Networked Systems
ETH Zürich — seit 2015

Stealthier BGP Hijacking Attacks

Semester Thesis

Author: Dominique Portenier

Tutor: Theo von Arx, Dr. Muoi Tran, Dr. Stefano Vissicchio (UCL)

Supervisor: Prof. Dr. Laurent Vanbever

October 2023 to February 2024

Abstract

Artemis is the state-of-the-art defensive system to detect BGP hijacks. This thesis analyzes Artemis' implementation and design and provides a proof of concept attack model. Artemis uses historical data to check if a suddenly appearing link was already used in the past to reduce false positives. We show, that an attacker can search for attack paths in the publicly available data and perform a BGP hijack without being detected by Artemis. This shows that although Artemis reduces the impact of a BGP hijack drastically, it is still possible to operate a hijack in a completely stealthy way.

Contents

Chapter 1

Introduction

This chapter will give a motivation for this thesis, present the milestones and end with an overview of this report.

1.1 Motivation

BGP hijacks are a well-known internet security vulnerability. The lack of cryptographic mechanisms in the BGP protocol has led to many attacks in the form of traffic interception, traffic manipulation and denial of service (DoS) by blackholing. While many improvements to the BGP protocol were proposed, they lack adaption in the real world and are often useless if not everybody implements them.

Artemis [23] is the state-of-the-art BGP hijack detection system. Artemis uses BGP monitors such as RIPE-RIS and Routeview to detect hijacks, which typically leave traces on the control plane.

This thesis seeks to analyze Artemis in-depth and find its weaknesses.

1.2 Task and Goals

As a starting point, the Artemis paper has to be analyzed in depth to classify what parts might be vulnerable on the theoretical side. Next, the open-source implementation has to be examined to find weaknesses there. Based on the analysis of this, attacks should be discussed and simulated.

1.3 Overview

Section 2 gives the required background about BGP and Artemis. Section 3 presents the findings from reverse engineering Artemis. The Type-N/E detection and how to attack it is shown in section 4, while in section 5, the results are evaluated. We conclude in section 6 where an outlook and a summary of this thesis are given, respectively.

Chapter 2

Background

This section presents some background about BGP, BGP Hijakcs and Artemis.

2.1 BGP

The *Border Gateway Protocol (BGP)* is used to communicate routing and reachability information among *autonomous systems (ASes)*. It was first designed in 1989 in RFC 1105 without any security thoughts in mind. BGP is a path-vector routing protocol that communicates known paths with its neighbours. The current version of BGP, BGP4, still lacks cryptographic integrity.

While BGP4 has multiple message types, we will only focus on the *update announcement* message, which contains, among others, the following fields: timestamp, type, path, and prefixes. Table 2.1 shows the most important fields with exemplary values. Live BGP messages can be seen on the RIS Live Project [22]. It is important to note that the path reads from end to beginning, which means that AS40288 owns the prefix 204.152.0.0/24. Indeed, we can verify this using bgp.tools [6].

Field	Value
timestamp	1707291925.21
type	UPDATE
path	[24482, 2914, 701, 40288]
prefixes	["204.152.0.0/24"]

Table 2.1: Fields in a BGP update Announcement with exemplary values

2.2 BGP Hijacks

Lacking cryptographic integrity, any AS can craft arbitrary messages and try to hijack foreign prefixes. Several cases of BGP hijacks are known ([5],[10],[11]). We will use the taxonomy used in the Artemis paper [23] to classify hijacks.

Classification by Path

Assuming a BGP announcement, we classify the type by the position the hijacker is in the path. If the hijacker is in the first position, i.e., the hijacker announces a prefix that he doesn't own, it is called a type-0 or Origin-AS hijack. A hijacker who puts himself in the 1st hop, i.e. fakes the first link, is called a type-1 hijack.

Given the Internet topology in figure 2.1, when AS8 announces the prefix 1.1.0.0/16 with path AS8, this is a type-0 hijack. An example of a type-1 hijack launched by AS6 would be to announce the prefix 1.1.0.0/16 with the path "AS6 AS1". And lastly, AS6 can also generate a type-3 hijack if he announces the prefix 1.1.0.0/16 with the path "AS6 AS3 AS2 AS1". The first three ASes are correct, and only the link between the third and fourth ASN in the path is spoofed, therefore this is a type-3 hijack.

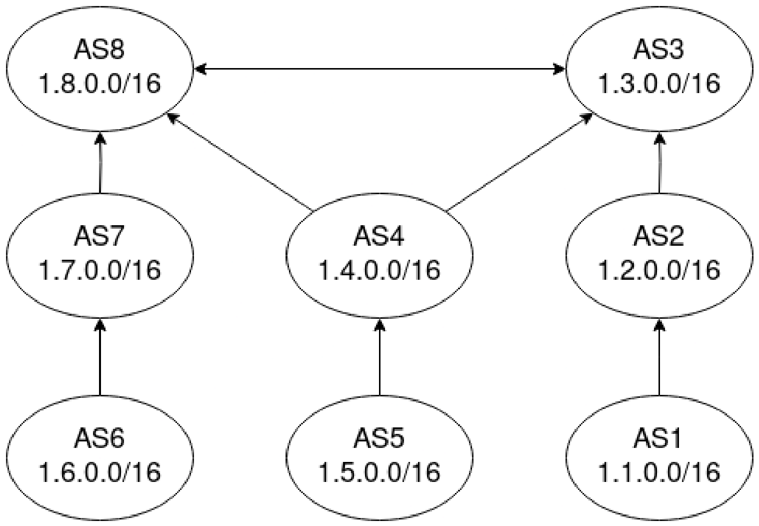

Figure 2.1: Example configuration to show different attack scenarios. Every node represents an AS with its ASN and the prefixes the AS owns.

This thesis will mostly focus on type-N, where $N \geq 2$ hijacks, as we'll see later, where the hijacker is in the N-th position of the path.

Classification by Affected Prefix

If the prefix in the hijacker's announcement is the same as the legitimate, it is an *exact prefix hijack* (type-E). If the hijacker announces a sub-prefix, we speak of a sub-prefix attack. The last case, where a victim AS owns a prefix it doesn't currently announce, but a hijacker does, is called squatting.

For example, if AS1 only announces 1.1.0.0/17, although owning the prefix 1.1.0.0/16, a hijacker can launch a squatting attack by announcing, e.g. 1.1.128.0/19.

Classification by Data-Plane

During an ongoing hijack, the data plane is affected as at least a fraction of packages get sent to the hijacker, not the prefix's legitimate owner. The hijacker can do three things. He can drop the packages (*blackholing, BH*), he can manipulate or eavesdrop on the packages (*man-in-the-middle, MITM*), or he can impersonate the victim's services (*imposture, IM*). While BH is rather obvious, as services cannot be reached any more, IM and especially MITM can be invisible to the victim.

2.3 BGP Hijack Defences

Additional defences were introduced to make BGP more reliable, although it lacked cryptographic integrity.

2.3.1 Ressource Public Key Infrastructure (RPKI)

With RPKI, which is specified in RFC 6483 [17], prefixes get cryptographically bound to an *Autonomous System Number (ASN)*. So if an AS announces a prefix it doesn't own, other ASes can easily detect this and drop the announcements. The downside of this system is that type-1 and type-N hijacks are still perfectly possible as the origin of the path still matches the announced prefix.

2.3.2 Artemis

The state-of-the-art defensive system Artemis is based on the fact that every attack leaves some data on the control plane and uses them to detect hijacks automatically and can also be configured to mitigate detected hijacks automatically.

Out of the Artemis paper, a start-up called CodeBGP [1] was founded and sold to Cisco in Summer 2023. The open-source implementation is still accessible on Github [2] but lacks some of the key features, such as type-N detection.

How to run Artemis is clearly described in their documentaiton [3]. It is straightforward to run Artemis locally.

Functionallities

Artemis uses different data sources to get BGP updates and check for hijacks against the AS operating Artemis. The two most used sources are BGP update messages from BGP monitors, i.e. RIPE RIS [22] and RouteViews [21], and the BGP update messages from the ASes boarder-routers ingested into Artemis via exaBGP.

Artemis references a yaml-config file as a source of truth. This config defines the ASN running Artemis, the prefix they own and announce, as well as their direct neighbours. The reasoning is that the AS operating Artemis also knows these things, and Artemis can then use it to validate the messages found on the monitors and validate them against the config file. This is used to check for type-0/1E hijacks, subprefix, and squatting hijacks, as table 2.2 shows.

Prefix	AS-Path	Detection Rule
Sub-prefix	*	Config vs. BGP updates
Squatting	*	Config vs. BGP updates
Exact	0/1	Config vs. BGP updates
Exact	≥ 2	Past Data vs. BGP updates (bidirectional link)

Table 2.2: Detection of different BGP Hijacks.

As table 2.2 shows, only type-N/E, $N \geq 2$ hijackas are validated using historical data. As the AS operators maintain the configuration file, it contains, unless a misconfiguration happens, the correct state of the AS. Therefore, this thesis will focus on how to craft type-N/E hijacks, which are stealthy against Artemis. Section 4.1 will introduce and analyze the algorithm for type-N/E detection in-depth.

2.4 BGP Monitors

BGP Monitors publish live and historical data of BGP messages and Routing Tables from different locations around the world. Two projects worth mentioning are RouteViews [21] and RIS Live [22]. These live feeds can be used as a looking-glass mechanism to see your ASes BGP Messages at a different location. The live stream feature enables Artemis to grab all these messages and search for BGP Hijacks in the control plane.

RIS has, at the time of this thesis, 24 monitors online. They show every peer on their website [19]. RouteViews, on the other hand, has 46 Monitors online [20].

2.5 Notation

Throughout this report, we will use Alice and Mallory for attack modelling: Alice is the victim, which operates an AS and runs Artemis to defend it, while Mallory tries to hijack traffic intended for Alice without Alice noticing it.

Chapter 3

Reverse Engineering Artemis

As this thesis aims to find weaknesses in Artemis and ways to attack it, an effort was put into reverse engineering the codebase. This chapter will present the findings of this analysis and what tools were used. We'll start in section 3.1 with the relevant parts of a C4 Software Architecture analysis [7]. Section 3.2 will then present the findings and their implication.

3.1 Architecture

We follow a systematic approach to software analysis by doing a C4 analysis. We'll go into the context, container and component view. Additionally, we show the connectors.

3.1.1 Context View

The context view is shown in figure 3.1. A config file is needed to specify the used BGP update stream and the ASN and prefixes Artemis should look for. Artemis checks every BGP update message to see if it contains a hijack and displays that on a dashboard.

3.1.2 Containers

Artemis' Github repo contains a representation of the containers under docs/images/arch_containers.jpg [8]. Unfortunately, the resolution isn't good enough to read the labels. Therefore, we start by getting a first dependency graph. We illustrate the docker-compose file using Docker Compose Viz [9] and get figure 3.2.

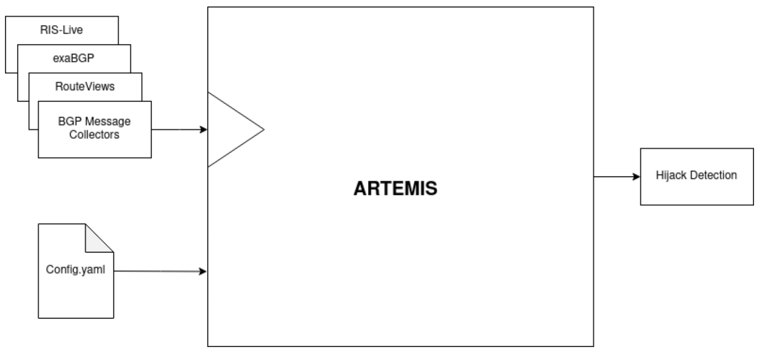

Figure 3.1: Context View of Artemis.

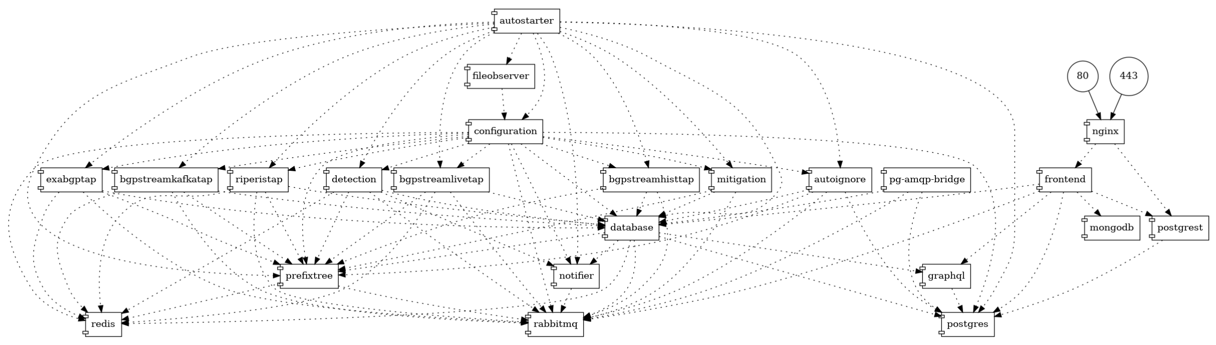

Figure 3.2: Visualization of the Docker Compose to get a first impression of the services. The arrows only show their startup dependencies, not the dependencies at runtime.

We split the components into third-party and Artemis components. We also present state-of-the-art tools for the 3rd party containers to inspect their state at runtime. The description of Artemis' containers results from code inspection.

3rd Party Containers

In fig. 3.2, we see some familiar names such as Postgres and Redis. These are third-party components, which we will explain here.

Redis Redis is an in-memory key-value store for distributed systems [15]. The software RedisInsight [16] can be used to inspect the store at runtime.

RabbitMQ RabbitMQ is a lightweight message broker for distributed systems. In Artemis' case, it is the main communication channel to pass the BGP updates around the microservices.

Postgres PostgresQL [13] is a relational database. PGAdmin4 [12] is the tool of choice to administrate the database and check what data is stored in the database.

Artemis' Containers

We also provide further insights from the code analysis for Artemis' containers.

Every micro-service consists of a REST-API for control interaction, such as stopping a service and applying a new config. The messaging between the micro-services relies on the RabbitMQ [14] message broker.

Stream Taps There are multiple microservices, which can ingest BGP messages into Artemis. These are called taps, which all work similarly, just with different data sources, e.g. the Hist-tap takes historical data from a CSV file, while the Kafka-tap consumes from a Kafka stream. They get a stream of BGP updates from their source and produce a message in the format as shown in listing 3.1. Type contains either A or W, depending on whether it is an announcement or a withdrawal. The service contains the identifier of the BGP monitor which logged the message. The communities refer to BGP communities found in the message. While the prefix contains the prefix in the message, the peer_asn is the ASN where the message was monitored. In general, peer_asn is equal to the first entry in the path argument. Only messages with a prefix of interest, i.e. the prefixes of the AS operating Artemis, are injected into the system. The rest is dropped.

Listing 3.1: BGP Message Format

```
msg = {
    "type": type_,
    "timestamp": timestamp,
    "path": as_path,
    "service": service,
    "communities": communities,
    "prefix": this_prefix,
    "peer_asn": peer_asn,
}
```

Prefixtree The Prefixtree module mainly works on parsing the configuration and keeping track of which prefixes a hijack detection should be performed. It annotates messages produced by a Tap with information about the prefixes.

Detection This micro-service runs the hijack detection. It checks if a BGP update is a hijack of any type. The open-source implementation of Artemis doesn't provide all the detections described in the paper [23]. Among others, the type-N detection is only given as an empty placeholder.

3.1.3 Component Analysis

Every of Artemis' containers contains a REST API. It serves basic endpoints such as /health and other basic asynchronous interactions. Additionally, the components also contain consumers and producers for rabbitMQ.

3.1.4 Connectors

As part of the study of Artemis, the message flow between the micro-services was analyzed. Appendix A shows two visualizations of the most important links in the Artemis system. We see under which routing key a service publishes messages and what function is configured as a subscriber to a routing key in a certain micro-service.

3.2 Key findings

This section provides some key findings gathered during the reverse engineering phase.

3.2.1 Multiple detection parts not implemented

In the detection module, multiple detection functions are provided but not implemented. One such example is the type-N/E detection given in listing 3.3.

This finding was a major step backwards, as we identified the type-N detection as the most interesting to attack. As a consequence, we implemented it ourselves, as described in section 4.1

Listing 3.2: Type-N/E detection in Artemis open source implementation

```
def detect_path_type_N_hijack(
    self,
    monitor_event: Dict,
    prefix_node: Dict,
    prefix_node_conf: Dict,
    *args,
    **kwargs
) -> Tuple[int, str]:
    # Placeholder for type-N detection (not supported)
    return -1, "-"
```

3.2.2 Loop Cleaning

Before the detection stage processes a BGP message, the AS path is "cleaned". The cleaning function is found under utils/artemis_utils/updates.py [4]. While we didn't have a way to exploit this function, it might be a useful finding for the future. It removes loops from the AS-path field. It might be possible to create BGP updates, which remove a hijacked link before Artemis processes it.

Listing 3.3: Type-N/E detection in Artemis open source implementation

```
def clean_as_path(path: List[int]) -> List[int]:
    """
    Method for loop and prepending removal.
    """
    (clean_path, is_loopy) = __remove_prepending(path)
    if is_loopy:
        clean_path = __clean_loops(clean_path)
    return clean_path
```

Chapter 4

Design

This section will explain in depth the workings of Artmis' Type-N hijack detection and also present pseudo-code. Next, we present our method and how we want to exploit this detection, and lastly, we present how we fix the data sets for the evaluation chapter 5.

4.1 Type-N/E detection

As the type-N/E detection is not available in Artemis' open-source implementation, we will implement it. We analyze the Artemis paper [23], extract the data models we need and present a full pseudo-code.

4.1.1 Detection Methodology

To understand the algorithms presented in section 4.1, we need to understand how and when Artemis classifies a BGP update as a hijack. So, assume a BGP announcement containing a prefix for which Artemis is configured to run the hijack detection enters the system. This announcement contains the AS-path P. P has n hops and therefore $n - 1$ links. As the type-N/E detection is for $N \geq 2$, we assume that P contains at least two links (or three ASN). Now, for every link after the first link (the type-0/1 hijack detection covers the first link), we check if Artemis has already verified the link in the past. If so, we continue with the next link. If not, two rules are applied to path P. If both are evaluated to be true, the link gets added to the verified link list, and the next link of P is checked. The announcement is marked as a type-N/E hijack if one of the two rules is unmet.

Rule 1 – Bidirectionality

Given the link (AS_X, AS_Y), we check if the reverse link (AS_Y, AS_X) was monitored towards any prefix. Normal links are used in both directions. If a link is only used in one direction, it is likely to be a hijack. Therefore, AS_X will be blamed as a hijacker, as he announced a link nobody else has used/seen before.

Rule 2 – Left AS Intersection

This rule is only evaluated if rule 1 passes for the link (AS_X, AS_Y) of path P. This rule checks if a single AS create the link (AS_X, AS_Y) in both directions. If so, the AS which did so will be blamed for being a hijacker. We will first present an example and then build the math for the rule.

Example We refer to the AS topology presented in figure 4.1. AS1 is the victim, running Artemis, while AS666 will try to create a fake link to hijack traffic. Every BGP message monitored at any AS is fed into Artemis. AS666 wants to announce the path "AS666 AS5 AS2 AS1". This would allow AS666 to attract traffic from AS6, as the path provided is shorter than via "AS5 AS4 AS3 AS2 AS1". To prevent rule 1 from triggering (the link AS5-AS2 would only show up in one direction), AS666 crafts a second BGP announcement containing the path "AS666 AS2 AS5" and AS5's prefix. Artemis doesn't check this path, as it doesn't contain AS1's prefix. If AS666 now runs his hijack, Artemis' rule 1 doesn't trigger, as the (fake) link (AS_5, AS_2) exists in both directions.

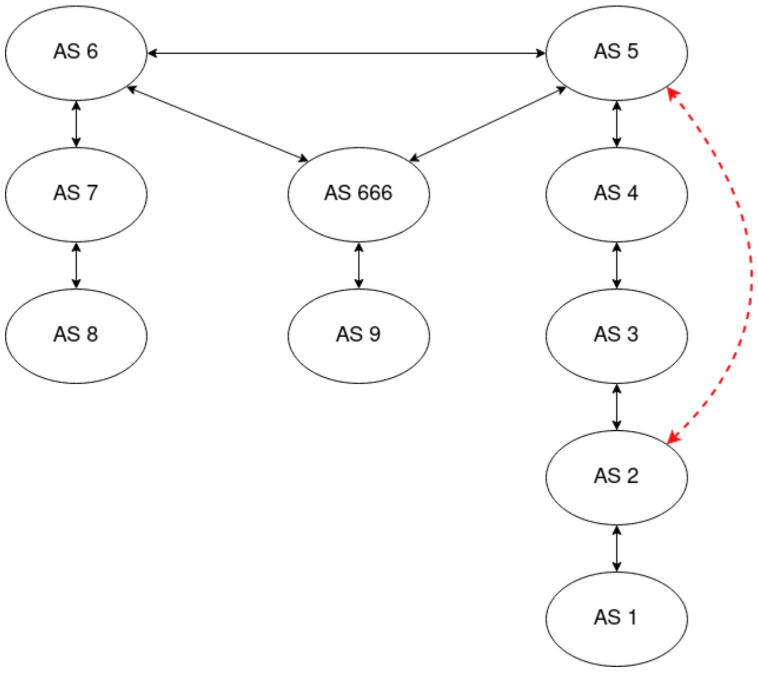

Figure 4.1: Example Attack. AS666 tries to hijack AS6 by using the red link to create a shorter path to AS1 than AS5 can.

Artemis can detect this as a hijack nevertheless by doing what we call *left AS intersection*. Say AS666 propagated the path "AS666 AS2 AS5" to AS6, and AS6 is a monitor, then Artemis took note that the link (AS_2, AS_5) showed up, with AS666 on the left side of the link. Then, when the attack is launched, Artemis checks the link (AS_5, AS_2) in the path "AS6 AS666 AS5 AS2 AS1" and notes that AS5, AS666, and AS6 are on the left side of the given link. Then Artemis takes the intersection set and will note that both AS666 and AS6 show up on the left side of the path, no matter in which direction the link is used. This is suspicious, and Artemis marks that either AS666 or AS6 is a hijacker.

Mathematics We start by defining the L-Set in definition 1.

Definition 1 *Let P be an AS-path with $P = (AS_{l1}, AS_{l2}, ..., AS_X, AS_Y, AS_{R1}, AS_{R2}, ..., AS_V)$ and let the link $l = (AS_X, AS_Y)$ be in P. AS_V is the origin of the BGP update, announcing V's prefix, while AS_{l1} is the peer that monitored this specific BGP update.*

We then call every ASN that is on the left side of the link l the L-Set of P, i.e. $LSet(P,l) = \mathcal{L}_{P,l} = \{AS_{l1}, AS_{l2}, ...\}$.

For every BGP announcement towards any prefix, we extract the L-Set for every link in the path and store every link together with its L-Sets. Note that a link can have multiple L-Sets. If Artemis logs a BGP announcement containing a prefix, Artemis is configured to check for hijacks; it checks every single link one by one. For the link l, we extract the L-Set from this BGP message and take the interset set with all L-Set previously extracted for the link in the reverse direction. If the intersection set off all these L-Sets is the empty set, then rule 2 is satisfied, and no hijack is detected. Otherwise, the ASes in the intersection set will be blamed as hijackers.

Section 4.1.3 will provide complete algorithms.

4.1.2 Raw Data and Data Model

The Artemis paper states three data sources for checking a just-received BGP update towards an owned prefix in section IV.C as follows [23]:

- "*previously verified AS-links list*: all the AS-links that appear in a path towards an owned prefix and have been verified by Artemis in the past."
- "*AS-links list from monitors*: all the AS-links in the AS-path towards any prefix (i.e., owned by any AS) observed by the monitors, in a sliding window of the last 10 months. This list represents a historical view of observed (directed) AS-links. The 10-month time frame should accommodate the observation of most of the backup routes."
- "*AS-links list from local BGP routers*: all the AS-links observed in the BGP messages received by the BGP routers of the network operating Artemis. The list is collected by connecting to the local BGP routers (e.g., via ExaBGP or with BGPStream and BMP and receiving every BGP update seen at them, or alternatively querying a route server. This list is also updated continuously within a 10-month sliding data window."

When starting Artemis, the set of previously verified AS links is empty. The second and third bullet points are equivalent from the algorithmic viewpoint. They only use a different data source.

From the analysis of the algorithms presented by Artemis, we need two different formats of a linked list. The set of *previously verified AS-links list* doesn't have a notion of expiry. If a link is verified, it is verified until a reboot of the system. The *verified link list* can be represented as a set of tuples of all valid AS links. Bullet points two and three from the above list require a different notation, as these links expire after ten months. These links must also store every L-Set as defined in definition 1.

Table 4.1 presents the *extended link list*, which is used to verify links in a later step. Listing 4.1 shows how this *extended link list* can be stored using primitive types.

Field-name	ASN-X	ASN-Y	L-Set	last seen
Type	int	int	Set(int)	timestamp

Table 4.1: Defines the data structure of the extended link list. The first three values taken together are unique, although each two of them on their own might not be. Listing 4.1 shows how the data structure is implemented as a dictionary. The combination ASN-X and ASN-Y will also be referenced by simply calling it a (directed) *link*.

Listing 4.1: Link-List definition in json format

```
[
    {
        "hop": ASN-X,
        "prevHop": ASN-Y,
        "lSets": [
            {
                "asns" : [
                    asn1,
                    asn2,
                    ...
                ],
                "lastSeen": timestamp
            }, {
                ...
            }
        ]
    },
    {
        "hop": Other-ASN-X,
        "prevHop" : Other-ASN-Y,
        ...
    },
    ...
]
```

4.1.3 Algorithms

With the analysis of the detection rules from section 4.1.1 and the analysis of the data models from section 4.1.2, we can present Artemis' type-N/E detection in an algorithmic way.

Algorithm 1 generates the *exteded link list* from the historical data. This algorithm is used when Artemis starts up to create and build the historical view on the Internet. It serves at runtime to verify new links.

Algorithm 2 then checks every BGP update announcement, whether it must be checked for containing hijacks or otherwise is used to update the *extended link list*.

4.2 Attacking Artemis

As we have shown in section 4.1, Artemis uses historical data to analyze if a new upcoming link was already used in the past. We showed in section 4.1.1, how Artemis can detect if a single AS creates new links.

As we didn't find a way around the detection rules, our approach is to use legit but outdated links to create hijacks. For example, assume that the red link in figure 4.1 was used two months ago, but the peering contract of AS2 and AS5 expired. Then, the AS666 can successfully use this link for a hijack; although nobody uses it today, Artemis has verified it in the past.

We first build a model and exploit these historical links to perform a hijack without being detected. Next, we check if one or more ASes exist which are able to use the historical links for a

Algorithm 1 Generate extended AS-link list from BGP updates towards any prefix

```
Require: $BGP_{updates}$ of the last 10 Months
  $M \leftarrow \emptyset$                                         ▷ Empty Extended Link List
  for $m \in BGP_{Updates}$ do
      for $link \in m["path"]$ do                                  ▷ link = (hop, nextHop)
          $lPath \leftarrow LPath(m["path"], link)$
          if $(link, lPath) \notin M$ then
              $M \leftarrow M \cup (link, lPath, m["timestamp"])$
          else
              update timestamp of $M[link, lPath]$
          end if
      end for
  end for
  return $M$
```

hijack and eventually present a way to analyze the effectiveness of such an attack.

4.2.1 Datamodel

We assume that BGP is in a converged state at the time $t = 0$. So extracting the routing tables at $t = 0$ of every router and transforming them in a directed graph $G_t = (V_t, E_t)$, where V_t is the set of vertices, i.e. ASN, and E_t is the set of links that connect two ASNs, gives us the current routing state of the Internet. The suffix t stands for today, as this is the routing state of today's internet. As we do not have access to the routing table of every router, some adaptions are performed:

As a first simplification, we work with an undirected graph, as we will see most links are only used in one direction (towards a monitor), although they are, in general, bidirectional. As a second adaption, we feed in the BGP update messages of a timeslot from minutes to hours previous to $t = 0$. These updates may contain links which do not end up in the routing tables of the monitors but may be used by ASes which are not monitors. This graph G_t gives us now the current state of the Internet.

Next, we want to build an internet graph containing every link that Artemis doesn't detect as a hijack. We start by initializing the historical graph $G_h \leftarrow G_t$ and now add every link to G_h from every BGP update towards any prefix seen in the last ten months. Note that ten Months is the default setting given in the Artemis paper for how long backup links are valid. As ten months of BGP updates is more data than a naive implementation can handle in a reasonable time, a simplification was made just to select snapshots within the last ten months. Section 4.3 will define these.

4.2.2 Finding Attack Vector

Given two nodes, Alice A and Mallory M, where $A, M \in V_t \cap V_h$, we can check if there is a shorter path in the history than is currently used. If there is such a path, the hijacker at M can use this path to attract traffic intended for A. Note that we are interested in the additional ASes we can attract, not in the absolute numbers. If M is the only provider of A, then M has complete control over A without performing any hijack; therefore, this is not an attack against Artemis.

The attack path Mallory can use for stealthy hijacks is the shortest path between Mallory and Alice in G_h. As this path is shorter than the shortest path in G_t, Mallory might be able to attract

Algorithm 2 Type-N detection

```
Require: Historical $BGP_{Updates}$ from Monitors and local BGP-Routers (last 10 Months)
Require: Live-Feed of $BGP_{Updates}$
Require: ownPrefixes $p$
  $M \leftarrow$ Generate extended AS-link list
  $L \leftarrow \emptyset$                                                    ▷ verified Links
  $H \leftarrow \emptyset$                                                    ▷ Hijacks
  for $m \in BGP_{updates}$ do
      if $m[”prefix”] \in ownPrefixes$ then
          for $link \in m[”path”]$ do
              if $link \in L$ then
                  continue
              end if
              if $link^{-1} \notin M$ then                                    ▷ Link not bi-directional
                  $H \leftarrow H \cup link$
              else
                  $\mathcal{L}_{link}^{new} \leftarrow LSet(m[”path”], link)$
                  $\mathcal{L}_P \leftarrow M[link^{-1}][”lSets”]$
                  $\mathcal{L}_{link}^{old} \leftarrow \cap_{l_p \in \mathcal{L}_P} l_p$
                  if $\mathcal{L}_{link}^{new} \cap \mathcal{L}_{link}^{old} \neq \emptyset$ then      ▷ Every update with new link has common AS
                      $H \leftarrow H \cup link$
                  else                                                        ▷ Link is validated
                      $L \leftarrow L \cup link$
                  end if
              end if
          end for
      else                                        ▷ non-owned prefix, update ’historical view’
          for $link \in m[”path”]$ do
              if $(link, lSet) \notin M$ then
                  $M \leftarrow M \cup (link, lSet, m[”timestamp”])$
              else
                  Update timestamp in $M$
              end if
          end for
      end if
      Remove outdated data from M                  ▷ Can be done by a separate worker
  end for
```

traffic from its peers or even further away. In the next section, we will analyze the effectiveness of such an attack.

4.2.3 Analyze Attack Effectiveness

RFC4271 [18], which defines BGP list as the first tiebreaking rule for selecting where to forward an IP packet: "Remove from consideration all routes that are not tied for having the smallest number of AS numbers present in their AS_PATH attributes". Therefore we calculate what path data can take, based on the distance of two nodes.

We use the distance between two nodes to estimate if a route via another node is used. I.e. given three nodes $A, B, M \in V$ we say that there exists a shortest path from A to B via M in Graph G iff $dist(A, B, G) == dist(A, M, G) + dist(M, B, G)$. This might not be the only path and due to BGP policies not even being used. But our model currently only works based on distances, and we say that node B is under the effect of M when communicating to A if the previous equation is true.

For a given A, M pairing, we want to know which nodes are affected in the current state of the internet (G_t) and which nodes can be affected when Mallory uses the shortest route he found in G_h. We can use the above condition. To check if B is under the effect of M without a hijack, we check $dist(A, B, G_t) == dist(A, M, G_t) + dist(M, B, G_t)$. To check if B is under the effect of M when Mallory attacks, we check if $dist(A, B, G_t) \geq dist(A, M, G_h) + dist(M, B, G_t)$. Note that the distance between B and M is checked in today's graph, as Mallory's attack path has to propagate through today's internet.

An additional check has to be done for the nodes on the path between Mallory and Alice in G_h. Assume that node B is a neighbour of Mallory, and B is in the shortest path between Mallory and Alice in G_h. If Mallory tries to announce this shortest path to B, the BGP loop detection will trigger and drop the message. B is not attackable with this path. Therefore, we remove B from V_h and search for the shortest path in this new set. B is only attackable if $dist(A, B, G_t) \geq dist(A, M, G_h \setminus \{B\}) + dist(M, B, G_t)$ is true as well. The complete algorithm for the impact detection is given in algorithm 3.

4.3 Dataset

This section will briefly analyze the amount of data for ten months of BGP updates and show how the data samples were selected for this thesis.

4.3.1 Data Analysis

When visiting RIS Live [22] and selecting all Monitors and no further filtering, we get the data amount of roughly $10'000\ kbit/s \approx 2\ MB/s \approx 7.2\ GB/h \approx 172\ GB/d$. Taking 30 days per month and taking the last ten months, as proposed by Artemis, would yield around 50 TB of data.

While working with such a large dataset is possible, it is not feasible for first tests. Therefore we will take a subset of the data. It has also to be noted that this estimate is only for the RIS-Live project, not RouteViews.

4.3.2 Dataset Selection

To run the attack model described in section 4.2.1 and the type-N/E detection described in section 4.1, one must select a time $t = 0$, which in our case is 1.1.2024, 12 pm (GMT+0). Furthermore,

Algorithm 3 Get nodes affected by Mallory, w/ and w/o running stealthy hijack

```
Require: $G_t = (V_t, E_t), G_h, A, M, A \neq M$
  $\mathbb{R} \leftarrow \emptyset$                          ▷ Nodes affected by Mallory w/o hijack
  $\mathbb{A} \leftarrow \emptyset$                          ▷ Nodes affected by Mallory w/ hijack
  $d_{ref} \leftarrow dist(M, A, G_t) - 1$                 ▷ -1, as Mallory gets otherwise counted 2 times
  $d_{atk} \leftarrow dist(M, A, G_h) - 1$
  for $B \in V_t$ do
    if $dist(B, A, G_t) \geq dist(B, M, G_t) + d_{ref}$ then
      $\mathbb{R} \leftarrow \mathbb{R} \cup \{B\}$
    end if
    if $dist(B, A, G_t) \geq dist(B, M, G_t) + d_{atk}$ then
      $\mathbb{A} \leftarrow \mathbb{A} \cup \{B\}$
    end if
  end for
  for $B \in path(M, A, G_h) \setminus \{M, A\}$ do                ▷ In order from M to A
    if $B \notin \mathbb{A}$ then
      break
    end if
    $G_h \leftarrow G_h \setminus \{B\}$
    if $\neg(dist(B, A, G_t) \geq dist(B, M, G_t) + dist(M, A, G_h) - 1)$ then
      $\mathbb{A} \leftarrow \mathbb{A} \setminus \{B\}$
    end if
  end for
  return $\mathbb{R}, \mathbb{A}$
```

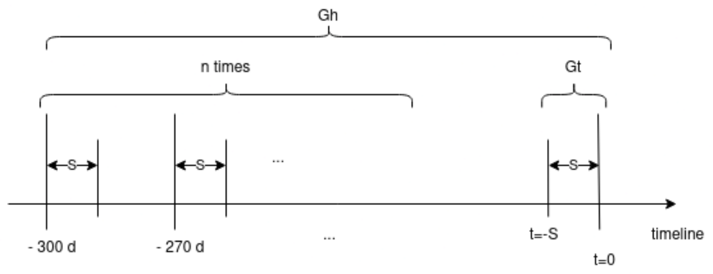

Figure 4.2: Illustration of how the timeslots are chosen to generate the datamodels

one has to define a time slice S, from which the BGP update messages flow into *today's* graph G_t. In our case, this was set to 30 minutes.

Next, we pick the same time interval S from 10 months back and add it to G_h. We repeat adding $n \leq 10$ timeslots of size S to G_h (in our case, three times), where the start of each timeslot is 30 days after the previous. Fig. 4.2 illustrates this process.

Chapter 5

Evaluation

In this Chapter, we will inspect the datasets we generated in chapter 4 and analyze through this the effectiveness of the algorithms.

5.1 Comparing current link information against historical

In section 4.2.1 we presented the datamodel. It consists of two graphs, G_t, for the current state of the internet, and G_h, which summarizes every valid link within the last ten months. From the fixed dataset given in sec. 4.3.2, table 5.1 gives a first insight into the two graphs. We note that G_h has quite some more nodes and edges than G_t. This implies that there are Alice and Mallory pairings, which do have an attack impact, as adding a new link to a graph implies that there is at least one tuple of nodes which have a new shortest path.

Graph	#Nodes	#Edges
G_t	82264	259461
G_h	85765	349781

Table 5.1: Comparing the internet graphs G_t and G_h for the dataset defined in section 4.3.2.

5.2 Attack Evaluation

To evaluate an attack, we must first select a victim and an attacker and use then the algorithm presented in section 4.2 to evaluate the effectiveness.

5.2.1 Selecting Alice

To evaluate the attack's effectiveness, we need first a way to select Alice and Mallory. For our model to work without further adaption we need Alice to be a monitor. The rationale is, that for Artemis to work properly, BGP Messages from the border routers operating Artemis need to be ingested, which in our model is only given for the monitors.

If we operate Artemis for a monitor, we see paths from all over the internet towards Artemis, and also from this AS away towards the other monitors, therefore giving us bidirectional links. If we operate Artemis for an AS which is not a monitor, we will only see the paths from this AS away towards all monitors, but not necessarily towards this AS; therefore, Artemis' bi-directionality rule would trigger for every BGP update message. While it is not hard to add these links in reverse order to Artemis, it is simpler to set Alice to a BGP monitor. Searching for the AS hosting RouteViews.org on bgp.tools returns us AS3582. Alternatively, we could choose AS3333, which hosts the RIPE RIS Live website and, therefore, most likely also monitors the BGP updates.

Figure 5.1 illustrates an example of links in the internet graph we see with and without feeding the BGP Updates from border routers into Artemis.

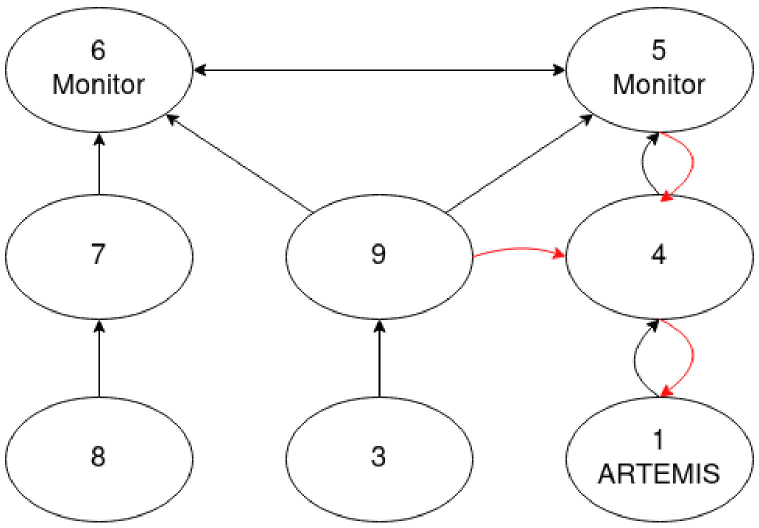

Figure 5.1: In black, all links Artemis can see when only BGP messages from monitors are fed into the detection. The red arrows are known to Artemis if the border router's messages are fed into Artemis.

5.2.2 Selecting Mallory

Given Alice, we can now search for an AS as a potential hijacker. As the messages must eventually propagate through today's internet, we must select from G_t. Next, we must remove every stub AS as they cannot announce a hijacked BGP message without immediately triggering an AS-path-loop at their only peer. As a last step, we take the distance from Alice to any node and remove the ASN from the remaining options if the distance is the same in G_t and G_h. The set we now have contains all ASes which are potentially able to attract some traffic towards Alice with a hijack. We select at random from this set.

5.2.3 Evaluate Attack impact

Given Alice and Mallory we use algorithm 3 to get all nodes which are under effect of Mallory with and without a hijack in progress. We get three different cases presented in the following.

No Attack impact

For example, for AS40945, we get the same ASes, which are under the effect of Mallory with and without ongoing hijack. So this AS is not ideal for attacking AS3582.

Attack Successfull

For AS44020 as Mallory, we see some successful attacks. 4 ASes are under the effect of Mallory when no hijack is in place. When AS44040 now starts to announce a new route, which is found in

G_h, the traffic from 11 additional ASes can be attracted, without Artemis detecting a hijack.

Hijack Detected

If we set AS264920 as Mallory 2 ASes are under the effect of Mallory without a hijack in progress, and 9 additional AS get under the effect of Mallory if Mallory starts hijacking. However, Artemis notes that new non-bidirectional links appear and mark them as hijacks. There are two things to note about this:

First, the new paths are only detected as a hijack, because of an inaccuracy of the model. We generate the extended link list only with the traffic logged on the monitors. To now test if a hijack is successful or not, we ingest BGP messages logged at a non-monitor into Artemis. If we had logged BGP messages for other prefixes on this non-monitor the links would appear bidirectional. While it is certainly possible to adapt the model, we decided not to at this stage.

Second, the path Mallory announces to its peers is valid, as seen by Artemis. So Artemis does never blame Mallory, but some other AS to be the hijacker.

Chapter 6

Discussion

This section will first briefly summarize the achievements of this thesis and then provide insights on how to continue this project.

6.1 Summary

This thesis aims to find the limitations of the defensive system Artemis. As such, we started by reverse engineering the open-source implementation of Artemis, where we found the type-N/E detection to be not implemented. Next, we provided pseudocode and implementation for the type-N/E detection as proposed by Artemis. With the Artemis implementation running, we started to generate a data model that compares the current state of the internet to the historical view of what Artemis accepts as new links.

Eventually, we analyzed the attacks against Artemis and presented which attacks were effective and which were not in our model. Furthermore, a complete pipeline is available to reproduce the results.

6.2 Outlook

The next step with the model available is to make a statistical analysis of the attack successes and how big the impact is. So far, we have only used one dataset and one AS as Alice. The impact analysis at scale is left for future work. This analysis would eventually lead to the question: Who would be a good attacker/ victim?

As our model currently uses a simplified Internet topology, one can also put effort into making it more accurate. Currently, we do not consider business relationships to check if Mallory is actually in the path between two nodes or not. Using a BGP simulator the results would also be more accurate.

Another briefly touched topic is who Artemis blames for being a hijacker. Artemis blames the AS on the left side of the first false link in the path as the hijacker, although the real hijacker might be a lot later in the path. So, crafting messages that blame somebody else should be easily possible, but it has not been analyzed yet, and possible countermeasures are not discussed.

Bibliography

[1] www.codebgp.com, Accessed: 6.2.2023.

[2] Artemis' github repo. https://github.com/FORTH-ICS-INSPIRE/artemis/tree/704a7b2c32f64cb65560665d6afadb913c6e18fe, Accessed: 8.2.2024.

[3] Artemis user documentation. https://bgpartemis.readthedocs.io/en/latest/, Accessed: 8.2.2024.

[4] Artemis utils updates.py. https://github.com/FORTH-ICS-INSPIRE/artemis/blob/master/utils/artemis_utils/updates.py, Accessed: 16.2.2024.

[5] Bgp hijacking of twitter prefix by rtcomm.ru. https://isc.sans.edu/diary/BGP+Hijacking+of+Twitter+Prefix+by+RTComm.ru/28488, Accessed: 20.2.2024.

[6] Bgp.tools. https://bgp.tools, Accessed: 13.2.2024.

[7] The C4 model for visualising software architecture. https://c4model.com/, Accessed: 8.2.2024.

[8] Containerview of artemis. https://github.com/FORTH-ICS-INSPIRE/artemis/blob/master/docs/images/arch_containers.jpg, Accessed: 16.2.2024.

[9] Docker compose viz. https://github.com/pmsipilot/docker-compose-viz, Accessd: 8.2.2024.

[10] How pakistan knocked youtube offline (and how to make sure it never happens again). https://www.cnet.com/culture/how-pakistan-knocked-youtube-offline-and-how-to-make-sure-it-never-happens-again/, Accessed: 20.2.2024.

[11] Klayswap crypto users lose funds after bgp hijack. https://therecord.media/klayswap-crypto-users-lose-funds-after-bgp-hijack, Accessed: 20.2.2024.

[12] pgadmin4. https://www.postgresql.org/, Accessed: 8.2.2024.

[13] Postgresql. https://www.postgresql.org/, Accessed: 8.2.2024.

[14] Rabbitmq. https://www.rabbitmq.com/, Accessed: 13.2.2024.

[15] Redis. https://redis.com/, Accessed: 8.2.2024.

[16] Redis insight. https://redis.com/redis-enterprise/redis-insight/, Accessed: 8.2.2024.

[17] Rfc 6483 – validation of route origination using the resource certificate public key infrastructure (pki) and route origin authorizations (roas). `https://www.rfc-editor.org/rfc/rfc6483.html`, Accessed: 20.2.2024.

[18] Rfc4271. `https://www.rfc-editor.org/rfc/rfc4271#section-9.1.1`, Accessed: 19.2.2024.

[19] Ris route collector peering list. `https://www.ris.ripe.net/peerlist/all.shtml`, Accessed: 8.2.2024.

[20] Routevies collectors. `https://www.routeviews.org/routeviews/index.php/collectors/`, Accessed: 8.2.2024.

[21] Routeviews. `https://www.routeviews.org/routeviews/`, Accessed: 8.2.2024.

[22] Routing information service (ris). `https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris/`, Accessed: 8.2.2024.

[23] Sermpezis, P., Kotronis, V., Gigis, P., Dimitropoulos, X., Cicalese, D., King, A., and Dainotti, A. ARTEMIS: Neutralizing BGP Hijacking Within a Minute. *IEEE/ACM Transactions on Networking 26*, 6 (Dec. 2018), 2471–2486.

Appendix A

Artemis Architecture Connectors

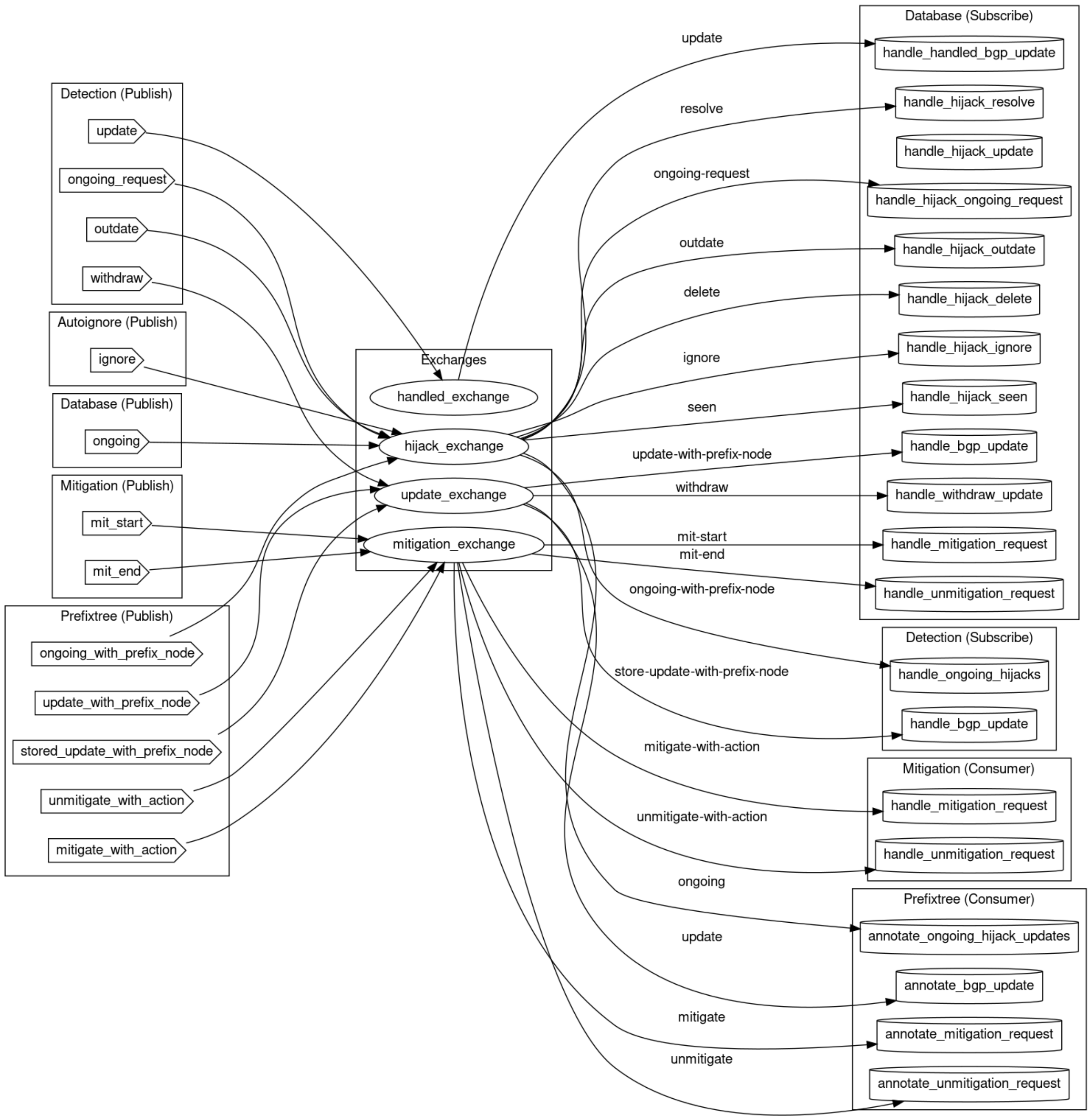

Figure A.1: Visualization of the most relevant RabbitMQ Exchanges with routing keys and what functions are subscribers to the respective exchanges.

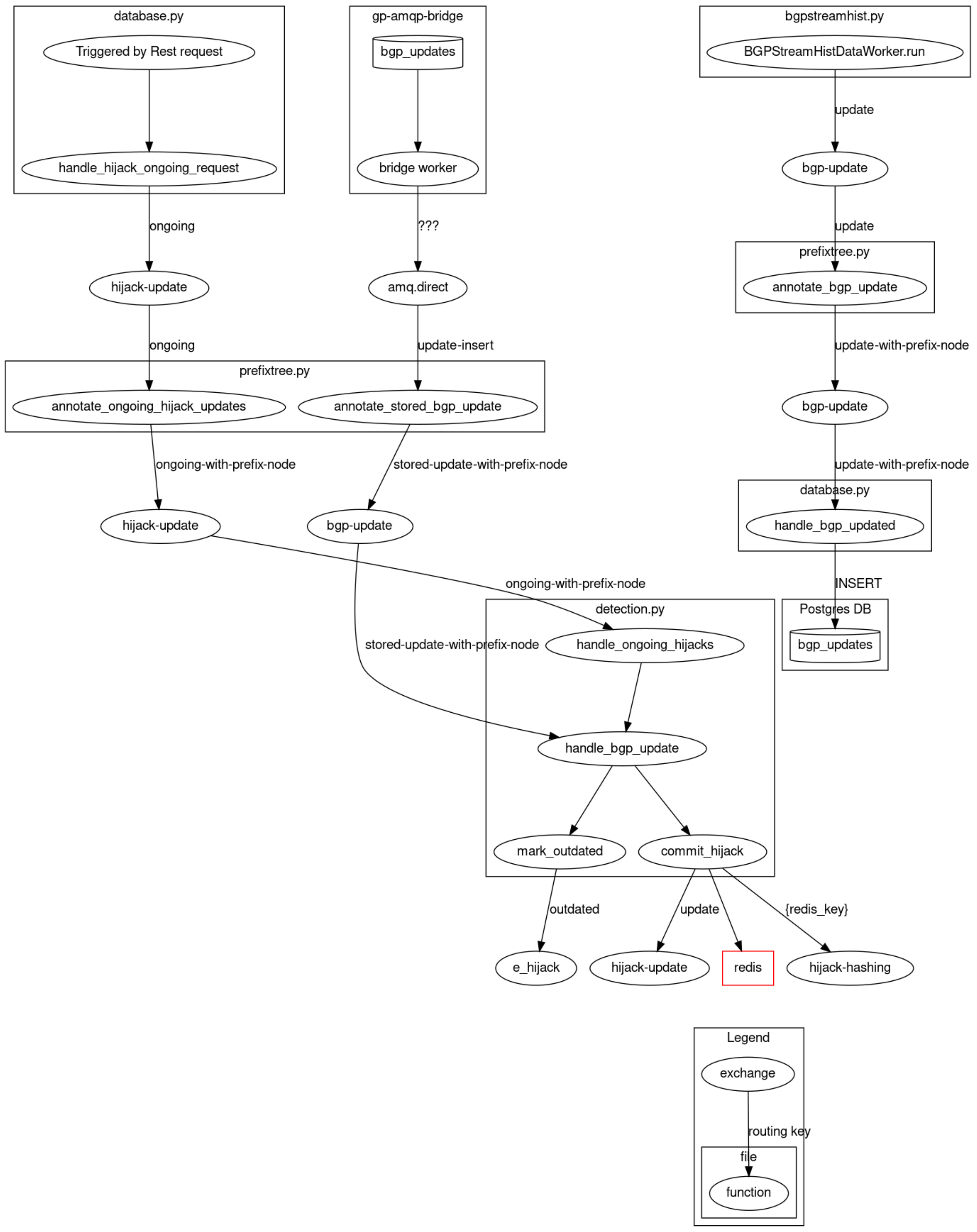

Figure A.2: Dataflow from message injection in the bgpstreamhisttap.py towards the Postgres DB. A AMPQ-Bridge