The ISP Column
A monthly column on things Internet

October 2020
Geoff Huston

Securing Routing Q&As

Over the past few months I've had the opportunity at various network operator meetings to talk about BGP routing security and also highlight a measurement page we've set up that measures the extent to which Route Origin Validation (RoV) is actually "protecting" users (https://stats.labs.apnic.net/rpki). By this I mean we're measuring the extent to which users are prevented from having their traffic misdirected along what we can call "bad paths" in the inter-domain routing environment by virtue of the network operator dropping routes that are classified as "invalid". As usual, these presentations include an opportunity for questions from the audience. As a presenter I've found this question and answer segment in the presentation the part that is the most fun. It covers topics that I've not explained well, things I've missed, things I've got wrong, and things I hadn't thought about at all right up to the point when the question was asked! Here are a small collection of such questions and my efforts at trying to provide an answer.

Why doesn't your measurement tool ever show 100%?

There are a number of network service providers that clearly perform ingress filtering of route objects where the associated ROA shows that the route object is invalid. The network AS 7018 (AT&T Internet) is a good example, as is AS1221 (Telstra). The numbers keep growing as deployment of this technology spreads. The question is: Why doesn't the measurement tool report 100% for these networks? Why is there "leakage" of users? This means that some users are reaching a site despite the invalid state of the only route object to that site. If the network is dropping invalid routes, then how can this happen?

A number of possible explanations come to mind. It could be that there are a small proportion of end users that use split horizon VPN tools. While the measurement system reports the user as having an IP address that is originated by a particular network, there might be a VPN tunnel that relocates this user to a different point in the inter-AS topology. However, it's not a very satisfactory explanation.

There is a different explanation as to why 100% is a largely unobtainable number in this RPKI measurement system. The reason lies in the measurement methodology itself. The issue here lies in the way a control element is implemented, and in the nature of the experiment itself.

We could've designed the experiment to use two beacons. One object would be located behind an IP address that has a valid ROA that never changes. A second object would be located behind an IP address that has a ROA that casts the route object as "invalid". Again, this would never change. What we would be counting is the set of users that can retrieve the first object (valid route object) and not the second (invalid route object).

However, there are a couple of issues with this approach. Firstly, we want to minimise the number of web fetches performed by each user. So, one web fetch is better than two or more in this measurement framework. Secondly, this approach would be incapable of looking at the dynamic capabilities of the overall system. How long does it take for a state change to propagate across the entire routing system?

This thinking was behind our decision to build a measurement system that used a single beacon that undertook a regular state change. One half of the time the route object is valid, and the other half it's invalid (We admit to no Schrodinger's cat-like uncertainty here - in our case that cat is alive or dead, and there is no in between!). We designed a single measurement that changed state: one half of the time it was a 'control' state where the route was RPKI-valid, and the other half of the time was RPKI invalid.

This might lead one to believe that a ROV-filtering network would be able to track this state changing, installing a route when the RPKI ROA state indicated a valid route object, and withdrawing the route when the RPKI ROA state indicated an invalid route object. This is only partially true, in that there is an appreciable time lag between a state change in the RPKI repository that publishes the new ROA and the state of the local BGP FIB that processes packet forwarding.

Once the RPKI repository has published a new Certificate Revocation List that revokes the old End Entity certificate and publishes a new End Entity certificate and a new ROA signed by this new certificate, then a number of processes come into play.

The first is the delay between the RPKI machinery and the publication point. For hosted solutions, this may involve some scheduling and the delays may be appreciable. In our case all this occurs on a single system and there is no scheduling delay. This is still not instantaneous, but an inspection of the publication point logs indicates that the updated ROA, CRL and manifest files are generated and published all within 1 second, so the delays at the RPKI publication point, in this case, are small.

Now the remote RPKI clients need to sweep across the repository and discover that there has been a state change. This is the major element of delay. How often should the RPKI client system sweep across all the RPKI repository publication points to see if anything has changed? Don't forget that there is no update signalling going on in RPKI publication (yes, that's another question as to why not, and the answer is that publishers don't know who the clients will be in advance. This is not a closed system.) Each client needs to determine its "sweep frequency". Some client software uses a 2-minute sweep interval, some use 10 minutes, some use 1 hour. But it is a little more involved than this. The sweep process may start every 2 minutes, but it takes a finite amount of time to complete. Some RPKI publication points are slow to access. This means that a "sweep" through all RPKI repository publication points might take a few seconds, or a few minutes, or even many minutes. How frequently your RPKI client system sweeps through the distributed RPKI repository system to identify changes determines the lag in the system. Should we perform this sweep in parallel to prevent head of queue blocking causing in the sweep intervals? Maybe, but there is a limit to the number of parallel processes that any system can support, and no matter what number you pick there is still the situation that a slow publication point will hold up a client.

The result? It takes some time for your system to recognise that my system has revoked the previously valid ROA and issued a new ROA that invalidates the prefix announcement (and vice versa). On average it takes 30 minutes for everyone to see the valid to invalid transition and five minutes on the invalid to valid transition. In every 7-day window there are 6 "shoulder" periods where there is a time lag between the RPKI repository state change and your routing state. The errant users who are stopping any network from getting to 100% lurk in these shoulder periods. Here's one snapshot of one such transition period for one network (Figure 1). In this case the routing transition took effect somewhere between 410 and 427 seconds after the change in the source RPKI repository. Those 51 sample points out of the one-hour sample count of 319 individual experiments are essentially the "errors" on the fringe of this measurement.

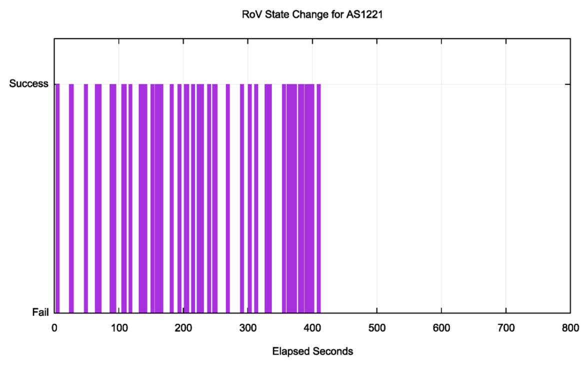

Figure 1 – RoV State Change Lag

It is likely that the time lags between the repository state and a network's routing state that is the factor that presents the network recording a 100% measurement, but it's not really the network's fault. The RPKI RFC documents are long and detailed (I know, as I authored a lot of them!) but they do not talk about timers and the expected responsiveness of the system. So different implementors made different choices. Some setups are slow to react, others are faster. A clear standard would've been helpful here. But we would still have this problem.

The underlying issue is the parallel RPKI credential "just in case" distribution system.

How fast, or how responsive, should the RPKI distribution system be in theory? "As fast as BGP propagation" is the theoretical best answer. The compromises and design trade-offs that were used to construct this system all imposed additional time penalties and made the system less responsive to changes.

Can we use this ROV setup to perform saturation DDOS defence?

This question is a bit like asking: Can I perform delicate microsurgery with a mallet and a chisel? You can give it a try but it's going to be a really bad idea!

The objective of a saturation DDOS defence is to push back on the incoming traffic that is overwhelming the target. One approach is to use the routing system and instruct other networks to discard all traffic destined to the victim address. The current convention in the operational community is to use BGP communities attached to a specific route to signal that you want all traffic destined to this prefix to be dropped (Remotely Triggered Black Hole, or RTBH, RFC5635).

Given that the RTBH signal is saying "drop traffic to this prefix" it's probably a good idea to make the prefix as specific as you can, yet still allow the prefix to be propagated through BGP. The idea of using a more specific route ensures that any destinations that share the same aggregate route are not affected. The signal propagates through the BGP space as fast as BGP, which means that the signal is usually effective in the order of 2 or three minutes after the RTBH route is first announced.

Could you achieve the same with a deliberately invalid ROA? What if you advertised a prefix with a ROA of AS0, for example?

There are a few issues here:

- The network operator can't generate the blocking ROA. Only the prefix holder can sign a ROA. So unless the prefix holder and the network operator are one and the same the network operator has to wait for the prefix holder to mint this AS0 ROA.

- The effect of the ROA will take time to propagate through the BGP. It appears that it takes some 30 minutes for a blocking ROA to propagate to the point that packets are being discarded.

- The effect of the AS0 ROA is not to change the BGP next hop attribute of a route hop, but to withdraw the route completely from the local BGP FIB. This means that any covering aggregate route, or any default route, will take over and the traffic will still be passed through, assuming that aggregates or defaults exist. Now the network provider can control whether or not aggregate routes are being originated, but default routes are a local configuration, and the victim has no control over such routes.

Taken together the result is that it's slow to take effect, requires additional orchestration and my not have any effect on the attack traffic in any case.

Is Rsync that Bad?

The use of X.509 certificates as the vehicle to convey the public keys used in the RPKI led inevitably to an outcome of distributed certificate publication points. Each client, or relying party, of the RPKI now has a problem, in so far as the client has to regularly comb all these publication points and gather up the signed products and load them into a locally maintained cache. This can be done in a number of ways, and about the most inefficient to take that previous statement literally!

Yes, it's an option to start each pass through the distributed RPKI publication points, starting with the trust anchors and follow the publication point links in each validated CA certificate and collect a copy of everything that you find there. It's also an incredibly inefficient option. The relying party client ends up grabbing a precise copy of what they got from the last pass through the RPKI system. What the client really wants to know is what's new, what's changed and what's been removed since the last visit. That way they can simply fetch the differences and move on.

There was one application that performs this "what's changed" function very conveniently and that's the "rsync" application. Rsync can simply take the URL of the remote publication point, the URL of a local cache and it will update the local cache to mirror the remote state. It will try and do so efficiently as well.

This seems like an ideal fit to the problem.

With just one problem.

Rsync is not a good idea to use in the big bad environment of the Internet.

As APNIC's George Michaelson and Byron Ellacot told us back in 2014 (https://www.ietf.org/proceedings/89/slides/slides-89-sidr-6.pdf) rsync is just not up to the task. It can be readily attacked and its synchronisation algorithm can be mislead. As the final slide of their presentation to the SIDR WG said: "It seems a little strange to build routing security on top of a protocol which we have demonstrated is inefficient, insecure and dangerous to run as server or client".

Yes, rsync is that bad. Don't use it out there in the Internet.

Why does the system rely on Pull? What's wrong with Push?

Instead of Rsync, what should we use?

Today's answer is the RPKI Repository Delta Protocol (RRDP), described in RFC 8182. Its foundation is a more conventional HTTPS transport subsystem, and it certainly appears, so far, to be a far better match to our requirements than rsync.

The channel is secure, the delta files direct the synchronisation process, and neither the server nor the client are exposed any more than conventional HTTPS exports the end points of a transaction. Better, yes. But is it really fit for purpose? Probably not!

It's not the RRDP protocol, but the data model that I believe is broken. The basic requirement here is a highly reliable information flooding requirement. All the data sources need to keep all the data clients up to date. When a source changes their data all the clients need to be updated with the new data set. What RRDP represents, and rsync too for that matter, are "poll and pull" models. It's up to the client to constantly check with every data source to see if the source has changed anything since the last check (poll) and then retrieve these changed items (pull). The burden is placed on the client. All the source has to do is simply make the data available.

But there are a few issues here with this approach. How often should clients check with each source? If sources can alter their data at any time, then clients may fall behind the sources and make decisions based on old data. Clients are motivated to poll sources at a high frequency in order to stay up to date. But this intensive polling places a large load on servers, and as the number of RPKI publication points increases the load imposed on the clients increases. What we have today is both ends of the Goldilocks problem but no real concept of what might be "just right". The 2-minute polling intervals seem to be crazy fast and exacerbate scaling pressures. On the other hand, 1-hour polling intervals seem to be geologically slow for a routing system. What's the "right" answer?

A similar issue was observed in the DNS between primary and secondary servers, and one response was the adoption of the NOTIFY and IXFR mechanisms. These allowed the primary source to notify the secondary clients of a change and then allow the clients to retrieve only the changes. Would this work in the RPKI space? Probably not. The issue is that there are an unknown number of clients, so the server has no a priori knowledge of whom to notify. Perhaps there is also a deeper problem here in that this framework of sources and clients makes no use of intermediaries. If we want to scale up the system, then perhaps we need to consider a different distribution structure.

What we need is a reliable distributed data flooding model that can propagate routing-related meta-data across the realm of BGP speakers and have the same dynamic properties in terms of propagation times as BGP itself. What's the best protocol where we have experience and knowledge to achieve this outcome? BGP itself of course! The same mechanisms that propagate route object updates and withdrawals across the inter-domain space is equally capable of propagating any other data payload. All that's needed is to extend BGP to allow the support of other data objects. Can we do this? Of course, we can! The BGP session Open messages conversation exchange a set of capabilities, and if both parties support a particular capability then the peers can exchange information based on this capability. This is not a new attribute of a route object, but an entirely new object.

Sources originate changes to their information base and pass them into BGP. Each BGP speaker integrates this new information into their local information model and then send updates to their BGP peers based on the changes to the local information model.

Why haven't we gone down this path? Why are we configuring RRDP in a way to try and replicate the capabilities and performance of BGP in terms of reliable and efficient information flooding? If BGP could achieve all this then why aren't we using it?

I suspect a bit of IETF history is relevant here. The original brief to the SIDR Working Group included the admonition: "Don't change the BGP protocol!" So, when the working group needed a reliable flooding capability that was equivalent in so many ways to BGP, altering BGP to add a new protocol object was just not an available option for the Working Group.

In answer to the original question, it appears that there is nothing wrong with push. BGP is a push protocol and it seems to be doing just fine! The RPKI system adopted pull largely because of a constrained set of options available to the design group. Personally, I see this as an unfortunate outcome!

Why don't we attach credentials to BGP updates?

TLS is an interesting protocol in many ways. TLS uses X.509 certificate validation as a means of authenticating that the party at the other end of a connection is the party that they are purporting to be. However, the TLS protocol can do this without distributed repository publication points, without pull or push to maintain local caches or any of the other mechanism used in the RPKI system for BGP security. Yet both TLS and Route Origin Validation use X.509 certificate systems to validate digital signatures. In the case of TLS, the difference is that within the initial exchange of information from the server to the client, the server includes the entire set of certificates that allow the client to construct a validation chain form the public key to a trust anchor. All the client needs is the trust anchor and it's up to the server to demonstrate a chain of transitive trust from this trust anchor to the entity that is presenting a public key to the client as a proof of authenticity.

Could we undertake an analogous operation in RPKI?

The basic answer is yes. If all we are looking for is validation of the authority granted to an AS to originate a route for this prefix, then it's possible to affix the certificate chain to the route object and just propagate the digital credentials along with the object. Affixing the entire validation path with every update can lead to significant levels of duplication in the BGP exchange, but we can leverage the observation that BGP itself uses TCP and is a reliable protocol. Once a certificate is passed to a peer within the context of a BGP session it can be assumed that the peer possesses the certificate and further use of this certificate need not reproduce the entire certificate, but simply refer to the previously sent item. The result is similar to the use of the 4-byte AS transition in BGP, where information was passed through the 2-byte BGP world in the form of an opaque community attribute.

So why didn't we do this?

The same IETF history referred to above is relevant here. This would change the BGP protocol, and the admonition to the SIDR Working Group was to change nothing. Taking out the entire side-channel of attempting to pass the certificate credentials in advance of their use in a BGP updates takes out a rather significant source of operational complexity. Like TLS, if a BGP update message contained sufficient metadata to allow the prefix to be validated the entire system would be far simpler to operate. Again, I see the current design as an unfortunate outcome.

Why don't we outsource validation?

DNSSEC has been deployed in a rather odd fashion. End users, or stub resolvers don't generally validate DNS responses. Instead, they rely on recursive resolvers to perform DNSSEC validation. The recursive resolver passes the response back to the stub resolver in an unencrypted DNS response with a single bit set to indicate that the recursive resolver has performed DNSSEC due diligence on the answer. And the stub resolver believes it! If we believe that this rather insubstantial veneer of security is good enough for the DNS, then surely its good enough for BGP!

We've taken steps in this direction with RFC 8097. We just tag all BGP updates with the extended community "validation state" with a value of 0 and everything is valid! Right?

Ok that was a low shot, and yes, this particular RFC is clear in saying that this is a non-transitive attribute that should not leak outside of an AS. But what about an exchange point operator? As part of their service as a trusted broker of routing then what's the problem in not only validating the BGP routes being passed across the exchange, but marking these routes with a community attribute to show to exchange peers that the exchange has validated this route?

I have often heard the observations that "outsourced security" is a contradiction in terms and "transitive trust" is equally a misnomer! For the same reason that outsourced DNSSEC validation is a somewhat inappropriate leap of faith, my view is that outsourced routing validation is a leap of faith. But perhaps there is more to it than this and we all are sucked into outsourced validation in RPKI, like it or not.

In DNSSEC a client can say in a query: "Please don't do validation on my behalf, and just tell me what you know, irrespective of whether the data passed or failed your validation". The EDNS(0) Checking Disabled Flag is an interesting breakout of the inferred model of outsourced validation.

What about BGP? Can a BGP speaker say to its neighbour "Please send me what you have, RoV valid or not", allowing the local BGP speaker to perform its local validation without reference to the validation outcomes of its peer? Well, no, it can't do that. A router that performs RPKI validation these days typically drops invalid routes. This "invalid drop" mechanism essentially imposes an outsourced validation model on its peers by not even telling them of routes that failed validation. If it told them of these invalid routes, then what is it supposed to do with the consequent packet flow?

Shouldn't we view this with the same level of scepticism that we use for outsourced DNSSEC validation? What's the difference?

The answer lies in the very nature of the routing process itself. BGP speakers tell their neighbours a promise: "If you pass me a packet destined to this address I promise to pass in onward". If the BGP speaker's RPKI validation process results in a dropped route then the BGP speaker cannot convey such a promise to its BGP neighbours, as the dropped route says that it will not pass any such packets onward. In so many ways routing is a cooperative undertaking that relies on trust in what neighbours tell each other in the first place.

I would say that by its very nature the action of dropping routes that are invalid each BGP speaker is not only making a local decision, but is also making a decision on behalf of its BGP neighbours. The Drop Invalid local decision in in fact an implementation of outsourced security already.

Why should I use a Hosted RPKI service?

As a part of a program to encourage the deployment of RPKI a number of folk, including the Regional Internet Registries offer a so-called "hosted service" for RPKI,. where they operate the RPKI publication point for the client, and perform all the certificate management services on their behalf, offering the client a functional interface that hides the inner details of the RPKI certificate system.

It seemed like a good idea at the time.

But this is another case of adding an external point of vulnerability to the system. While all of our experience points to the futility of expecting comprehensive perfection, our innate optimism often triumphs over the disappointments of past experiences. Pushing this function to another party does not necessarily mean that the function will be performed to a higher level of performance, and at the same time you lose control over the service itself. If this is critical to your online service then perhaps this is not what you want to do.

At the same time the tools available to host your own repository are improving, and current hosting tools offer a similar level of functional interface to hosted solutions. In many ways the two approaches are now similar in terms of operational complexity to the network operator. Does that mean we should all host our own RPKI publication points so as to assume greater control over our own security environment?

Well, maybe not.

The RPKI publication model is very rudimentary, and it strikes me as having a lot in common with the web publication models before content distribution networks just took over the web world. Having distinct RPKI publication points for each network means that in a few years we would be looking at some 100,000 distinct RPKI publication points. And having achieved the goal of universal general adoption (!) then each of these 100,000 networks would also be sweeping across all of these publication points every two minutes. Right? So that’s 100,000 sweepers passing across 100,000 publication points every 120 seconds. Yes, computers and fast and networks are big, but that seems a little too demanding. Perhaps we should all use hosted services, so these 100,000 clients need to sweep across 10 or so publication points every 120 seconds. Surely that’s an easier target? Well yes, but the implications of having one of these hosting services drop out would have a dramatic impact on routing. After all, in a universal adoption model the absence of credentials is as bad as fake credentials as everything that is good is signed and everything else is bad. That’s what universal adoption really means. So, our tolerance for operational mishaps in the routing security infrastructure now drops to zero, and we are now faced with the challenge of making a large system not just operationally robust, but operationally perfect!

But let's not forget that we are dealing with signed objects, and as long as the object is validated though your trust anchor then you know its authentic and cannot be repudiated. Who cares whether you get the object from an elegant (and expensive) certificate boutique in the fashionable part of town or pick it out of the gutter? Authenticity is not about where you found it, but whether you are willing to trust its contents. If it validates its good! So why don't we get over all this hosted or distributed publication point issue and just toss it all into the CDN world? For the CDN world these numbers of the order of millions of objects with millions of clients is not only comfortably within current CDN capabilities, but so comfortably within their capability parameters that the incremental load is invisible to them. As has been observed in recent years in the conversation on the death of transit, who needs an Internet at all when you have CDNs?

What’s Egress Filtering and why should I do it?

There is this theory that in this yet-to-happen nirvana when every network does ROV dropping that when two networks become BGP peers then that peering session needs to have only one point of origin validation filtering. Applying a filter on both export and import seems like just having double the amount of fun any network should have.

What about the imperfect world of partial adoption?

Earlier this year one largish network enabled RoV filtering over incoming BGP updates, but then was observed leaking routes with its own AS as the origin AS. Some of these leaked prefixes had ROAs, and route origin validation would've classified these leaked routes as invalid. RFC8893 on "Origin Validation for BGP Export" specifies that implementations must support egress filtering in additional to ingress filtering. It uses a normative MUST to make the point, but fails to provide any further discussion as to whether network operators should enable this function.

If we come back to the leaked route incidents, it's true that egress filtering would've prevented this leak. Well not quite. It would've stopped leaking those prefixes where there was an extant ROA. Other prefixes, and in a world of partial adoption there are probably many other prefixes, would still have leaked. In general, it's true that egress filtering stops a network from "telling evil" in as much as ingress filtering stops a network from "hearing evil" and the combination of ingress and egress filtering is a pragmatic measure when the network is unaware of the origin validation capabilities of a BGP peer. But it's a senseless duplication of effort when both networks who directly peering are performing origin validation.

This occurs because RPKI validation is performed as an overlay to BGP rather than as an integral part of the protocol. Neither BGP speaker is aware when the other is applying an RoV filter to the updates they are announcing or hearing. This structural separation of RPKI from BGP leads to the duplication of effort.

But perhaps there is more to this than simply ingress and egress filtering. It was evident that in leaking these routes the network was announcing more than it had intended. But other networks had no way of knowing that. Now in the magical nirvana of universal adoption of route origination all these forms of route leak would be evident to all because of the origination mismatch. But that nirvana may be some time off. IPv6 is taking more than two decades. DNSSEC is looking pretty similar. Gone are the days when we could contemplate an uplift of the entire routing infrastructure in a couple of months.

So pragmatically we should ask ourselves a slightly different question: "In a world of partial adoption how can we take further steps to mitigate route leaks?" Here some of the earlier efforts in RPSL might be useful. Route Origin Authorisations describe a permission from a prefix holder for an AS to originate a route for this prefix into the routing system. But that granted permission does not mean that the AS in question has accepted this permission. Indeed, there is no way of knowing in the RPKI the AS's view of this permission. What is the set of prefixes that the AS intended to originate? If this information was available, signed by the originating AS of course, then any other network could distinguish between an inadvertent route leak and deliberate intention by passing all the route objects that they see as originating from this AS through such a filter.

Are we ever going to secure the AS Path?

No!

Really?

With Path protection the entire RPKI structure is still ineffectual as a means of preventing hostile efforts to subvert the routing system. Any would-be hijacker can generate a fake route and others will accept it as long as an authorised AS is used as an originating AS. And poor use of the MaxLength parameter in ROAs and excessive AS prepending in AS paths make route hijacking incredibly simple. If RPKI rout origin validation is a routing protection mechanism, then its little different to wearing an impregnable defensive shield made of wet lettuce leaves!

So let's secure the AS Path and plug the gap!

Easier said than done, unfortunately.

The BGPSEC models of AS Path protection borrowed from the earlier s-BGP model. The idea is that each update that is passed to an eBGP neighbour is signed by the router using the private key of the AS in RPKI. But what is signed is quite particularly specified: It's the prefix, the AS Path and the AS of the intended recipient of the update. Actually, it's a bit more than this. It is signed over the signed AS Path that this network received. As an update is propagated across the inter-AS space the signing level deepens, This signing-over-signing makes any tampering with the AS Path incredibly challenging. Any rouge router in the path cannot alter the AS path that they've received, as they have no knowledge of the private keys of these ASes, and of course it cannot sign for future ASes that may receive the route object. And if it does not sign this AS Path as it propagates it then the AS Path can no longer be validated. So in theory its trapped and can't fake the AS Path.

All good, but there are quite a few practical problems with this approach:

- It's incredibly expensive to both calculate and validate these signatures. Even if you suppose that we can use high speed crypto hardware there is still the challenge that we'd like to be able to complete a full BGP session reset in no more than a couple of seconds. That's of the order of millions of crypto operations per second. That's hard to achieve on your everyday middle of the road cheap router. "Impossible" springs to mind.

- Getting private keys to routers is a nightmare!

- Routing policy is not addressed. Protocol-wise is quite ok for a multi-homed customer to re-advertise provider-learned route objects to another provider. Policy-wise it's a real problem and can be as bad, if not worse, than any other form of route leak.

- partial deployment is integral to this model. AS Path protection can only be afforded to internally connected "islands" and only to prefixes originated within these "islands" and the system cannot bridge gaps.

High cost. Low benefit.

It was never going anywhere useful.

An old idea first aired in soBGP from 20 years ago was dusted off and given a second airing. Here as AS declares its AS neighbours in a signed attestation. The assumption here is that each AS makes an "all or nothing" decision. Either an AS signs an attestation listing all of its AS neighbours or not. It can't just list a few. The implication is that if a party wants to manipulate an AS Path, then if it uses one of the ASes that maintains on of these AS neighbour attestations then it must also add a neighbour AS into the synthesised path (the all-or-none rule). The benefit to an AS of maintaining this neighbourhood attestation is that its more challenging to include this AS into a synthetic path, as then an attacker also has to include a listed neighbour AS. As more AS's create such neighbour attestations, they suck in their neighbour ASes. The result is that it is still possible to lie in the AS Path, but the scope for such lies is severely curtailed.

Such AS neighbour attestations can be handled in a manner analogous to ROAs. They are generated and published by the AS owner. clients can collect and validate these attestations in the same way that they collect and validate ROAs. And the result of a set of permitted AS adjacencies in AS Paths can be passed to a router as an externally managed filter set, much the same as ROAs.

It's close to full signing of an AS Path but does not require universal adoption and does not extract a heavy cost on the router.

It's looking a whole lot more promising, but it still has that policy hole.

And this is where ASPA comes in. It's still a draft in the SIDROPS working group, but the idea is simple. A customer AS signs an attestation that lists all its provider and peer ASes (all or none, of course). Now there is the concept of policy directionality and a BGP speaker can apply the "valley free" principle to AS paths. It's not comprehensive AS Path protection, but it further limits the space from which synthetic AS Paths can be generated and it's still useful even in the scenario of partial deployment.

But good as it sounds, I'm still not optimistic about its chances for deployment.

Not for any technical reason, but for the observation that in the engineering world we appear to have a limited attention span, and we seem to get just one chance to gain attention and access to resources to make the idea work. Route Origin Validation is what we came up with and all we came up with at the time. Perhaps cynically, I suspect that it may be what we are stuck with for the foreseeable future.

Any more questions?

This story is far from over, but I trust that this has shed some light on the design trade-offs that were behind the work so far, and point to some directions for further effort that would shift the needle with some tangible improvements in the routing space.

Disclaimer

Author

Geoff Huston B.Sc., M.Sc., is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region.

www.potaroo.net