perform diagnostic commands and check router state during periods of instability. Table 1 highlights the delay in establishing new terminal sessions and examining the current BGP sessions when under heavy load.

Table 1 highlights the effect that heavy load on a router has on its ability to perform common tasks. Particularly for Patsy, we see a significant increase in the time to process a set of 20,000 updates under heavy load. We also measured the effect of debugging under heavy load, as it is common for network operators to triage router instability by turning on BGP debugging, and saw that it had a much more significant impact on Patsy than the Quagga router.

A router whose CPU cycles are exhausted by other tasks can thus take much longer to re-establish convergence after session failures. Backlogged updates to a CPU exhausted router can also cause memory exhaustion in its peers, as shown in the following section. Later, in Section 5, we develop attacks that cause excessive CPU usage in a targeted router.

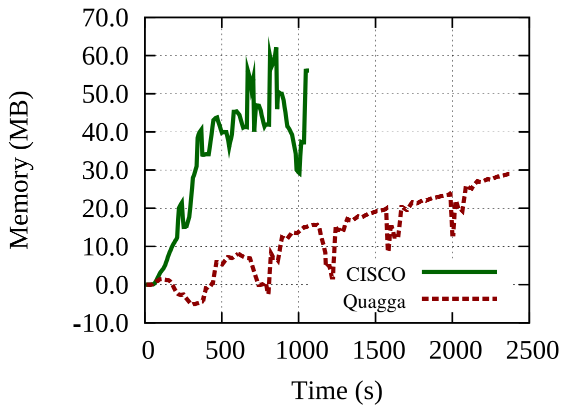


Figure 1: Increase in memory load for a Quagga router and Patsy when advertising to a CPU starved peer compared to a normal peer.

3.3 Update Back Pressure

One of the most interesting behaviors we observed in response to a CPU starved router was the exhaustion of its *peer's* memory. 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 router and Patsy will only buffer a fixed number of BGP messages. When those limits have been reached, the BGP process will stop fetching packets 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*. Figure 1 shows the increase in memory usage over time for a router that are attempting to exchange routing tables with a CPU starved peer versus a peer with sufficient processing power. Patsy experienced an increase in memory consumption of more then 40 MB, with spikes over 60 MB. Quagga saw an increase of 30 MB by the end of our experiment. Depending on the amount of free memory in these routers, something we will touch on in Section 5.3.1, this excess memory consumption may exceed the available memory, triggering session failures.

This increase in memory usage was not the strangest behavior that resulted from update back pressure. On Patsy, we noticed that if the amount of back pressure was large enough, the processes controlling BGP I/O started to fail. Specifically, Patsy ceased interacting correctly with the peers responsible for the back pressure. Patsy 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 Patsy. This I/O issue was limited to BGP, however, as we could initiate a telnet console with Patsy from the Linux box hosting the troubled peer. This problem was only fixed when Patsy was restarted.

	No Load		Heavy Load		With Debugging	
Activity	CISCO (s)	Quagga (s)	CISCO (s)	Quagga (s)	CISCO (s)	Quagga (s)
Establish terminal session	0.198	0.976	0.603	2.053	26.730	1.941
"show ip bgp summary"	0.202	1.795	1.648	2.272	40.233	2.301
Establish new BGP session	0.003	0.047	0.808	0.063	4.046	0.069
Process 20k routes	4.448	14.662	65.402	124.570	725.400	144.052

Table 1: We compared the effects of different loads on completing common tasks for both CISCO and Quagga routers. "Heavy load" consisted of an injector constantly pushing large updates to the target router. Debugging was added on top of the heavy load, and included all BGP debug statements. We saw little or no significant difference between the heavy load and the debugged heavy load for Quagga.

3.4 Interdependence of Unstable States

Back pressure is just one example of an unstable state in one router leading to an unstable state in neighboring routers. Another example of this behavior is memory exhaustion leading to CPU starvation in neighboring routers. Our example starts with a router becoming memory exhausted. As we covered in Section 3.1, this results in the failure of all of the router's BGP sessions. The router's peers are then inundated with BGP updates as the network attempts to re-converge. The dedication of processing power to dealing with this rush of updates in turn forces those routers into a CPU starved state. Of course as we have seen, CPU starvation leads to back pressure, which can lead to memory exhaustion.

Operators can also play a role in causing a new unstable state in response to an existing one. For example, consider the likely course of action an operator would take if his router were receiving a large number of BGP NOTIFICATION messages or exhausting its free memory. The operator would more than likely turn on BGP debugging in an effort to ascertain what was causing the issue. As we saw in Section 3.2, for CISCO routers this would result in an increase of several orders of magnitude in processing demands placed on the router. In other words, the operator would place a router into a CPU starved state in response to a different unstable state.

4 Unexpected Inputs, Unexpected Behaviors

As stated previously, we found it surprisingly easy to force a router into one of the states mentioned in Section 3. We will present several ways we were able to place our test routers into a terminal state. The majority of these methods are the result of taking commonly held assumptions about path attributes and invalidating those assumptions. In the scenarios we present, routers fail to handle these "corner cases" in a reasonable fashion. It is important to realize that the following examples are not an exhaustive list. We did not perform exhaustive checking of BGP implementations—we simply wished to demonstrate the fragility of modern routers.

4.1 BGP Update Properties

In this section, we will start with a simple example of how presenting atypical input to a router can place it in one of the states from Section 3. We will focus on the AS path of a BGP update message. In general, paths are short, with a path length

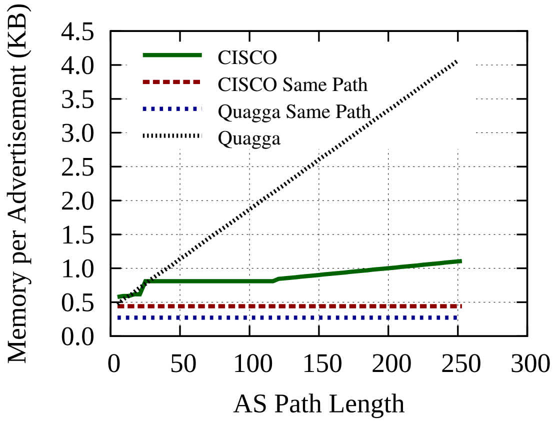
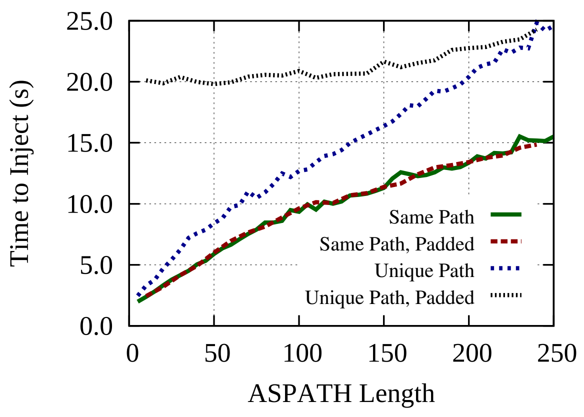


Figure 2: Per update memory usage as a function of path length for Quagga and CISCO routers for both unique and identical sets of paths. Note how Quagga's memory allocation is always a function of the path length, while Patsy allocates fix size blocks for all paths between 24 and 108 ASes in length. Also note how memory requirements are independent of path length when the AS paths are identical. This is because both routers store the repeated path only once.

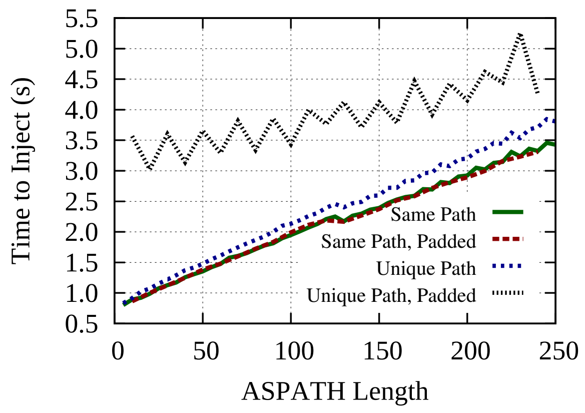
in the range of 3 to 4 being quite common, and path lengths longer than 15 being rare. It is also quite common for multiple routes to share identical AS paths. By presenting a router with a set of updates that do not fit this mold, we will place strain on both a router's memory and its CPU.

Presenting abnormal AS paths is one way to increase memory load. Measured memory usage as a function of path length is shown in Figure 2. These values were established by advertising synthetically generated routes to our test routers and measuring their BGP process's memory consumption. The fact that routers store each unique AS path only once is quickly apparent by examining the results for trials where we advertised routes with identical AS paths. In those situations, memory consumption is independent of path length. For updates with unique AS paths, the figure shows that increasing the path length from the typical length of 3 to 4 ASes to a length of 255 causes can increase of the per prefix memory cost by a factor of 2 in the case of the hardware router and by a factor of 8 for the software router. While these increases are small on an individual scale, the problem is when they are applied in aggregate. For example, the mere 0.5 KB increase in size seen on Patsy, when applied to a full routing table, currently roughly 350,000 routes, translates into an increase of 175 MB in the BGP process, nearly half of the *total* memory of our test router.

We also found that updates with long, distinct AS paths require more time for a router to process than typical updates. To



(a) Quaggga



(b) CISCO

Figure 3: Time to inject 5,000 prefixes as a function of path length for Quagga (Figure 3(a)) and CISCO (Figure 3(b)) routers. Shown here are prefixes with and without unique AS paths. For each, we show the effects of padding the packet with community attributes to match the packet size of a prefix with an AS path of length 255.

measure this, we repeated the previous memory consumption experiments, but timed how long the injector took to advertise all updates. We used community attributes to pad the update messages to a controlled length. This allows us to control for the size of each BGP update message. Figure 3 shows the time-to-inject 5,000 routes for various AS path lengths. We can see that by increasing the size of the AS path from a typical size to a length of 250 and using unique AS paths, we can increase the time-to-inject by a factor of 4.6 in the case of the hardware router and by a factor of 8 for the software router. Padding prefixes that all share the same AS path with community attributes appears to not significantly affect the time to inject compared to not padding. On the other hand, padding updates which have unique AS paths had a marked difference for both routers.

4.2 Algorithmic DoS

In our experimention we came across two different ways to algorithmically DoS a router, one for Quagga and one for CISCO. In each case we utilize valid routes that are designed to take advantage of slow or buggy code. These routes take orders of magnitude more time to process than normal routes. We will present the path constructions here, and develop an attack with them in Section 5.3.2.

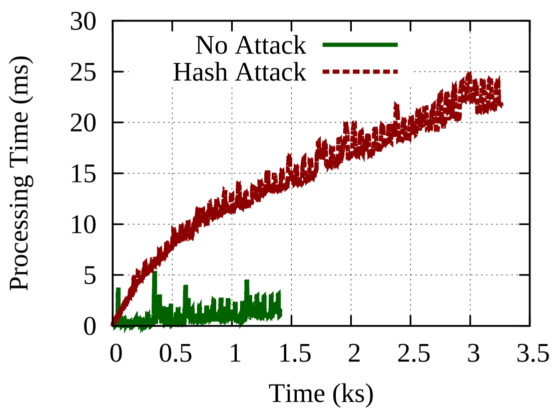


Figure 4: Comparison of the time required to process a set of paths designed to collide into one hash bucket versus a set of random paths.

4.2.1 Hash Attack

Quagga stores AS paths inside a hash map, allowing for rapid access. In the Quagga hash map implementation, the map is of fixed size, in this case 2^{15}, and the hashing function is predictable. This is acceptable so long as the assumption that AS paths will be spread evenly over all of the buckets holds. However, an adversary can violate this assumption. By computing a large number of AS paths that hash to the same value and advertising them to a router, we can increase the amount of time route processing takes. This class of attack, first proposed by Crosby and Wallach [8], exploits the fact that while inserting n elements into a hash map would normally take $O(n)$ time, if each element hashes to the same value it will take $O(n^2)$ time. Plots of the time to process updates with colliding AS paths compared to random AS paths can be seen in Figure 4.

4.2.2 Path Sequences and Sets

AS paths are made up of one or more segments. These segments come in two different flavors: sequences and sets. An AS sequence denotes an ordered list of the exact ASes the path utilizes. On the other hand, an AS set contains a collections of ASes and asserts that the path will utilize one or more of them, without giving further details. The overwhelming majority of paths on the Internet are formed by a single AS path sequence. In nearly all cases, AS paths end with a sequence, but there is nothing that prevents a path from ending with a set. While our Quagga router handled this scenario normally, Patsy experienced issues.

Patsy took several orders of magnitude more time to process routes that ended in an AS set compared to routes that ended normally. In a series of experiments, we injected paths with variable lengths of trailing sets while holding the overall AS path length constant. The results of these runs can be seen in Figure 5. In these runs we injected 8,000 routes with unique

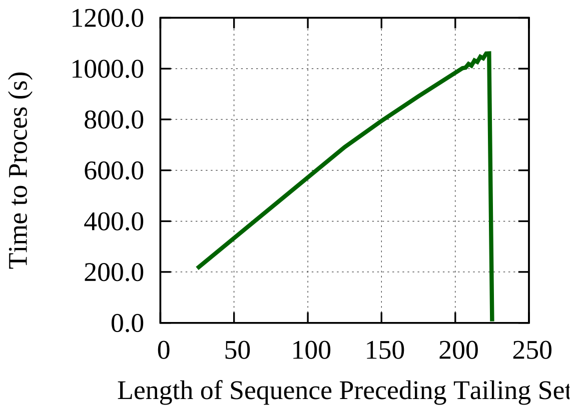


Figure 5: The time required for Patsy to process 8,000 routes with fixed length unique paths that ended in an AS set as a function of the size of the AS sequence preceding the tailing set. The far right hand value is for paths without a tail set.

paths of length 225. The paths with no trailing set took 6 seconds to be processed, while the paths that added an AS set of length 2 took 1,060 seconds. Note that the length of time to process is not dependent on the size of the AS set, but rather on the size of the AS segment that precedes it. It is also interesting that this behavior only exists if the path ends in a set: if the set of paths that took 1,060 seconds is modified to have an AS segment of size 1 after the set, then Patsy again takes 6 seconds.

4.3 BGP Notifications

We also found a large number of BGP messages that are valid according to the RFC, but cause the recipient to send a BGP NOTIFICATION. Given the disruption caused by a BGP NOTIFICATION, as discussed in Section 2.4, this presents a problem. This problem is compounded by the fact that since all of these messages are valid, they can be generated by honest routers.

One set of these valid but problematic messages revolves around AS path length. As we know from Section 4.2.2, BGP breaks paths up into segments. As defined by the RFC, these segments can be at most 255 ASes long. If a router is passed a segment that is longer than 255 ASes that path will result in a BGP NOTIFICATION. In order to send a path longer then 255 ASes, the path needs to be split into two sequences. For example, to send a path of 256 ASes the first segment could have a length of 1 and the second a length of 255. This path is perfectly legal, but both Patsy and Quagga respond as though it is invalid. In the case of Quagga, we can look at the source code to discover that when Quagga encounters a pair of adjacent sequences in an AS path, it merges then without checking their sizes first.

Patsy handles even more path length cases incorrectly compared to Quagga. Unlike Quagga, Patsy responds to *any* path of length 255 or larger with a BGP NOTIFICATION. We noticed that Patsy would often claim routes were longer then 255 ASes when they were actually shorter. We noticed this behavior when experimenting with AS paths that contained several AS segments. We established through test cases that if the sum of the actual AS path length plus half the number of segments in the path was greater then 255, then Patsy would send a BGP NOTIFICATION to the peer that advertised the route, claiming that the route was invalid because its length was greater then 255. For example, a valid route comprised of 25 AS segments of length 10 each would be considered to have a total length longer then 255 AS segments, even though its actual total length is 250. More than likely, Patsy computes AS path length by taking the total byte size of the AS path portion of the update message and simply dividing by two, the expected wire size of an ASN, instead of reading from the packet the actual segment sizes.

This assumption about wire size was also be an issue when Patsy interacted with a Quagga router. Our Quagga routers attempted to write ASNs to the wire as 4 bytes rather than the 2 bytes seen from Patsy. This presented several issues. First, Patsy interpreted AS paths from Quagga routers as longer than they actually were, with all of the consequences we have outlined previously. Second, if updates were very large in size, this expansion from 2 to 4 bytes resulted in update messages going over their maximum size of 4,096 bytes. The real problem here is the asymmetric behavior of Quagga and CISCO routers. A collection of CISCO routers could pass between each other update messages near the 4,096 byte limit without error. When those messages reach a Quagga router, the AS path size will increase as the Quagga router expands the AS size. If that expansion pushes the update messages over the 4,096 byte limit, they will be responded to with a BGP NOTIFICATION.

One last bug we noticed was Patsy's aversion to any update that had an attribute larger then 1,024 bytes. A good example of this was the community attribute field. The community attribute's field is in theory only bounded by the maximum packet size. If Patsy saw a community attribute field larger then 1,024 bytes, a BGP NOTIFICATION would result. Quagga did not do exhibit this behavior.

5 Case Studies: Targeted Attacks

The examples of BGP inputs in Section 4 provide a peek into how a router can end up in one or more of the states we discussed in Section 3. However, one might feel skeptical about these examples. For instance, given that a modern desktop computer has gigabytes of memory, is an increase in memory usage of kilobytes per route going to have a noticeable impact? Also, would these examples not be exceedingly unlikely, only happening in the rare case when one of my direct peers misconfigures his or her router? The problem with these statements is that they fail to take into account an adversary. 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. In our first example attack, an adversary will exhaust the memory of a target router. We will show that the seemingly small impact of an extra kilobyte here or there quickly adds up. Our adversary, depending on the specifics of the router in question, can consume his target's free memory with on the order of 10,000 updates, a small number relative to what routers normally advertise. Additionally, we will lay out how our adversary can launch this attack while minimizing his impact on the memory of other routers in the system. In addition to our first attack, we will discuss how other flaws from Section 4 can also be turned into attacks, degrading the processing power of victim routers and causing BGP sessions between two victims to fail.

5.1 Threat Model

Our threat model focuses on legitimate BGP speakers in transit ASes [1] 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.

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 5.2.

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 7.

5.2 Propagating Malicious Updates

As stated 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 the path 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 shall see in Section 5.3, some of the malicious updates will have exceedingly long 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 `128.101.0.0/17`. 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. By doing 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 7.

The adversary cannot simply send these "best paths" blindly out to all of his peers in the hope that they eventually reach his target. Our adversary is going to take advantage of the fact that honest routers operate in a predictable manner in order to setup "flows" of updates from himself to the victim. When determining whether to propagate a best path to its neighbors, a router takes into account both the customer/provider relationships it has with its neighboring routers and with the route's next hop. A well known set of policies called Valley Free Routing [13] are applied based on these relationships. 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 [27] and applying Valley Free Routing policies to this topology, a model for how the malicious routes will travel through the network can be built. With this model, the adversary can see which peers it should send advertisements to in order to have them propagate to his victim. We shall refer to these paths from the malicious router, through honest routers that forward updates, to the victim, as *attack flows*.

Once attack flows are found, it might be in the attacker's interest to containing 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 infeasible, 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. 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. A toy example of attack flows can be seen in Figure 6.

5.3 Attack Payloads

Now that our adversary has the ability to propagate updates from his malicious router to arbitrary routers in the topology, we will examine what forms of attacks he can launch. We will examine two different types of attacks. In the first, our adversary will exhaust the memory of a target router by forcing it

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

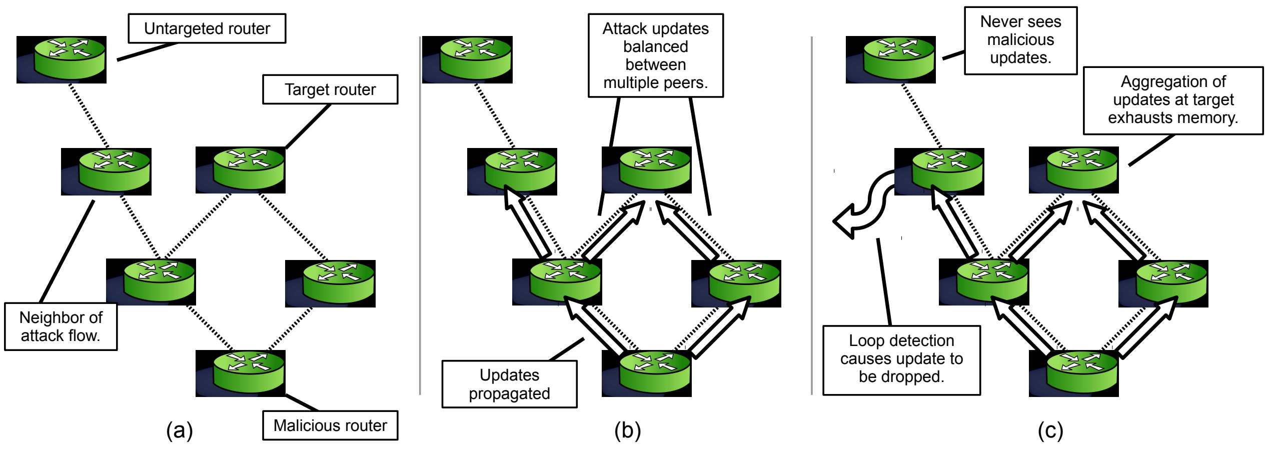

Untargeted router
Target router
Neighbor of attack flow.
Malicious router
(a)
Attack updates balanced between multiple peers.
Updates propagated
(b)
Never sees malicious updates.
Aggregation of updates at target exhausts memory.
Loop detection causes update to be dropped.
(c)

Figure 6: Illustration of how an adversary successfully propagates malicious updates to a target router. In part (a) we define routers in this scenario. In part (b) our attacker sends malicious updates to the target via two flows, malicious updates are also propagated outside of the attack flows. In part (c) the aggregate memory consumption resets the target, and loop detection causes updates to be dropped outside of the attack flows.

to store routes that consume a far larger than expected amount of memory. In the second set of attacks, our adversary will degrade the computational capacity of a target by launching a denial of service attack or by causing BGP sessions to fail.

5.3.1 Attacking Memory

In this attack, our adversary will cause victim routers to run out of memory, forcing the targeted BGP process to reset. The idea of this attack is simple: essentially our adversary "leaks" a relatively small number of updates which are designed to consume inordinate amounts of memory. If a router accepts these BGP updates it will cause the router's memory usage to far exceed expected usage. It is important to note that the malicious routes do not need to be used or deemed best, they need only be considered valid in order to cause the router to store them. These updates will be sent via multiple attack paths to the victim router. The reason for using multiple paths is two-fold. First, if a single malicious route is sent on multiple attack paths, the victim router will be forced to spend memory to store the route once *for each attack path*. That is a result of the routers along the attack path prepending themselves to the path, making each instance of the malicious update unique. The second reason malicious updates are spread over several attack flows is to avoid disrupting the routers on the path to the victim. Essentially, the adversary load balances the memory requirements across multiple "forwarding" routers; the memory load is aggregated only at the victim.

In order to maximize the memory consumed our adversary will focus on three items covered in Section 4.1: AS path length, uniqueness, and community attributes. Our adversary will start with the AS path. Longer paths consume more memory, so he will want the longest possible AS path he can supply. Given that we have demonstrated routers responding with a BGP NOTIFICATION for paths of more than 255 ASes, he will

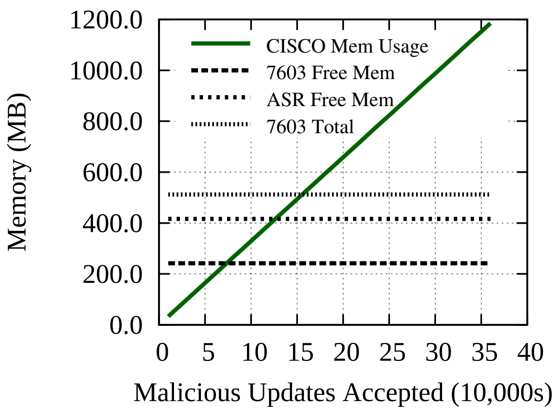


Figure 7: Memory consumption seen in Patsy under our attack. Various memory thresholds are also shown as a reference. The thresholds are, in increasing size, free memory in a fringe router, free memory in a mid-tier ISP router, and total memory in Patsy.

cap path length there. Since each AS along the attack flow will prepend itself to the AS path, the actual path advertised by the attacker is smaller than 255, but will grow to full size in transit. Each malicious AS path used will be unique, forcing the target to store each AS path received. AS paths are built by adding a unique permutation of the attack flow neighbor ASNs and afterwords padding up to the desired size with random ASNs. The remaining space in the update message will be filled with community attributes. The free space in the update messages is enough for 855 community attributes. These are unique as well, which forces memory to be expended for each one. The main question is, will the attack succeed at exhausting the target's memory? The answer to this question is dependent on both the amount of memory the updates can consume and the amount of free memory in the target.

To discover how much memory the attack is capable of consuming, we performed it on our test routers and measured their memory consumption. Our Quagga router consumed more memory per malicious update because it dealt with them in a less efficient manner and was additionally vulnerable to subdivided AS paths. We found Quagga's memory consumption to be roughly 19 KB per update message and Patsy's to be 3.2 KB per update.

Answering the second question—how much free memory a router has—is far more difficult. In the case of CISCO routers we can start by looking at the total amount of memory various classes of CISCO routers have. Some representative examples include Patsy with 512 MB of memory and a CISCO ASR 1004 with 2048 MB. These values give us an extreme upper estimate of the number of updates our adversary would need in order to succeed at his attack. To be more accurate, we could reduce these values by the amount of memory consumed by the BGP process. This would still be an upper estimate, as it does not take into account other processes on the router. This number will vary widely based on the specific router deployment, making it impossible to declare in general how much free memory a router will have available. By combining routing tables from RouteViews and common situations where each class of router is deployed, it is possible to generate some example scenarios. We have provided two of these to give some idea of total memory versus free memory in routers. A router such as Patsy would commonly be deployed in a fringe AS which average between one and three providers. After taking a pair of full routing tables from RouteViews and advertising them to Patsy we found that the router had 242 MB of free memory. It should be noted that this is the *steady state* free memory; during the actual injection Patsy had as little as 13 MB of free memory. Our second scenario looks at an ASR 1004 deployed in a mid-size ISP. We found the routing tables from customers and providers of an ASN with degree 8 in RouteViews. Due to the memory limitations of our test router, we measured the memory consumption for fractions of these tables and extrapolated to estimate the total consumption. We found that the ASR 1004 router would have 416 MB of free memory. Estimation of the free memory in software routers running on commodity boxes is even more difficult due to the varied nature of the hardware used.

At this point we can turn to Figure 7 and examine exactly how many updates our adversary would need to have accepted *at the victim* in order to achieve success. We have drawn in various memory milestones to illustrate the attack's effectiveness. The first milestone to be reached is our estimate for the free memory of Patsy deployed in a fringe AS. As can be seen, it takes roughly 70,000 updates to exhaust the free memory. The next milestone we cross is our estimate of free memory in an ASR 1004 deployed in a mid-sized AS. Here, 125,000 updates are required. The last milestone we will point out is the milestone at 512 MB, where our adversary has consumed *all* of the memory in Patsy. He achieves this with 155,000 updates, less than half of the size of a modern routing table. Current best practices suggest limiting the number of prefixes accepted from a peer; we discuss the impact of this best practice interact in Section 7.

5.3.2 Attacking BGP Sessions

The memory attack is not the only option open to the attacker. The algorithmic denial of service attacks from Section 4.2 present another attack surface for our adversary. There are two slightly different attacks here: one for a Quagga router and one for a CISCO router. Additionally, the attacker can directly attack BGP sessions by inducing honest routers into sending messages that will result in a BGP NOTIFICATION.

As was mentioned in Section 4.2.1, Quagga uses a predictable hashing function and a relatively small hash table size to store AS path information. This allows our adversary to increase route processing time in a target Quagga router by flooding it with paths that hash to the same value. Our attacker will use the flow based propagation method outlined for the memory attack to deliver his updates. As noted in Section 5.3.1, paths grow as they travel from our adversary to the target, to our attacker's advantage. Our adversary will compute hash collisions based on what the paths will look like *when they reach the target*, i.e. after all the ASes along the attack flow prepend their ASN. This means that the malicious updates will only collide at the target, and will not directly impact other routers.

Using the tailing set bug discussed in Section 4.2.2, a similar attack can be launched against CISCO routers. In this attack, our adversary floods updates whose paths end in an AS set. In the same way updates from the memory attack consume the resources of each router carrying them, updates from this attack will slow the processing time of each router along an attack flow. This is a problem for the adversary, as it will slow the rate that these updates reach his target. This problem can be solved by spreading the malicious updates along multiple attack flows and merging those flows at the target. As in the memory attack, this pits the aggregate resources of each hop against the resources of the lone target router.

In the last attack, the attacker directly pushes the network out of convergence by causing BGP sessions to fail. Our attacker will cause session failures by triggering BGP NOTIFICATION messages. While our attacker could trivially send BGP NOTIFICATION messages to his peers, he could launch a more powerful attack by convincing honest routers to send them to their peers. We discussed a number of messages in Section 4.3 that result in a BGP NOTIFICATION; the trick is getting the error to occur near the target rather than near the adversary. One option is again to use growing path length. For example, the adversary picks a router whose sessions he wishes to kill, which sits at a distance D from his router. He then propagates an update to that router which starts with AS path length equal to $256 - D$. Each hop will add another ASN to the path, resulting in a path length of 255 by the time the update reaches the target. When the targeted router advertises the path it will add itself to the front, resulting in a 256 AS long path, which will result in a BGP NOTIFICATION from peers that receive it. Another method takes advantage of the asymmetric behavior between different makes of router. If our adversary targets a session between a CISCO router and Quagga router, he can take advantage of Quagga's desire to send ASNs as 4 bytes and CISCO's shortcut method of computing path length. Our adversary propagates an update with a path length of 128 or greater to the Quagga router. The Quagga router will in turn pack the path as 4 byte values. The CISCO router, upon receipt, will compute the path length based on the field size, and incorrectly establish a length of 256 or more, resulting in a BGP NOTIFICATION.

6 Related Work

As mentioned in Section 4, Chang, Govindan, and Heidemann [4] investigated router responses to memory exhaustion. Their work in 2002 first highlighted the fact that routers reset under large memory loads. It did not, however, examine how to cause large memory loads. Previous work by Bu, Gao, and Towsley [3] has examined how BGP consumes memory under normal conditions. They studied routing tables to determine how multi-homing, load balancing, IP fragmentation, and lack of aggregation affect routing table size. Unlike our paper, which focuses on abnormal inputs, their paper focused on route properties seen in normal operation.

Previous research by Wu et al [33] focused on benchmarking CPU performance in various routers. Their work focused on the performance differences of various types of CPUs, not on various types of BGP inputs. Other papers, for example Agarwal et al's [1], focused on the CPU performance of deployed routers using BGP logs. This paper, like Bu's work in memory usage, focuses on CPU activity under normal operating conditions rather than atypical conditions.

There is a large body of existing work on the impact of router stability on the data plane. For examples, see [12, 30, 31]. These works do an excellent job of highlighting why the data plane fails to function correctly as a result of router instability. Our paper is orthogonal to these, as we are concerned with the causes of instability rather than its results.

Recent work by Yin, Caesar, and Zhou [34] focused on examining bugs in implementations of routing protocols. Their research focused on categorizing bugs by their locations in the code and root cause. Their paper is narrower than ours as it focuses exclusively on software bugs while we are interested in both bugs and "proper" handling of unexpected input. Additionally, their paper provides limited insight into the consequences of these bugs.

Other papers have focused on different ways instability in collections of routers can be generated. Deng et al [11] present a straightforward method in which a malicious set of routers create instability by simply building and tearing down their BGP sessions repeatedly. Bellovin and Gansner [2] present a method of instability generation based around cutting links between routers, forcing them to redo route discovery. In a similar vein, Schuchard et al [28] generate instability by using botnet traffic to generate session failures as a result of BGP timers. Our work differs from all of these in that we generate instability in target routers by exposing them to unexpected BGP inputs.

7 Examining Best Practices

In this section, we examine how various "best practices" interact with the proposed attacks of Section 5. We focus mainly on four different practices: prefix filters, prefix aggregation, prefix 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 of these findings can be seen in Table 2 We will also briefly touch on current proposals that seek to secure BGP by use of cryptographic methods, namely BGPSEC.

7.1 Prefix Filters

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 5.2, 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 getting filtered? To answer this, we examined what length of prefixes we can *actively observe* being forwarded by various Autonomous Systems based on RouteViews dumps. What we found was straightforward: 88.5% of transit ASes forwarded prefixes that were /24 or shorter, while 6.8% forwarded prefixes longer than this. Thus, in the majority of cases, our adversary can advertise prefixes of length /24 or shorter successfully.

This leads us to our next question: Given that we can advertise no more specific a prefix then a /24, can our adversary find enough un-advertised prefixes to complete his attacks? This can be answered with a quick back of the envelope calculation. There are approximately $1.6x10^7$ prefix blocks of length /24, and of those 98% correspond to routable IPs (the other 2% are un-routable bogons [9]). The current size of the full Internet routing table is roughly $4x10^5$ prefixes [15], meaning that if all IP blocks advertised were /24s (which they are not obviously, the majority of prefixes seen are for larger blocks of IP addresses), then there would still be over $1.5x10^7$ /24s un-advertised. Clearly this means that our adversary can find a sufficient number of un-advertised /24 blocks to utilize for his attack.

7.2 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 manu-

Best Practice	Why It Does Not Help	Experimental Evidence
Prefix Filters	Limits still give the attacker access to millions of prefixes	/24s advertise by 88.5% of transit ASes
Prefix Aggregation	Not done to routes from transit ASes	Observation of hole punches and non-aggregated IP blocks in RouteViews
Prefix 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
AS Path Limits	Weakened by generous path limits and how Routers allocate memory	Patsy allocates memory in fixed size blocks (Fig 2)

Table 2: Summary of best practices we considered, why they fail to stop the attacks from Section 5, and what experimental evidence backs each conclusion.

ally 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 manner he builds prefixes makes them appear identical to hole punches.

More over, 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 `123.101.1.0/24` and `123.101.0.0/24`, 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.

7.3 Prefix Limits

A different best practice that directly impacts our adversary is the limiting of the number of prefixes one router will accept from another. While there have been historical incidents that call into question if a majority of ASes actually do this [24], let us assume the best case: that all ASes follow this practice. While some of the attacks covered in Section 5 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 to not 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 setup 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.

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 [27] 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 margins as large as 25% on prefix limits, so as to not accidently exceed them [10]. This means our adversary can allow normal operation to continue, while using that safety margin to advertise his malicious routes.

7.4 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 [35]. Even if routers set a small AS path length limit (the current recommendation is 100 or less), we see in Figure 2 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 0.811 KB of memory. With a full path of length 253, we only see a marginal improvement to 1.108 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, as seen in Figure 7. Here we achieve 3.292 KB per update accepted. If our attacker instead was restricted to using paths of length 22 and added community attributes as before, he could still achieve a per update memory cost of 2.995 KB, which would not significantly reduce the effectiveness of the attack.

7.5 BGPSEC

Various projects have worked on securing BGP advertisements, most notably BGPSEC [22]. Sadly, BGPSEC fails to address many of the issues presented in our paper. This is a consequence of the fact that all of the messages we cite in Section 4 are *valid* messages. While BGPSEC might complicate the creation of these corner case messages, it does not prevent them. Moreover, BGPSEC places additional memory and processing demands on routers. BGPSEC does make many of the techniques discussed in Section 5.2 more difficult to implement. However, we note other research, notably Goldberg et al. [14] demonstrate how an adversary can manipulate honest BGPSEC speakers into forwarding messages based on customer/provider preferences.

8 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 that there are a collection of states that cause unstable behavior in BGP speakers. We have provided simple examples to highlight how routers can end up in these states. Lastly, we expanded these examples into full scale attacks which function even when best practices are employed.

Many of the examples we covered center around flaws in BGP implementations. We are in the process of submitting bug reports to CISCO and a patch to Quagga in order to fix the particular bugs discussed in this paper; however, these efforts are an incomplete solution. The list of bugs in Section 4 was not obtained through exhaustive search—there are likely to be other bugs lurking in the shadows. These examples highlight the fact that it is not difficult for a modern router to become unstable. More critically, we have shown that these examples are not limited to "Internet accidents." Rather, they can be exploited by an adversary to intentionally cause instability for his own benefit. This last part is especially troubling given that many protocols which claim to be secure also assume router stability. Because of this assumption, we recommend that BGP implementations are tested for protocol correctness in addition to software correctness.

References

[1] S. Agarwal, C. Chuah, S. Bhattacharyya, and C. Diot. Impact of BGP dynamics on router CPU utilization. *Passive and Active Network Measurement*, pages 278–288, 2004.

[2] S. Bellovin and E. Gansner. Using link cuts to attack Internet routing. *AT&T Labs Research Technical Report*, 2003.

[3] T. Bu, L. Gao, and D. Towsley. On characterizing BGP routing table growth. *Comput. Netw.*, 45:45–54, May 2004.

[4] 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.

[5] Cisco Systems. CISCO 7603 series router. `http://www.cisco.com/en/US/products/hw/routers/ps368/ps370/index.html`.

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

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

[8] S. A. Crosby and D. S. Wallach. Denial of service via algorithmic complexity attacks. In *Proceedings of the 12th conference on USENIX Security Symposium - Volume 12*, pages 3–3, Berkeley, CA, USA, 2003. USENIX Association.

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

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

[11] W. Deng, P. Zhu, X. Lu, and B. Plattner. On evaluating BGP routing stress attack. *Journal of Communications*, 5(1):13–22, 2010.

[12] 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.

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

[14] 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.

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

[16] K. Ishiguro et al. Quagga Routing Suite. `http://quagga.net/`.

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

[18] 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.

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

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

[21] Network Working Group. RFC4724 — Graceful Restart Mechanism for BGP. http://www.rfc-editor.org/rfc/rfc4724.txt, January 2007.

[22] Network Working Group. BGPSEC Protocol Specification. https://tools.ietf.org/html/draft-ietf-sidr-bgpsec-protocol-01, October 2011.

[23] 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.

[24] 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.

[25] 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.

[26] QEMU open source processor emulator. http://wiki.qemu.org/Main_Page.

[27] RouteViews. RouteViews Dataset. http://www.routeviews.org/.

[28] M. Schuchard, E. Vasserman, A. Mohaisen, D. Foo Kune, N. Hopper, and Y. Kim. Losing control of the Internet: using the data plane to attack the control plane. In *Proceedings of the 18th Annual Network & Distributed System Security Symposium*. ISOC, 2011.

[29] C. Systems. CISCO Express Forwarding Overview. http://www.cisco.com/en/US/docs/ios/12_2/switch/configuration/guide/xcfcef.html.

[30] 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.

[31] F. Wang, J. Qiu, L. Gao, and J. Wang. On understanding transient interdomain routing failures. *IEEE/ACM Trans. Netw.*, 17:740–751, June 2009.

[32] 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.

[33] Q. Wu, Y. Liao, T. Wolf, and L. Gao. Benchmarking BGP routers. In *Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization*, IISWC '07, pages 79–88, Washington, DC, USA, 2007. IEEE Computer Society.

[34] Z. Yin, M. Caesar, and Y. Zhou. Towards understanding bugs in open source router software. *ACM SIGCOMM Computer Communication Review*, 40(3):34–40, 2010.

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

Appendices

A Experimental Equipment

We used the following equipment for our experiments:

- A CISCO 7603 series hardware router [5] running IOS 12.2(18), affectionately named "Patsy", running BGPv4.
- Two Linux machines directly connected to Patsy via Gigabit Ethernet connections.
- Virtual machines running the Quagga software routing suite [16], version 0.99.20, also running BGPv4. The virtual machines ran under the QEMU hypervisor [26] running on our lab's servers, each given a single 2.67GHz CPU core, and networked using a shared multicast bus.
- A custom BGP injector, called "Morkai", written to maximize update throughput while minimizing message loss. Writing a custom injector also allowed us to perform experiments with atypical BGP updates and sessions. We have made Morkai publicly available.
- Real-world BGP routing tables from the RouteViews [27] project.