Remedying BGP Myopia: A Blockchain Solution

University of Oregon

Nick Anthony nanthon7@uoregon.edu

Geoffrey Brendel gbrendel@uoregon.edu

Joseph Erlinger jerling2@uoregon.edu

1. Abstract

A growing constriction in the world of BGP security is the nearsightedness of BGP routers. Simply put, BGP routers cannot reconstruct route propagation history, so their knowledge of their own surrounding topography is limited at best. Additionally, with the most common BGP security protocol, BGPSec, security features are dropped if a secure BGP packet passes through a router that does not implement the protocol. This requires that all routers in a path must implement the protocol for the path to be secure. To remedy this security issue using blockchain, multiple solutions have been suggested, but previous research in this subject has introduced methods that modify base BGP communications and administrative bodies in such a way that successful implementation would require exceeding the adoption rate of BGPSec and other security protocols. In this paper, our approach to a blockchain solution to BGP security is thus: we choose not to focus on the decentralization aspect of blockchain, but rather its ability to be an immutable ledger. By storing paths both heard and sent, routers aiming to evaluate the security of a given BGP announcement may check the blockchain and verify that the announcement traveled a secure path before arriving at the destination. Additionally, through a method we call “neighborhoods”, security may also be achieved by verifying neighbors and storing their positions in the blockchain, preventing path and origin spoofing. Our results show – preliminarily and with some restrictions – the general algorithm behind our implementation is sound, and we can ensure that some paths can be one hundred percent secure without all routers in said path subscribing to our protocol.

2. Keywords

BGP security, blockchain, prefix hijacking, internet security, BGP

3. Introduction

Border Gateway Protocol (BGP) is the glue that holds the internet together. Through a complex network of Autonomous Systems (ASes), BGP routers around the world communicate with each other by maintaining a list of which IP prefixes belong to which AS, and a path to follow to reach them. By nature, BGP is a relatively simple protocol: all paths are trusted by default, and the shorter path is always preferred. This dynamic creates a prime opportunity for threat actors to take advantage. Prefix/origin hijacking, wherein a malicious AS advertises a shorter path to a prefix it does not own, is a prime example of such a vulnerability.

To remedy this issue, two primary solutions emerged: Resource Public Key Infrastructure (RPKI) and BGPSec. RPKI is an adaptation of Public Key Infrastructure (PKI), and allows for prefixes to be cryptographically bound to their AS through an object called a Route Origin Authorization (ROA). ROAs are kept in a centralized repository and utilized in Route Origin Verification (ROV) when the owner of a given prefix or set of prefixes needs to be confirmed or verified [1]. BGPSec is an opt-in protocol that allows network operators to append signatures and hashes in an encapsulated fashion so that the security and authenticity of a given path is ensured [2].

Although these solutions do work and are commonplace in the real world, there are a couple of glaring issues that still need to be addressed. Firstly, the security that RPKI and BGPSec respectively provide are limited by the rate at which they are adopted by network operators - and BGPSec has still not been widely deployed despite the glaring security issues with BGP [3]. Second, BGPSec is not able to provide security for a given path unless every router in the path implements it [2]. Some promising solutions to this second problem have been addressed, such as BGP-iSec [4], but practical implementation is still a ways off. The challenge with BGP security is the question of adoption versus security, given that any security solution is seldom adopted by everyone: how can a path achieve one hundred percent security without every single AS in the path implementing a given security protocol?

Recent investigations in the applications of blockchain to BGP security have borne some fruit, expanded on further in our discussion in the Related Work section, but one common theme among blockchain solutions is the aim to modify core aspects of BGP infrastructure and the chain of authority. Transforming the internet into a decentralized, blockchain-forward network is an admirable aim, but without compulsion to do so would split the internet between the decentralized blockchain nodes and legacy BGP infrastructure.

Our protocol, henceforth called S-eBGP, is comparatively non-invasive, with each secure eBGP router being completely compatible with any other eBGP router running a different (or no) security protocol. While other solutions focus on transforming the internet into a wallet-based, consensus-focused, decentralized model, we believe that the true value of the blockchain is in its ability to act as an immutable ledger. Without danger of threat actor modification, S-eBGP routers can trust that the information contained in the blockchain is legitimate, and therefore provides a basis to validate paths. With a focus on ledger capabilities, much of the complexity of decentralization can be avoided, therefore making it easier for network operators to understand and implement S-eBGP themselves.

In this paper, we will describe and demonstrate the methodology with which we use to comprise S-eBGP, as well as multiple simulations showing its efficacy and plausibility in realistic network topology scenarios. We will discuss some assumptions we put in place while developing this protocol and limitations that we encountered or that will be encountered in future research. We expect that this solution is relatively novel in the field of blockchain-based BGP security, and hope that it offers a different take on the role of blockchain in BGP security. Further research and development may be required in order for our solution to be practically viable.

4. Related Work

Over the course of our research, we encountered myriad blockchain-based BGP solutions, each purporting to solve different issues. We will focus on the ones that specifically aim to enhance security and prevent hijacks. Our focus was trained mostly on Auckland University of Technology’s solution [5],

DeBGP [6] and RouteChain [7]. AUT's proposal, like ours, has two types of blocks. One type acts as an ROA, and the other maps ASes to their directly connected neighbors [5]. We felt we could improve upon this structure, as our protocol is not aiming to replace ROAs (rather, we would like RPKI to be compatible in future iterations) and doing so would disrupt and complicate route origin infrastructure. Instead, we opted to store heard route announcements in an attempt to allow routers to reconstruct route propagation history. This allows for a more symbiotic relationship between the two types of blocks, which we will expand upon further in the Implementation Methodology section.

DeBGP aims to solve the single-point-of-failure problem that traditional RPKI presents by decentralizing route origin authorizations and, similar to s-eBGP, storing BGP updates in the blockchain for further verification [6]. ASes are assigned to "consortiums" which share a common node and access to the master blockchain/ledger [6]. The primary issue that we saw in this method was the disconnect between the complexity and level of change DeBGP introduces to the BGP zeitgeist and the relative willingness or likelihood of network operators and regulatory bodies to implement said solution.

RouteChain is fairly similar to our solution, with a similar acknowledgment that blockchain presents a unique opportunity for backwards compatibility with non-secure ASes. However, one fatal flaw with RouteChain is that its focus is too narrow: the fundamental aim of RouteChain is to prevent prefix hijacks [7], but does little to improve the nearsightedness of BGP routers nor does it protect its non-secure neighbors when a hijack is detected. When researching RouteChain, we found that one way we could improve upon that model would be to drop non-secure paths when heard by a secure router in order to protect its neighbors.

To conclude and summarize, AUT's solution and DeBGP do not achieve higher security than adoption, because for said solutions to work, the entire internet (i.e every AS) must convert. Even in edge cases like RouteChain, the efficacy leaves something to be desired. These related works helped us realize some of the gaps in practicality that were present in recent state-of-the-art solutions.

5. Implementation Methodology

Our implementation involves multiple classes and modules consisting of the following: "EBGPRouter", "SecureEBGP", "Blockchain", and "Block". In addition the simulations folder consists of 6 simulations to demonstrate the capabilities of this s-eBGP. The "EBGPRouter" is a class to represent an external BGP router in an AS, there are numerous methods that will be discussed but this is a foundational class used in our implementation. The "SecureEBGP" router inherits from the regular "EBGPRouter". We add functionality to the secure router to enhance its ability to provide the security measures needed (i.e. protection against route hijacking and false prefix announcements). Furthermore this added functionality is directly related to the "Blockchain" which is inherently composed of the "Block" class. The specifics of the design will be further described in the following sections.

5.1 Design

The "EBGPRouter" class has the following attributes: self.ip, self.as_number , self.port, self.neighbors, self.routes, and self.routing_table. The self.ip is the given ip address of this BGP router. The self.as_number is the assigned Autonomous System number to which this router belongs. The self.port is the port that this router will be communicating with other routers in our simulations. Self.neighbors is the neighbors of the AS in which this router belongs. Self.routes is a list of all the routes

this router has advertised. self.routing_tables is the routing table for this router which is a mapping of prefixes to paths.

The “SecureEBGP” inherits all attributes from “EBGPRouter” and in addition has the following attribute: self.blockchain. This attribute is of type Blockchain and is unique and shared among all SecureEBGP routers in a system. It is the main repository for the security functions of our system.

The “Block” class is a simple class that represents a block in our blockchain. It has the following instance attributes: self.index, self.previous_hash, self.timestamp, self.data, self.nonce and self.hash. The self.index attribute is the index in which this block resides in the blockchain. self.previous hash is the hash of the previous block likewise self.hash is the hash of the current block which is produced by the calculate_hash method. self.timestamp is the time at which the block is created. self.data is the data for a block that is stored. It can be two types of data: Neighbor and Path types. Neighbor type is a list: [“N”, AS#, [neighbors]], where “N” identifies the type to be a neighbor block, AS# is the number of the AS creating the block and [neighbors] is a list of true neighbors of the AS. self.nonce is a random number used to add randomness for the hash values.

The “Blockchain” class is the main storage tool used by our secure routers. It has the following instance attribute: self.chain, which is a list of “Blocks” where the leftmost index is the “Genesis Block” and the rightmost index is the most recently added Block.

5.2 Algorithm

The verify path algorithm pseudocode will be presented here for clear understanding of the implementation:

Algorithm 1 verify_path(path)

```
1:  function VERIFY_PATH(path)
2:      for (asx, asy) in path do
3:          (asx_secure, asy_is_neighbor) ← CONFIRM_NEIGHBOR(asx, asy)
4:          if asy_is_neighbor then
5:              Continue
6:          end if
7:          if asx_secure and not asy_is_neighbor then
8:              return False
9:          else
10:             (asy_secure, asx_is_neighbor) ← CONFIRM_NEIGHBOR(asy, asx)
11:             if asx_is_neighbor then
12:                 Continue
13:             else
14:                 return False
15:             end if
16:         end if
17:     end for
18:     return True
19: end function
```

Algorithm 2 confirm_neighbor(asx, asy)

```
function CONFIRM_NEIGHBOR(asx, asy)
    for block in blockchain do
        if block is not a neighbor block then
            Continue
        end if
        if asx wrote the block then
            for neighbor in block_neighbors do
                if neighbor == asy then
                    return True, True
                end if
            end for
            return True, False
        end if
    end for
    return False, False
end function
```

5.3 Implementation

EBGPRouter

The "EBGPRouter" class has the following methods: add_neighbors(self, neighbor_list: list), add_neighbor(self, neighbor_ip, neighbor_as, neighbor_port), advertise_route(self, route), send_route(self, route, neighbor_as, source_ip, as_path), routing_decision(self, route_info, route), log_advertisment(self, route, as_path), reconstruct(self, prefix), handle_client(self, connection, address), listen_for_routes(self), start(self), and _advanced_feature(self, keyword: str, json_data: str). A brief description of each method proceeds.

- add_neighbors: takes in a list of neighbors and adds them to its self.neighbors attribute.
    - This method uses the add_neighbor method so the list input must be a neighbor_list that coincides with add_neighbor parameters.
    - It also calls our advanced feature that is specifically for secure routers, so it will just pass in "EBGPRouter" classes but perform another function in "SecureEBGP".
- add_neighbor: takes a neighbors ip, as number and port and adds an entry in it self.neighbors dictionary.
- advertise_route(self, route): This method takes a route/prefix input (e.g. "127.1.0.0/24) and will advertise to its neighbors that it has access to this route/prefix.
- send_route: This is called by advertise_route when it is time to propagate the route to its neighbors. It establishes a connection with each of its neighbors' ports and sends the route.
- routing_decision: Updates its routing table if the received route is shorter than the current one for that prefix/route received.
- log_advertisement: Logs information related to an advertisement in a file called chain.log.
- reconstruct: searches through the chain.log to reconstruct the history related to a given prefix and prints the history.

- handle_client: This is called when another router advertises a route to this router. It takes in a connection and an address and receives the data, handles the received data and propagates the route to all its neighbors (avoiding loops) with its AS appended to the received path.
- listen_for_routes: Binds its port to accept connections and creates a thread to handle the connection/client (by calling handle_client).
- start: Starts the router to begin listening for routes and connections.
- _advanced_feature: this is a function that is called in add_neighbors but has no effect in the "EBGPRouter" and is used for "SecureEBGP".

SecureEBGP

"SecureEBGP" has the following methods: write_to_blockchain(self, data), search_for_block(self, block_data), verify_path(self, path: list), confirm_neighbor(self, asx, asy), _add_as_neighbors(self, data_dict: str), add_path_to_chain(self, data_dict: dict), find_origin(self, prefix: str), routing_decision(self, route_info, route), _advanced_feature(self, keyword: str, json_data: str). Explanation of each method proceeds.

- write_to_blockchain: take input of data and will create a block of class type "Block" with the corresponding data, then it will get the latest block on the blockchain and call a blockchain method to add the block to the chain.
- search_for_block: Searches the blockchain for a given data value and returns True if it is found.
- verify_path: This is an all or nothing verification of a given path through the blockchain. It uses neighbor blocks written to the chain and forms a complete path with secure routers every other hop along the path. If this can be done then the verification of the path's integrity can be ensured. Further explanation of the algorithm is in the proceeding section.
- confirm_neighbor: Takes two AS numbers "asx", "asy", and verifies if the first asx is a secure router. Then it also verifies whether asx has asy as a neighbor. It returns 2 True or False statements with the first being whether asx is secure or not. This is mainly used to verify paths.
- _add_as_neighbors: Will form data out of a data dictionary string (convenient for receiving json data) and will call write_block_to_blockchain with the data formed as an argument. This is called for neighbor block types (blocks whose data is the neighbors of an AS).
- add_path_to_chain: Forms data out of a dictionary where the data will represent a path block, containing the prefix, source as number and the as path. This data is then passed as an input to write_to_blockchain if it is a legal path, if it can be verified by the verify_path function and the block is not present in the blockchain already.
- find_origin: Given a prefix as input, this method returns the first block that announces a prefix in the blockchain (This implies they own it).
- routing_decision: This method will override the method from the parent class. It attempts to find an origin for a given prefix/route (argument), then it gets the as_path from the route_info (argument) and compares the two. If they are equal then the standard routing_decision from the parent is called with arguments of route_info and route.
- _advanced_feature: Will call the appropriate method of _add_as_neighbors or add_path_to_chain based on the argument keyword. If the keyword is "neighbor announcement" the _add_as_neighbors method is called on the json_data. If the keyword is "route advertisement" the

method add_path_to_chain is called. If the keyword is “route update” it will first load the data then set the source AS number to be the source of the prefix produced by find_origin. Then this path is added to the chain.

Block

The “Block” class has the following method: calculate_hash(self). Description of this method proceeds. calculate_hash: will use the index, timestamp, data and nonce attributes as inputs to a SHA256 hash function and produce a unique hash that cannot be reproduced.

Blockchain

The blockchain used inspiration from “Building a Blockchain from Scratch with Python!”[2]. The “Blockchain” has the following methods: create_genesis_block(self), get_latest_block(self), add_block(self, new_block) and is_valid(self). The methods are explained:

- create_genesis_block: Creates an initial “Genesis Block” to initialize the blockchain.
- get_latest_block: Returns the block that is at the end of the chain or the most recently appended block to the chain.
- add_block(self, new_block): Takes input of a Block and appends it to the Blockchain.
- is_valid: verifies the integrity of the Blockchain by checking the hash values of each block and the previous_hash values.

6. Evaluation Methodology

The primary mode of collecting data we employed was setting up network topographies with strategically placed s-eBGP routers and simulating the propagation of route announcements in that given topography. Our criteria is thus: if s-eBGP routers are able to detect a malicious announcement, it will be logged in the blockchain and dropped, thereby confirming the accuracy of detection and the ability to make informed routing decisions. If this chain of events occurs, then the simulation result emulates the desired behavior.

7. Results and Analysis

The results of the proposed protocol suggest that if the following premises are true:

1. S-eBGP routers share their neighbors to a communal ledger.
2. All neighborhoods in the communal ledger are correct and complete.
3. ASes that are the first to announce a prefix ‘p’ to the communal ledger must own it.

Then, ASes using the proposed protocol will gain protection against origin and prefix hijacking by consulting the communal ledger to detect discrepancies in illegal announcements. Moreover, ASes that

are not using the proposed protocol can still benefit from the security given by the proposed protocol by being strategically placed relative to S-eBGP routers.

Protection Against Origin Hijacking for S-eBGP Routers:

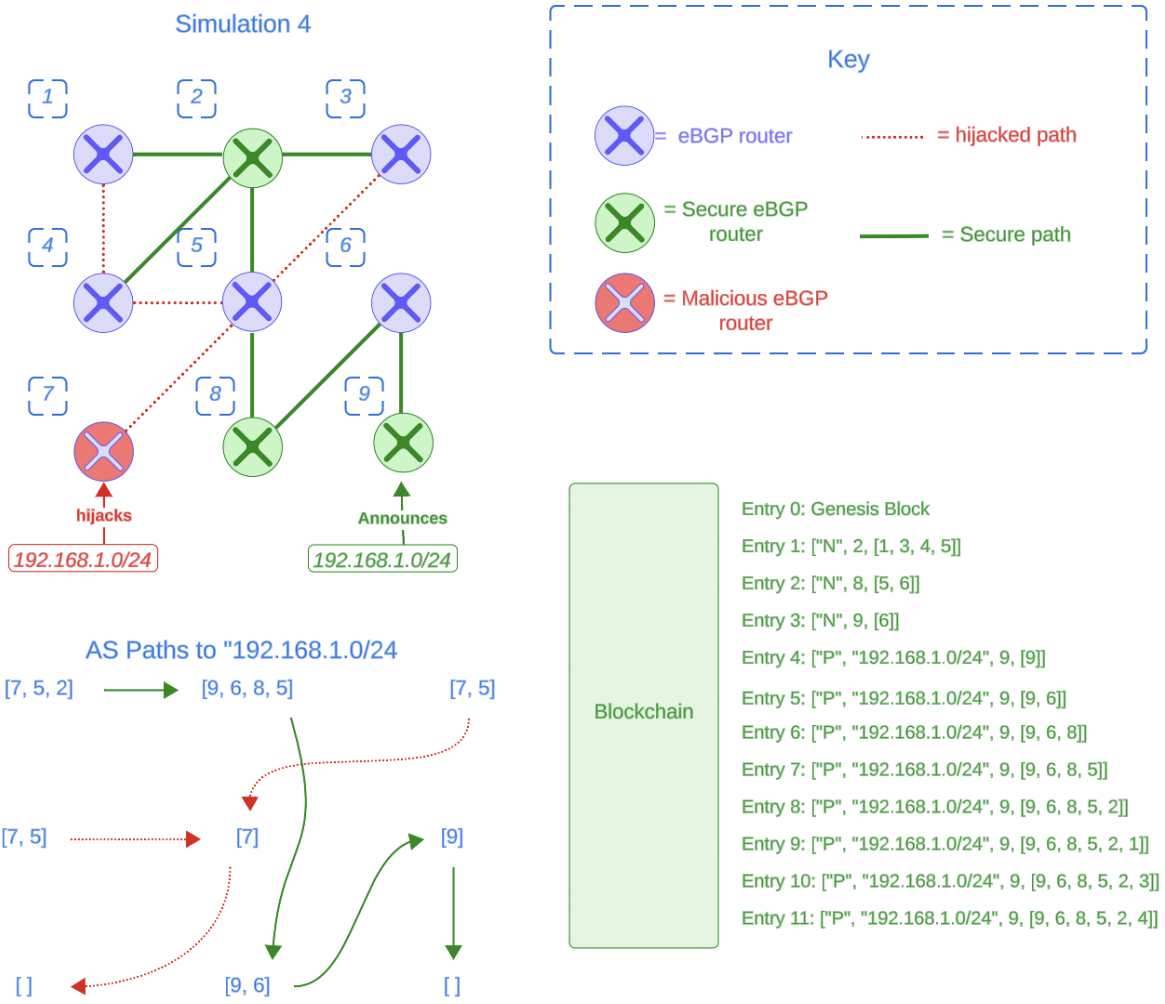
Using the communal ledger, an S-eBGP router can validate if a received BGP origin announcement is correct. Upon hearing the BGP origin announcement, the S-eBGP router locates the first entry referencing that prefix within the communal ledger. By premise three, the first entry referencing a prefix ‘p’ corresponds to the owner of that prefix. Therefore, the S-eBGP router compares the claimed owner of the prefix ‘p’ in the communal ledger with the claimed owner of the prefix ‘p’ in the BGP prefix announcement. For the origin announcement to be valid, the owner of the prefix ‘p’ must match both the communal ledger and the BGP announcement. If the origin announcement is valid, then it will be accepted by the S-eBGP router. Otherwise, the origin announcement is ignored.

Protection from Prefix Hijacking for S-eBGP Routers:

Using the communal ledger, an S-eBGP router can validate if a received BGP prefix announcement is correct. Upon hearing the BGP prefix announcement, the S-eBGP router consults the communal ledger to reconstruct the AS path. Premises one and two states that the communal ledger contains the correct and complete neighborhoods of each S-eBGP router. For the prefix announcement to be valid, every consecutive pair of ASes in the AS path must be neighbors in the communal ledger. If the prefix announcement is valid, then it will be accepted by the S-eBGP router. Otherwise, the prefix announcement is ignored.

Protection Against Origin and Prefix Hijacking:

Simulation 4 shows how a network of S-eBGP routers can maintain the integrity of a route “192.168.1.0/24”. First, all of the S-eBGP routers announce who their neighbors are (entries 1-3). Then, AS 9 announces the prefix “192.168.1.0/24” and writes to the blockchain (entry 4). This prefix is propagated throughout the network, and S-eBGP routers write the following to the blockchain: 1. who they received the route from, and 2. who they are sending the path to. The full propagation history is illustrated in entries 1-11 in the blockchain. When AS 7 attempts to hijack the prefix “192.168.1.0/24”, it compromises the non-secure paths. However, all S-eBGP routers were able to detect the discrepancy in the announcement by seeing that the fraudulent message advertised the wrong origin. Therefore, the S-eBGP routers successfully ignored the fraudulent message and kept the correct routing tables (shown in the diagram under AS Paths to “192.168.1.0/24”). Notice that AS 1 contains the wrong route to 192.168.1.0/24 in its routing table, but since AS 1 routes through S-eBGP 2, any messages sent to 192.168.1.0/24 will make it to AS 9. Therefore, this simulation illustrates how S-eBGP routers can protect themselves from fraudulent BGP messages, and provide some security to their neighbors.

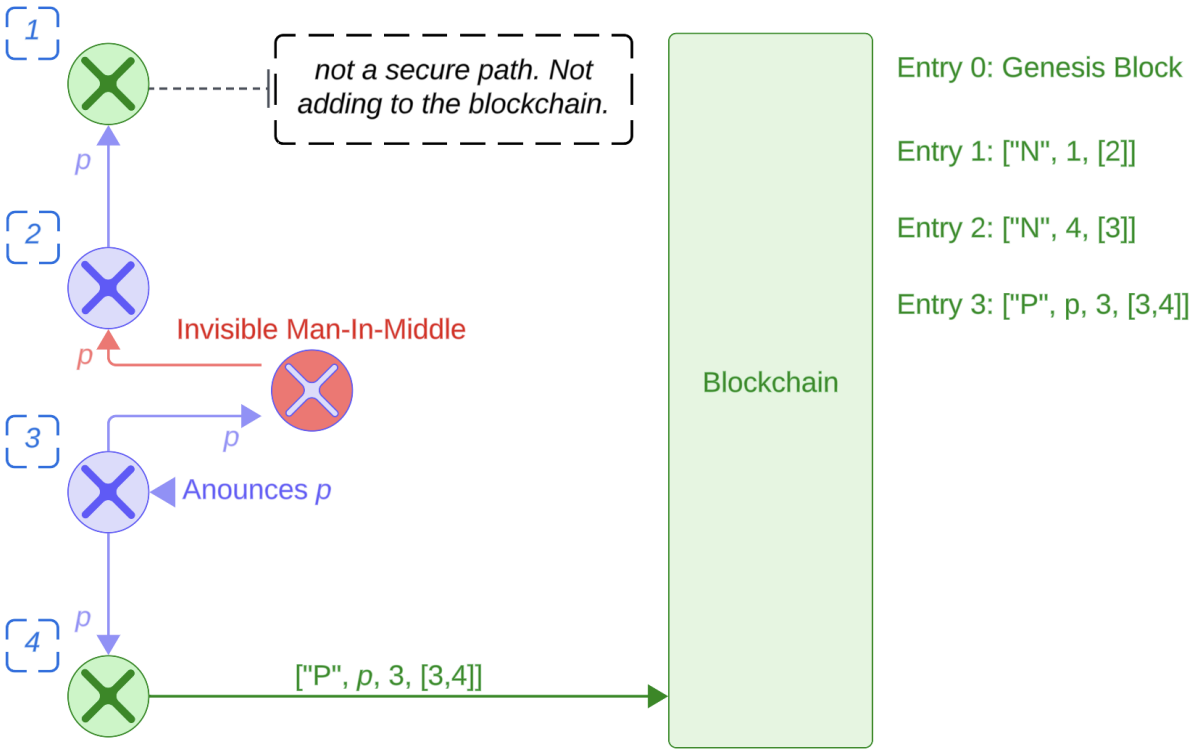


Protection from Outside Threats for Normal eBGP Routers:

Normal eBGP routers that do not support the proposed protocol can still benefit from its security by being strategically placed relative to S-eBGP routers. Premises one and two state that the communal ledger contains the correct and complete neighborhoods of each S-eBGP router. Therefore, all valid eBGP routers (regardless of design) that are one hop away from an S-eBGP router will be documented in the communal ledger as a neighbor. As such, normal eBGP routers that are one hop away from an S-eBGP router can indirectly add valid prefix announcements to the communal ledger when every consecutive AS pair in the AS path corresponds to a neighbor in the communal ledger. Therefore, a normal eBGP router can propagate its information into the communal ledger so that S-eBGP routers can protect its integrity.

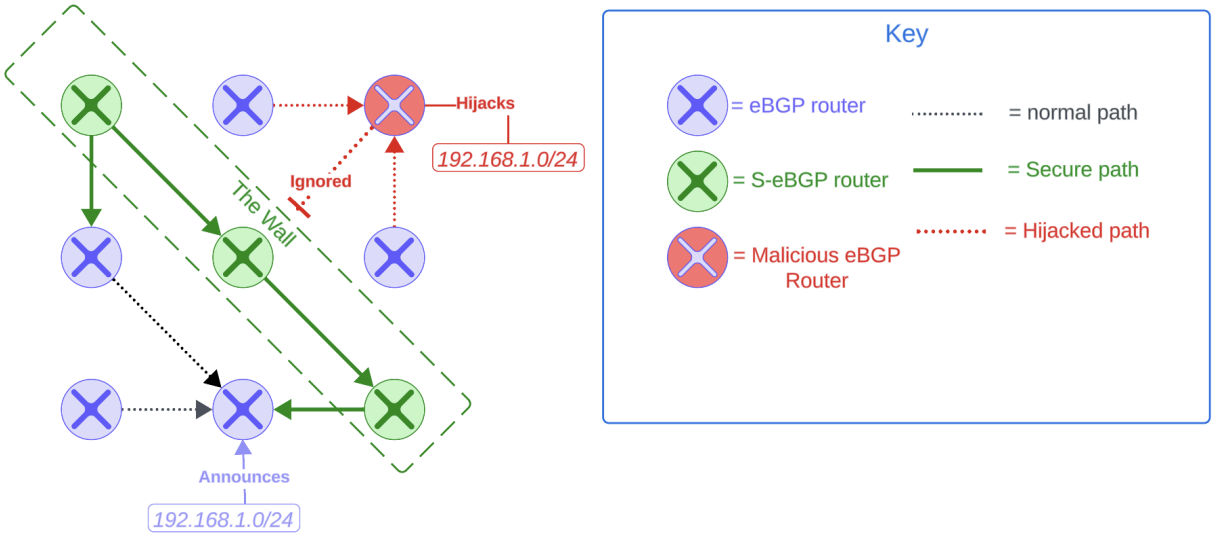
Geographic Dependency:

The following diagram illustrates why two-hop paths that do not contain a S-eBGP router cannot be trusted. If there are any two normal eBGP routers along a path, then the integrity of that path cannot be guaranteed.



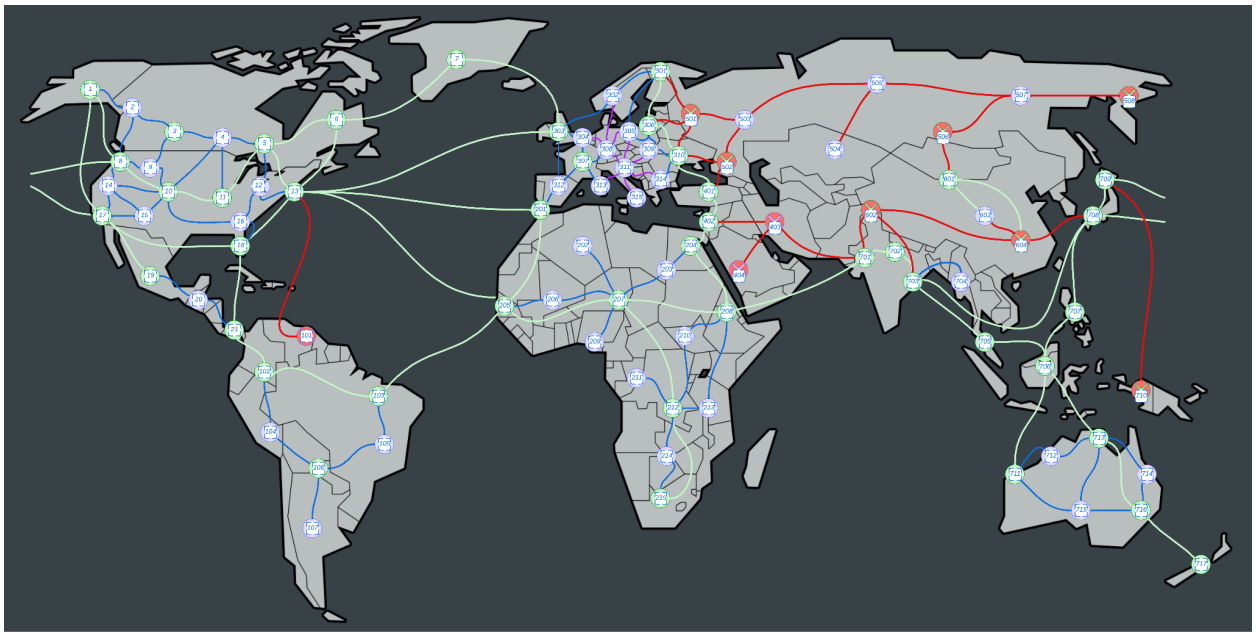
The Implication of Walls:

S-eBGP routers spaced one hop away from each other can act as a “wall” that prevents fraudulent messages from crossing. Therefore, the strategic placement of S-eBGP can improve the control of information. For example, walls of S-eBGP routers can protect the soft interior of entire nations which allows for a ridiculously low required adoption rate in comparison to other existing solutions. However, a limitation of this design is that it results in some positions holding more value than others. For instance, any fault in “the wall” could allow fraudulent messages to penetrate into the soft interior. In this example, the S-eBGP routers in the wall contribute more value to the network's security than any S-eBGP routers placed behind the wall.



Strategic Placement of S-eGBP Routers:

The following diagram comes from simulation 6 in our project. The simulation plays on the idea of eBGP routers acting as a part of geopolitics. Furthermore, the simulation suggests the idea of only fortifying strategic regions with S-eBGP routers to save on resources. To defend from outside threats, there are ‘walls’ of S-eBGP routers around most nations/regions. For example, malicious routers in Russia are prevented from hijacking paths in Europe by the S-eBGP wall across Eastern Europe. This design implies that the strategic placement of S-eBGP routers can completely secure an entire network.



8. Issues

The issues with our implementation of a secure blockchain solution are mainly composed of the strong assumptions made, first to announce a prefix owns it, performance/speed and blockchain controls. For the strong assumptions we first have the assumption that all secure routers announce their neighbors upon joining our S-eBGP. Second we have that all announced neighbors upon becoming a secure router are correct and real neighbors based on the true network topology. In addition we have the property that the first AS to announce ownership of a prefix to the blockchain effectively owns the prefix. This leads to issues of having authentic prefix announcements and opens up vulnerabilities of AS claiming prefixes that they do not truly own. RPKI could be a solution to this issue to prevent ASes from claiming non verified prefixes in the blockchain. Performance and speed may also be a limitation, as the blockchain increases in complexity and size the algorithms to verify a path may become time consuming and inefficient. Finally we don't have blockchain control measures, so the blockchain is effectively immutable and this leads to issues with a realistic network that often dynamically changes its topology. If neighbors change in the topology it would require a change in the blockchain which is currently not implemented.

9.Conclusion

As we discussed in the introduction, BGP security is a field that has faced significant obstacles to efficient implementation and adoption of best practices. Due to the interconnected nature of the internet, total implementation of a given security measure is difficult to achieve without everyone adopting it. Additionally, the nearsighted nature of BGP prevents good, informed security decisions, keeping each AS in a "fog of war" that blinds it to its surroundings.Therefore, the need for a BGP security solution that takes the issue of adoption into account and at the same time improves the functionality of BGP routers is required. With s-eBGP – barring some limitations – both of these requirements are met, and as has been shown, s-eBGP is efficient when strategically placed and can provide security measures even to routers that do not implement any security protocol. This could have strong implications for the field of BGP security, as this approach could be interwoven with other security solutions like RPKI to further enhance efficiency. However, one important note is that this solution is purely Proof-of-Concept (PoC). A PoC solution is far from being scalable and applicable to the real world, and besides the assumptions we make such as the verification of neighbors and a static internet, problems could arise with such an effort. The majority of this project was completed in Python 3.9, while a real-world implementation would require building off a blockchain network such as Ethereum and using specific programming languages such as Solidity. To conclude, we posit that s-eBGP is a strong step towards a holistic BGP security solution, but faces issues of scalability and requires more development.

10. References

[1] Bush, Randy, and Rob Austein. "RFC 6810: The Resource Public Key Infrastructure (RPKI) to Router Protocol." *IETF Datatracker*, datatracker.ietf.org/doc/html/rfc6810. Accessed 16 Mar. 2024.

[2] Lepinski, Matt, and Kotikalapudi Sriram. "RFC 8205: BGPSEC Protocol Specification." *IETF Datatracker*, datatracker.ietf.org/doc/html/rfc8205. Accessed 16 Mar. 2024.

[3] "The Federal Register." *Federal Register :: Secure Internet Routing*, www.federalregister.gov/documents/2022/03/11/2022-05121/secure-internet-routing. Accessed 16 Mar. 2024.

[4] Morris, Cameron & Herzberg, Amir & Secondo, Samuel. (2023). BGP-iSec: Improved Security of Internet Routing Against Post-ROV Attacks. 10.14722/ndss.2024.241035.

[5] K. F. Awe, Y. Malik, P. Zavarsky, and F. Jaafar, "Validating BGP Update Using Blockchain-Based Infrastructure," in Decentralised Internet of Things, vol. 71, M. A. Khan, M. T. Quasim, F. Algarni, and A. Alharthi, Eds., in Studies in Big Data, vol. 71. , Cham: Springer International Publishing, 2020, pp. 151–165. doi: 10.1007/978-3-030-38677-1_7.

[6] J. Li *et al.*, "DeBGP: Decentralized and Efficient BGP Hijacking Prevention System". zilimeng.com/papers/debgp-icccn23.pdf.

[7] Saad, Muhammad, et al. “Routechain: Towards blockchain-based secure and efficient BGP routing.” *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, May 2019, https://doi.org/10.1109/bloc.2019.8751229.

[8] Islam, Aarafat. “Building a Blockchain from Scratch with Python!” Medium, The Pythoneers, 17 Feb. 2023, medium.com/pythoneers/building-a-blockchain-from-scratch-with-python-489e7116142e.