Assignment 2: BGP Hijacking Attacks (ver1.1)

CS348 Introduction to Information Security, KAIST
(2024 Spring)

Due: 11:59 PM (KST), March 27, 2024

INSTRUCTIONS TO STUDENTS:

- Any collaboration or assistance of any kind is strictly prohibited; all work must be your own.
- Your submission *will surely* be compared with the submissions for your peers for plagiarism detection (e.g., MOSS). Any academic dishonesty will be directly reported to the university.

1 BGP Announcement Simulator

For this assignment, you will be given a full-functioning BGP simulator written in Python. While the simulator is a simple, single-threaded implementation, it is capable of simulating the basic BGP route announcement propagation and the BGP policy implementation at the Internet scale (e.g., simulating real AS topology of the Internet). In particular, it takes a public CAIDA AS business relationship dataset (which can be found in `20240301.as-rel.txt}` and simulates the route propagation decisions based on the BGP policy implementation in each AS. Users can also customize the BGP policy for each AS so that they can experiment the impact of different BGP policy on global routing.

Main input: BGP prefix announcements. The main input to the simulator is the list of BGP prefix announcements. Multiple BGP prefix announcements can be made in a single simulation run and the simulator will propagate each announcement from the corresponding destination AS to the rest of all ASes in the world. Note that this simulator process only one BGP prefix announcement at a time. That is, it simulates the global propagation of a single BGP prefix announcement, and then the next BGP prefix announcement is simulated. For example, if you simulate two cases of BGP prefix announcements {147.201.27.0/24, [10]} (i.e., an AS with AS number 10 announces the prefix 147.201.27.0/24) and {4.3.74.0/24, [20]} (i.e., another AS with AS number 20 announces the prefix 4.3.74.0/24), the simulator will first simulate the global propagation of the first announcement and then the second announcement. Only after the first announcement is fully propagated (i.e., no more ASes to propagate the announcement to others), the simulator will start the simulation of the second announcement.

Discret time-driven simulation. This simulator is a discret time-driven simulation in the sense that the propagation of BGP announcements is simulated in discrete time steps, called ticks. Each tick represents a small unit of time, sufficient for an AS to process the BGP announcement and forward it to its neighboring ASes. Thus, after each tick, all the ASes that have received one or more new BGP announcements process them and forward them to their neighboring ASes (if needed). For each BGP announcement, the simulator advances the time in ticks until the announcement is fully propagated to all ASes in the world. When the announcement reaches all the ASes in the world, and no more ASes are left to propagate the announcement, the simulator stops the simulation for that announcement and moves on to the next BGP announcement. When all the BGP announcements are fully propagated, the simulation ends.

Main output: BGP table. The simulation outputs the global BGP table, which contains the best route for each BGP prefix at each AS. This global BGP table is maintained in the simulation as a single

Python dictionary object `bgp_table` and is updated at each tick as the BGP announcements are propagated. The table is defined with the AS number (or asn) as the key and the value is another dictionary object that contains the prefix as the key and the value is the AS path to the destination AS. For example, the global BGP table can be represented as follows: `{7551, {147.201.27.0/24, 10>10753>3356<4826}}`, which means that AS 7551 has the best route to the prefix 147.201.27.0/24 and it can reach the prefix at the destination AS 10 via AS 4826, AS 3356, and AS 10753, in that order. Notice that the ASes in the AS path are ordered from the destination AS (leftmost) to the next-hop AS (rightmost); thus, any data packet sent by AS 7551 will be forwarded to AS 4826 first, then to AS 3356, to AS 10753, and finally to AS 10. Note also that the AS path is represented as a string of AS numbers separated by the character $>$, $<$, or $-$, to help the manual inspection of the AS path. The direction of the brackets indicates the payment direction of the two ASes; e.g., AS 10 pays AS 10753. The symbol $-$ is used to indicate the two ASes are in a free peering relationship. All these business relationships are based on the public CAIDA AS business relationship dataset[1]. When the simulation ends, the simulator outputs the global BGP table to a file named `bgp_table.txt`. By inspecting the output BGP table, you can see how the BGP policy of each AS has affected the global routing.

Caveat. While this simulator is a very useful tool for understanding the BGP route propagation and it provides a good approximation of the real-world BGP route propagation, it is not perfect. The simulator is based on the public CAIDA AS business relationship dataset, which is a good approximation of the real-world business relationships among ASes, but it may contain some errors. Also, a single AS may have multiple BGP policies applied differently at different point-of-presences (PoPs) and the simulator does not model such complex BGP policies. Therefore, the simulated BGP routes should not be used to make any real-world routing decisions.

Copyright. The simulator is provided by the instructor Min Suk Kang and is copyrighted. It is provided to the students for educational purposes only, particularly for the BGP hijacking assignment in CS348 Introduction to Information Security at KAIST. The students are not allowed to distribute the simulator to others or use it for any other purposes than the assignment. Particularly, sharing the original or modified version of the simulator through github or any other public or private repositories is strictly prohibited. Please refer to the copyright notice in the simulator code.

2 Task 1: Basic BGP Policy Implementation (40 points)

The simulator is missing some basic BGP policy implementation. Your first task is to implement the following basic BGP policy in the simulator. By filling in the missing parts of the simulator and completing the implementation of a specific function that compares the costs of two routes, you will be able to simulate the BGP announcements that reflect the realistic BGP policy implementation.

The provided `program_0` may run but its results can be incorrect. You should complete the basic BGP policy implementation to correct it. Look for the Python comments `TASK 1` in the simulator code and complete the missing parts. Once you complete the implementation, you can run the simulator as follows:

```
python3 bgp_announcement_sim_program_1.py bgp_announcement_list.txt
```

Cost comparison. You will need to complete a function that compares the costs of two routes (one is the current best route and the other is the new route to be compared with the current best route). You can find the necessary rules from the lecture slides and a summary is provided below.

1. Local preference (a.k.a., business relationship): Each AS has a strong local preference for the routes received from its customers over the routes received from its peers, and the routes received from its peers over the routes received from its providers. If the two routes are received from the same type of ASes (e.g., both are from customers), the AS considers the two routes to be equal in terms of the local preference.

[1]`https://www.caida.org/catalog/datasets/as-relationships/`

2. AS path length: If the two routes are received from the same type of ASes (thus, they are tied in the local preference), the AS prefers the route with the shorter AS path.

3. Tie-breaking: If the two routes are still tied after the above two rules, the AS breaks the tie by the following simple rule: The AS chooses the next-hop AS with the smaller hash value of the AS number. For this assignment, use the SHA-1 hash function `int(hashlib.sha1(str(ASN).encode()).hexdigest(), 16)` to calculate the hash value of the AS number.[2]

Submission. Your submission will include the file `bgp_announcement_sim_program_1.py` in the final tar file (see Section 7). Any deviation from the submission format (e.g., wrong file names) will result in a penalty.

Evaluation. We will evaluate your implementation by running the simulator with the BGP policy you implemented and checking the output BGP table. The command to evaluate the simulator is as follows: `python3 bgp_announcement_sim_program_1.py bgp_announcement_list.txt`. This should result in the output file `bgp_table.txt`, which is compared with the expected BGP table. We will not provide the BGP announcement list file we used for the evaluation or the expected BGP table.

How do I know if my implementation is correct? You can check the correctness of your implementation by inspecting the output BGP table. With the knowledge of the BGP protocol and the BGP policy you learn in the class, you can manually inspect the output BGP table and see if the BGP policy you implemented is correctly reflected in the BGP table.

3 Task 2: Prefix Hijacking Attack (15 points)

In this task, you will be able to test whether the prefix hijacking attack is successful in the simulator. To test attacks, you need to modify the BGP announcement list file, which contains multiple BGP prefix announcements. The same prefix can be announced by multiple ASes, where one of them is the legitimate destination AS and the others are the attackers. When the announcement by the legitimate destination AS is propagated to all the ASes first, the announcement by the attacker ASes will be sent to all the ASes in the world. Some ASes are said to be the victims of the prefix hijacking attack if they end up choosing the route announced by the attacker ASes. Note that some ASes may still choose the route announced by the legitimate destination AS, even if the attacker ASes announce the same prefix. This is because the legitimate announcement may still seem to be the best route to some ASes, based on the BGP policy.

Submission. You do not need to submit any code for this task as we will use the simulator you submitted in Task 1 `bgp_announcement_sim_program_1.py`.

Evaluation. The command to evaluate the simulator is as follows:

`python3 bgp_announcement_sim_program_1.py bgp_announcement_list.txt` (i.e., the same command as in Task 1).

For this evaluation, we will include one or more prefix hijacking attacks in the BGP announcement list file and check the output BGP table. The output file `bgp_table.txt` will be compared with the expected BGP table. Again, we will not provide the BGP announcement list file we used for the evaluation or the expected BGP table.

4 Task 3: Prefix Hijacking Detection with RPKI (15 points)

In this task, you will be able to test a simplified deployment of RPKI in the simulator. In your simulation, you will apply the RPKI Route Origin Validation (ROV) to some ASes and see whether the RPKI ROV can mitigate the prefix hijacking attacks to some extent.

[2]Note that this tie-breaking rule is a hypothetical one proposed only for this assignment. Real-world BGP routers may use other tie-breaking rules.

RPKI ROV. When an AS receives a BGP prefix announcement, if the AS is configured with RPKI ROV, it checks the RPKI database to see if the origin AS of the prefix announcement is authorized to announce the prefix. If the origin AS is not authorized to announce the prefix, the announcement is considered invalid and the AS will not propagate the announcement to its neighboring ASes.

RPKI database. We provide a simplified RPKI database for the simulation. In your simulation, read the file `20240301.asn-prefix-database.txt` and finds the pairs of AS numbers and the prefixes that are authorized to be announced by the ASes. For simplicity, our RPKI database contains only one prefix for each AS number and all the prefixes are /24 prefixes. Note that the legitimate, authorized prefixes in the RPKI database are made up for our simulation and do not reflect the real-world RPKI database.

Where to deploy RPKI ROV? You are asked to deploy RPKI ROV to all the tier-1 ASes but not to other ASes. Tier-1 ASes are the ASes that have no provider ASes and only have customer ASes and peer ASes. Since the deployment of RPKI ROV is not universal, you may still see some prefix hijacking attacks successful at some ASes.

Submission. Your submission will include the file `bgp_announcement_sim_program_2.py` in the final tar file (see Section 7). Any deviation from the submission format (e.g., wrong file names) will result in a penalty.

Evaluation. The command to evaluate the simulator is as follows:

```
python3 bgp_announcement_sim_program_2.py bgp_announcement_list.txt.
```

We will test several prefix hijacking attacks in the BGP announcement list file and check the output BGP table. The prefix announcement list and the expected BGP table will not be provided.

5 Task 4: 1-Hop Prefix Hijacking Attack (15 points)

In this task, you will be able to test a 1-hop prefix hijacking attack in the simulator. In the 1-hop prefix hijacking attack, the attacker AS pretends to be the 1-hop neighbor of the legitimate destination AS of the prefix. While RPKI ROV can mitigate the 1-hop prefix hijacking attack to some extent, it is completely ineffective against the 1-hop prefix hijacking attacks because the destination AS is indeed the legitimate origin AS of the prefix in the 1-hop prefix hijacking attacks.

Submission. You do not need to submit any code for this task as we will use the simulator you submitted in Task 3 `bgp_announcement_sim_program_2.py`.

Evaluation. The command to evaluate the simulator is as follows:

```
python3 bgp_announcement_sim_program_2.py bgp_announcement_list.txt.
```

We will test several 1-hop prefix hijacking attacks in the BGP announcement list file and check the output BGP table. The prefix announcement list and the expected BGP table will not be provided.

6 Task 5: 1-Hop Prefix Hijacking Detection with BGPSEC (15 points)

In this last task, you will be able to test a simplified deployment of BGPSEC in the simulator. BGPSEC is the security extension of BGP that provides the origin authentication and path validation of BGP announcements. That is, with BGPSEC, an AS can tell whether the attacker AS is actually the 1-hop neighbor of the legitimate destination AS of the prefix or not. If it sees an announcement that contains an invalid AS path (due to the 1-hop prefix hijacking attack), the AS will be ignored and not propagated to its neighboring ASes.

BGPSEC deployment. In the real world, BGPSEC deployment requires intricate digital signature operations across all the ASes on a path of a BGP announcement. However, to (greatly) simplify our simulation of BGPSEC, we will simply assume that the ASes that are configured with BGPSEC can validate whether the

AS path of a BGP announcement is valid or not by referring to the CAIDA AS business relationship dataset. That is, if the CAIDA dataset (which is already used in the simulator) contains the business relationship between two ASes, our BGPSEC implementation will consider the AS path to be valid.

Where to deploy BGPSEC? In this task, you are asked to deploy BGPSEC to all the ASes in the world. That is, we assume the complete deployment of BGPSEC in our test. Note that in the real world, the deployment of BGPSEC is almost non-existent due to several security and operational challenges.[3]

Submission. Your submission will include the file `bgp_announcement_sim_program_3.py` in the final tar file (see Section 7). Any deviation from the submission format (e.g., wrong file names) will result in a penalty.

Evaluation. The command to evaluate the simulator is as follows:

```
python3 bgp_announcement_sim_program_3.py bgp_announcement_list.txt.
```

We will test several 1-hop prefix hijacking attacks (along with prefix hijacking attacks) in the BGP announcement list file and check the output BGP table. The prefix announcement list and the expected BGP table will not be provided.

7 Submission

Total three Python scripts are required for the submission:

`bgp_announcement_sim_program_1.py`,
`bgp_announcement_sim_program_2.py`, and
`bgp_announcement_sim_program_3.py`.

Tar three scripts into a single tar file named `HW2_#.tar`, where `#` denotes your student id. The tar file should not contain any other files or directories. Any deviation from the submission format (e.g., wrong file names, wrong formats, etc) will result in a penalty.

Submissions should be made to the gradescope system. Any other means of submission will not be accepted. There will be no exceptions to this rule.

Late policy. 80% credit will be awarded for 1-day late submission, 50% credit for 2-days late submission. NO credits will be awarded for submissions made later than 2 days after the given deadline. No exceptions will be made to this policy.

Plagiarism. Any collaboration or assistance of any kind is strictly prohibited; all work must be your own. Your submission will be compared with the submissions for your peers for plagiarism detection (e.g., MOSS). We will hold a zero-tolerance policy for academic dishonesty.

[3] See the following paper for more details: Lychev, Robert, Sharon Goldberg, and Michael Schapira. “BGP security in partial deployment: Is the juice worth the squeeze?.” *In Proceedings of the ACM SIGCOMM*, 2013.