Distributed Security Monitoring and Federated Learning of BGP Announcements Report

Abie Safdie
CS 630 - Distributed Systems
4 June 2024

ABSTRACT

Malicious BGP announcements present a serious security threat for networks in today's internet. By deploying distributed security monitoring and federated learning, which consists of sharing BGP announcement data to form central and local models, we are able to analyze the security risk of BGP announcements. To achieve this, each BGP peer possesses a "trust" attribute denoting its current trustworthiness. The more irregular path updates occur, the less trustworthy a peer becomes. When trust crosses a threshold, we flag irregular path announcements and notify a system admin that this path announcement is potentially malicious. Ultimately, by distributing information across BGP listeners we create a model of trustworthy paths and use this data to monitor the security of BGP announcements.

Keywords: Border Gateway Protocol (BGP), BGP Prefix Hijacking, BGP Announcements, Autonomous Systems, IP addresses, distributed security monitoring, federated learning, multithreading, synchronization, BGP Updates, naming, fault tolerance, consistency, distributed systems

1. Introduction

1.1. Routing, BGP, and BGP Prefix Hijacking

The internet's routers forward traffic using a procedure known as longest prefix matching. When a router receives an incoming packet, it examines its destination IP address and compares it with the IP prefixes in its routing table. The router chooses the entry in the table that has the longest prefix match with the destination IP, i.e., the longest matching number of initial bits. This entry then tells the router which interface to send the packet out on. This procedure is critical for forwarding packets across the internet [1].

BGP (Border Gateway Protocol) is an essential internet protocol that exchanges routing information among Autonomous Systems (ASes). To start, BGP provides each router with two significant pieces of information: prefix reachability and the "best" route to each prefix. Prefix reachability means that BGP enables a subnet to announce its existence and location to the rest of the internet. The "best" route to each prefix is determined based on policy and reachability information, which ensures data takes a preferred, specific path to its destination [1]. Ultimately, routers will exchange routing information over semi-permanent TCP connections. These connections are known as BGP connections, in which routers share BGP updates. Updates can take the form of announcements, an advertisement of a new path to a specific destination, or a withdrawal, the removal of a path to a specific destination [1].

BGP Prefix Hijacking is when an Autonomous System (AS) advertises a path to a specific IP address with malicious or false information. This presents a significant risk to internet security. By rerouting internet traffic, malicious actors can gain unauthorized access to data and cause significant economic damage to business. For instance, in 2008, a Pakistani AS announced a "shorter, more efficient" path to Youtube, causing traffic to be redirected to Pakistan. This BGP Prefix hijacking caused significant security and economic consequences for Youtube [2].

1.2. Federated Learning

Federated Learning is a distributed approach to machine learning where local nodes independently train models on their own data. Each node sends its trained model to a central source, which then uses the local models to form a centralized model. This process is iterative, as the nodes continuously send and receive data from the global model, improving the local and global models in the process [3].

Federated Learning can be leveraged in security monitoring by utilizing BGP listeners. These listeners, which are nodes that collect – "listen to" – all BGP updates from routers within a specific AS, can train local machine learning models on their collected BGP update data. In turn, these listeners can share their models to form a central model. However, to prevent this from being a centralized system, listeners will elect a leader, who will handle the mutual exclusion and receive the model updates. This will be discussed more in-depth.

1.3. Distributed Systems

Distributed systems seek to achieve six goals: sharing resources, transparency, openness, dependability, security, efficiency and scalability. However, in this class, we are omitting the security topic. Therefore, a distributed system with regards to federated learning would address each of these goals as follows:

1) Sharing resources: Sharing trained models across multiple BGP listeners to ensure each listener can accurately use their model to detect hijacking.
2) Transparency: Ensuring that local models are unaware that the model they use was trained using distributed resources, in this case BGP announcement data.
3) Openness: Allowing this distributed security monitoring system to be easily integrated into other systems. For instance, this system can easily be integrated into the longest prefix matching procedure.
4) Dependability: If one BGP listener's model flags an update as malicious, other routers can depend on this judgment to inform their future decisions, creating a reliable network of dependability and correctness.
5) Efficiency and Scalability: Distributing data across various listeners to more efficiently train models that monitor BGP announcements. Additionally, it is easily scalable as adding more listeners improves the local and global models by increasing the amount of usable data. This scalability aligns with distributed systems' goal of spreading and improving effectiveness as they grow.

By addressing these goals, a distributed system for monitoring BGP announcements can enhance the robustness, reliability, and efficiency of internet routing.

2. Existing Methods to Prevent and Detect BGP Prefix Hijacking and Monitor Announcements

Trust plays a key role in BGP announcements. Trusting a BGP peer to advertise a valid and legitimate update is critical in today's internet. However, there are currently a few methods employed to prevent BGP prefix hijacking. First, the deployment of BGPsec [4]. In standard BGP, the attribute AS-PATH contains the list of ASes that the advertisement has passed through. In BGPsec, however, the AS-PATH attribute is replaced by the SEC-PATH attribute [4], which cryptographically verifies the path that the ASes advertise. This attempts to prevent false announcements. However, the overhead associated with BGPsec is severe. Each path update requires cryptographic key verification, which hinders the performance of the system. Furthermore, for BGPsec to work more efficiently it requires wide-spread adoption. Another method to prevent BGP hijacking is Route Origin Validation (ROV) [5]. This method cryptographically validates only the origin of the BGP announcement. Again, this creates a

similar overhead. Ultimately, however, these methods are used to *prevent* BGP prefix hijacking.

The current solution for monitoring BGP announcements is real-time monitoring. This can be achieved in many ways, however, the most common is real-time monitoring by a system administrator. My prototype, on the other hand, aims to use machine learning to achieve this goal. As a result, it is possible in a future implementation to combine the existing methods of BGP prefix hijacking prevention with my distributed monitoring system to create a robust system.

4. Distributed Security Monitoring and Federated Learning Prototype

4.1. Federated Learning Architecture

I developed a federated learning architecture (Figure 1) where each BGP listener trains their local model on their own BGP data. The listeners use their local models to detect if incoming BGP announcements are potentially malicious and alert a system admin if necessary. At a set time, every month's worth of data in my prototype, each listener updates their own local model with respect to the new data. Then, each listener sends their model updates to the central model, which in turn shares and distributes their model to all BGP listeners. This iterative process continues throughout the lifetime of the system.

This approach capitalizes on the vast amount of BGP data, while keeping the training data local to each listener. The use of sharing models theoretically improves effectiveness. Furthermore, by having one BGP listener's model serve as the "central model," the BGP listener can take the role of "leader" and handle the coordination that is needed in a distributed system. This is explained further in the Multithreading section.

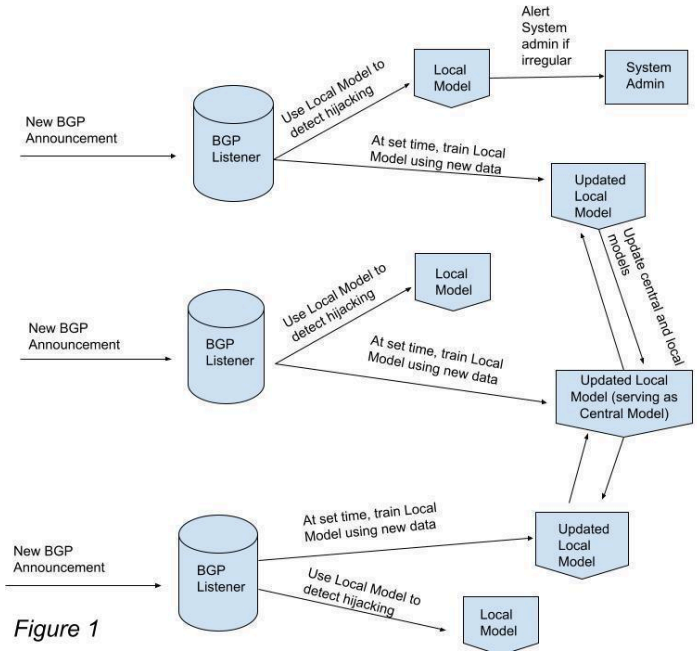

Figure 1

4.2. Multithreading and Consistency

I utilized multithreading in my prototype to simulate a real world environment. Each BGP listener object possesses its own thread. Therefore, each BGP listener receives their BGP updates in parallel. With this parallelism comes the need for synchronization. To solve this issue, a random BGP listener is chosen as "leader" and handles the coordination among the listeners.

This includes coordinating mutual exclusion for updating trust values and sharing and receiving the federated learning models. Note, for simplicity in my prototype, I used Python's built in lock method to avoid race conditions. Meaning, the OS is technically handling the mutual exclusion for the updating of BGP peer trust and the sending and receiving of models, but this can easily be adjusted in a real-world system. Ultimately, my system is resistant to deadlocks because I ensure that none of the four required conditions to create a deadlock occur simultaneously:

- Mutual Exclusion
- Hold and wait
- No preemption
- Circular Wait

All in all, by utilizing locks and synchronization methods, I effectively maintain consistency within the distributed system.

4.3. Federated Learning Procedure and Trust Algorithm

In order to effectively monitor BGP announcements using machine learning, each BGP listener needs to have an adequate model that is trained on real-world BGP data. Therefore, each BGP listener stores the BGP announcement data it receives from the RIPE database and uses it to train its model. However, due to time, knowledge, and resource restraints, I do not apply sophisticated machine learning techniques and algorithms. Instead, I apply basic frequency analysis to train models. Furthermore, I only use BGP announcement data to train my models. I do not use withdrawal data. These limitations are further expanded on in the discussion section.

With respect to my Python prototype, each BGP listener is an object, or class, that has access to a shared list of BGP peers that every listener has received announcements from. These individual BGP peers are objects themselves that possess two attributes: an IP address and a trust value.

Additionally, each listener possesses models for certain aspects of a BGP announcement. These aspects include: AS path length, AS path, and BGP community members. Each BGP listener creates three Python dictionaries that serve as "models." These dictionaries have keys of AS path length, AS path, and community. The values of the dictionary are the number of times a specific path length, path, or community has been observed. Therefore, when a new announcement is received, the listener runs the BGP announcement through the models and returns the frequency of how often that path has been seen. For example, if a specific AS path was advertised by a BGP peer to a specific destination, the listener that observed this announcement would check how many times they have seen this specific path to this specific destination. The model would then divide the number of times they have observed this specific path by the total number of path announcements to the specific destination. This returned frequency value is what is used in my trust algorithm.

The returned frequency value denotes how "trustworthy" an announcement is. Consequently, this value is then applied to the BGP peers' overall trust attribute. The change in trust to a BGP peers' overall trust attribute is illustrated in the following formula, which is applied every announcement that the BGP peer advertises:

$$\Delta\text{Trust} = \sum_{i=0}^{n-1} \left(1 - e^{-k|1.75 \cdot \text{bgp\_attribute}_i - 1|}\right) \cdot \text{sgn}(1.75 \cdot \text{bgp\_attribute}_i - 1) \cdot d$$

- Where n is the number of BGP announcement attributes used, of which there are three: path length, path, and community
- k and d are scaling constants
    - k = 0.2 $\forall$ bgp_attributes
    - d = 0.4 for path length and community, 0.75 for path
- Sgn returns 1 if the returned value is positive, negative 1 otherwise

This formula iterates over the specific BGP attributes – path length, path, and community members – and sums their respective outcomes to create a delta trust value. This delta trust value is then added to the BGP peer's trust attribute. For example, if a BGP peer has a trust attribute of 93 and advertises an announcement that returns a delta trust of -3, the BGP peer's new trust attribute is 90.

If trust falls below a certain threshold, 60 percent of the 100 base trust, then any negative delta trust update will get flagged and a system administrator will be notified of the potential malicious BGP announcement.

The listener finds the $BGP\_attribute_i$ value by referencing its models and calculating the ratio of similarly observed data to the total amount of data. All attributes have the same scaling constant k. For d, path has a value of 0.75, while all other attributes have a value of 0.4. The purpose of this is to apply greater weight to the path attribute, so it affects the delta trust the most.

The purpose of this formula is to make announcements that are further away from a base frequency (~57 percent) exponentially more punishing or rewarding to a BGP peer's trust. Therefore, announcements that advertise new paths are treated as extremely untrustworthy, however, the more this new path gets advertised over a long period of time, the more trustworthy it becomes. This simple frequency analysis provides the ability for a BGP listener to analyze BGP peers and their announcements as potentially malicious.

4.4. Fault Tolerance, Leader Election, and Naming

The leader is elected via an election by bullying algorithm. The BGP listener with the highest ID gets the role of leader and handles the mutual exclusion and the sharing of models. In case the leader goes out of commision, the system must be able to elect a new leader in order to achieve fault tolerance. The new leader is elected via the same election by bullying algorithm. If new BGP listeners with higher IDs were to enter the system, an election is not called. In my system, an election is only called when the current leader is no longer responding.

My system utilizes listener ID's and BGP Peer IP addresses as the naming scheme. To achieve name resolution, I simply run a search algorithm in the respective list of listeners and BGP peers. However, if my system were to be scaled up, a faster, more efficient name resolution method should be developed.

4.5. Distributed Security Monitoring

Each BGP listener has access to a global, shared list of BGP peers. For a BGP peer to be added to this list, they must have sent an announcement that was received by any listener. The listener then adds the peer to the shared list. These peers possess a trust attribute denoting their current trust. Due to each listener being able to access and update any BGP peers' trust value, the system is able to perform security monitoring in a distributed manner. This is due to each listener acting as its own system and being able to share resources (information on BGP peers) with other listeners. Therefore, by monitoring trust values in a distributed manner, this system effectively performs distributed security monitoring and achieves the goals of a distributed manner by doing the following:

- Sharing resources: Shares BGP peer information. Including trust and IP addresses
- Transparency: Listeners are unaware that other systems are editing BGP trust values
- Openness: Can easily integrate into other internet protocol systems. For instance, a BGP announcement monitoring.
- Dependability: One listener can reliably trust another listener's edits to a BGP peer's trust value. These edits will inform future alerts to system administrators if the trust level falls below a specified threshold.
- Scalability: Adding more listeners and BGP peers to the system can be seamlessly integrated, enhancing both effectiveness and efficiency. The distributed system will benefit from an increased data pool, enabling more accurate predictions.

5. Evaluation Methodology

To evaluate the system's effectiveness, I manually examined several BGP announcements to verify the expected behavior. For example, if a BGP peer announces an unseen path to a destination, I expect the peer's trust to decrease significantly. Conversely, if a BGP peer announces a path to a destination that has been seen infrequently, I expect the trust to decrease, but to a lesser extent. I expected the inverse results for paths that have been seen frequently. Ultimately, when performing these manual examinations, my expectations were validated as this behavior is what my system produced for those specific BGP updates.

To test the effectiveness of the entire system, I saved the "final" data that my system produced. Note, theoretically my system would run indefinitely if applied to the real-world.

Therefore, “final” results just refer to the data at the time I stopped allowing my listeners to receive BGP announcement data. The data included the “final” trust values of all the BGP peers that listeners observed updates from. I plot this data using the MatPlotLib python library for analysis.

6. Results and Analysis

Test results were collected for two sites: Youtube.com (208.65.153.238) and University of Oregon (140.211.0.0/16). For Youtube, I observed the following results:

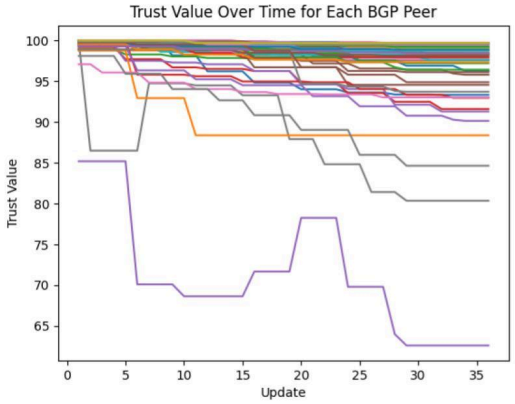

Figure 2 - Youtube.com

Each line on the corresponding graph (Figure 2) represents a BGP Peers trust value over time. Note, due to space limitation, peers IP addresses are not labeled on the graph. However, the identification of each peer’s trust value is saved locally in “{name_of_site}.txt” at the termination of the program for further analysis. An additional note, the x-axis label “update” does not refer to BGP updates, but rather to a randomly chosen time in which I plot, or “update”, every peer’s trust value on the graph. With regards to the data in Figure 2, most peers send minimal announcements over the timeframe I observed and as a result do not have their trust value change significantly. On the other hand, some peers send numerous announcements and have their trust fluctuate dramatically. This is seen most evidently in the bottom purple line. Starting at update zero, this peer exhibited the following sequential behavior:

1) Multiple announcements of very untrustworthy paths, causing significant decrease in overall trust
2) A moment of minimal announcements, causing minimal change in trust
3) A brief period of announcing trustworthy paths
4) Lastly, a period of announcing untrustworthy paths

A path is defined as "untrustworthy" or "trustworthy" by the machine learning model of the listener that heard the announcement. The returned delta trust value is the “trustworthiness” of the advertised path.

Overall, this behavior resulted in an overall trust value of ~62. Based on the aforementioned trust algorithm, if this peer had crossed the 60 trust value threshold, the system would alert a system administrator for any future negative delta trust announcements (as long as the overall trust remains below the threshold).

For the University of Oregon’s prefix range (140.211.0.0/16), I observed the following results:

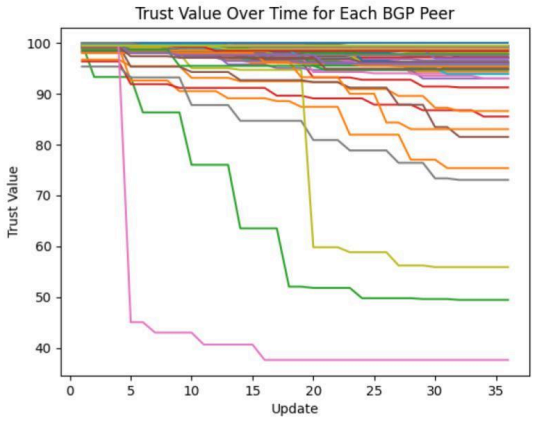

Figure 3 - UOregon

With regards to the data in Figure 3, there is a notable, drastic drop in a few peers' trust value. These drops are due to the announcement of very irregular, untrustworthy paths. For the bottom pink and green lines, once they crossed the 60 trust value threshold, system administrators were notified every negative delta trust announcement. Note, in my prototype, notifying a system admin was just printing to standard out: “alerting a system admin.” This would obviously change in a real-world, scaled up system.

As a whole, this system produced results that are indicative of an effective Distributed Security Monitoring system. Paths that are notably irregular and obscure get correctly identified and cause a negative impact on a BGP peer's trust. Paths that are well-established have a positive impact on a BGP peer’s trust.

More data and further analysis could be collected and conducted, however, to *prove* the effectiveness of this system beyond a reasonable doubt. As well as scaling the system up and using more robust methods to perform distributed security monitoring of BGP updates. These limitations and future developments are discussed further in the discussion and lessons learned sections.

7. Discussion

7.1. Federated Learning

The distributed security monitoring system I developed does not use sophisticated machine learning techniques or tools. The reasons for this were due to time, knowledge, and resource constraints. I had originally wanted to use tensorflow, sci-kit learn, and other machine learning libraries, but I had very little experience in them and did not have the time and resources to use them the way they are intended. Instead, I applied very basic frequency analysis on BGP announcements to develop “models”. Furthermore, the RIPE database does not state what updates are malicious or false in nature. Therefore, my system performs unsupervised learning, which is the process of interpreting data that is not labeled. This causes a severe limitation in my system. I immediately label any unknown, new announcement as untrustworthy and therefore potentially malicious. If I had more time and access to more computing power, I would use sophisticated machine learning to develop the models that would avoid this limitation.

Additionally, my system looks specifically at BGP announcement data. It does have withdrawal updates impact the machine learning models. I avoided using withdrawal data because of the extra layer of complexity it would add to my system. However, in a more developed system, adding withdrawal data to the training data would create more robust and sophisticated machine learning models.

7.2. RIPE and BGP Update Attributes

The RIPE database provides BGP updates for a predefined resource. Meaning, give RIPE the destination IP and a listener and RIPE will return all the BGP updates for that destination the listener observed over an inputted time frame. Each individual BGP update contains the following:

- The type of update (Announcement or Withdrawal)
- Timestamp of update
- 4 specific attributes
    - Target prefix (i.e. destination IP range)
    - AS Path
    - Communities (i.e. information on traffic engineering and routing policy)
    - Source ID (i.e. Listener’s ID)
- Query start and endtime
- Number of Updates observed in specified time window

To limit the scope of my project, I only used four bits of information in my system: the type of update, the AS path, the AS path length (found by counting the ASes in the path), and the communities. However, to scale my system

up and have it factor in more important information, a future system should utilize all the information that the RIPE database provides on BGP updates.

For example, a typical machine learning use case might involve using timestamps and the number of updates within a specific time frame to analyze the behavior of a particular BGP peer. This approach helps to gain a deeper understanding of the paths they announce, including when and how many, providing better insight into the peer's overall behavior.

7.3. Trust

In the currently developed system, each BGP peer has a dynamic trust value that changes in accordance with each announcement it advertises. My defined trust algorithm, however, has certain limitations. First, it treats all BGP announcement attributes (except the AS path) equally. To fix this, I would develop a more sophisticated formula that assigns appropriate weights to each attribute. Furthermore, I would adapt my algorithm so that it is compatible with whatever form the more sophisticated machine learning models take. Lastly, I would re-define the 60 trust value threshold that is required to notify a system admin. I would allow cases where if the sophisticated machine learning model detects an extreme irregularity, the listener would have the ability to alert the system admin immediately, regardless of the trust value of the peer.

7.4. Distributed Security Monitoring as a distributed system

With regards to the distributed system as a whole, it possesses a couple notable shortcomings. First, a real-world system would run indefinitely. My system does not do that. Rather, it processes data in bunches (1-month's worth) and terminates at the timestamp specified by the user. Additionally, with the limited time to develop this project, my code has some simplifications that hinder its authenticity. For example, I use python's built in lock method to handle mutual exclusion, which means that the OS is handling the coordination among the listeners. In a real-world system, one of the listeners would take control of the coordination. Furthermore, I keep my data stored locally in shared data structures. In a real world system the amount of data collected would be colossal and would require dedicated data storage.

8. Conclusions

Overall, the Distributed Security Monitoring and Federated Learning of BGP Announcements prototype produced promising results, demonstrating its potential for scalability and real-world application. It effectively identified irregular advertised paths and established a federated learning-driven security monitoring system by continuously monitoring BGP peers and their announcements in real-time.

Although elementary in nature, the developed federated learning system and its architecture showed promise as a suitable method that could be applied and scaled up.

Furthermore, monitoring BGP peers by defining them a "trust" attribute worked to create a history and "biography" on each BGP peer to better understand their behavior over time.

Lastly, the developed system was effective as a distributed system. As previously stated, it achieved the goals of a distributed system. Additionally, it adhered to many key distributed systems concepts, including coordination, naming, communication, consistency, and fault tolerance.

9. Lessons Learned

This project was extremely challenging and time-intensive. At the start of the term, it

was particularly difficult to lay the foundation for building a distributed system without any prior knowledge of what a distributed system was, its goals, or the concepts it should adhere to. Furthermore, I believe I was overly ambitious in the scope of the project I decided to undertake. Networking and machine learning are extensive disciplines within computer science and attempting to combine the two, along with all their intricacies, proved to be especially challenging. Ultimately, however, at the conclusion of this project, I feel I have gained a significant amount of knowledge and practice in building a distributed system.

10. References

[1] Kurose, J. F., & Ross, K. W. (2022). *Computer networking: A top-down approach*. Pearson Education Limited.

[2] *YouTube hijacking: A ripe NCC RIS case study*. RIPE Network Coordination Center. (n.d.). https://www.ripe.net/publications/news/youtube-hijacking-a-ripe-ncc-ris-case-study/

[3] Martineau, K. (2023, August 18). *What is federated learning?*. IBM Research. https://research.ibm.com/blog/what-is-federated-learning

[4] Li, Q., Hu, Y.-C., & Zhang, X. (n.d.). *Even Rockets Cannot Make Pigs Fly Sustainably: Can BGP be Secured with BGPsec?* https://www.ndss-symposium.org/wp-content/uploads/2017/09/02_3-paper_0.pdf

[5] *BGP origin validation*. RIPE Network Coordination Center. (n.d.-a). https://www.ripe.net/manage-ips-and-asns/resource-management/rpki/bgp-origin-validation/

11. Appendix

11.1. RIPE Database and Python

RIPE is an internet registry that provides a vast database of BGP updates. They collect BGP data via “route collectors,” which serve as BGP listeners, listening and storing BGP updates from various BGP Peers. RIPE possesses 23 BGP listeners across the world. I utilized data from four listeners: RRC 11 (New York City), RRC 14 (Palo Alto), RRC 16 (Miami), and RRC 21 (Paris, France). All of the listeners I used collected their data from internet exchange points.

In my prototype, each listener trains their initial base models on three months worth of data. Furthermore, when training the models, RIPE requires a resource to get the BGP updates for. This could be an IP prefix range, an IP address, or an entire Autonomous System. I collected BGP data for 208.65.153.238 (youtube.com) and 140.211.0.0/16 (University of Oregon). In a real-world system, this can be scaled up.

I use multiple Python libraries to aid in my prototype. First, I use the threading library to introduce multithreading and synchronization methods into my prototype. Second, I use Python’s math library to introduce elementary math functions to aid in the development of my trust algorithm and federated learning system. Lastly, I use the matplotlib library to graph and analyze my results.

See the attached README.md for more information on how to use the system I developed.